S	
E 1	
E 2	
For Office Use Only	

Masters Project Final Report

(MCS)

2019

Project Title	Optimized Real-time object detection on Mobile phones
Student Name	Kuganathan Rajeevan
Registration No. & Index No.	2017/MCS/062 17440623
Supervisor's Name	Dr.D.A.S.Atukorale

For Office Use ONLY

Optimized Real-time object detection on Mobile phones

A dissertation submitted for the Degree of Master of Computer Science

Kuganathan Rajeevan
University of Colombo School of Computing
2020

Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: Kuganathan Rajeevan

Registration Number: 2017/MCS/062

Index Number: 17440623

_____

Signature: Date:

This is to certify that this thesis is based on the work of Mr./Ms.

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name:

_____

Signature: Date:

Abstract

The usefulness and feasibility of optimized object detection models in real time applications for mobile phone and embedded device platforms is analyzed in the research. Such models will be useful as the number of mobile devices and IoT devices and the applications are widely increasing.

The problem in mobile hardware platforms is that the CPU is not designed to do heavy operations as PC hardware to conserve power. To achieve real-time detection it is required for a different model or optimizing and improving the existing model. The objective of the research is to find a solution which gives real-time performance without compromising accuracy of the results on mobile phones.

Existing object detection techniques were analyzed including Deterministic approaches and Machine learning models. Through literature review it is found that YOLO machine learning model can be improved to make it suitable for mobile hardware.

Various techniques applied and tested including model trimming, add redundant layers, changing layer sizes and use different tools to compress the trained model. In the results it is found that combining such techniques improve the performance while not sacrificing much accuracy.

Tests were done on Mobile Phone and PC hardware with same data set and compared. And also test with different models on Mobile phone also compared. The results show around 8 times improvement on inference time on mobile devices than using base models.

A proof of concept application created by training the street sign data and used in android camera application to detect street signs on real time. The results show significant effectiveness on optimized object detection models on mobile phones.

Table of Contents

List of Figures

Index of Tables

Abbreviations

IoT	**Internet of Things**
FLOPS	**Floating Point Operations per Second**
CNN	**Convolutional Neural Network**
DNN	**Deep Neural Network**
CPU	**Central Processing Unit**
GPU	**Graphical Processing Unit**
SSD	**Single Shot Multi-box Detector**
RoI	**Region of Interest**

Chapter 1: Introduction

1.1. Introduction

In recent years the number of IoT devices used in day to day applications are in several billions. Advanced semiconductor technologies make it possible to use smaller processing units with ability to do multiple G-FLOPS of calculations. This much processing ability enables many new applications for these kind of devices. There is a considerable amount of IoT devices that use cameras as a hardware component, which can be used as a sensor to learn the environment and the data can be used in multiple ways. For example, Drones, Biometric Security cameras, Surveillance cameras, Vehicle Dash cams, Face recognition locks etc.

Similar to the development of hardware technologies, in the field of AI also there are several new techniques introduced in multiple fields. Huge amount of Big Data and powerful hardware platforms for processing the data allows us to create large Neural Network models to perform various predictions and operations.

Object detection is one of the application of Computer Vision. When an image with various objects with background is given, it is possible to detect the location of the objects and classify them to a predefined classes is called object detection. Even though there are several methods proposed, the methods which use CNN are most efficient in learning new objects in different domains. CNN is a type of Deep neural network which use Convolution operation in some layers. Even though CNN operations are computationally expensive, they are efficient in learning and generalizing the data set of images.

IoT devices with camera hardware with enough computational power can be combined with an Object detection model can be used for various applications, Such as Face detection, Person tracking drones, Robots which can recognize objects to perform various tasks, Self Driving vehicles, etc.

1.2. Research Problem

The basic problem with CNN based Object detection models is computational complexity. In a general purpose CPU an object detection model can perform at an acceptable level, but most general IoT device's CPU is not designed for the purpose of computation intensive tasks.

For example, a Raspberry PI 4 Model B device has Quad core Cortex-A72 CPU, which can run a lightweight operating system with considerably high FLOPS. It can be taken as a typical example for a low powered device as it uses less powerful CPU and there were various applications built on top of this hardware platform. When it is applied to a simple face recognition task the frame rate drops to less than 5 FPS, which is very low for a real time application. Nowadays mobile phones come with optimized GPU for specific tasks such as

Face detection and recognition, still it requires high power to operate and not suitable for continuous operation.

Another issue with such devices are the power-supply. Computationally intensive tasks such as object detection will require a lot more than usual power supply. If the device is battery operated, it will become another bottleneck for the usage of such applications. Finding an optimized object detection model with reasonable accuracy, which can perform object detection with acceptable frames per seconds is crucial for real time object detection applications on IoT devices.

1.3. Motivation

There are efficient CNN based Object detection models available such as R-CNN, Fast-RCNN, Faster R-CNN, SSD, YOLO. There is a considerable amount of research already done on improving accuracy of the models. Also these models are using CNN as a back bone for their object detection tasks. But it is not possible to directly run the models on low end CPU's with acceptable performance.

There are object detection applications which don't require high accuracy but real-time performance is important. For example, a vehicle detection IoT based device will receive continuous frames as input, So it is enough for the application to detect a vehicle in-front in one of the few frames within a time interval to alert the driver about the vehicle.

An object detection model which can perform faster on these CPU's without sacrificing much accuracy will be useful in these kinds of scenarios. By optimizing the models by applying various techniques to improve the performance it is possible to create an object detection model which can be used in applications on low end CPU devices.

1.4. Scope of the study

The research will mainly focus on object detection models for low level CPUs, and the performance will be compared with the same model on PC CPU. As the analysis on performance and accuracy already done on CPU based devices[1] for the most of machine learning object detection models with COCO dataset, in this research, the analysis will be extended to Mobile CPU but only on the selected YOLO v3 model as it is performing well on PC.

For optimization various techniques including model trimming, remove CNN layers, introduce redundant connections, Floating point optimizations will be studied on small dataset and the techniques which have significant improvements only will be chosen to train the whole dataset for evaluation.

Because COCO dataset is large enough with wide range of classes and it is used to compare other object detection models[1], it will be used as benchmark dataset in this research aswell for comparing the results with previous analysis.

There is no programming language level optimization will be considered in this research. The standard Java based Android programming and Python based Tensorflow 2.0 framework will be mainly used to construct the model and evaluate.

Chapter 2: Review of the Literature

2.1. Introduction

In this chapter literature of the various existing models of object detection architectures and the internal CNN implementations are analyzed and their performance metrics will be compared. Here the main focus of research is on CNN based models because DNN models easily outperform manual algorithms in generalization and robustness. Earlier implementations use manually designed feature extractors with some classifiers such as SVM. E.g.: SIFT, HOG and Haar-like. But the issue with these implementations was robustness.[4] The models will not generalize over a wide range of similar class objects. Similar objects with different lighting, rotation, scaling, position will reduce the probability of detecting the object.

Object detection involves Object classification and Object localization. In an image frame the model should locate the object from the background and classify the object from a set of labels.

Usually object detection pipeline involves three stages.

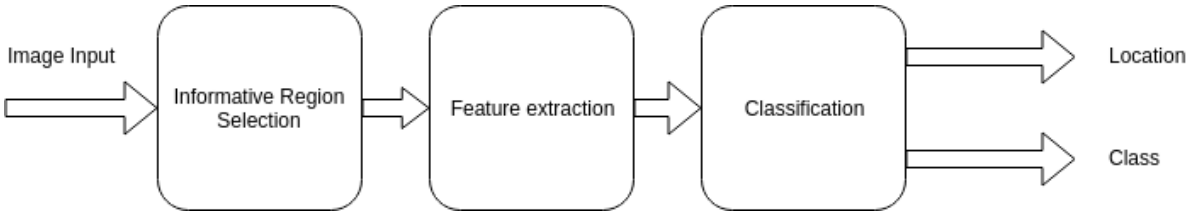

Figure 1: General Object detection Pipeline

Informative Region Selection: In an image frame may contain several objects and background. The object of interest can be any size and aspect ratio. The naive way of locating the object is to scan through the frame with multiple aspect ratio windows. The various object detection models use different strategies to reduce the scanning time.

Feature Extraction: While scanning the model should detect the object in a particular scanning window. To detect we need to extract the visual features of the region and compare with the actual object classes.

CNN is another type of feature extractor which solves most of the issues with manual design, as it is a neural network which can be trained on the domain of objects with various training examples to generalize the extractor to extract the features from a wide range of different images. The main issue with CNN is the computational complexity. When the network becomes deeper and deeper the number of floating-point operations will proportionally increase.

Classification: The detected object should be classified between different classes. Usually Support Vector Machine, AdaBoost and Deformable Part-based Models are used for classification task.

The time consuming and complex part of the model is Feature extraction. Also, the accuracy of the model also depends on the same layer. CNN is widely used for this purpose nowadays in most popular Object detection models.

Different object detection architectures apply CNN in classification and localization in different methods. But the core component is still CNN and which decides the performance of the particular architecture.

2.2. Convolutional Neural Networks

Most of the previous research work done on object detection to measure the accuracy and performance is General purpose Computers. There is Not enough research work done with various underlying CNN architectures specifically for CPU based mobile phones. Also, there is no study on detection models for applications where the real-time performance is crucial than the accuracy. In this research the study is focused on Object detection models for real time applications on CPU based mobile phones.

CNN are fundamental building blocks for computer vision applications in Artificial neural networks. There are two main advantages of using CNN Instead of using a fully connected network for computer vision tasks. CNN's use a small number of parameters to learn compared to fully connected networks because of convolutional operations, and they are immune to translation variances in the input images.

By combining multiple layers of CNN blocks with other layers such as Activation, fully connected and Regularization there are many architectures proposed. Early implementations such as LeNet-5[5] used to classify handwritten digits use few layers of CNN. By improving the LeNet-5 model AlexNet[6] was introduced with deep layer architecture. As they shown the deeper the layer is the accuracy increased. AlexNet has around 61M parameters which takes around 249MB memory and do 1.5B floating point operations per one image classification. GoogleNet[7], VGGNet[8] are some evolutions of CNN with each introduce some new features. The general trend here is to use deeper layers to make the model learn more specific features and as a result they give more accurate results.

Instead of adding deep layers, ResNet[9] use residual blocks where they show that making shortcut connections to the deeper layers, it is easy to train the network and solves issues in vanishing gradient. DenseNet[10] proposed as an improvement to ResNet with dense shortcut connections to the forward layers. The residual blocks add more addition operations instead of multiplication operations the computation cost is not increase as it is happened in adding deep layers of network.

As the above proposed architectures mainly concern on accuracy of the model, they didn't cover the prediction time and computational complexity of the total network. But in mobile devices as they have limited CPU and real-time applications require low processing time

there were alternative approaches proposed. Usually the models are trained on powerful computers and then the learnt models deployed in mobile devices and used by applications. To improve performance on mobile devices, it is important to consider the test-phase efficiency rather than train-phase efficiency.

Flattened CNN[11] take an approach to reduce the redundant filters in the CNN to reduce the computational cost. Here they replace the 3-dimensional convolution operation with series of 1-dimensional convolution layers to reduce the number of matrix multiplication operations. Quantized CNN models accelerate and compress the network parameters by which they achieve better performance in test phase computation.

SqueezeNet[12] combines three different approaches to reduce the number of parameters while maintaining the accuracy. First, they replace the 3x3 filters with 1x1 filters which effectively reduce the number of parameters by 9 times. Then they decrease the number of input channels to the convolutional layer through another layer they called squeeze layer. When the above two strategies try to reduce the parameters, they use delayed downsampling to preserve the accuracy. Usually in CNN the layers are down sampled by using strides (>1) in combination with pooling layers. But SqueezeNet paper proposes not to down sample in the first layers, instead pass the information up to the deep layers and start down sampling in the end to preserve the accuracy of the network. Through these techniques the SqueezeNet model was able to be compressed below 0.5Mb and they were able to implement it on FPGA modules.

XNOR[13] network uses binary numbers instead of floating-point numbers in the convolution layer operations. When all the operands are in binary the convolutions can be estimated by XNOR and bitcounting operations. XNOR Nets can offer accuracy close to CNN with approximately 58 times speedup. Another advantage is the operations can be efficiently performed on CPU s which is an important factor on Mobile devices as those doesn’t have powerful GPU s.

MobileNet[14] is another CNN architecture focused on efficient models on Mobile and embedded devices. This model is based on streamlined architecture and makes use of Depthwise separable convolutions. Depthwise separable convolution is a factorized version of standard convolution layer into a depth-wise convolution and a point-wise convolution. The main feature of this network is it is easy to tune the network according to the requirement to match Accuracy-Speed trade-off through the hyper-parameters provided by the model. One hyper-parameter called the width multiplier, which thin the network uniformly throughout the layers which effectively control the computation cost and accuracy. When applying this parameter at each layer the number of input channels and output channels will reduce by times. The other parameter called the resolution multiplier, will reduce the width and height of the filters and input by a factor. By controlling both parameters, it is easy to create and compare the network performances and choose the best values according to the various requirements.

The existing research work done on object detection model performance comparison are based on different object detection pipeline architectures or an improvement on the existing

pipeline. The research papers on different CNN architectures there are comparison on performance, it is clearly showing the selection of CNN models can greatly impact the Image processing performance. But There is no considerable performance evaluation work done on effect of changing the base CNN layers in the CNN based object detection models and also the performance of particular models on different hardware platforms.

For example the YOLO v3 model directly uses a CNN back-end which can be replaced with different CNN architectures such as MobileNet, SqueezeNet without modifying the functionality. By choosing appropriate back-end models and optimizing their parameters will improve the performance in low end CPU platforms.

2.3. Object detection architectures

Earlier models use brute force method to slide through the image and check with a CNN classifier weather it contains any required images in that particular region. Then R-CNN models were introduced (Region proposal CNN) which instead of sliding through the entire image, the CNN layers were applied only for the proposed regions. The region proposals are done through some selective search algorithms.

Based on the number of stages of processing of images the object detection models can be categorized into two.

a) Two stage detection

Here the image is given as input to the network. A part of the model will generate the region of interests (Possible rectangle regions of objects). Another component will process the image through CNN and detect the objects. The results will be combined and returned as output.

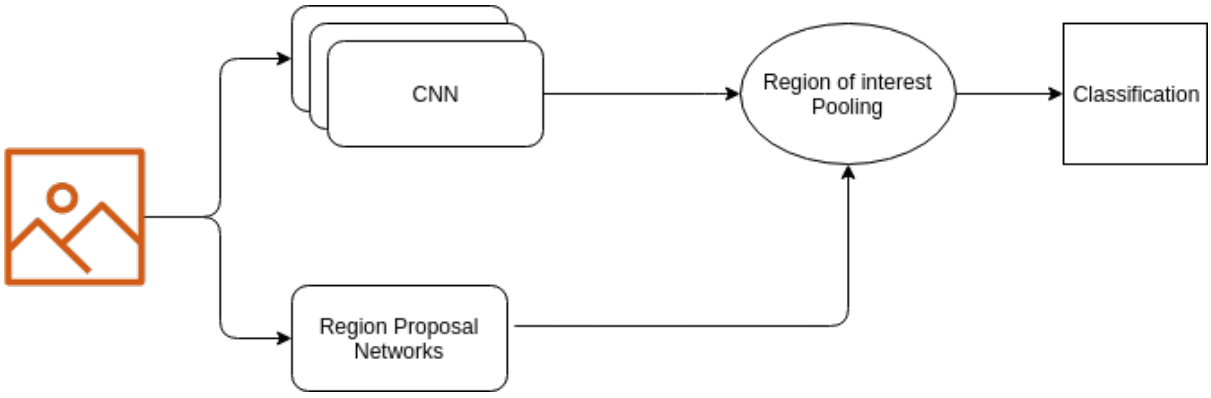

Figure 2: Two stage object detection architecture

RCNN[15], Fast RCNN[16], Faster RCNN[17] are examples of two stage object detectors. RCNN based network proposed to use a selective search to extract only a certain number of regions from the image instead of sliding through the image and checking all frames.

Selective search can be implemented through various ways depending on the application. In the paper they suggested 2000 region proposals. Here the issue with the region proposal is that it is very slow and also it is not using the image data set to learn the image features and propose the regions.

As an improvement to R-CNN another network called Fast R-CNN introduced, where the region proposals are made on extracted features using RoI pooling layer. At the end the SoftMax layer will identify the object and generate the bounding box. With this technique Fast R-CNN becomes significantly faster than the RCNN model.

Faster R-CNN models remove the usage of selective search for region proposals and only use the single CNN network to generate both Regions and classifications.

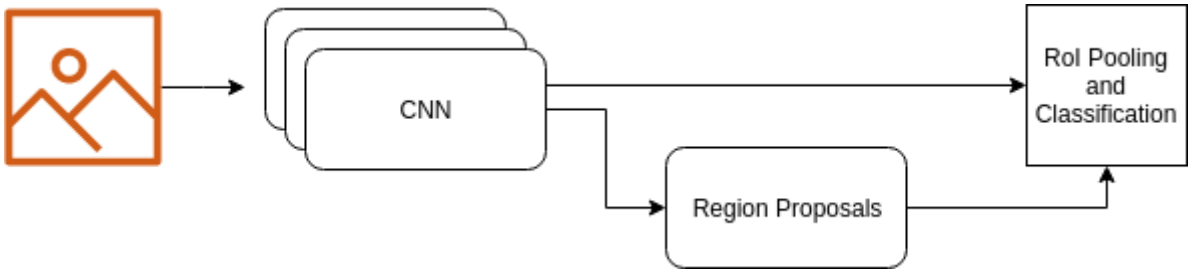

Figure 3: Faster R-CNN Object detection model

b) Single stage detection

Single stage detectors generate Region of objects and features through the same CNN layers in the same component.

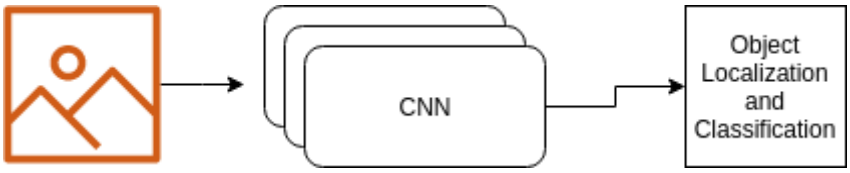

Figure 4: Single stage object detector architecture

SSD uses a single pipeline of Convolutional layers combined as a series with various size feature maps in various layers. At different stages of the layers different aspect ratio object bounding boxes will be predicted and passed to the output.[18]

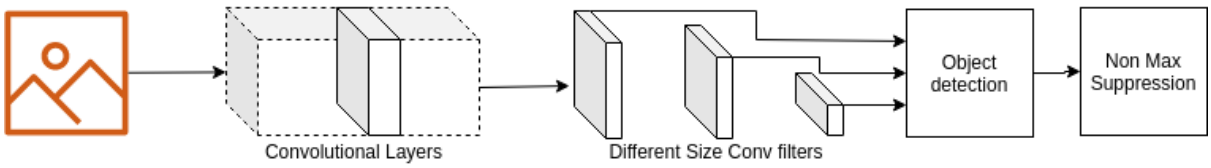

Figure 5: Single Shot Multi-box Detector

YOLO is another single stage object detection architecture. The difference between SSD is, that instead of getting the outputs from different layers of CNN, YOLO get the different size bounding box with different aspect ratios from the final layer of the CNN network.[19]

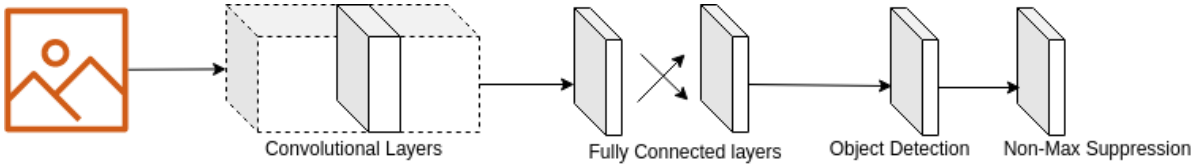

Figure 6: YOLO detector

Instead of a Separate algorithm to propose the region of the network as in R-CNN models, here YOLO divides the image into regions by a grid, and then the image directly passes through the CNN blocks. At the last layer in addition to the Class the Region also provided as the target to learn. Because of a single pass through the convolutional layer it doesn't take much time to localize and detect the object. The performance is close to Real-time and acceptable accuracy. There are improvements on YOLO proposed. Mainly by changing the underlying CNN architecture it directly affects the accuracy and performance of the entire model. For the implementation YOLO uses a CNN model called Darknet-53Figure 7 which consists of a series of Convolution layers and Residual connections and at the end it has an Average Pooling layer.

YOLO v2 introduced new techniques within the model such as Batch Normalization, High Resolution Classifier, Anchor boxes, Dimension Priors, Location Predictions to improve the accuracy while keeping the performance without lacking.[20] YOLO v3 improved on v2 model by modifying the feature extractor to darknet-53 CNN model. [1]

In the paper, different frame sizes including 256 x 256, 320 x 320 and 416 x 416 pixel images are used as input for training and validation. For the COCO dataset the output consists of 80 different classes and for each class 3 bounding boxes with different scale were predicted by the model.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 7: Darknet53 Architecture

When comparing the Two stage detectors and Single stage detectors[1], even though RetinaNet performs with high accuracy in COCO data-set, it is lacking in object detection speed. It takes 3.8 times more seconds than YOLO v3. When comparing to Two stage

detectors (Faster R-CNN) One stage detectors have less accuracy but higher performance, because the number of computation is significantly lower.

c) YOLO Lite

YOLO-lite[21] is an approach to implement an efficient Object detection model for non-GPU based hardware. The research was mainly done to demonstrate that

- The shallow networks capability of object detection on non-GPU based platforms
- Batch normalization is not necessary for shallow networks

The research work done on YOLO lite to test the real-time performance are,

1. Image Size: Reducing the input size will have an effect on all the layers. The changes will have an exponential effect on overall performance. But the accuracy will reduce as some of the data is lost and not available for the network.
2. Batch Normalization: Batch normalization is taking the output of the previous layer and normalize it to mean zero variance and standard deviation before applying to the next layer on each mini batch. As each mini batch have different mean and variance of ground truth distribution, it is hard for the layers to learn the features (Covariate shift). Batch normalization helps to stabilize the network by keeping the input distributions in same scale to learn the features quicker while training. It is shown that Batch normalization has improvements in larger networks on training accuracy.[20]

   Smaller networks like YOLO-Lite doesn't have much issues with covariate shift it doesn't require Batch normalization in all the layers. Also it is shown that feedforward process slows down by the batch normalization calculation significantly compare to actual neural net calculations.
3. Pruning: Pruning is removing certain weights based on the contribution of that node to the entire network. It is showing that the number of parameters reduced by 9x on Alexnet and 13x on VGGnet.[22]

Layer	Filters	Size	Stride
C1	16	3 × 3	1
MP		2 × 2	2
C2	32	3 × 3	1
MP		2 × 2	2
C3	64	3 × 3	1
MP		2 × 2	2
C4	128	3 × 3	1
MP		2 × 2	2
C5	128	3 × 3	1
MP		2 × 2	2
C6	256	3 × 3	1
C7	125	1 × 1	1
Region			

Figure 8: YOLO-Lite architecture

As shown in Figure 8 the YOLO – Lite has simple architecture compared to YOLO by removing several complex layers and batch normalization layers.

When YOLO-Lite was compared with other object detection networks, it had around 21 FPS which is enough for a real-time application even though the accuracy is reduced into 12.26%.

Model	mAP	FPS
Tiny-YOLOV2	**23.7%**	2.4
SSD Mobilenet V1	21%	5.8
YOLO-LITE	12.26%	**21**

Table 1: YOLO-Lite Performance comparison

Chapter 3: Methodology

3.1. Introduction

The objective of the research work is to find an efficient object detection artificial intelligent model which is most suitable for real-time applications in low end CPU based devices. The object detection model should be capable of performing detection much faster than the existing models without sacrificing accuracy on the same hardware platform.

As long as the existing models and optimized are evaluated on the same hardware (CPU based) it is safe to assume the model will behave in a similar way in other CPU based devices aswell. Even though the performance of the model may depend on CPU architecture such as Instruction set, number of cores, number of threads used by the model, it doesn't affect much on the results as all the models use a single thread and are compiled with the same libraries.

The object detection models which use hard coded search algorithms can be ignored as the objective is to use the model for more generalized applications, and the hard coded models are not robust enough to support a wide range of applications. It will narrow down the available object detection models for evaluation to few Two stage and Single stage detection models.

3.2. Research Questions

The main question analyzed in the research is, through applying various optimization techniques on existing object detection model is it possible to maintain the accuracy without reducing much while improving the performance on Mobile CPUs. If it can be improved, to what extend it can be improved and weather it can be applied on real-time applications on mobile phones. To answer the question through literature review the appropriate model and data set will be collected and modified in evaluation.

3.3. Object detection model

Based on the analysis on literature review section YOLO based model is most suitable as a starting point for the current objective because,

a) Architecture

As shown in Figure 6 the official implementation of YOLO consists of Darknet based CNN models. The input is an image / frame from a video and the output will be bounding boxes of object detection and the classification of the object within the area.

The overall network is simple and modular, as it contains only CNN layers and Fully connected layers. Figure 7 shows the darknet-53 model used in official YOLO V3[1] implementation which consists of a combination of CNN and Residual Connections. It is possible to replace with various CNN models and evaluate the performance. Another single

stage detector SSD contains similar structure but the output depends on several layers of CNN, which makes it difficult to try with various CNN networks and evaluate.

b) Performance

As compared in the YOLO V3 [1, Fig. 3] the accuracy of YOLO v3 is quiet similar to other object detection models and at the same time the performance much higher than others. Which is a suitable characteristic for the objective of the research.

3.4. Backbone CNN

Choosing an appropriate CNN model for the optimization can be done on comparing various characteristics of the CNN networks. The main objective of the research is to get an optimized network for real-time applications. The main performance bottleneck of a CNN model is the number of parameters in the layers, which is directly proportional to the number of floating point operations on calculating matrix multiplications. By choosing an appropriate CNN architecture it is possible to optimize the overall performance of the entire object detection model.

There are multiple parameters which can be changed and optimized to support the objective of the research.

Number of layers

The depth of the layers has an effect on accuracy and performance, it is important to choose the right number of layers to balance the accuracy performance trade-off. Usually the more layers in the network, it can more generalize the images trained, but it will increase the number of parameters and require more training data.

Residual Connections

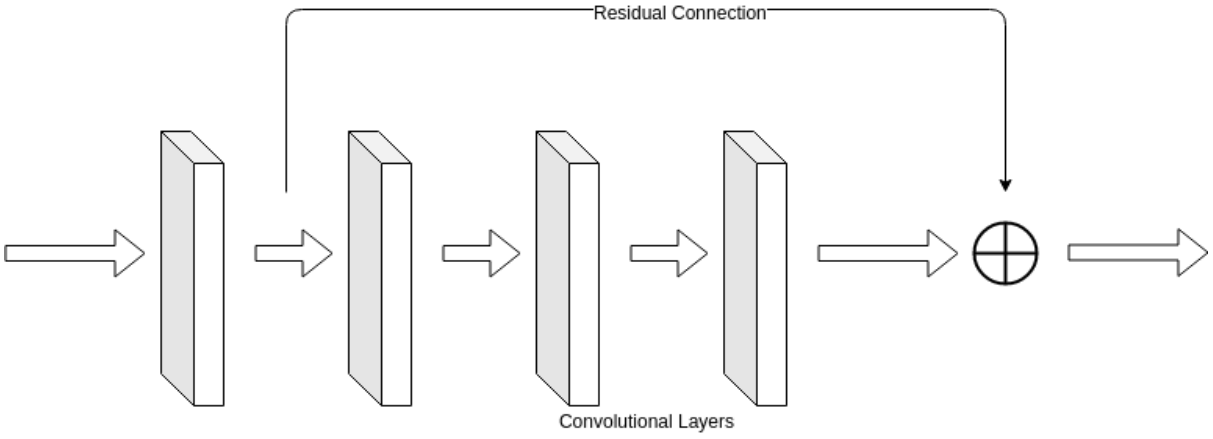

Figure 9: Residual Block in CNN

Residual connections / Bypass connections between CNN layers help them to learn the features in the image without adding additional computational parameters. ResNet is a model which uses Residual connections in the model to learn the features in training data. MobileNet is another model which makes use of Residual connections with other optimizations.

a) Optimized Model

YOLO lite[21] provides an approach to reduce the number of layers in the YOLO model to support non-GPU based Neural Networks. Similar approach can be used to improve the YOLO model to optimize the network for the Mobile phone CPUs as the main bottleneck of performance depends on the number of calculations. As real-time applications can trade-off the accuracy for the inference time of the model, because of the availability of huge input data, it is possible to shrink the model by removing the layers and introducing the residual connections to improve the train speed.

b) Tensorflow Lite Optimizations

Tensorflow lite libraries provide certain optimizations on the trained models to support embedded devices and mobile devices. It allows us to compress the model to a smaller size to improve the performance without much effect on accuracy.

Quantization is another technique used in Tensorflow lite to improve the performance on low end CPUs. It is done through reducing the precision of floating point parameters and operations within the model. FLOPS will be increased by a considerable amount as the calculations can be done in fewer cycles than directly using the lengthy floating point parameters.

3.5. Data-sets

For the training of the models COCO data-set[2] will be used. The weights of the base YOLO model is already available and the optimized model will be trained on the data-set to learn the weights. COCO is a standard data-set containing 350,000 images with labels and bounding boxes to support object detection with 80 different commonly used types of objects.

Evaluation will be done on Pascal VOC 2012 data-set[3] as the COCO data-set was already used to train the networks, it will not give the accurate results on evaluation on the same data-set. Pascal VOC 2012 data-set contains 20 classes and around 11000 images with object detection annotations.

To evaluate the application another data-set of images will be collected which include street signs and vehicles. Data-set collection done through recording from the dash-cam and get images on Google Street View.

There are mainly two different tools used for data-set preparation.

- Computer Vision Annotation Tool (CVAT)

This tool is best for annotating the video files. It will read the video data and generate the frames and allow to annotate two frames and interpolate the annotations on the frames between them. Usually the objects in frames are not changing quickly if the frames are close enough. It allows to generate a large number of annotated data-set without much effort.

Also the tool support exporting the annotations in VOC standard format which can be directly used for training the model in Tensorflow.

- Visual Object Tagging Tool (VOTT)

  VOTT is useful when annotating the individual images. It support copy one annotation to another and lock the annotation so it is easy to annotate class by class without drawing bounding boxes each time. VOTT also supports VOC format annotation export which can be directly used for training.

After exporting the annotations and images in the VOC format it is required to split the data set for training and validation. During the training the data-set divided into 80% training and 20% validation images.

3.6. Model Preparation

YOLO V3 model implementation of Tensorflow V2 was used as a Base model. The pre-trained weights already available with the DarkNet paper. Optimized YOLO V3 model is implemented by removing the Convolutional blocks by reducing the number of layers similar to Mobile-net.

To train the model the VOC format dataset needs to be converted to Tensorflow records (Protobuf format). Dataset is shuffled and divided into train and validation set and fed into the model for training. Tensorboard is used for visualizing the training progress as it can provide the error/accuracy based on the training set and validation set on each Epoch of training.

While training the model weights are saved into checkpoints so the training can be continued into several iterations without starting from the beginning.

3.7. Training

For the evaluation the per-trained base model (COCO data-set) was trained on VOC data-set and the Optimized model also trained on VOC data-set. VOC data-set divided into three sets Training, Validation, Testing. Training and Validation image sets were used for training the model and Test data-set used in the evaluation of the model. Test set only contains 100 images and the rest of the images are divided into 80% and 20% for training.

When pre-trained weights are used, the accuracy will start from a higher value than starting from a random weights. Also the models were optimized quickly compared to a fresh-model.

To avoid overfitting training of the model stopped when validation accuracy start to reduce while training accuracy continue to reduce.

Adam optimizer[23] was used for training which has certain advantages than other optimizers. It is a combination of two optimizers

- Adaptive Gradient Algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)

Adam optimizer make use of the second moments of gradients than the first moment of gradient (Mean) to regulate the learning rate to speed up the optimization.

Optimized models were saved with weights for evaluating the models based on test data-set. Then the trained models were converted into Tensorflow-lite format by Tensorflow converter tool. This tool will optimize the model by improving the node graphs and changing the way floating point operations are done. The output is a tflite file which can be used on Mobile platforms with Tensorflow Lite libraries.

a) Hardware and Tools

For training the dataset it is required to have high performance CPU and GPU. A PC with Intel Core i7-6700 CPU and NVIDIA GeForce RTX 2070 Graphics Card will be used for Training the data-set. Tensorflow 2.1 with GPU Support will be used to create the models. Tensorflow lite will be used to optimize the models for Mobile phones.

3.8. Evaluation

a) Parameters

Model Size and Parameters: The number of operations is directly proportional to the number of parameters in the network. Direct comparison of parameter sizes will give overall computational cost metric.

Accuracy: After creating the CNN model run random test samples through various architectures (MobileNet, SqueezeNet, etc) and compare with the results with the proposed architecture. Mean Average Precision (mAP) is one of the metrics which measures the accuracy of an object detection model. It can measure the accuracy of detected object location in with the labeled data-set.

Performance: With the same data-set by providing a series of images and log the average time taken can be a measure of speed of computation of the network. It can be done on various hardware platforms such as CPU, GPU, Mobile CPU and will be used to compare how the parameter changes depending on hardware on various models. The output of the evaluation can be ultimately expressed number of frames/images per second, which gives an idea on how the model will perform on video streams on mobile applications.

b) Approach

Evaluating the level of optimization will be done by experiments on the optimized model and compare the results with existing models. The evaluation of optimized neural network model for object detection will be done on three levels.

1 Compare the object detection performance of Optimized model with the Base model on a PC CPU.

2 Compare the object detection inference time of Optimized model and Base model on a PC CPU after Tensorflow-lite graph optimizations.

3 Compare the object detection inference time of Optimized model and Base model on a Mobile CPU after Tensorflow-lite graph optimizations.

The performance of the object detection model depends on two factors

- Inference time: How fast an image can be evaluated by the detection model.

  Calculating the inference time can be done by applying several standard images with different image sizes through Object detection models and calculate the time it take to evaluate the image.

  Log the time between start and end of each evaluation and get an average time.

- Accuracy: How accurate the object detection model detect the objects in the image.

  Mean Average Precision (mAP score) is the standard way to evaluate the accuracy of object detection models.

  mAP score will depend on two factors

  - Detecting the existence of the object (Classification)
  - Determine the location of the object (Localization)

Usually the accuracy doesn't depend on the CPU's as all the CPU's perform the multiplications in the same way. So it is not needed to evaluate the accuracy changes in the third step as it doesn't provide any additional measures.

Even though Tensorflow-lite graph optimizations doesn't change the model parameters but it change the precision of calculations, it also has some effect on accuracy of the results. The changes on optimizations can be measured by comparing the results of Step 1 and Step 2.

The base model performance provide a ground truth values and the optimized model performance can be used to calculate the percentage of improvement of optimization of the model to be used on a less powerful CPU. In this research the main concern is inference time of the model as it is the main factor to decide on real time performance. Without reducing much accuracy if the model can evaluate the image in less inference time, it will indicate the optimization of real-time application usage of the model.

Chapter 4: Results

4.1. Introduction

As described in the evaluation plan, the models deployed on relevant hardware and tested with datasets. Tests done on hardware while other processes stopped as much as possible to reduce interference on results. Experiments done multiple times to make sure the results are not varying largely. Inference time is directly logged and the accuracy is calculated using the standard mAP equation.

4.2. Experiment Results

a) PC CPU

PC with Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz was used for evaluating the performance on a PC environment.

The Base model and Optimized model (before and after converting to lite format) were tested on PC CPU with the VOC test data. Test consist 100 image samples, with 20 classes.

Model	**mAP score**
Base model	56.1
Optimized model	32.4

Table 2: Accuracy - PC CPU

Model	**Inference Time**
Base model	877 ms
Optimized model	202 ms

Table 3: Inference Time - PC CPU

b) Mobile CPU

Huawei Y9 mobile with Octa-core CPU (4x2.2 GHz Cortex-A73 & 4x1.7 GHz Cortex-A53) was used for evaluating the results.

Both base and optimized model were tested on Mobile phone, by uploading sample data-set to the mobile phone.

Model	**Inference Time**

Base model TF-Lite format	5611 ms
Optimized model TF-Lite format	751 ms

Table 4: Inference Time - Mobile CPU

4.3. Findings

From the results, When comparing the performance of the optimized model with base model, It shows around 4 times improvement in inference time with a reduction of 23.7 mAP score. When testing on Mobile phone the inference time improved by 7.47 times. The optimized model doesn’t show any difference on mAP score compare to PC evaluation as expected.

Chapter 5: Discussion

5.1. Introduction

With the optimized model it is possible to achieve around 1.33 FPS detection rate with 416 x 416 size frames from the camera video-stream. Experiment results show that the optimized model is able to achieve a real-time performance for mobile applications which require object detection where the frame rate is critical and the reduced accuracy is tolerated.

5.2. Future Research

In this research the scope is limited only to YOLO which is one of the Single Stage Detector model. As SSD is another Single Stage model the optimization technique can be applied there aswell and the results can be compared.

As current results are obtained on Mobile phone CPU, it can be extended by modifying the android to use Mobile phone GPU for inference to evaluate the performance.

5.3. Conclusion

As the results shows the significant improvement on the object detection performance can be used in various real-time applications. Without many large scale changes in the original object detection model parameters, it is possible to optimize for Mobile phones

References

[1] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *ArXiv180402767 Cs*, Apr. 2018, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1804.02767.

[2] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” *ArXiv14050312 Cs*, Feb. 2015, Accessed: Jan. 29, 2020. [Online]. Available: http://arxiv.org/abs/1405.0312.

[3] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, pp. 303–338, Jun. 2010, doi: 10.1007/s11263-009-0275-4.

[4] X. Wu, D. Sahoo, and S. C. H. Hoi, “Recent Advances in Deep Learning for Object Detection,” *ArXiv190803673 Cs*, Aug. 2019, Accessed: Sep. 20, 2019. [Online]. Available: http://arxiv.org/abs/1908.03673.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[6] “ImageNet Classification with Deep Convolutional Neural Networks | Alex Krizhevsky,” *ResearchGate*. https://www.researchgate.net/publication/267960550_ImageNet_Classification_with_Deep_Convolutional_Neural_Networks (accessed Jul. 27, 2019).

[7] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *ArXiv14094842 Cs*, Sep. 2014, Accessed: Jul. 15, 2019. [Online]. Available: http://arxiv.org/abs/1409.4842.

[8] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *ArXiv14091556 Cs*, Sep. 2014, Accessed: Jul. 27, 2019. [Online]. Available: http://arxiv.org/abs/1409.1556.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[10] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *ArXiv160806993 Cs*, Aug. 2016, Accessed: Jul. 14, 2019. [Online]. Available: http://arxiv.org/abs/1608.06993.

[11] J. Jin, A. Dundar, and E. Culurciello, “Flattened Convolutional Neural Networks for Feedforward Acceleration,” *ArXiv14125474 Cs*, Dec. 2014, Accessed: Jul. 15, 2019. [Online]. Available: http://arxiv.org/abs/1412.5474.

[12] 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 <0.5MB model size,” *ArXiv160207360 Cs*, Feb. 2016, Accessed: Jul. 15, 2019. [Online]. Available: http://arxiv.org/abs/1602.07360.

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,” *ArXiv160305279 Cs*, Mar. 2016, Accessed: Jul. 15, 2019. [Online]. Available: http://arxiv.org/abs/1603.05279.

[14] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *ArXiv170404861 Cs*, Apr. 2017, Accessed: Jul. 14, 2019. [Online]. Available: http://arxiv.org/abs/1704.04861.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *ArXiv13112524 Cs*, Nov. 2013, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1311.2524.

[16] R. Girshick, “Fast R-CNN,” *ArXiv150408083 Cs*, Apr. 2015, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1504.08083.

[17] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *ArXiv150601497 Cs*, Jun. 2015, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1506.01497.
[18] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *ArXiv151202325 Cs*, vol. 9905, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *ArXiv150602640 Cs*, Jun. 2015, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1506.02640.
[20] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” *ArXiv161208242 Cs*, Dec. 2016, Accessed: Jul. 16, 2019. [Online]. Available: http://arxiv.org/abs/1612.08242.
[21] J. Pedoeem and R. Huang, “YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers,” *ArXiv181105588 Cs*, Nov. 2018, Accessed: May 27, 2020. [Online]. Available: http://arxiv.org/abs/1811.05588.
[22] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *ArXiv151000149 Cs*, Oct. 2015, Accessed: Jul. 22, 2019. [Online]. Available: http://arxiv.org/abs/1510.00149.
[23] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ArXiv14126980 Cs*, Jan. 2017, Accessed: May 27, 2020. [Online]. Available: http://arxiv.org/abs/1412.6980.