# Improving the Energy Efficiency of Real-time DNN Object Detection via Compression, Transfer Learning, and Scale Prediction

Debojyoti Biswas
Dept. of Computer Science
Texas State University
San Marcos, TX, US

M M Mahabubur Rahman
Dept. of Computer Science
Texas State University
San Marcos, TX, US

Ziliang Zong
Dept. of Computer Science
Texas State University
San Marcos, TX, US

Jelena Tešić
Dept. of Computer Science
Texas State University
San Marcos, TX, US

***Abstract*— In recent years, computational accessibility has enabled the use of Deep Neural Network (DNN) for computer vision applications on devices with limited computational resources. We focus on the real-time object detection algorithms deployed on UAV-friendly devices. The hardware deployed on UAV must be lightweight and thus limited in processing power, memory, and storage capacity. Lightweight modeling architecture does not suffice for high-recall reconnaissance applications. In this paper, we propose to reduce power consumption of YOLOv5 DNN architecture. We decided to use compressed convolutional technique, transfer learning, backbone shrinkage, and scale prediction to reduce the number of learnable parameters from the YOLOv5model. Our approach reduced the size of the model significantly and lowered the power consumption in turn. GPU memory and the Billion Floating-Point Operations Per Second (GFLOPS) for the YOLOv5model will keep the performance measure of the model as the baseline state-of-the-art. The best resulting model has a 63.86% mean average precision (mAP) and a GFLOPS of 97.7 on "DIOR", an overhead imagery data set. The proposed approach has lowered GPU memory consumption of the model by 34% and lowered the energy consumption by 10 Watts compared to the baseline model.**

***Keywords— Energy Efficiency, YOLOv5, real-time object detection, Mobile Computing, UAV***

## I. INTRODUCTION

Overhead video sensors capture data at a rate far higher than current human-in-loop can effectively monitor and analyze. Real-time detection in overhead videos significantly reduces the volume of video data that needs to be transmitted and evaluated by humans on the ground for a range of applications: traffic management and control on land and sea, disaster control (floods, fires, earthquakes), surveillance and recovery missions in the mountains or at sea, wild-life migration patterns, object localization and detection, etc. Large video stream transmissions from drones and satellites to Earth-bound computing hardware pose a burden for the wireless communication channels and introduce latency. It takes considerable time and attention for a human-in-loop to evaluate and validate the intelligence gathering in video streams. Mission-critical real-time applications, such as fire-rescue operation and traffic monitoring systems, do not tolerate latency.

The solution is to deploy state-of-the-art deep neural network (DNN) real-time object detection algorithms on edge and mobile devices. The Convolutional Neural Network (CNN) architectures are used as a backbone in complex DNN architectures for reliable feature extraction, e.g., ResNet101 [1], Darknet [2], and MobileNets [3]. DNN architectures with CNN backbones are the most successful for computer vision tasks because of their nature of preserving information from images. Overhead cameras typically capture large images at high resolution; for example, the image dimension in the DIOR [4], DOTA2.0 [5] data set is 20,000 pixels by 20,000 pixels. The large size of the images increases the number of learnable parameters and the amount of computational complexity in the DNN pipeline. Recent research has proposed lighter models that use less computational power and have been deployed on edge computing devices [6], [7]. Large high-resolution images contain a lot of objects. The size of the objects is much smaller than the size of the image, and the number of local features is also small, making the overhead object detection notoriously difficult; therefore, the performance of Retina-Net or YOLO-ReT models significantly degrades. This problem can be solved with a light model with less computational power and energy that can (1) fit on edge devices; and (2) detect small objects with the same effectiveness level as a full DNN model.

We selected YOLOv5 [24] as a baseline architecture due to its ability to detect small objects in large images [8], [9]. We used the bottleneck convolution architecture to consume less energy than the original version and employed compressed convolutional technique and backbone shrinkage to make large CNN blocks smaller. Thus, we reduced the number of matrix operations, which in turn reduced the number of GFLOPS and made the model smaller and more energy efficient. Moreover, we utilized contextual information on the size distribution of the objects (Fig. 1) to make a scale prediction where object size was below 200px. We designed the transfer, learning and freezing the partial network approach to prevent model degradation and achieve up to 34% reduction in GPU memory and power consumption. The remainder of the paper is organized as follows: Section II contains related work, Section III discusses the data set, Section IV is the methodology, Section V includes the results and discussions, and Section VI concludes the findings.

## II. RELATED WORK

The DNN architectures for the object detection approaches originate from the two schools: Region Proposal Network (RPN) [10] and You Only Look Once (YOLO) [11]. RPN-reliant DNN architectures rely on CNN backbones to extract deep features from input images, e.g. [7]. Next, a small DNN, the Region Proposal Network (RPN), generates and filters object regions and feeds them into the detection network layers [12]. RPN-based architectures require more computational power due to the sequential processing of features, regions, and classes.

YOLO-based detectors parallelize the process and consider object detection as a regression task with respect to spatially separated bounding boxes associated with class probabilities [2]. YOLO9000 predicts the coordinates of the bounding boxes, directly using fully connected layers on top of the convolutional feature extractor [13]. YOLOv3 adds multiple convolutional layers to generate multiscale feature

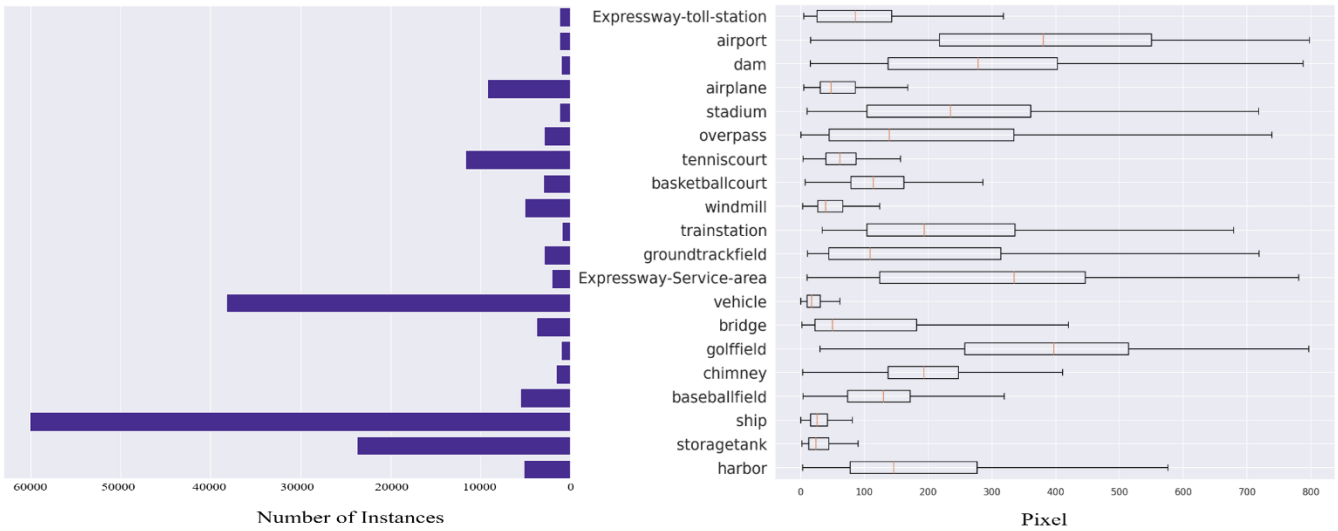


Fig.1. DIOR dataset analysis: (a) number of class instances and (b) object pixel size distribution in an image dataset.

maps and predicts boundary boxes using dimension clusters [1]. The incremental improvement of YOLOv3 is YOLOv4, which adds a couple of new features such as Weighted-Residual-Connections (WRC), Cross-Stage-Partial-Connections (CSP), and Self-adversarial-training (SAT). All these new architectures help the YOLOv4 to gain the state-of-art result in large scale datasets. The latest YOLOv5 adds more bags-of-tricks, and, with the addition of Faster Spatial Pyramid Pooling (SPPF), increases respective fields in the backbone features with almost no overhead in the detection operation speed. The TPH-YOLOv5 modifies YOLOv5 using the transformer prediction head to obtain a diverse range of features from the CNN input image and the attention mechanism for feature extraction [9]. SSD uses low-level convolutional feature maps from high-resolution images to detect small objects [14]. All these models are 100MB+ in size and range from 100MB to 372MB depending on the complexity of the networks. The models also require high-memory GPU workstations. The GPU memory requirement for most of these models is 8GB when trained with aerial images [8]with a batch size of 2 and the required RAM is 32GB.

Overhead large high-resolution images contain a lot of objects. The size of the objects is much smaller than the size of the image, and the number of local features is small. The stochastic region proposal attempted to address these shortcomings without reducing the size of the model. The stochastic method [15]proposes regions using the probabilistic goodness score of each region based on the size, shape, and recognition score of each object. Furthermore, Berat et al. proposed SyNet [16]which combines the features of signal stage and multistage detectors. The goal of combining two different types of models is to decrease the high false-negative rate from the multistage detector and increase the region proposal quality from the single stage detectors. This network is considered a popular aerial/overhead image dataset called “Visdrone” for model evaluation. For the baseline set, CenterNet and Mask RCNN are used in “SyNet”.

Researchers have reduced the size of the DNN models for object detection tasks. Region Proposal Networks (RPN) in traditional two-stage detectors become increasingly computationally expensive for overhead imagery. As the number of small objects and potential region candidates rises, the number of Non-Max Suppression (NMS) operations exponentially increases. The Center-Net [17] is based on the key points in the image, and the center of each object in the ground truth is considered as a key point. Model size and inference savings come from the CenterNet use of the heatmap for region proposal,,not RPN.

Researchers have lessened the GPU memory and power consumption by reducing the load of the heaviest layer in the DNN architectures. The lightweight version of RetinaNet [18]was able to reduce noticeable numbers of learnable parameters and GFLOPS as compared to the RetinaNet [19]. RetinaNet uses FPN for different scales of features from the input images followed by different shapes and aspect ratios of anchor boxes to create regions of interest. The anchor boxes are placed on each cell of the FPN output features and IOU is calculated with the ground-truth for prediction. The reduction analysis in this work [18] is based on GFLOPS and compares the performance of different models on the MSCOCO dataset in terms of mAP and GFLOPS, without mentioning the power consumption metrics and the memory usage.

CNN compression is one of the most prominent techniques for DNN model size reduction. Up to 87% of the operations in DNNs are convolution operations. Reducing the convolution operations from the network can reduce the number of learnable parameters as well as the GFLOPS. One of the first works towards CNN compression is SQUEEZENET [20]. The idea behind CNN compression is that a smaller filter size would require a smaller number of convolution operations as well as matrix storage space in the GPU. SQUEEZENET uses different approaches for CNN compression, such as reducing the filter size from 3x3 to 1x1, dropping the input channel size to 3x3, and downsampling the latter part of the network for a smaller number of activation functions. MobileNets achieves CNN compression by using model shrinking parameters, reduced input size, and transfer learning to build a smaller model [3]. This work [21] uses Parallel architecture, multi-GPU parallel processing, hierarchical on-chip storage organization, and the efficient application specific hardware designed to achieve efficient object detection. SKYNET is another hardware-efficient

neural network designed to deliver state-of-the-art detection accuracy and speed for embedded systems [22]. SKYNET provides a novel bottom-up DNN design approach taking hardware constraints into account.

## III. DATASET

The DIOR [4] overhead imagery dataset was released recently to address the lack of diversity in the properties of the image features in existing benchmarks. The dataset consists of 23,463 Google Earth images and 192,472 object instances. The size of the images in the dataset is 800 × 800 pixels, and the spatial resolutions range from 0.5 to 30 m. The objects are manually localized and classified into 20 classes of objects, as illustrated in Fig. 1(a). The DIOR dataset is specific in the following: (i) heterogeneous object categories, large variations of the pixel size of objects per category, and large variations of the number of instances per category, as illustrated in Fig. 1(a) ; (ii) large number of spatial resolutions and of inter- and intraclass size variability across objects that comes from the different orientations of the objects from the overhead view; (iii) large content variability over geographical area (80 countries), including weather conditions, seasons, and image quality; and (iv) high interclass similarity and intra-class diversity. The interclass similarity also arises by adding new classes in the dataset, such as "Bridge" vs. "Overpass" and "Bridge" vs. "Dam". On the other hand, the diversity was created by collecting images of different color, shape, and scale variations. All of these variations make this data set a great candidate, as the data set depicts the type of imagery the real-time object detection architecture will process on board the UAV.

## IV. METHODOLOGY

We improved the state-of-the-art DNN modeling to achieve the following: (1) the performance of the model does not deteriorate in terms of precision and recall (i) for overhead imagery; (ii) when the number of objects per image is larger than 50 (256 proposals per image); and (iii) when the objects in the image are small (less than 10px); and (2) a lightweight model that can fit on mobile devices such as Jetson Nano and Jetson TX2 with limited GPU memories of 4GB and 8GB, respectively, and system storage with a minimum of 32GB. We addressed the challenges presented in the following order. The overhead imagery challenge has been resolved with more training data and an adjusted threshold [23], and we planned to use a similar approach for 1.i.

When the number of objects per image increased, the number of prediction tasks per image also increased. This presented a great challenge for RPN-based methods, as the number of regions to consider in the filtering stage increases, and the DNN network incurs more GPU memory and a higher number of GFLOPS (ranging from 180 to 320) in detection [24]. RPN based architectures proved to be a showstopper due to the model size and memory needed in the test phase. During our experiments, Faster RCNN with ResNet 101 backbone & ResNet 50 required around 8GB of GPU memory for 2 images per batch and 6.5GB of GPU memory from 2 images per batch, respectively, on the DIOR dataset. Also, the model size was approximately 332MB which is larger than most of the Single Stage Detectors.

The mAP of the experiments ranges from 0.53 to 0.58, depending on the number of epochs. Thus, we chose the YOLOv5 architecture as the baseline model to address the problems of 1.ii: the small batch size (only 2 images per batch), large model sizes, and higher GPU memory consumption. To address the variability of the pixel size of the objects in 1.iii, we separated the object classes in the DIOR dataset according to the Fig. 1(a) object size distribution. If the object size was above 200 pixels, then the object was categorized as a large object, and if it was below 200 pixels, it was categorized as a small object. This resulted in an evenly separated 10 large object classes and 10 small object classes as illustrated in Fig. 1(b).

### *A. **YOLOv5 Baseline Architecture***

The YoloV5 architecture model in Fig. 2 consists of Backbone, Neck, and Detection Head.

#### *1) **Backbone**:*

The Backbone extracts global and local features from the images. In aerial object detection, in-depth local feature extraction is very important due to the very large number of small objects in the images. DarkNet53 is the backbone architecture introduced in YOLOv4 [8], [12], and continues to be used in YOLOv5. The Darknet53 has Convolution Layers (ConV), Cross Stage Partial Network (CSP), Bottleneck Layers, and Faster Spatial Pyramid Pooling Layers (SPPF). The CONV architecture has four parameters (In channel, Output channel, Kernel Size, and Stride), and the sigmoid linear unit (SiLU) was used as an activation function. The partial cross stage network (CSP C3) has 2n parameters, n for the input channel and n for the Output channel, where n is the number of repetitions of the CSP C3 layer.

#### *2) **Neck:***

Neck predicts bounding boxes using high-level deep features from the backbone. The YOLOv5 neck is the Feature Pyramid Network (FPN) [25]. From Fig. 2, we can see that there are concatenations between earlier layers and the later layers. This kind of concatenation helps the network to preserve high-level feature information merging with low-level deep features. The FPN block consists of four different modules, CONV, Upsample, Concatenation, and C3 as illustrated in Fig. 2.

#### *3) **Detection Head:***

The Dectection Head is the final stage toward object prediction that makes multi-scale predictions by dividing the image in cell grids of sizes 8, 16, and 32. Cell size 8x8 is tuned for small object detection while cell size 32x32 pixels aids large object detection. The first prediction is made from layer 17. The output of this layer is of shape [256, na(nc+5), 1,1]. Here, na & nc are the number of anchors and the number of classes in the dataset. 5 means the object-ness score, X, Y, W, H. Second and third scale predictions are made from the layer 20 with 512 output channels and from layer 23 with 1024 output channels.

#### *4) **Loss Function:***

Loss function in YOLO detectors need to balance positive and negative examples fed to the detection head. Most of the time the negative/ background examples are dominated by the positive examples. To solve this problem, YOLOv5 uses focal loss [19] instead of Cross Entropy (CE) loss in class prediction. Focal Loss improves the imbalance problem by down-weighting the easy examples and reducing the loss from easy examples close to zero. This way it gives more focus on the hard examples. In Fig. 4 we can see that all losses with ground truth (GT) probability greater than 0.5 are turned to zero. However, the blue line with = 0 represents the CE

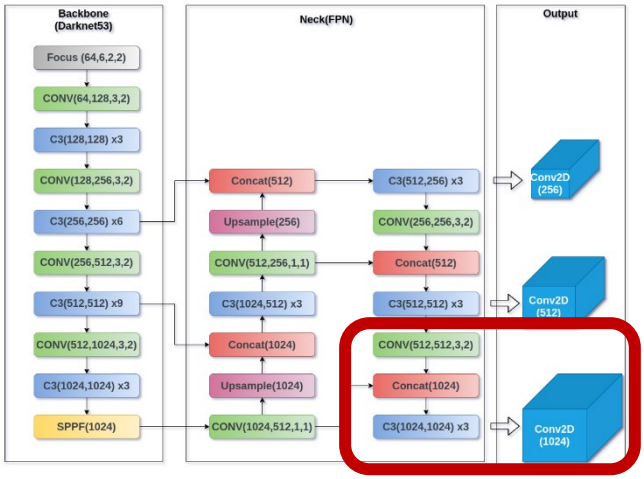


Fig. 2. YOLOv5 backbone architecture. Layers in the red shape are removed in the first phase of the proposed improvement

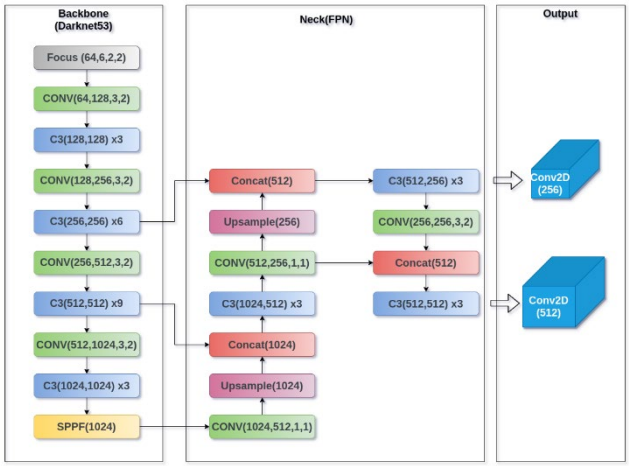


Fig. 3. Small and Medium adjustment.

Loss. The graph Fig. 4 shows that, when considering CE loss, even GT Probability of 0.9 is contributing to the total loss, thus making the loss function biased toward easy and negative examples.

### B. *Implemented Improvements for YOLOv5 Architecture*

#### 1) *Backbone:*

The CSP is used in the backbone to reduce the size of the images, as well as the channels. We experimented with our data set and found that our modified CSP network can reduce the noticeable number of GFLOPS from the network. Our first finding is that increasing the number of repetitions n (6 & 9) respectively in the Backbone C3 layers does not help improve the performance after 27 epochs. The loss curve for the object class and the box object becomes flat, and the mAP did not increase significantly. We modified the architecture of the CSP C3 layer by down-sampling the inner convolution operation by increasing the stride size from 1 to 2 at layer 1, 3, & 5. We also applied CNN compression by reducing the kernel size from 3x3 to 2x2 in C3 layers because C3 (Fig. 2) layers are concatenated with later layers in the detection head and fpn. Therefore, the downsample will not lose significant information due to channel-wise concatenation. Further, to make the model more lightweight, we reduced the number of repetitions of the CSP network from 6 to 4 (Fig. 6) and 9 to 6 (Fig. 5) at layer 4 and 6 respectively.

#### 2) *Neck:*

Here in our modified architecture (Fig. 2) we have four concatenations. The first two concatenations were done with C3 bottleneck architecture Layers 4 and 6 from Layer 10 and 14 (Fig. 2) as we had several numbers of repetitions with rich feature information. Then we had two concatenations at layers 19 and 22 with Conv layers 12 and 16 (Fig. 2). Here, the concatenations were done channel-wise, which increased the number of channels in the object features. Also, an increase in the number of channels would increase the number of GFLOPS in the model. So, after each concatenation, the C3 bottleneck was used to reduce the number of channels with the help of a 1x1 convolution. Therefore, the above modifications of the backbone and neck stage led to our first version of lighter model, the "YOLOv5 lighter".

#### 3) *Detection Head:*

We reduced the number of scale predictions for the detection head and removed the large scale prediction Fig. 2 (elements in the red box) as our specific application focused on real-time small object detection from UAV. This new architecture in Fig. 3 reduced the number of GFLOPS used at the cost of the decreased mAP for large-sized object classes. Large object detection from satellite imagery was a solved problem, and our focus was the light architecture for real-time traffic and disaster monitoring systems and fast inference times. This reduced scaled prediction version is named "Small and mid scale" and will be referred to with this name in subsequent sections and discussions.

#### 4) *Loss Function:*

In our research we used the orbital focal loss as used in the "YOLOv5 baseline" model. But we tackled the problem of an unbalanced easy-hard example using a weighted over sampler during the training. We gave more weight to small-size object classes Fig. 1(b) such as vehicle, ship and storage-tank, etc. to have more examples of these classes in each batch of the dataset. This helped the model to focus more on these classes during the training and gain better accuracy during inference.

As our final model, we assembled all the small improvements and applied transfer learning using a pre-trained weight from

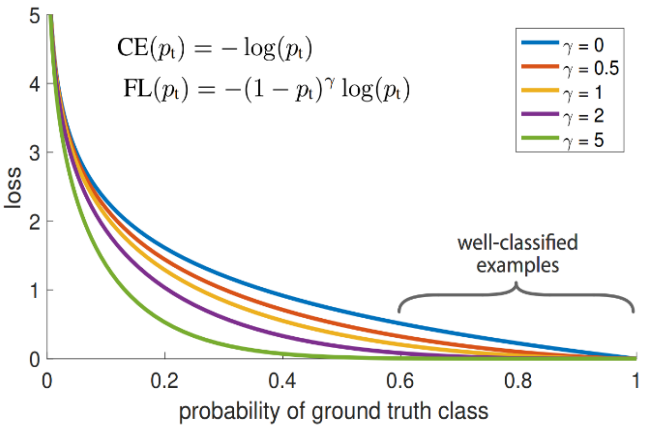


Fig. 4. Loss Function Analysis[19]

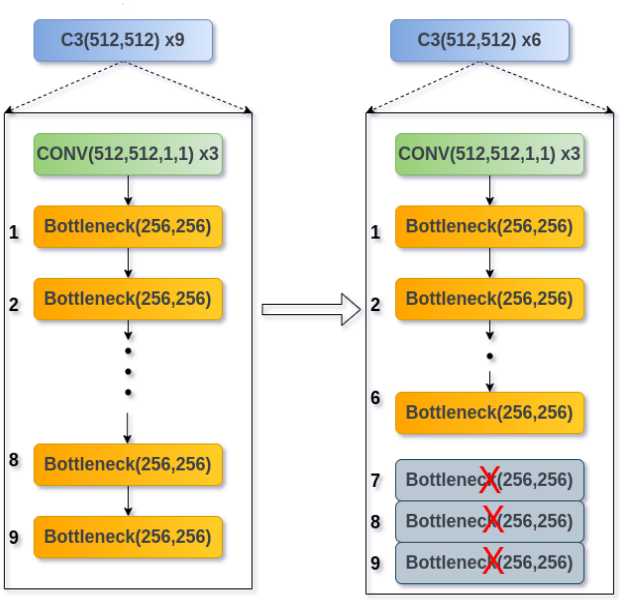


Fig. 5. Medium Scale CSP C3 architecture with Backbone Truncate

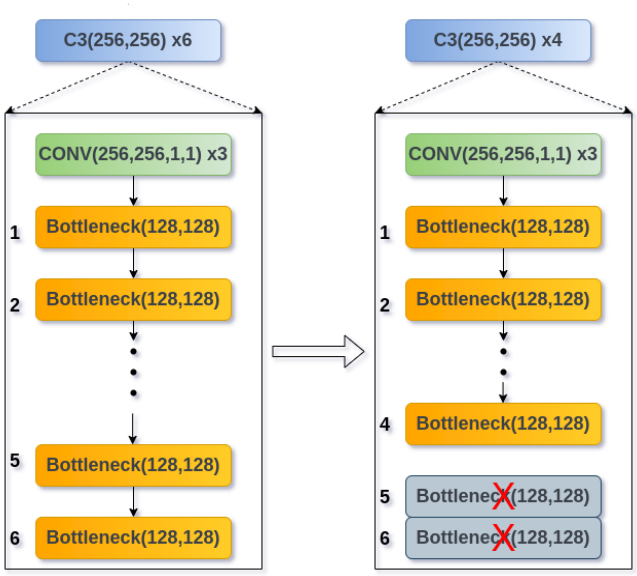


Fig. 6. Small Scale CSP C3 architecture with Backbone Truncate

a longer run (50 epochs) on the DIOR dataset. Additionally, we stopped gradient updates from the first 6 layers of the backbone to reduce GPU memory and power consumption (Table 3). From Fig. 2 we saw that the first two concatenation layers were from earlier backbone (before the 6th layer), so we decided to freeze first 6 layers. We found that the performance of the model did not decrease (Fig. 10) due to the backbone freezing. We name this version of our model "YOLOv5 freezed pretrained".

## V. Results and Discussions

The DIOR training set contains 22,450 images and validation set contains 1012 images. For preparing our baseline model, we have used tph-YoloV5 [9] GitHub repository and all experiments were conducted using the system specification from Table 1. For the mAP calculation, YOLOv5 always looked for the center of the object in the grid cell. If a grid cell contained any object center, then YOLOv5 matched the ground truth with all available anchor boxes and calculated the width-height IOU between them. The anchor with highest IOU was assigned to that grid cell. In this way, each grid cell was attached to a specific anchor box which was later used to regress the bounding box for the object. In this experiment, we generated anchor boxes using K-means clustering on the annotation of the training dataset. K-means clustering tries to cluster all data points in different clusters and provide the best possible set of anchor sizes and aspect ratios that covers most of the data points. The resulting 3D anchor boxes for this dataset:

[10,13, 15,28, 34,22] for Small Objects

[30,61, 60,43, 59,122] for Medium Objects

[114,91, 159, 208, 378,322] for Large Objects

Above, each row of anchor boxes contains 3 different anchor boxes with 3 different aspect ratios: 3 different scales and 3 different aspect ratios of anchor boxes, totaling 9 anchor boxes for each cell in the pooled feature matrix.

### A. *Performance measure analysis*

We compared the five different models in terms of Precision, Recall, mAP_0.5 and mAP_0.5:0.95 metrics. The model was trained using the DIOR training set and we froze the model's gradients update after every epoch. The model performance at every epoch was evaluated on the DIOR validation set, as illustrated Fig. 7 to 10.

Precision is the fraction of relevant occurrences among recovered instances (also known as positive predictive value). It can be defined as TP/(TP+FP). Fig. 7 shows the overall precision of the four different models. The "YOLOv5 lighter" and "YOLOv5 freezed pretrained" models outperformed the baseline model in terms of precision (Table 2). However, the precision dropped for the "Small and mid scale" models because it did not perform well for the large objects. The precision of "Small and mid scale" can be improved by using pretrained weights shown in Fig. 7 ("Small and mid scale freezed pretrained").

Recall is the fraction of relevant occurrences among all instances. It can be defined as TP/(TP+FN). Fig. 8 shows the overall recall for the five different models. Our "YOLOv5 freezed pretrained" model outperformed the baseline model in terms of recall Fig. 8. For the other three models, the recall dropped by a small margin.

mAP_0.5 is a metric that indicates the average precision (AP) where Intersection Over Union (IOU) is greater than

Table 1. System Specifications

| System | Configuration |
|---|---|
| **Operating System** | Ubuntu 18.04 |
| **CPU** | 11th Gen Intel® Core™ i9-11900K @ 3.50GHz × 16 |
| **GPU** | NVIDIA Corporation GP102 [TITAN Xp] |
| **GPU Memory** | 12GB |
| **RAM** | 125GB |

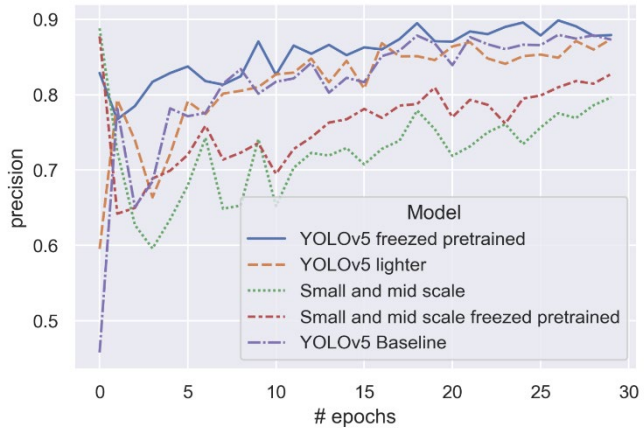


Fig. 7. Precision wrt # Epochs for different models

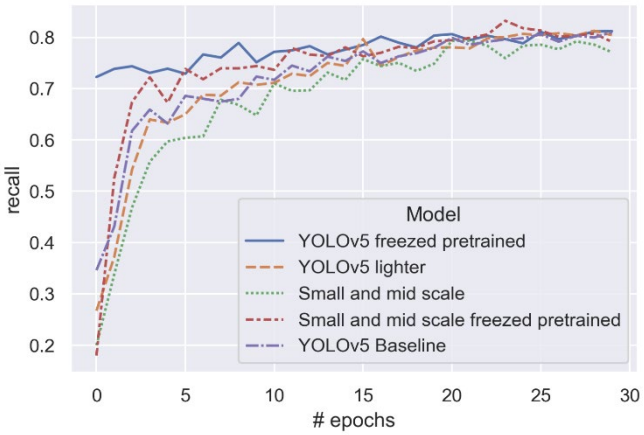


Fig. 8. Recall wrt # Epochs for different models

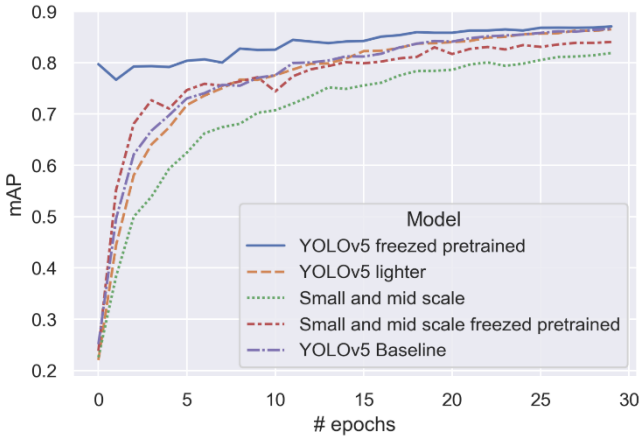


Fig. 9. mAP 0.5 wrt # Epochs for different models

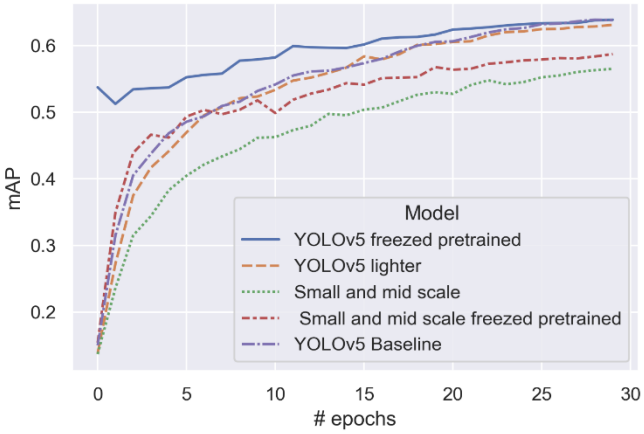


Fig. 10. mAP 0.5:0.95 wrt # Epochs for different models

50%. Fig. 9 shows the comparison mAP_0.5 among the five different models. Here, the `YOLOv5 freezed pretrained outperformed the baseline model (Table 2). The mAP_0.5 dropped by a small margin for the lighter YOLOv5 model. However, the "Small and mid-scale" and "Small and mid-scale freezed pretrained" models did not perform well in terms of mAP_0.5.

Table 2 Performance Summary

| Model name | Precision | Recall | mAP_0.5 | mAP_0.5:0.95 | Inference Time (ms) |
|---|---|---|---|---|---|
| **YOLOv5(Baseline)** | 0.8731 | 0.8098 | 0.8662 | 0.6381 | 76.2 |
| **Small and mid-scale** | 0.7966 | 0.7857 | 0.8188 | 0.5649 | **68.21** |
| **YOLOv5 lighter** | 0.8740 | 0.8041 | 0.8651 | 0.6310 | 69.6 |
| **YOLOv5 freezed pretrained** | **0.8792** | **0.8118** | **0.8706** | **0.6386** | 70.3 |
| **Small and mid-scale freezed pretrained** | 0.8274 | 0.7911 | 0.8403 | 0.5871 | 69.13 |

mAP_0.5:0.95 metric indicates the AP for IOU from 0.5 to 0.95 with a step size of 0.05. Fig. 10 shows the mAP_0.5:0.95 for the five different models. We were able to maintain quite similar mAP_0.5:0.95 as the baseline model for the "YOLOv5 lighter" and "YOLOv5 freezed pretrained" models (Fig. 10). However, the mAP_0.5:0.95 dropped for the "Small and mid-scale" and "Small and mid-scale freezed pretrained" models because of their poor performance on large object classes.

Table 3 shows inference time for proposed models matches most of the real time state-of-art object detectors. Reducing the GFLOPS and repetitions in C3 module (Fig. 5, 6) decreases inference time in the YOLOv5 lighter model by 7ms. Reducing. Among them the "Small and mid-scale" model is the fastest model with 68.21ms of inference time per frame (Table 3). Lower inference time leads to faster detection in real time, which is the key to mission critical applications.

### *B. Power Efficiency Analysis*

We used a physical device P4400 p3 Kill A Watt meter to measure the power consumption of models, as shown in Fig. 11. Using this device, we measured the power consumption of the CPU line. We conducted all the experiments for power efficiency analysis with a similar setup. First, the performance of the "YOLOv5 baseline" model was very satisfactory (Table 2 & 3). It achieved a mAP of 63.81% and recalled around 81%. The number of learnable parameters, GFLPOS, GPU Memory Consumption and Power Consumption were 46240609, 108.3 GFLOPs, 3.83 GB, and 384 Watt, respectively.


Fig. 11. P4400 p3 Kill-A-Watt meter for measuring power usage of each model during training.

Second, for the "Small and mid-scale" model, we removed the large-scale prediction from the detection network (Fig. 3). In this way, we were able to lower the power consumption by 10 watts (Table 3). Also, we reduced 10 GFLPOs. This model used 3.48GB of GPU memory, which was 0.35 GB lower than the baseline model. Furthermore, we were able to reduce the number of learnable parameters to 33,831,702. This model performed better than the baseline model in terms of load reduction and power consumption.

Third, for our "YOLOv5 lighter" model, the number of learnable parameters, GFLPOS, GPU Memory Consumption, and Power Consumption were 43942753, 97.7 GFLOPs, 3.63 GB, and 380 Watt, respectively. We were able to lower the number of learnable parameters, GLOPS by 10.6, GPU memory usage by 0.20 GB, and power consumption by 4 watts. Moreover, this model achieved state-of-the-art accuracy.

Table 3 Analysis of power efficiency

| Model name | GFLOPs | Power(watt) | Number of parameters | GPU memory Usage(GB) |
|---|---|---|---|---|
| **YOLOv5 Baseline** | 108.3 | 384 | 46,240,609 | 3.83 |
| **YOLOv5 lighter** | **97.7** | 380 | 43,942,753 | 3.63 |
| **Small and mid-scale** | 98.3 | 374 | **33,831,702** | 3.48 |
| **YOLOv5freezed pretrained** | **97.7** | 377 | 43,942,753 | 2.54 |
| **Small and mid-scale freezed pretrained** | 98.3 | **372** | 43,942,753 | **2.42** |

Fourth, in our "YOLOv5 freezed pretrained" model, we used the configuration from the "YOLOv5 lighter" model and then used Transfer Learning and Layer freeze method to reduce the GPU usage further. We used one of our pre-trained models and reused the weights from the model. Then we froze the first 6 layers of the backbone network. This experiment gave us the best overall result in terms of GPU memory usage as well as performance metrics. We were able to reduce GPU memory usage by 1.29 GB. The number of learnable parameters, GFLPOS, GPU Memory Consumption and Power Consumption for this model were 43942753, 97.7, 2.54GB and 377 Watt, respectively. This model used 7 watts less power and had 10.6 less GFLOPs than the baseline model. Finally, applying Transfer Learning and Layer freeze method on the "Small and mid scale" model, we further reduced the GPU memory usage and power consumption. Our "Small and mid-scale freezed pretrained" model used 12 watts less power, 1.41 GB less GPU memory and had 10 GFLOPS less than the baseline model.

In summary, our "YOLOv5 freezed pretrained" and "YOLOv5 lighter" models use less power and GPU memory and achieve the state-of-the-art accuracy. Therefore, these two models are suitable for real-time object detection in resource-constrained edge devices. Moreover, the "Small and mid-scale" and "Small and mid-scale freezed pretrained" models are suitable for application specific tasks for small object detection such as traffic monitoring. From Table 3, we can see that our best model, the "YOLOv5 freezed pretrained" model, takes a small amount of GPU memory (2.54 GB) with an mAP of 63.86 (see Fig. 10) on the DIOR dataset, which is slightly better than the YOLOv5 baseline model produces (see Table 2).

## VI. CONCLUSION

DNN architectures for the real-time object detection from overhead imagery are hard to scale to edge and mobile processing devices due to rapid deterioration in their performance when the number of objects in the image is large, the image itself is large, and the sizes of objects vary. Even the best models do not perform well with this dataset. We have proposed a lighter model, which takes advantage of the assumptions in the DNN architecture that are not applicable for the overhead imagery. Our proposed model meets full-model mAP and consumes 34% less power and GPU memory.

## ACKNOWLEDGMENT

This work is partially supported by the NAVAIR SBIR N68335-18-C-0199 and NSF grant CNS-1908658. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense, National Science Foundation, or the U.S. Government paper references. The Titan Xp used for this research was donated by the NVIDIA Corporation.

## REFERENCES

[1] J. W. Yun, "Deep Residual Learning for Image Recognition arXiv:1512.03385v1," Enzyme and Microbial Technology, vol. 19, no. 2, pp. 107–117, 1996, [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

[2] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.02767

[3] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017, [Online]. Available: http://arxiv.org/abs/1704.04861

[4] K. Li, G. Wan, G. Cheng, L. Meng, and J. Han, "Object detection in optical remote sensing images: A survey and a new benchmark,"

ISPRS Journal of Photogrammetry and Remote Sensing, vol. 159, pp. 296–307, Jan. 2020, doi: 10.1016/j.isprsjprs.2019.11.023.

[5] J. Ding et al., "Object Detection in Aerial Images: A Large-Scale Benchmark and Challenges," Feb. 2021, doi: 10.1109/TPAMI.2021.3117983.

[6] P. Ganesh, Y. Chen, Y. Yang, D. Chen, and M. Winslett, "YOLO-ReT: Towards High Accuracy Real-time Object Detection on Edge GPUs."

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation." [Online]. Available: http://arxiv.

[8] S. Ali, A. Siddique, H. F. Ates, and B. K. Gunturk, "Improved YOLOv4 for aerial object detection," Jun. 2021. doi: 10.1109/SIU53274.2021.9478027.

[9] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, "TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios."

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." [Online]. Available: https://github.com/

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.02640

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934

[13] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger." [Online]. Available: http://pjreddie.com/yolo9000/

[14] W. Liu et al., "SSD: Single Shot MultiBox Detector," Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.

[15] D. Yi, J. Su, and W. H. Chen, "Probabilistic faster R-CNN with stochastic region proposing: Towards object detection and recognition in remote sensing imagery," Neurocomputing, vol. 459, pp. 290–301, Oct. 2021, doi: 10.1016/j.neucom.2021.06.072.

[16] B. M. Albaba and S. Ozer, "Synet: An ensemble network for object detection in UAV images," in Proceedings - International Conference on Pattern Recognition, 2020, pp. 10227–10234. doi: 10.1109/ICPR48806.2021.9412847.

[17] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint Triplets for Object Detection." [Online]. Available: https://github.com/

[18] Y. Li and F. Ren, "Light-Weight RetinaNet for Object Detection," May 2019, [Online]. Available: http://arxiv.org/abs/1905.10011

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection."

[20] 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," Feb. 2016, [Online]. Available: http://arxiv.org/abs/1602.07360

[21] L. Li, S. Zhang, and J. Wu, "Efficient object detection framework and hardware architecture for remote sensing images," Remote Sensing, vol. 11, no. 20, Oct. 2019, doi: 10.3390/rs11202376.

[22] P. Graff, F. Feroz, M. P. Hobson, and A. Lasenby, "SKYNET: An efficient and robust neural network training tool for machine learning in astronomy," Monthly Notices of the Royal Astronomical Society, vol. 441, no. 2, pp. 1741–1759, 2014, doi: 10.1093/mnras/stu642.

[23] N. Warren, B. Garrard, E. Staudt, and J. Tesic, "Transfer learning of deep neural networks for visual collaborative maritime asset identification," in Proceedings - 4th IEEE International Conference on Collaboration and Internet Computing, CIC 2018, Nov. 2018, pp. 246–255. doi: 10.1109/CIC.2018.00041.

[24] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," May 2020, [Online]. Available: http://arxiv.org/abs/2005.12872

[25] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection." Writer's Handbook. Mill Valley, CA: University Science, 1989.