Vaibhav Kurrey

Technical Document

Cylinder Surface Inspection System

Introduction

This documentation serves as a comprehensive guide to the development of the Cylinder Inspection System. The system's primary purpose is to identify defects in cylinders, such as surface dent marks, read date codes using Optical Character Recognition (OCR) technology, and verify the presence of safety caps on cylinder tops. Additionally, it includes an efficient alert mechanism to promptly notify operators of any detected defects.

Tools and Technology

Ide and code editors used for this project are Visual studio 2019 and Visual studio code. Languages used for development are C#(.netFramework 4.8) and python (3.6).

Front end of the project is developed and designed within Visual studio 2022 using winforms

.net framework.

Event handling, api consuming and cameras are controlled with C#.

Database management of this project is done with the help of PostgreSql. Database has been created locally on the system.

Camera Control

The camera used in this project are 6 HikVision area scan cameras which can be controlled with the help of C# in the following way.

Install MVS sdk application into the system.

Samples can be found in the location "C:\Program Files (x86)\MVS\Development".

Take a look at the basic demo Sample and get a basic understanding on how things are working.

Add the MvCameraControl.Net Dll into your project and use the required code to start and control the camera.

For this project we have created two classes to control the cameras and keep the code seperate. Which are cameraInstances.cs and HRcam.cs

Consuming python API with C#

Python api has been created using FastApi. When the project starts it runs the python API using cmd line arguments and sends requests to test if the API is running successfully or not.

If API is not running on the first request a thread tries to start it again. This process repeats till the API is running successfully.

After successfully running the API another request is sent to load the deep learning model into python API.

System/Hardware Specifications

Cpu - intel i9 12 gen

Gpu - Nvidia 3060-ti

Ram - 32Gb , Storage - 2Tb (1 Tb SSD 1Tb HDD)

INSTANCE SEGMENTATION ON CUSTOM DATASETS IN WINDOWS - 11

Instance segmentation is a cutting-edge computer vision technique that not only detects objects in an image but also precisely outlines and classifies each individual object. This level of granularity allows machines to understand images in a highly detailed manner, making it invaluable in various applications, including autonomous vehicles, medical imaging, and robotics.

Backbone Architecture (Resnet101):

The core architecture used as the backbone for this project is ResNet101, a deep neural network known for its exceptional performance in image-related tasks.

GitHub Repository:

The project's source code is available on GitHub

at - https://github.com/ahmedfgad/Mask-RCNN-TF2

Research Paper:

For in-depth understanding, the research paper associated with this project can be found

at https://arxiv.org/abs/1703.06870

CUDA Requirements:

CUDA 11.x: This project relies on CUDA for GPU acceleration. You must install CUDA version 11.x.
CUDA Download Link: CUDA 11.x Download

cuDNN:

cuDNN is a crucial GPU-accelerated library for deep neural networks. Make sure to install cuDNN.
cuDNN Download Link: cuDNN Download

Visual Studio 2019:

Visual Studio 2019 is the integrated development environment used for this project. Ensure you have it installed.
Visual Studio 2019 Download Link: https://visualstudio.microsoft.com/downloads/

Setup Details:

To set up the project environment and get it up and running, follow these step-by-step instructions:

Step 1: Create a Conda Virtual Environment with Python 3.8

First, create a clean Conda virtual environment with Python 3.8. This environment will isolate your project and its dependencies from your system's global Python installation.

Step 2: Install the Dependencies

Once your virtual environment is activated, install the necessary project dependencies. This ensures that your environment has all the required libraries and packages to run the project smoothly.

Step 3: Clone the Mask_RCNN Repository

Clone the Mask_RCNN repository from the GitHub repository you provided. This will give you access to the project's source code and assets.

Step 4: Install pycocotools

Install the pycocotools package, which is essential for working with COCO datasets and annotations. It's a critical part of the project's functionality.

Step 5: Download the Pre-trained Weights

Download the pre-trained weights required for the project. These pre-trained weights are trained on a large dataset and serve as a starting point for your model.

Step 6: Test It

Finally, after completing all the previous steps, test the project setup to ensure that everything is working as expected. This typically involves running some sample code or scripts to verify that the system, dependencies, and pre-trained weights are correctly integrated.

Anaconda Environment Setup:

To set up your project environment using Anaconda, follow these steps:

Anaconda - https://www.anaconda.com/products/distribution

Create Environments using anaconda:

command - conda create -n "yourenvname "

python=3.8.0 Than Activate env in anaconda command

line - Activate your Env - conda activate yourenvname

Than install all the requirements :

conda install numpy scipy Pillow cython matplotlib

Install pycocotools

NOTE: pycocotools requires Visual C++ 2019 Build Tools

download here if needed - https://visualstudio.microsoft.com/downloads/

clone this repo

git clone https://github.com/philferriere/cocoapi.git

use pip to install pycocotools

pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

Download the Pre-trained Weights:

Go here https://github.com/matterport/Mask_RCNN/releases

download the mask_rcnn_coco.h5 file

CREATE DATASET USING VGG ANNOTATOR (VIA) ————————————>

link - https://www.robots.ox.ac.uk/~vgg/software/via/

Create folder : Dataset

In the dataset folder create 2 folders : train and val Put training images in train folder and validation images in Val folder.

STEP BY STEP DETECTIONS

1. Anchor sorting and filtering -

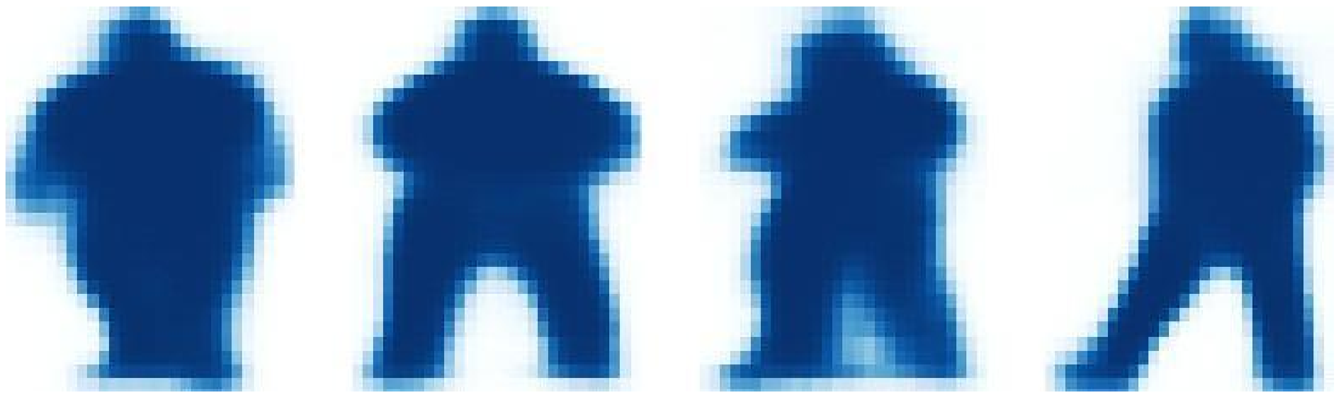
Visualizes every step of the first stage Region Proposal Network and displays positive and negative anchors along with anchor box refinement.

2. Bounding Box Refinement-

This is an example of final detection boxes (dotted lines) and the refinement applied to them (solid lines) in the second stage.

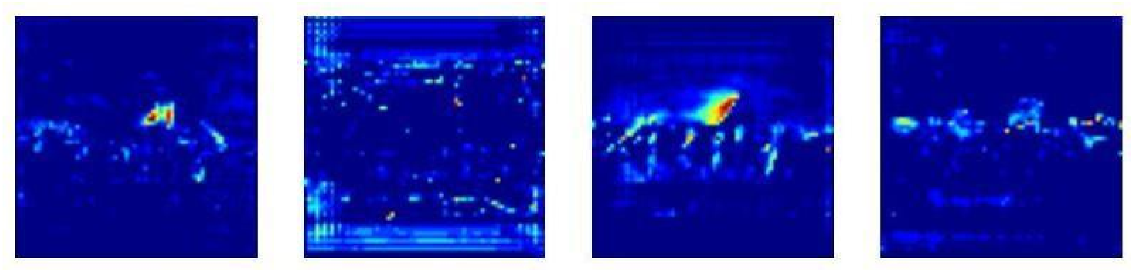
3. Mask Generation-

Examples of generated masks. These then get scaled and placed on the image in the right location.



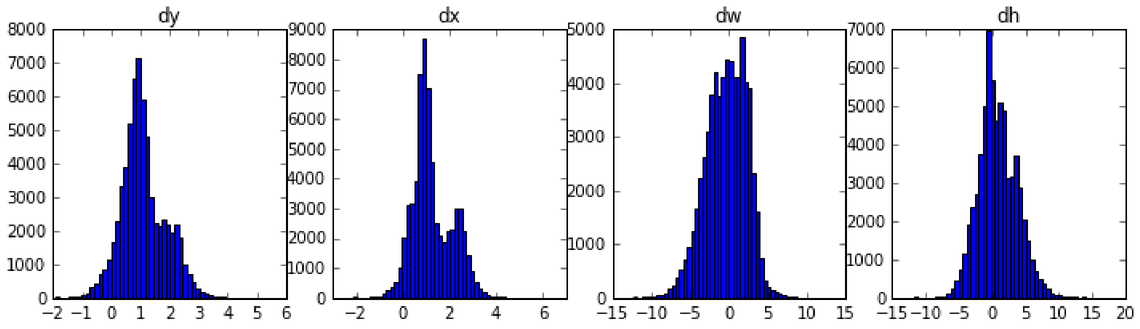
4. Layer activations-

Often it's useful to inspect the activations at different layers to look for signs of trouble (all zeros or random noise).



5. Weight Histograms-

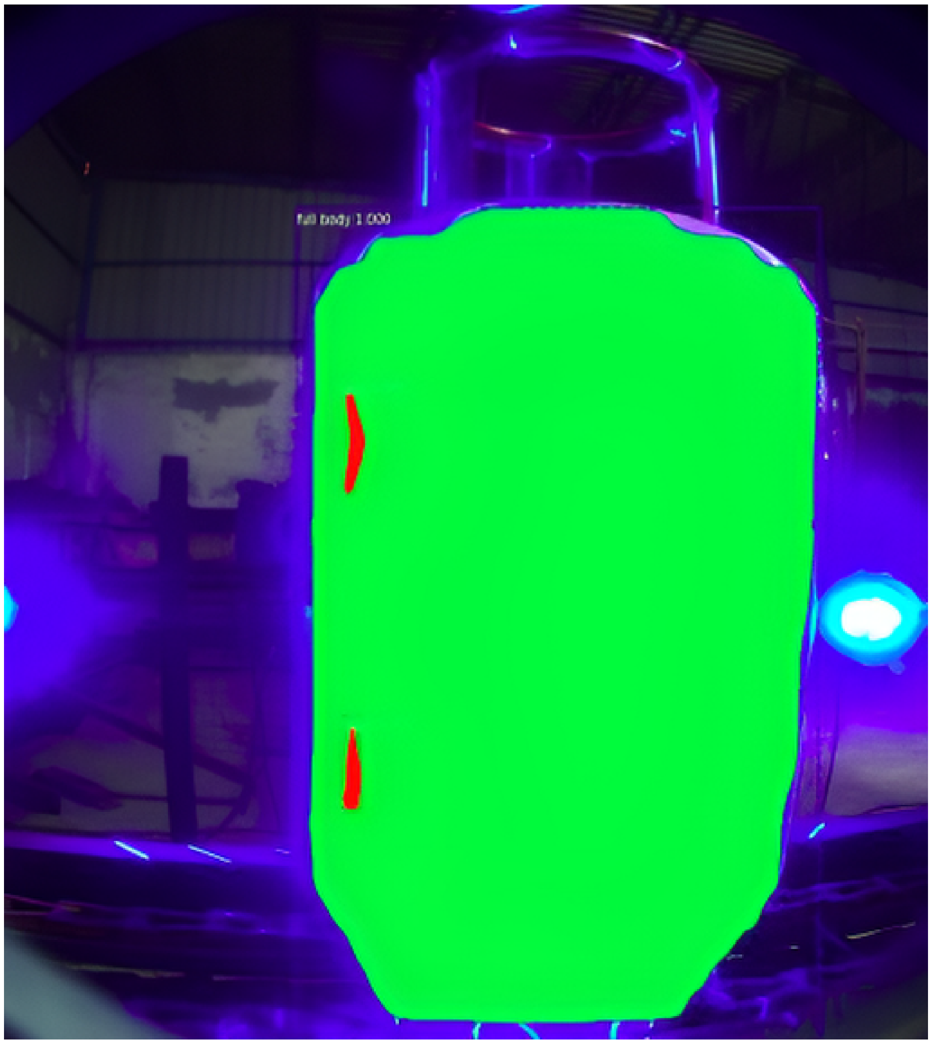
Another useful debugging tool is to inspect the weight histograms. These are included in the inspect_weights.ipynb notebook.



6. Logging to TensorBoard-

TensorBoard is another great debugging and visualization tool. The model is configured to log losses and save weights at the end of every epoch.

7. Composing the different pieces into a final result



PYTHON SCRIPTS FOR TRAINING AND TESTING ->

model.py file – A script for loading RESNET101 architecture and model layer creation.

custum.py file –A script for Training custom datasets.

visualize.py file – A script for visualizing the final result.

Cylinder_surface_Inspection.py file – A script for image Segmentation and Mask.

LET ' S START

Building a custom ocr using YOLO (you only look once) in windows - 11

What is OCR?

OCR stands for Optical Character Recognition. It is used to read text from images such as a scanned document or a picture. This technology is used to convert virtually any kind of images containing written text (typed, handwritten or printed) into machine-readable text data.

Here, we are going to build an OCR which only reads the information you want it to read from a given image.

OCR has two major building blocks:

1. Text detection

2. Text recognition

Our first task is to detect the required text from images. Detecting the required text is a tough task but thanks to deep learning, we'll be able to selectively read text from an image.

In deep learning we are using YOLOv4 here mainly because,

1. No one can beat it when it comes to speed.
2. Has good enough accuracy for our application.
3. YOLOv4 has Feature Pyramid Network (FPN) to detect small objects better.

Enough said, let's dive into YOLO

YOLO is a state-of-the-art, real-time object detection network. There are many versions of it. YOLOv4 is the fastest version.

YOLOv4 uses Darknet-53 as its feature extractor. It has overall 137 convolutional layers, hence the name 'Darknet-53.conv137'. It has successive 3 × 3 and 1 × 1 convolutional layers and has some shortcut connections.

For the purpose of classification, independent logistic classifiers are used with the binary cross-entropy loss function.

Training YOLO using the Darknet framework

We will use the Darknet neural network framework for training and testing. The framework uses multi-scale training, lots of data augmentation and batch normalization. It is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

You can find the source on GitHub.

LINK - https://github.com/AlexeyAB/darknet

Paper YOLOv4: https://arxiv.org/abs/2004.10934

Here is to install the Darknet framework. (If you are going to use GPU then update GPU=1 and CUDNN=1 in the makefile.)

git clone https://github.com/pjreddie/darknet.git

1. cd darknet
2. make

Let's start building

TOOLS :------->

Cuda Requirements - Cuda 11x & Cudnn

Cuda link - https://developer.nvidia.com/cuda-11.0-download-archive

Cudnn link - https://developer.nvidia.com/rdp/cudnn-archive

Visual studio 2019 - https://visualstudio.microsoft.com/downloads/

Get your data first...

Data is the first and most important thing in any machine learning based project. So, whatever is your application make sure you have around 100 to 1000 images for it. If you have a fewer number of images, then use image augmentation to increase the size of your data. In image augmentation, we basically alter images by changing their size, orientation, light, color, etc.

There are many methods available for augmentation, and you can very easily pick any method you like. I would like to mention an image augmentation library called Albumentations and keras data augmentation ,

I collected 1000 to 2000 images of cylinder top view , and using image augmentation, created a dataset of 5000 cylinder top view images.

Data Annotation

Once we have collected the data, let's move to the next step, which is to label it. There are many free data annotation tools available. I used Labelimg because it is a simple tool and works like a charm.

Follow this link, to understand the process of data annotation.

LINK - https://github.com/HumanSignal/labelImg

Note that it is important we tag all the text fields that we want to read from the image data. It also generates the data folders which will be required during training.

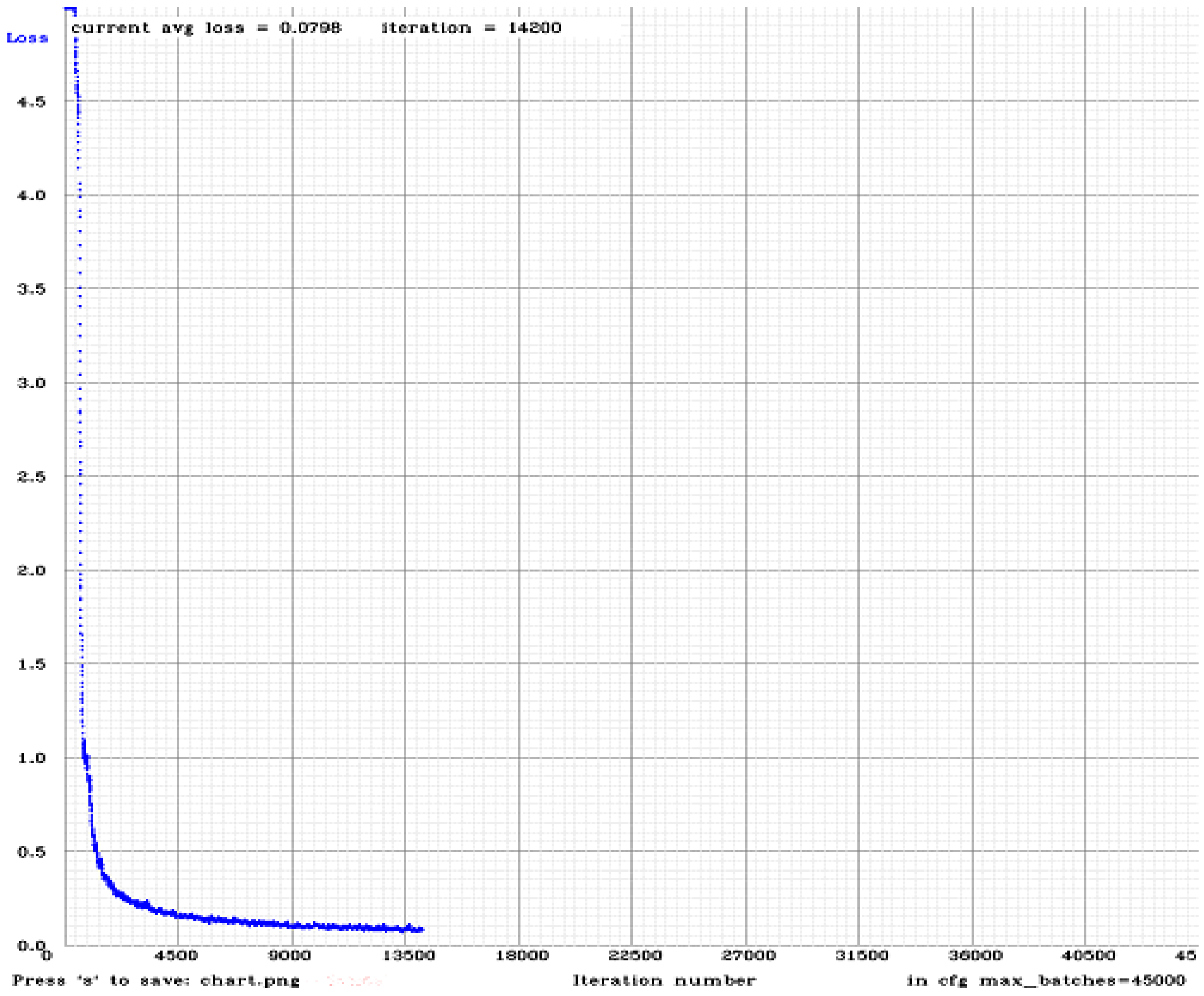
Make sure to set export format to YOLO after tagging. After annotation, copy all the generated files to the data folder of the cloned repository.
Training

To start training our OCR, we first need to modify our config file. You will get your required config file in the ‘cfg’ folder named ‘yolov3.cfg’. Here, you need to change the batch size, subdivision, number of classes and filter parameters. Follow the required changes needed in the config file.
Testing

To start the training hit this command

```
./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.137
```

The best thing is it has multi GPU support. When you see that average loss ‘0.xxxxxx avg’ no longer decreases after a certain number of iterations, you should stop training. As you can see in the below chart I stopped at 14200 iterations as the loss became constant.



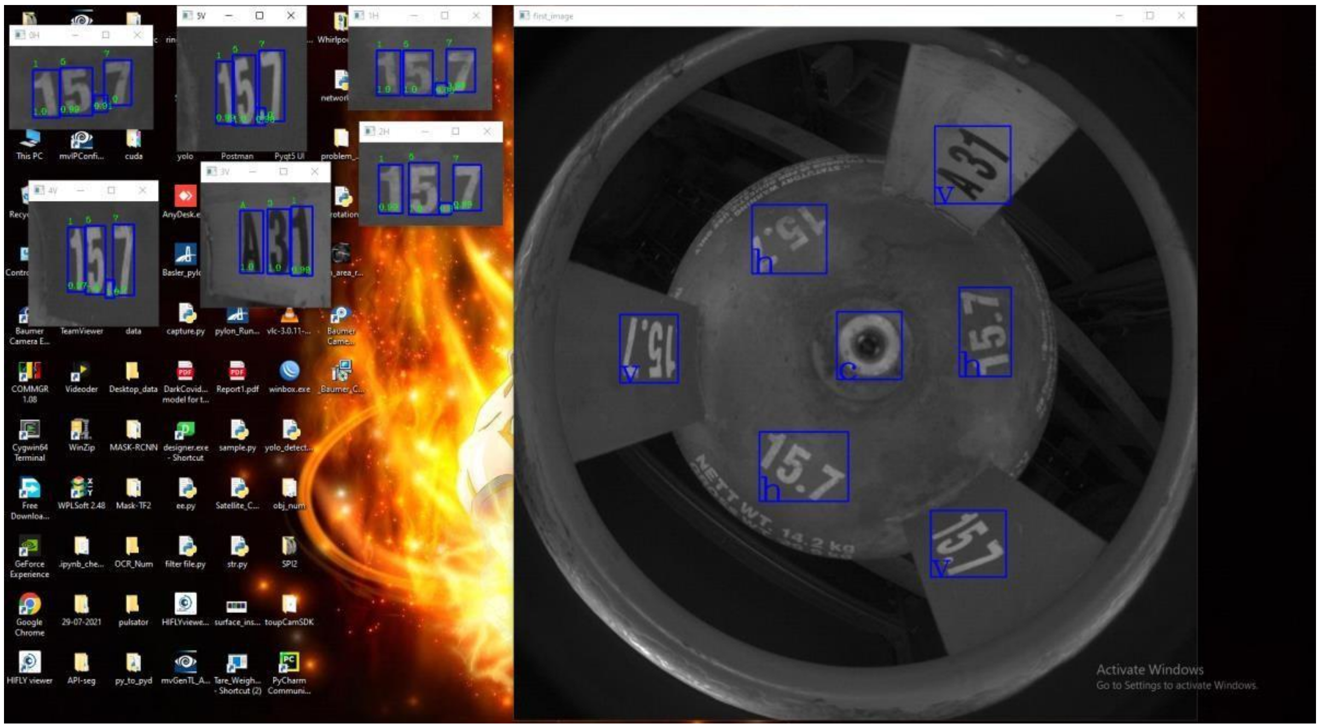
It's not always the case that you will get the best results from the last weight file. I got the best results on the 8000th iteration. You need to evaluate them on the basis of the mAP(Mean Average Precision) score. Choose a weights-file with the highest mAP score.

Here link - https://github.com/AlexeyAB/darknet#how-to-compile-on-windows

They have provided the details of how to use the mAP score. So now, when you run this detector on a sample image you will get the bounding box of the detected text field from which you can easily crop that region.

Final Detection Result -

here are the image -



first_image
A31
15.7
Activate Windows
Go to Settings to activate Windows.