Incremental Training for Image Classification of Unseen Objects

Harshil Jain[1] and S.K.Nandy[2]

[1] Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India

[2] CAD Laboratory, Department of Computational and Data Sciences, IISc Bengaluru, India

jain.harshil@iitgn.ac.in, nandy@iisc.ac.in

Abstract

Object detection is a computer vision technique for locating instances of objects in images or videos. It basically deals with the detection of instances of semantic objects of a certain class in digital images and videos. It is a spine of a lot of practical applications of computer vision including image retrieval, self-driving cars, face recognition, object tracking, video surveillance, etc. Hence, object detection is significantly encompassing many fields in today's world. Object detection can be achieved through traditional machine learning approaches which are histogram of oriented gradients (HoG) or scale-invariant feature transform (SIFT) features and also through various deep learning approaches which include two broad categories. First is an architecture which uses two neural networks which includes region proposals (R-CNN, Fast R-CNN & Faster R-CNN) & second is single shot detectors which includes You Only Look Once (YOLO) and Single Shot MultiBox Detector (SSD). YOLO and SSD are way faster than RCNN and its derivatives. YOLO basically uses Darknet for feature extraction followed by convolutional layers for object localization while SSD uses VGG-16 for feature extraction. Though the problem of object detection is gaining the attention of the research community, most of the works have concentrated on improving current object detection algorithms. Detection of objects on unseen classes for which the networks were never trained has been overlooked. In this work, an attempt has been made to understand the YOLO architecture and answer various questions related to it and also to improve the existing single shot detectors like YOLO and SSD to classify unseen classes in real time by incremental learning. This can prove very robust as it is very difficult retrain these huge convolutional networks as and when new classes are added, that too in real time.

Keywords: *Object Detection, Single Shot detectors, YOLO, Convolutional Neural Networks, Incremental Learning, Computer Vision*

1 Introduction

1.1 Background

Humans have the ability to glance at an image and tell what objects are there in that image and where they are situated. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

Current object detection algorithms take a classifier for that object and evaluate it at various locations and scales in a test image. Some systems use a sliding window approach - where a particular size window is run throughout the image at specific strides to check where the object might lie. Recent approaches like Regional - CNN use region proposals where they propose a region using image segmentation to propose probable regions of the location of the object and then pass them through convolutional layers for proper localization and detection. These algorithms take quite a lot of time to come up with the bounding boxes and the classes, the object belongs to. This is because it passes through the huge convolutional network many times. Hence, these detectors cannot be used to detect objects in real time because of very high latency.

You Only Look Once (YOLO) algorithm is very much different from all of these. It reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. It only runs a single pass of the image through the convolutional network and predicts where and what the objects are. Because of this, YOLO algorithm can be used to detect objects in real time and it also outperforms traditional object detection methods.

1.2 Statement of Problems

- Investigating confidence score of objects belonging to unseen classes in YOLO.
- Figuring out whether background of images had an effect on YOLO training.
- Checking whether the size of ground truth bounding boxes matters in training YOLO algorithm.
- Training YOLOv1 on Tensorflow
- Checking whether incremental training is possible for classification on CIFAR-10 dataset on VGG16 network.
- Classification of unseen objects on VGG16 network

1.3 Objectives of Research

Collecting and cleaning datasets, annotating images for YOLO based experiments, segregating seen and unseen classes from CIFAR-10 for incremental training, training models and evaluating the results.

1.4 Scope

The present work work involves experimenting with YOLO and mainly on incremental training for unseen classes for classification on the VGG16 network. This thing can be extended for object localization of unseen classes by incremental training.

2 Literature Review

2.1 Information

2.1.1 YOLOv1

YOLOv1 is a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

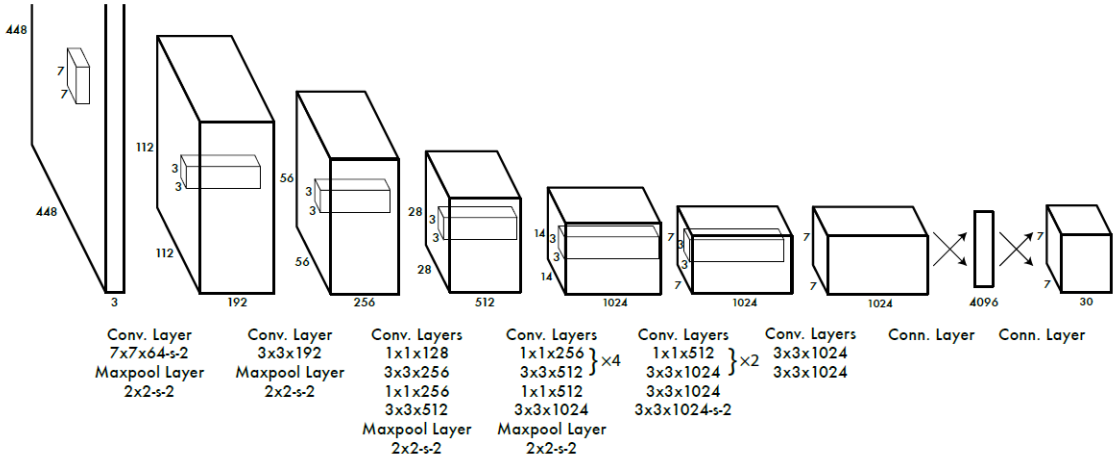


Figure 1: **The architecture has 24 convolutional layers followed by 2 fully connected layers. The model is pretrained on the ImageNet classification task at half the resolution** (224×224 **input image) and then double the resolution for detection.**

Network Design: The model has 24 convolutional layers followed by 2 fully connected layers (the Darknet framework). The convolutional layers are mainly responsible for feature extraction while the fully connected layers mainly predict the output probabilities and coordinates. The final output is $7 \times 7 \times 30$ tensor.

The model divides the input image into an $S \times S$ grid. If the centre of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell has B bounding boxes. The algorithm predicts confidence scores for each of these boxes. Confidence score is a measure of accuracy of prediction as to whether there is an object in the box and if there is how accurately the bounding box has been drawn.

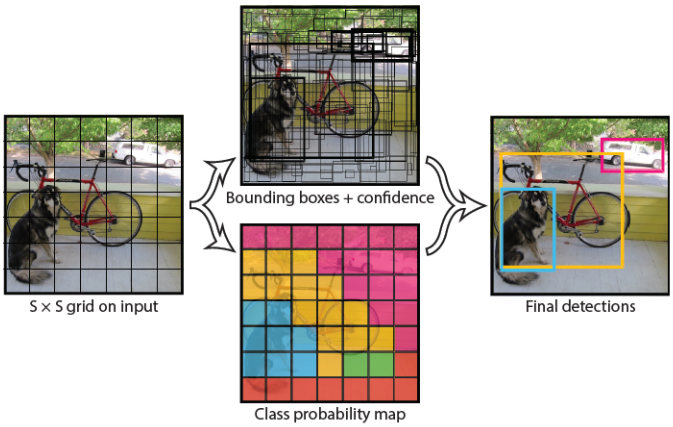


Figure 2: **The image is divided into an** $S \times S$ **grid and for each grid cell predicts** B **bounding boxes, confidence for those boxes, and** C **class probabilities. These predictions are encoded as an** $S \times S \times (B * 5 + C)$ **tensor.**

Each bounding box has 5 predictions $-x, y, w, h$ and confidence score. The (x, y) are the co-ordinates of the centre of the box relative to the grid cell and w and h are with respect to the entire image. Each grid cell also has C conditional class probabilities which represents the probability of the centre of the object falling in that particular grid cell. The output tensor should have the dimensions $(S \times S \times (B * 5 + C))$. YOLOv1 divided the image into 7×7 grid cell and there were 2 bounding boxes per grid cell. Since YOLO was evaluated on PASCAL VOC which has 20 classes, we have $S = 7$, $B = 2$ and $C = 20$. The final prediction is a $7 \times 7 \times 30$ tensor.

Loss Function used for training:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [(\sqrt{w}_i - \sqrt{\hat{w}}_i)^2 + (\sqrt{h}_i - \sqrt{\hat{h}}_i)^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{S^2} I_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

where

I_i^{obj} denotes if object appears in cell, I_{ij}^{obj} denotes jth bounding box in cell i is responsible for prediction, $\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i, $\hat{C}_i$ is the box confidence score of the box j in cell i and I_{ij}^{noobj} is the complement of I_{ij}^{obj}. [Here, we fix $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$]

2.1.2 YOLOv2

YOLOv2 brought a lot of changes to YOLOv1. It divides the image into 13×13 grid cells and it removed the fully connected layers. The final output here is a $13 \times 13 \times [5 \times (4+1+20)]$ tensor for the 4 bounding box offsets, 1 confidence score per box, and 20 class predictions. Following are the major changes in YOLOv2 compared to YOLOv1.

- **Batch Normalization**

Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization. By adding batch normalization on all of the convolutional layers in YOLO an improvement in mAP is observed. Batch normalization also helps regularize the model. Batch normalisation prevents overfitting by adding regularization and dropouts can be eliminated.

- **High resolution classifier**

For YOLOv2, the classification network is fine tuned at the full 448×448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high resolution classification network gives us an increase of almost 4% mAP.

- **Convolution with anchor boxes**

Anchor boxes are predefined bounding boxes which are used as guidelines by the network to predict offsets for bounding boxes from the anchor box under consideration.

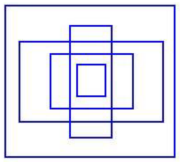


Figure 3: **An illustration where 5 anchor boxes of different shapes and sizes are selected for the model**

Instead of predicting 5 arbitrary boundary boxes, we predict offsets to each of the anchor boxes above. If we constrain the offset values, we can maintain the diversity of the predictions and have each prediction focuses on a specific shape. So the initial training will be more stable.

$$b_x = \sigma t_x + c_x$$
$$b_y = \sigma t_y + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

where t_x, t_y, t_h and t_w are outputs of YOLO algorithm, c_x and c_y is the top left corner of the grid cell of the anchor, p_w and p_h are the width and height of anchor and b_x, b_y, b_h and b_w are the co-ordinates of the predicted bounding box.

• **Dimension clusters**

YOLOv2 uses k-means clustering on the ground truth boxes of the training dataset to pick the k anchor boxes instead of handpicking the anchor boxes. $k = 5$ is chosen as a good trade off between model complexity and high recall.

Here is the YOLOv2 architecture which mainly uses Darknet-19 framework.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	$416 \times 416 \times 32$
Maxpool		$2 \times 2/2$	$208 \times 208 \times 32$
Convolutional	64	3×3	$208 \times 208 \times 64$
Maxpool		$2 \times 2/2$	$104 \times 104 \times 64$
Convolutional	128	3×3	$104 \times 104 \times 128$
Convolutional	64	1×1	$104 \times 104 \times 64$
Convolutional	128	3×3	$104 \times 104 \times 128$
Maxpool		$2 \times 2/2$	$52 \times 52 \times 128$
Convolutional	256	3×3	$52 \times 52 \times 256$
Convolutional	128	1×1	$52 \times 52 \times 128$
Convolutional	256	3×3	$52 \times 52 \times 256$
Maxpool		$2 \times 2/2$	$26 \times 26 \times 256$
Convolutional	512	3×3	$26 \times 26 \times 512$
Convolutional	256	1×1	$26 \times 26 \times 256$
Convolutional	512	3×3	$26 \times 26 \times 512$
Convolutional	256	1×1	$26 \times 26 \times 256$
Convolutional	512	3×3	$26 \times 26 \times 512$
Maxpool		$2 \times 2/2$	$13 \times 13 \times 512$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	512	1×1	$13 \times 13 \times 512$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	512	1×1	$13 \times 13 \times 512$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	1024	3×3	$13 \times 13 \times 1024$
Convolutional	125	1×1	$13 \times 13 \times 125$

Table 1: The YOLOv2 architecture based on Darknet-19 framework

2.1.3 YOLOv3

YOLOv3 improvises the previous versions in the following aspects:

- **Class Prediction**

Many classifiers make the assumption that classes are mutually exclusive. But, YOLOv3 does multi-label classification. YOLOv3 replaces the softmax function with independent logistic classifier. Classes like 'pedestrian' and 'man' are not mutually exclusive and hence sum of confidences can be greater than 1. Also, the loss is changed from mean squared error to binary cross entropy loss.

- **Bounding Box Predictions**

YOLOv3 assigns only 1 bounding box anchor for each ground truth object. YOLOv3 predicts an objectness score for each bounding box using logistic regression. This value is 1 if the bounding box anchor overlaps a ground truth object to a greater extent than any other anchor. For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost.

- **Predictions Across Scales**

YOLOv3 predicts boxes at 3 different scales. It extracts features from those scales like Feature Pyramid Network. It predicts 3 boxes at each scale so the tensor is $N \times N \times [3 \times (4+1+80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

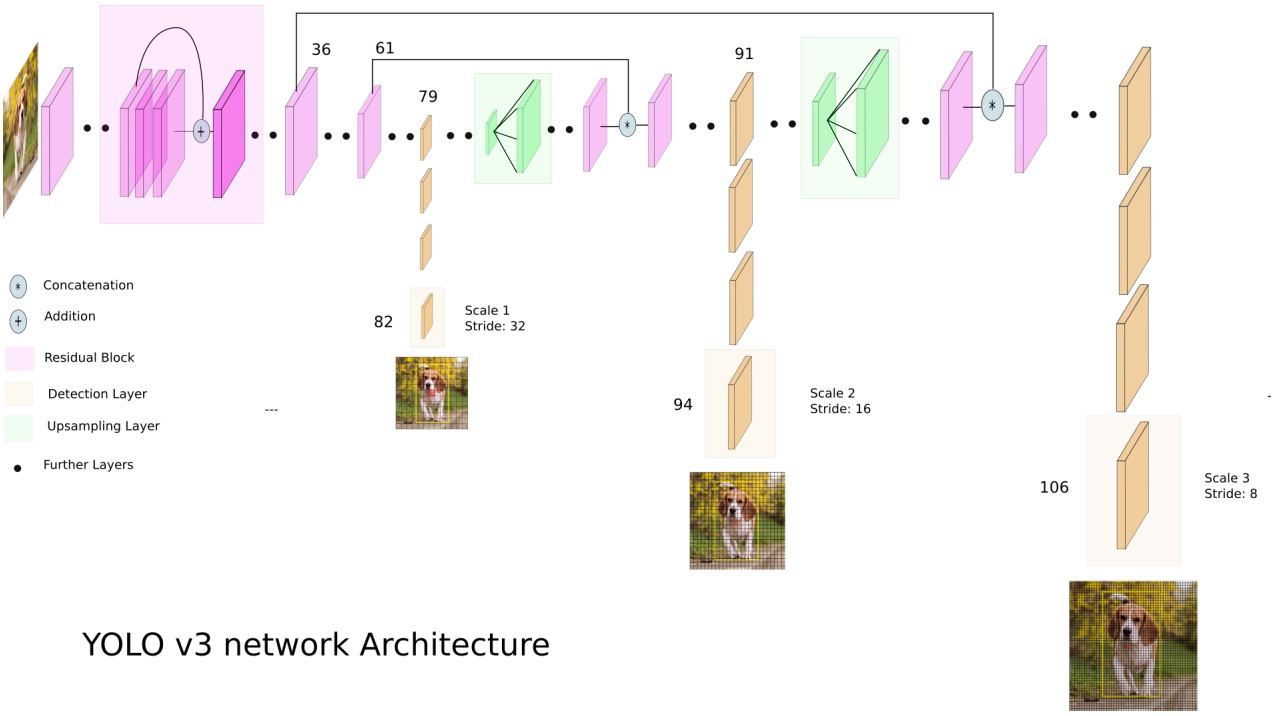


Figure 4: **Multi Scale Prediction used in YOLOv3**

- **Feature Extractor**

YOLOv3 uses **Darknet-53** feature extractor which has 53 convolutional layers. This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 5: **Darknet-53**

2.1.4 Non Maximal Suppression

It is an algorithm which removes bounding boxes which predict the same class and overlap to a great extent during detection. This algorithm uses the x,y,w,h generated by the bounding boxes to calculate the extent to which they overlap with other bounding boxes. It first rejects all the bounding boxes whose confidence score is less than the threshold. After this it picks the bounding box with max probability and rejects all other bounding box with an IOU (Intersection over Union) $IOU = \frac{AreaofOverlap}{AreaofUnion}$ greater than a particular threshold.

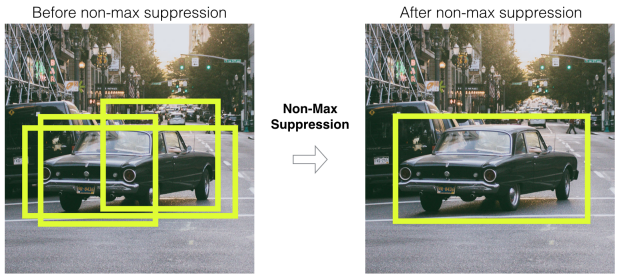


Figure 6: **Non Maximal Suppression**

2.2 Summary

All details pertaining to various versions of YOLO algorithm have been summarised. YOLO algorithm can be used to predict only images containing trained classes. Incremental training for prediction of unseen classes (which can be used for applications like video surveillance) is important because it is practically impossible to retrain the entire network as and when new classes are added.

3 Experiments Related to YOLO Architecture

3.1 Confidence Score of New Objects

The definition of confidence score stated in the YOLO paper was ambiguous, it did not state clearly whether confidence score is high if an object is present which belongs to the classes on which it was trained on or not. The YOLO algorithm outputs the actual confidence score multiplied by the class probability as the final confidence score while predicting an object. I modified the code it to only output the actual confidence score. The results obtained on classes for which YOLO wasn't trained are as follows. (Here we are able to see the bounding boxes because the confidence threshold was reduced to 0.01 just to indicate the boxes. The actual YOLO algorithm would output nothing for these images.)

Figure 7: **Confidence Score of New Objects**

This experiment concludes that the confidence score is high only for the objects for which YOLO algorithm was trained on.

3.2 Effect of Background on YOLO training

On going through the YOLO architecture, we wanted to check whether background has an effect on how YOLO was trained. So, to check this out we performed the following experiment.

3.2.1 Dataset Collection and Data Augmentation

We handpicked a dataset of 200 images each of dogs and cats in light background (mostly white). We then used data augmentation to horizontally flip these 400 images and create another set of 400 images. Thus, we had a dataset of 800 images in total, 400 each for cats and dogs.

3.2.2 Labelling the Data

We then labelled the 800 images by hand using this tool. There were only two classes - cat and dog for simplicity.

3.2.3 Training

We trained the model on yolov3-tiny.cfg which is a tiny version of YOLOv3 on these 800 images on NVIDIA GeForce GTX 1080Ti GPU. We then used the .weights file after 10,000 iterations when the loss was around 0.14.

3.2.4 Testing images on darker background

We then tested images of dogs and cats with dark backgrounds and saw that the model was mostly able to predict even with darker background.

Figure 8: **Prediction on darker background**

This shows that YOLO training doesn't depend on the background of images it was trained on.

3.3 Effect of Bounding Box Size on training

An experiment was made to check whether ground truth bounding boxes' size made an effect on YOLO training.

3.3.1 Dataset Collection and Data Augmentation

We used the dataset used in the previous experiment and also 900 images (for each class) from Kaggle dataset for pets. This made the size of the dataset to 2600 images (1300 images each of classes cats and dogs).

3.3.2 Dataset Preprocessing

A python script was run to take only those images for training which had the ground truth bounding box having both length and breadth greater than 0.5 times the length and breadth of the entire image.

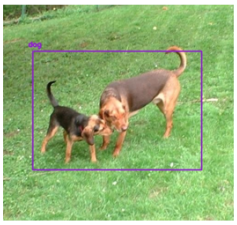
3.3.3 Training

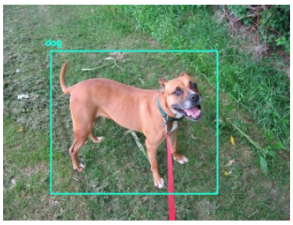
We trained the model on yolov3.cfg on these 2600 images on NVIDIA GeForce GTX 1080Ti GPU. We then used the .weights file after 23,000 iterations.

3.3.4 Testing

We then tested on new images which had an length and breadth greater than 0.5 times the image and YOLO was able to detect dogs and cats in those images with very good accuracy. However, on giving test images with ground truth boxes having length and breadth less than 0.5, either the bounding box was not accurate to exactly enclose the object or YOLO didn't draw the bounding box at all [Figures 1 and 2]. For those images for which it drew the bounding box, it gave a bigger bounding box having aspect ratio greater than 0.5 as shown [Figures 3 and 4]. However as expected for images with ground truth box having length and breadth greater than 0.5 times the image dimension, it was able to draw bounding boxes accurately. [Figures 5 and 6]. Only 9.37% of the test images threw up some bounding box.

Figures 1 and 2: No prediction





Figures 3 and 4: Prediction with a bigger bounding box (the ground truth boxes cover less than 25% of the image)

Figures 5 and 6: Accurate prediction (the ground truth boxes cover greater than 25% of the image)

This shows that the size of the ground truth bounding boxes used for YOLO have an effect on the training.

3.4 Training YOLOv1 on Tensorflow

YOLOv1 was trained purely on Tensorflow on the PASCAL VOC dataset. Here is the code which was implemented. Transfer Learning was employed. Darknet weights trained for classification were used to initialise the first 20 convolutional layers of YOLOv1 and the weights of fully connected were randomly initialised. We trained it for 2,30,000 iterations when the loss was around 1.61 on NVIDIA GeForce GTX 1080Ti GPU for around 2.5 days. It did give some mis predictions probably because the training was stopped at a loss of 1.61, to achieve better accuracy it should fall even lower at around 0.06.

Here are some of the predictions on sample images.

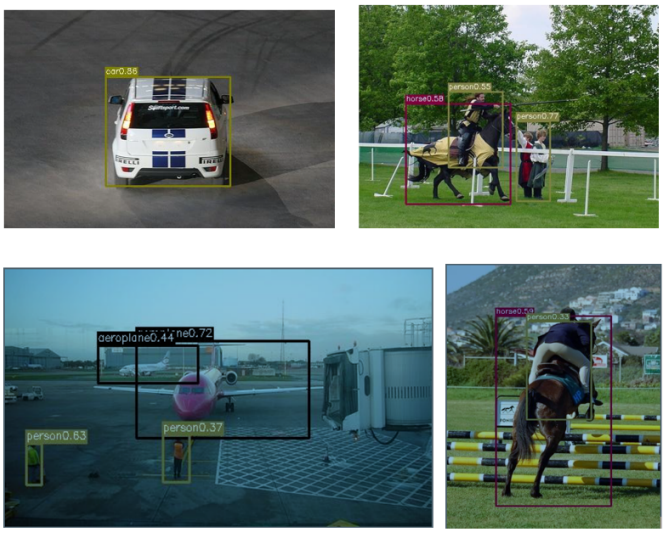


Figure 9: **Predictions made by YOLOv1 Tensorflow implementation**

We achieved **an mAP of 54.47%** with confidence threshold of 0.5 on VOC test dataset containing 2500 images.

Class	**AP**
Aeroplane	69.42%
TV Monitor	53.01%
Chair	31.67%
Sofa	56.15%
Motorbike	56.05%
Potted Plant	23.57%
Bottle	14.46%
Horse	73.05%
Sheep	36.81%
Dog	68.66%
Dining Table	55.59%
Person	56.65%
Train	78.93%
Cow	51.36%
Cat	75.78%
Car	63.08%
Bird	66.79%
Bicycle	46.39%
Boat	50.39%
Bus	61.58%

4 Methodology for Transfer Learning and Incremental Training

Experiments were performed to train CIFAR-10 dataset on the VGG16 network which consists of 13 convolutional layers, 3 fully connected layers and 5 max-pool layers as shown below. There is a ReLU activation function after every convolutional layer.

Figure 10: **VGG16 Network Architecture**

Several experiments were performed where,

- Hyper-parameters (learning rate, etc) were tweaked
- Different optimizers (adam, SGD, momentum) were used in back propagation
- Additional layers (batch normalization, dropout) were introduced for improving accuracy, reducing overfitting.

Only the key observations that have significant impact on the training procedure are reported.

4.1 Concepts and Methods

4.1.1 Transfer Learning

Transfer learning is the application of knowledge gained from completing one task to help solve a different, but related, problem. Training CIFAR10 from scratch, i.e., by initializing VGG16 with random initial weights, takes longer for the network to converge. Using NVIDIA GTX 1080Ti GPU, training from scratch should have taken 10-12 hours. But, pre-trained weights, from training VGG16 on ImageNet dataset, were used to initialize the training for CIFAR10. Hence, training takes only fewer iterations. Since the ImageNet and CIFAR10 datasets have different distributions, and the images are of different sizes, the network, initialized with ImageNet's weights, has to be tuned for CIFAR10 for producing considerable accuracy.

Transfer Learning can also be used in object detection. In Object Detection networks, feature extraction layers from classification networks can be appended with detection and regression layers. The complete Object Detection network can now use pre-trained weights for classification layers and randomly initialized weights for detection layers. This speeds up the training considerably. Between Adam and Momentum optimizers, either they produce similar accuracy or Adam outperforms Momentum when the latter hits a saddle region and stagnates.

4.1.2 Partial Learning followed by Incremental Training

Introducing batch normalization (BN) between all layers improves accuracy by eliminating co-variate shift that's introduced when the normalized image is passed through the layers. The network is expected to learn to normalize the images between the layers. Hence, BN introduces trainable parameters alongside the weights (convolutional/FC). The network co-optimizes the weights and the BN parameters. When only BN parameters were trained throughout the network (and not the weights), it still produces reasonable accuracy as we will see in the results. We use these weights for the next experiment as follows.

It is sufficient to train a part of the dataset on only the last few convolution layers and FC layers. It could be reasoned that the initial layers are mostly involved in extracting the features and the final layers are involved in regression. Not modifying the initial features also help in preventing over-fitting (to a dataset). Now, we do incremental training and train the entire dataset on the last 3 convolutional and fully connected layers.

4.2 Incremental Training for Unseen Classes

As and when new classes are added to the existing dataset, it is very inefficient to retrain the entire model again from the beginning. So, incremental training is done for new classes. This is tested on the CIFAR-10 dataset. The network is initially trained only for 8 classes out of the 10 classes excluding bird and truck. Then, the model is incrementally trained only the last 3 convolutional and last 3 fully connected layers for the two unseen classes bird and truck. It is tested on the test dataset containing images only of birds and trucks.

5 Results and Discussions

The graphs for Loss vs Iterations and Accuracy vs Iterations when the entire network is trained for 100% training set and tested for 100% test set is as follows for Adam and Momentum optimizers. The accuracy is 85% after about 1000 iterations for Adam Optimizer.

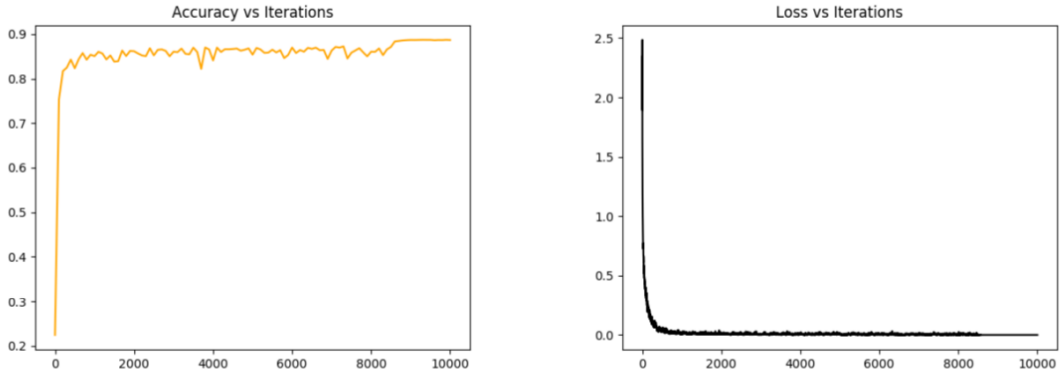


Figure 11: **Training entire network using Adam Optimizer**

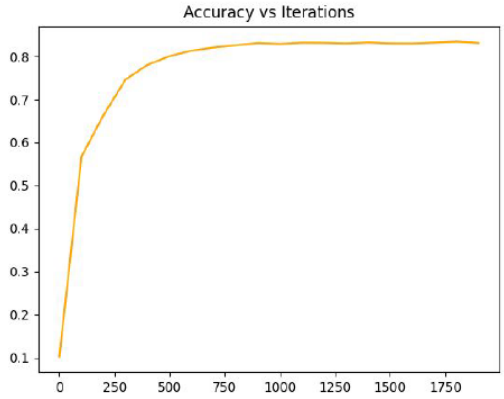


Figure 12: **Training entire network using Momentum Optimizer**

The Accuracy vs Iterations when only the parameters involved in batch normalization (γ and β) for Adam Optimizer is as follows:

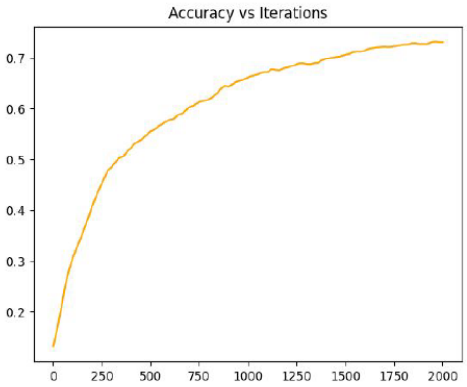


Figure 13: **Training BN parameters for entire network using Adam Optimizer**

The weights of the previous experiment are used for the following couple of experiments.

The graphs for Loss vs Iterations and Accuracy vs Iterations for Partial Learning i.e. the last 3 convolutional layers and the last 3 fully connected layers along with the BN parameters for all the layers only for 75% of the train dataset is as follows. The accuracy achieved is 80% after about 1000 iterations.

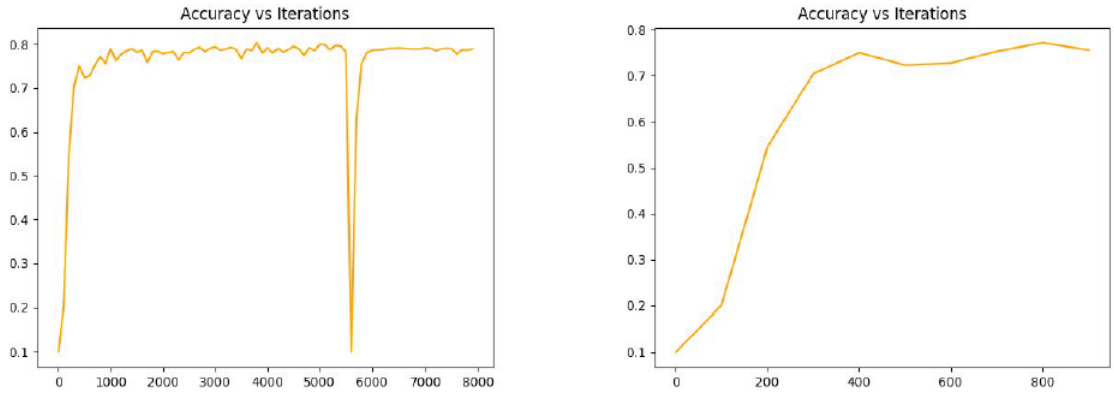


Figure 14: **Training 3 Conv + 3 FC and all BN parameters using Adam Optimizer on 75% train dataset [The image on the right is the magnified version of the left]**

The previous experiment was repeated with everything identical except that the BN parameters for only the last layers were trained and not for the entire network only for 75% of the train dataset. The results are as follows, The accuracy achieved is 80% after about 1000 iterations.

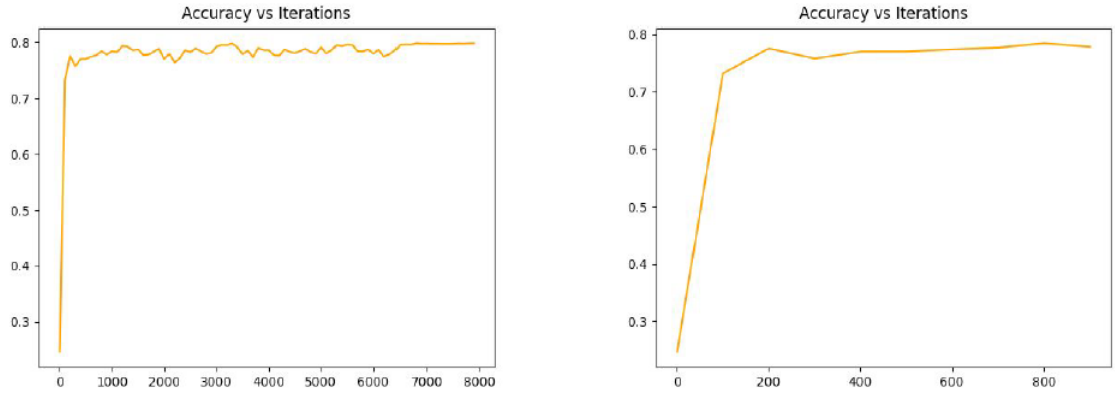


Figure 15: **Training 3 Conv + 3 FC and corresponding BN parameters using Adam Optimizer on 75% train dataset [The image on the right is the magnified version of the left]**

Finally, for incremental training, we incrementally train 100% of the train dataset using the last few layers. Here too, there are two variants, one with all BN parameters and the other with just corresponding BN parameters. Here we use the weights of previous experiment for initialisation. It achieves a decent accuracy of 82% after about 1000 iterations. This confirms that incremental training for the last few layers yields good accuracy for unseen objects.

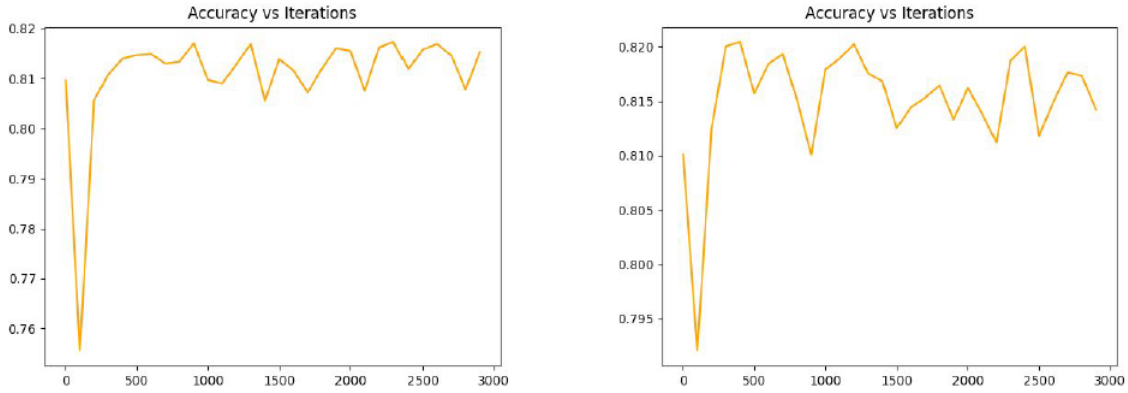


Figure 16: **Training 3 Conv + 3 FC and LEFT) all BN parameters RIGHT) corresponding BN parameters using Adam Optimizer on 100% train dataset**

As described in section 4.2, we train the entire network for 8 classes excluding birds and trucks and the results are as follows:

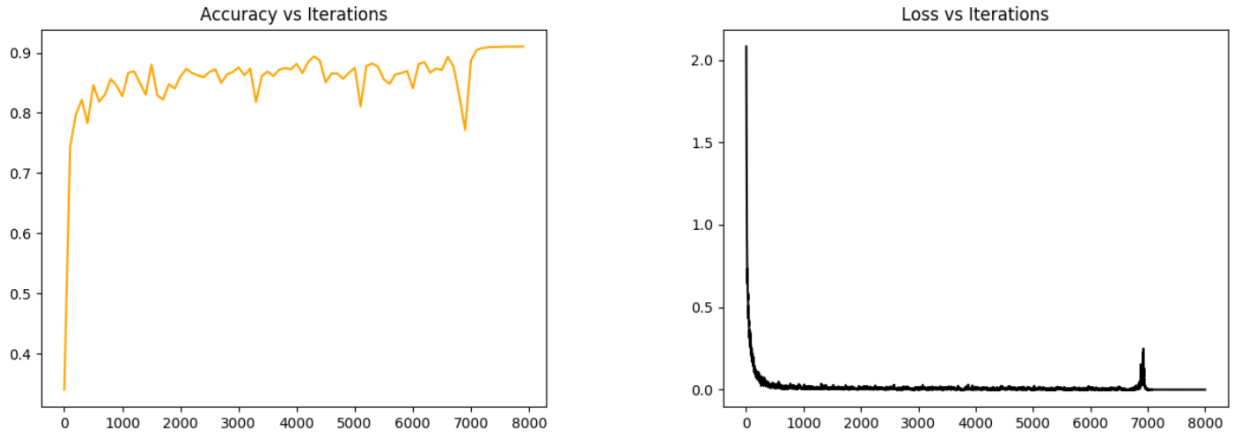


Figure 17: **Training entire network using Adam Optimizer on 8 classes**

Then, it is incrementally trained for the two unseen classes - birds and trucks on only the last 3 convolutional + 3 fully connected layers using the weights from the previous experiment and the results obtained are as follows:

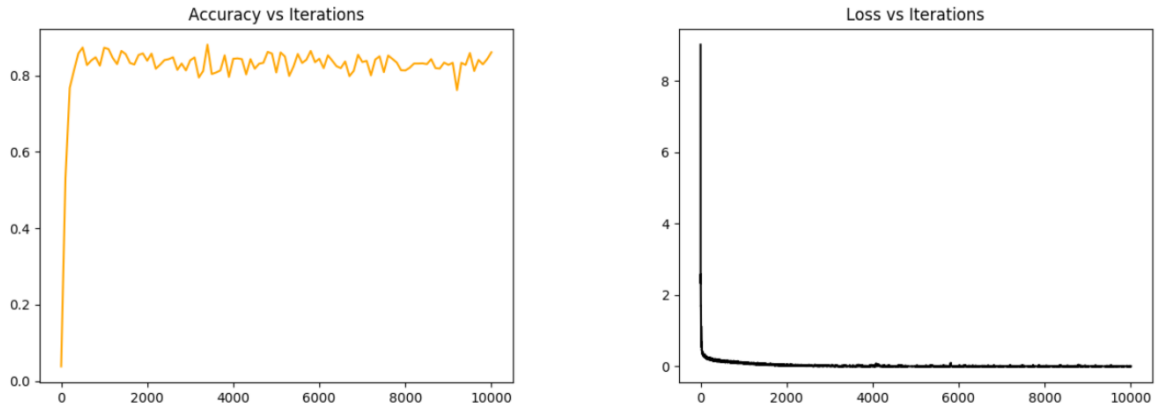


Figure 18: **Incrementally training the last few layers of the network using Adam Optimizer on 2 unseen classes**

6 Conclusions

Incremental training for object detection and localization of unseen objects can be very useful for real time scenarios, for example video surveillance cameras. It can also be used for online training of these new classes so that the model can be continuously retrained on the device itself. Apart from this, newer techniques need to be introduced to improve the overall accuracy. Nevertheless, these techniques prove very useful for instances like video surveillance.

7 References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.

[2] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517-6525. IEEE, 2017

[3] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement arXiv preprint arXiv:1804.02767, 2018.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. CoRR, abs/1312.6229, 2013.

[5] J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck. H. Zhang1 C.-C. Jay Kuo. Class-incremental Learning via Deep Model Consolidation arXiv preprint arXiv:1903.07864v2, 2019

[6] R. Istrate, A. Cristiano, I. Malossi, C. Bekas, D. Nikolopoulos. Incremental Training of Deep Convolutional Neural Networks arXiv preprint arXiv:1803.10232, 2018

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[9] J. Redmon. Darknet: Open source neural networks in C. http://pjreddie.com/darknet/, 2013-2016.

[10] Blog on YOLO: [Online] Available: https://machinethink.net/blog/object-detection/

[11] YOLOv1 Tensorflow. GitHub [Online] Available: https://github.com/MingtaoGuo/yolo_v1_v2_tensorflow

[12] Tzutalin. LabelImg. Git code (2015). Available: https://github.com/tzutalin/labelImg

[13] DW2TF, GitHub. [Online]. Available : https://github.com/jinyu121/DW2TF