EFFECTIVE DEPLOYMENT AND MODEL IMPROVEMENT METHODS FOR DEEP LEARNING MODELS ON UAV DEVICE

A Thesis

by

YUNHE XUE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2020

Major Subject: Computer Engineering

ABSTRACT

In recent years, deep neural networks have outperformed traditional methods in many tasks of computer vision, such as image classification, object detection, semantic segmentation and so on. However, as the performance of deep models goes higher, the model becomes more complicated, which makes it hard to be deployed on devices with poor performance such as UAVs or Raspberry Pi. In this work, we focus on the computer vision deep learning tasks applied to UAVs in terms of improving model performance and effective deployment. In this paper, we assign two tasks for these two directions separately: 1. Object detection for small objects on rooftop images taken by UAVs. 2. Optical Character Recognition (OCR) system for videos taken by UAVs.

First, we assign the object detection for small objects on rooftop images taken by UAVs task for the model performance improvement direction. We propose several data augmentation strategies to improve the small objects detection performance. We also use model ensemble methods to improve the performance. Finally, we improve the model performance by approximately 15%.

For the effective deployment direction, we assign a OCR system for videos taken by UAVs task to it. We use Raspberry Pi as our target deployment device. We propose a two-stage OCR text detection system using EAST as detection part and CRNN as recognition part for low-resolution images and design an early-exit module to speed up the detection process. Then we use model compression methods such as pruning and quantification to process the OCR system and successfully deployed it on Raspberry Pi. For the pruning part, we also do experiments on CRNN model to find out the effectiveness of original weight initialization of the model to our pruning results and get to a conclusion that the original weight of a model is not that important for pruned model. Then we manually export the pruned model after the pruning. For the quantization part, we use pytorch's built-in static/dynamic/QAT quantization methods to quantize the two parts from float32 to int8. Finally, we deploy the OCR system on the Raspberry Pi under pytorch framework and decrease the OCR system time latency on the Raspberry Pi from about 120s to about 4-6s with little accuracy loss.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to Professor Wang, my research supervisors, for his patient guidance, enthusiastic encouragement and useful critiques of this work. I would also like to thank Professor Jiang and Professor Liu for their advice and being the committee members.

My grateful thanks are also extended to the group of VITA lab directed by Dr. Wang for their insightful suggestions.

I would also like to extend my thanks to all the staff of ECEN department for their help in offering me the resources and advice throughout my master program.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Professor Wang and Professor Jiang and Professor Liu of the Department of Electrical & Computer Engineering.

Some data collection work is done by John Hu and Samuel Leach.

All other work conducted for the thesis (or) dissertation was completed by the student independently.

Funding Sources

No outside funding was received for the research and writing of this document.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Recently, deep learning methods achieve remarkable performance in many computer vision challenging tasks, such as image classification[1][2][3][4], object detection[5][6][7][8][9][10], semantic segmentation[11][12][13], super-resolution[14][15][16] and so on. With the development of high-performance GPUs, researchers have a tendency to propose increasingly complicated deep learning models to improve the performance without regard to model complexity and computational efficiency. However, with the development of deep learning, more and more applications need to be applied on the mobile phone CPUs, and even on some devices with lower performance, such as UAVs and Raspberry Pi. On these low-performance devices, we not only need to consider the performance of the model, but also the size, time latency, and energy consumption of the model. Recently, many works emerged focusing on how to make effective model deployment. In general the low-power inference methods can be categorized as several directions, such as light-weight model structure design[17][18], model pruning[19][20][21][22] and quantization[23][24], knowledge distillation[25][26][27] and network architecture search[28][29][30]. In this work, we apply two different computer vision tasks to UAVs in terms of improving model performance and effective deployment. One task is to detect small objects from rooftop images taken by UAVs, and the other is develop a Optical Character Recognition (OCR) system for videos taken by UAVs and use model pruning and quantization method for model compression.

Many work has been done on how to design, optimize deep neural networks to get a better result. For example, ResNet[2] raised the image classification top-1 accuracy on ImageNet from 62.5%, which is achieved by AlexNet[31], to about 78.6%. But the number of parameters of ResNet[2] is about 2 times more than that of AlexNet. StyleGAN2[32] is recently proposed and can generate almost the best realistic images comparing to all other GANs[33], but it has about 10 times more parameters comparing to original GAN and it takes days to converge. These large networks make them hard to be applied to low-performance device in reality. In our work, we tried to improve the performance for small objects in object detection task, but only in data level. Our goal is to improve the performance for specific task without increasing computation complexity. To solve the small object detection problem, in this paper we propose several data augmentation methods for object detection task on images taken by UAVs. We also use ensemble several models to get a better performance.

To make the model run on a device with low performance, we should use light-weight models with less computation complexity. Also, we should compress the model as much as we can to reduce its computation complexity and decrease its time latency on low performance device with little accuracy loss. In our work, we propose a light-weight OCR system for videos we got from UAVs. OCR is short for Optical Character Recognition, and this system includes both text area detection and text recognition. We use EAST[34] model as character detect model, which is light-weight and also has good performance, and use CRNN[35] as character recognition model. We also design an early-exit module upon early stage in EAST model to speed up the detection process. Then we use model pruning and model quantization methods to reduce the computation complexity of this system. We also use CRNN[35] model to compare the effect of different initial weights to the pruning results as suggested in[21][36][37]. We found that in CRNN case, the initial weight is not as important as proposed in[21]. After compression, we decrease size of both models by 3/4. We deploy the model on Raspberry Pi and decrease the OCR system time latency on the Raspberry Pi from about 120s to about 4-6s without loss of accuracy.

2. RELATED WORK

We divide our work into two tasks: (1) UAVs object detection for small objects task, and (2) Optical Character Recognition (OCR) system deployed on low-performance device

2.1 Object Detection for Small Objects Task

In this task, we focus on how to improve the performance of Mask R-CNN[7] object detection task for small objects without introducing additional computation complexity and do not pay much attention to the efficiency. Previous related work about this task is divided into two aspects: (1) Object Detection Models (2) Data Augmentation and Model Ensemble Methods.

2.1.1 Object Detection Models

Many object detection models have been proposed in recent years. In general, they can be classified as three categories: (1) Two-stage models (2) One-stage models (3) Anchor-free models

2.1.1.1 Two-stage models

Two-stage models do the object detection task in two stages. In general, after we extract features from backbone network, the Region Proposal Network (RPN)[6] is used to propose candidate bounding boxes indicative of where the objects could be located in the image. After we got features maps and candidate bounding boxes, we extract corresponding features of these boxes from features maps, resize them into fixed size. In the second stage, the extracted aligned features are fed into the second stage branches and we got the final classification and bounding box regression results. Many models are proposed as two-stage models such as Fast R-CNN[5], Faster R-CNN[6] and Mask R-CNN[7]. They have a better performance but their speed maybe slower.

2.1.1.2 One-stage models

Instead of getting candidate bounding boxes from stage 1, one-stage models do object detection task only in one stage. Yolo[8] is a very-known one-stage object detection model. It divides the input into a N*N grid and treat each one as an anchor. Yolo passes the input through a neural network that looks similar to a normal CNN, and we get a vector of bounding boxes and class predictions in the output. SSD[10] is another one-stage detection model. The difference between SSD and Yolo is that SSD uses multi-scale feature maps as input and prior boxes with different ratios, which makes it perform better than Yolo. Since one-stage model does not need RPN part, so in general they are faster than two-stage ones.

2.1.1.3 Anchor-free models

Recently, many anchor-free object detection methods came up, such as CornerNet[38], FSAF[39], FCOS[40] and so on. Anchor-free methods outputs a map-like result, each point in the map indicates its confidence score for being in a bounding box. These methods treat each point whether in a bounding box as a classification problem, and the distances from the point to each side of the bounding box as regression problem. In general, anchor-free methods are more concise and can achieve a good result comparing to other one-stage or two-stage algorithms.

Though many one-stage detectors are more computationally efficient, in most cases, they perform worse than two-stage detectors. Mask R-CNN is also well-developed in many platforms and many anchor-free methods is not open-sourced and need to be verified. In this task we mainly focus on how to improve the model's performance and do not care much about the efficiency. After exploring many models and codes, the Mask R-CNN model was chosen as the best model for this work since it performs well among these models and is stable to use.

2.1.2 Data Augmentation and Model Ensemble Methods

2.1.2.1 Data Augmentation

Data plays an important role in deep learning tasks. In general, the more data one has, the better the deep learning model will perform. Data augmentation is a very useful method to enlarge a dataset without adding new data (thus, where more data is difficult to obtain), and it has been widely used in many deep learning areas, especially for image tasks. By translating, rotating, flipping and cropping the image data, a more diverse dataset can be obtained, and the resulting model will be more robust.

2.1.2.2 Model Ensemble Methods

Model ensemble methods[41] combine the predictions from multiple models to improve the overall performance. They operate on the similar idea. The main causes of error in learning models are due to noise, bias and variance and ensemble methods help to minimize these factors. Bagging and boosting are two common model ensemble methods. In our work, we use bagging method[41] to ensemble three models to get a better performance.

2.2 Optical Character Recognition (OCR) System Task

In this task, we not only focus on the performance of OCR system, but also its computation complexity and time latency. We use Raspberry Pi to simulate embedded computers of UAVs, and deploy the developed OCR system on it. Previous work about this task is divided into three aspects: (1) Raspberry Pi (2) OCR Models (3) Model Compression

2.2.1 Raspberry Pi

Raspberry Pi is a series of small single-board computers with limited performance. We use Raspberry Pi as our target device since the embedded computers of most commercial UAVs are not user-programmable and its limited performance is suitable to simulate the embedded computers. We use Raspberry Pi 3b+ in our work. The CPU of Raspberry pi 3b+ is quad-core A53 (ARMv8) 64-bit @ 1.4GHz with hardware Float point support. By default, the Raspbian native OS only has 32-bit. Since 64-bit ARM are also common is modern mobile devices, we install 64-bit Fedora 32 OS on our Raspberry Pi.

2.2.2 OCR Models

Optical Character Recognition (OCR) System contains two parts in it: optical character detection and optical character recognition. OCR methods can be divided into two categories: (1) Two-stage system, and (2) One-stage system.

2.2.2.1 *One-stage OCR System*

One-stage model treat OCR process as one whole part. In general, it means that the character detection part and recognition part are connected and can be trained end-to-end as a whole pipeline. Many works are focus on this direction. MaskTextSpotter[42] is a one-stage algorithm for OCR system. It can recognize text from region with any shape. But it is implemented based on Detectron2[43] and use segmentation branch to help predict bounding box, which makes it very time-consuming. FOTS[44] is a light-weight one-stage model but without good implementation. ABCNet[45] uses beizer points to fit arbitrarily shaped regions and use bezier-align method to align the feature maps of these regions between detection part and recognition part. After trying out all these one-stage algorithms, we chose ABCNet as our one-stage model choice since it is simpler and has a better performance.

2.2.2.2 *Two-stage OCR System*

Two-stage model treats OCR process as two parts separately. It means that there are two models for character detection and character recognition correspondingly. There are not share feature maps or inner connect between these two models. For example, we can use a Faster-RCNN[6] detection architecture with a rotation RPN as detection part, and use character sequence encoding (CHAR)[46] for recognition part. The input of recognition part is the output of the detection part and these two parts are trained separately.

There are many character detection algorithms exist. CTPN[47] is a classical OCR detect model, but it can only detect horizontal texts. RRPN[48] model has the ability of detecting rotated text regions, but it is developed based on Faster R-CNN and is a little complicated. EAST[34] model is a anchor-free character detection model. It is very light weight and also has a good performance. So, we choose EAST as the detection model in our two-stage model. As for the recognition model, we choose CRNN[35] as our model since it is light-weight, stable on many tasks, and performs well.

In our work we choose ABCNet as our one-stage model and EAST + CRNN as our two-stage model. We do experiments of these two models and compare their performance. Though theoretically one-stage model has less computation complexity since the recognition part just uses feature maps from detection part and does not need to do feature extract any more, in practice the time latency of ABCNet is much more than that of two-stage EAST + CRNN model. So, in the end we propose the two-stage EAST + CRNN model as the proper OCR system for low-performance device.

2.2.3 Model Compression

Model compression is a very useful method to reduce model's computation complexity. In this work, we do model compression to deploy the OCR system on Raspberry Pi and speed up the inference process. We mainly focus on model pruning and model quantization in our work.

2.2.3.1 Model Pruning

Model pruning is an efficient method to make the model smaller and decrease the computation complexity. As we know that to get a better performance, deep learning model goes deeper and deeper, and there are many redundant parameters in a model. Model pruning aims at pruning out non-important part of the model to reduce its size while maintaining its performance. Model pruning can be classified as two categories. One is fine-grained pruning and another is filter pruning. Fine-grained pruning generally results in unstructured models, which need specialized hardware or software to speed up the sparse network. Filter Pruning, also known as structured pruning, achieves acceleration by removing the entire filter. Since our goal is to deploy the pruned model on Raspberry Pi, we choose filter pruning to do model pruning since there is no specialized accelerating hardware for sparse operations on Raspberry Pi.

Many strategies about how to select pruned filters have be proposed. L1/L2 filter pruner[19] is a very useful and easy one. L1 filter pruner[19] prunes filters in the convolution layers. It will calculate the sum of the filter's absolute kernel weights from a trained model, sort the filters and prune filters with smallest values. L1 filter pruner[19] is an one-shot pruner, which means it selects channels to be pruned in one shot. AGP[20] is a gradual pruning method, it selects only a small part to be pruned each time and reaches the pruning rate in the end. Lottery Ticket hypothesis[21] indicates that there exists a small sub-network inside the original network. It says that the initial weight is important when we do pruning. ADMM[49] is another filter pruning method. It is a mathematical optimization technique, by decomposing the original non-convex problem into two sub-problems that can be solved iteratively. In this work, we choose L1 filter pruner model pruning methods. We also do experiments to verify the Lottery Ticket hypothesis[21] in our CRNN case.

2.2.3.2 Model quantization

Model quantization is another method to speed up inference process. The general idea of model quantization is to reduce the number of bits required to represent weights or activations, which can reduce the computations and the inference time. Many quantization techniques have been proposed, such as int8 naive quantization method[23] , quantization aware training[23], DoReFa-Net[24], and some extreme low-bit networks, such as BNN[50], XNOR-Net[51]. To deploy the model on Raspberry Pi, we use int8 naive quantization method implemented in Pytorch to do model quantization for those two OCR models. PyTorch supports INT8 quantization compared to typical FP32 models allowing for a 4x reduction in the model size and a 4x reduction in memory bandwidth requirements. The quantization method uses two values, scale and zero-point, to convert float operations into integer operations while maintaining the original output. These two values are observerd per channel, which means that float points in each channel has their own scale and zero point. Per-channel quantization method allows for lesser error in converting tensors to quantized values. Also we do quantization awaring training (QAT) for both models. Computations in QAT will take place in FP32 but with values clamped and rounded to simulate the effects of INT8 quantization in the training process, which provides much more accurate results. In our work, we use Pytorch in-built static quantization method and QAT[23] to quantize our models.

3. METHODOLOGY

We divide the methodology part into two parts, the first part is the object detection for small objects task, focusing on improving the performance in data level. The second part is developing a Optical Character Recognition (OCR) system task, focusing on effective model deployment on Raspberry Pi.

3.1 Object Detection for Small Objects Task

This task aims at developing an object-detection model for use in the automation of unmanned aerial roofing inspections. The proposed framework utilizes UAVs to gather high resolution images of the exterior of the potentially impacted structures. Ten object classes were selected as items of interest to insurance evaluations: Dish Antenna, Vehicle, Box Vent, Chimney, AC Unit, Solar Panel, Skylight, Ridge Vent, Plumbing Vent, and Vent Other. In this work, we improve the performance of the model with several data augmentation and model ensembling methods.

Our work consists of three parts:

1) Dataset: High resolution imagery of residential single-family structures is gathered via UAVs. This data is then labeled using the object classes selected.

2) Proposed data augmentation methods and model ensembling: The Mask R-CNN object detection algorithm is selected, data augmentations are developed for the dataset, and ensembling techniques implemented.

3) The final phase consists of training and testing the model: Appropriate evaluation methods, such as average precision, are examined. After model ensembling and data augmentation, the final mAP score was improved by approximately 10% compared to the original result.

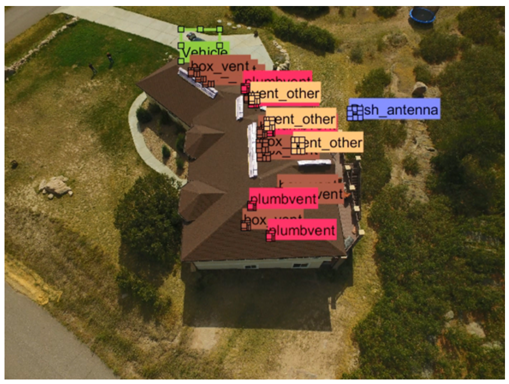
3.1.1 Dataset

The database developed for the purpose of this paper consists of high-resolution images of single-family residential structures gathered during insurance inspections. These images were gathered via UAVs. 109 different structures were photographed from a minimum of five unique angles for a total of 4,562 images. When possible, four oblique images, one from each side of the structure, and one nadir image were taken. The size of each image is 4000 x 3000 pixels. Spatial resolution varies from image to image because the structures photographed are of varying sizes. Ten object classes were defined as objects of interest during an insurance adjustment. The classes are shown in Table 3.1.

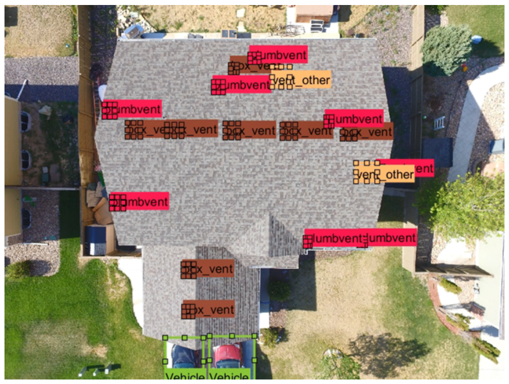
Class	Training Set	Testing Set	Total
Box Vent	7700	1844	9544
AC Unit	1021	200	1221
Solar Panel	257	371	628
Chimney	939	152	1091
Vent Other	3740	1063	4803
Vehicle	1762	477	2239
Skylight	936	172	1108
Plumbing Vent	5948	1433	7381
Dish Antenna	245	82	327
Ridge Vent	298	36	334

Table 3.1: Classes and statistic of the dataset.

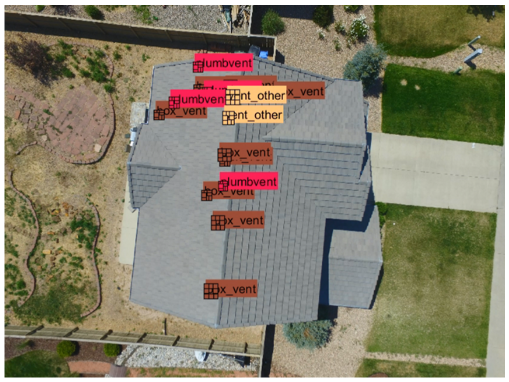
These objects appear with varying frequency in the database. Not all objects are found in each image. The images were labeled by several Texas A&M Civil Engineering students utilizing MATLAB's image labeler. A bounding box was placed around each class object in the dataset. These labels were used as the ground truth for training the Mask R-CNN model. Figure 3.1 are some examples:



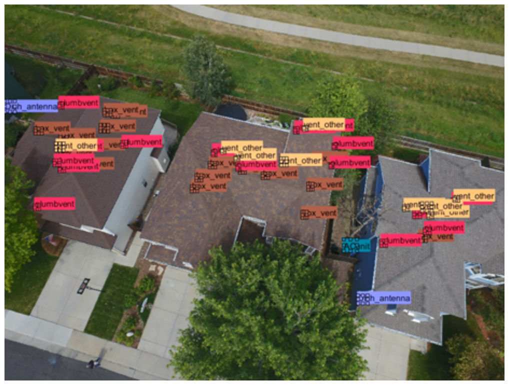
(a)



(b)



(c)



(d)

Figure 3.1: Dataset examples

3.1.2 Data Augmentation

In this work, three cropping methods are applied to improve the model performance. This combination of data augmentation methods is novel in their application to object detection. They are as follows:

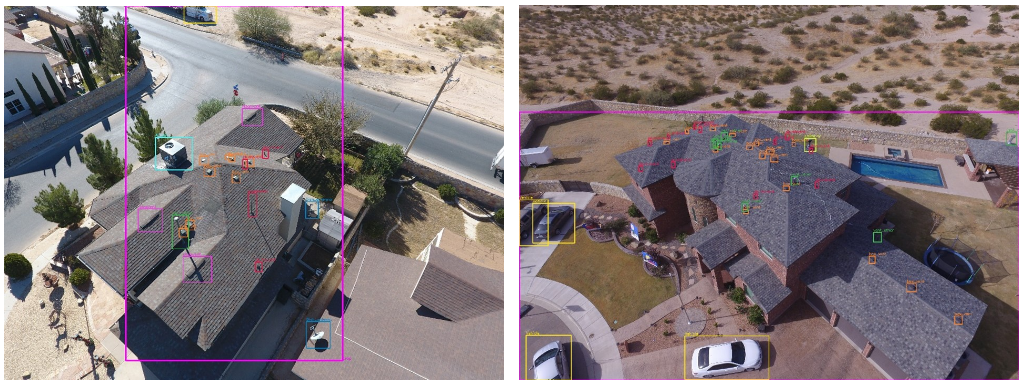
1) Augmentation 1: Cropping out the background portion of an image.

2) Augmentation 2: Cropping one high-resolution image into several low-resolution parts.

3) Augmentation 3: Cropping areas containing vent-related class objects.

3.1.2.1 Cropping out background portions

Given that the pictures are taken around one single building in most cases, there are few objects of interest at the edges and corners of the image. The surrounding background information may bring in noise and irrelevant information. If these irrelevant pixels can be cropped out, it will help the model to focus more on the objects of interest. Besides, this judgement is consistent with the reality. In most cases, we care more about what the camera focus on and do not care much about the surrounding background. In this work, the background portion is defined as the perimeter area containing no ground truth bounding boxes. Every image is cropped and added to the training dataset. Figure 3.2 contains an example of this augmentation. The following contains an example of this augmentation.

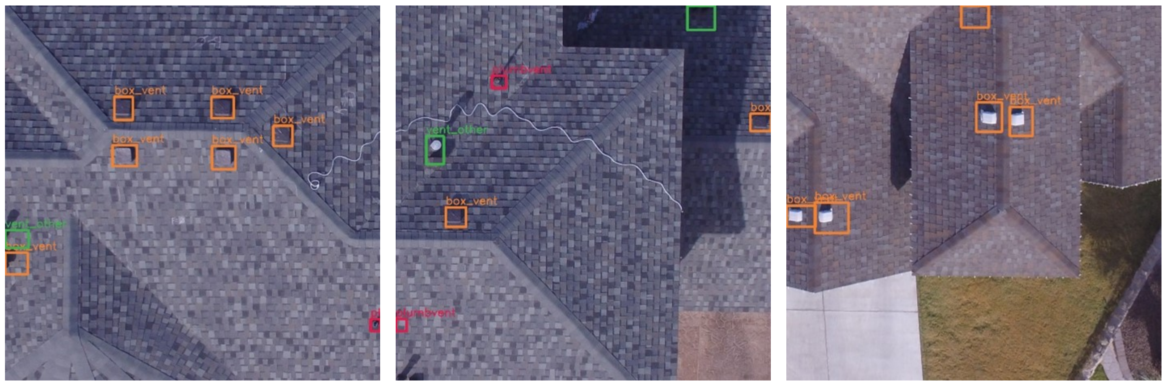


(a) (b)

Figure 3.2: Cropping Out Background Results

3.1.2.2 Cropping one high-resolution image into several low-resolution sections

The original dataset for this research consists of images of size 4000*3000. The pre-processing step of the Mask R-CNN involves reducing the shortest size of the image to 1000. Therefore, the image of size 4000*3000 would be reduced to 1333*1000. This reduces the scale of the objects present in the image. We believe that maintaining the size of the objects in the image is important and reducing the size of the objects in the image would make it harder for the model to detect objects especially for small objects. Therefore, each 4000*3000 image is split into twelve images, each of size 1000*1000, thereby preserving the size of the objects in the image. Figure 3.3 are some cropped results.

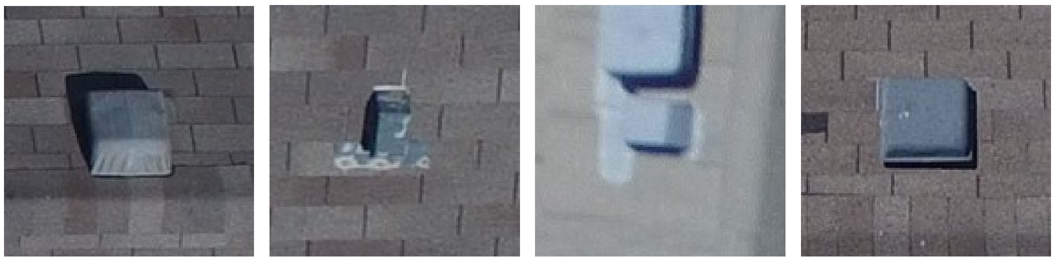


(a) (b) (c)

Figure 3.3: Cropping Low-resolution Results

3.1.2.3 Cropping areas containing vents

It was observed that the Mask R-CNN model exhibited a lower performance when classifying vent-related class objects than for objects of other classes despite being the most prevalent of classes. In order to mitigate the performance discrepancy, images containing only a singular vent object were added to the dataset. These images were generated by cropping the original dataset images until they contained one object of a vent class. This served to provide the most detailed information about the vent class to the object detector. Figure 3.4 shows some cropped results.



(a) (b) (c) (d)

Figure 3.4: Cropping Vents Area Results

3.1.3 Ensembling Using Bagging

The bagging method is a re-sampling technique used to estimate statistics on a population by sampling a dataset with replacement. The bagging method can be used to estimate a quantity of a population. This is done by repeatedly taking small samples, calculating the statistic, and taking the average of the calculated statistics. This procedure can be summarized as follows:

1) Choose a number of samples to perform

2) Choose a sample size

3) For each sample

    - Draw a sample with replacement with the chosen size

    - Calculate the statistic on the sample

Here the number of samples was chosen to be three – three different models are trained based on different sub-datasets. Each sub-dataset was sampled from the original dataset and the size of it was 0.9 times the size of the original dataset. These sub-datasets were sampled with replacement. To find the ensembled results from three models, the average of the bounding box predictions from the three models is calculated.

3.2 Optical Character Recognition (OCR) System Task

This task is the UAV video track of CVPR 2020 Low-Power Computer Vision Challenge[52]. The input of our OCR system is a video taken by UAVs. The video is taken indoor, and posters or papers with characters are posted on the wall. All words are horizontal or inclined ones. Only English letters and numbers considered. The final score is calculated by correctness / energy. We deploy the model and code on Raspberry Pi and calculate the energy consumption on it. So, we should consider both model performance and energy consumption in this task.

In this section, the methodlogy is devided into four parts:

1) Model Selection: One-Stage System or Two-Stage System: In this section we tried both one-stage model and two-stage model, and finally choose two-stage model as our final choice.

2) Early-Exit Module Design: In this section we discuss the insight of the early-exit module and how to design it.

3) Model Pruning: In this section we discuss some model pruning methods. We also do the experiments to discuss two 'conflicting' theories in our case.

4) Model Quantization: In this section we discuss the theorem of quantization and how to do quantization under Pytorch platform.

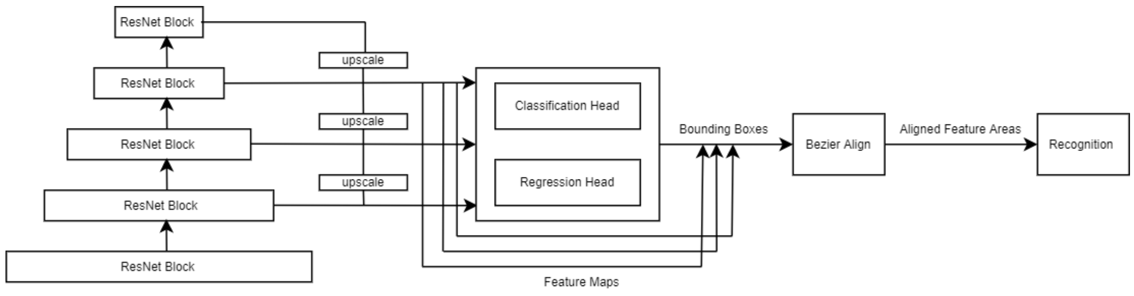
3.2.1 Model Selection: One-Stage System or Two-Stage System

In this section, we first introduce both one-stage and two-stage OCR systems. Then we discuss the problems we met of the one-stage system, and how we overcome these problems with proposed two-stage OCR system.

3.2.1.1 One-stage OCR System and Two-stage OCR System

As discussed in Section 2.2.2, in general OCR system can be divided into two categories: one-stage OCR system and two-stage system. Two-stage system means that the system treats text area detection and text recognition as two separate tasks. After the detection part gets the bounding box from the original image, we use affine transformation to transform the box into fixed size and pass it to the recognition part. One-stage model is trained end-to-end, which means that the feature map of recognition part is connected with the detection, so that we do not need to do the affine transformation operation.

After doing research, we decide to try one-stage OCR system first. Mask TextSpotter[42], FOTS[44] and ABCNet[45] are selected as candidates. We discard Mask TextSpotter first since it uses instance segmentation branch to do the detection and it has high computation complexity. We discard FOTS model then because we find it performs bad comparing to ABCNet. We finally choose ABCNet as our one-stage model. The architecture of ABCNet is shown as below:



(a)

Figure 3.5: Architecture of ABCNet.

The backbone of ABCNet is classical ResNet + FPN. The detection part uses output from three FPN layers to get results. There are two components in detection head module, and both are composed of 4 convolution layers. ABCNet is a one-stage OCR system. The input of recognition part is the aligned feature maps generated from detection part, so that the recognition part does not need to generate feature maps again.

3.2.1.2 Problems of One-Stage OCR System

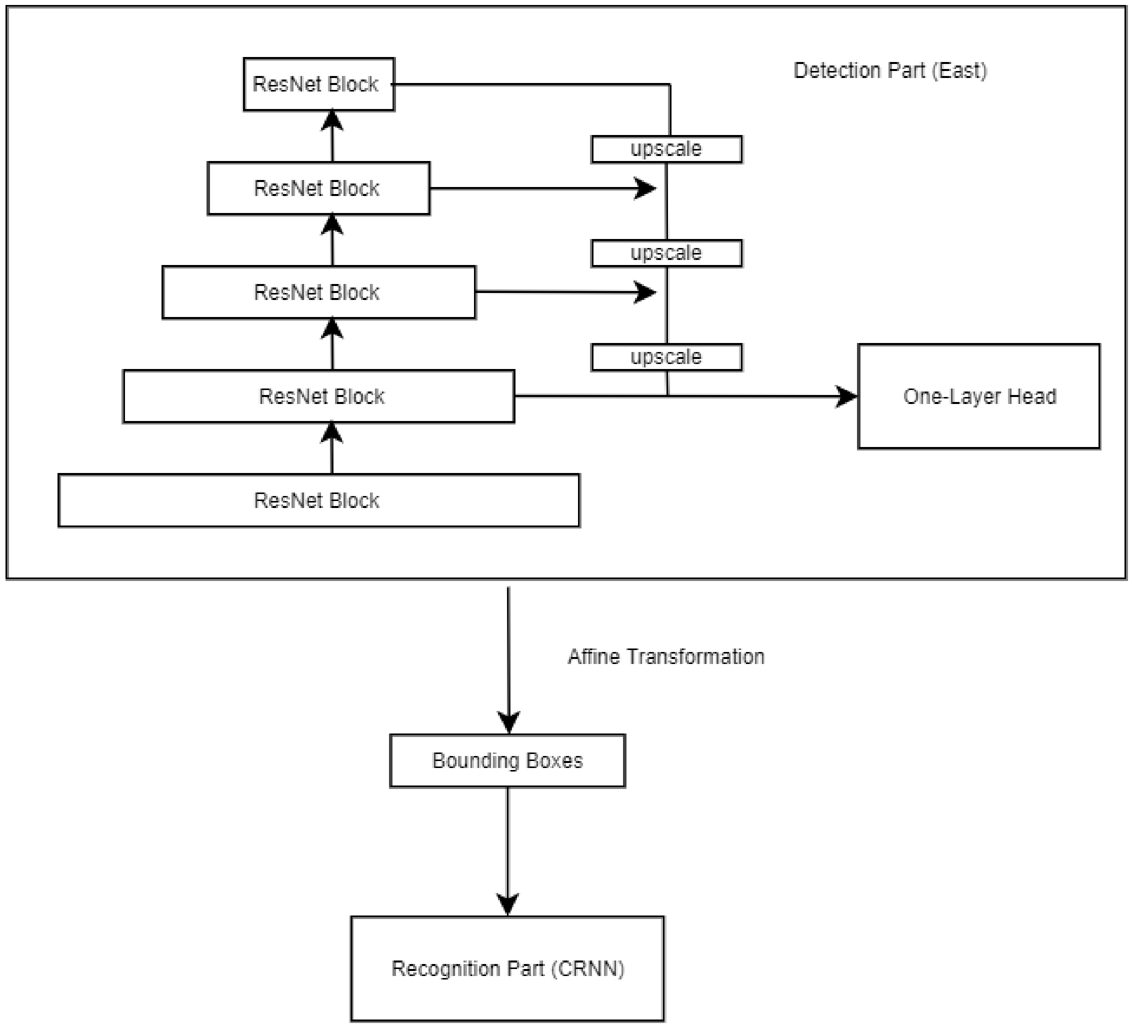
In theory the computation complexity of one-stage model should be less and its performance should be better comparing to two-stage OCR system. However, we met two serious problems when we deploy the one-stage ABCNet model on Raspberry Pi:

1) Long Inference Time. Though the one-stage ABCNet model saves computation in recognition part, it requires more computation in detection part. ABCNet uses three layers of FPN output and its detection head is heavy. Detail inference time comparison can be found in Section 4.2.

2) Bad Recognition Performance with Small Size Inputs. Our goal is to reduce the energy consumption while maintaining performance. So, we hope to resize 2K/4K frame of the video into about 270*360 size. As mentioned above, the recognition part of one-stage model gets input from feature maps we got from detection part. In most case, text area is only a small part of the original image, so if we get the features from resized small images, there will be almost no feature selected when it comes to recognition part. This will be a serious problem when the size of input image is small.

3.2.1.3 Proposed Two-Stage OCR System

To overcome problems mentioned above, we propose a two-stage OCR system. We use East[34] as text detection model, CRNN[35] as text recognition model. We use affine transformation to connect detection part and recognition part. The pipeline of our proposed two-stage OCR system is shown as below:

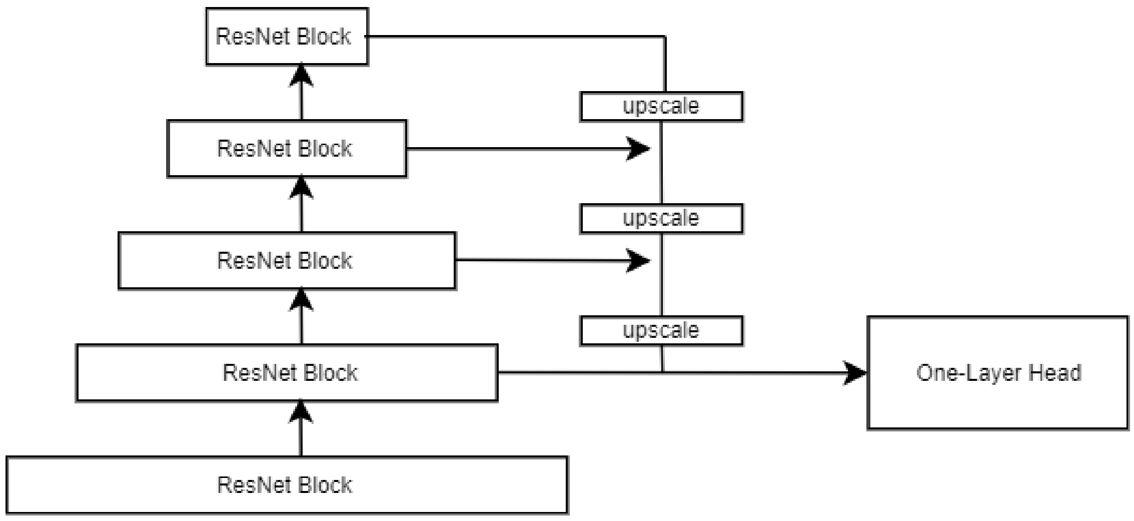


(a)

Figure 3.6: Pipeline of Proposed Two-Stage OCR System.

The proposed two-stage OCR system solves the problems mentioned in Section3.2.1.2. Detailed experiments can be found in Section 4.2:

1) Long Inference Time. We choose East[34] model as the text detection model in our two-stage OCR system. East[34] uses only one additional convolution layer as detection head, while ABCNet uses two branches composed of 4-layer convolution block as head. Also, East only uses one layer output of FPN while ABCNet uses three. This makes the inference time of East faster than ABCNet detection part. The architecture of East is shown in Figure3.8



(a)

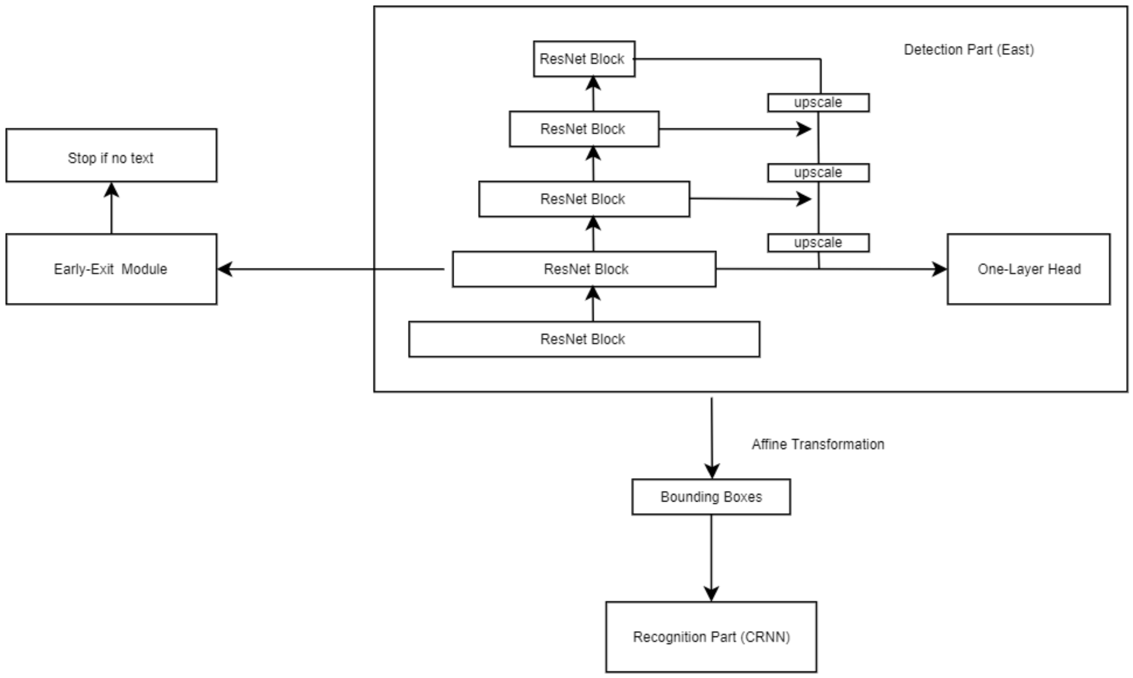
Figure 3.7: Architecture of East Model.

2) Bad Recognition Performance with Small Size Inputs. In two-stage OCR system, the input of recognition part is generated from original images. After we get bounding boxes from East model, we use affine transformations to transform the inclined box to horizontal ones, resize it to a fixed size and feed it to CRNN model. This solves the bad recognition problem with small input in one-stage model.

3.2.2 Early-Exit Module Design

The input of our OCR system is a video, and we need to recognize all English words and numbers from it. However, most of the video is composed of background and blurred frames, only a small part of it has clear text. Our insight is that, we can judge whether a frame contains text region in early stage, so that we can stop the pipeline early to save energy if there is no text in this frame. We call this module early-exit module.

We treat this problem as a classification problem. We add the early-exit module upon different blocks of ResNet, and use the feature maps as input to the module. We try several different settings to find a best one. Experiment result can be found in Section4.2. In the end we choose to add the module upon the second block of ResNet, and the module includes two convolution layers. After adding the early-exit module, the pipeline looks like:



(a)

Figure 3.8: Pipeline of Proposed Two-Stage OCR System.

3.2.3 Model Pruning

To speed up the inference process further, we do model pruning after we get the trained model. In general, mdoel pruning have two categories: structured pruning and unstructured pruning. Structured pruning mainly prunes out structures like channels or layers. Structured pruned models have new structures and are easy to be deployed. Unstructured pruning makes the model sparse and need hardware support to deploy and speed up. So, in this work, we focus on structured pruning since there is no such hardware on Raspberry Pi.

In this section we also use CRNN model to do some pruning experiment with different settings. First we compare the performance of one-shot and adaptive gradual pruning methods. Then we use one-shot method to do some experiments to verify the effect of initial weights on pruned models. In [21], the author claims that we should keep initialize the network with original weight and prune out weights gradually. The author indicates that the original weights are important and the performance of pruned model will drop if we re-initialize the weights. However, in [36] the author proposed an opposite opinion. They claimed that the weight is trivial and the architecture is what really matters. In our case, we use CRNN to do some experiments to verify that the weight is not that important in CRNN case. More detailed results can be found in Section 4.2

After that, we choose a proper sparsity rate and use L1 Filter Pruner method to prune both East and CRNN model with little performance drop. After we got masked pruned model, we export the actual pruned model by pruning out channels with zero weight and only non-zero weights left.

3.2.4 Model Quantization

Quantization is a method to do computation and store weights in lower bitwidths instead of float 32 bitwidths. This allows us to get a more concise model. In this work, we focus on how to do quantization in Pytorch platform.

Pytorch supports converting the model trained in Float32 in Int8 by post-quantization. It also supports quantization aware training by adding fake-quantization modules in both forward and backward passes. For Raspberry Pi case, Pytorch has qnnpack backend to accelerate the quantized model on ARM CPUs.

We do static post quantization for all convolutional and fully-connected layers and dynamic post quantization for LSTM module in CRNN model. After doing quantization, the inference time decreases about 10 times on Raspberry Pi. Inference time comparison for some settings is shown in Table 3.2.

Setting	Input Size	Quantized	Inference Time Per Frame
ABCNet	560*960	No	120s-130s
ABCNet	560*960	Yes	12s-15s
East+CRNN	560*960	Yes	12s-15s
East+CRNN	270*480	Yes	4s-8s

Table 3.2: Inference Time Per Frame for Different Settings.

As shown in Table 3.2, we can see that model quantization with backend support decreases the inference time of the model in a large extent. Even though the inference time of our two-stage OCR system is almost the same with the one-stage ABCNet model, we can reduce its inference time further by resizing input images into smaller size. ABCNet model performs much worse than the two-stage OCR system when the size of input is smaller than 560*960. So, two-stage OCR system is better than one-stage ABCNet model in this sense. More detail experiment results can be found in Section 4.2.

4. EXPERIMENTS

In this section, we do experiments for two tasks separately. The first part is the result for the object detection for small objects task. The second part is for the Optical Character Recognition (OCR) system task.

4.1 Object Detection for Small Objects Task

In this part, we first do experiments to verify the effect of data augmentation methods, and then shows the result of model ensembling.

4.1.1 Data Augmentation Results

In this section, we do experiments with several different data augmentation settings. Data augmentation methods are defined in Section 3.1.2. We define methods as following:

1) Method 1: Train the model only with original images.
2) Method 2: Train the model with original images and Augmentation 1.
3) Method 3: Train the model with original images, Augmentation 1 and Augmentation 2.
4) Method 4: Train the model with original images, Augmentation 1, Augmentation 2 and Augmentation 3.

We do fine-tune with our training dataset and test the performance with test dataset. All data augmentation methods are only applied on training dataset. First, we do experiments with Method 1, 2 and 3. The results are shown in Table 4.1:

Method	mAP	Chimney	Vent Other	Vehicle	Antenna	Box Vent	Skylight	AC	Plumb Vent	Panel
Method 1	49.97	60.34	17.79	91.33	45.09	30.02	88.36	59.64	20.25	56.56
Method 2	52.55	45.46	23.80	88.16	71.96	32.11	76.93	43.17	29.27	62.07
Method 3	55.65	56.04	30.40	91.05	71.74	35.84	83.89	38.02	30.83	63.04

Table 4.1: mAP Results with Data Augmentation Method 1, 2 and 3.

As shown in 4.1, we got a mAP score of 49.97 on test dataset if we do not do any data augmentation. After we doing Augmentation 1 as suggested in Section 3.1.2, the average mAP increases from 49.97 to 52.55, and it increases further to 55.65 after we doing Augmentation 2. We can see that the mAP scores of small objects such as Vent Other, Box Vent keep increasing after data augmentation.

After the first experiment, we got more labeled data from Civil Engineering students. Then we got new training and test dataset from new data, re-do the experiment with Method 1, 3, 4 to verify the effect of Augmentation 3. Experiment results are shown in Table 4.2.

Method	mAP	Chimney	Vent Other	Vehicle	Antenna	Box Vent	Skylight	AC Unit	Plumb Vent	Panel
Method 1	54.07	39.38	31.90	88.53	68.21	33.19	78.76	44.86	35.89	65.99
Method 3	59.10	66.08	33.09	99.14	80.52	35.79	80.78	43.32	33.76	67.09
Method 4	60.00	66.76	36.39	92.77	78.34	36.95	83.65	48.79	35.55	60.83

Table 4.2: mAP Results on New Dataset with Data Augmentation Method 1, 3 and 4.

Results in 4.2 shows that the after doing Augmentation 3, the mAP scores of small objects such as Vent Other, Box Vent, Skylight and so on are improved. The average mAP score also increases a little comparing to Method 3. The performance of the Mask R-CNN model is improved approximately 10% after implementing data augmentation techniques as shown in Table 4.1 and Table 4.2.

4.1.2 Model Ensembling Results

In this section, we show the results of model ensembling. Three models were trained using different sub-datasets with a bootstrap method, and they are all trained with Method 4 as mentiond in Section 4.1.1. Then, their results were ensembled to achieve enhanced performance. To find the ensembled results from three models the average of the bounding box predictions from the three models is calculated. The corresponding bounding boxes in each of the three models that have high intersection-over-union are only considered while averaging so that similar bounding boxes are averaged.

Method	mAP	Chimney	Vent Other	Vehicle	Antenna	Box Vent	Skylight	AC Unit	Plumb Vent	Panel
Model 1	61.22	70.87	37.90	91.80	70.83	41.29	83.10	47.82	39.98	67.42
Model 2	61.69	67.72	36.84	93.58	73.92	44.84	81.56	53.75	40.49	62.54
Model 3	63.93	68.61	40.71	92.51	79.69	44.24	88.08	52.93	42.78	65.78
Ensemble	65.31	72.63	40.94	93.64	78.28	47.95	86.49	54.33	43.96	69.53

Table 4.3: mAP Results on New Dataset with Data Augmentation Method 1, 3 and 4.

After ensembling, the average mAP result achieves 65.31, which is better than the best one from three models. In conclusion, after doing data augmentation and model ensembling, the performance of Maks R-CNN on this task is improved by about 15%.

4.2 Optical Character Recognition (OCR) System Task

In this part, we do experiments include the following:

1) Raspberry Pi setup.

2) Performance comparison between two-stage and one-stage OCR system.

3) Early-exit module design and results.

4) Model pruning methods comparison and initial weights verification.

5) Final submitted results

Notice that all the submitted results shown in this section have been quantized since the inference time of the original float32 model is much more than the quantized int8 model.

4.2.1 Raspberry Pi Setup

We use Raspberry Pi 3b+ with 1GB RAM as our target device. The CPU of Raspberry Pi 3b+ is ARMv8 64-bit with hardware Float point support. But the Raspbian native OS only has 32-bit. So, we replace the operating system with a 64-bit OS called Fedora 32. Then we recompile pytorch and torchvision wheels for the platform.

4.2.2 Comparison Between Two-Stage and One-Stage OCR System

We use the submitted results to compare the performance of two-stage and one-stage OCR system. The detailed results can be found in Table 4.4.

System Type	Input Size	Quantized	Energy	Accuracy	Performance Score
ABCNet(One-stage)	540*960	Yes	0.784	0.08535	0.128
ABCNet(One-stage)	360*640	Yes	0.741	0.20623	0.263
ABCNet(One-stage)	270*480	Yes	0.640	0.12458	0.187
East+CRNN(Two-stage)	270*480	Only East	0.685	0.31857	0.503

Table 4.4: Submitted Results for Two Kinds of OCR systems with Different Settings.

We can see from the first two rows in Table 4.4 that, when the size of input image is smaller, then performance score of ABCNet increases. This is because the organizer requires the program to be finished in a limited time. When the input size gets smaller, we can process more frames in a limited time, which means that we are more likely to recognize clear text results from the video. So, we get a better result when we make the input smaller.

However, when we make the input size to 270*480, the performance drops. Though the energy keeps decreasing but the accuracy drops a lot. This is caused by the problem mentioned in Section 3.2.1.2. The recognition part of ABCNet suffers from small inputs. In comparison, the performance of two-stage OCR system, which is the last row in Table 4.4, is much better than the one-stage OCR system. The energy score of the two-stage OCR system is a little higher than that of ABCNet, but it has much higher accuracy, which makes its performance score much better than that of ABCNet.

In conclusion, we decide to use East+CRNN two-stage OCR system in the rest of our experiment.

4.2.3 Early-exit module design and results

In this section, we discuss the design of early-exit module and compare their performance to get the best one. We treat this task as a classification problem. We use feature maps generated from ResNet blocks as input, and a sigmoid function output as classification score. If the score is larger than 0.5, we think the predicted label is 1, otherwise the label is 0.

4.2.3.1 Dataset

We generate a dataset with binary labels. We set label 1 to frames with clear text region, and label 0 to frames without text region or with blurred text region. Some dataset examples are shown in Figure 4.1:

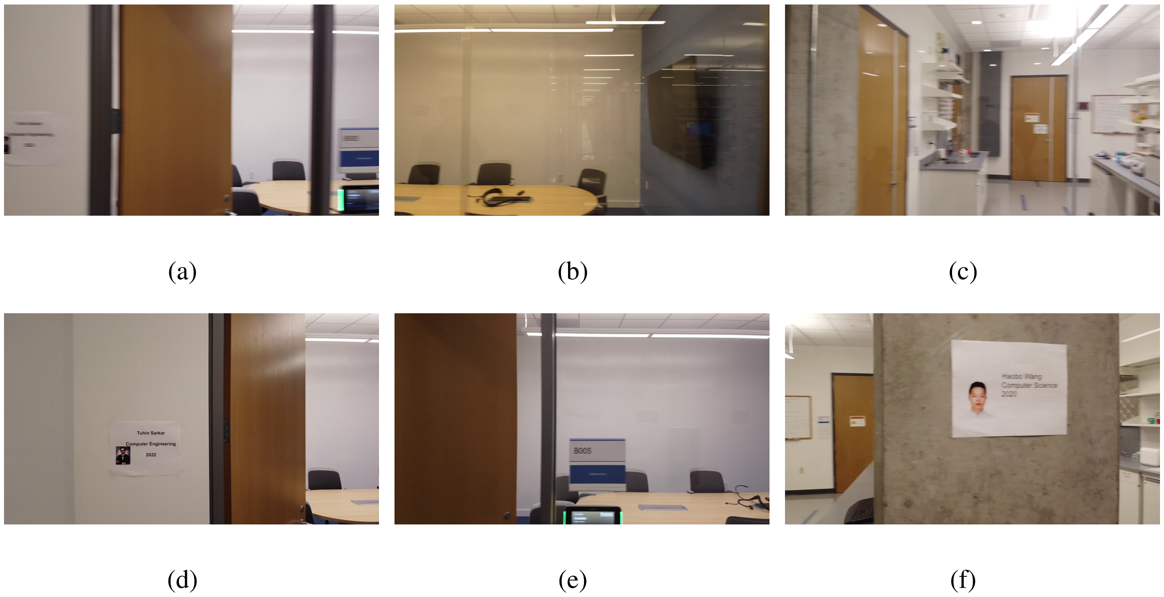


Figure 4.1: Dataset for Early-Exit Module. Figures in the first row have label 0. Figure in the second row have label 1.

4.2.3.2 Module Design and Results

We define the following terms in this section:

1) Conv: Module{conv3x3(stride=1, padding=1, dilation=1), BatchNorm2d, Relu}

2) Down: Module{conv3x3(kernel size = 3, stride = 4)}

3) FC: Module{Avagepool, Flatten, Fully Connected}

We use these modules as blocks to build up our early-exit module upon the output of different ResNet blocks. We add the early-exit module after different blocks of ResNet of East. We fix the weight of ResNet and only train the early-exit module part. We use AUC, Recall and Precision to measure the performance. The results are shown as below:

Module Setting	ResNet Block	AUC	Recall	Precision	Loss
Down+Conv+FC	3rd	0.9881	0.8996	0.9788	0.1557
Down+FC	3rd	0.9975	0.9270	0.9880	0.1391
Down(Kernel 1)+FC	3rd	0.9478	0.8201	0.9580	0.3397
Down+Conv+FC	2nd	0.9247	0.8276	0.8947	0.3579
Down+FC	2nd	0.9235	0.8681	0.8578	0.3703
Conv+Maxpool+Down+FC	2nd	0.9447	0.8471	0.9464	0.2973
Down+Conv+FC	1st	0.8727	0.7076	0.8872	0.4764

Table 4.5: Results for Adding Modules Upon ResNet Block

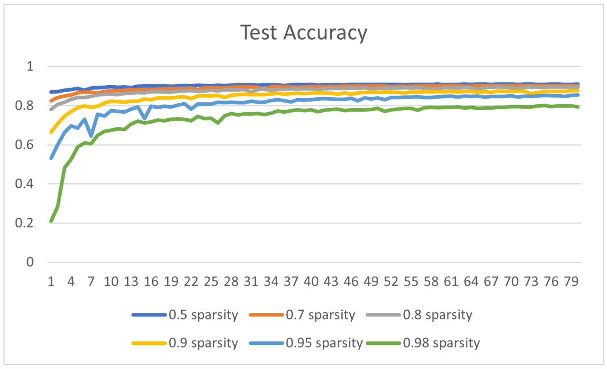
We hope the early-exit module to have high AUC value and reduce as much computation as possible. After comparing all results we got in Table 4.5, we finally decide to add a Conv+Maxpool+Down+FC module upon the 2nd block of ResNet.

4.2.4 Model Pruning Results

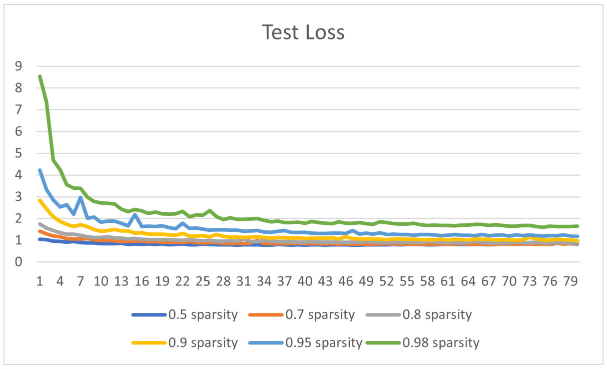
In this section, we use the CRNN model to do the experiment. We first prune CRNN with different sparsity rate using L1 Filter Pruner Algorithm[19]. Then we use CRNN model to compare two pruning strategies: one-shot pruning and adaptive gradual pruning. We also use CRNN model to verify the effect of original initial weights. All experiments we done in this section use MJSynth Dataset[53], we select 3 million training data and 50k test data from the overall training and test dataset separately.

4.2.4.1 Pruning Results

We first use L1 Filter Pruner[19] in NNI[54] to prune the CRNN model with different sparsity rate. The sparsity rate means the percentage of zero-weight in each filter, which means that the model is smaller when the sparsity rate is higher. The visualization result can be found in Figure 4.2. The best performance is listed in Table 4.6.



(a)



(b)

Figure 4.2: Visualization Results for Different Sparsity Rate

Sparsity Rate	Test Loss	Test Accuracy
0 (No Prune)	0.789823	0.910276
0.5	0.772062	0.911013
0.7	0.798839	0.903898
0.8	0.869268	0.895986
0.9	0.985479	0.877312
0.95	1.173275	0.854293
0.98	1.602774	0.800941

Table 4.6: Results for Different Sparsity Rate

We get these results by using one-shot pruning strategy. All pruned models are initialized with the trained model (model in the first row in Table 4.6). We see from above that the accuracy and loss begin to drop when sparsity rate is larger than 0.7. So, in the end we choose 0.7 as the sparsity rate to prune our CRNN model to maintain its performance.

4.2.4.2 *Comparison between One-Shot and AGP*

In last section, all results are got by applying one-shot pruning strategy. It prunes the weight with small magnitude in one-shot. For example, it prunes out top 50% channels of each filter if we set sparsity rate as 0.5. However, as proposed in [20], the author claimed that it should be better if we prune out the channel step by step. We do experiments to see whether this adaptive gradual pruning (AGP) strategy performs better than the one-shot strategy. Experiments results are shown in Table 4.7

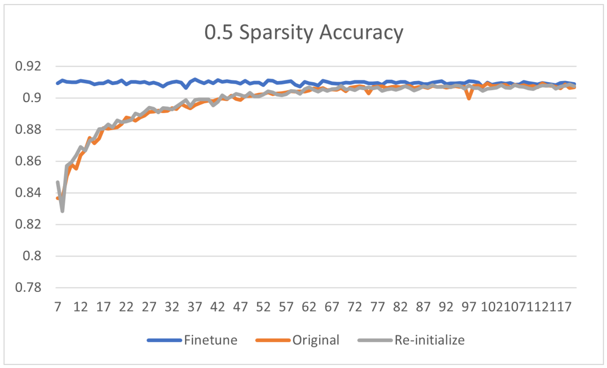
Sparsity Rate	Strategy	Test Loss	Test Accuracy
0 (No Prune)	NA	0.789823	0.910276
0.8	One-shot	0.869268	0.895986
0.8	AGP	0.905571	0.887217
0.9	One-shot	0.985479	0.877312
0.9	AGP	1.086188	0.864118
0.95	One-shot	1.173275	0.854293
0.95	AGP	1.381404	0.827926
0.98	One-shot	1.602774	0.800941
0.98	AGP	1.832472	0.762893

Table 4.7: Comparing One-Shot with AGP

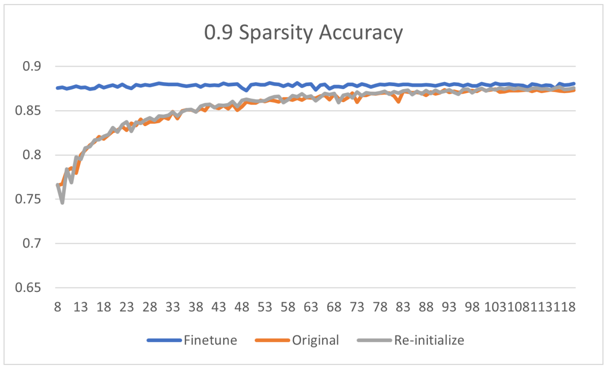
We can see from Table 4.7, AGP strategy performs worse than one-shot strategy in most cases, and it perform even worse when the sparsity rate increases. So, we use one-shot pruning strategy as our final choice.

4.2.4.3 Is Original Initial Weight Matters?

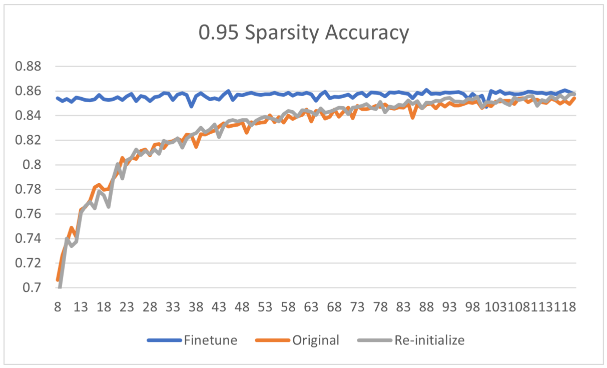
During research, we have some interesting founding. In [21], the author claimed that when pruning models, the original initial weights of the model is crutial. However, author of [36] proposed a contrary opinion, indicating that the weight does not matter and only the architecture of the pruned model is the key point. We do some experiments on CRNN to test these two contrary theories. We prune out the channels with zero-weight and export the final actual pruned model with new architecture. We initialize the new model with different initialization methods and test their performance. Figure 4.3 shows the visualization results we got from different



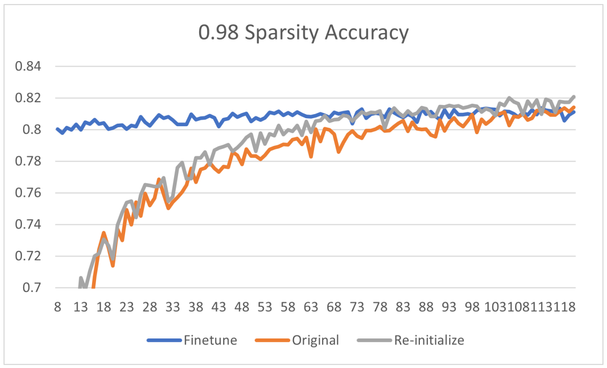
(a)



(b)



(c)



(d)

Figure 4.3: Visualization Results for Different Initialization Methods. Initialize the new pruned model with (Finetune): trained weights. (Original): original initialization weights. (Re-initialize): re-initialized weights.

We can see from figure above, the performance of the model with re-initialized weights is as good as, or even better than the one with original initialization weights. The two models trained from scratch also reach the same accuracy comparing to the finetuned one. It indicates that the original initialization weights does not matter in our CRNN case. In the end, we choose the architecture we got from sparsity 0.7, re-initialize it and train it from scratch.

4.2.5 Final Submitted Results

We prune East model the same step as CRNN model. After doing model pruning to both two models, we do model quantization to them. We got many submitted results as shown in Table 4.8.

Finally we received a second price in this challenge.

No.	Score	Energy	Accuracy	Early-Exit	Prune	Quantization
1	0.503	0.685	0.31857	NA	NA	Only East
2	0.703	0.740	0.49263	NA	NA	All
3	1.146	0.603	0.68048	Yes	NA	All
4	1.592	0.440	0.66716	Yes	Only CRNN	All
5	1.693	0.447	0.71229	Yes	All	All

Table 4.8: Submitted Results

5. CONCLUSION

In this work, we focus on two computer vision tasks applied to UAVs in terms of improving model performance and effective deployment. In the first task, we propose several different data augmentation methods, use model ensembling method to improve the performance of mAP score by about 15%. In the second task, we propose an effective two-stage OCR system and successfully deploy it on Raspberry Pi. We design early-exit module to ignore non-text frames with little computation. We use model pruning and quantization methods to compress the model to speed up its inference time. We also do experiments to show that in our CRNN case, the original initialization weights are trivial. Our work explores the effective deployment method on low-performance device of deep learning models and the improvement of models in data level.

REFERENCES

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[5] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[6] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, pp. 91–99, 2015.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[9] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[11] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[12] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[13] G. Lin, A. Milan, C. Shen, and I. Reid, “Refinenet: Multi-path refinement networks for high-resolution semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1925–1934, 2017.

[14] C. Dong, C. C. Loy, K. He, and X. Tang, “Learning a deep convolutional network for image super-resolution,” in *European conference on computer vision*, pp. 184–199, Springer, 2014.

[15] C. Dong, C. C. Loy, and X. Tang, “Accelerating the super-resolution convolutional neural network,” in *European conference on computer vision*, pp. 391–407, Springer, 2016.

[16] T. Tong, G. Li, X. Liu, and Q. Gao, “Image super-resolution using dense skip connections,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4799–4807, 2017.

[17] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[20] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[21] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

[23] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[24] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[25] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[26] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," *arXiv preprint arXiv:1910.10699*, 2019.

[27] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[30] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," *arXiv preprint arXiv:1904.00420*, 2019.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[32] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[34] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: an efficient and accurate scene text detector," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 5551–5560, 2017.

[35] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 11, pp. 2298–2304, 2016.

[36] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[37] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," *arXiv preprint arXiv:2003.02389*, 2020.

[38] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 734–750, 2018.

[39] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 840–849, 2019.

[40] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 9627–9636, 2019.

[41] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999.

[42] P. Lyu, M. Liao, C. Yao, W. Wu, and X. Bai, "Mask textspotter: An end-to-end trainable neural network for spotting text with arbitrary shapes," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–83, 2018.

[43] F. M. W.-Y. L. YuxinWu, Alexander Kirillov and R. Girshick, "Detectron2." `https://github.com/facebookresearch/detectron2/`.

[44] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan, "Fots: Fast oriented text spotting with a unified network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5676–5685, 2018.

[45] Y. Liu, H. Chen, C. Shen, T. He, L. Jin, and L. Wang, "Abcnet: Real-time scene text spotting with adaptive bezier-curve network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9809–9818, 2020.

[46] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014.

[47] Z. Tian, W. Huang, T. He, P. He, and Y. Qiao, "Detecting text in natural image with connectionist text proposal network," in *European conference on computer vision*, pp. 56–72, Springer, 2016.

[48] J. Ma, W. Shao, H. Ye, L. Wang, H. Wang, Y. Zheng, and X. Xue, “Arbitrary-oriented scene text detection via rotation proposals,” *IEEE Transactions on Multimedia*, vol. 20, no. 11, pp. 3111–3122, 2018.

[49] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, “A systematic dnn weight pruning framework using alternating direction method of multipliers,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 184–199, 2018.

[50] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.

[51] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European conference on computer vision*, pp. 525–542, Springer, 2016.

[52] “Low-power computer vision challenge 2020 cvprworkshop, uav video track.” `https://lpcv.ai/2020CVPR/video-track`.

[53] “Mjsynth dataset.” `http://www.robots.ox.ac.uk/vgg/data/text/`.

[54] “Nni(neural network intelligence).” `https://github.com/microsoft/nni`.