International Journal of Future Generation Communication and Networking
Vol. 13, No. 4, (2020), pp. 2072–2086

Transfer Learning with Tiny YOLO-RV1 Deep Learning Network Model for Better Performance with Small Training Dataset

T.Kavitha[1], Dr.K.Lakshmi[2],
[1]*Research Scholar, Periyar Maniammai Institute of Science & Technology, Vallam, Thanjavur, India*
[2]*Professor, Periyar Maniammai Institute of Science & Technology, Vallam, Thanjavur, India*

Abstract

In a desktop computer or laptop, training a deep learning network with large datasets is very difficult. The deep learning network will take several days for training the model on a higher end GPUs or TPUs even for a small-scale application. Hence, it decreases the interest of doing research in deep learning area with normal desktop computer or laptop. The objective of this paper is to make a YOLO model can be trained on a normal computer with a descent CPU power.

In this paper, we trained the standard tiny YOLOv3 model and the network complexity reduced version tinyYOLORV1deep learning network model for better performance with small training dataset. The scope of this work is to improve the performance of training with small dataset using transfer learning. After training the standard tiny YOLOv3 and the proposed tiny YOLO-RV1 models by transfer learning with few drone images (2 images only), their performances are compared with standard tiny YOLOv3 and tiny YOLORV1 models without transfer learning.

The arrived results proved that the possibility of training the deep learning network with a very few samples of images or any kind of data through transfer leaning achieves better accuracy. Further, our proposed Tiny YOLO-RV1 achieved an ideal performance in terms of accuracy even with the very small training set by using transfer learning.

Keywords *- Unmanned Aerial Vehicles, Computer Vision, Convolutional Neural Network, YOLOv3, Transfer Learning*

I. INTRODUCTION

The deep learning network requires a large amount of data for its training which uses a cluster of CPUs and GPUs. Since it becomes a big hurdle to the researchers, the training may be done with small dataset. Today, many researchers work with such limited data from different directions. Some of the learning techniques available to work with limited data are as follows:

In the few shot and one-shot learning, the learning of a model with sufficient training data for a given set of classes is used to improve the performance of another and newer set of classes with a very few labelled data. In Metric learning techniques, the model learns from the highly discriminative features from a large dataset so that it generalizes well to new classes (Vinyals et al., 2016; Snell et al., 2017; Sung et al., 2018; Wang et al., 2018; Wu et al., 2018). The like It automatically constructs task-specific distance metrics from supervised data. Then, the learned distance metric can be used to perform various tasks such as k-NN classification, clustering, and information retrieval. K-Nearest Neighbor, K-means, and SVM machine learning algorithms use this metric to identify the distance or similarity between data instances. The metric being used determines the performances of these algorithms. Another approach to few-shot learning is meta-learning which is also known as learning to learn. In this learning, the model learns either new skills or adapt to new environments quickly with a few training examples (Li et al., 2017; Qiao et al., 2018). In Zero-Shot Learning, the machine learns by itself without training. For example, the machine will classify the unseen classes without training. The machine learns from the meta data of the images.

In the information era, there are huge amounts of data available in many domains. Hence, deep learning methods are very well being used to extract the features to understand the patterns. (Sun et al., 2017). Though it needs very powerful hardware components for processing, the deep learning approaches are widely used with these large datasets (Krizhevsky et al., 2012; Sun et al., 2017). However, there are certain domains and applications in which the collection of large volume of data is a big problem due to high costs involved in collecting and annotating of suitable data. To overcome

ISSN: 2233-7857 IJFGCN

this problem, transfer learning can be used. In this transfer learning, pre-trained models which are trained with a large amount of data such as ImageNet dataset (Deng et al., 2009) can be used for training with similar kind of problems (e.g., Zeiler & Fergus, 2014; Girshick et al., 2014).

When we use transfer learning, no additional costs involved for training the model. This will increase the number of research projects in academia. In some cases, the transfer learning is still problematic because the target domain might be highly specialized. For example, in medical image analysis (cf. Litjens et al., 2017), there may be biases between the source and target domain. Nevertheless, the majority of research applying deep learning to small datasets focuses on transfer learning.

The use of a small dataset is highly subjective and depends on the task at hand and the diversity of the data. In the work (CUB; Wah et al., 2011), datasets with less than 100 training images per class was considered as small, such as the Caltech-UCSD Birds dataset, which has 30 images at the maximum per class. The ImageNet LSVRC 2012 dataset (Russakovsky et al., 2015) contains between 700 and 1,300 images per class.

Challenges of Deep Learning Network [17]

Deep learning is one of the top-ranking techniques which are heavily used in applications such as healthcare, materials, energy, self-automation etc. It is also predominantly used in image recognition, personalization, demographic and election predictions, autonomous driving, automatic game playing, quality control in manufacturing and medical imaging. Now-a-days, the use of deep learning spread across the fields such as cybersecurity, drug design, voice recognition, chip design and optimizing manufacturing operations.

However, the challenges of deep learning must be overcome before adopting it for the task.

- Datasets: Availability and processing of massive dataset for training is a major problem. Though datasets are available either publicly or on demand basis, it is still a problem in collecting data for industrial applications.
- Training: Training the deep networks with massive amounts of data may use a big cluster of CPUs or GPUs or TPUs and may take several days or months to complete the training with the specified hardware. To reduce the training time, some promising techniques such as transfer learning or generative adversarial networks may be used.
- Parameters: The deep learning networks works well with a large number of interconnected neurons or free parameters in deep layers so that they detect fine-grained information from the data. However, it is difficult to identify hyperparameters before starts the training. It will be fixed by experience only.
- Overfitting: A network model with a small dataset cannot learn the problem well whereas a model with large dataset can learn the problem too good and overfit the training dataset. Hence, a model cannot generalize well. The danger of overfitting is reduced either by training the network on more examples or by changing the complexity of the network.
- Large deep learning networks are difficult to implement on the edge because of their sheer size. Deep learning networks are very hard to quickly retrain using newly available information. Pre-trained models can be used for retraining with new datasets.
- Number of layers, nodes, and connections are huge in deep learning network. Hence, it is difficult to understand the insight of its internal processing. Visualization techniques are available for rectifying this issue.

Challenges of supervised learning [2] include the amount of data needed to approximate the network mapping function from input to output and the data needed to estimate the performance of an approximate mapping function. It also includes overfitting and underfitting the model with available dataset depending on its size. If the test data is very small, it results in an optimistic and high variance estimation of model performance.

Hardware Requirements of Deep Learning [19]

ISSN: 2233-7857 IJFGCN

Deep learning network involves the processing of voluminous data which involves an intensive computation for giving better accuracy. Since it uses a large dataset, it cannot be given the outcome quickly in a normal computer with CPU. When the hardware capacity of a system increases, it is possible to use multiple layers to perform richer computations. Sometimes, we may depend on expensive third-party services to perform experiments in a space that could have staggering ramifications in myriad verticals. Even a simple application in machine learning such as chatbots costs much higher [18].

CPU: Central Processing Unit or processor is able to transfer data stored in memory or in external devices to GPU and vice-versa. When the storage happens across a network, CPU does the job of data transfers by communicating with the network components.

GPU: Graphics Processing Unit, or graphics card is initially designed video memory transfers. Recently, the advancements in gaming industry and in computer vision fields, GPUs play a vital role with large processing powers by parallel programming. GPUs were initially designed for graphic processing then later it was used for scientific computing well due to its parallel computations. Hence, it reduces the training time rapidly from moths to days. GPUs are now an important component of a computer capable of video rendering, image rotation, shadowing, compression etc., and in deep learning systems.

CUDA is one such language developed by Nvidia in 2006 which is useful to write programs for graphics processing.

TPU: Tensor Processing Unit is an ASIC (Application Specific Integrated Circuit) chip specially designed by Google for handling a large number of multiplications and additions for neural networks at a greater speed by reducing the use of too much power. It accelerates neural computations such as matrix to matrix and vector to matrix. A TPU requires a constant flow of data such as the weights of the neural network called the parameters. It is designed to meet a specific, fixed functionality with no programming involved.

While using TPU for training, it increases the performance 15 to 30 times over the present day's CPUs and GPUs and with 30 to 80 times higher performance-per-watt ratio. The TPU is a 28nm, 700MHz ASIC that fits into SATA hard disk slot and is connected to its host via a PCIe Gen3X16 bus that provides an effective bandwidth of 12.5GB/s.

Google's TPU can be accessed only through TPU cloud services. The TPU hardware available for public is called as Coral edge TPU. The edge TPU will be a lesser capable device compared with the normal TPU cloud services.

Data Requirements for Deep Learning

Data requirements is a major concern for deep learning than other analytic methods [3].

For an effective use of neural networks in most applications require large labeled training datasets with an access to sufficient computing infrastructure. The deep learning techniques are particularly powerful in extracting patterns from complex, multidimensional data types such as images, video, and audio or speech [3].

Deep-learning methods require thousands of data records for models to become relatively good at classification tasks and, in some cases, millions for them to perform at the level of humans. By one estimate, a supervised deep-learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category and will match or exceed human level performance when trained with a data set containing at least 10 million labeled examples. In some cases where advanced analytics is used, so much data are available—million or even billions of rows per data set—that AI usage is the most appropriate technique. However, if a threshold of data volume is not reached, AI may not add value to traditional analytics techniques [3].

These massive data sets can be difficult to obtain or create for many business usecases, and labeling remains a challenge. Supervised models are greatly used in most of the artificial intelligence techniques which require humans to label and categorize the underlying data. However, some promising new techniques overcome these data bottlenecks, such as generative adversarial networks, reinforcement learning, one-shot learning and transfer learning. They allow a trained AI model to learn about the problem domain based on a small number of real-world examples [3].

ISSN: 2233-7857 IJFGCN

About the work

Conventional learning of a machine is based on specific tasks, datasets and the training is done by a separate model on them. No knowledge is retained for transferring between models. In transfer learning, we can use the information such as features, weights etc., gained from the previously trained models for training newer models. It is also suitable for training the model with small dataset.

Transfer Learning

In this technique, a network model first gets trained with a problem domain which is similar to the problem to be solved. A new model is trained on the problem of interest with few layers from the already trained model of another similar problem. The training time of network model is greatly reduced and results in lower generalization error [18].

To implement a transfer learning, two main approaches are available. They are Weight Initialization and Feature Extraction.

The weights of re-used layers are used as a starting point for the training process and are adapted to the problem to be solved. Hence, weight initialization is done by transfer learning. This works well because the model is pre-trained with more labeled data. We can make use of this pre-trained model for the training of our problem of interest which will produce good results due to similarity in problem regions [18].

This kind of weight transfer is possible if the network structure is similar to one another. In our case, we can do transfer learning on our version Tiny YOLO-RV1 using the weights of a trained Tiny YOLOv3 model. Before starting the training of the new Tiny YOLO-RV1 model, we can only load the weights of some layers of a Tiny YOLOv3 model to the new Tiny YOLO-RV1 model which are identical and are similar in structure. After initializing the weights of Tiny YOLO-RV1 model with some starting point, now we can start training with our small dataset. This will tremendously decrease the training time and will give more improved prediction results.

In this work, we try to train the standard Tiny YOLOv3 and the Network Complexity Reduced Version Tiny YOLO-RV1 Deep Learning Network Model for better performance with Insignificantly Small Training Dataset. The scope of this work is to improve the performance of training with small dataset using transfer learning. As we did earlier, we hereby try to train the model by transfer learning in the standard Tiny YOLOv3 and in the proposed Tiny YOLO-RV1 deep learning networks only with 2 drone images and comparing their performances with standard Tiny YOLOv3 and Tiny YOLO-RV1 Models without transfer learning. The arrived results proved that training the deep learning network with few samples of images or any kind of data through transfer leaning is possible and also achieved good accuracy. Further, we proved that our proposed Tiny YOLO-RV1 achieved ideal performance in terms of accuracy even with the very small training set by using transfer learning. Our results proved the possibility of training our version Tiny YOLO-RV1 deep learning network with insignificantly low number of images by using transfer learning.

II. MODELING PARAMETERS

Challenges of Training

The training part is the most challenging process in deep learning networks. It is time-consuming process which requires much effort to configure the model's training and requires intense computations to execute the process. To reduce the model error, a set of weights must be repeatedly updated and evaluated to check for minimal error. The derivatives of the model error for a specific set of weights are updated using backpropagation method. The stochastic gradient descent optimization algorithm is most commonly used for learning from these gradients.

Components of the Learning Algorithm

When training a deep learning model by stochastic gradient descent with backpropagation, the number of components and hyperparameters must be chosen before starting the training.

First, an error function must be chosen. It is also called as an objective function or the cost function, or the loss function. Then a probabilistic framework with maximum likelihood is chosen for an inference.

ISSN: 2233-7857 IJFGCN

For classification problems, the cross-entropy loss function and for regression problems mean squared error loss function is used for an inference.

There are five important hyperparameters must be considered for controlling the learning algorithm in deep learning networks. They are loss function, weight initialization, batch size, learning rate, and epoch.

Loss Function

The loss function estimates the model's performance with some specific set of weights on training dataset examples. In the optimization process, the updates of model parameters start with some initial weights which are small random values. The starting point is important to the optimization algorithm due to the non-convex error surface. Different weight initialization methods are used to select the scale and for the distribution of these values.

Weight Initialization

Initialization of weight parameter is done with some small random values at the start of the training process of a model. When updating a model, the model error or a loss is calculated by the number of samples from the training dataset.

Error gradient is estimated when the samples are chosen either completely or only one from the training dataset which depends on the size of the dataset. We can also choose a specific set of samples from the training set. The number of samples is known as the batch size.

Batch Size

The number of samples taken from the training dataset is used to estimate the error gradient and is known as batch size. It is done before updating the model parameters.

At the end of the training of each batch, the predictions are compared to the expected output and ultimately an error is calculated. From this error, the update algorithm is used to improve the model to move down the error gradient.

A training dataset can be divided into one or more batches.

- In a batch gradient descent, all the training samples are completely taken as only one batch.
- In a stochastic gradient descent, the batch size is only one sample.
- In a mini-batch gradient descent, the condition for choosing batch size as $1 \geq batch\ size \leq$ *size of entire training dataset*.

Learning Rate:

Learning rate is a hyperparameter which are updated in an amount in every iteration of the algorithm during training. It is also known as step size. After performing the iteration many times, a set of model hyperparameters are fixed. This hyperparameter is configurable and has a small value in the range from 0 to 1.The total number of iterations of the training process depends on the number of complete passes through the training dataset before the training ends. This is known as the number of training epochs [26].

Epochs

It is a hyperparameter which is defined before the start of training of a model. One epoch is equal to one forward pass and one backward pass of all the training examples.

III. OBJECT DETECTION USING YOLO AND TINY YOLO-RV1

Object detection is a computer vision technology which detects the instances of certain objects. To detect the object, there are many techniques available and are classified into two categories based on how they perform their tasks.

In the first category, algorithms are based on classification and they work in two steps. First, regions of interest are selected from the image and then the objects present in the regions are

ISSN: 2233-7857 IJFGCN

classified using Convolutional Neural Network (CNN). Since predictions are made for every selected region, these algorithms are very slow and are very slow for realtime object detection.

Region based Convolutional Neural Network (RCNN) is the well-known example of this category.

The algorithms in the second category are based on regression. Instead of selecting the interested regions, they scan the entire image in one run and make predictions for localizing, identifying and classifying the objects in an image. You Only Look Once (YOLO) is faster in detecting the objects in real-world scenario.

YOLO Object Detection

You Only Look Once (YOLO) is a regression based deep learning network for object detection. It uses regression algorithm. YOLO detects the objects by classifying them in an image and predicts the location of an object by a bounding box.

YOLO versus R-CNN Methods

The R-CNN techniques use regions to locate the objects present in an image. Selective search technique is used in R-CNN to generate bounding box region proposals. Then it extracts CNN features from each region separately for classification. These tasks will be done in multi steps. R-CNN focuses on a specific region in the image and then each individual component is trained. The selective search algorithm proposes 2000 region proposals approximately in an image. The classification of these regions is very time-consuming. Thus, it cannot be used in real-time.

Since R-CNN needs to classify 2000 region proposals in an image, it takes a lot of time to train the network. It also requires a lot of disk space to store the feature map of region proposals. Hence, these techniques are harder to optimize and are very slow compared to YOLO. YOLO is faster compared to R-CNN and we can easily optimize because it uses only one network to run through all the image parts.

YOLO Deep Learning Network Architecture

For better understanding of YOLO deep learning network, we must clearly understand the algorithm, the network, and the loss function.

YOLO Algorithm

In YOLO, the input image is divided into S x S grid. When the object center falls in a grid cell, then that grid cell is responsible for finding the object. 'B' bounding boxes and confidence scores of these boxes are predicted by each cell in a grid. From the confidence scores, the model understands whether the predicted box contains an object. It is represented by $x, y, w, h, \Pr(Object) \times IOU_{\Pr}^{Gt}$, where (x,y) coordinates represent bounding box center coordinates relative to the grid cell, w and h represent the box width and height respectively, Pr*(Object)* indicates the probability that the current bounding box has a valid object, IOU is the overlap probability between the predicted(Pr) box and the ground truth(Gt). Next, each grid cell calculates conditional class probabilities $C_i = \Pr(Class \mid Object)$. While predicting the class-specific confidence score for each box, the conditional class probability and confidence of individual box are multiplied and is denoted as $\Pr(Class_i \mid Object) \times \Pr(Object) \times IOU_{\Pr}^{Gt} = \Pr(Class_i) \times IOU_{\Pr}^{Gt}$.

When a large object is present in an image, it is essential to perform non-maximal suppression for keeping the bounding box with a highest score and the boxes with less scores are deleted. If there are multiple categories in a grid, there will be problems. YOLO produces better results for small objects [20].

Network

YOLO network is structured as convolutional neural network. It includes convolutional layers and is followed by max-pooling layers and finally 2 fully connected layers.

Loss Function

ISSN: 2233-7857 IJFGCN

Multiple bounding boxes are predicted per grid cell in YOLO. But only one of the boxes is responsible for the objects in an image. Hence, we should calculate the true positive loss.

The box that has the maximum Intersection over Union (IoU) value with the ground truth is selected. The predictions are improved by the particular bounding boxes at certain aspect ratios and sizes.

YOLO Updates

The versions of YOLO are focusing on the accuracy of the model in detecting the objects.

YOLO v1

The first version YOLOv1 was introduced in 2015 and it uses a Darknet framework that is trained on ImageNet-1000 dataset. The use of YOLOv1 is restricted due to its limitations. One of the limitations of this architecture is to find out small objects that appeared as a cluster. It is inefficient for generalizing the objects of the images with different scales than the trained image. Hence, it leads to poor performance in locating the objects within the input image.

YOLO v2

The second version YOLOv2 was released in 2016 with the name YOLO9000. YOLOv2 meets the performances of a Faster R-CNN and Single Shot multi-box Detector in a much better way, both produces good object detection scores. It is better and faster than YOLOv1. YOLOv2 uses darknet-19, a 19-layer network with 11 more layers for object detection.

YOLO v3

YOLOv3 is an upgraded version of YOLOv2 which uses a Darknet53. Its architecture consists of 53 layers trained on ImageNet and another 53 layers are used for object detection, totally 106 layers. Though the accuracy of the network is improved by YOLOv3, it reduces the speed from 45 fps to 30 fps.

Updates of YOLOv3 over YOLOv2

Bounding box predictions: YOLOv3 uses logistic regression for predicting the confidence score for an object present in the bounding box.

Class predictions: YOLOv3 replaces the SoftMax function by independent logistic classifiers for each class for multi-labeling classification. The logistic classifier calculates the likeliness of the object in the input which belongs to a specific label.

Improved abilities at different scales: YOLOv3 predicts the objects with 3 different scales in each location in an input image. The quality of the output is improved by a fine-grained information obtained by upsampling.

YOLO model requires multiple training and running the experiments for getting high accuracy in object detection. It is a time-consuming process with a CPU based system. To overcome this issue, tiny YOLOv3 network is better choice to satisfy the real-time requirements based on limited hardware resources.

Table 1. Structure of a Tiny YOLOv3 Network

Layer #	Layer Name	# of Filters	Filter Size / Stride	I/P Dimension	O/P Dimension	BFLOPs
0	conv l	16	3x3/ 1	416x416x3	416x416x16	0.150
1	max		2x2/ 2	416x416x16	208x208x16	0.003
2	conv l	32	3x3/ 1	208x208x16	208x208x32	0.399
3	max		2x2/	208x208x3	104x104x3	0.001

ISSN: 2233-7857 IJFGCN

4	conv l	64	2 3x3/ 1	2 104x104x3 2	2 104x104x6 4	0.399
5	max		2x2/ 2	104x104x6 4	52x52x64	0.001
6	conv l	128	3x3/ 1	52x52x64	52x52x128	0.399
7	max		2x2/ 2	52x52x128	26x26x128	0.000
8	conv l	256	3x3/ 1	26x26x128	26x26x256	0.399
9	max		2x2/ 2	26x26x256	13x13x256	0.000
10	conv l	512	3x3/ 1	13x13x256	13x13x512	0.399
11	max		2x2/ 1	13x13x512	13x13x512	0.000
12	conv l	1024	3x3/ 1	13x13x512	13x13x102 4	1.595
13	conv l	256	1x1/ 1	13x13x102 4	13x13x256	0.089
14	conv l	512	3x3/ 1	13x13x256	13x13x512	0.399
15	conv l	18	1x1/ 1	13x13x512	13x13x18	0.003
16	yolo					
17	route	13				
18	conv l	128	1x1/ 1	13x13x256	13x13x128	0.011
19	up-samp le		2x13 x1	3x128	26x26x128	
20	route	198				
21	conv l	256	3x3/ 1	26x26x384	26x26x256	1.196
22	conv l	18	1x1/ 1	26x26x256	26x26x18	0.006
23	yolo					
					Total BFLOPs	5.448

The Proposed Network Complexity Reduced YOLO-RV1

The following table shows the layers of the network complexity reduced version of YOLO.

Table 2. Tiny YOLO-RV1

Layer #	Layer Name	# of Filters	Filter Size / Stride	I/P Dimension	O/P Dimension	BFLOPs
0	conv l	16	3x3/ 1	416x416x3	416x416x1 6	0.150
1	max		2x2/	416x416x1	208x208x1	0.003

ISSN: 2233-7857 IJFGCN

			2	6	6	
2	conv l	32	3x3/ 1	208x208x1 6	208x208x3 2	0.399
3	max		2x2/ 2	208x208x3 2	104x104x3 2	0.001
4	conv l	64	3x3/ 1	104x104x3 2	104x104x6 4	0.399
5	max		2x2/ 2	104x104x6 4	52x52x64	0.001
6	conv l	128	3x3/ 1	52x52x64	52x52x128	0.399
7	max		2x2/ 2	52x52x128	26x26x128	0.000
8	conv l	256	3x3/ 1	26x26x128	26x26x256	0.399
9	max		2x2/ 2	26x26x256	13x13x256	0.000
10	conv l	512	3x3/ 1	13x13x256	13x13x512	0.399
11	max		2x2/ 1	13x13x512	13x13x512	0.000
12	conv l	1024	3x3/ 1	13x13x512	13x13x102 4	1.595
13	conv l	256	1x1/ 1	13x13x102 4	13x13x256	0.089
14	conv l	512	3x3/ 1	13x13x256	13x13x512	0.399
15	conv l	18	1x1/ 1	13x13x512	13x13x18	0.003
16	yolo					
					Total BFLOPs	4.234

The Tiny YOLO-RV1 is a reduced version of Tiny YOLOv3 and it will only contain 16 layers – but the standard Tiny YOLOv3 model will contain 23 layers and the reduction in layers will rapidly reduce the memory requirements and little bit of processing power.

Another major difference is that the standard Tiny YOLOv3 model will contain two YOLO detection layers. But the proposed tiny YOLO-RV1 will contain only one YOLO detection layer. The total BFLOPs of the Tiny YOLO-RV1 is obviously lower than that of the standard Tiny YOLOv3 model. Hence, we may expect considerable reduction in training time in the case of the proposed model.

IV. RESULTS AND DISCUSSION

Training Dataset

The dataset that contains two drone images has been taken from [ee] for training. These images have been already taken in our previous evaluation. The ground truth bounding box information is also available with these images. The deep learning network is trained with this data by repeatedly introducing the above two images as training images.

ISSN: 2233-7857 IJFGCN

Deep Network Design using Darknet

A familiar open source framework used in the design of deep learning network is Darknet and is written in C and CUDA. Darknet acts as a basis for training the YOLO network. It is easy and fast to install, supports CPU and GPU computations and highly accurate framework for real-time object detection. The framework features You Only Look Once (YOLO) real-time object detection system [9].

When Darknet loads the config file and weights, the information useful for classifying the images is shown and it prints the matching classes of the images.

Darknet is also capable of handling the networks without using CUDA or OpenCV. For example, Recurrent Neural Network (RNN) is one such powerful model for representing dynamic data which is handled by Darknet without using either CUDA or OpenCV.

In this work, a fork of Joseph Redmon's Darknet [14] called Alexey's Darknet has been used for the implementation [8].

Darknet can be used easily and quickly since it is written in C and it includes the features for using YOLO also. Hence, we have chosen Darknet for designing our fast drone detection system.

Implementation Requirements

A normal laptop with Core i7 and 16Gb RAM is used for implementation. The setup depends only on the computing power of CPU not the higher capability GPU.

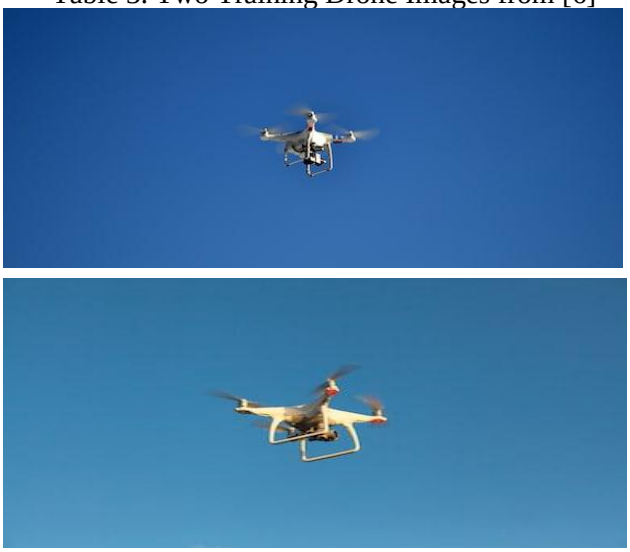
64-bit version of lUbuntu 16.04 is used as the operating system on the laptop. Alexey's implementation of Darknet from [11] is used and is compiled with maximum CPU power of our Laptop.

While compiling Darknet, we can notice that GPU option has been disabled and Enabled Multi-Processing option.

Hence, our experiments can be repeated as many times on any 64bit laptop or desktop without any additional GPU/TPU hardware or higher end computing resources.

Training and Testing Images

Table 3. Two Training Drone Images from [6]



ISSN: 2233-7857 IJFGCN

Table 4. 20 Different Size Drone Images from [6] used for Testing

Batch Size and Epochs

We used the same two drones image dataset for training as we used in our previous evaluation. For a successful training with such a small dataset, the batch size, subdivisions size, and epochs are significant.

For a successful training of a model, a batch size is selected as 2 and subdivisions size as 1. It means that both the images are trained in each epoch and repeated it for 700 epochs.

For our experiment, we used 2 Training Images and 20 test images. The number of Epochs of training is 800.

We have trained all the three models such as standard Tiny YOLOv2, standard Tiny YOLOv3 and the Tiny YOLO-RV1, and then trained the proposed Tiny YOLO-RV2 upto 800 epochs.

The weights are saved at each 100 epoch and are used to attain the highest performance in terms of mAP. When implementing, the best performance is achieved at the 700th epochs in both the network models.

ISSN: 2233-7857 IJFGCN

In the case of TL Tiny YOLOv3 and TL Tiny YOLO-RV1, we stopped training at 100th epoch itself since the maximum accuracy achieved at that epoch itself.

The following table shows the performance of the three compared algorithms in term of Precision, Recall and F1-Score.

Model	Precision	Recall	F1-Score	Average IoU (%)	mAP	
Tiny YOLOv3	0.64	0.45	0.53	38.17	0.486513	5.448
TL Tiny YOLOv3	0.95	0.95	0.95	75.81	0.90	
Tiny YOLO-RV1	0.75	0.6	0.67	48.09	0.615702	
TL Tiny YOLO-RV1	1.00	1.00	1.00	79.07	0.95	

Table 5. Transfer Learning Results of Tiny YOLOv3 and Tiny YOLO-RV1

The following bar chart shows the performance in terms of precision, recall and f1-score. It is obvious that, with respect to these three metrics, the proposed Transfer Learning based Tiny YOLO-RV1 (TL Tiny YOLO-RV1) performed good and provided the ideal performance of 1.00 in terms of precision, recall and f1-score. Note that, even with transfer learning, the standard Tiny YOLOv3 (TL Tiny YOLOv3) can not achieve the ideal performance of 1.00. This proves the effectiveness of the proposed TL Tiny YOLO-RV1.

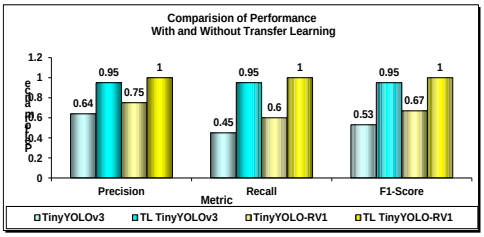


Figure 1. The Comparison of Performance in terms of Precision, Recall and F1-Score

The following bar chart shows the performance in terms of IoU. It is obvious that, with respect to IoU, the proposed TL Tiny YOLO-RV1 performed better than the other three compared models.

ISSN: 2233-7857 IJFGCN

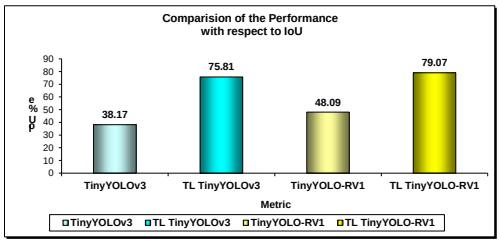


Figure 2. The Comparison of Performance in terms of IoU

The following bar chart shows the performance in terms of mAP. It is obvious that, with respect to mAP, the proposed TL Tiny YOLO-RV2 performed better than the other three compared models.

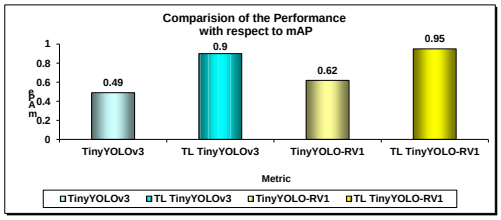


Figure 3. The Comparison of Performance in term of Map

Generally, deep learning networks consume the training time from several hours to weeks for obtaining high precision in detection in very higher end GPUs and TPUs. But in our work, we have performed the same kind of training process with the conventional CPU and showed the improvement in accuracy.

We observed that YOLOv3 took around 20 minutes and Tiny YOLO-RV1 and YOLO-RV2 took around10 minutes for its training with two images. By transfer learning, the system achieved a maximum accuracy even at the 100th epoch and only took 1/7th of the time required for Tiny YOLO-RV1. With this tremendous reduction in training time, the proposed TL Tiny YOLO-RV2 gave ideal accuracy of 1.00. Hence, we proved that a deep learning network can be trained within few minutes on a normal CPU and to attain significant detection performance in terms of accuracy and mAP.

CONCLUSION

In this work, we have successfully implemented a drone detection system by four different ways with a small number of training images. From this, we showed that training a deep learning network is possible with a small number of training images. We achieved some improvement in terms of accuracy and mAP in the proposed TL Tiny YOLO-RV1 Model. In our experiment, only two drone images have been used to train the Tiny YOLOv3, TL Tiny YOLOv3, Tiny YOLO-RV1 and the proposed TL Tiny YOLO-RV1 networks. We achieved an ideal performance in our proposed TL Tiny YOLO-RV1. We have proved the successful detection of drones with 20 images test dataset and measured the performance with different metrics. From the results obtained with different metrics, it is proved that the training of a deep learning network is possible with a very few images per class for a successful detection.

Our improved results with TL Tiny YOLO-RV1 proved that any kind of research on deep learning can be done in a normal desktop or laptop without any higher end CPUs or GPUs. It provides a

ISSN: 2233-7857 IJFGCN

solution to the hardware related issues in deep learning related research. We have also shown that the training time of huge dataset in a higher end hardware has been significantly reduced. From this, it is proved that training a deep learning network with a small dataset in a CPU enabled computer is possible.

We proved that training a deep learning network is performed within an hour and some meaningful detection performance is obtained in terms of mAP. In addition to that we showed the ideal performance can be achieved using transfer learning techniques. Hence, anyone who wants to do research in deep learning can be able to do a prototype of a complex deep learning system for a small number of samples such as images or any data. It is possible to use our Tiny YOLO-RV1 and TL Tiny YOLO-RV1 models for some small-scaled object detection problem where the time and hardware are major constraints.

REFERENCES

1. Arne Schumann, Lars Sommer, Johannes Klatte, Tobias Schuchert, Jurgen Beyerer, “Deep Cross-Domain Flying Object Classification for Robust UAV Detection”, IEEE August 2017.
2. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “Unified, Real-Time Object Detection”, IEEE 2016.
3. Cemal Aker, Sinan Kalkan, “Using Deep Networks for Drone Detection”, IEEE July 2017.
4. Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas and Vladimir Golkov, “FlowNet: Learning Optical Flow with Convolutional Networks”, IEEE 2015.
5. Muhammad Saqib, Abin Sharma, Sultan Daud Khan and Michael Blumenstein, “A Study on Detecting Drones Using Deep Convolutional Neural Networks”, IEEE 2017.
6. Christian Reiser, Bounding box detection of drones (small scale quadcopters) with CNTK Fast R-CNN, https://github.com/creiser/drone-detection.
7. Chuan-en Lin, “drone-net”, https://github.com/chuanenlin/drone-net.
8. Alexey, “Windows and Linux version of Darknet Yolov3 & YOLOv2 Neural Networks for Object Detection”, https://github.com/AlexeyAB/darknet.
9. Joseph Redmon, "Darknet: Open Source Neural Networks in C", http://pjreddie.com/darknet/, 2016.
10. 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.
11. 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, Washington D.C., March 2015.
12. Dinesh Sathyamoorthy, "A Review of Security Threats of Unmanned Aerial Vehicles and Mitigation Steps", an article published at ResearchGate.net, October 2019.
13. Jason Brownlee, “How to Improve Performance with Transfer Learning for Deep Learning Neural Networks”, https://machinelearningmastery.com
14. Jason Brownlee, "A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models”, https://machinelearningmastery.com, February 2019.
15. Jason Brownlee, “Impact of Dataset Size on Deep Learning Model Skill and Performance Estimates”, https://machinelearningmastery.com, January 2019.
16. Michael Chui et al., “Notes from the AI frontier: Insights from hundreds of use cases, McKinsey Global Institute, Discussion Paper, April 2018.
17. Shriram Ramanathan, Senior analyst, artificial intelligence and big data analytics at Lux Research "Five Challenges for Deep Learning", an article at eetimes.com
18. Colin Adams, "Expensive, Labour-Intensive, Time-Consuming: How Researchers Overcome Barriers in Machine Learning", an Article at journal.binarydistrict.com, July 2019.
19. Himanshu Singh, "Everything you Need to Know About Hardware Requirements for Machine Learning", An article at einfochips.com, February 2019.

ISSN: 2233-7857 IJFGCN

20. YOLO Deep Learning: Don't Think Twice, https://missinglink.ai/guides/computer-vision/yolo-deep-learning-dont-think-twice.
21. Jason Brownlee, "A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models", https://machinelearningmastery.com, 2019.

ISSN: 2233-7857 IJFGCN