Summer Internship 2020- Software
Image processing and OCR on Kranti meters

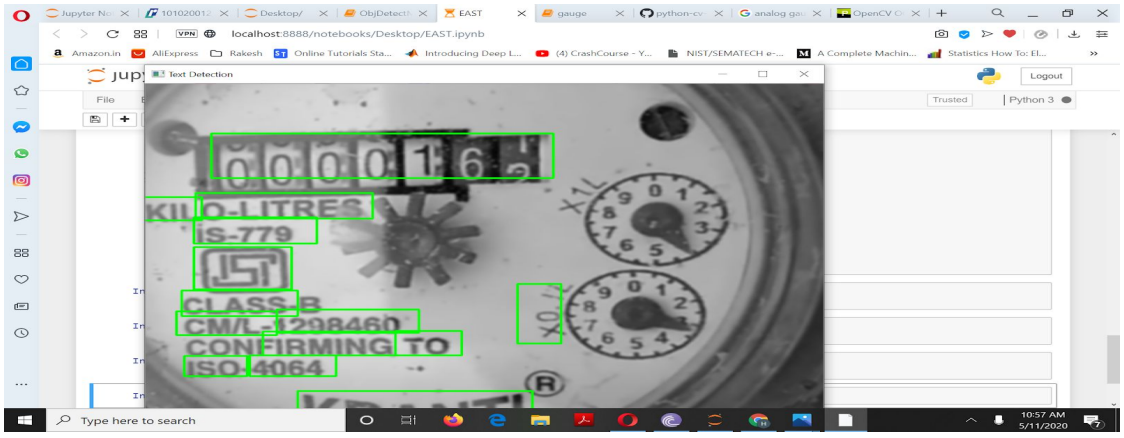
Introduction

The aim of this task was to be able to read the reading on the meter and also read the number the dial(s) is pointing towards. This can be achieved with image processing and pre-existing Optical Character Recognition frameworks.

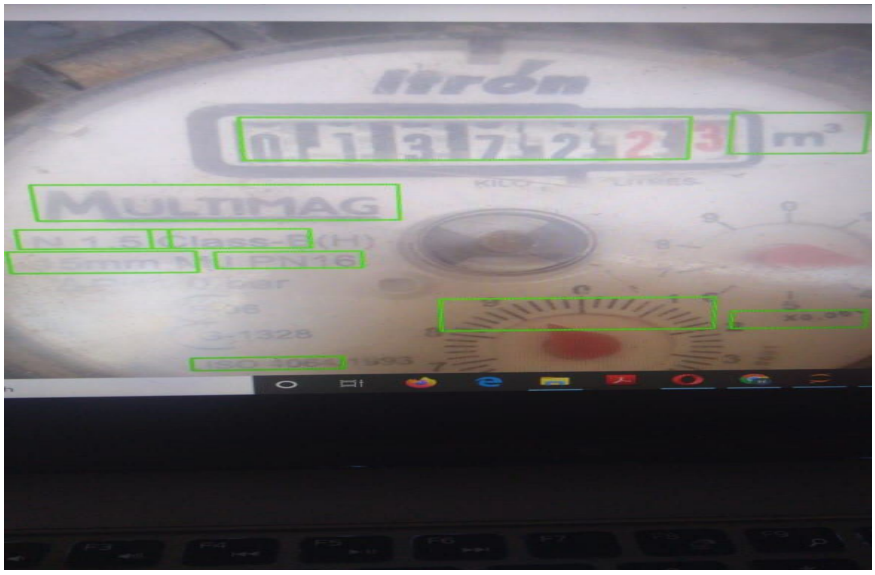
Current Progress

As of 22nd May 2020, we have tried to implement seven-segment OCR , PyTesseract , EAST and multiple street view house number (SVHN) detection models in order to detect the reading of the main meter. We followed the following process:

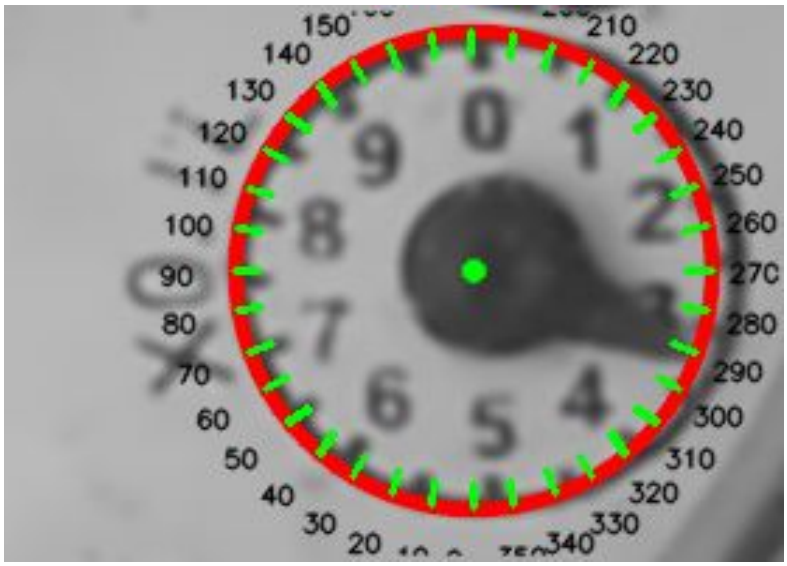
1. Applied Gaussian Blur Thresholding to the image and ran Pytesseract on it- All text other than that on the meter and dials was read.
2. We realized that working on the meter and dials separately might yield better results. We tried to use EAST (a pre-trained text detection model implemented with Tensorflow) for the meter, which yielded the following result:

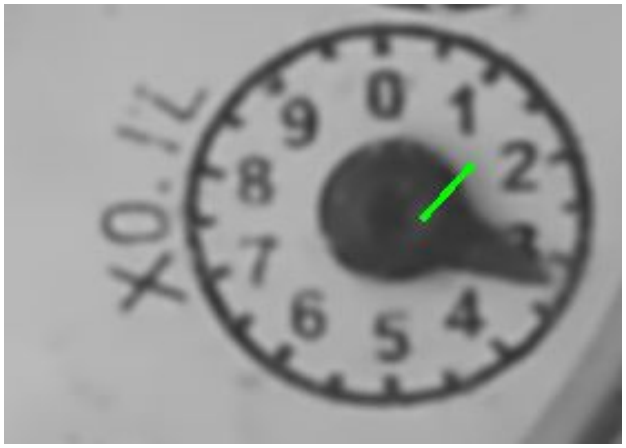


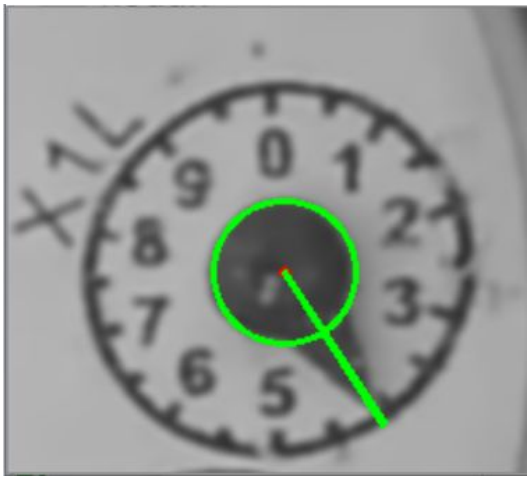
3. We checked the font but the OCR module being used worked on that too, so we came to the conclusion that the font is not the problem, and that the segmentation of the digits might be hampering the readings. We had to standardize the orientation of the counter to be horizontal to ensure that text detection worked. We might have to train our own model for counter-related readings, since we were unable to extract the numbers from the meter.



4. For the dials, we found and tried to implement the Intel IoT devkit- analog gauge reader. In order to make it work, we needed to detect the dials separately and then run the script inside the detection box pixels. We thought of standardizing the image by alignment and then using template-matching to find the top-left pixel of the dial, then by knowing the size of the template we can crop out the dial and feed it to the gauge reader. We were able to find the pixel. We didnt write code for the actual cropping yet as we thought of primarily reading the dials, before finishing the cropping part.
5. The gauge reader script uses the Houghcircles function in OpenCV to locate the outer boundary of the dial and mark the centre of the dial and creates a rough calibration along with the degrees marked at the sides.
6. Upon finishing calibrating, we try to find the required line on the pointer using the HoughLinesP function in OpenCV. Once we find out the angle of the pointer with respect to the 0 position, we can find the reading, with minimum and maximum readings as additional parameters.


150
140
130
120
110
100
90
80
70
60
50
40
30
20
210
220
230
240
250
260
270
280
290
300
310
320
330
340
0
1
2
3
4
5
6
7
8
9


0
1
2
3
4
5
6
7
8
9


0
1
2
3
5
6
7
8
9

The green lines were successfully drawn over the dial location.

7. To read the actual dial, we thought of getting standardized images from the phone camera (prototyped a thunkable app with camera, which then uploaded the pictures to CloudinaryDB), and then extracting the image of the dial.
8. We found a company that does work very similar to this: https://anyline.com/news/water-meter-reading/
9. As suggested by Alekh Sir, we also tried implementing the following Street View House Number (SVHN) models:
   https://github.com/thomalm/svhn-multi-digit

   https://www.kaggle.com/olgabelitskaya/svhn-digit-recognition

   https://github.com/aditya9211/SVHN-CNN

   https://github.com/jiweibo/SVHN-Multi-Digit-Recognition

   https://github.com/bdiesel/tensorflow-svhn

   https://github.com/tohinz/SVHN-Classifier

   However, they did not work. The object was not getting detecting by these algorithms.

Recommendations based on the Paper Titled **Convolutional Neural Networks for Automatic Meter Reading**

Link : https://arxiv.org/pdf/1902.09600.pdf

The paper designs a two-stage approach that employs the Fast-YOLO object detector for counter detection and evaluates three different CNN-based approaches for counter recognition.

The paper made use of a public dataset called UFPR-AMR dataset,with 2,000 fully and manually annotated images. Furthermore, we propose the use of a data augmentation technique to generate a balanced training set with many more examples to train the CNN models for counter recognition.

Link to Dataset : https://web.inf.ufpr.br/vri/databases/ufpr-amr/

Some information about the dataset

Camera	Images
LG G3	947
J7 Prime	584
iPhone 6s	469
Total	2,000

Info	Counters	Digits
Minimum Size	238×96	35×63
Maximum Size	$1{,}689 \times 365$	168×283
Average Size	674×178	76×134
Aspect Ratio	3.79	0.57

Remark that a brand new meter starts with 00000 and the most significant digit positions take longer to be increased. Then, it is natural that the less significant digits (i.e., 0 and 1) have many more instances than the others. Nonetheless, digits 4-9 have a fairly similar number of examples.

The Counter Detection was done using FAST-YOLO implementation and then 3 CNN's were used for reading the numbers. CR-NET, Multi-Task Learning and Convolutional Recurrent Neural Network (CRNN), for the counter recognition stage (i.e., digit segmentation and recognition). CR-NET is a YOLO-based model proposed for license plate character detection and recognition, while Multi-Task and CRNN are segmentation-free approaches designed respectively for the recognition of license plates and scene text.

Brief Summary of the Accuracies

Approach	Accuracy (%)	
	Digits	Counters
Multi-Task (original training set)	24.64 ± 0.25	00.00 ± 0.00
CRNN (original training set)	92.85 ± 0.93	77.75 ± 2.39
CR-NET (original training set)	97.78 ± 0.17	91.95 ± 0.52
Multi-Task (with data augmentation)	95.96 ± 0.25	87.69 ± 0.40
CRNN (with data augmentation)	97.87 ± 0.21	92.30 ± 0.56
CR-NET (with data augmentation)	$\mathbf{98.30 \pm 0.09}$	$\mathbf{94.13 \pm 0.50}$

The authors of the paper are in process of publishing the code and model weight and we believe it would be much more useful to **use their weights and code** and apply ***transfer learning*** on the Kranti Meter data which would be collected manually.

The paper implements an end to end approach which is basically detecting the counter using YOLO and then 3 approaches to get the digits. We believe in case if YOLO is not accurate in detecting the counter the EAST detection which we wrote could be used instead and the 3 approaches applied on them

The paper makes no mention of dials whatsoever. We believe that once the images are standardized by using fixed cameras mounted over the meter we can implement the above mentioned work done on dial detection.

For future use once we deploy this Model and dial reading scripts and collect a very large amount of data we can attempt to run an **End to End Deep learning model** with some pre trained weights from the above Paper which could generalize well to other meters as well.

Recommendations on TinyML using Tensorflow Lite

TensorflowLite is a library built on the tensorflow backend that is used to build and deploy machine learning models on microcontrollers. At the moment, TFlite supports the following microcontrollers:

1. Arduino Nano 33 BLE Sense
2. SparkFun Edge
3. STM32F746 Discovery kit
4. Adafruit EdgeBadge
5. Adafruit TensorFlow Lite for Microcontrollers Kit
6. Adafruit Circuit Playground Bluefruit
7. Espressif ESP32-DevKitC
8. Espressif ESP-EYE

TFlite currently works for processors based on the ARM Cortex M-Series architecture, like the ones above. TFlite for microcontrollers is written in C++ 11 and requires a 32-bit platform. The framework is available as an Arduino library. It can also generate projects for development environments such as Mbed. It is open source and can be included in any C++ 11 project.

The workflow required to build a tensorflow light model is as follows:

1. Build the model using tensorflow or Keras with TF backend
2. Convert the model to TFlite FlatBuffer using inbuilt function
3. Convert the FlatBuffer to a C byte-array
4. Integrate the TensorflowLite for Microcontrollers for C++ library with your microcontroller IDE
5. Deploy/ upload to the microcontroller.

A very useful tutorial for this: https://www.youtube.com/watch?v=BzzqYNYOcWc

The aforementioned video shows us how a model is built on tensorflow, converted to tensorflow lite and the second part of the video shows the deployment.

Limitations of TFlite:

1. Support for only a limited subset of Tensorflow operations (Click link to see available operations).
2. Support for only the aforementioned devices.
3. Low-level C++ API requires us to set the memory manually, we have to guess the initial values and then work from there. Estimation is required with no one-shot method to guess the required memory.
4. Training is not supported on the device. We need to have trained the model already before converting to TFlite and deploying.