An Improved Object Detection Model based on Optimised CNN

Senthil Kumar Jayapalan[1], Syahid Anuar*[2]

[1,2]Razak Faculty of Technology and Informatics,
Universiti Teknologi Malaysia,
Kuala Lumpur, Malaysia
[1]kjsenthil@graduate.utm.my,[2]syahid.anuar@utm.my

Article history

Received:
13 Sept 2022

Received in revised form:
29 Nov 2022

Accepted:
10 Dec 2022

Published online:
15 Dec 2022

*Corresponding author
syahid.anuar@utm.my

Abstract

Object detection is a computer vision technique that gives the ability to individually locate, recognise, and interpret multiple objects in an image with a better understanding. Modern image understanding tasks like image classification have been improved by state-of-the-art deep learning methods, particularly by convolutional neural networks (CNN). Region-based object detection algorithms such as Fast-RCNN achieve classification by CNN but over a longer period of time. You only look once (YOLO) prompts the object location and classification, treating object detection as a regression problem in an end-to-end network in a single step, whereas its accuracy decreases when the image has similar objects in a confined area, particularly when independent of the surrounding context. The aim of the current study is to improve YOLOv3 by optimising Darknet-53 to address the memory issue, using switchable normalisation techniques. We investigated the performance of five pre-trained networks, SqueezeNet, GoogleNet, ShuffleNet, Darknet-53, and Inception-V3, using a confusion matrix employing various epochs, learning rates, and mini-batches based on transfer learning. Darknet-53 took five times longer to complete the training and also ran into errors, most likely due to GPU memory shortages, whereas GoogleNet virtually obtained the same results in a fraction of the time. Using switchable normalisation techniques with the 10 class CIFAR-10 dataset, and utilising deep network designer (DND) of MATLAB R2021a, optimised versions of Darknet-53 increased the validation accuracy, considerably reducing the training time, and rectified the memory issue, which were then used as a backbone for YOLOv3 for effective object detection. The enhanced YOLOv3 was then assessed using a vehicle dataset and a sample Kuala Lumpur traffic scene using average precision. YOLOv3 with optimised CNN dNet-CIN as the backbone produced the best experimental results, with an FPS of 3.21 and a mAP-50 of 97%.

Keywords: *Object Detection, Transfer Learning, Computer Vision, Optimisation, CNN.*

1. Introduction

Object detection is the ability to locate an instance of an object in any image, and object recognition is the ability to determine whether a given object belongs to a specific class. Object extraction from natural images, along with image processing, is often considered an elementary and critical problem in the field of computer vision. Meanwhile, research in this area has revealed that it is still a looming and overwhelming challenge [1]. This is due to the fact that images of natural environments depict real-world adaptations, which are distinguished by a wide range of shapes, colours, and textures. In particular, with regard to some challenging issues in other computer vision tasks, such as objects under different viewpoints, illuminations, and intraclass variations, the challenges in object detection include, but are not limited to, the following aspects: object rotation and scale changes, accurate object localisation, dense and occluded object detection, rise in detection speed, and so on [2]. Deep learning techniques are currently used to drive high-performance vision systems. In terms of speed and accuracy, deep learning-based object detection outperforms classical machine learning techniques [3]. Deep networks may face obstacles and hurdles throughout the training process, such as exploding/vanishing gradients and degradation. When the depth of a network exceeds the maximum, it suffers from the degradation problem, which results in a decline in accuracy [4]. Although deeper networks are more accurate due to vanishing/exploding gradient information during network training, merely increasing the number of layers alone will not enhance accuracy indefinitely [5]. The internal covariate shift, which is the change in the distribution of the input data to a layer during training, is another matter of concern. CNN's unique qualities, such as incremental feature extraction in subsequent layers, make it possible to use parts of a pre-trained model for a completely new task without retraining the entire network. Transfer learning is a technique that may be used to retrain a CNN model that has been trained on a large dataset.

1.1 Study Background

The field of detecting and classifying objects in real-time images has made substantial progress, but it still has some shortcomings, especially with objects that occupy smaller areas and are similar in shape [6]. R-CNN and other most recent approaches generate potential bounding boxes in an image using region proposal methods before running a classifier on these proposed boxes [7]. Some region-based object detection algorithms, such as Fast R-CNN, achieve classification over CNN by extracting proposals, but it takes a long time [8][3]. The architectures such as Faster R-CNN are precise, but the model itself is very complex, whereas a pre-trained network is ideal for object proposals to be classified. Nevertheless, both training and testing are inefficient because the network performs a forward pass on each and every object proposal independently [9]. Convolutional filters are deliberately applied to thousands of redundant and expensive object proposals. Single Shot Detector (SSD) is a popular object detection algorithm that was developed by Google Inc. It is based on the VGG-16 architecture which gives only 10-20% less average precision [10]. Joseph Redmon et al. proposed a state-of-the-art object detector named YOLO (You Only Look Once), which outperforms the other object detectors like Fast R-CNN. In YOLO, it is proposed to incorporate object position and classification by a single CNN [3]. It is currently one of the fastest algorithms known for its detection speed. However, it shows a substantial inaccuracy in the recognition of tiny targets, making it difficult to detect targets accurately in complicated backgrounds [11]. YOLO trains on full images and optimises its detection performance directly. However, due to the strong spatial constraints imposed on bounding box predictions, it faces difficulty in dealing with small objects in groups [7] [12]. A slight drop in accuracy is increased with the use of anchor boxes. While the mean average precision (mAP) declines, the improvement in recall means more space for the model to boost accuracy [13].

2. Related Studies

The previous studies have been primarily focused on the detection of various objects, vehicles, human detection, and face detection using YOLO. In some of the previous studies, not only the detection of objects but also the counting of objects like vehicles and people has been carried out by using traffic video records [14]. Alvar et al. modified the algorithm into a motion vector: MV-YOLO to track the motion information of vehicles from the compressed video stream [15]. Xu et al. (2018) proposed a new method called Dense YOLO to improve the original YOLOv2 structure and tested the performance of that model with other one-stage detection models in an aerial dataset with an accuracy achieved of 76.2% with a detection speed of 26 frames per second [16]. Yi Tan et al. proposed a unique technique to detect vehicles based on change detection followed by the generation of vehicle proposals, whereas the CNN classifier determined the proposal as non-vehicle and vehicle. From the previous studies, it is clear that deep learning based CNNs play a part in the detection and classification of objects. Most of the research work focused on the detection of vehicles, whereas some authors even classified the vehicles based on colour and type [3]. The YOLOv3 version has been used for the majority of the application studies beside the optimisation of YOLO. Some of the major applications were moving object detection via motion prediction, multiple object tracking, obstacle detection, pedestrian detection, object detection in foggy conditions, real-time abandoned baggage detection and people counting approach.

Transfer Learning has been utilised in a range of fields by fine-tuning pre-trained CNN models learned on ImageNet. The earlier studies on pre-trained CNNs based on transfer learning that uses CIFAR-10 and 100 datasets performed image classification on the embedded systems, used to train very deep residual attention networks, and used to evaluate the performance of deep learning models using MNIST and CIFAR-10 datasets. Similar to the scope of the present study, some of the previous studies were using CIFAR-10 with batch normalisation and ELU activation to test the performance of residual networks [17], and to find the impact of training set batch size [18]. YOLO, a state-of-the-art deep learning algorithm, seems to be the fastest when it comes to object detection but still lags in accuracy with other methods like Faster R-CNN. The aim of the current study is to improve YOLOv3 by optimising Darknet-53 to address the memory issue, using switchable normalisation techniques.

3. Methodology

CNN is a special form of neural network that is more suitable for computer vision tasks because of its capability. Currently, CNN-based object detection techniques are in use, which has three layers: convolution, pooling, and fully connected. Girshick et al.'s approach, based on linear classifiers with CNN-derived features, was the first to conclusively defeat HOG with DPM, with a mean average precision gain of roughly 20 percentage points across the 20 PASCAL VOC 2007 dataset classes [25]. Deep neural network training is difficult because the dispersion of each layer's inputs varies during training as the parameters of the preceding layers change. The requirement of lower learning rates and exact parameter initialisation slows down training, and thus makes training models with saturating nonlinearities notoriously difficult. This is known as internal covariate shift, which is probably handled by normalising layer inputs [26]. Many normalisation algorithms, such as batch normalisation (BN), group normalisation (GN), instance normalisation (IN), and layer normalisation (LN), have been developed in recent years as crucial components of deep learning, as shown in Figure 1. Batch normalisation allows us to use much higher learning rates while being less concerned with initialisation. It also serves as a regulariser, minimizing the need for dropouts in some cases. Normalisation techniques are essential for improving neural network training and generalisation. Many studies have shown that this enhances optimisation and allows deep networks to merge [27]. Batch normalisation is a key technique in the advancement of deep learning, allowing diverse networks to train. Many methods have demonstrated that this facilitates optimisation and allows very deep networks to converge. The stochastic uncertainty of the batch statistics functions as a regulariser, which can help with generalisation.

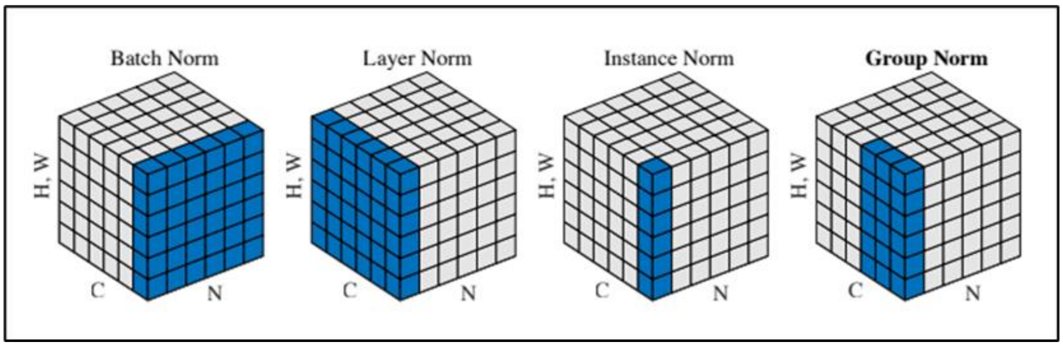


Figure 1. Normalisation Methods with a Feature Map Tensor [27]

Despite its enormous success, BN has limitations, which are exacerbated by its unique behaviour of normalising along the batch dimension. BN must, in particular, work with a suitably large batch size. Despite their considerable success, existing techniques frequently used the same normaliser in all normalisation layers of a network, resulting in inferior performance. Furthermore, multiple normalisers are utilised to tackle different tasks, making model creation more difficult. To address the aforementioned difficulties, Luo et al. introduced switchable normalisation (SN), which combines three forms of statistics calculated channel-wise, layer-wise, and minibatch-wise using IN, LN, and BN, as presented in Figure 2.

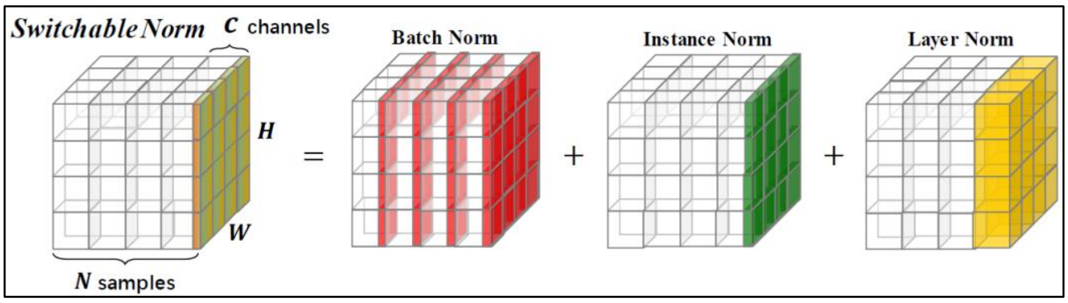


Figure 2. Switchable Normalisation Method [28]

Switchable normalisation is a normalisation technique capable of learning alternative normalisation operations for distinct normalisation layers in end-to-end manner in a deep neural network. IN primarily occurs in lower layers to eliminate variability in low-level features, LN appears in deeper levels to improve learning ability, and BN appears in the center [29]. According to the author, adopting a single normalisation method uniformly is not the best way to normalise the layers. Image categorisation and object detection, for example, favour a mix of three normalisers.

3.1 Data Preparation

In order to encourage computers to view images as object compositions, data plays a particularly critical role, an accomplishment that humans can do seamlessly while so far it has been obscuring for machines. In particular, humans would like machines to automatically identify what objects are present in an image, where they are precisely located, and which of them interact and how they interact [19]. All throughout the history of object recognition work, datasets have played an important role, not only as a common ground for assessing and comparing the performance of competing algorithms, but also in driving the field toward increasingly complex and difficult challenges. Figure 3 depicts the course of sequence in the data preparation that explains the selection and collection of datasets. Fundamentally, the dataset selection is divided into 2 major categories: object classification datasets responsible for feature extraction and object detection datasets, which are responsible for object localization and feature detection.

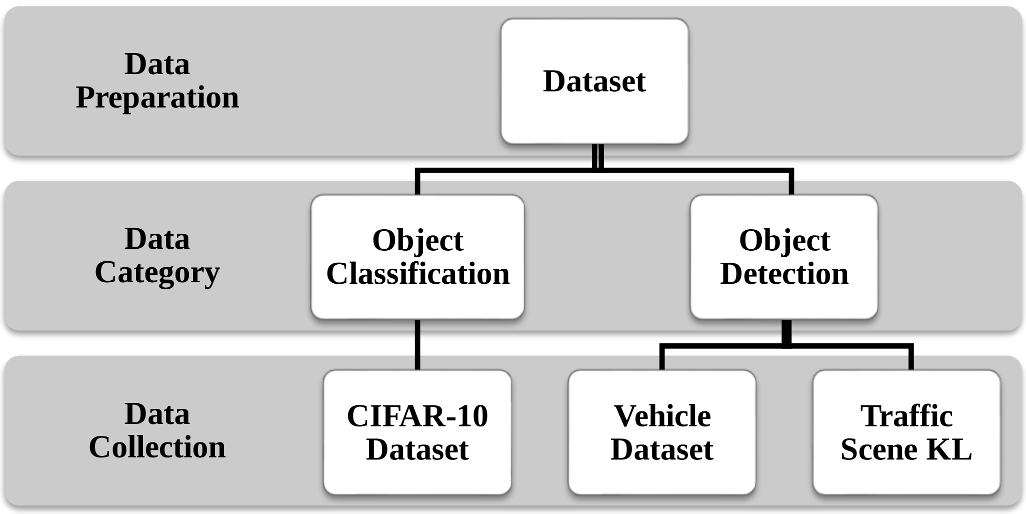


Figure 3. Selection and Collection of Dataset

CIFAR-10 is currently one of the most widely used benchmark datasets in machine learning and serves as a test ground for many computer vision methods. A concrete measure of popularity is the fact that CIFAR-10 was the second most common dataset in NIPS 2017 [20]. The benchmark image classification dataset, CIFAR-10, was chosen for the object classification. For object detection, a MATLAB-based vehicle dataset and a traffic scene from Kuala Lumpur are used. There are no rigid rules for separating training and test datasets, although the most typical split ratios are 80:20; 70:30; and 60:40, depending on the nature of the issue, the size of the dataset, and the architecture utilised. In this definition, a training dataset is used to build the model, and a test dataset is used to evaluate the model's predictive ability. Researchers have noted that the performance of the model performed best when the training-to-test ratio was 70:30 to distinguish between the datasets [21]. The CIFAR-10 dataset has 32 x 32 colour images that are divided into ten classes, each with 5000 training images and 1000 test images [22]. It contains ten classes with a total of 50,000 training images and 10,000 testing images. In addition, image augmentation was performed at random on the training datasets using different values to expand the dataset. The vehicle dataset is a MATLAB built-in dataset that consists of 295 images that each contain one or two labelled instances of a vehicle. The images are 720-by-960-by-3 in size, and this small data set is useful for investigating the object detection training procedure, which is then used to test the improved YOLOv3 model. The data is extracted and loaded into the workspace, and the ground truth labels, and bounding boxes are stored in a mat file, which is then extracted and stored as vehicleDataset. During the training phase, the images in each batch were randomly subjected to the following operations, as specified in the majority of the MATLAB examples: horizontal reflection; horizontal and vertical translation with a random value in the range [-30 30] pixels; and horizontal and vertical scaling with a random rate in the range [0.9 1.1].

MAT-Script 1: Object Detection Data

```
# Extract and Load the Vehicle Data
  filename = 'vehicleDatasetImages.zip';
  dataFolder = fullfile(tempdir,'vehicleImages');
  data = load('vehicleDatasetGroundTruth.mat');
  vehicleDataset = data.vehicleDataset;
# Load the KL Traffic Data
  dataImage = load('imageFilename.mat');
  currentDir = 'C:\Users\User\Documents\MATLAB\YOLOv3\trafficImages';
  imageFilename = fullfile(currentDir, dataImage.imageFilename);
# Load the gTruth Labels
  dataLabel = load('trafficData.mat');
  labelData = dataLabel.gTruth.LabelData;
  vehicleDataset = [imageFilename labelData];
```

Source: https://www.videvo.net/video/kuala-lumpur-streets-malaysia/4610/

The sample traffic scene data for Kuala Lumpur was downloaded from the opensource website as mentioned in the source in MAT-Script 1. The video is read frame by frame and the corresponding images are stored in a separate mat file. The images are then later used for the training process and used to detect the vehicles. Once the labelling is done, the mat file is stored and later loaded into the workspace for further training. As for the object detection, both the vehicle dataset and the sample traffic data needed to be stored in separate datastores. Two specific datastores are created for the images and bounding boxes. Then the two datastores are combined together to form the training and test data. Later, the input data is validated to check for invalid labels and bounding boxes. After validation, both the training and test data are pre-processed, and the image augmentation is done to expand the dataset. Finally, the anchor boxes are estimated to continue with the training process and the detection of vehicles.

3.1.1 Data Preprocessing: Data preprocessing plays an important role in machine learning and deep learning algorithms, and proper preprocessing of data is essential for achieving better performance. Kotsiantis et al. defined data preprocessing as including data cleaning, normalization, transformation, feature extraction, and selection [23][24]. Preprocessing data is intended to transform the raw data into a format that is easier and more effective to use for future processing steps. When it comes to the CIFAR-10 dataset, it is processed, and the data is split properly as training and test datasets. So, the dataset is downloaded from the corresponding source, then the training set is further split into training and validation sets in a 70:30 ratio and used for training the networks with MATLAB. Vehicle data is a small, processed and labelled MATLAB dataset derived from the Caltech Cars 1999 and 2001 datasets. The images of traffic data need extensive labelling to be done either manually or through automatic object detector algorithms through the video labeler app in MATLAB. In a video or image sequence, the video labeler app makes it simple to add rectangular region of interest (ROI) labels, polyline ROI labels, pixel ROI labels, and scene labels.

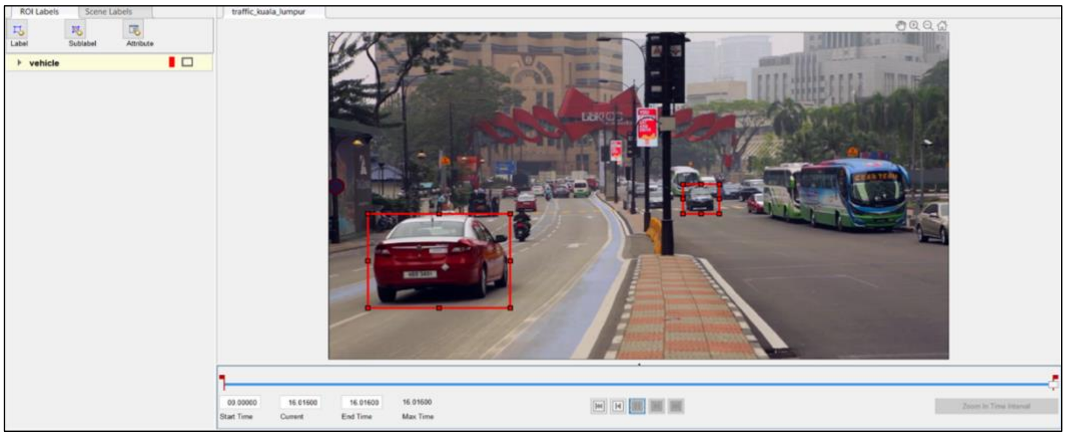


Figure 4. Labelling of KL Traffic Scene

An ROI label represents a rectangular, polyline, pixel, or polygon region of interest. A scene label, such as "sunny," describes the nature of the scene. Image classifiers, object detectors, and semantic and instance segmentation networks can all benefit from labelled data when they are validated or trained. The moving vehicles are labelled one by one in each of the images, as shown in Figure 4. The datasets are used to implement the methodology that is described in the following section, following the selection of the dataset and data preprocessing

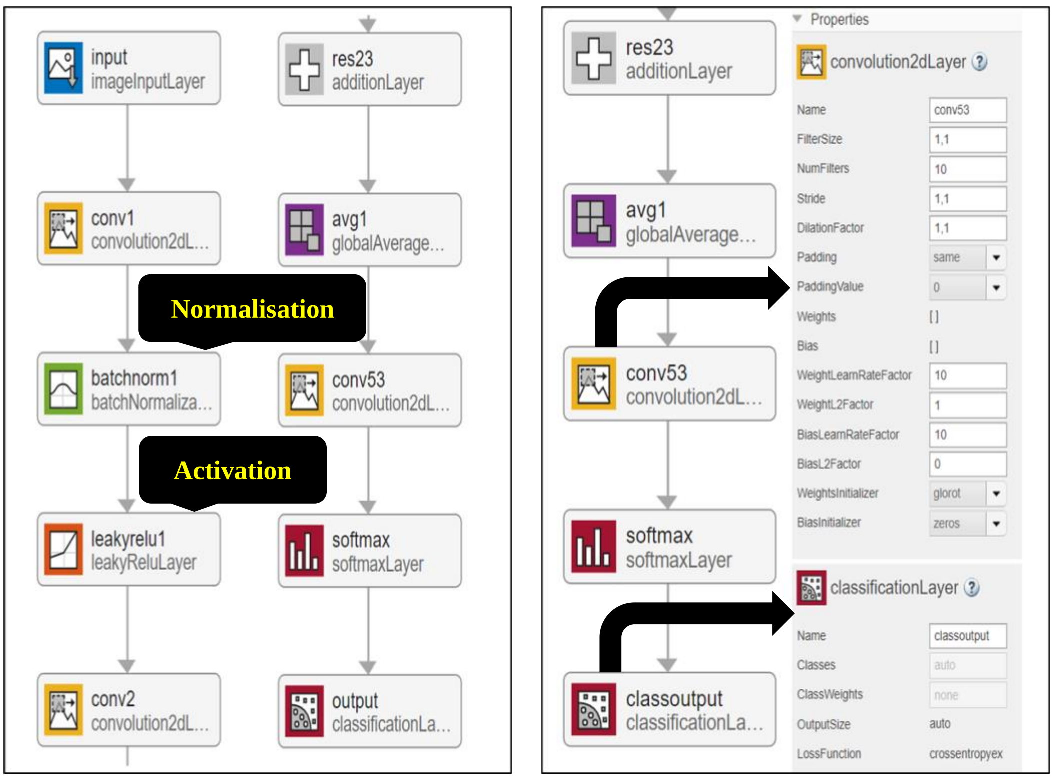
3.2 Optimisation of Darknet-53

In our previous study [30], a transfer learning based performance comparison between the selected five pre-trained networks was facilitated to select a network for CNN optimisation. Based on the findings, Darknet-53 was chosen to continue with the optimisation process. The process was entirely done in MATLAB using deep network designer (DND), which enables us to view the entire CNN architecture layer by layer as shown in Figure 5. This helped us largely focus on the hyperparameters of the entire network, which paved the way for the modification of parameters.

(a) The image on the left (a) Layers of Darknet-53, shows the network architecture of Darknet-53 with a deep network designer.

(b) Transfer learning involves the replacement of final convolutional layers and the freezing of initial layers to keep the weights intact.

(c) The image on the right (b) Replacing the Final Layers, shows the replacement of the final convolutional layer to match the classes (NumFilters=10) of CIFAR-10 and the classification layer.



Properties

convolution2dLayer

Name	conv53
FilterSize	1,1
NumFilters	10
Stride	1,1
DilationFactor	1,1
Padding	same
PaddingValue	0
Weights	[]
Bias	[]
WeightLearnRateFactor	10
WeightL2Factor	1
BiasLearnRateFactor	10
BiasL2Factor	0
WeightsInitializer	glorot
BiasInitializer	zeros

classificationLayer

Name	classoutput
Classes	auto
ClassWeights	none
OutputSize	auto
LossFunction	crossentropyex

(a) Layers of Darknet-53 (b) Replacing the Final Layers

Figure 5. Modifying the Parameters of Darknet-53

In terms of the optimisation process, normalisation and activation layers are considered as highlighted under (a) Layers of Darknet-53 as shown in Figure 5 apart from the replacement of the final convolutional layer and the classification layer. By default, Darknet-53 includes batch normalisation as a normalisation layer and as an activation function. Using switchable normalisation techniques, the default initial batch normalisation layers are updated by instance normalisation (IN), and the final layers are updated by layer normalisation (LN). The activation function LeakyReLU is updated by clippedReLU for the entire layers of Darknet-53. The complete optimisation process of Darknet-53, such as data preparation, training in deep network designer (DND), analysing the network, freezing the initial layers, training and testing in the MATLAB workspace, classification of data, and classification accuracy obtained using a confusion matrix, is presented in MAT-Script 2.

MAT-Script 2: Optimisation Process of Darknet-53

1. Preparation of Training Data

- url = https://www.cs.toronto.edu/ kriz/cifar.html;
- rootFolderTrain = "cifar10Train";

```
    ▪ [imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomize');
2. Training in Deep Network Designer
    ▪ Design = Default Darknet-53 | Optimised Darknet-53 (tNet-FC - tNet-4).
    ▪ Data = Load the augmented data into Image Datastore.
    ▪ Training = Batch-16 | Epoch-10 | LR-0.0001.
3. Load Pre-trained Network
    ▪ load("trainedDN/tNet-1.mat");
    ▪ net = tNet-FC | tNet-IN | tNet-1 | tNet-2 | tNet-3 | tNet-4;
    ▪ Analyse: analyzeNetwork(net);
4. Freeze Initial Layers
    ▪ layers(1:14) = freezeWeights(layers(1:14));
    ▪ lgraph = createLgraphUsingConnections(layers,connections);
5. Train the Network
    ▪ Training options: {'MiniBatchSize', 'MaxEpochs', 'InitialLearnRate'}
    ▪ net = trainNetwork(augimdsTrain, lgraph, options);
6. Data Classification
    ▪ valPred = classify(net,augimdsValidation);
    ▪ testPred = classify(net,augimdsTest);
7. Classification Accuracy
    ▪ val_accuracy = mean(valPred == valLabels);
    ▪ test_accuracy = mean(testPred == testLabels);
8. Confusion Matrix
    ▪ cm_mat = confusionmat(testLabels, testPred);
    ▪ cm_chart = confusionchart(testLabels, testPred);
```

3.3 Designing a YOLOv3 Detection Network in MATLAB

The YOLOv3 object detector is a multi-scale object detection network that makes predictions at multiple scales by utilising a feature extraction network and multiple detection heads. On an input image, the object detection model runs a deep learning CNN to generate network predictions from multiple feature maps. The object detector collects and decodes predictions to generate the bounding boxes. YOLOv3 uses anchor boxes to detect object classes in an image.

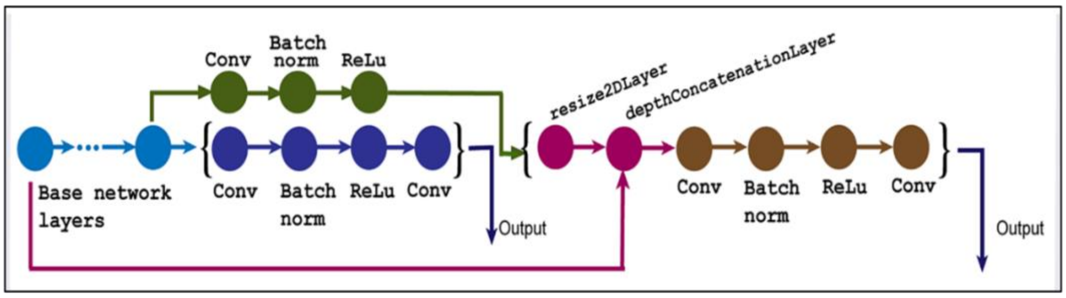


Figure 6. YOLOv3 Detection Network Design Sequence

The YOLOv3 object detection network is designed following the steps as described below. As a feature extraction base network, Darknet-53 is used, the pre-trained optimised CNN. As a detection network source, each and every layer of the feature extraction network can be used.

(a) Start the model with a feature extraction network (CNN), that serves as the base network for creating the YOLOv3 deep learning network, which can be a pretrained or untrained CNN.

(b) Create detection subnetworks (heads) by using convolution, batch normalisation, and ReLU layers.

(c) Add the detection subnetworks to any of the layers in the base network that can be used as a detection network source.

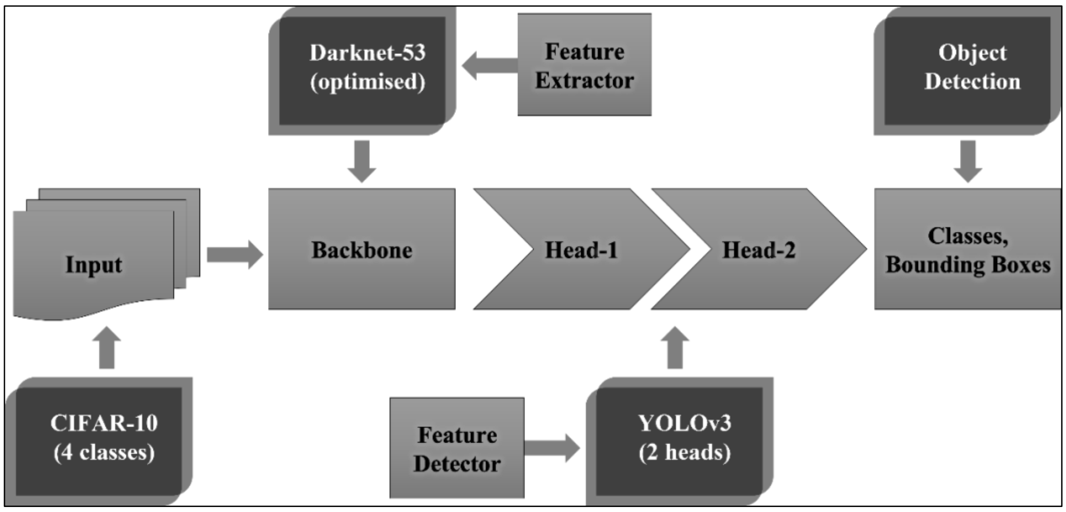


Figure 7. Modified YOLOv3 Model with Optimised CNN

The above Figure 7 depicts the modified detection network included with the optimised CNN as described in Figure 6. The optimised CNN acts as a feature extractor, and as for the detection, YOLOv3 does the feature detection part with 2 heads specifically designed for effective object detection. For implementation, the network input size, number of anchors, base network, the object detector, and its network detection source are described below in MAT-Script 3.

MAT-Script 3: Define YOLOv3 Object Detector

```
# Anchor Boxes
  netSize = [256 256 3];
  DataEstimation = transform(trainingData, preprocessData(data, netSize));
  [anchors, meanIoU] = estimateAnchorBoxes(DataEstimation, numAnchors);
# Detection Head
  area = anchors(:, 1).*anchors(:, 2);
  anchorBoxes = {anchors(1:3,:) anchors(4:6,:)};
# Object Detector
  baseNetwork = trainedNetwork (Optimised CNN);
  yolov3Detector = yolov3ObjectDetector(baseNetwork, classNames,
                   anchorBoxes, 'DetectionNetworkSource', {'res23', 'res11'});
```

4. Experiment and Result Analysis

We conducted a study to compare the performance of the pre-trained networks and from the experimental results, Darknet-53 was the only network faced with training error (out of memory). Therefore, Darknet-53 was selected among the five CNN architectures, and the experiment was further proceeded to continue with the optimisation of Darknet-53. The optimised Darknet-53 is to be employed in YOLOv3 to enhance the accuracy of the model for effective object detection. The entire experiment was carried out with a laptop GPU, and the experimental setup, including the hardware, software, and its specifications, is described in Table 1.

Table 1. Experimental Setup

Hardware/Software	Specifications
Microprocessor	AMD Ryzen 7 5800H-Radeon Graphics@3.20 GHz
RAM	16.0 GB
GPU	NVIDIA GeForce RTX 3060 Laptop GPU
Dedicated Video RAM	6.0 GB
Framework	MATLAB R2021a – 64 bits
Programming Language	MATLAB
Operating System	Windows 10 Home Single Language

4.1 Transfer Learning based CNN Investigation

In our previous study, the selected pre-trained CNN architectures were investigated based on transfer learning in order to evaluate the network's performance using a confusion matrix. The discriminative filters of state-of-the-art pre-trained models that have been trained on difficult datasets such as "ImageNet" can be used to recognise objects that have never been trained on [31]. The fundamental idea would be to use the initial layers of a previously trained model and only retrain the last few layers on new images. According to the evaluation results, the networks performed well on the LR-0.001 compared to LR-0.0001, except for Darknet-53. In comparison to the other four networks, Darknet-53 showed promising results with LR-0.0001, whilst the other networks' performances were on the decline. The performance comparison of the five pre-trained networks obtained from our previous study, encompassing both LRs, is shown in Table 2.

Table 2. Performance of the Pre-Trained Networks [30]

CNN Models	Precision (%)	Recall (%)	F1-Score (%)	Accuracy (%)
SqueezeNet	80.53	78.9	79.16	91.56
GoogleNet	89.16	88.82	88.85	95.53
ShuffleNet	86.15	85.08	85.13	94.03
Darknet-53	88.29	86.76	86.70	94.70
Inception-V3	**92.63**	**92.46**	**92.49**	**96.98**

According to the comparison of the pre-trained networks, Inception-V3 has achieved the highest accuracy of 96.98%, as well as other metrics such as precision, recall, and F1-score. The other networks produced somewhat lower results than Inception-V3, but altogether, all five pre-trained networks attained an accuracy of 90% or higher.

4.2 Darknet-53 Optimisation Outcomes

In order to test for the memory problem, we performed experiments both with and without freezing the initial layers. The experimental results for both the freezing and non-freezing layers to compare with the optimised Darknet-53 experimental results are presented together in Table 3. It clearly shows the training error (out of memory) and the time taken for the training. Especially with the non-freezing layers, it has taken too much time for the network to train. High classification accuracy can be attained through transfer learning, but accuracy must be ensured by accelerating the training speed as much as is feasible. The CNN's front-end network structure is used to mine for common features, and the extensive data training ensures that the network parameters have a strong capability for generalisation [32]. In the process of non-freezing layer training, the gradients of the whole network are updated in which the back propagation will change its parameters and adjust them in the direction of extracting the feature that requires more time in training the network. In the process of freezing layer training, the gradients of the backbone network are not updated, thus preventing the initial backbone weights from being destroyed in the early stage of training [33]. Thus, through freezing the initial layers, high accuracy can be maintained while the training time can be reduced.

Table 3. Default Darknet-53 Results

Hyper Parameters	**Accuracy (%)**	**Time (mins)**	**Accuracy (%)**	**Time (mins)**	**Accuracy (%)**	**Time (mins)**
LR – 0.001	Epoch – 10		Epoch – 20		Epoch – 30	
Mini Batch Size - 64	89.17	**781.00**	Out of Memory (Non-freezing)		Out of Memory (Non-freezing)	
Mini Batch Size - 32	Out of Memory (Freezing)		86.16	140.22	89.89	207.13
LR – 0.0001	Epoch – 10		Epoch – 20		Epoch – 30	
Mini Batch Size - 64	90.85	**760.37**	Out of Memory (Non-freezing)		Out of Memory (Non-freezing)	
Mini Batch Size - 32	Out of Memory (Freezing)		91.20	134.70	90.58	203.30

The experiments began by replacing the network's final convolutional and classification layers while freezing the network's initial layers. To accomplish the optimum, we tested all the possible combinations of updating the activation and normalisation layers. The final convolutional and classification layers were replaced based on transfer learning to match the CIFAR-10 classes. The augmented data was extracted and loaded into the image datastore before being added to the data section prior to the training process in DND. The training process began once the data was loaded with the appropriate training options. The training options are LR-0.0001, mini batch size-16, and epoch-10, which were chosen based on the experimental findings from our previous study. The replacement of final convolutional and classification layers, as well as their parameters, is described in Table 4.

Table 4. Description of Darknet-53 Optimised Versions

Sl. No.	Optimised Darknet-53	Description of the Replaced Layers and the Corresponding Parameters
1.	dNet-C53	Final and conv53 layers replaced (Default).
2.	dNet-CFC	Conv-53 updated by FC (Fully Connected) layer.
3.	dNet-CSN	IN+BN+LN – Conv-53.
4.	dNet-FIN	IN+BN+LN – FC.
5.	dNet-CLR	LeakyReLU updated by clippedReLU.
6.	dNet-CIN	IN+BN+LN – FC.

Aside from the final layers, the activation and normalisation layers have also been changed, such as updating the activation function with clippedReLU, and updating the initial batch normalisation layers with instance normalisation and the final layers with layer normalisation in the order of (IN+BN+LN). To distinguish it from the others, each optimised Darknet-53 relevant to the replacement layer and other parameters was given a unique name.

Table 5. Optimised Darknet-53 Results

Sl. No.	Representation of CNN	Optimised Darknet-53	Validation Accuracy (%)	Elapsed Time (min)
1.	Darknet-1	dNet-C53	97.88	137.70
2.	Darknet-2	dNet-CFC	97.48	136.29
3.	Darknet-3	dNet-CSN	95.08	187.45
4.	Darknet-4	dNet-FIN	94.40	184.30
5.	Darknet-5	dNet-CLR	96.42	125.28
6.	Darknet-6	dNet-CIN	95.02	173.28

The best performing network results are presented in Table 5. With the optimisation of Darknet-53, the accuracy has increased, and the training time has been reduced considerably.

4.3 YOLOv3 Experimental Outcomes

YOLO is the first single-staged object detection model and one of the fastest object detection algorithms available. A CNN serves as the backbone for feature extraction, which is followed by a feature detector for object detection. Switchable normalisation techniques are used to optimise the pre-trained CNN Darknet-53. The default CNN backbone was updated in sequence by a set of optimised Darknet-53 CNNs and, utilising two detection heads, the experiment was conducted to improve the accuracy of the YOLOv3 model for effective object detection. The parameters designated for the experiment are described below in Table 6. Following the experimental phase, the YOLOv3 models must be evaluated using average precision (AP), and the results can be presented using the precision recall curve (PRC). The methods for testing the model are shown in MAT-Script 4 with an overlap value of 0.5 to ensure anchor boxes overlap well with the bounding boxes in the training data. For the prediction to be declared positive, there must be a marginal overlap between the ground truth and the prediction boxes.

Table 6. Parameters for YOLOv3 Object Detection

Parameters	Description
Dataset – Detection	vehicleDataset and sample KL traffic scene data.
Dataset – Classification	CIFAR-10 (4 Classes).
Anchor Boxes	numAnchors = 6;
Pre-Trained CNN	baseNetwork = Darknet53 | Optimised Darknet-53;
Image Dimension	networkInputSize = [256 256 3];
Detection Source	DetectionNetworkSource = {'res23', 'res11'};
Epochs	numEpochs = 50;
Mini Batch Size	miniBatchSize = 8;
Learning Rate	learningRate = 0.001;
Average Precision	ap = evaluateDetectionPrecision(results, testData);
Average Miss Rate	am = evaluateDetectionMissRate(results, testData);
Confidence Score	[bboxes, scores, labels] = detect(yolov3Detector, I);

MAT-Script 4: YOLOv3 Model Evaluation

```
# Evaluate Detection
  results = detect(yolov3Detector,testData,'MiniBatchSize',8);
  overlap = 0.5;
  [ap, recall, precision] = evaluateDetectionPrecision(results, testData, overlap);
  [am, fppi, missRate] = evaluateDetectionMissRate(results, testData, overlap);
# Precision Recall Curve
  subplot(1,2,1);
  plot(recall, precision)
  xlabel('Recall')
  ylabel('Precision')
  title(sprintf('Average Precision = %.2f', ap))

  subplot(1,2,2);
  loglog(fppi, missRate);
  xlabel('False Positives Per Image');
  ylabel('Log Miss Rate');
  title(sprintf('Average Miss Rate = %.2f', am))
```

Starting with the custom backbone, the experimental results for both the vehicle dataset and the sample KL traffic scene data are discussed. The default pre-trained networks trained on ImageNet, SqueezeNet, GoogleNet, ShuffleNet, Darknet-53, and Inception-V3, served as the backbone for YOLOv3, and the results are provided in Table 7.

Table 7. YOLOv3 Custom CNN Backbone Results

Sl. No.	Model	Backbone	Scores	AP_{50}	AP_{75}
1.	YOLOv3	SqueezeNet	0.88	0.82	0.57
2.	YOLOv3	GoogleNet	0.98	0.79	0.35
3.	YOLOv3	ShuffleNet	0.96	0.83	0.35
4.	YOLOv3	Inception-V3	0.98	0.82	0.49
5.	**YOLOv3**	**Darknet-53**	**0.97**	**0.85**	**0.55**

Precision values for mAP-50 and mAP-75 are calculated using the COCO IOU metric. YOLOv3 performs well and provides good results in the old detection metric of 0.5 IOU, as mentioned in the original paper. As a result, mAP-50 is selected for the final model comparison for both the vehicle dataset and the sample KL traffic data. The results for a series of optimised pre-trained Darknet-53 used as the backbone for YOLOv3 with a vehicle dataset including the prediction confidence scores and average precision for mAP-50 and mAP-75 are presented in Table 8.

Table 8. YOLOv3 - Vehicle Dataset Results

Sl. No	Model	Backbone	Scores	AP_{50}	AP_{75}
1.	YOLOv3	dNet-C53	0.69	0.78	0.40
2.	**YOLOv3**	**dNet-CFC**	**0.99**	**0.86**	**0.62**
3.	YOLOv3	dNet-CSN	0.98	0.85	0.62
4.	YOLOv3	dNet-FIN	0.98	0.84	0.48
5.	YOLOv3	dNet-CLR	0.98	0.84	0.56
6.	YOLOv3	dNet-CIN	0.97	0.84	0.58

Among the models, YOLOv3 with optimised dNet-CFC as a backbone and a fully connected layer in place of the final convolutional layer provided the best results when compared to the model with the default CNN dNet-C53. In terms of precision, the same model with optimised dNet-CFC scored the highest among the other models, with a mAP-50 of 86%. The top-performing YOLOv3 model is emphasised and compared to related contemporary object detection models such as Faster R-CNN, a two-staged model, and SSD, a single-staged model, as well as YOLOv3 that uses the default Darknet-53 as a backbone. The comparison results are provided in Table 9.

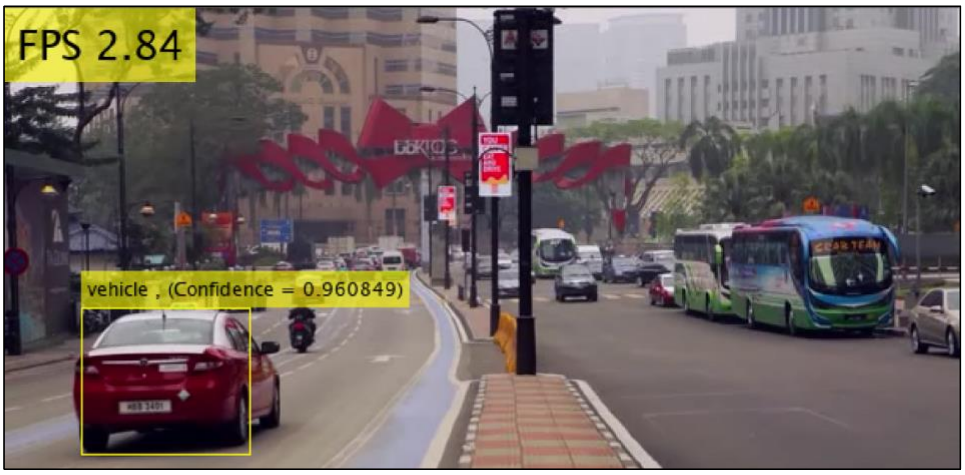
Table 9. Comparison of YOLOv3 Models using Vehicle Dataset

Sl. No.	Model	Backbone	AP_{50}
1.	Faster R-CNN	ResNet-50	0.82
2.	SSD	ResNet-50	0.79
3.	YOLOv3	Darknet-53	0.83
4.	YOLOv3	dNet-CSN	0.85
5.	**YOLOv3**	**dNet-CFC**	**0.86**

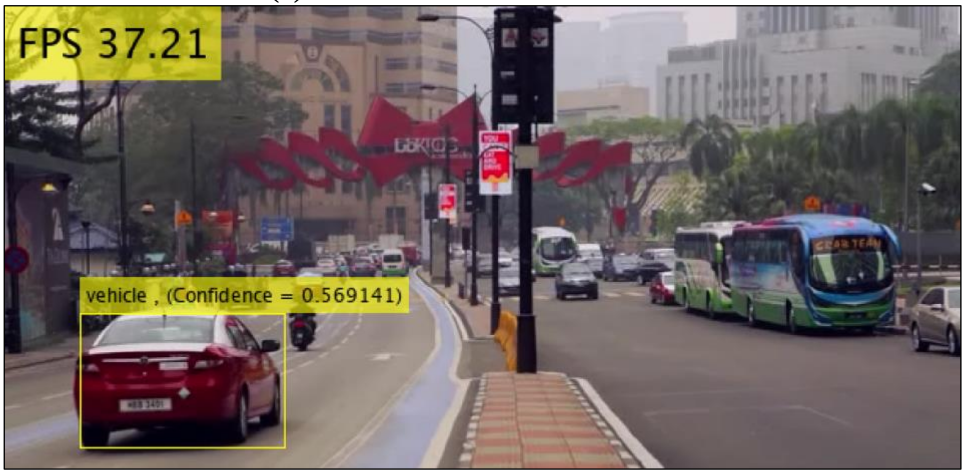
For the sample KL traffic scene data, YOLOv3 results with optimised pre-trained networks as a backbone are provided in Table 10, along with the frame rate per second (FPS), AP-50, and AP-75.

Table 10. YOLOv3 - KL Traffic Scene Results

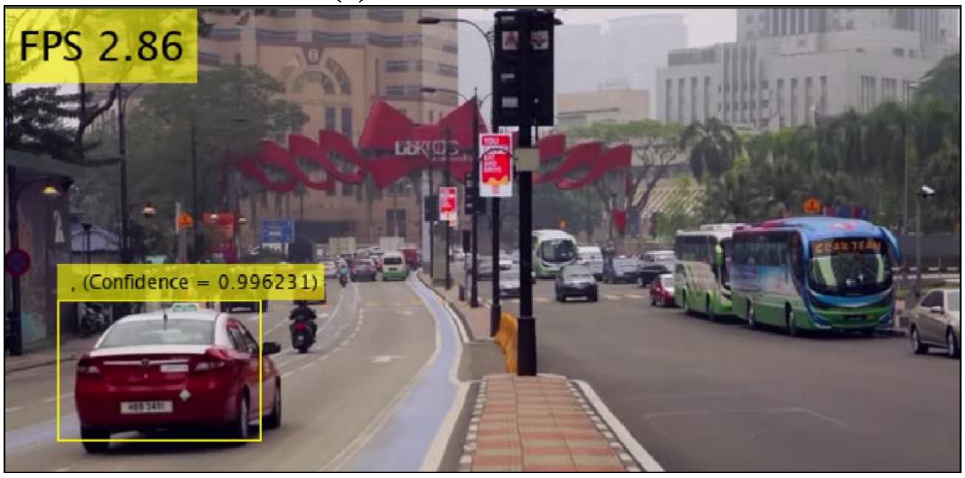
Sl. No	Model	Backbone	FPS	AP_{50}	AP_{75}
1.	YOLOv3	dNet-C53	2.91	0.97	0.18
2.	YOLOv3	dNet-CFC	2.92	0.96	0.26
3.	YOLOv3	dNet-CSN	2.91	0.98	0.20
4.	YOLOv3	dNet-FIN	2.88	0.96	0.18
5.	YOLOv3	dNet-CLR	2.70	0.97	0.16
6.	**YOLOv3**	**dNet-CIN**	**3.21**	**0.97**	**0.26**



(a) Faster R-CNN with ResNet50



(b) SSD with ResNet50



(c) YOLOv3 with dNet-CIN
(ClippedReLU + Switchable Normalisation)

Figure 8. Comparison of Confidence Scores for Vehicle Detection

The model that maintained a balance between FPS and mAP-50 was chosen since practically all of the YOLOv3 models received nearly equal mAP-50. Finally, the model with dNet-CIN as a backbone that received the highest confidence scores, and FPS was selected as shown in Figure 8.

In terms of FPS and mAP-50, the YOLOv3 models with the two best performing CNNs as backbones are compared, Faster R-CNN, and SSD. The results are shown in Table 11, whereas the SSD model shows a massive variation between FPS and mAP-50. Despite the model's high speed, its average precision is far below that of other models. So, when YOLOv3 is compared to other models, it performs well using an optimised dNet-CIN as the backbone, with an FPS of 3.21 and a mAP-50 of 97%.

Table 11. Comparison of YOLOv3 Models using KL Traffic Data

Sl. No.	Model	Backbone	FPS	AP_{50}
1.	Faster R-CNN	ResNet-50	2.84	0.94
2.	SSD	ResNet-50	36.07	0.16
3.	YOLOv3	Darknet-53	2.75	0.94
4.	YOLOv3	dNet-CFC	2.92	0.96
5.	**YOLOv3**	**dNet-CIN**	**3.21**	**0.97**

The current study proves that transfer learning can be useful for various computer vision problems, especially ones with small datasets. The primary goal of this study was to optimise Darknet-53 based on our previous study findings to improve the YOLOv3 model. We conducted the experiment in MATLAB R2021a, where the complete CNN architecture can be seen, which helped us in the selection of freezing the initial layers. Darknet-53 optimisation using MATLAB created a different perspective in image classification to enhance the speed and accuracy of a CNN architecture. With the availability of proper datasets, CNN models have the capability to take medical imaging technology further, providing a higher level of automation in medical imaging, including image processing and analysis. The improved YOLOv3 object detection model provided a better prospect for detecting vehicles with different viewpoints even with the small amount of training data. With a huge amount of data and proper training, the YOLOv3 model can be implemented with webcams for real-time traffic monitoring and other relevant object detection.

5. Conclusion and Future Work

We conducted a study to investigate the performance of five pre-trained networks: SqueezeNet, GoogleNet, ShuffleNet, Darknet-53, and Inception-V3, employing various epochs, learning rates, and mini-batch sizes based on transfer learning. The initial convolutional layers of selected pre-trained CNNs were frozen to keep the weights intact, and the final layers of the pre-trained CNN models were replaced either with a fully connected layer or a convolutional layer, and a new classifier replaced the classification layer. A confusion matrix was used to evaluate each of the models after they had been trained, and in terms of LR, Darknet-53 delivered impressive results with LR-0.0001, achieving a maximum accuracy of 94.70%. The fact that Darknet-53 was the only CNN that produced an error (out of memory) over batch-64 is most likely due to GPU memory shortages. Darknet-53 was optimised using switchable normalisation techniques to address the memory issue. The optimised Darknet-53 increased the validation accuracy, reducing the training time considerably, and the GPU memory issue was rectified as well. The Darknet-53 optimisation yielded a combination of new results, which were then used as a backbone for improving the YOLOv3 object detection model. The YOLOv3 model with dNet-CFC as a backbone provided the highest mAP of 86% for the vehicle dataset. YOLOv3 with dNet-CIN as the backbone delivered the best results for the KL traffic data, with an FPS of 3.21 and a mAP of 97%.

For the Darknet-53 optimisation process, only a few activation functions and switchable normalisation by updating the initial layers of batch normalisation layers with instance and final layers with layer normalisation were tested. In the near future, other activation functions will be prioritised, and group normalisation techniques will be considered for experimental work. In order to further optimise Darknet-53, fine-tuning the weights and other parameters will be prioritised. A diverse dataset will be focused on and prioritised in the future to begin the evaluation and testing of the YOLOv3 object detection model. Two detection heads were chosen for effective object detection in the improved YOLOv3 model. It can also be explored with a single or multiple detection heads depending on the size of the object and the dataset.

References

[1] K. G. Shreyas Dixit, M. G. Chadaga, S. S. Savalgimath, G. Ragavendra Rakshith, and M. R. Naveen Kumar, "Evaluation and evolution of object detection techniques YOLO and R-CNN," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2 Special issue 3, pp. 824–829, 2019, doi: 10.35940/ijrte.B1154.0782S319.

[2] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 years." 2019.

[3] T. Jing, X. Luo, T. Yang, and K. Kita, "An Object Detection System Based on YOLO in Traffic Scene," *2018 IEEE 4th Int. Conf. Comput. Commun. ICCC 2018*, pp. 1532–1536, 2018, doi: 10.1109/CompComm.2018.8780944.

[4] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," *Comput. Electron. Agric.*, vol. 161, no. October 2017, pp. 272–279, 2019, doi: 10.1016/j.compag.2018.03.032.

[5] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li, "Computer vision algorithms and hardware implementations: A survey," *Integration*, vol. 69, no. June, pp. 309–320, 2019, doi: 10.1016/j.vlsi.2019.07.005.

[6] A. M. Algorry, A. G. García, and A. G. Wofmann, "Real-Time Object Detection and Classification of Small and Similar Figures in Image Processing," *Proc. - 2017 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2017*, pp. 516–519, 2018, doi: 10.1109/CSCI.2017.87.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[8] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[9] F. Yang, W. Choi, and Y. Lin, "Exploit All the Layers: Fast and Accurate CNN Object Detector with Scale Dependent Pooling and Cascaded Rejection Classifiers," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2129–2137, 2016, doi: 10.1109/CVPR.2016.234.

[10] G. Chandan, A. Jain, H. Jain, and Mohana, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV," *Proc. Int. Conf. Inven. Res. Comput. Appl. ICIRCA 2018*, no. Icirca, pp. 1305–1308, 2018, doi: 10.1109/ICIRCA.2018.8597266.

[11] Z. Yi, S. Yongliang, and Z. Jun, "An improved tiny-yolov3 pedestrian detection algorithm," *Optik (Stuttg).*, vol. 183, no. February, pp. 17–23, 2019, doi: 10.1016/j.ijleo.2019.02.038.

[12] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[13] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[14] P. Ren, W. Fang, and S. Djahel, "A novel YOLO-Based real-time people counting approach," in *2017 international smart cities conference (ISC2)*, 2017, pp. 1–2.

[15] S. R. Alvar and I. V. Bajic, "MV-YOLO: Motion vector-aided tracking by semantic object detection," *2018 IEEE 20th Int. Work. Multimed. Signal Process. MMSP 2018*, 2018, doi: 10.1109/MMSP.2018.8547125.

[16] Z. Xu, H. Shi, N. Li, C. Xiang, and H. Zhou, "Vehicle detection under UAV based on optimal dense YOLO method," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, 2018, pp. 407–411.

[17] V. Thakkar, S. Tewary, and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks - A comparative study with CIFAR-10 data," *Proc. 5th Int. Conf. Emerg. Appl. Inf. Technol. EAIT 2018*, 2018, doi: 10.1109/EAIT.2018.8470438.

[18] P. M. Radiuk, "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets," *Inf. Technol. Manag. Sci.*, vol. 20, no. 1, pp. 20–24, 2018, doi: 10.1515/itms-2017-0003.

[19] A. Kuznetsova *et al.*, "The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale," *Int. J. Comput. Vis.*, vol. 128, no. 7, pp. 1956–1981, 2020, doi: 10.1007/s11263-020-01316-z.

[20] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do CIFAR-10 Classifiers Generalize to CIFAR-10?,” pp. 1–25, 2018, [Online]. Available: http://arxiv.org/abs/1806.00451.

[21] Q. H. Nguyen *et al.*, “Influence of data splitting on performance of machine learning models in prediction of shear strength of soil,” *Math. Probl. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/4832864.

[22] A. Krizhevsky, G. Hinton, and others, “Learning multiple layers of features from tiny images,” 2009.

[23] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, “Data preprocessing for supervised leaning,” *Int. J. Comput. Sci.*, vol. 1, no. 2, pp. 111–117, 2006.

[24] X. Zheng, M. Wang, and J. Ordieres-Meré, “Comparison of data preprocessing approaches for applying deep learning to human activity recognition in the context of industry 4.0,” *Sensors (Switzerland)*, vol. 18, no. 7, 2018, doi: 10.3390/s18072146.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.

[26] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[27] Y. Wu and Kaiming He, “Group Normalization – Facebook Research,” *Eccv*, no. Figure 1, 2018, [Online]. Available: https://research.fb.com/publications/group-normalization/.

[28] P. Luo, J. Ren, Z. Peng, R. Zhang, and J. Li, “Differentiable learning-to-normalize via switchable normalization,” *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–19, 2019.

[29] P. Luo, R. Zhang, J. Ren, Z. Peng, and J. Li, “Switchable Normalization for Learning-to-Normalize Deep Representation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 2, pp. 712–728, 2021, doi: 10.1109/TPAMI.2019.2932062.

[30] J. S. Kumar, S. Anuar, and N. H. Hassan, “Transfer Learning based Performance Comparison of the Pre-Trained Deep Neural Networks,” vol. 13, no. 1, 2022.

[31] S. Kumaresan, K. S. J. Aultrin, S. S. Kumar, and M. D. Anand, “Transfer Learning with CNN for Classification of Weld Defect,” *IEEE Access*, vol. 9, pp. 95097–95108, 2021, doi: 10.1109/ACCESS.2021.3093487.

[32] Y. Shi, Y. Li, Y. Zhang, Z. Zhuang, and T. Jiang, “An Easy Access Method for Event Recognition of Φ-OTDR Sensing System Based on Transfer Learning,” *J. Light. Technol.*, vol. 39, no. 13, pp. 4548–4555, 2021, doi: 10.1109/JLT.2021.3070583.

[33] M. A. F. Fusion, M. Zhang, S. Xu, W. Song, Q. He, and Q. Wei, “Lightweight Underwater Object Detection Based on YOLO v4,” pp. 1–22, 2021.