Optimization of the Real-Time State-of-the-Art YOLOv4 Object Detector By Modified Neck Structure

Bibarys Mussagaliyev, B.Eng

Submitted in fulfilment of the requirements
for the degree of Master of Science
in Electrical and Computer Engineering

School of Engineering and Digital Sciences
Department of Electrical and Computer Engineering
Nazarbayev University

53 Kabanbay Batyr Avenue,
Nur-Sultan, Kazakhstan, 010000

Lead Supervisor: Ablaikhan Akhazhanov
Co-supervisor: Almas Shintemirov

May 2022

Declaration

I hereby, declare that this manuscript, entitled "Optimization of the Real-Time State-of-the-Art YOLOv4 Object Detector By Modified Neck Structure", is the result of my own work except for quotations and citations which have been duly acknowledged. I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

Signature: BM

Name: Bibarys Mussagaliyev

Date: 24.04.2022

Abstract

The state-of-the-art YOLOv4 object detector has already demonstrated its effective inference (65 frames per second (FPS) on V100 Tesla) and relatively high accuracy on MSCOCO dataset (mAP 43.5 %) in real-time mode. Moreover, simplicity of the model's training and testing appears as another advantage for machine learning community. The ability of the model to be learned as a unified system on just a single graphic processing unit (GPU) unsurprisingly established itself as the milestone in the real-time object detection field. This work aims to review the fundamental and most recent academic work in the field and suggest the incremental research towards the optimization of the YOLOv4 architecture. We propose a model, named SAMD-YOLOv4, with modified neck structure, which reduces number of learning parameters by decreased number of filters with 1×1 kernel, which is followed by spatial attention module and dilated convolutional layers. We demonstrate that method is capable to reduce model's complexity by 7.3% with no effect on model's precision as well as lowered inference time by 6.9%. In Chapters below, we provide experimental results and comparison study on baseline YOLOv4 and our SAMD-YOLOv4. Furthermore, the TensorRT-based inference's results will be revealed and studied.

Acknowledgements

With this thesis work, I would like to thank my family and my wife for supporting me during these years of studying at NU. I am thankful for their love and support, which made me who I am now. In the end, I want to thank myself for all the hard work and discipline I took so far.

Table of Content

List of Abbreviations and Symbols

Bi-FPN	Bi-directional Feature Pyramid Network
BN	Batch Normalization
CAM	Channel Attention Module
CBAM	Convolutional Block Attention Module
CNN	Convolutional Neural Network
CSP	Cross Stage Partial (Network)
DL	Deep Learning
EMA	Exponential Moving Average
FPN	Feature Pyramid Network
FPS	Frames Per Second
GPU	Graphic Processing Unit
ML	Machine Learning
MSCOCO	Microsoft Common Context Objects
NAS-FPN	Network Architecture Search Feature Pyramid Network
PANet	Path Aggregation Network
R-CNN	Region-proposal Convolutional Neural Network
ResNet	Residual Network
SAM	Spatial Attention Module
SPP	Spatial Pyramid Pooling
SSD	Single Shot Detector
SOTA	State-of-the-Art
VGG	Visual Geometry Group
YOLO	You Only Look Once

List of Tables

List of Figures

Chapter 1 - Introduction

The deep learning (DL) is widely exploited in the field of computer vision providing different solutions for the complex problems such as image classification, segmentation, detection, and many others. One of the popular and highly effective technology of DL is the convolutional neural networks (CNNs), these networks substantially enhance the prediction performance using the big data sets and highly efficient GPU-based computational resources and have brought the breakthrough in this field. The problems that were considered impossible to be resolved, now are easily solved with a very high performance and efficiency. For instance, considering the classical image classification problem contest, where the famous AlexNet model by [1] dominated and set a record (top-1 of 37.5% and top-5 17.0% error rates) in recognition task ImageNet LSVRC-2010 in 2012 [1]. The AlexNet is a solution purely made by DL architecture, which outperformed the traditional ML models in various evaluation metrics. Far from the image classification models, the object detection algorithms are also valuable these days, due to various reasons, starting from high demand in the industry and academic research and ending with proportional progress in hardware technologies [2]. One such model is YOLOv4, which is taken as the target model of this work. YOLOv4 is the one-stage object detection algorithm which set new standards of object detection problem as a whole. The original authors of YOLO [3] started a massive popularization of this domain by introducing this super fast and accurate CNN model, which has already got a great deal of modifications and researches. Therefore, the main motivation of this thesis is to research and optimize the fastest real-time object detector by eliminating the redundant learnable parameters within the neck of the model.

This work will review recent literature and ideas on relevant academic papers related to the thesis. In Chapter 2, we will present the fundamental DL techniques, which are irreplaceable in the modern SOTA CNN models. As well as, we cover relevant CNN architectures for feature extraction, their main benefits and limitations. Moreover, the advanced path aggregation and information propagation techniques and architectures for object detection and semantic segmentation models will be reviewed. Then, Chapter 3 aims to review the YOLOv4 end-to-end architecture, its backbone, neck and head parts. Chapter 4 will cover information about the proposed design, neck structure design and others. Next, the experimental results, comparison studies and TensorRT implementation will be presented in Chapter 5. Finally, Chapter 6 will compose the discussions of the work, its core ideas, motivation, trials and errors, and proposed method's limitations. In the end, Chapter 7 would conclude the entire work done.

Chapter 2 - Background

Firstly, the object detection problem in fact is the extension of the image classification problem. The former requires the predictor and other modules coming after the main feature extraction module called backbone. However, before diving into the details, it is worth to mention that there are two types of object detection algorithms, one-stage, and two-stage detectors. Prominent one-stage detectors models: You Only Look Once (YOLO) family [3], Single Shot Detector (SSD) [4], SqueezeDet [5], DenseNet [6], two-stage detectors models: Region-proposal Convolutional Neural Network (R-CNN), Fast-RCNN [7], Faster R-CNN [8] and others. As it was discussed in [9], the difference between one-stage and two-stage detectors lies in the additional network module for sparse prediction. Everything before this module is identical in both detector types. The object detector consists of four modules: backbone, neck, dense prediction module and sparse prediction module. Starting with first, the backbone is the main body of any object detection algorithm, its main task is to extract features from the data input. CNNs form the backbones, they can be different in size, depth, architecture. Some of the popular are: AlexNet [1], Visual Geometry Group-16 (VGG16), VGG18, VGG19 [10], Darknet53 [9], EfficientNet [11] and many others. The main function of these CNNs, is the extraction of the most important features from the image data. Starting from the low-level features, such as vertical and horizontal lines to more advanced and complex shapes like squares, circles, and ending with high-level features such as scenes, faces, animals and other sophisticated image patterns [1].

These CNNs are mainly designed to resolve image classification problems on famous publicly available datasets such as ImageNet, Pascal-VOC, CIFAR and many others [12], [13], [14]. The optimal designs achieved in this classification competition, then are exploited in more advanced computer vision tasks, mainly object detection and semantic segmentation. The object detection problem is a classification with object localization, whilst segmentation is about classifying and finding objects with pixel-wise precision, namely careful segmentation of object's borders. Indeed, there are lots of other DL-based computer vision problems which are emerging every day due to the high popularity of DL models and proportional advancement in hardware technology. For brevity, we will not cover them. Thus, the success of any CNN-based deep learning model is directly related to the performance of the feature extractor or backbone. There are already tones of research that was done in this scope, featuring the technologies, concepts, network architectures, and modules which have become irreplaceable parts of any CNN model. In the following paragraphs below, we will cover fundamental deep learning regularization techniques, essential CNN architectures and more advanced object detection-oriented techniques of information propagation within the network.

2.1 Regularization Techniques

In this section, we will introduce the fundamental deep learning techniques that aim to suppress the over-fitting of the models. These techniques are universal and applicable in various deep learning fields such as computer vision, optimizations, natural language processing and many others.

2.1.1 Dropout Technique

The dropout layers are responsible for intentional parameters cancellation with particular probability for better generalization [15], [16] as well as DropBlock, which works analogously for the 2D data such as feature maps in CNNs [17]. See Figure 1. The feature map on the left represents dropout layer, which cancels randomly chosen elements in the map, whilst on the right, dropblock module drops out spatially related elements in the feature map. The latter is more preferred in case of computre vision tasks.

The dropout based models constantly change by dropping out several nodes during the training process. Such a technique eliminates the so-called layers' co-adaptation to specific errors and patterns in data, it prevents model to memorize the vector space of data distribution [16]. The dropout layers can be easily adapted to any network architecture, before or after any layer. With a given probability value, specific nodes of layers are dropped out from the training process. Thus, by using the dropout layers in the network model we can achieve extremely higher performance [15]. This will generally make it possible to build and design neural network models that perform on validation data set better than on training set, thus the models become more robust [17].

For instance, the application of dropout layers in standard neural networks show substantial improvements, [23] implemented several models with and without dropout layers. The models consist of 3 fully connected layers with 100 nodes per layer. The results at 1500 epochs, with dropout number of errors around 100 per epoch, whilst without dropout layer, number of errors per epoch is more than 140. The training was done with CIFAR-10 dataset [16]. In can be concluded that, the dropout technique appears to be fundamental tool for any modern deep learning model. It makes the models to be robust, that have low generalization error. Similarly, the DropBlock is exploited by CNNs, making the spatial elimination of several elements in the feature map. The idea behind is similar as in dropout layer [17].

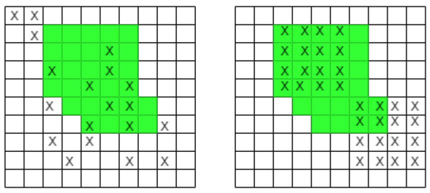


Figure 1: Dropout (left) and Dropblock (right) modules.

2.1.2 Batch Normalization

Next technique that is worth to be mentioned is the batch normalization (BN). The training process of neural networks is very complicated process that requires proportionally sophisticated training methods to conduct proper data fitting [18]. Such a challenge is concerned with network's deepness, the backpropagation updates the weights with gradients layer by layer, this process goes from the output to the input layers [18]. In fact, layer's parameters should be updated layer by layer, not all at the same time, but in practice it is exactly what happens. Since, all the layers' parameters are updated at the same time, the optimizer always searches for the new optimum and that is the problem [19]. In details, the distribution of each layer's input constantly changes as the parameters of previous layer change. To preserve this issue, it is required to use small learning rates, accurate initialization of parameters [19]. This phenomenon is called the covariance shift.

The batch normalization technique scales the output of each layer by standardizing the activation neuron of each input variables per batch, such as the activations of a node from the previous layer [18]. In fact, the input variable to this layer become distributed at mean 0 with standard deviation of 1. Such a standardization of inputs of the layers substantially reduces the amount of epoch required for convergence, as well as it can has a regularization effect. With other words, all the layers should have their inputs to be normalized after the activation from the prior layers. Below formulae of batch normalization, with two learnable parameters as γ and β and two non-learnable parameters as moving averages.

$$\mu_i = \frac{1}{M} \sum A_i \tag{1}$$

$$\sigma_i = \sqrt{\frac{1}{M} \sum (A_i - \mu)^2} \tag{2}$$

$$\hat{A}_i = \frac{A_i - \mu_i}{\sigma_i} \tag{3}$$

$$\tilde{BN}_i = \gamma \otimes \hat{A}_i + \beta \tag{4}$$

$$\mu_{mov_i} = \alpha \mu_{mov_i} + (1 - \alpha)\mu_i \tag{5}$$

$$\sigma_{mov_i} = \alpha \sigma_{mov_i} + (1 - \alpha)\sigma_i \tag{6}$$

The equations (1) and (2) correspond to mean and standard deviation for each feature column per batch. Equation (3) relates to standardization of the features. Equation (4) is for batch normalization operation, namely element-wise product of γ and $\hat{A}_i$ and addition with β. Eventually, the equations (5), (6) are called exponential moving average (EMA) of mean and standard deviation, they are calculated during the training but never used in this process [18]. These variables are saved as states for inference of the model. Since the training takes the batch operations, but inference only one sample at a time, thus no mean and standard deviation can be obtained from single sample. So, the EMA variables are used here for inference of the single samples.

Also, there is thought that batch normalization makes the optimization process smoother, the gradients become more predictive which brings up the faster model convergence [19]. In practice, the batch normalization is very applicable in any sort of deep learning problem [18]. It can be used with any kind of neural network model, it supports high learning rates for faster convergence, less sensitiveness to proper weight initialization. Thus, the exploding and vanishing gradients problems are not considered as critical for batch normalization [18].

In case of the target model of this thesis project, YOLOv4, this model exploits the extension of BN called cross mini-batch normalization (cmBN). The cmBN technique resolves the main issue of BN – problem of small mini-batches, small size of mini-batch prevents statistical knowledge for normalization to be calculated correctly [19]. Using the cmBN, the multiple examples from several batches can be used together to estimate more stable statistics. Thus, using the cmBN method based on Taylor polynomials, the BN techniques can be successfully utilized in small-batches training scenario.

2.2 Backbone Architectures

2.2.1 Visual Geometry Group

One of the most popular family of CNN models is the VGG. This architecture was the first model that beat the AlexNet on ImageNet contest in 2014 [10]. This model was considered as the new standard for image classification problem. The VGG model is a quite large model, it exploits from 11 to 19 convolutional layers with 3-by-3 filter size and max-pool layers, and 3 fully connected layers at the end before SoftMax layer.

In details, first two fully connected layers have 4096 nodes and third has 1000 nodes. The [10] authors have suggested that the depth of the model plays a crucial role in the accuracy metric and largescale image classification design. Since the main contest VGG was used is the ImageNet, the 1000 node SoftMax layer was used, each node for each class. The VGG family has beaten the previous winners of the same contest, AlexNet model, overall VGG's top-1 validation error was only 23.7 %, that was impressive [10]. VGG has established the new standard of CNN design, there are currently a great deal of VGG-family CNN models that are used as the main feature extractors for problems of various complexity [10]. In our work, we have done numerous experiments with VGG family of models as the backbone for the YOLOv4.

2.2.2 Residual Network

Another milestone model that made a change in the image classification contests is the Residual Network (ResNet) model. ResNet introduces the residual connections in the network definition. This residual connection brings skip connection between several nodes of network, making it preserved from vanishing gradient problem, which is very critical problem in ML, in general. The skip connection provides alternate path for gradients to follow [20], [16]. The use of extra deep layers in ResNet model became more stable, the performance increases with this residual blocks. Since the residual blocks allow model to learn the identity functions, which guarantee that higher layer performs at least as the lower level. The succeeding layer do not degrade performance since the identity function is now present in the structure.

The ResNet architecture is inspired by the previously discussed VGG model, namely VGG-19 [20]. However, ResNet is now added residual blocks between convolutional layers of the model. The ResNet architecture has already proved its efficiency in ImageNet and MSCOCO contests in 2015-2016 years, many modern object detection models use the ResNet family as the backbone [21], [20]. Practice and time have shown the efficiency and power of ResNet in various computer vision problems, however one thing that makes ResNet not a universal solution is the model's very long training process in case of large networks [10]. This problem makes the entire architecture to be not practically applicable in the fast real-world environment [20]. The ResNet-101 and ResNeXt-101 models were candidates for the proposed model's backbone architecture during the development stage.

$$y = \mathcal{F}(x, W_i) + x \tag{7}$$

Where, y, x are the output and input vectors for the specific layer and $\mathcal{F}(x, W_i)$ is the function of the residual mapping. The ResNet models are still considered as the adequate choice for any computer vision problems, they are balanced in terms of model complexity, accuracy, and inference time. As well as this diversified family of models can recreate baseline results for tasks of any complexity [20].

2.2.3 EfficientNet

The EfficienNet is the model developed by the Google team, it is the CNN architecture and scaling model that uniformly scales the all the network inner dimension such as width, depth, resolution via the compound coefficient [11]. The scaling is structured with constraints and is not arbitrary, thus these scaling coefficients are fixed and predetermined [11]. As an example, say we want to increase the $2N$ more computation resource, we can easily increase the network depth by αN, width by βN and resolution by γN, these α, β and γ coefficients are calculated using the small grid search on the original model configuration [11]. Basically, the intuition of this CNN architecture is quite straightforward, the bigger size of images model will get, the bigger the receptive size of filters and layer dimensions the model should have [11]. Considering the details of this research input, the depth scaling of CNN is commonly thought to be very promising, since by increasing the number of layers usually returns the capability of learning more complex patterns in data distribution [22].

However, large networks became very difficult to train due to the vanishing gradients problem, even though several regularization techniques are applied, large deep CNN become ineffective. Next, the width scaling can potentially bring increased accuracy, since more complex features can be learned from the data. However, the increase in the width only cannot provide good results, since shallow networks tend to have issues in learning the higher-level features [20]. Same thing happens for network resolution increase, the higher the input images the higher information from image to be learned, more fine-grained patterns. However, after increase in resolution, the accuracy increase saturates, signalizing about the further inefficiency of the resolution size increase. The authors of [11] come up with following idea, that scaling up any dimension of the network, namely depth, width or resolution makes it capable to improve overall accuracy, but the accuracy increase rate diminishes for large models.

Another point that can be concluded on the work, better accuracy can be achieved by gentle balancing of all the scaling parameters (width, depth, resolution) together [11], [23]. The EfficientNet architecture itself consists of the nine convolutional layers with kernel size of 3, max-pooling layer and one fully connected layer in the end. Here, the comparison of few models is done, the first EfficientNet model – B0, top-1 accuracy was 77.1%, ResNet-50 – 76%, and DenseNet-169 – 76.2%, considering that EfficientNet has only 5.4 million parameters while other two have 26 and 14 million of parameters, respectively. Authors have tried to scale the MobileNet and ResNet architectures with their approach to see the improvement. Scaled-Res-Net-50 with compound scale achieves extremely high 78.8% accuracy while its baseline mode has only 76.0% [11]. Same happens for MobileNet, compound scale version has accuracy of 77.4%, baseline model has 72%. We can see the effectiveness of the proposed scaling method in optimization of the popular models that won lots of contest on image classification. The equation (8) above shows the corresponding scaling parameters that adjust the network scaling in different width, depth and resolution. We have tested EfficientNet family of models with YOLOv3 detector in our experiments. We achieved optimal results for detection, but faced several problems with detector training procedures, which will be discussed below.

$$\begin{gathered}\alpha\beta^2\gamma^2 \approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1\end{gathered} \tag{8}$$

2.2.4 EfficientNet v2

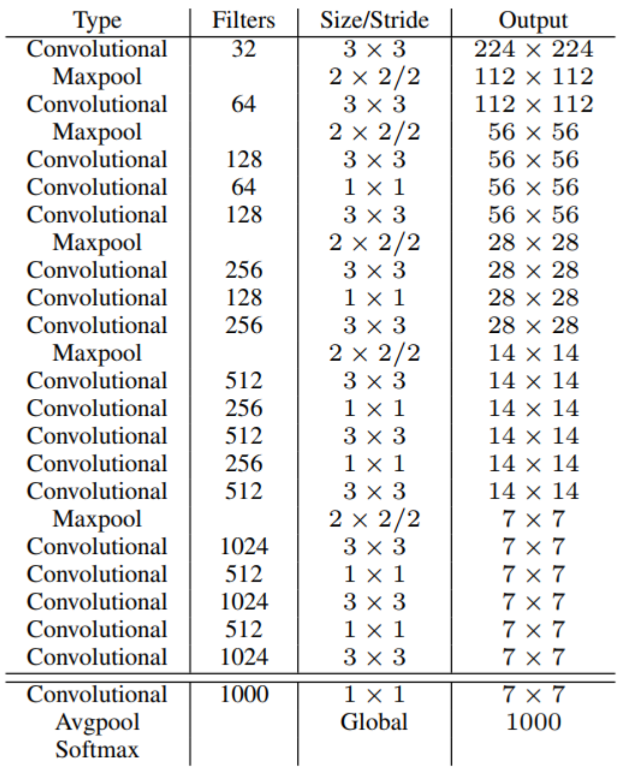
This model is the advancement of the EfficientNet towards the elimination of redundant calculation giving a family of lightweight models with higher accuracy and parameter efficiency and lower inference time. From authors [24], the EfficientNet2 has 24 million of parameters against 43 and 164 million of parameters of EfficientNet and ResNet-RS models respectively, while achieving 83.9% top-1 accuracy. The EfficientNet and ResNet-RS have 84.3% and 84.0% top-1 accuracies on the same ImageNet 2012 dataset [24]. The key idea lies within the training-aware neural architecture search, this module heuristically finds the best possible design of architecture for feature maps propagation. However, the model still lacks the clarity of the entire training process as it was with EfficientNet.

Even though, the model became faster, lighter, more accurate the training of this family of models still requires a vast computational power [24].

Another modification of this model is the progressive training concept. In other words, it means that the training images' size will gradually increase with time, so that model will be training on images of grates size than they were in the beginning of the training. This approach is aimed to remedy the EfficientNet's weakness on training of large images. This increase of image size works as regularizing effect, the authors propose that the regularizing effect also must be dynamic during the dynamically training model. Thus, the resolution of images increases to meet the better training results [24].

2.2.5 Darknet

This CNN model is developed primarily for object detection YOLO architecture. There are various versions of it, Darknet-19, Darknet-53, CSP-Darknet and many others [9]. In fact, Darknet 19, 53 has 19 or 53 convolutional layers and it is pretrained mainly on ImageNet dataset. Over the years, this architecture was chosen as the main feature extractor of YOLOv3, YOLOv4 models. The main reason is the simplicity and training availability of the network [25]. Figure 2 demonstrates the Darknet-19 model, it has 19 convolutional layers. It should be mentioned about CSP-version of Darknet, which stands for Cross Stage Partial Network, below we will consider this model in detail. By the end of our experimental work, we come up with belief to choose Darknet CNN architecture as the main feature extractor for our work due to its simplicity in terms of training and architecture design. In additon, authors in [26], [27] assure the effectiveness of Darknet models with YOLOv4 detector.



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 2: Darknet-19 CNN Model.

2.3 Advanced Techniques of Information Propagation

This section is aimed to give a review on the most relevant DL techniques that improve and optimize the information path within the network. It is not only how features are generated in model but how they propagate to the end of the model and which path and mathematical manipulations they are exposed to.

2.3.1 Cross Stage Partial Network

The CSP network is the popular technique applied to CNN backbones to implement following targets: reduce amount of computation and achieve a richer gradient combination of the CNN networks. In essence, CSP implements model lightening so that low-efficient processors could inference models effectively. These modifications bring a significant boost in terms of reduced computations by 20% [28]. Authors of [28] propose the issue of duplicate gradients during the network optimization, a great deal of gradient information is redundant during the training and can be effectively eliminated. In details, we can suggest three main advantages of CSP in object detection applications. This concept is illustrated in Figure 3 below.

Firstly, the strengthening learning ability of network. The process of model lightening significantly degrades the CNN model performance. The CSP suggests reinforcing the learning ability, so that the new lightweight CNN model becomes faster and remains as accurate as before [28].

Secondly, the removal of computational bottlenecks of the model. Any CNN model contains computational bottlenecks during the inference, obviously this is not desirable for real-time object detection. Authors [28] suggest eliminating these computational bottlenecks by evenly distributing the computations within the model layers. As a result, the inference process becomes more optimized, for example: CSP-YOLOv3 modification has shown the decrease of 80% in computational bottleneck when tested on MSCOCO dataset [21].

Thirdly, CSP suggests reducing of memory costs, namely consumption of the dynamic random-access memory during the training/inference is substantially reduced. Authors [28] propose the cross-channel pooling that compresses the activation maps, in result the feature pyramids take on compressed input features. For instance, testing on PeleeNet model, the memory consumption reduction was recorded to be 75% [28]. CSP provides a great framework to efficiently eliminate redundant mathematical computations within the network.

Below, we can consider the CSP-DenseNet [6] model to get a clearer understanding. The main concept of DenseNet is that model is reusing features coming from first up to

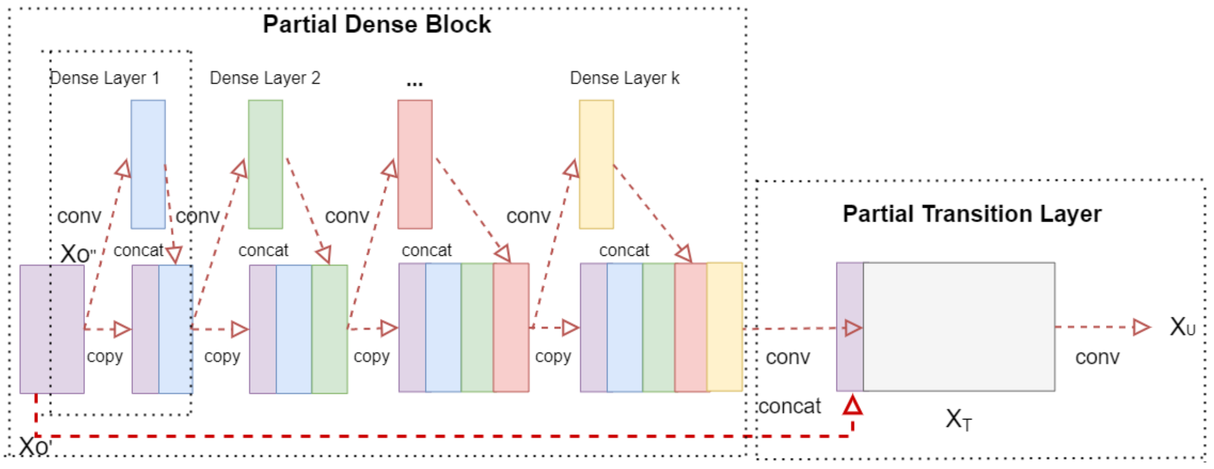


Figure 3: Cross Stage Partial Network.

last layers, namely, outputs of all preceding layers become inputs to all the succeeding layers [6]. The CSP-DenseNet remain this feature reuse property of DenseNet as well as it blocks a redundant gradient information by slicing the information flow [28]. The CSP-DenseNet provides two main layers: partial dense block and partial transition layer.

The partial dense block is used to increase gradient path, balance the computations within each layer and reduce memory traffic. Due to the split and merge process, the number of gradient paths is doubled. The channel number of DenseNet layers is greater than the rate of growth. Thus, layer channels are responsible for half of the original number, they can also release half of the computational bottlenecks [28]. And the memory traffic is reduced due to the eliminated duplicate gradients in network [28].

Next, there is partial transition layer, which is responsible for the maximization of differences between the gradient combinations [28]. In other words, this layer is a hierarchical feature fusion mechanism by preventing layers to learn duplicate gradient information. There are two strategies developed authors, the fusion-first, and fusion-last strategies. The fusion-first concatenates feature maps first, and then does transition part, in this case great deal of gradient information will be reused. In case of fusion-last strategy, the output first goes through the transition layer and does the concatenation

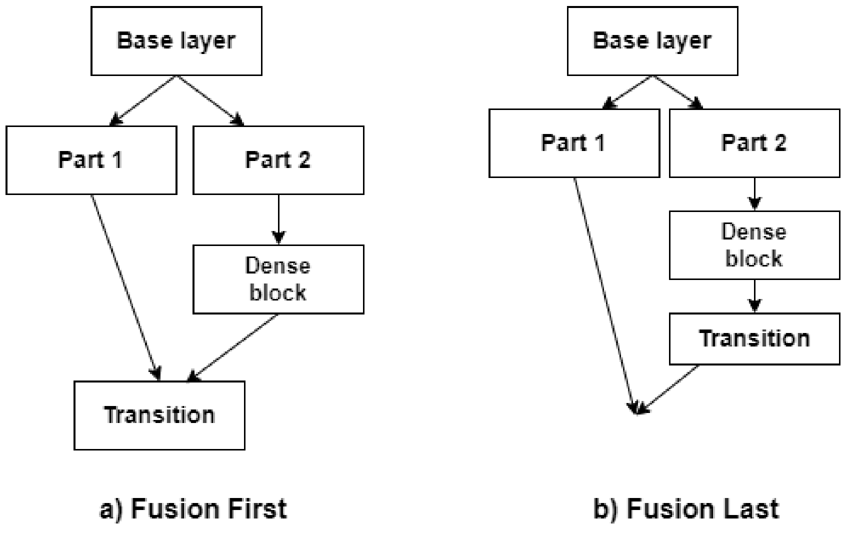


Figure 4: CSPNet Variations.

with feature map coming from residual connection, this scenario excludes reuse of gradient information due to the truncation [28]. Figure 4 above demonstrates this process. For the SAMD-YOLOv4 model, we have used CSPDarknet-53 backbone which inherits all the modifications of the CSPNet.

2.3.2 Convolutional Block Attention Module

This subsection is going to introduce the highly effective module of attention mechanism in the CNN networks. The attention module is the simple yet effective module that makes CNN models to learn and focus on more semantically rich information, rather than non-valuable background information of the input [29]. The attention mechanism is applied to spatial and channel dimensions, namely block attention module consists of spatial attention module (SAM) and channel attention module (CAM). See Figure 5 below.

Firstly, the CAM block is implemented. The input feature map F goes as input to the CAM with dimension of $C \times H \times W$, CAM outputs a $1D$ channel attention map with dimension of $C \times 1 \times 1$. The equation below corresponds to the output channel map of CAM. See Figure 6 for details.

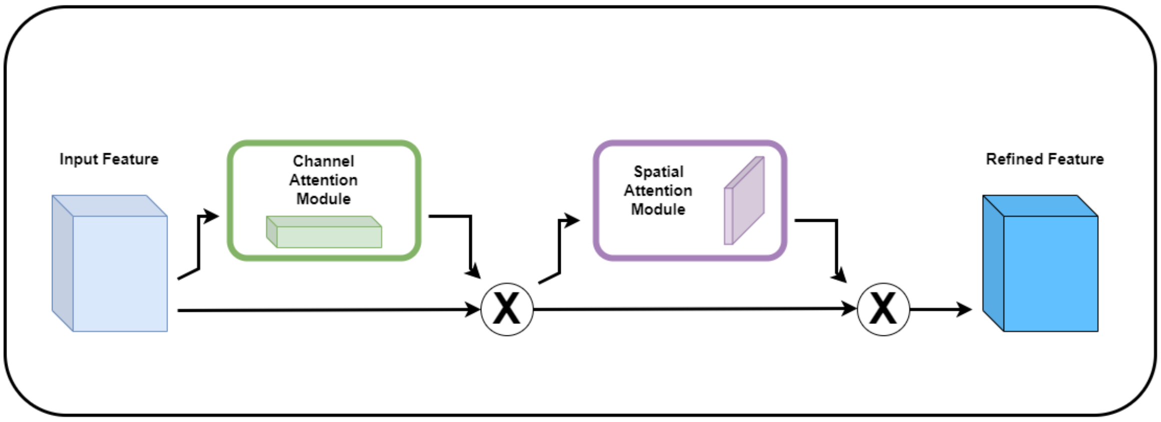


Figure 5: Convolutional Block Attention Module.

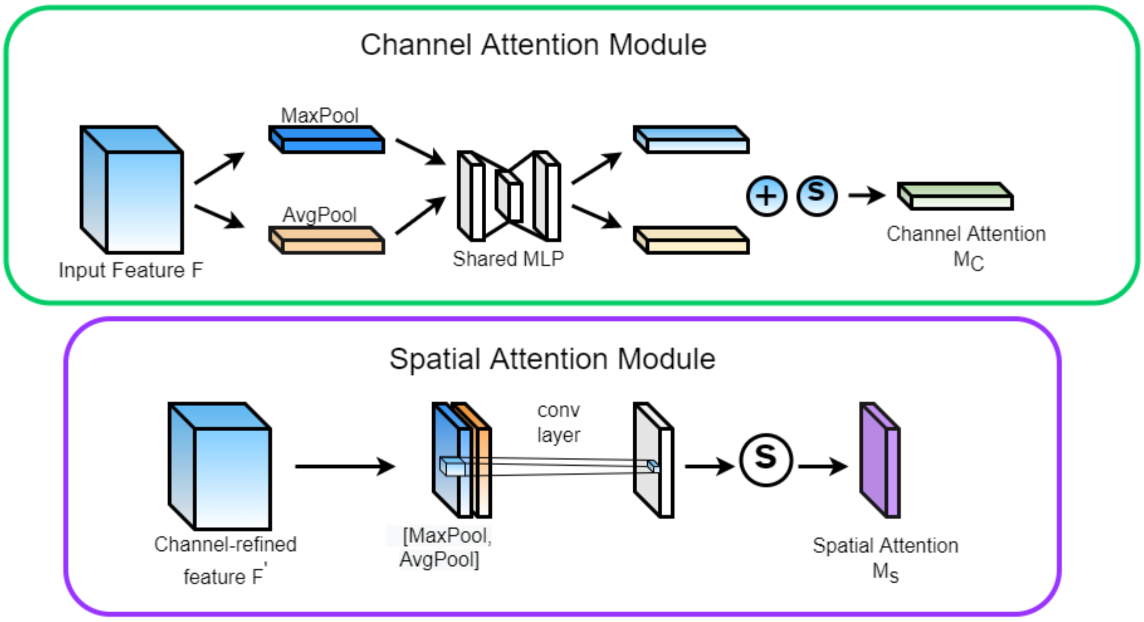


Figure 6: CAM and SAM modules.

$$M_C(F) = \sigma[MLP(AvgPool(F)) + MLP(MaxPool(F))] \tag{9}$$

Secondly, the F feature map is then multiplied with CAM output element-wisely, forming some refined feature map F'. This refined feature map undergoes the SAM module as shown in equation below.

$$M_S(F') = \sigma[f^{7\times7}(AvgPool(F'); MaxPool(F'))] \tag{10}$$

The SAM outputs a feature map with dimension of $1 \times H \times W$. Then, this output is multiplied with F' element-wisely to get the final feature map. See Figure 6 above. Then, this resulting feature map of CBAM is added to the feed-forward CNN modules. To conclude, this attention module suggests both channel and spatial feature processing, as a result we have CNN models with higher performance metrics with no change in computational load [29].

2.3.3 Path Aggregation Network

This subsection introduces the path aggregation network (PANet). PANet resides in the neck part of the model, after backbone and before head (predictor). Mainly, we can conclude three main points of PANet use below. Firstly, the information path from low-level is shortened as well as feature pyramid stricture is improved by adding localization signals from low levels. Also, the bottom-up path augmentation is introduced, coming after the feature pyramid [30]. See Figure 7 below. Secondly, the broken information path is recovered for all feature levels by using the adaptive pooling layer. This pooling layer aggregates features from all levels, such as technique with a better framework for object detection and semantic segmentation, since information is gathered from different feature levels [30]. Finally, the fully connected layer is added to aggregate all the information and provide outputs for class probabilities and bounding box coordinates [30].

The proposed bottom-up path augmentation structure significantly improves model performance since it augments all the information from spatially rich to semantically rich information level (bottom-up) and introduces the lateral connection from lower level

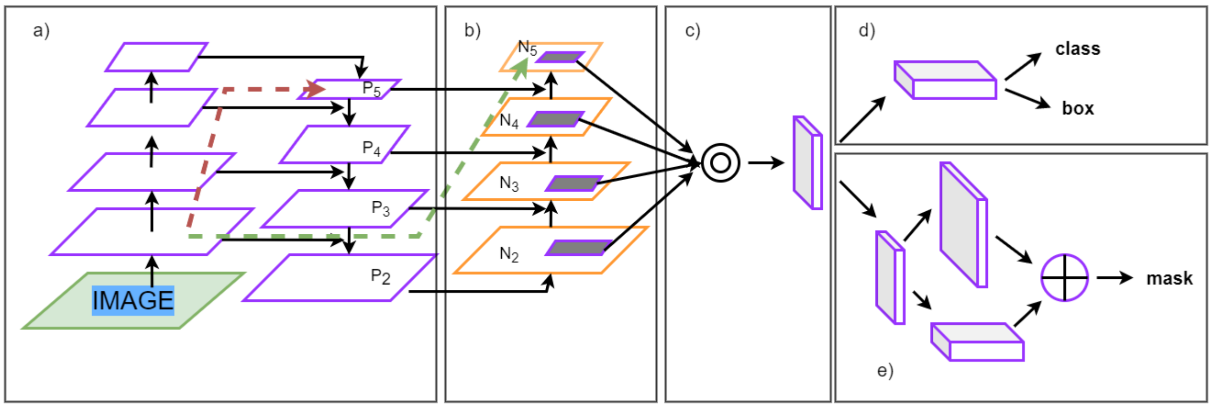


Figure 7: Path Aggregation Network. a) FPN backbone b) Bottom-up augmentation c) Adaptive feature pooling d) Box branch e) Fully-connected fusion

of FPN to high level of bottom-up path. This lateral connection is 10 layers and can efficiently shorten the information path [30]. The proposed solution significantly improves model aimed at semantic segmentation and object detection by enhancing information propagation. For example, the authors' PANet got 1st place at COCO2017 Challenge Instance Segmentation Task and 2nd place at Object Detection [30], [21]. The PANet is used in the YOLOv4 model in the neck part of the model.

2.3.4 Feature Pyramid Network

The feature pyramid network (FPN) is the technique of information propagation between feature extractor and predictor [31]. Namely, the key idea is following: the FPN suggests a multi-level feature maps creation in the bottom-up and top-down fashion [31]. See Figure 8. This information flow provides the combination of both spatial and semantic information from different scales. The bottom-up path provides the propagation from spatially rich feature maps to semantically rich ones. In other words, the size of feature maps gradually decreases on the way up. Analogously, in the top-down path, the feature maps goes from high to low levels, from semantically rich information to spatially rich information. The approach uses lateral connections to save information and later combine feature maps. Such a strategy helps a model to develop an induction from different scales of input data [31]. This guarantees improved robustness and generalization, since different feature map levels were provided during the learning process [31] as well as the the representational power of CNNs substantially rises [31]. Specifically, the FPN model has substantially contributed to the popularization of this approach. There are lots of different feature pyramid like structures that extend main idea originated at FPN. Other examples Bi-directional FPN (Bi-FPN) [11], Network architecture search FPN (NAS-FPN) [32] and others.

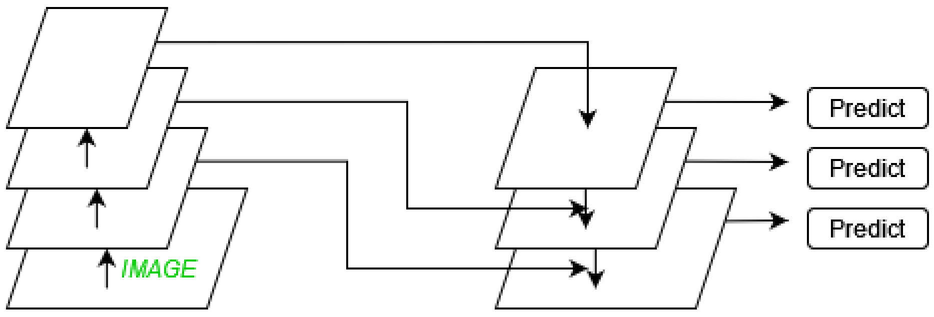


Figure 8: Feature Pyramid Network.

2.3.5 Spatial Pyramid Pooling

The last technique to be discussed is the spatial pyramid pooling (SPP). SPP is the module that aims to pool activations from feature maps of the backbone and locate them in spatial bins of different sizes. Then, these spatial bins are flattened and concatenated together providing the output of fixed size. This approach is reasoned by the fact that CNN models mainly need a fixed size image during the training, even if the images originally have different sizes, they need to be resized/cropped before processing [33]. This brings several disadvantages such as lowered spatial information knowledge of the data distribution. The results of [34] has shown that use of SPP provides model with images of different sizes which implies that model learns much more variable information. This can be seen in the Figure 9 below. Authors mainly used max pooling for retrieving the information from filters. The YOLOv3 [9] and YOlov4 [26] used SPP in the models.

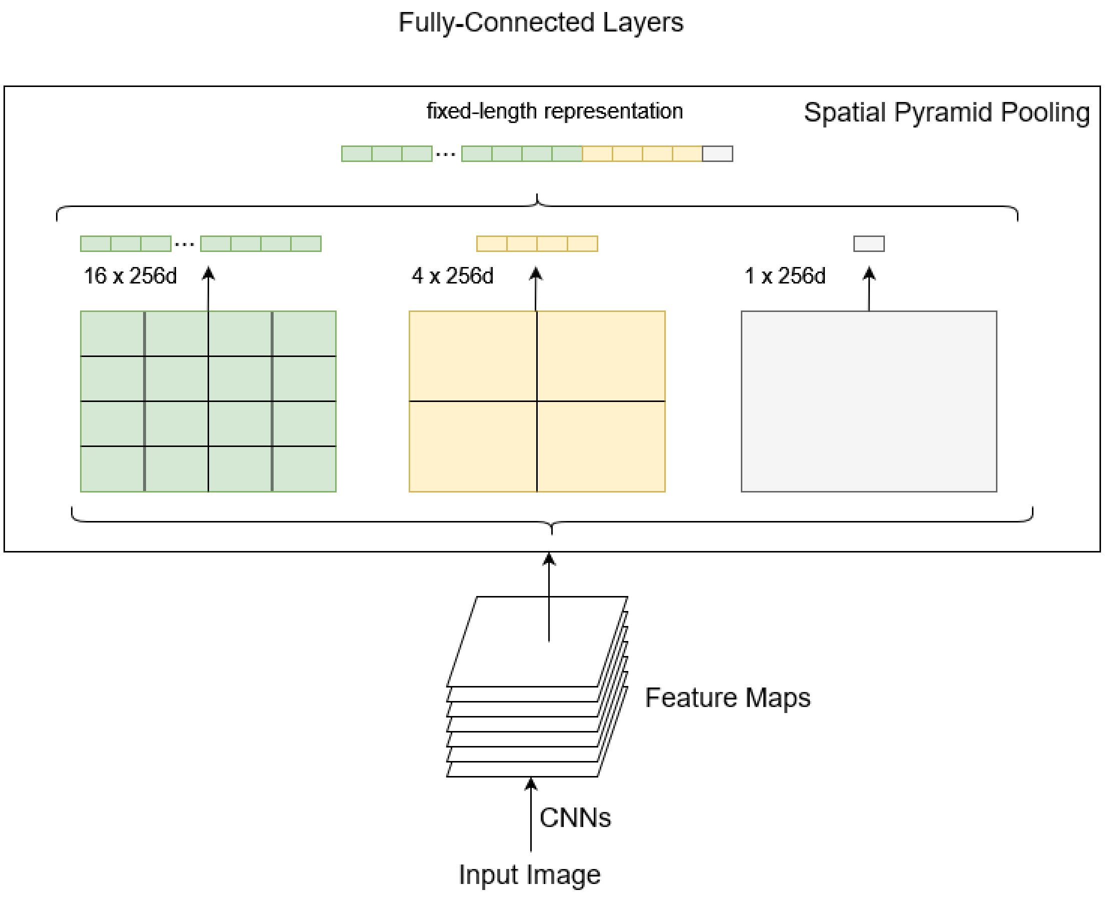


Figure 9: Spatial Pyramid Pooling.

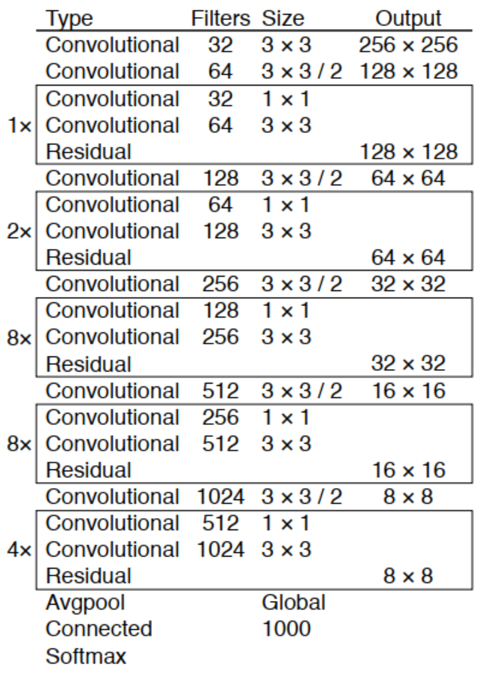
Chapter 3 - YOLOv4 Architecture

In this Chapter the detailed description of the YOLOv4 model will be presented, starting from the backbone design for feature extraction, neck where the sophisticated information propagation from backbone to head is established. Eventually, the head module of the model will be discussed, this part is related to the class prediction and bounding box generation.

3.1 Backbone

As it was mentioned above, the backbone is responsible for features extraction, it scans and outputs out the most interesting information in terms of statistical patterns from the training data distribution. Starting from the 2015, YOLO family models exploited different backbone designs such as VGG-18, VGG-19, ResNet, ResNext, and its native one Darknet family of models. Authors of [9] have used Darknet-19 and Darknet-53 in their experiments, bringing the top-1 score of 74.1 and 77.2 on ImageNet [12], respectively, these models provided relatively high billions floating operations per second (BFLOP/s). Thus, authors of [26] decided to stick to this family of models but modifying it with CSPNet [28]. The primary backbone architecture of the YOLOv4 is taken as CSPDarknet-53 with 512-by-512 input resolution, 53 convolutional layers with kernel size of 3×3 which provides a 725×725 receptive field and 27.6 millions of parameters [26]. This modified backbone is considered as the baseline for further out-of-backbone modification in neck and head of the model, since it produces optimal results for being a backbone for a detector.

Additionally, the authors [26] introduced two techniques for enhanced performance, Bag of Freebies (BoF) and Bag of Specials (BoS). The BoF is aimed to introduce addi-



	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 10: Darknet-53 CNN Model.

tional techniques for robust training with almost no effect on computational cost during the inference of the model. The BoF includes data augmentation techniques which provide variability in data distribution [26]. The main distortion types are geometric and photometric, former introduces random cropping, scaling, rotation, and others, latter introduces variations in rightness, saturation, hue and others. One more BoF feature is the bounding box objective function, which estimates the loss of the generated bounding box. There are many variations to calculate regression of the box coordinates such as mean square error (MSE), intersection over union (IoU), generalized IoU (GIoU), Distance IoU (DIoU) and Complete IoU (CIoU), each of these performs more fine-tuned estimation of the IoU loss of two bounding boxes.

Another feature of [26], is the BoS which introduces a significant boost in model's performance with a negligible overhead in computations. BoS is mainly related to the in-model modules and post-processing operations. The performance is enhanced by increasing the receptive field size, attention mechanisms, and many others. They are SPP, ASPP, RFB, SAM, FPN, BiFPN and activation functions such as rectified linear unit (ReLU), LRELU, Swish, Mish. The BoS techniques mainly improves the model accuracy by a negligible increase in computational cost. It should be mentioned that there are great details about both BoF and BoS, but it is out of the scope of this work.

3.2 Neck

This part reveals information about how the information propagates from backbone to head predictor. Authors of [26] suggest different blocks to be used, such as SPP, ASPP, RFB, SAM, FPN, PANet, NAS-FPN, BiFPN and many others, several experiments with different combinations of these block within the neck were done in [26], [27]. The ablation study on these blocks is provided in [26]. The baseline model of [26] is comprised of CSPDarknet-53 backbone, SPP, SAM, PANet blocks and YOLOv3 head predictor. The PANet and SAM modules were modified by authors, SAM module now performs point-wise attention instead of spatial-wise attention, whilst PANet now uses concatenation of maps.

Additionally, the detector of YOLOv4 exploits these modules: CIoU-loss, cross mini batch normalization (CmBN), DropBlock regularization, Mish activation and DIoU-NMS. According to authors, these modules demonstrate optimal results for the baseline performance. The research and development communities are actively develop and modify YOLOv4 models with various kinds of network architecture manipulations. There are currently a great deal of YOLOv4 models with different module combinations.

3.3 Head

The YOLOv4 models exploits YOLOv3's predictor to predict class probabilities and bounding boxes [9]. It should be mentioned that starting from [25] YOLO's family exploits anchor-based approach for bounding boxes generation. Namely, the model's head predicts four parameters t_x, t_y, t_w, t_h for each bounding box, which are combined with prior bounding box parameters and top-left corner of the image to output final prediction of bounding box. Following equations below show the dependencies. The parameters c_x, c_y are offset from the top-left corner of the image, e is exponential function, p_w, p_h are the prior bounding box width and height, respectively. See Figure 11 for details. The sigmoid activation $\sigma(x)$ is used to range coordinates between 0 and 1, these coordinates are relative to the cell.

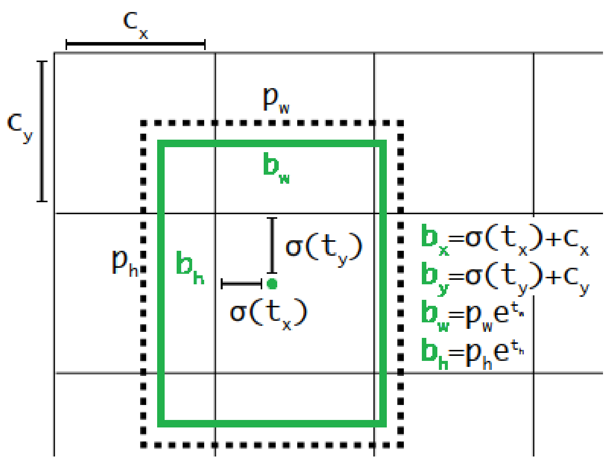


Figure 11: Bounding Boxes Prediction Parameters.

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{11}$$

Authors of [9] designed YOLOv3's detector to predict bounding boxes at three different scales, it is similar to FPN approach. This provides better prediction of the objects of different scales: small, medium and large objects, this scaling is done by down-sampling the feature maps by 32, 16, 8 [9], [35]. Eventually, the output is a 3-D tensor which encodes bounding box, objectness and class predictions. For instance, for MSCOCO dataset, there is $N \times N \times [3 \times (4 + 1 + 80)]$ shaped tensor, which outputs 4 bounding box parameters, 1 objectness and 80 class predictions.

Chapter 4 - Proposed Design

In this section, we will propose a method of YOLOv4 modification, called SAMD-YOLOV4. As a baseline YOLOv4 model, we took a version presented in [26], which exploits CSPDarknet-53 backbone, SPP module and YOLOv3 head detector. Our approach aims is to alter the neck of the model by introducing the SAM module [29] followed by dilated convolution layers on different feature map scales and reduce number of convolutional filters before SAM modules to reduce model complexity. The activation function of all the convolutional layers is leaky-ReLU. We believe this method would positively affect model's general performance, since we implemented spatial attention module as well as dilated convolution layers in the model, so that more spatially correlated features are activated together and wider receptive field is used to extract global information from map at the lower computational cost [36]. As it was said above, the modifications are done only at model's neck part, no backbone and head predictor are affected. Figures 12 and 13 demonstrate SAM module and dilated convolutional layer.

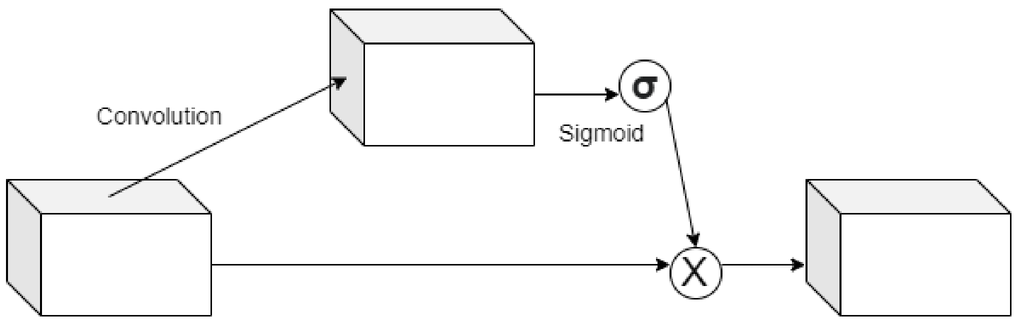


Figure 12: Spatial Attention Module.

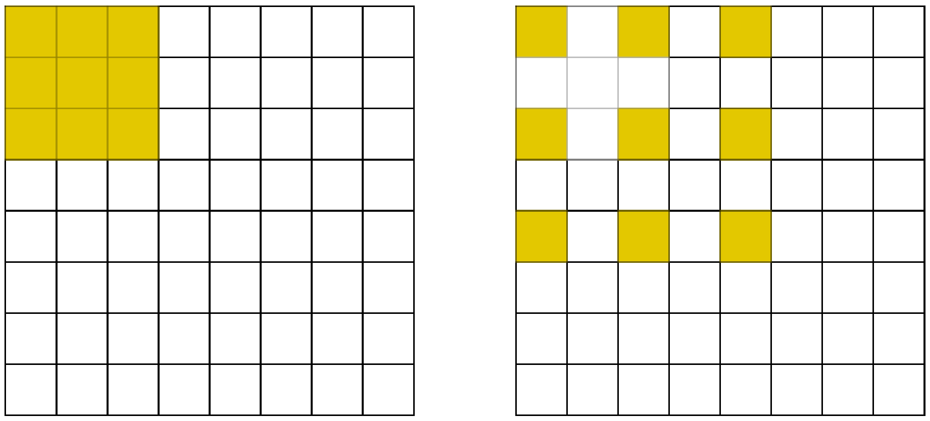


Figure 13: Standard Convolution (Left) and Dilated Convolution with Rate 2 (Right).

4.1 Spatial Attention Module

Main motivation of our approach is related to the fact that feature maps coming from the backbone contain redundant information about the background and other irrelevant objects during the training. By introducing the SAM module, we can bring up the attention mechanism and make network to focus on more relevant features. It should be mentioned that we did not use the CBAM completely, only SAM module without CAM was used. This means that attention mechanism is implemented in spatial dimension, without channel dimension, no depth-wise operations are done. We have used three SAM modules at three different scales of YOLOv3 head predictor, so that all three prediction scales get spatial attention mechanism.

In the network, first SAM module starts at 135^{th} convolutional layer with 256 filters, each with kernel size 3×3. Layer then splits into residual connection and convolutional layer with 256 filters, each with kernel size 1×1 which is followed by sigmoid activation. Eventually, these splits are then multiplied element-wisely giving the final refined feature map of dimension $64 \times 64 \times 256$. This process is identically replicated for other two SAM modules at 149th and 163rd layers for medium and small scales, respectively. See Table 1 for SAM details.

Table 1: Spatial Attention Module Description.

Layer Number	Input Dimension	Output Dimension	Kernel Size	Kernel Size
135	$64 \times 64 \times 128$	$64 \times 64 \times 128$	3×3	1×1
149	$32 \times 32 \times 256$	$32 \times 32 \times 256$	3×3	1×1
163	$16 \times 16 \times 512$	$16 \times 16 \times 512$	3×3	1×1

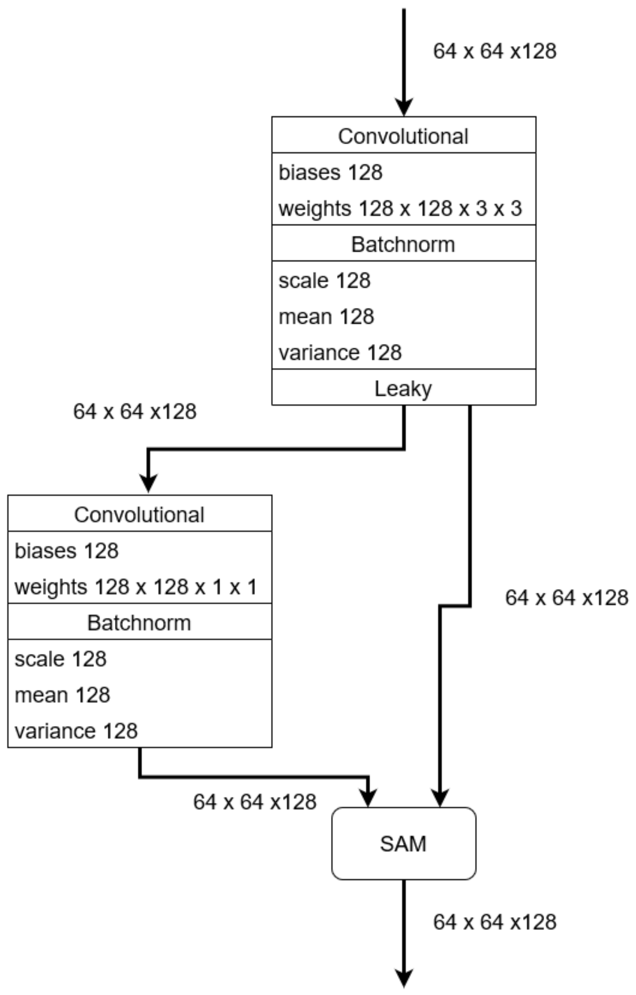


Figure 14: First SAM Block Visualization.

4.2 Dilated Convolution

Then, after the SAM module, the dilated convolutional layers take place. At first scale, the 138th layer consists of 256 filters each with kernel size 3×3 and dilation rate 3.

The output's dimension stays same $64 \times 64 \times 256$. Again, same process is done for other two prediction scales. The main idea behind use of dilated convolution is to increase the receptive field of the kernel, which provides a wider global information about features on which kernel is convolving. This approach introduces no extra computations or increase in number of model parameters. See Table 2 for dilated convolution details.

Finally, our approach in the SAMD-YOLOv4 model utilizes SAM modules and dilated convolutional layers to compensate the reduction in model's learnable parameters. By implementing more effective feature encoders the lots of redundant parameters can be discarded from the model, which can reduce computational time during the inference. In the next Chapters experimental results of the SAMD-YOLOv4 and baseline YOLOv4 models will be reviewed.

Table 2: Dilated Convolutional Layers Description.

Layer Number	Input Dimension	Output Dimension	Dilation Rate	Kernel Size
138	$64 \times 64 \times 128$	$64 \times 64 \times 128$	3	3×3
152	$32 \times 32 \times 128$	$32 \times 32 \times 256$	3	3×3
166	$16 \times 16 \times 512$	$16 \times 16 \times 1024$	2	3×3

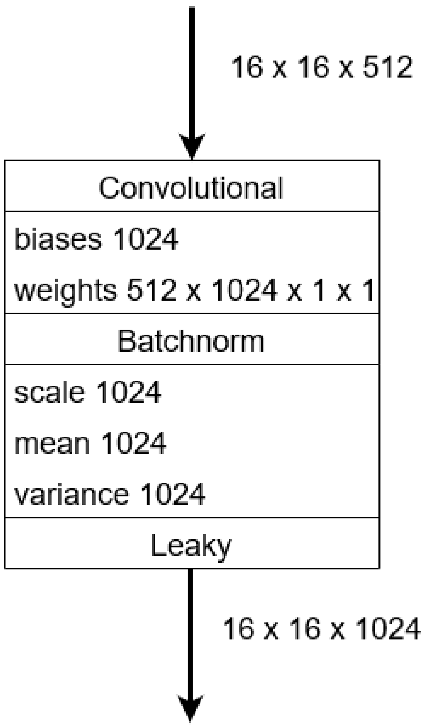


Figure 15: Third Dilated Convolutional Layer Visualization.

Chapter 5 - Results

In this section we will demonstrate the results achieved in the experiments during the model design. We present results of our model and compare them with baseline YOLOv4 model. The comparison study can be considered fair and objective, since both software and hardware setups of authors [26] and ours are identical.

5.1 Experimental Setup

The entire work was done remotely using the desktop provided by university. The hardware experimental setup is depicted in Table 3. Authors [26] had approximately same configurations during their experiments.

Table 3: Table to test captions and labels.

Hardware	Model
CPU	Intel Core i9-7900X CPU @ 3.30 GHz
GPU	NVIDIA GeForce RTX 2080 Ti, 11 GB GDDR6
RAM	50 GB

Considering software specifications, we have used Ubuntu 18.03 and Darknet as main deep learning framework. The Darknet framework is created and maintained by [3] and [26], it is a native framework for YOLO family models. We also considered PyTorch for this work, however due to complications in PyTorch-based realization of YOLOv4, we decided to stick to the Darknet framework. Also, since baseline YOLOv4 model and its paper results are implemented in Darknet, for a fair comparison it would be better to do experiments in Darknet framework. Darknet is a very simple and effective deep learning framework, which provides fundamental building blocks for CNN. We train, validate and test entire model performance in this framework. To install Darknet, several libraries and packages are required such as OpenCV, Cmake, CUDA, CuDNN and others. Far from the main framework, we also have used NVIDIA's high performance inference accelerating framework - TensorRT [37]. TensorRT is used to optimize the CNN model by eliminating redundant computations, fusing layers and tensors, kernel auto-tuning and other manipulations. We will reveal TensorRT exploitation in the next section.

As the primary task of our model is the object detection, we will use famous MSCOCO (trainvalno5k 2014) dataset, to train and validate our model. We will not train the CSPDarknet-53 backbone on ImageNet, since no changes are done to the backbone architecture, thus we will use pre-trained CSPDarknet-53. The YOLO model zoo is maintained by [26] on GitHub account, where we downloaded the ImageNet pre-trained CSPDarknet-53. We take only first 105 layers of the CSPDarknet-53 for object detection training, freeze these 105 layers and train detector on MSCOCO dataset.

The training configuration is depicted in Table 4. The number of subdivisions is the number of batches within the batch. Also, it is suggested to train the detector on MSCOCO dataset for 500500 iterations. The initial learning rate is set to 0.0013, but after warm-up phase, model uses the cosine annealing scheduler for dynamic learning rate. Eventually, it took about 24 days to train YOLOv4 detector on MSCOCO dataset using pre-trained CSPDarknet-53 backbone.

Table 4: Detector Training Configuration.

Input Size	Batch Size	Subdivisions	Learning Rate	Maximum Batches
$512 \times 512 \times 3$	64	32	0.0013	500500

5.2 MSCOCO Evaluation Performance

This subsection will demonstrate evaluation results of trained model on MSCOCO dataset. Namely, we will provide comparison of our model and baseline YOLOv4 model based on average precision (AP), average recall (AR) scores on different threshold levels, BFLOPs, FPS, inference time (without TensorRT).

Tables 5 and 7 depict AP and AR for different IoU thresholds. The AP relates to average AP over IoU thresholds [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]. The AP_{50} and AP_{75} for IoU thresholds of 0.5 and 0.75, respectively. The AP_S, AP_M, AP_L relate to the AP for small, medium and large objects, respectively, all three take average over IoU thresholds [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]. See Table 6 for details. In the same way, results for AR are calculated for different IoU thresholds.

Table 5: Average Precision with 512 × 512 Input Size.

Model	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4	43.6	64.4	47.9	24.7	46.8	55.7
SAMD-YOLOv4 (Ours)	43.3	64.0	47.8	24.5	46.1	55.3

Table 6: Objects' Area for Different Levels

AP	$Area(pixel^2)$
AP_S	$Area < 32^2$
AP_M	$32^2 < Area < 96^2$
AP_L	$Area > 96^2$

Table 7: Average Recall with 512 × 512 Input Size.

Model	AR_1	AR_{10}	AR_{100}	AR_S	AR_M	AR_L
YOLOv4	34.0	55.7	59.7	38.9	64.1	75.7
SAMD-YOLOv4 (Ours)	33.8	55.3	59.3	38.4	63.2	75.5

From above tables, it can be seen that, the AP and AR values are more or less same. SAMD-YOLOv4 demonstrates no changes in precision and recall comparing to the baseline YOLOv4 model. However, when considering the model complexity in terms of BFLOPs from Table 8, we can see significant improvements, proposed model has 7.3% less computational parameters. The decreased BFLOPs value in SAMD-YOLOv4 signalizes about efficient modification of the neck structure, redundant computations were eliminated while still maintaining state-of-the-art performance. The introduced SAM modules and dilated convolutions compensated the removal of the several parameters, so that model's detection performance remained same as the baseline model's.

Table 8: Model Parameter Complexity

Model	BFLOPs	Advantage
YOLOv4	91.095	-
SAMD-YOLOv4	84.452	+7.3% lower parameters

5.3 Darknet Inference

This subsection demonstrates the inference of the baseline YOLOv4 and proposed SAMD-YOLOv4 models within the Darknet framework. Firstly, we measure the inference time per image, i.e. how long it takes to detect objects in one image. We have tested models on 1000 random images from MSCOCO dataset, and took average inference time from both models. Table 9 depicts the inference results. It can be seen, the SAMD-YOLOv4 has a 6.9% lower inference time comparing to YOLOv4 model. The lowered model complexity results in the significantly faster inference.

Table 9: Models Inference Time on Darknet Framework.

Model	Inference Time (s)	Advantage
YOLOv4	0,029	-
SAMD-YOLOv4	0,027	+6.9% faster

Furthermore, we tested the FPS performance of the models via Darknet framework.

We took a short (3:36) video with resolution of 1280×720 and tested two models on it. The Table 10 shows the results obtained. It can be seen that, FPS for both models remained almost the same. The SAMD-YOLOv4's FPS did not increase in similar proportion of 6.9%, because the video inference requires other computations related to the video stream. The testing of the single image and video stream inference are not comparable, since different running environments are involved.

Table 10: Models FPS on Darknet Framework.

Model	FPS (s)	Advantage
YOLOv4	76.3	-
SAMD-YOLOv4	76.6	+0.4% higher

5.4 TensorRT Inference

This subsection introduces results obtained with TensorRT optimization tool by NVIDIA. The TensorRT is library that optimizes DL models through various ways such as: reduced model's precision, tensor and layer fusion, auto-tuning of model's kernel and many others. In other words, TensorRT fuses model and reduces non-valuable precision and providing a model with preserved accuracy and substantially increased throughput. We will not cover the details of TensorRT, since it is out scope.

Firstly, to obtain TensorRT optimized model engines we need to convert the Darknet type models to the Open Neural Network Exchange (ONNX) format. The ONNX is the unified standard of neural networks models, we need to transform our models to the ONNX format first. Next, we use NVIDIA's TensorRT modules to convert ONNX format models to TensorRT formal models, we end up with TensorRT model engines, which are calibrated to the FP16 precision. Moreover, we also could implement other precision options such as INT8, but it needs additional calibration with model parameters. Table

11 shows comparison of Daknet and TensorRT based YOLOv4 model's inference time, it can be seen, the TensorRT version is faster by 66.5%. Similarly, Table 12 shows that TensorRT based SAMD-YOLOv4 is 77.8% faster than its Darknet version. Eventually, Table 13 shows the comparison of two TensorRT versions of YOLOv4 and SAMD-YOLOv4 models, it can be seen that SAMD-YOLOv4 is 38.1% faster than YOLOv4. These results show the high efficiency of TensorRT implementation in DL models.

Table 11: YOLOv4 Daknet and TensorRT.

Model	Inference Time (s)	Advantage
YOLOv4	0,029	-
YOLOv4-TensorRT	0,0097	+66.5% faster

Table 12: SAMD-YOLOv4 Daknet and TensorRT.

Model	Inference Time (s)	Advantage
SAMD-YOLOv4	0,027	-
SAMD-YOLOv4-TensorRT	0,0060	+77.8% faster

Table 13: TensorRT Models Inference.

Model	Inference Time (s)	Advantage
YOLOv4-TensorRT	0,0097	-
SAMD-YOLOv4-TensorRT	0,0060	+38.1% faster

Finally, we present results of the TensoRRT models on FPS. As before, we use same video file and conditions for testing. Table 14 shows the results, it can be seen, the SAMD-YOLOv4 has 188 FPS which is 50.4% higher than baseline YOLOv4 model's. By this, we have shown the crucial improvements in results by exploiting the TensorRT library for the models.

Table 14: TensorRT Models on FPS Inference.

Model	FPS (s)	Advantage
YOLOv4-TensorRT	125	-
SAMD-YOLOv4-TensorRT	188	+50.4% higher

5.5 Model Demonstration

In this part, we demonstrate models' outputs as images with bounding boxes and probabilities, both on Darknet and TensorRT frameworks. Figure 16 shows Darknet based YOLOv4 model, which detects laptop, tv-monitor, cell phone, book, mouse in the image.

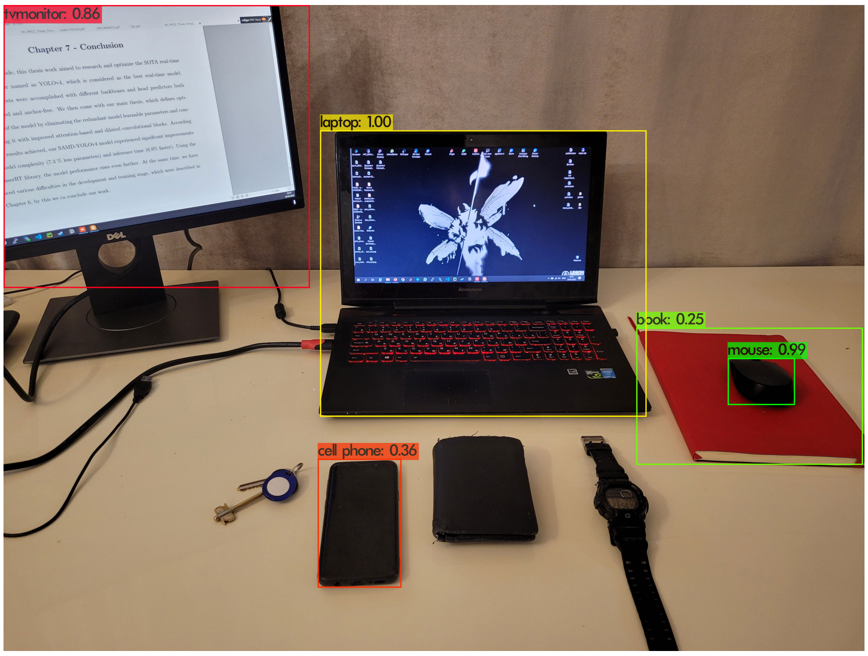


Figure 16: Darknet YOLOv4.

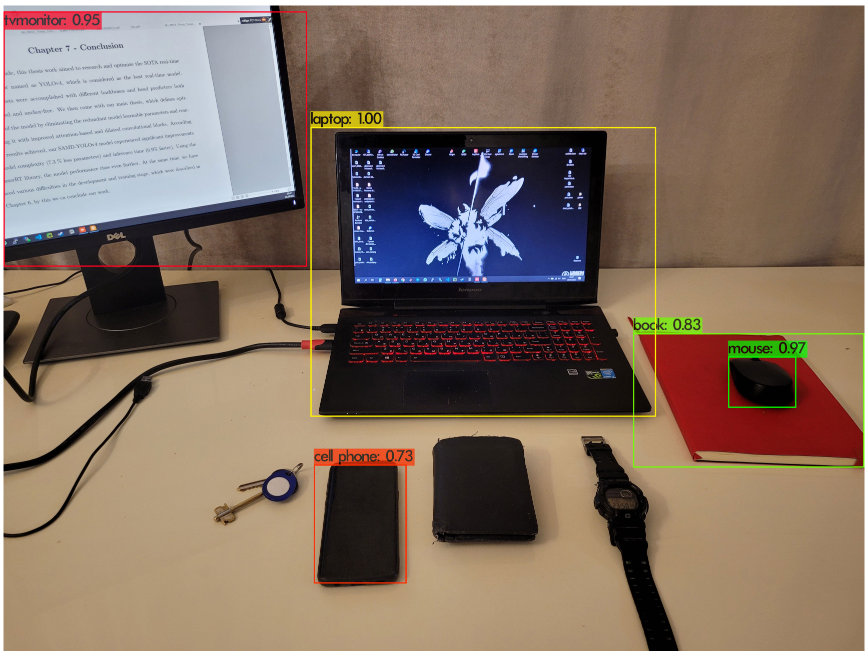


Figure 17: Darknet SAMD-YOLOv4.

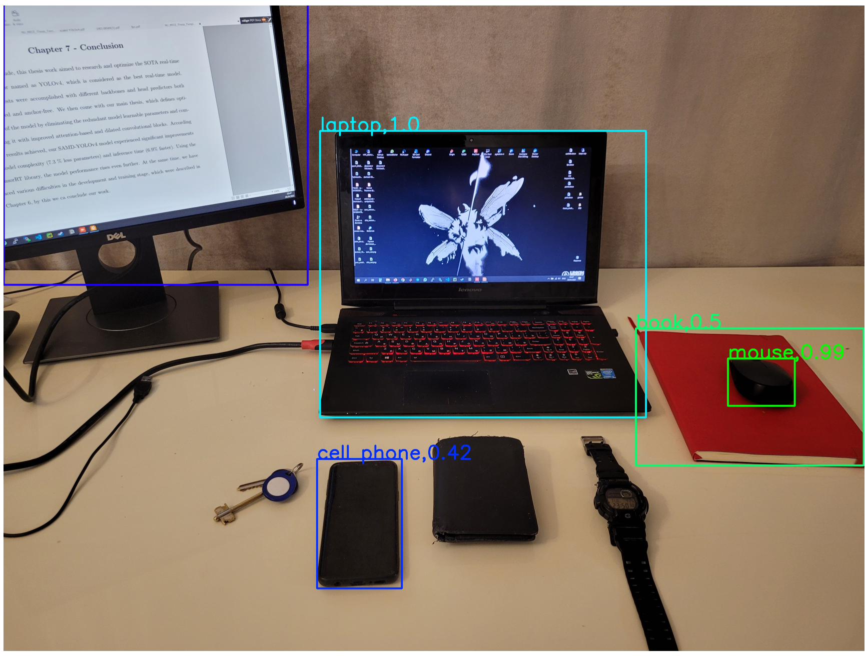


Figure 18: TensorRT YOLOv4.

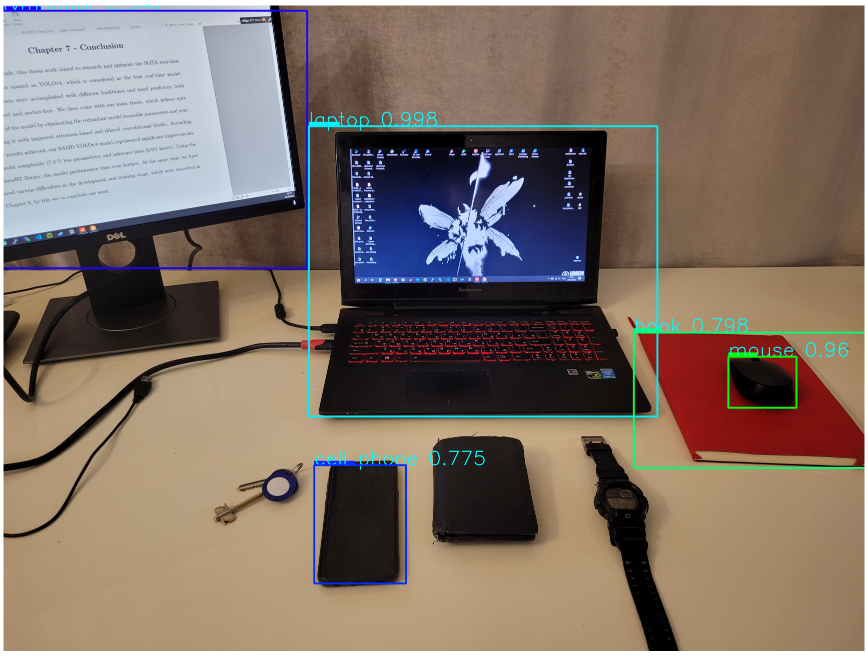


Figure 19: TensorRT SAMD-YOLOv4.

Chapter 6 - Discussions

In this Chapter we provide discussions on the work done. We will address the main motivation of our proposed method, our trials and errors, and limitations faced so far. Also, we provide the potential future research work.

The main motivation of this research work is related to research the SOTA real-time object detector and try to understand and optimize it as much as possible. We have chosen YOLOv4 as the baseline model, which already established itself as the best real-time object detector with SOTA accuracy and extremely low inference time [9], [26].

Firstly, we were focused on model's backbone structure, namely we aimed on model's feature extractor. We have research lots of backbone architectures which were described in details in Chapter 2.2. We tried Darknet, VGG, ResNet, ResNeXt, Efficient-Net (both 1 and 2), the results were unsatisfied. For example, the EffcicentNet family of models appears to be strong and effective, [22] paper demonstrated SOTA results better than YOLOv3. The EfficientNet-B0 was lighter and more accurate than YOLOv3's Darknet53 as well as EfficientDet detector was lighter and more accurate than YOLOv3 detector.

However, the optimal performance of EfficientNet backbone was complicated with its training process. This model is trained tremendously long time and requires a whole GPU cluster to be trained. In our case, this was not a best option, due to limited computational resources. Then, we checked the ResNet-101, VGG-19 and ResNeXt-101 models, ResNet-101 and ResNeXt-101 backbones demonstrated sub-optimal performance when plugged with YOLOv4 detector giving results a bit worse than YOLOv4's native CSPDarknet-53. The VGG-19 performed poorly, barely meet the results of Darknet19 of

YOLOv2, we removed it from candidate list instantly. Thus, we come up with idea to stay with the native CSPDarknet-53 as the primary backbone of our model, since it represented SOTA results on ImageNet dataset as a classifier as well as [26] authors already have done ablation study on it. Authors in [26] and [27] demostrated that CSPDarknet-53 is the best option for SOTA real-time YOLOv4 object detector.

Then, we also tried to consider the different approaches for detector itself. Namely, different anchor-free detectors such as CornerNet [38], CenterNet [39] and CentripetalNet [40] models. The performance of Hourglass backbone of these models was not a good option due to the incompatibility with SAM modules as well as authors [26] concluded ineffectiveness of anchor-free detectors with CSPDarknet-53 in their own studies. Thus, we concluded to use CSPDarknet-53 as the backbone and YOLOv3 as the detector for our studies and optimization.

As it is said above, we focused our work on studying and optimizing the neck part of the model, no backbone and head predictor were changed or modified. The backbone part is already optimized as much possible as well as predictor of YOLOV3 is considered as the optimal solution. Thus, we approached our method to the neck optimization. The neck part of the model is where the information propagates from backbone to the head predictor. In the neck part, we have a huge room for design and development.

The proposed method was aimed to lower the model complexity by eliminating the redundant parameters. We decreased the number of filters in the neck convolutional layers bu using the 1×1 kernel filter. By, this number of parameters significantly decreases, however this drop in parameters number should be compensated. To counteract this modification, we add SAM modules and introduced dilated convolutions to increase the efficiency of the feature maps. The SAM introduces attention mechanism, so that more relevant features are to be considered and dilated convolution increases the receptive field of the filter, which captures makes global information. We believe that decrease in number of learning parameters can be compensated, (preserved accuracy) by introducing the modules that encodes feature information more effectively. Please, refer to Chapter 4 for complete details regarding the proposed design.

Our model, called SAMD-YOLOv4 was then compared to the baseline YOLOv4 model on various evaluations. In Chapter 5, we provided a detailed review on those evaluation results for both models. We compared prediction precision and recall on MSCOCO dataset, models complexity, inference time and FPS with Darknet framework, inference time and FPS with TensorRT library. From results obtained, we can conclude that SAM-YOLOv4 obtained significant modifications, which preserved almost the same prediction precision and recall results on MSCOCO dataset while demonstrating substantially improved inference time and FPS metrics, with and without TensorRT technology. With results obtained, we have proven our initial thesis, redundant parameters were eliminated and neck structure was updated to compensate reduced model complexity.

However, we did not achieved several goals while developing the model design. Namely, we aimed to implement the CAM module in the neck structure to enable the channel-wise attention as well. We could not do that due to the issues with Darknet framework, we could not define custom CAM module in this framework. Also, the contrasstive loss functions was initially thought to replace the CIoU loss function. Again, this function has several implementation problems in the Darknet framework. In addition to this, as the main limitation of our model, the learning rate calibration during the model training is not straightforward. Due to changes introduced in the model architecture, the training gradients explode to the infinity if the learning rate is not dynamically tuned for particular epochs. The learning rate tuning is required for stable and effective model training.

For future research work, scaled-YOLOv4 [27] family models can be considered as the good starting baseline for further modifications. The scaling feature of this family ensures the easy and effective scaling mechanics for small, medium and large models. Authors in [27] used CSPNet not only in backbone but also in neck part of the model, which additionally eliminates model complexity. Also, the work done by [41] presents a promising real-time SOTA object detector architecture named as YOLOR. These are possible research direction in the field of real-time object detectors. Also, the SAM module is widely used in different CNN architectures, the research in attention mechanism can be substantially beneficial for the field of real-time object detectors.

Chapter 7 - Conclusion

To conclude, this thesis work aimed to research and optimize the SOTA real-time object detector named as YOLOv4, which is considered as the best real-time model. Numerous tests were accomplished with different backbones and head predictors both anchor-based and anchor-free. We then come with our main thesis, which defines optimization of the model by eliminating the redundant model learnable parameters and compensating it with improved attention-based and dilated convolutional blocks. According to the results achieved, our SAMD-YOLOv4 model experienced significant improvements in model complexity (7.3 % less parameters) and inference time (6.9% faster). Using the TensorRT library, the model performance rises even further. At the same time, we have faced various difficulties in the development and training stage, which were described in Chapter 6, by this we can conclude our work.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[2] M. Pietron and D. Zurek, "When deep learning models on GPU can be accelerated by taking advantage of unstructured sparsity," *CoRR*, vol. abs/2011.06295, 2020. [Online]. Available: https://arxiv.org/abs/2011.06295

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *ECCV*, 2016.

[5] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 446–454, 2017.

[6] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[7] R. B. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.

[8] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[9] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *ArXiv*, vol. abs/1804.02767, 2018.

[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2015.

[11] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *ArXiv*, vol. abs/1905.11946, 2019.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, pp. 211–252, 2015.

[13] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2009.

[14] F. O. Giuste and J. C. Vizcarra, “Cifar-10 image classification using feature ensembles,” *ArXiv*, vol. abs/2002.03846, 2020.

[15] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[16] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv*, vol. abs/1207.0580, 2012.

[17] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *NeurIPS*, 2018.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015.

[19] Z. Yao, Y. Cao, S. Zheng, G. Huang, and S. Lin, "Cross-iteration batch normalization," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12 326–12 335, 2021.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[21] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[22] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10 778–10 787, 2020.

[23] A. Sinha, S. Banerjee, and P. Chattopadhyay, "An improved deep learning approach for product recognition on racks in retail stores," 2022. [Online]. Available: https://arxiv.org/abs/2202.13081

[24] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," *CoRR*, vol. abs/2104.00298, 2021. [Online]. Available: https://arxiv.org/abs/2104.00298

[25] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, 2017.

[26] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *ArXiv*, vol. abs/2004.10934, 2020.

[27] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13 024–13 033, 2021.

[28] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1571–1580, 2020.

[29] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Convolutional block attention module," in *ECCV 2018*, 2018.

[30] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8759–8768, 2018.

[31] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017.

[32] G. Ghiasi, T.-Y. Lin, R. Pang, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7029–7038, 2019.

[33] Z. Huang and J. Wang, "DC-SPP-YOLO: dense connection and spatial pyramid pooling based YOLO for object detection," *CoRR*, vol. abs/1903.08589, 2019. [Online]. Available: http://arxiv.org/abs/1903.08589

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 1904–1916, 2015.

[35] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: exceeding YOLO series in 2021," *CoRR*, vol. abs/2107.08430, 2021. [Online]. Available: https://arxiv.org/abs/2107.08430

[36] M. Guo, T. Xu, J. Liu, Z. Liu, P. Jiang, T. Mu, S. Zhang, R. R. Martin, M. Cheng, and S. Hu, "Attention mechanisms in computer vision: A survey," *CoRR*, vol. abs/2111.07624, 2021. [Online]. Available: https://arxiv.org/abs/2111.07624

[37] NVIDIA, "Tensorrt," 2022. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

[38] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," *ArXiv*, vol. abs/1808.01244, 2018.

[39] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," *CoRR*, vol. abs/1904.08189, 2019. [Online]. Available: http://arxiv.org/abs/1904.08189

[40] Z. Dong, G. Li, Y. Liao, F. Wang, P. Ren, and C. Qian, “Centripetalnet: Pursuing high-quality keypoint pairs for object detection,” *CoRR*, vol. abs/2003.09119, 2020. [Online]. Available: https://arxiv.org/abs/2003.09119

[41] C. Wang, I. Yeh, and H. M. Liao, “You only learn one representation: Unified network for multiple tasks,” *CoRR*, vol. abs/2105.04206, 2021. [Online]. Available: https://arxiv.org/abs/2105.04206