T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

OPTICAL CHARACTER RECOGNITION FROM DIGITAL SCREEN

FATIH KAAN SALGIR

SUPERVISOR
YUSUF SINAN AKGÜL

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

OPTICAL CHARACTER RECOGNITION FROM DIGITAL SCREEN

FATIH KAAN SALGIR

SUPERVISOR
YUSUF SINAN AKGÜL

2023
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 19/01/2023 by the following jury.

JURY

Member
(Supervisor) : Yusuf Sinan Akgül

Member : İbrahim Soğukpınar

ABSTRACT

Optical character recognition (OCR) is a well-studied problem in the field of computer vision, and it involves the automated recognition of characters in images. In this project, we propose to develop a deep learning-based OCR system for the specific task of recognizing characters from digital screens.

Our system will utilize convolutional neural networks (CNNs) and other advanced machine learning techniques to extract features from the input images and classify them into the corresponding characters. We will explore the use of artificially generated data using various digital display fonts for training our OCR system. We will compare various pre-trained deep learning architectures. We will investigate the use of data augmentation techniques to improve the performance of the OCR system, such as random brightness and color distortion.

To evaluate the performance, we will use existing dataset of images of digital screens with known character annotations and create synthetic data. We will measure the accuracy of our system on these datasets and compare various architectures and methods.

Overall, this project aims to contribute to the advancement of OCR technology and provide a useful tool for various applications.

ACKNOWLEDGEMENT

I am deeply grateful to my supervisor Prof. Dr. Yusuf Sinan Akgül for his valuable suggestions and leadership throughout the project. I would like to express my sincere appreciation for his time and dedication.

Fatih Kaan SALGIR

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
CNN	:	Convolutional Neural Network
mAP	:	Mean Average Precision
AR	:	Average Recall
IoU	:	Intersection over Union
SSD	:	Single Shot Detector
R-CNN	:	Region Based Convolutional Neural Networks
TF	:	Tensorflow

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Problem & Motivation

Many sensing systems used in various industries uses embedded digital displays to show the data. However often times this devices have limited communication interfaces, means the device itself doesn't provide a way to transfer data digitally. The data can only be perceived visually. This is often due to cost or outdated systems. In such systems, recording data from LCD displays is performed manually. The lack of real-time data transmission makes it challenging to analyze the information. An OCR system can improve data quality by reducing errors and making it easier to record the data digitally.

An illustration of use case motivates the need for the system to process images of scoreboards used in sport competitions. In basketball courts, for instance, digital displays are commonly used to display the score during a game. However, when it comes to live streams of these games, it can be a challenge to visualize the score. The proposed system aims to address this issue by being able to convert images of scoreboards into any desired format, thus enabling real-time score visualization for live streams.

Another specific example is devices with an LCD used in factories to monitor environment, such as temperature and humidity of soil. Monitoring of these systems in real-time is important for plant growth and crop yield. But the deployed devices do not allow to automatize the process. A fixed camera integrated with an OCR system enables data to be obtained in real-time.

These systems often use LCDs, seven-segment, or dot-matrix displays. The problem with OCR for these displays is that the lines in the numbers don't connect, making it hard for conventional OCR techniques to recognize them. The literature suggests that, deep learning methods are highly effective in OCR tasks, since they can recognize patterns in images even when the image is distorted or of low quality. This allows it to be more robust and accurate than traditional OCR methods.

In conclusion, the objective is to develop a deep computer vision system that given an image of a digital screen obtained through a camera, will locate the digital display and recognize the characters within the region.

1.2. Related Works

There are various approaches in research papers for recognizing characters of digital displays. These approaches can be classified into two categories: handcrafted methods, which involve manually designing features and rules to recognize characters, and deep learning methods, which use neural networks to learn features and patterns for recognition.

An approach was proposed by [1], didn't use deep learning methods but involves the use of image processing and computer vision techniques. The researchers started by converting the input image to the HSV color space. This conversion allows for better separation of color information, which can be useful for identifying the individual segments of the seven-segment digits. Next, the researchers applied a Wiener filter to remove noise from the image, and then converted it to a binary image. This helps to simplify the image and make it easier for the algorithm to identify the segments of the digits. Then applied connected component analysis method to find the largest object in the image followed by morphological bridge operation and connected component extraction to enhance the accuracy of the detection. Finally, they passed each connected component to an OCR algorithm for recognition. The researchers reported that their method achieved a detection rate of 96.4

In [2] focuses the use case for people with blindness or visual impairment who have trouble reading LED or LCD displays on household appliances like microwaves and DVD players. Their connected-component based algorithm to detect LED/LCD digits achieving over 15 frames/second on an Intel Pentium Dual-Core desktop with 2GB of RAM.

Handcrafted methods, which rely on manually designing features and rules, can achieve high accuracy for specific cases, but their overall effectiveness is limited. They cannot be applied to a wide range of scenarios. Deep learning is currently the state-of-the-art method for automatic detection and recognition. It is known to be efficient and flexible, achieving high accuracy and generalization in a wide range of scenarios.

Digital Display Recognition towards Connected Sensing Systems for Precision Agriculture [3] gives an example display at a tea factory with multiple panels showing measurement parameters that need to be observed and noted periodically. The measurements are shown on seven-segment displays. The researchers encountered difficulty using the popular OCR tool Tesseract, which failed to determine the digits with a high degree of accuracy until pre-processing steps such as canny edge detection and morphological operations were applied. With these pre-processing steps, the accuracy increased to 45%. To improve accuracy further, the researchers used data augmentation techniques such as random shear, brightness, and exposure. These techniques artifi-

cially expand the size of the training dataset by applying various transformations to the original images. The researchers then evaluated several neural network architectures for the task of recognizing seven-segment digits, including SSD-MobileNet, ResNet and Faster R-CNN. They found that these neural networks achieved significantly higher accuracy than Tesseract, with SSD-MobileNet achieving 98% accuracy, and ResNet and Faster R-CNN achieving 100% accuracy.

Seven Segment Display Detection and Recognition via Deep Learning Technique [4] uses custom dataset of 240 images with a resolution of 1280 x 720 pixels. After comparison of different CNN architectures, the researchers found out that Cascade R-CNN is the most appropriate model in terms of precision and recall stability. It achieves precision, recall, and F1-score of 0.999, which is higher than other conventional and state-of-the-art methods such as Faster R-CNN, RetinaNet, NAS-FPN, CornerNet and CenterNet.

Overall, these works demonstrate the potential of deep learning-based approaches for OCR from digital screens, and highlight the importance of using a large and diverse dataset for training. Particularly in the recognition of characters in the presence of noise and variations in font, size, and style.

1.3. Success Criteria

- The system will reach at least 90% accuracy rate in terms of recognizing the characters.
- Given an image with digital display, bounding boxes should be drawn, outlining the characters at least 0.8 mAP at 0.5 IoU.
- OCR system should be able to run on live video stream at 10 FPS.
- The model will be trained with at least 5 different digital display font.

2. OCR FROM DIGITAL SCREEN

2.1. Dataset Acquisition

Synthetic data generation is the process of creating artificial data that is similar to real-world data. This is done by using computer algorithms to generate data that simulates the characteristics of real-world data. It is a valuable tool when real-world data is limited to train deep networks. Additionally, it allows for creating diverse and large amounts of data which can be used to improve the robustness and generalization of models.

For the acquisition of the dataset, the first step was the creation of a synthetic dataset. Additionally, annotated datasets of digital displays were collected. These datasets included images of digital displays on their own as well as images of devices that contained a digital displays.

2.1.1. Synthetic Data Generation

To create the dataset for our OCR model, I have first collected more than 50 digital display fonts. These consist of seven-segment, fourteen-segment as well as dot-matrix display fonts.

I developed a python script that generates images of characters on a canvas with different font sizes and positions. The script is capable of generating images using only specified characters or all possible characters supported by the font, and can generate images with characters side by side, as well as number of specified characters long.

For each image, the script labels each character with a bounding box and creates a Pascal VOC XML annotation file. This allows us to easily extract the ground truth text for each image and use it to train and evaluate our OCR model.

The Python PIL library was used to draw characters. The `PIL.ImageFont.truetype` function was used to load a TrueType or OpenType font from a file and create a font object. The `getmask` method of `PIL.ImageFont.ImageFont` was then used to create a bitmap for a given text. This allowed for the extraction of the size and determination of the width and height of a single character. The `getsize` method of `PIL.ImageFont.ImageFont` was also used to obtain the width and height of a given text. With this information, the x and y coordinates of the top left and bottom right corners of the bounding box, which is required in the Pascal VOC format, could be derived.

In the experiments, various font were used and the words used were 5-15 characters long. Only digits were used in the text and the resulting images were drew on a 512×512 canvas.

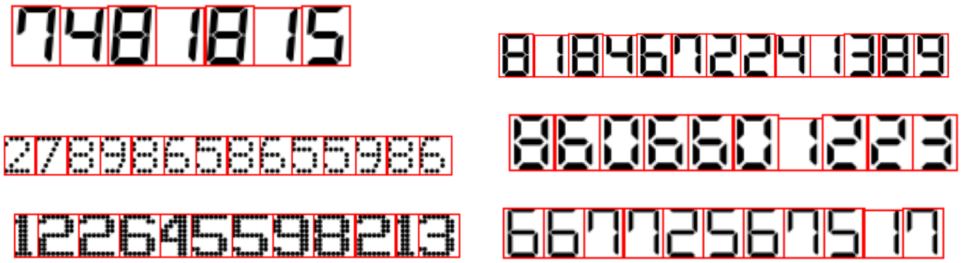


Figure 2.1: Examples of syntheticially generated images

2.1.2. Open Source Datasets

In the process of collecting online datasets, a variety of sources were used. These included images of devices with digital displays such as energy meters and thermometers, as well as images of digital displays on their own. The majority of the images featured seven segment displays, with fewer featuring dot matrix displays. In order to ensure a diverse and representative dataset, care was taken to select datasets that included a range of colors and display types.

Figure 2.2: Example images in the traing & test set

2.2. Working Environment

To create synthetic data and develop utility functions, I used a Google Colab environment connected to Google machines.

To train and evaluate object detection models, I used Google Cloud Jupyter Notebook VM instance with an Nvidia Tesla T4 GPU and 30 GB RAM. The VM instance was configured with CUDA version 11.3 and TensorFlow 2.8.4.

2.3. Data Preparation

Annotations are saved as XML files in PASCAL VOC format. Models based on the TensorFlow object detection API need a special format for all input data, called TFRecord. TensorFlow also requires a label map pbtxt file, which maps each of the used labels to integer values. This label map is used both in the training and detection processes.

For the all models, there are 10 entries in the label map, each corresponding to a digit. For any model, we need a label map and a TF record's generated using that label map.

In order to ensure that all the data could be used in a consistent manner, it was necessary to convert all the datasets to the same format. TensorFlow API has already implemented necessary tools to convert PASCAL VOC to TFRecords. To take advantage of these tools all datasets needed to be in the PASCAL VOC format. Not all of the datasets used in the study were in the correct format. For example energy meter dataset [5] were in the COCO format. In order to use these datasets, it was necessary to convert them to the PASCAL VOC format. To do this, a Python script was implemented that download the images and convert the annotations to the PASCAL VOC format using pycocotools library.

There were some challenges of using multiple datasets:

1. They use different labels for the same digit. For example, while one dataset might use the label "zero" for the digit 0, another dataset might use the label "0".

2. They have different numbers of objects. Some of the datasets were labeled not only with digits but also with special characters such as the dot. These characters needed to be filtered out.

3. In some cases, the image paths used in the annotations were full paths rather than relative paths.

To overcome the challenges a python script was implemented to normalize the various features of the dataset. This was done in order to facilitate the merging

of the various datasets and ensure that they were consistent with one another. The normalization process involved the following steps:

1. Parsing the given set of XML annotation files
2. Extracting all objects from the files
3. Converting full paths to relative paths
4. Filtering out unsupported characters
5. Assigning related labels to proper ones to match the label map

Each dataset was split into a training set and a test set. The ratio of images in the test set ranged from 10-20%, with the remainder of the data being used for training. For datasets with a smaller number of images, it was often necessary to keep as much data as possible in the training set in order to ensure that the model had sufficient data to learn from. Train and test TF Record files are created for each datasets using TensorFlow API and label map. The sets were merged into a single set to be used as input for the model.

2.4. Training Job

The TensorFlow Object Detection API uses protobuf files to configure the training and evaluation process. The pre-trained models allow model configuration via pipeline.config protobuf file. The config file is split into 5 parts.

- The `model` configuration that defines what type of model will be trained (ie. meta-architecture, feature extractor).
- The `train_config`, which decides what parameters should be used to train model parameters (ie. SGD parameters, input preprocessing and feature extractor initialization values).
- The `eval_config`, which determines what set of metrics will be reported for evaluation.
- The `train_input_config`, which defines what dataset the model should be trained on.
- The `eval_input_config`, which defines what dataset the model will be evaluated on.

2.4.1. Model Configuration

Basic configurations is the the same for all of the models:

- `num_classes`: Exact number of class that the model is going to detect. For all the models this is set to 10.
- `batch_size`: Batch size set depending on how much memory available. This is set to 4 or 8 according to model.
- `fine_tune_checkpoint`: Checkpoint of the pre-trained model. Set to checkpoint to corresponding pre-trained model.
- `label_map_path`: Generated label map pbtxt file. Path is set according to related task (original or normalized).
- `train_input_reader.input_path`: Train record file. Configured according to related dataset.
- `eval_input_reader.input_path`: Validation record file. Configured according to related dataset.

To use mean average precision and recall, the metrics set parameter in the evaluation config set to coco detection metrics. mAP is calculated for different IOU values. The validation process is configured to run in CPU parallel to model training. The job waits for checkpoints that the training job generates during model training and uses them to validate the model on a separate dataset.

2.5. Model Training and Evaluation

The goal of object detection is to draw bounding boxes around detected characters. Training a model from scratch to achieve this goal would be time-consuming and require a large amount of input data. To streamline the process, transfer learning methods were explored using the TensorFlow Object Detection API. This allowed for the training of a model that was able to accurately detect characters within images, while also reducing the amount of time and resources required.

Mean Average Precision (mAP) is a common evaluation metric used in object detection tasks. It is used to measure the average precision of a model across multiple classes or categories. mAP is calculated by first computing the average precision for each class and then taking the mean of these values. The average precision for a class is calculated by computing the precision at each recall level, and then interpolating these

values to create a smooth curve. The area under this curve is then taken as the average precision for that class.

Intersection over Union (IoU) is another common evaluation metric used to measure the accuracy of object detection models. It is used to calculate the overlap between the predicted bounding box and the ground truth bounding box, and it is expressed as a ratio between the intersection area of the two boxes and the union area of the two boxes.

When evaluating and comparing the models, the primary metric used was mean average precision (mAP) at specific intersection over union (IoU) thresholds. Specifically, the IoU threshold values of 0.5 and 0.75 were used for detecting bounding boxes. By using both the mAP and IoU metrics, the model's performance is evaluated both in terms of overall accuracy and the accuracy of detecting objects with a specific level of overlap.

The model used in the first experiments was SSD MobileNet v2 320x320. The configuration for this model included default parameters and no data augmentation. Training was conducted for 30,000 iterations using a dataset that was synthetically generated and included only digits. The digits were randomly placed on a 512x512 canvas with a varying font size. The dataset consisted of 1000 images and featured a single font, specifically a seven segment display. In order to properly evaluate the performance of the model, the dataset was split into a training set and a test set. The training set comprised 80% of the data, while the test set made up the remaining 20%. This model was used to establish a baseline for the study and was compared against the results of the following experiments.

The decision not to use data augmentation or alter the default parameters was made in order to create a base model that could be used as a reference point for understanding the effects of these methods. By training a model with no data augmentation or no changes to the default parameters, it was possible to establish a baseline performance level. This baseline could then be used to compare against models that did utilize data augmentation or modified parameters, allowing for a better understanding of the impact of these techniques on model performance.

For the second experiment, a dataset consisting of raw images of digital energy meters [5] was used. This dataset, which was previously used in a study on text detection and recognition in raw image datasets of seven segment digital energy meter displays, contained 169 images of seven segment numerals. The dataset was split into a training set (90%) and a test set (10%), with the goal of maximizing the amount of data in the training set due to the limited size of the dataset. The second experiment was conducted using the same parameters as the first experiment, using SSD MobileNet v2 320x320 with default parameters. However, the training was conducted for 50,000 iterations in order to improve the model's performance.

In both experiments, the models were able to recognize digits on the test set with a high level of accuracy. However, they did not perform as well on images that were not similar to the training set, indicating that the models were not able to generalize well to new data. This led to the decision to conduct another experiment with a more diverse dataset, to improve the model's ability to recognize digits in a wider range of images.

For the third experiment, an open source dataset consisting of images of various digital utility meters [6] was used. This dataset included 90 training images and 21 test images. The images had a wide range of different features, including differences in quality, type of device, and screen type. The first attempt to train the model used the same model as in the previous experiments, but the model with the default configuration was not able to achieve satisfactory results. After 60,000 iterations, the model was only able to achieve a mAP of 0.15 at 0.5 IoU, with an unstable learning curve. In order to improve the performance of the model, the scope of the analysis was expanded to include the use of the EfficientDet D1 model. This model was able to achieve a mAP of 0.52 at 0.5 IoU after the same number of iterations, demonstrating a significant improvement in performance.

This model was able to generalize well to new data, but that the accuracy was not as good as desired. Therefore, in the next experiment, the EfficientDet D1 model was used with a combined dataset consisting of multiple sources of data. This included the synthetic dataset, the Energy Meter Dataset, the Digits Image Dataset, and the 7 Segments Custom Dataset.

The Digits Image Dataset [7] consisted of images of various devices such as gas station liter meters, digital scales, fitness machines, and calculators. These images were captured from different angles. It consist of 757 training images and 84 test images.

The 7 Segments Custom Dataset [6] included images of various screens with different seven segment fonts and colors, as well as blurry and oversaturated images. Some of the LED displays in this dataset had very high contrast and suffered from bleeding, resulting in fuzzy edges around the digit segments. The images in this dataset were cropped to cover only the screen, and were captured or transformed to a direct angle. It consist of 2147 training images 243 test images. Overall, the combined datasets used in the the experiment provided a realistic representation of the challenges that the model would face in real-world scenarios.

The evaluation results for the experiment, using the combined datasets and the EfficientDet D1 model, showed that the model was able to achieve a mAP of 0.98 at 0.5 IoU and a recall of 0.81. These results were significantly better than those obtained in the previous experiments, indicating that the model was able to effectively recognize digits in a wide range of different contexts and environments. According to experiments, the improved performance is due to use of a more powerful and sophisticated model architecture, and the inclusion of a more diverse and representative dataset.

2.6. Data Augmentation & Parameter Tuning

In pre-processing, the following augmentation methods from Tensorflow API are used in training of the final model:

- `random_distort_color`
- `random_adjust_hue`
- `random_adjust_contrast`
- `random_adjust_saturation`
- `random_adjust_brightness`

Applying data augmentation increase the overall precision of the model in the test set. However, it is important to note that in real-world scenarios, data augmentation might have a negative impact on the model's performance. For example, when detecting digits in a seven-segment display, the background of the display might be visible. In this case, the model might be confused by the presence of multiple segments, leading to a decrease in accuracy. Therefore, it is important to consider the real-world scenarios and limitations of the model when applying data augmentation techniques.

To better avoid overfitting cosine learning rate decay is used that controls the learning rate over time. Starting with a low learning rate gives control over gradients at the beginning of training. It is important to save the weights of the original model. The model is fine-tuned on a custom dataset, there's no need to change low-level features that the neural net has already learned. An initial increase in learning rate helps the model to have enough capacity not to get stuck in a local minimum. Smooth learning rate decay over time will lead to stable training, and will also let the model find the best possible fit for the data.

```
learning_rate {
cosine_decay_learning_rate {
  learning_rate_base: 0.04
  total_steps: 25000
  warmup_learning_rate: 0.013333
  warmup_steps: 2000
}
```

Listing 2.1: Learning Rate Configuration

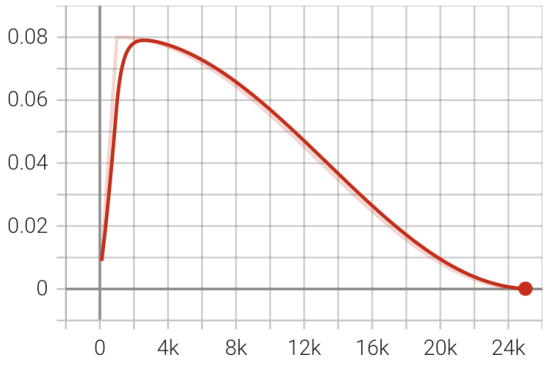


Figure 2.3: Cosine Decay Learning Rate over 25k iterations

According to results obtained by the baseline model, the model is good at localizing objects but performs poorly in classification. So in order to get a better classification, the weight of the classification loss is increased.

```
classification_weight: 1.2
localization_weight: 0.8
```

Listing 2.2: Localization & Classification Loss Weights

EfficientDet D1's default pipeline configuration uses focal loss as classification loss function. According to RetinaNet paper [8], focal loss is very useful for classification when the dataset is highly imbalanced. It down-weights well-classified examples and focuses on hard examples. The loss value is much higher for a sample that is misclassified by the classifier as compared to the loss value corresponding to a well-classified example.

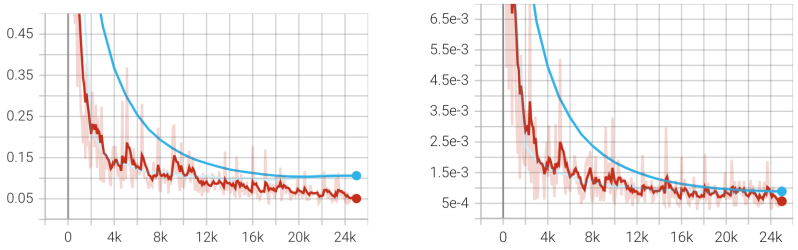
```
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid_focal {
      gamma: 1.5
      alpha: 0.25
    }
  }
}
```

Listing 2.3: EfficientDet D1 Loss Function

2.7. Training & Evaluation Results

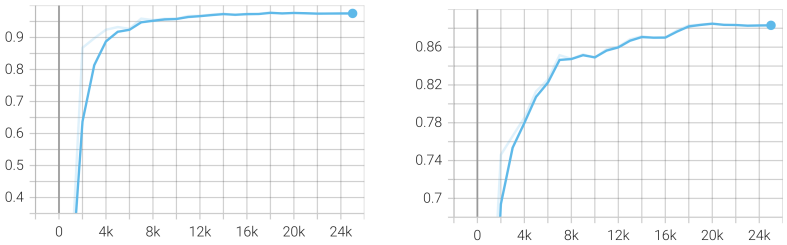
Table 2.1: Evaluation results of object detection models

	maAP@0.5	maAP@0.75	AR@100
SSD MobileNet V2 (Synthetic data)	0.80	0.78	0.70
SSD MobileNet V2 (Energy meter dataset)	0.95	0.74	0.69
SSD MobileNet V2 (Digital meter dataset)	0.15	0.14	0.16
EfficientDet D1 (Digital meter dataset)	0.52	0.43	0.41
EfficientDet D1 (Combined dataset)	0.98	0.94	0.81
EfficientDet D1 (+Data Augmentation)	0.99	0.97	0.88



Classification Loss Localization Loss

Figure 2.4: EfficientDet D1 Loss



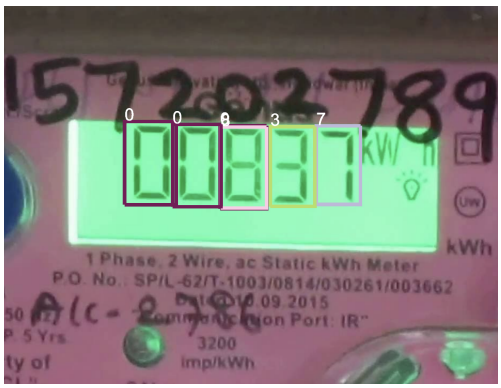
mAP @.75IOU AR with 100 Detections

Figure 2.5: EfficientDet D1 Detection Boxes Precision/Recall

Digital Thermometer

Scoreboard

Digital Clock



Energy Meter Display

Figure 2.7: Object Detection Demonstration

2.8. User Interface

The model used in this work is designed to output the detections with their scores and bounding boxes. This information need to be interpreted depending on the context. The system is designed to detect and interpret digital information from a variety of sources, such as scales, energy meters, thermometers, and digital clocks.

The input could be an image of a scale displaying a weight in kilograms, an energy meter displaying a reading in kilowatts, a thermometer displaying a temperature in celsius degrees, or a digital clock displaying the time. To convey this information to the user and represent it in a desired format, a web based user interface has been developed using React. The user interface takes an URL of an image and sends it to the backend server. After a successful response, it visualizes the detections by drawing them in the input image. For demonstration purposes, the detections are also listed in a table.

The users can also specify a score threshold, suitable for their needs. Only the detections that are higher than the specified score will be displayed. The detections are demonstrated as text, with the digits sorted by their positions both vertically and horizontally. The sorting algorithm aims to sort characters vertically first, and if they are in the same line, they are sorted horizontally. The position of digits can be different depending on the camera position and errors in the model output. This can be configured using a vertical sort threshold.

The backend server is implemented using FastAPI. It imports the TensorFlow model and label map and listens for incoming connections. When a connection arrives, it downloads the image, preprocesses it according to the model's input, and runs TensorFlow's detect function. Finally, it returns the output of the model, which is 100 detections: the class, the coordinates of the bounding box, and the score for each detection.

After sorting of the digits, the user can transform the input using regular expressions. The first input is the pattern, specifically the groups indicating how many digits have in each group. The second input is how these groups will be formatted. It takes a string formed using custom separators and capture groups corresponding to the first input. The groups are 1-indexed and they start with the dollar sign. This feature allows the user to customize the display of the detections in a specific format, such as a time or a date. For example, the pattern could be `(\d{2})(\d{2})` and the format could be `$1:$2`, where `\d` represents a digit, the parentheses represent groups. In the format `$1` represents the hour and `$2` represents the minute and they are separated with colon. This feature provides flexibility and convenience for the user to display the information in the desired format.

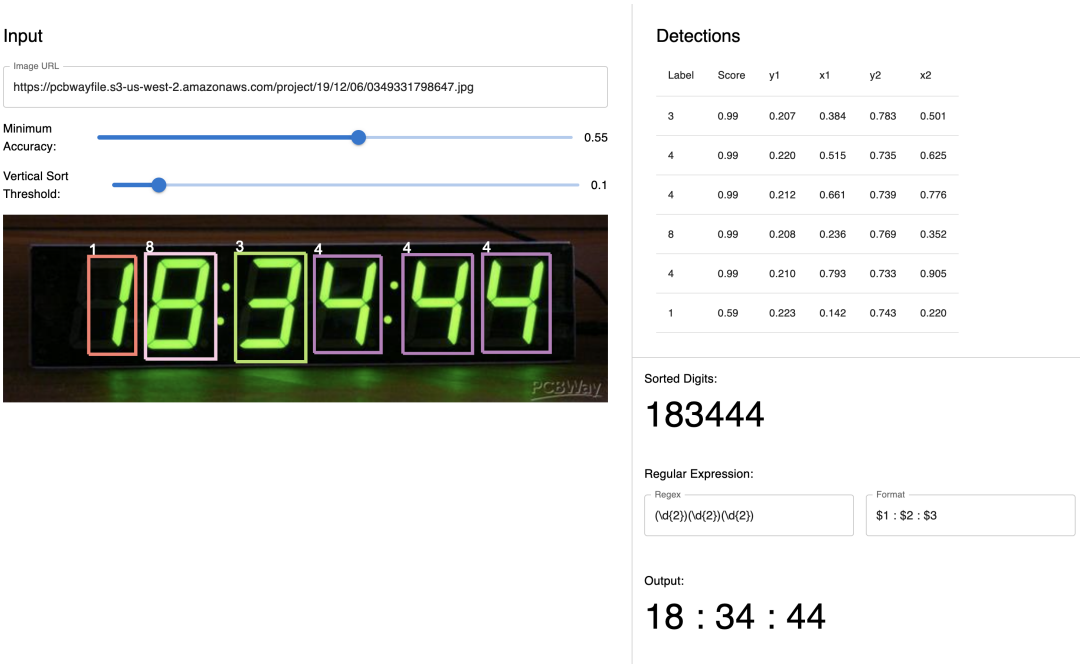


Figure 2.8: User interface demonstration on a digital clock

2.9. Evaluating Success Criteria

In summary, the final model used in the work is based on the EfficientDet D1 architecture and was trained using a combined dataset of real-world images and synthetically generated images. The results of the model using this dataset show that it is able to achieve a mean Average Precision (mAP) of 0.98 at an Intersection over Union (IoU) of 0.5. This indicates that the model is able to accurately detect and recognize digits with a high level of precision for certain displays.

Additionally, the system was tested for object detection in a video stream using a custom setup. The test set consisted of 344 frames, and the estimated frames per second (FPS) was 16.2 FPS on an Nvidia Tesla T4 GPU. This demonstrates that the system is able to process video streams in real-time, making it suitable for various applications such as monitoring.

The synthetically generated dataset, which forms part of the combined dataset, uses 15 different digital display fonts. This ensures that the model is able to recognize digits in various font styles, further increasing the model's robustness and flexibility.

Overall, the present work meets all of the success criteria, and demonstrates a reliable and efficient system for digit recognition of specific displays using OCR techniques.

3. CONCLUSIONS

In conclusion, this project presents a study on the recognition of digits from digital displays using various object detection architectures, data augmentation techniques, and transfer learning.

The experimentation was conducted using a combination of synthetically generated data and multiple open-source datasets. The results show that the use of EfficientDet D1 architecture achieved the highest accuracy of 98% in recognizing the digits. The system system is able to recognize digits from various types of digital displays, such as scoreboards, scales, energy meters, thermometers, and digital clocks. However it fails recognizing images that are too different from the training set.

The user interface allows for the sorting and interpretation of the recognized digits, and formatting using regular expression, making it easy for the user to understand and utilize the information extracted from the digital displays.

Overall, this project demonstrates the potential of deep learning techniques to accurately extract information from digital displays in real-world applications. It highlights the importance of using appropriate data augmentation techniques and architectures in achieving high accuracy, as well as providing a user-friendly interface to interpret the results.

BIBLIOGRAPHY

[1] S. Chaki, S. Ahmed, N. N. Easha, M. Biswas, G. M. T. A. Sharif, and D. A. Shila, "A framework for led signboard recognition for the autonomous vehicle management system," in *2021 International Conference on Science Contemporary Technologies (ICSCT)*, 2021, pp. 1–6. DOI: 10.1109/ICSCT53883.2021.9642525.

[2] E. Tekin, J. M. Coughlan, and H. Shen, "Real-time detection and reading of led/lcd displays for visually impaired persons," in *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, 2011, pp. 491–496. DOI: 10.1109/WACV.2011.5711544.

[3] S. Junagade, P. Jain, S. Sarangi, and S. Pappula, "Digital display recognition towards connected sensing systems for precision agriculture," in *2021 IEEE Global Humanitarian Technology Conference (GHTC)*, 2021, pp. 155–162. DOI: 10.1109/GHTC53159.2021.9612477.

[4] U. Suttapakti, T. Titijaroonroj, W. Nunsong, and D. Kakanopas, "Seven segment display detection and recognition via deep learning technique," in *2022 19th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2022, pp. 1–4. DOI: 10.1109/ECTI-CON54298.2022.9795620.

[5] K. Kanagarathinam and K. Sekar, "Text detection and recognition in raw image dataset of seven segment digital energy meter display," *Energy Reports*, vol. 5, pp. 842–852, 2019, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2019.07.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235248471930174X.

[6] frani1999, *7 segments custom dataset*, https://universe.roboflow.com/frani1999-do9am/7-segments-custom, Open Source Dataset, Sep. 2022. [Online]. Available: https://universe.roboflow.com/frani1999-do9am/7-segments-custom.

[7] P. Workspace, *Digits dataset*, https://universe.roboflow.com/phils-workspace/digits-coi4f, Open Source Dataset, Aug. 2022. [Online]. Available: https://universe.roboflow.com/phils-workspace/digits-coi4f.

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

APPENDICES

Appendix 1: Github Repository

To allow easy replication and further development of the system, all of the Jupyter notebooks and Python scripts used for synthetic data generation and model development, as well as exported models with configuration files can be found in the following git repository:

github.com/fatihkaan22/ocr-digital-display/