Evaluating the Performance of Training in YOLO Deep Learning Networks with Insignificantly Small Dataset

[1] T. Kavitha, [2] K. Lakshmi
[1] Research Scholar, [2] Professor, Computer Science and Engineering,
Periyar Maniammai Institute of Science & Technology, Thanjavur, India
[1] tkavitha07@yahoo.com, [2] lakshmi@pmu.edu

Article Info
Volume 82
Page Number: 9630 – 9637
Publication Issue:
January-February 2020

Abstract:
In the last few years, Deep Learning is the one of the top research areas in academia as well as in industry. Every industry is now looking for a deep learning-based solution to the problems in hand. As a researcher, learning "Deep Learning" through practical experiments will be a very challenging task. Particularly, training a deep learning network with huge amount of training data will make it impractical to do this on a normal desktop computer or laptop. Even a small-scale application in computer vision using deep learning techniques will require several days of training the deep network model on a very higher end GPU clusters or TPU clusters – that makes impractical to do that research on a conventional laptop.

In this work, we address the possibilities of training two versions of YOLO deep learning networks with an in significantly small dataset. Since we are going to design a prototype drone detection system with two different network models which are dealing with single class classification problem, we hereby try to train the deep learning networks only with few drone images (2 images only) and compare their performance in terms of mean average precision (mAP) and other suitable metrics.

The arrived results prove the possibility of training deep learning network with very few images (or any data). According to the results, YOLOv3 performed better than VOLOv2 and proves the possibility of training a deep learning network with insignificantly low number of images

Article History
Article Received: *18 May 2019*
Revised: *14 July 2019*
Accepted: *22 December 2019*
Publication: *12 February 2020*

Keywords - *Unmanned Aerial Vehicles, Computer Vision, Background Subtraction, Frame Differencing, Optical Flow, Edge detection, Convolutional Neural Networks.*

I. INTRODUCTION

Techniques for Learning from Small Data

There are a wide variety of problems that has very small datasets for learning. The connotation of "small dataset" depends on the application we work with. In this context, the small dataset means how much data required to give better results and how many classes are to be identified from the available data. N-Shot learning is one such technique which makes the learning effective with small datasets. This N-shot learning may be Few-shot or one-shot learning. In few-shot learning, the training examples are a few normally it lies between 0 and 5 examples. In one-shot learning, only one example is taken for training. A set of classes with adequate training is applied to one or more classes with few labelled examples. It solves the small dataset problem whereby improves the performance. Metric learning technique in which the discriminative features are learned by the network from a large dataset to generalize them for new classes. Another variant of few-shot learning is meta learning in which the examples trained from large datasets are used to learn from small datasets.

It is known that deep learning techniques require a huge amount of data for training the model and to improve its performance. But there are certain domains in which we cannot obtain data in enormous quantity because it involves high cost for collecting and annotating them. Due to the shortcoming of datasets, deep learning suffers a lot. To eradicate such hurdle, the above discussed techniques are used with small datasets. In such a situation, transfer learning is the better solution in which pre-trained models on similar tasks can be used for recognition.

Though it is useful, it is less likely suitable for medical imaging applications which brings a large bias between the source and target domain and for large imagery dataset where data are mostly collected from the web which may sometimes lead to copyright problems. But transfer learning is a technique which provides a solution with small datasets and is applied in many research areas using deep networks.

*Obstacles in Experimenting and Doing Research on Deep Learning [*aa]:*

Deep learning is the state-of-the-art technique and is rigorously used in many of the fields that require extensive computation. For example, it is very much used in autonomous self-driving car, medical imaging, image classification etc. The deep learning usage now spreads over in almost all applications where huge data is available such as speech recognition, health-care and medical diagnostics, and drug design. However, several challenges must be overcome before its wide-spread use. The common challenges are as follows:

i. Collecting large datasets for training – a large dataset can easily be collected from consumer applications but it still lags in data collection from some of the industrial applications.

ii. Requirement of expensive hardware such as GPU, TPU etc. for training the model – due to its high cost, it is not affordable to everyone who wish to carry out the research in deep learning.

iii. Identifying and fixing the value of the hyperparameters - it may sometime lead to overfitting of data.

iv. Understanding the insight of network is still a complicated one – due to its number of layers, nodes in each layer and connections, it is difficult to understand though it arrives at a good solution.

v. Maintaining a stable network – small inclusions or removals in the input shall lead to incorrect results. For example, the attacker may sometimes add noise to the data to make the model to give incorrect solution.

Challenges in constrained models are as follows:

- Mapping of input to output by a function – the amount of data required for approximating the unknown mapping function is still critical.
- Estimating the performance of the mapping function – it depends on the amount of data required.
- Poor approximation with the quantity of data taken to train the model for estimating the performance – overfitting or underfitting will occur based on the size of dataset. The performance of a model is estimated with test data which will give an optimistic and high variance [aa].

A. Hardware Requirements for Deep Learning Training

Even simple applications using deep networks cost heavily for availing the services provided by the third-party [bb]. The hardware required by deep networks for experimentation are as follows:

Central Processing Unit (CPU): This processor is able to perform different kinds of operations and memory transfers.

Graphical Processing Unit (GPU): This graphic device is otherwise known as graphics card. Video based memory transfers are carried out initially by GPU to reduce the burden of CPU. In the recent years, GPUs have a large processing power with the advent of gaming filed. GPU has a power of parallel processing where both CPU and GPU work together for processing and analyzing the data in an image or in any graphic form and even in scientific computation. For example, the processing power of GPUs are heavily used in video rendering, image transformations, and image compression etc. High Level Languages such as C, CUDA are used to write programs for graphical processing.

Tensor Processing Unit (TPU): This chip is specially designed to quicken the computations of the network. Like GPU, TPU also works with matrices. A TPU requires a constant flow of parameters.

TPU is a remarkable platform for training a deep network. TPU increases the performance 15 to 30 times over the performance with CPUs and GPUs.

The Google's TPU can only be accessed by TPU cloud services for training and for other computations. It cannot be purchased from vendors because it needs a dedicated infrastructure. But Coral edge TPU is the one available for public, has less capability hardware compared to Google's TPU cloud services.

Application Specific Integrated Circuit (ASIC) is a custom-made chip, specially designed for a fixed functionality. This is the only chip, there is no programming involved.

B. Data Requirements for Deep Learning

Data requirements for deep learning are considerably larger than other techniques for analysis [3].

The deep learning techniques help in extracting complex patterns from multimedia objects such as images, video, and audio [3] and are multidimensional data. To make this task effective, it requires a large labeled training data sets and access to relevant computing resources.

For complex modeling, deep-learning methods require vast data records in order to produce better classification results and, in some situations, the volume of data may exceed. By having 5000 labelled examples per class, it is possible to attain remarkable performance than human-level performance in one estimate with supervised deep network learning algorithm. In some cases where surplus of data available that is millions or even billions of rows per dataset, the usage of AI is perfect technique. However, if scarcity of data involves then deep networks are suitable [3].

In many businesses, the creation of massive datasets and labelling are difficult and challenging task. To overcome this data bottleneck, promising new techniques based on reinforcement learning, generative adversarial networks, transfer learning, few-shot and one-shot learning are evolving gradually. In these techniques, a trained model is allowed to learn from the subject with the help of a small number of real-world examples [3].

In addition, While modeling, we should be cautious about overfitting - this happens with a large dataset where the model closely matches the random features that results in poor accuracy and underfitting - this happens with a small dataset where the model fails to capture all relevant features [3].

II. Modeling

Components of the Learning Algorithm

To train a deep learning network model, we should choose

- a number of components such as nodes and layers etc., and hyperparameters such as weights, loss functions, epochs etc.
- an error function which is otherwise called as the cost or the loss or the objective function
- a specific framework for arriving an inference with maximum likelihood

Under this chosen framework, cross entropy and mean squared error loss functions are chosen for classification and regression problems respectively.

Loss Function

A set of weights on the training examples estimates the performance of the network model. To minimize the error, the optimization process needs an initial point from which the model begins its updates by repeated learning. The starting point is defined by the primary model parameters called weights. The starting point must be carefully chosen so that the optimization algorithm will improve the model performance by reducing the error surface. Though many weight initialization methods are available to choose initial model weights, small random values are normally chosen as initial weight.

Weight Initialization.

To optimize the network model, we have to choose the initial weight from which the model starts its learning. Weight initialization should be properly done. A procedure by which a model weight is assigned with an initial small random value at the beginning of the training process. The model error or loss is calculated while updating the model from a number of training examples in the training dataset. During training, it is possible to use either all the training examples in the training dataset when smaller dataset is used or a single example where the data changes frequently or the examples are streamed. In hybrid approach, to estimate the error gradient, the number of examples from the training dataset are chosen and these examples describes the batch size.

Batch Size

It is a hyperparameter of gradient descent that defines the number of training examples from the training dataset to work through before updating the internal parameters of the network model. Once the batch size is assigned, training is done iteratively and compares the predicted value with an expected output value and correspondingly the error is calculated. The calculated error is reduced by repeated updates in the model parameters to optimize the performance.

We can split the training dataset into one or more batches. It can be classified as follows:

i. When all the training examples are kept in one batch, the learning algorithm is called batch gradient descent.
ii. When the batch size is one, the learning algorithm is called stochastic gradient descent.
iii. When the batch size is greater than one example but less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

The most popular batch sizes for mini-batch gradient descent are 32, 64, and 128 examples.

Learning Rate

The rate at which the parameters in the learning algorithm is updated in every iteration in the training dataset. The fine-tuned model parameters are obtained by repeated training process. During training, the total number of iterations in training dataset depends on the complete number of passes which occurs before the training process terminates.

Epochs

The number of passes required before the training process completes.

The learning algorithm is greatly controlled by these five hyperparameters in deep neural networks.

III. Object Detection Using Deep YOLOV2 and YOLOV3

There are many different algorithms available for object detection and these algorithms are categorized into two types based on their processing ability.

The algorithms based on classification work in two steps. At first, the interesting regions of the image are selected then next the objects within those regions are classified using Convolutional Neural Networks (CNN). The algorithms under this category are too slow and are less suitable for real-time situations.Region based Convolutional Neural Network (R-CNN) is one such example of this category.

i. The algorithms based on regression, scan the entire image and predicts the objects in an image which includes localization, detection, and classification. These algorithms are faster and are suitable for real-time object detection. You Only Look Once (YOLO) follows regression based approach.

YOLO Object Detection

There is a progressive growth in deep learning networks. The recent development is You Only Look Once (YOLO) which is a network in which object detection is performed by deep learning algorithms. Object detection is done by classifying the objects present in an image after detecting and localizing it with bounding boxes.

YOLO differs from the previous region-based detection methods such as R-CNN, Fast R-CNN, and Faster R-CNN etc., which focus on a particular region in an image for detecting the objects by training each individual component separately.

This multi-step process is very time-consuming due to the region selection and it also uses selective algorithm for detection. Hence, there is no learning performed in detection. These methods are very slow and optimization is harder They are not suitable for real-time detection.

YOLO is faster and easier to optimize because the algorithm uses the network only once to run all the components of the task.

YOLO Architecture

The YOLO for object detection is clearly understood only after knowing its components thoroughly.

An algorithm, the network, and the loss functions are the three important components of YOLO network.

The YOLO Object Detection Algorithm

The computing process is divided into the following steps:

Step1: Split the input image into an S×S grid.

Step2: Predict B bounding boxes from each grid cell which encloses an object and the corresponding confidence scores to find out whether the predicted bounding box has an object. It is symbolized as $\boldsymbol{x, y, w, h, Pr(\,Object) \times IOU^{Gt}_{Pr}}$, where Pr*(Object)* is the probability that the current position is a valid object class, IOU is the overlap probability between the predicted(Pr) bounding box and the ground truth(Gt). The *x*, *y* is the center coordinate and the *w*, *h* is the size of the box.

Step 3: Calculate the conditional probability of prediction class in each grid $\boldsymbol{C_i = Pr(\,Class|Object)}$.

Step 4: Step 4: Multiply both the conditional probability of a class and confidence while predicting the object as

$$\Pr(Class_i \mid Object) \times \Pr(Object) \times IOU^{Gt}_{\Pr} = \Pr(Class_i) \times IOU^{Gt}_{\Pr}$$

When a large object is encountered, it is necessary to perform non-maximally suppressed operations. Since the bounding box B has a value of 2, a grid will return only two boxes which means that a grid will have only one category. If multiple categories present in a grid, there is a problem. YOLO gives good results for small objects.

The Network

YOLO network involves convolutional layers, max pool layers, and two fully connected CNN layers.

The Loss Function

Yolo uses sum-squared error for the loss function because it is easy to optimize. This function calculates the error equally for small and large boxes.

Since two bounding boxes are found in each of the grid cell, the loss function in YOLO is used to compute the loss for each true positive. The bounding box with highest Intersection of Union (IoU) with Ground Truth makes the error function effective.

Different Versions of YOLO

Each version of YOLO is only improving the accuracy in the performance of detecting objects. They have so far created three versions of YOLO. The current version of YOLO is YOLOv3.

Though we are not going to evaluate YOLOv1, we compare its architecture with other versions.

YOLO v1

YOLOv1 uses a limited darknet framework. There are many restrictions with this version. One such restriction was that YOLOv1 cannot identify a group of small objects say for example a flock of birds and was inefficient in generalizing objects with different dimensions other than the trained image. Hence, it led to poor localization of objects within the input image.

Table 1. Tiny YOLOv1

No.	Layer Name	Kernels	Kernel Size / Stride	Input Size	Output Size	BFLOPs
0	conv	16	3x3/1	448x448x3	448x448x16	0.173
1	max		2x2/2	448x448x16	224x224x16	0.003
2	conv	32	3x3/1	224x224x16	224x224x32	0.462
3	max		2x2/2	224x224x32	112x112x32	0.002
4	conv	64	3x3/1	112x112x32	112x112x64	0.462
5	max		2x2/2	112x112x64	56x56x64	0.001
6	conv	128	3x3/1	56x56x64	56x56x128	0.462
7	max		2x2/2	56x56x128	28x28x128	0.000
8	conv	256	3x3/1	28x28x128	28x28x256	0.462
9	max		2x2/2	28x28x256	14x14x256	0.000

10	conv	512	3x3/1	14x14x256	14x14x512	0.462
11	max		2x2/2	14x14x512	7x7x512	0.000
12	conv	1024	3x3/1	7x7x512	7x7x1024	0.462
13	conv	256	3x3/1	7x7x1024	7x7x256	0.231
14	connected			12544	539	
15	Detection					
					Total BFLOPs	3.185

YOLO v2

YOLOv2 is otherwise known as YOLO9000. YOLOv2 uses darknet-19 for object detection. YOLOv2 has been designed to show better object detection scores and this framework is used in Faster Region based CNN and Single Shot multi-box Detector (SSD).

Table 2. Tiny YOLOv2

No.	Layer Name	Kernels	Kernel Size / Stride	Input Size	Output Size	BFLOPs
0	conv	16	3x3/1	416x416x3	416x416x16	0.150
1	max		2x2/2	416x416x16	208x208x16	0.003
2	conv	32	3x3/1	208x208x16	208x208x32	0.399
3	max		2x2/2	208x208x32	104x104x32	0.001
4	conv	64	3x3/1	104x104x32	104x104x64	0.399
5	max		2x2/2	104x104x64	52x52x64	0.001
6	conv	128	3x3/1	52x52x64	52x52x128	0.399
7	max		2x2/2	52x52x128	26x26x128	0.000
8	conv	256	3x3/1	26x26x128	26x26x256	0.399
9	max		2x2/2	26x26x256	13x13x256	0.000
10	conv	512	3x3/1	13x13x256	13x13x512	0.399
11	max		2x2/1	13x13x512	13x13x512	0.000
12	conv	1024	3x3/1	13x13x512	13x13x1024	1.595
13	conv	512	3x3/1	13x13x1024	13x13x512	1.595
14	conv	30	1x1/1	13x13x512	13x13x30	0.005
15	Detection					
					Total BFLOPs	5.344

Comparison between YOLO v2 and YOLO v1

- ➢ YOLOv2 uses a high-resolution classifier so, mean Average Precision (mAP) has been improved in YOLOv2. The input size increases from 224x224 to 448x448 and improved the mAP to 2%.
- ➢ An image is divided into 13 x 13 grid to detect and localize smaller objects.
- ➢ It has improved the detection result of images with varying sizes once the training with images of different scales, is given to the algorithm.
- ➢ In YOLOv2, anchor boxes provides a single framework for both classification and prediction.

YOLOv3

YOLOv3 is an enhanced version of YOLOv2. It consists of totally 106 layers. Out of which, one set of 53 layers trained on ImageNet dataset and another set of 53 layers used for detecting the object. Its detection speed is reduced from 45 fps to 30 fps but it gives better accuracy than previous versions.

Comparison between YOLOv3 and YOLOv2

- ➢ Improved bounding box prediction: A confidence value is predicted for all the objects in a bounding box by logistic regression.
- ➢ More accurate class predictions: To label each class of objects, logistic classifiers are used.
- ➢ Improved abilities at different scales: three predictions are found for every location in an input image by up-sampling for getting fine-grained semantic information so that the quality of the output is improved.

Table 3. Tiny YOLOv3

No.	Layer Name	Kernels	Kernel Size / Stride	Input Size	Output Size	BFLOPs
0	conv	16	3x3/1	416x416x3	416x416x16	0.150
1	max		2x2/2	416x416x16	208x208x16	0.003
2	conv	32	3x3/1	208x208x16	208x208x32	0.399
3	max		2x2/2	208x208x32	104x104x32	0.001
4	conv	64	3x3/1	104x104x32	104x104x64	0.399
5	max		2x2/2	104x104x64	52x52x64	0.001
6	conv	128	3x3/1	52x52x64	52x52x128	0.399
7	max		2x2/2	52x52x128	26x26x128	0.000
8	conv	256	3x3/1	26x26x128	26x26x256	0.399
9	max		2x2/2	26x26x256	13x13x256	0.000
10	conv	512	3x3/1	13x13x256	13x13x512	0.399
11	max		2x2/1	13x13x512	13x13x512	0.000
12	conv	1024	3x3/1	13x13x512	13x13x1024	1.595
13	conv	256	1x1/1	13x13x1024	13x13x256	0.089
14	conv	512	3x3/1	13x13x256	13x13x512	0.399
15	conv	18	1x1/1	13x13x512	13x13x18	0.003
16	yolo					
17	route	13				
18	conv	128	1x1/1	13x13x256	13x13x128	0.011
19	upsample		2x13x1	3x128	26x26x128	
20	route	198				
21	conv	256	3x3/1	26x26x384	26x26x256	1.196
22	conv	18	1x1/1	26x26x256	26x26x18	0.006
23	yolo					
					Total BFLOPs	5.448

IV. Implementation Results and Discussion

The Small Dataset Used for Training.

In this work, we have used 2 images to form a tiny "2 image dataset" from [*ee] and used it for training. Along with the image, the ground truth bounding box information is available.

The deep learning network is trained with this information repeatedly by introducing the 2 images as training images.

Deep Network Design using Darknet

The open source framework Darknet is used to train the neural network which is written in C and CUDA. It serves as the basis for modeling YOLO deep network. The darknet framework is useful for real-time object detection. Darknet displays information when it loads the config file and weights, then it classifies the image and finally prints the classes for the image. Recurrent Neural Networks are powerful models for representing data that changes over time and Darknet can handle them without making use of CUDA or OpenCV.

In this work, we used Alexey's [13] implementation of Darknet which is a part of Joseph Redmon's original Darknet [14] implementation. Since it is based on C, and it contains features to use YOLO, we opted to use it as the best choice for designing our fast, real-time drone detection system.

The Experimental Setup

We used a Normal Core i7 Laptop with 16Gb RAM. We have not used any higher capability GPU and we only depend on the computing power of the CPU. We used 64bit version of Lubuntu 16.04 as the operating system on our laptop.

We used Alexey's implementation of Darknet from [11] and compiled. It uses the maximum CPU power of our Laptop.

While compiling Darknet, the following Make file parameters are used:

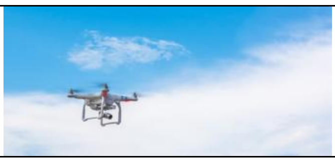
```
GPU=0
CUDNN=0
CUDNN_HALF=0
OPENCV=0
AVX=0
OPENMP=1
LIBSO=0
ZED_CAMERA=0
```

We have disabled GPU option and Enabled Multi-Processing option. So, our experiments can be repeated on any 64bit laptop without any other additional GPU/TPU hardware or any other computing resources.

Images Used for Training and Testing

Table 4. The Two Drone Images from [11] are Used for Training

Table 5. The 20 Different Size Drone Images from [11] are used for Testing.

About the Batch Size and Epochs

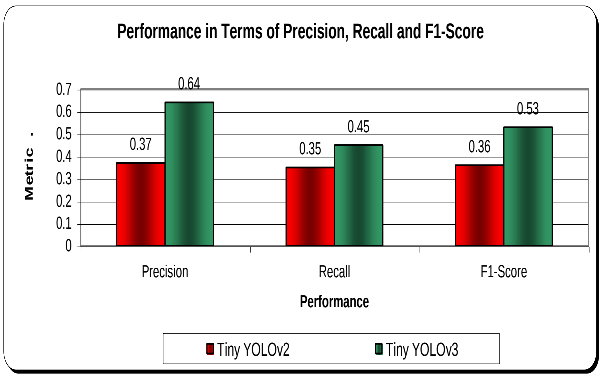
As far as these two-image small dataset is concerned, the Batch Size, subdivisions size and Epochs are too important for successful training.

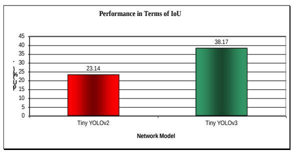
To make the training as a successful and progressive process, we used the batch size of 2 and subdivisions size of 1. It means that we train both the images in each epoch and repeated it for 700 epochs.

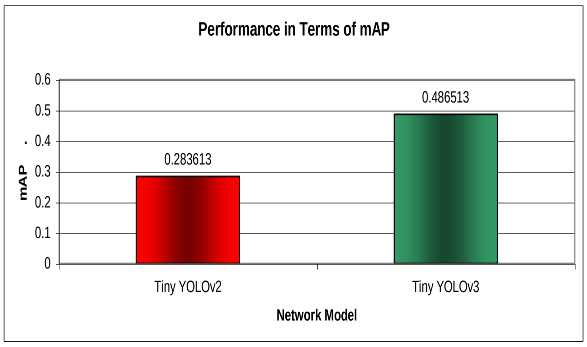
Number of Training Images	: 2
Number of Test Images	: 20
Number of Epochs of Training	: 800

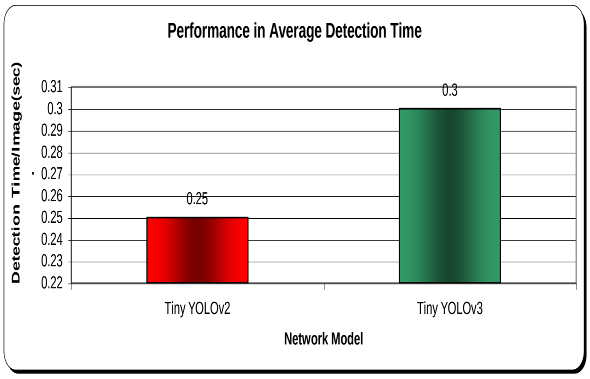
* We have trained both the models upto 800 epochs and saved the weight at each 100 epoch steps and used the weights which gave the highest performance in terms of mAP. In our case, we achieved the best performance at the 700th epochs in both the network models.

Model	Total Detection Time (sec)	Precision	Recall	F1-Score	average IoU (%)	Average Detection Time (sec)
Tiny YOLOv2	0.37	0.35	0.36	23.14	0.28	0.25
Tiny YOLOv3	0.54	0.35	0.42	34.79	0.40	0.3









Generally, deep learning network will consume several hours or days even weeks for getting some meaningful training to give high precision in detection even at the very higher end hardware such as GPU and TPU clouds. But the scope of this work is to complete the same kind of training process with in the limitations of a conventional CPU.

January - February 2020
ISSN: 0193 - 4120 Page No. 9630 - 9637

We observed that the training of YOLOv2 with two images consumed less than 30 minutes and YOLOv3 with two images consumed less than 20 minutes. So, we proved that it is possible to train a deep learning network within an hour and get some meaningful testing / detection performance in terms of mAP.

V. CONCLUSION

In this work, we successfully implemented a drone detection system with two different Tiny YOLO models and demonstrated the possibility of training a deep learning network with insignificantly small number of training images. In our experiment, we have used only 2 drone images to train the Tiny YOLOv1 and Tiny YOLOv2 network and achieved an acceptable detection performance. To prove the successful detection, we used a small test image set with 20 drone images and measured the performance with different metrics. As shown in the results of previous section, our work proves the possibility of training a deep learning images only with very few images per class. The achieved results in terms of different metrics shows the successful training with a few images.

Our results proved that any research on deep learning can be done on a normal computer without any sophisticated hardware such as GPU and TPU and giving solutions for the hardware related obstacles in front of Deep Learning Research. Generally, even with GPU and TPU clouds, the training process of deep learning will consume several days or even months since there will be huge dataset for training. In our work, we showed the possibility of training a huge deep learning network on a insignificant hardware (CPU) with a dataset of insignificant size.

We demonstrated that it is possible to train a deep learning network with in an hour and get some meaningful testing/detection performance in terms of mAP. So anyone who starts a deep learning research and trying to implement a complex deep learning system can really able to complete a simple prototype if they try to do the initial experiments with very low number of samples (images or any data).

REFERENCES

1. Jason Brownlee, “A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models, February 15, 2019 in Deep Learning Performance, machinelearningmastery.com
2. Jason Brownlee, “Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates”, January 2, 2019 in Deep Learning Performance, machinelearningmastery.com
3. Michael Chui et al., "Notes from the AI frontier: Insights from hundreds of use cases ,McKinsey Global Institute, Discussion Paper, April 2018
4. *aa Shriram Ramanathan, Senior analyst, artificial intelligence and big data analytics at Lux Research "Five Challenges for Deep Learning", An article at eetimes.com
5. *bb Colin Adams, "Expensive, Labour-Intensive, Time-Consuming: How Researchers Overcome Barriers in Machine Learning" , July, 2019, An Article at journal.binarydistrict.com
6. *cc Himanshu Singh, "Everything you Need to Know About Hardware Requirements for Machine Learning", An article at einfochips.com, Feb 2019
7. *dd https://missinglink.ai/guides/computer-vision/yolo-deep-learning-dont-think-twice/
8. Deep Cross-Domain Flying Object Classification for Robust UAV Detection, Arne Schumann, Lars Sommer, Johannes Klatte, Tobias Schuchert, Jurgen Beyerer, IEEE August 2017.
9. Object Motion Detection Based on Perceptual Edge Tracking, Gao Q, Parslow A,Tan M, IEEE 2016.
10. Unified, Real-Time Object Detection, Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, IEEE 2016.
11. Using Deep Networks for Drone Detection, Cemal Aker, Sinan Kalkan, IEEE July 2017.
12. FlowNet: Learning Optical Flow with Convolutional Networks, Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas and Vladimir Golkov, IEEE 2015.
13. Detection and Tracking of Moving Object Based on PTZ Camera, Xinghua Li, Qinglei Chen and Haiyang Chen, IEEE 2012.
14. Moving Object Detection and Locating Based on Region Shrinking Algorithm, Zhihui Li, Haibo Liu and Di Sun, IEEE 2012.
15. A Survey on Different Background Subtraction Method for Moving Object Detection, Rajkumari Bidyalakshmi Devi and Khumanthem Manglem, IEEE 2016.
16. A Study on Detecting Drones Using Deep Convolutional Neural Networks, Muhammad Saqib, abin Sharma, Sultan Daud Khan and Michael Blumenstein, IEEE 2017.
17. A Moving Objects Detection Algorithm in Video Sequence,Mingyang Yang, IEEE 2014
18. Christian Reiser, Bounding box detection of drones (small scale quadcopters) with CNTK Fast R-CNN, https://github.com/creiser/drone-detection
19. Chuan-en Lin, “drone-net”, https://github.com/chuanenlin/drone-net
20. Alexey, “Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection”, https://github.com/AlexeyAB/darknet
21. Joseph Redmon, "Darknet: Open Source Neural Networks in C", http://pjreddie.com/darknet/,2013-2016
22. Miasnikov, E., *Threat of Terrorism Using Unmanned Aerial*
Vehicles: Technical Aspects. Center for Arms Control, Energy
and Environmental Studies, Moscow Institute of Physics and
Technology, Moscow, 2015.
23. Humpreys, T., *Statement on the Security Threat Posed by Unmanned Aerial Systems and Possible Countermeasures*. Statement to the Subcommittee on Oversight and Management Efficiency of the House Committee on Homeland Security, 18 March 2015, Washington D.C.,2015.
24. Dinesh Sathyamoorthy, "A Review of Security Threats of Unmanned Aerial Vehicles and Mitigation Steps", Article Published at ResearchGate.net, October 2019.