DETECTING OUT OF DISTRIBUTION SAMPLES AND COUNTING MULTICLASS OBJECTS USING CONVOLUTIONAL NEURAL NETWORKS

A PROJECT REPORT SUBMITTED BY

MOHAMED SHAHEER (S/14/528)

to the

DEPARTMENT OF STATISTICS AND COMPUTER SCIENCE

in partial fulfillment of the requirement
for the award of the degree of

B.Sc. (Honours) in Computer Science

of the

UNIVERSITY OF PERADENIYA
SRI LANKA
2019

DECLARATION

I do hereby declare that the work reported in this project report was exclusively carried out by me under the supervision of Dr. Ruwan Nawarathna. It describes the results of my own independent work except where due reference has been made in the text. No part of this project report has been submitted earlier or concurrently for the same or any other degree.

Date: ……. …….

Signature of the Candidate

Certified by:

Supervisor: Dr. Ruwan Nawarathna

Date: ……. …….

Signature of the Supervisor

Head of the Department: Dr. Ruwan Nawarathna

Date: ……. …….

Signature of the Head of the Department

DETECTING OUT OF DISTRIBUTION SAMPLES AND COUNTING MULTICLASS OBJECTS USING CONVOLUTIONAL NEURAL NETWORKS

M. Shaheer

Department of Statistics and Computer Science,

University of Peradeniya, Sri Lanka.

Detecting out of distribution samples to avoid false confident prediction is one of the fundamental requirement for deploying a good system for object detection and classification in real world application. Using convolutional neural networks for detecting objects in same distribution as training distribution results in high accuracies, but these network with softmax classifiers result in overconfident false prediction for out of distribution samples. In this study we propose a methodology for detecting out of distribution samples which is applicable to any pre-trained softmax neural classifier. We compute a Mahalanobis distance based confidence score from a class conditional multivariate Gaussian distributions. Based on this confidence score we decide whether the test sample is out of distribution. Also we found that calibration techniques, input pre-processing and feature ensemble improves the accuracy. In Multiclass object detection an algorithm for counting multiclass objects and getting a class-wise count is another problem we have with object detection. We propose an algorithm for counting multiclass objects by creating a vector of objects for all the classes in the dataset with a count attribute which is incremented when an object is detected in that particular class. We have used YOLOv3 and Mask R-CNN object detection algorithms to validate the counting algorithm.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deep and sincere gratitude to my supervisor, Dr. Ruwan Nawarathne, for giving me the opportunity to do the research while providing necessary information and giving his invaluable guidance and supervision throughout this research amidst of his tight schedule. His motivation, sincerity and dynamism provided me with great enthusiasm and passion to headway in my research. It was a great honor to work and study under his profound guidance enclosed with great insight. I would like to extend my heartfelt gratitude for his eminent support and I believe I am greatly indebted to his extraordinary endeavor and support given throughout my research work and in regard.

Also, I would like to thank the staff members of the Department of Statistics and Computer Science for sparing their valuable time and giving suggestions and advice to carry on the research.

I am extremely thankful to my parents and siblings for their care, sacrifices and prayers and continuing to support me to complete this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN: - Convolutional Neural Network

R-CNN: - Regional Convolutional Neural Network

YOLO: - You Only Look Once

IID: - Independent and identical distribution

OOD: - Out of Distribution.

MD: - Mahalanobis Distance.

CHAPTER 1

INTRODUCTION

The use of deep learning and convolutional neural networks for object detection and classification in computer vision based applications have been evolved and developed in a major scale during the past decade. The breakthrough of deep learning was in 2012 where Alex Krizhevsky, Ilya Sutskever and Geoff Hinton entered a submission [14] to ImageNet challenge that would halve the existing error rate to 16%. Since then there are many techniques introduced with a deep learning approach which resulted in promising results and high accuracies compared to all other methods. Due to this advancement, computer vision based applications became more popular from unlocking your mobile phone with face to self-driving cars. They became popular and useful in many domains to automate manual processes in an intelligent manner. There are three main application areas in deep learning such as object recognition, speech recognition and text understanding. There are many inspirational applications of deep learning like automatic machine translations, Recommendations, self-driving cars, image caption generation etc. With deep learning you can visualize patterns of huge data that a human brain cannot process. In general, deep learning is training networks with many layers. Deep learning methods have been significantly outperforming the existing methods in major computer vision and speech recognition tasks since 2010.

In the field of computer vision in order to perform object classification, detection and tracking with a deep learning approach we use convolutional neural networks. It is a deep learning architecture which is inspired by the structure of visual system. A convolutional neural network is able to successfully capture the special and temporal dependencies in an image by applying relevant filters. It has many layers such as convolutional layers, pooling layers and fully connected layers. In general, the convolutional layers and the pooling layers are extracting features whereas the fully connected layer does the classification. There are various architectures of convolutional neural networks available which have been the key in building algorithms which power AI in whole. Some of them are LeNet, AlexNet, VGGNet, GoogleNet, ResNet, DenseNet etc. FasterRCNN, YOLO, SSD are powerful algorithms used in object detection. They have different implementations which has its own advantages and disadvantages. With these algorithms multiclass object detection which means detecting multiple objects in a single frame and real time object tracking has become possible. This is

used in many real world applications like self-driving cars, security applications, health care applications etc.

Even though deep learning has solved many issues in object detection and gives us higher accuracies than any other previous methods still there are certain common problems which in general are still open for optimized solutions. Problems such as multimodal learning, negative class effect, multi class problem, transfer knowledge from one architecture another and using small datasets to train generalized models. These are theoretical problems in deep learning. For an instance if we consider the negative class problem, it is the result of giving false confident predictions to a statistically out of distribution sample. In deep learning we train the models with a large dataset so that the features are extracted and weights are optimized based on that particular dataset by minimizing the error. Detecting and counting multiple objects in the same frame is also considered as a complex problem.

1.1 Problem Statement

In Object detection using convolutional neural networks, training is done using a large dataset where the models learn by examples which means it optimizes the weights to minimize the loss function based on the features extracted in the convolutional and pooling layers. Once the model is trained properly it works well giving high accuracies, we can validate the model with a test dataset. Most of the times the training data and the test data is from the same domain. To any such good model when we input a statistically out of distribution sample from its training dataset the model ends up in giving false confident predictions. This is called the negative class problem, novelty and anomaly detection, out of distribution problem where all means the same problem. For an instance if we have a food detection system once we input a non-food item to the model the model should be able to detect it as a non-food rather than misclassifying it to one of the available food classes.

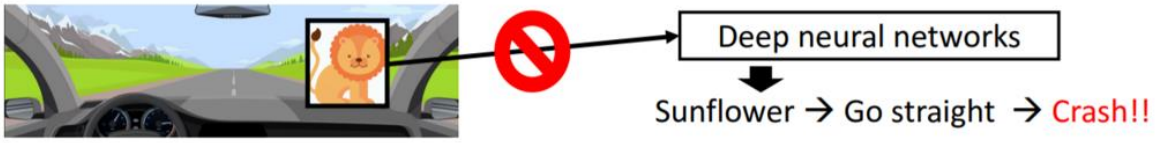

Figure 1.1: An example of an out of distribution problem where a self-driving car is misled by an out of distribution sample.

In Figure 1.1 it shows a self-driving car misclassifying an animal to a sunflower. Here the reason for the particular miss detection is, the network has no training samples of that particular animal so it predicts to a class which has the close feature relationship in its training labels. This can be serious problem in real world applications as shown in the figure 1.1 where a self-driving car ends up in a crash due to the wrong instructions given as a result of the miss detection.

Multiclass detection and counting is a complex problem in deep learning where we have to detect and count multiple objects in the same frame. There are powerful algorithms to achieve this task but multiclass counting is not yet implemented which can be useful in many real world applications. For an instance if we take a food detection system the plate might contain different food items in different quantities, the model should detect them separately and count the numbers separately.

The proposed system will address the out of distribution problem and how to apply that in a complex scenario like food detection. And also an algorithm for multiclass object detection and counting.

1.2 Objectives

There are two main objectives in this study. Developing a framework for detect false confident prediction or out of distribution problems in convolutional neural network models and perform multiclass counting with object detection algorithms.

1. A framework for detecting out of distribution samples for any pre-trained softmax neural network.
    - Application of detecting out of distribution samples in a complex domain like food detection.
2. Multiclass object detection and an algorithm for counting multiclass objects.
    - Implementing a counting algorithm for multiclass counting in a complex scenario with different algorithms and comparing the accuracies.

1.3 Approach

For detecting out of distribution samples in any pre-trained softmax neural network a confidence score is used.

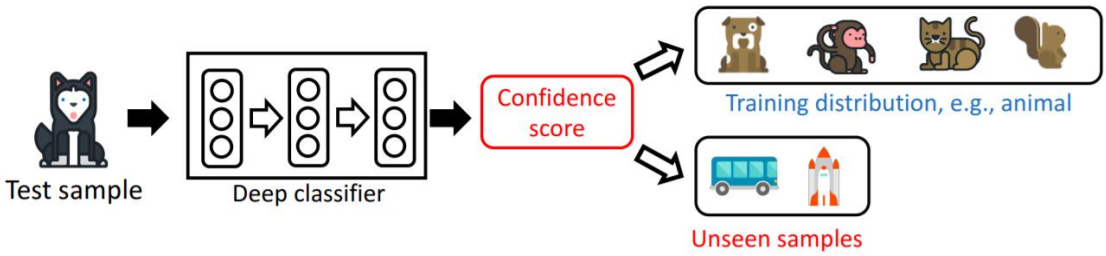

Figure 1.2: Approach for out of distributed problem.

Pre trained networks on different datasets are used for validating the method. ResNet and DenseNet trained on datasets like cifar-10, cifar-100, SVHN, tiny ImageNet are used with different combinations as in and out distributions. Then applying this method to a real world scenario in a complex domain like food detection. For this purpose, we use Food-101 dataset which has 101 different classes of food.

Multiclass object detection can be done using many powerful algorithms like YOLO, SSD, FasterRCNN etc. Each of these algorithms has its own advantages and disadvantages. Implementing YOLO and FasterRCNN which are considered as popular object detection algorithms on COCO dataset and implementing a counting algorithm with both of them to perform multiclass counting.

1.4 Outline of the thesis

The background study for the out of distribution problem and multiclass object detection is discussed in the next chapter which is followed by the literature review. In the fourth and fifth chapter a brief study of the methodology is presented for the out of distribution problem and the multiclass object detection and counting using convolutional neural network. Following the methodology, the results are presented for both problems along with the discussion. Finally, we have the references and the appendices of the study which concludes the thesis.

CHAPTER 2
BACKGROUND

2.1 Detecting out of distribution samples

In Deep Learning, a typical learning procedure would first split the available data in to train, validate and test sets. We train the models on the train set, validate the hyper parameters on the validation set, and demonstrate the performance of the model on the test set. But, how would a well-performing model on the test set function in real-world applications? If model A performs better than model B on our test set, does that mean model A would also be more suitable for real-world applications?

Statistical learning theory provides a framework to answer these inquiries through some general assumptions. For instance, if our test set is large enough and IID, and if the real-world data is IID too, we can relate the test set error to the real-world error with a high probability. From a theoretical standpoint, if the real-world data comes from another distribution than the test set, then there would be no way for us to relate the errors in the absence of more information, i.e., we do not have any idea how the model would perform on a new distribution.

This problem is perhaps more critical than it seems, especially in the deep learning age. Let us say we have a cat vs. dog convolutional network that has a 99% accuracy on our huge test dataset. We deploy our model into a pipeline, and now we rely on the output to perform a task. Given an image, our model predicts a cat with 100% probability. How confident are we that the input to our model is, in fact, a cat? We know through the adversarial attacks that in the presence of an adversary, we cannot trust anything our model says. Even if we assume there are no adversaries, we still would not have certainty that the input image is a cat. However, if we further assume that the input image is from the same distribution as the test set on which we had 99% classification accuracy, we could have some degree of confidence that the input image is a real cat. To be more accurate, we would know that if we were given an independently sampled set from the same distribution as the test set, our model would not make mistakes on more than a certain number of samples with a high probability. Determining whether an input is familiar (i.e., belongs to the population distribution of the training data), should be a pre-condition to prediction with deep learning models. If the sample is out-of-distribution, then the predictions are unreliable, and the error cannot be

bounded. Normally Multivariate Gaussian distribution is used in anomaly detection. And Mahalanobis distance between the test sample and the closest class conditional Gaussian distribution.

2.2 Multiclass object detection algorithms.

Faster R-CNN, Mask R-CNN and YOLO are popular approaches for object detection that are anchor based. Faster RCNN offers a region of interest for performing convolution and Mask R-CNN is a improved version of Faster R-CNN while YOLO does detection and classification at the same time.

2.2.1 Faster R-CNN and Mask R-CNN for object detection

Faster RCNN is an object detection architecture presented by Ross Girshick, Shaoqing Ren, Kaiming He and Jian Sun in 2015, and is one of the famous object detection architectures that uses convolution neural networks. Faster RCNN is composed of three parts. Convolutional layers, Region proposal and the classification.

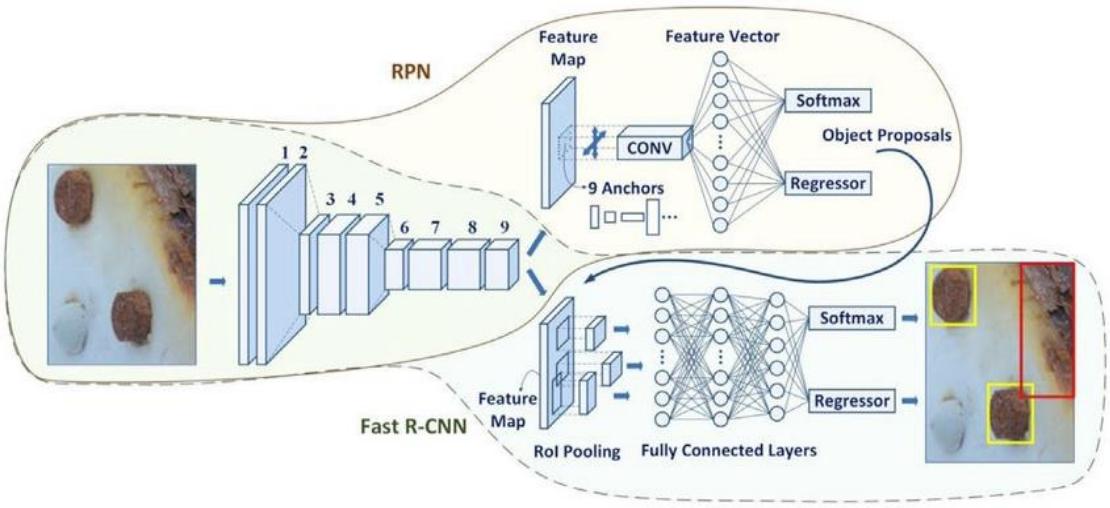

Figure 2.1: Three Main steps in faster RCNN

Convolutional Layers: In this layers we train filters to extract the appropriate features of the image. Convolution networks are generally composed of Convolution layers, pooling layers and a last component which is the fully connected layer for classification or detection. Region Proposal Network (RPN): RPN is small neural network sliding on the last feature map of the convolution layers and predict whether there is an object or not and also predict the bounding box of those objects. It uses n anchor boxes at each location.

Classes and Bounding Boxes prediction: Now we use another Fully connected neural networks that takes as an input the regions proposed by the RPN and predict object class (classification) and Bounding boxes (Regression). After Faster R-CNN Mask R-CNN was introduced which is an improved version of Faster R-CNN by including a mask predicting branch parallel to the class label and bounding box prediction. It only adds a small overhear to Faster R-CNN.

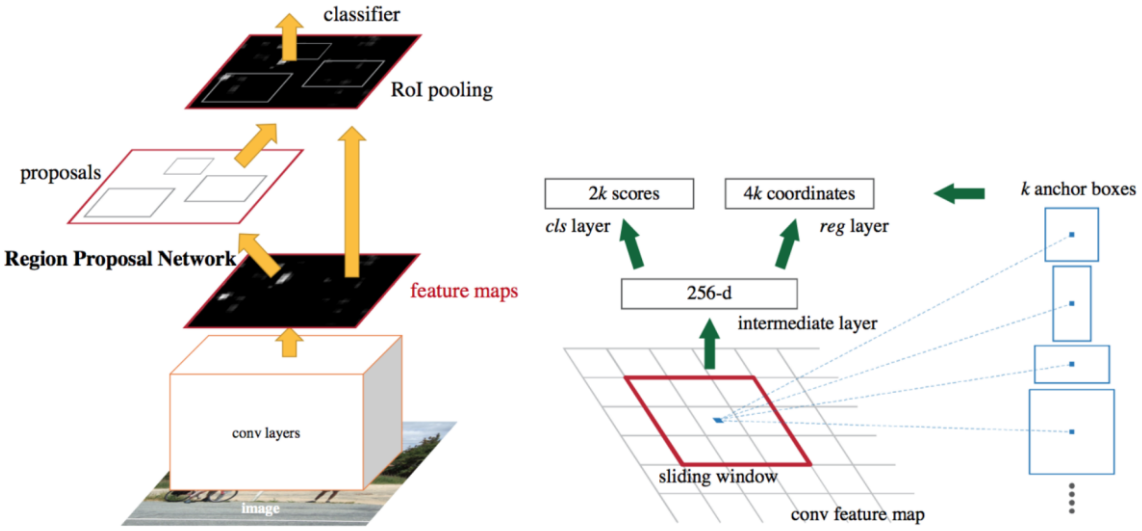

Figure 2.4: Architecture of faster RCNN

Mask R-CNN is s an improvement over Faster R-CNN by including a mask predicting branch parallel to the class label and bounding box prediction branch. The Mask R-CNN has two major parts. First one is the is the region proposal network which generates region proposals in an image and the second part is object detection and mask prediction network. For each object it outputs an array containing the predicted class score. left, top, right and bottom locations of the bounding box of the detected object in the frame

2.2.2 YOLO object detection

The algorithm applies a neural network to an entire image. The network divides the image into an S X S grid and comes up with bounding boxes, which are boxes drawn around images and predicted probabilities for each of these regions.

The method used to come up with these probabilities is logistic regression. The bounding boxes are weighted by the associated probabilities. For class prediction, independent logistic classifiers are used.

YOLO v1: It uses Darknet framework which is trained on ImageNet-1000 dataset. This works as mentioned above but has many limitations because of it the use of the YOL v1 is restricted. It could not find small objects if they are appeared as a cluster. This architecture found difficulty in generalization of objects if the image is of other dimensions different from the trained image. The major issue is localization of objects in the input image.

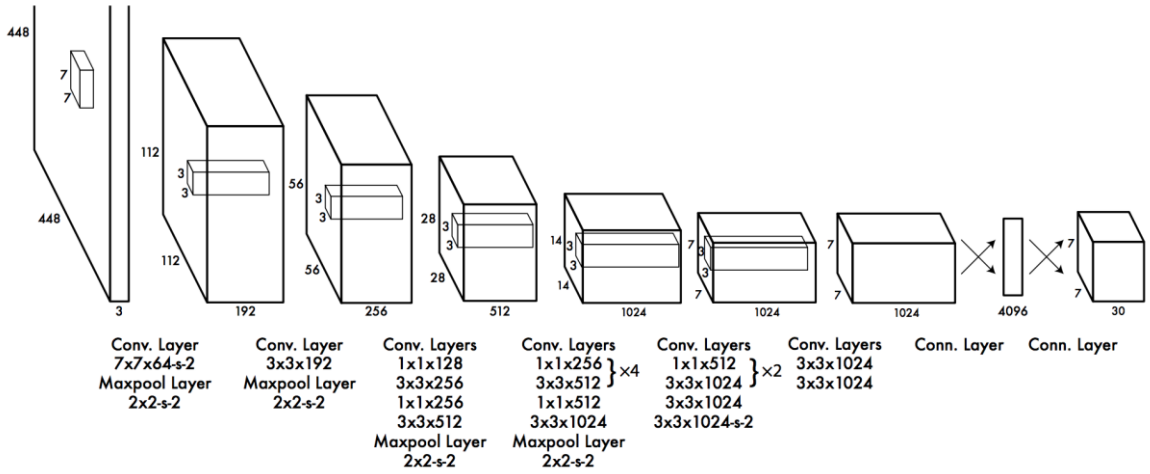

Figure 2.2: YOLO v1 architecture

YOLO v2: The second version of the YOLO is named as YOLO9000 which has been published by Joseph Redmon and Ali Farhadi at the end of 2016. The major improvements of this version are better, faster and more advanced to meet the Faster R-CNN and SSD (Single Shot Multi-Box Detector). it normalizes the input layer by altering slightly and scaling the activations. the input size in YOLO v2 has been increased from 224*224 to 448*448. one of the notable changes in YOLO v2 is the introduction of the anchor boxes. YOLO v2 does classification and prediction in a single framework. These anchor boxes are responsible for predicting bounding boxes and this anchor boxes. It uses Darknet 19 architecture with 19 convolutional layers and 5 max pooling layers and a softmax layer for classification.

YOLO v3: The previous version has been improved for an incremental improvement. In this method it gives the score for the objects in each bounding box. In YOLO v3 it uses logistic classifiers for every class instead of softmax which has been used in the previous YOLO v2. The predecessor YOLO v2 used Darknet-19 as feature extractor and YOLO v3 uses the Darknet-53 network for feature extractor which has 53 convolutional layers. It is much deeper than the YOL v2 and also had shortcut connections.

Using convolutional neural networks for object detection has become more popular because as it results in high accuracy. But one of the major problem with these networks is that they aren't robust to out of distribution samples and it ends up in false confident predictions. The simple approach that can be used to address this problem is to create a negative class with out of distribution samples and train the network. But this can only be implemented in a domain where we can identify the misleading examples. The baseline for detecting out of distribution samples [1] was based on the softmax difference and it was the breakthrough approach and after that with techniques like input pre-processing were introduced in order to make the out of distribution sample more separable. ODIN [2]. In object detection many algorithms are presented and the most successful method was YOLOv3 where the real time detection accuracy was high. Mask R-CNN which is a region based convolutional neural network that works with selective search also gives high accuracies, but it is comparatively slower than YOLOv3.

CHAPTER 3

LITERATURE REVIEW

3.1 Detecting out of distribution samples.

Object Detection and image classification is one of key areas in the field of computer vision. Even though image classification using CNN assure higher accuracies, still there are many challenges to overcome. Detecting out of distribution examples, adversarial examples, anomaly detection and novelty detection are similar kind of problems in neural networks. When the classifiers are deployed in real-world tasks, they tend to fail when the input differs from the training dataset. Recent works have shown that neural networks tend to make high confidence predictions even for completely unrecognizable. The network output has a direct correspondence with the solution of the class, namely a probability for each class. But the output vector always sums up to 1.0. Therefor when we input a sample which is not a part of the training distribution it will give probability values for the nearest class. This has led to overconfident false prediction.

Hendrycks and Gimpel (2017) proposed a baseline method to detect out-of-distribution examples without further re-training networks. In this proposed solution they have used the softmax probabilities to determine whether the example lies in the distribution or not. Correctly classified examples tend to have higher softmax scores than the out of distribution. Even though this approach [1] works for most cases, the abnormality module demonstrates that the baseline can be beaten in some cases.

Using Temperature scaling in the softmax function [2] and adding small controlled perturbation to inputs, the softmax score gap between in- and out-of-distribution examples is further enlarged. The combination of these two techniques reduced the False Positive Rate (FRP) from 34.7% to 4.3%. Shiyu Liang, Yixuan Li and R. Srikanth (2018) proposed ODIN (Out-of-Distribution detector for Neural networks) for detecting out-of-distribution examples in neural networks. ODIN does not require re-training the neural network and can be implemented easily on any modern neural architecture. This was considered as the state of art. The input image undergoes a pre-processing where small perturbations are added. Then the pre-processed image is fed to the neural network, calculate the softmax score and compare it with the threshold and detect whether the image in in or out of distribution. Kimin Lee, Kibok Lee, Honglak Lee and Jinwoo Shin (2018) proposed a simple effective

method for detecting abnormal test samples including out-of-distribution and adversarial ones [3]. This method is applicable to any pre-trained softmax neural classifier. By obtaining the class conditional Gaussian distribution with respect to features of the deep modules under Gaussian discriminant analysis, which result in a confidence score based on the Mahalanobis distance. The proposed method [3] outperforms ODIN [2]. In the same work they have made the framework robust to adversarial attacks.

In [4] they have used metric learning to detect novelty and anomaly. Most of recent works on out of distribution detection are based on cross entropy loss. In these cases, the network output has a direct correspondence with the probability for each class. Therefor with metric learning methods they minimize an objective which encourages images with the same label to be close and images with different labels to be a margin apart in an embedding space. Moreover, these networks do not apply softmax layer, therefor they are not forced to label out of distribution samples over known labels. The theory of metric learning was extended to deep neural networks by Chopra et al [5]. They Proposed to pass images through two parallel branches which share the weights. A loss considers both embeddings, and adapts such a way that similar classes are close and dissimilar classes are far in the embedding space.

In [6] They have developed an algorithm by adding two additional terms to the original loss or cross entropy and name it as confidence loss. The first one forces samples form out of distribution less confident by the classifier and the second one is for generating most effective training samples for the first one. This method does not effect the original classification accuracy. They have used generative adversarial network (GAN) [8] for generating samples from out of distribution. Unlike the original GAN their method generates boundary samples in the low density area of in distribution probability. Then a joint training scheme is introduced to minimize the classifiers loss and new GAN loss alternatively.

Training the anomaly detectors against an auxiliary dataset of outliers is another approach to detect out of distribution samples [7]. Here they call it outlier exposure. This enables anomaly detectors to generalize and detect out distribution samples. Outlier exposure uses an entirely different and disjoint dataset from the training dataset to train the network to detect anomalies.

3.2 Multiclass object detection and counting.

A lot of progress has been made in recent years in object detection after the breakthrough in 2012. Using Convolutional neural networks for object detection has become a key factor. Based on CNNs modern object detectors such as Faster R-CNN [9], YOLO [10], SSD [11] are now good enough to be deployed in real world applications. All these algorithms have different level of performance and has its own advantages and disadvantages. Based on the application the algorithm that gives the best accuracy may differ. In [12] they have discussed on how to choose a detection architecture that achieves the right speed, memory and accuracy balance for any application. They have compare Faster R-CNN, R-FCN and SSD systems which is called meta architectures. In [12] they have presented an experimental comparison of the meta architectures and techniques to improve speed without sacrificing much accuracy. One such technique is using fewer number of proposals than in usual case for faster R-CNN. In [13] they have done a survey of object detection in past 20 years. It clearly shows that object detection has developed in a major scale and current state of art can be applied in real world applications. Still there are some improvements to be done that would make the process more optimized. Figure 3.1 shows the road map of object detection and milestone detectors. A team from Facebook AI Research introduced an improved version of Faster R-CNN called Mask R-CNN [14]. This method extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps.

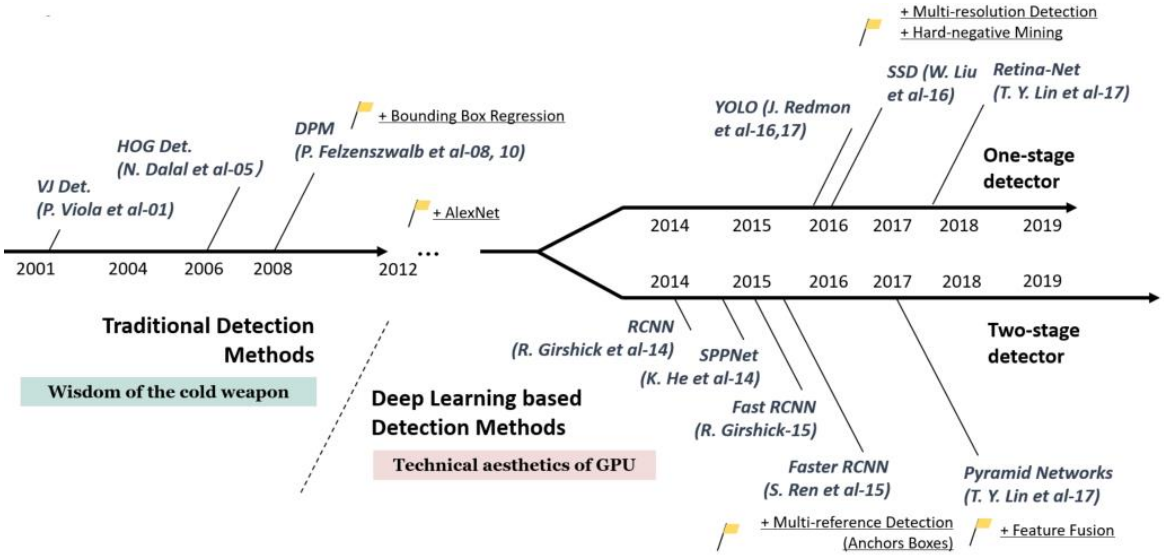

Figure 3.1: A roadmap of object detection and milestone detectors.

CHAPTER 4

DETECTING OUT OF DISTRIBUTION SAMPLES

4.1 Introduction to the methodology of detecting out of distribution samples

In most of the object detection and classification CNN models the final layers are of softmax distribution. One way that we can detect out of distribution samples is that train a negative class while training the models with possible out of distribution samples in the particular domain. For an instance if we are training a food classifier for automating billing in a restaurant, then we can identify possible samples that can be misleading the classifier for false confident predictions, consider objects like spoons, cups, etc. and put them under a non-food class label and train the model so that the model will be robust in the domain to detect anomaly. But in a practical scenario like a self-driving car adding all the negative classes would not be practical. It would be more useful and practical if we can deploy a method for a pre-trained network for detecting out of distribution samples.

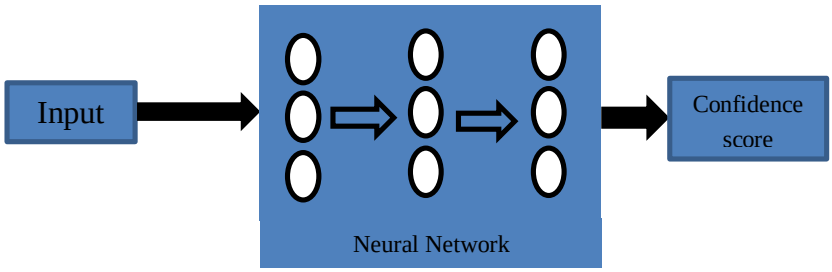

Figure 4.1: Approach to out of distribution problem.

As shown in Figure 4.1 the approach for the problem is to introduce a confidence score for the penultimate layer (layer before last) of the DNNs. Based on this confidence score we decide whether the sample is in distribution or out of distribution. A principal method of detecting anomaly is using multivariate Gaussian distribution and use the Mahalanobis distance between the class mean the sample as the confidence score.

4.2 Softmax Neural Classifier

It is one of the popular activation function used in multiclass problem. Softmax assigns decimal probabilities to each class in a multiclass problem and these probabilities should add up to 1.0.

Lex $x \in X$ be an input and $y \in Y = \{1, \ldots, C\}$ be its label. Suppose that a pre-trained softmax neural classifier is given,

$$P(y = c|X) = \frac{\exp(w_c^T f(x) + b_c)}{\sum_i \exp(w_i^T f(x) + b_i)} \quad \ldots\ldots\ldots\ldots (4.1)$$

We compute the softmax classifier for class c as in the equation 2.4 where w_c and b_c are weights and bias for class c.

4.3 Detecting out of distribution samples using multivariate Gaussian distribution

The multivariate Gaussian distribution is a generalization of the Gaussian distribution to higher dimensions. The parameters of an n-dimension multivariate Gaussian distribution are an n-dimensional mean vector and an n-by-n dimensional covariance matrix.

Parameters: μ ∈ $\mathbb{R}^n$ and Σ∈ $\mathbb{R}^{nxn}$ (covariance matrix)

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \ldots\ldots\ldots\ldots (4.2)$$

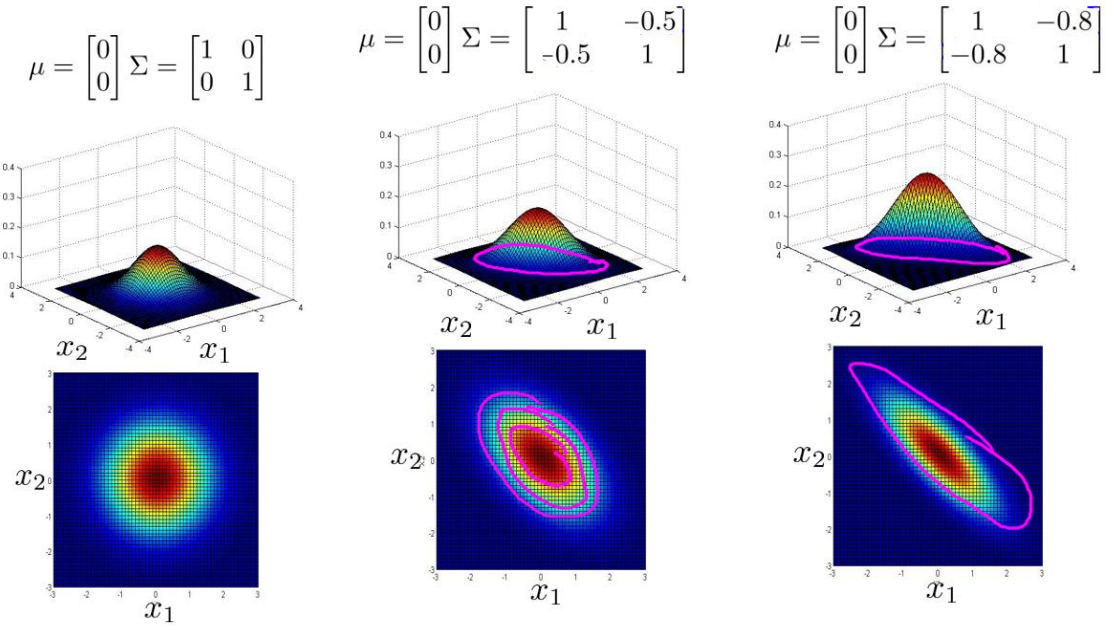

Figure 4.2: A graphical illustration of multivariate Gaussian distribution and how it changes with parameters

In Gaussian multivariate distribution shown in equation 4.2 has two parameters μ and Σ where μ is an n dimensional vector and where Σ is an n-by-n covariance matrix. First we are fitting the parameters μ and Σ.

Let the training set be $\{ x^{(1)}, x^{(2)}, \ldots\ldots, x^{(m)} \}$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \quad \ldots\ldots\ldots\ldots\ldots \quad (4.3)$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T \quad \ldots\ldots\ldots\ldots\ldots \quad (4.4)$$

For the given training set we set the parameters μ and Σ as shown in equations 2.2 and 2.3 respectively. Then for a given new example x we compute $p(x)$ from the equation 2.1. If $p(x) < \epsilon$ we detect it as an out of distribution sample.

4.4 Mahalanobis Distance

The Mahalanobis distance (MD) is the distance between two points in multivariate space. In a regular Euclidean space, variables (e.g. x, y, z) are represented by axes drawn at right angles to each other; The distance between any two points can be measured with a ruler. For uncorrelated variables, the Euclidean distance equals the MD. However, if two or more variables are correlated, the axes are no longer at right angles, and the measurements become impossible with a ruler. In addition, if you have more than three variables, you can't plot them in regular 3D space at all.

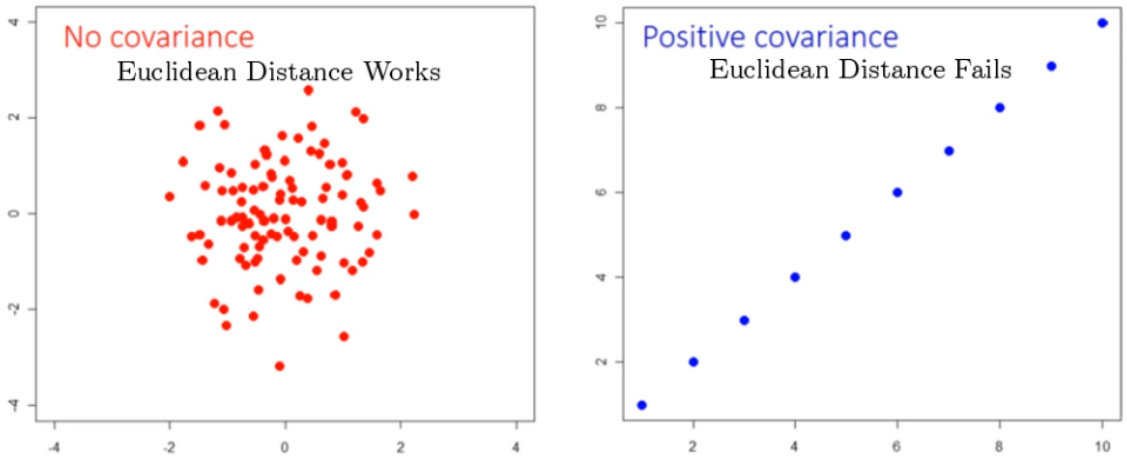

Figure 4.3: Example where Euclidian distance does not work in 2d space.

Figure 4.3 shows an instance where Euclidian distance fails with a positive covariance. The MD solves this measurement problem, as it measures distances between points, even correlated points for multiple variables. The Mahalanobis distance measures distance relative to the centroid which is a base or central point which can be thought of as an overall mean for multivariate data. The centroid is a point in multivariate space where all means from all variables intersect. Therefor Larger the MD, further away from the centroid.

4.5 Detecting out of distribution samples

As most of the convolutional neural networks last layer is of softmax without any modification to this we obtain a generative classifier assuming that the class-conditional distribution follows a multivariate Gaussian distribution. We define a C class conditional Gaussian distribution with a covariance and the mean of the multivariate Gaussian distribution of the class. We compute the class Gaussian distributions from Equation 4.1

$$P(f(x)|y = c) = N(f(x)|\mu_c, \Sigma) \quad \text{..... (4.5)}$$

The approach is based on the theoretical connection between the Gaussian discriminant analysis (GDA) and the softmax classifier. we calculate the empirical class mean and the covariance of the training samples respectively (from pre-trained softmax neural classifier). By fitting those two parameters we find the Mahalanobis distance based confidence score. We define the confidence score $M(x)$ using the Mahalanobis distance between the input sample x and the closest class conditional distribution. We compute $M(x)$ from the equation 4.6.

$$M(x) = \ max_c - (f(x) - \ \mu_c)^T \Sigma^{-1} (f(x) - \ \mu_c)^T \quad \text{..... (4.6)}$$

To make the in and out distribution more separable as in ODIN [2] they introduced temperature scaling we consider adding a small perturbation (small controlled noise) controlled noise for each test sample. To improve the performance, the confidence scores are measured not only on final layer features but also for other low level features in DNN. The figure 4.1 shows a graphical overview of the method

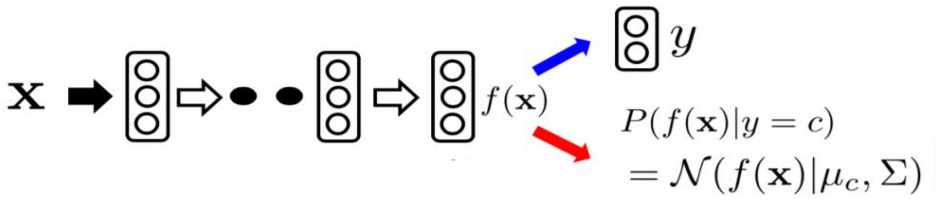

Figure 4.4: Graphical representation of the process of calculating confidence score.

4.6 Calibration Techniques

Input pre-processing: To make in and out of distribution samples more separable we consider adding a small controlled noise to the test sample. For each test sample x we compute $\hat{x}$ with noise added to it and $\hat{x}$ is used to compute the confidence score.

Feature ensemble: Further performance improvements are done by measuring the features not only in final layer features but also low-level features in the DNNs. We extract the hidden features and compute the confidence score for all the layers by using the equation 4.2. We extract the confidence scores from all layers and integrate them by computing the weighted average.

4.7 Algorithm to compute Mahalanobis distance based confidence score

The goal is to compute the Mahalanobis distance based confidence score based on which we decide whether the sample is in distribution or out of distribution. As stated above calibration techniques are used to improve the performance of the system. Adding a small controlled noise will make the in and out distribution samples more separable. Feature ensemble uses not only final layer features but also low level features and consider layer wise Mahalanobis score, and the final confidence score is computed by obtaining the weighted average of these scores. Since these operations have additional input parameters we have to initialize them along with the input image sample in order to process.

The value set for the noise parameter ε will be used to perform the input preprocessing. Weights of the logistic regression $\varphi_l: \forall\, l\, where\, l \in 1, \ldots., L$ and the parameter fitting for the Gaussian multivariate distribution μ_c and Σ should be done at the input stage along with the image sample. Now initialize a Mahalanobis distance based confidence score vector $M(x)$. Next we move forward for processing, for each layer $l \in 1, \ldots., L$ we find the closest class and perform input pre-processing then compute the confidence score for each layer M_l. Finally, to compute Mahalanobis distance based confidence score compute the weighted average of $M_l \varphi_l$. Based on this score we can decide whether the sample is out of distribution or not.

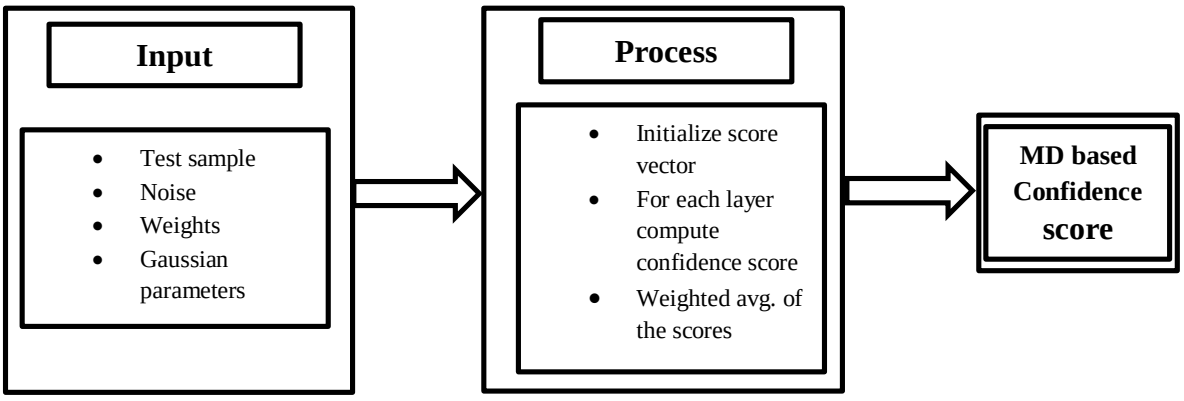

Figure 4.5: Overview of computing Mahalanobis distance based confidence score

4.8 Testing the algorithm for detecting out of distribution samples.

In order to test the algorithm, we use two pre-trained convolutional networks trained on different sets of data. We use DenseNet with 100 layers and ResNet with 34 layers trained on cifar-10, cifar-100 and SVHN datasets. And also we use TinyImageNet and LSUN datasets as out of distribution datasets to validate the method.

4.8.1 Datasets

CIFAR-10: It is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Figure 4.6 shows the classes in the dataset.

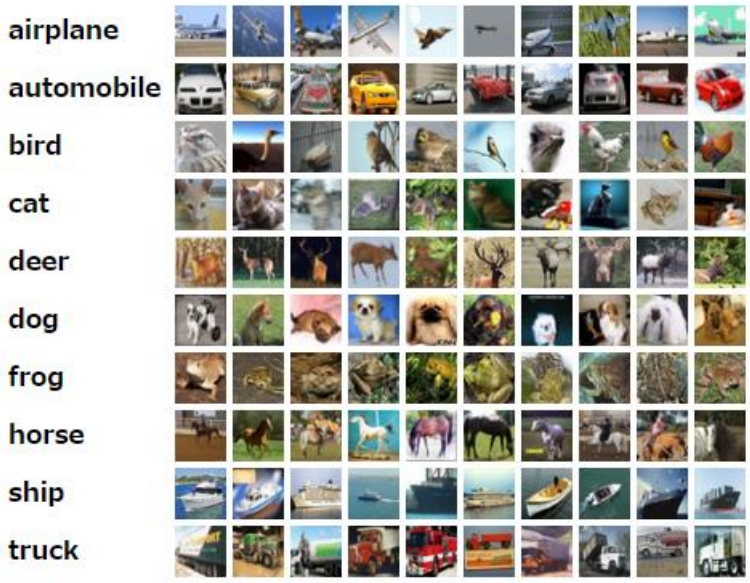

Figure 4.6: CIFAR-10 classes

CIFAR-100: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). Figure 4.7 shows the list of classes in this dataset.

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figure 4.7: CIFAR-100 classes

SVHN: It stands for Street View House Numbers which is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST, but incorporates an order of magnitude more labeled data (over 600,000 digit images). SVHN is obtained from house numbers in Google Street View images. It consists of 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10. There are 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.

Tiny ImageNet: It is a subset of ImageNet dataset. Tiny ImageNet has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. They have released the training and validation sets with images and annotations.

LSUN: The vast dataset, containing images from various rooms, it contains around one million labeled images for each of 10 scene categories and 20 object categories.

4.8.2 Pre-Trained Networks.

We test the models with two popular deep convolutional network architectures ResNet and DenseNet which are used to build models to detect the above mentioned datasets.

ResNet-34: ResNet means the Residual Network and this introduces residual learning. In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. In ResNet-34, "34" refers to the number of layers it has.

DenseNet-100: Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer – DenseNet network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters**.**

Figure 4.7 compares the connectivity in standard, ResNet and DenseNet.

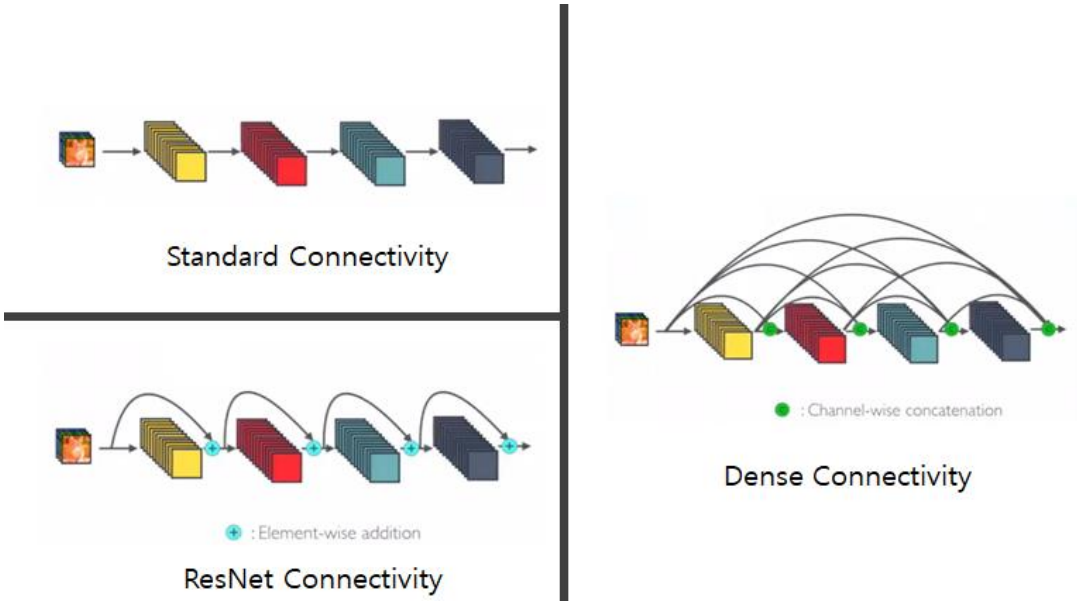

Figure 4.7: Comparison of connectivity in Standard, ResNet and DenseNet

4.9 Testing Environment

Software specification

- Operating System: Ubuntu 18.04

Hardware specifications

- Processor: - Intel Xeon Gold
- RAM: - 128 GB
- Disk Space: - 3 TB
- GPU: - NVIDIA® Tesla® P40 GPU
- GPU RAM: - 24 GB

Programming Technologies

- Python 3.6
- Pytorch 1.2
- CUDA 10.0
- Nunpy
- Scipy
- Scikit-learn

4.10 Application of detecting out of distribution samples in a complex domain

Food detection is one of the complex domain in object classification. Detecting out of distribution samples in real time should be implemented where the model should say non-food to any object detected out of the distribution. This can be implemented by extending the methodology stated above. We train or use a pre rained network which has high accuracies in detecting in distribution food classes. Whenever a test sample is sent to the network do pre-processing and calculate the Mahalanobis distance based confidence score based on the score if its above threshold detect the food class otherwise label it as non-food.

CHAPTER 5

MULTICLASS OBJECT DETECTION AND COUNTING

5.1 Overview of object detection with YOLOv3 and Mask R-CNN

For Multiclass object detection there are many powerful algorithms implemented using convolutional neural networks. The availability of large amount of data and powerful GPUs enables the models to learn faster and efficiently. Here the goal is to classify objects and localizing them using bounding boxes. In this study we are going to test YOLO and Mask R-CNN trained on the same dataset and implement a counting algorithm for a multiclass environment where objects of different classes are present in a single frame.

First we use YOLOv3 which is a state of the art object detector with OpenCV. In traditional computer vision approaches, a sliding window was used to look for objects at different locations and scales. Because this was an expensive operation, the aspect ratio of the object was usually assumed to be fixed. After R-CNN introduced this problem was narrowed down to selective search where the number of searched windows are reduced. This was followed by Faster R-CNN that used a Region Proposal Network (RPN) for identifying bounding boxes that needed to be tested. By clever design the features extracted for recognizing objects, were also used by the RPN for proposing potential bounding boxes thus saving a lot of computation. Mask R-CNN is an improvement over Faster RCNN by including a mask predicting branch parallel to the class label and bounding box prediction branch. YOLO does the detection in a complete different way where it only forward the image only once through the network.

We are using OpenCV for YOLO and Mask-RCNN as it has an easy integration with OpenCV and it supports python. We use MSCOCO dataset which has 91 classes of objects. We use the pre-trained weights to test this model. A common algorithm is used for counting multiclass objects in both the methods. We compare real test cases with different combination of these classes in a single with both the algorithms. Based on their accuracy rates the more suitable algorithm will be further tested with complex combinations for detection and counting.

5.2 Dataset

For training the object detection using YOLOv3 and Mask R-CNN we use the COCO dataset. COCO stands for Common Objects in Context. As hinted by the name, images in COCO dataset are taken from everyday scenes thus attaching “context” to the objects captured in the scenes. COCO was an initiative to collect natural images, the images that reflect everyday scene and provides contextual information. In everyday scene, multiple objects can be found in the same image and each should be labeled as a different object and segmented properly. COCO dataset provides the labeling and segmentation of the objects in the images. There are 91 object categories in COCO.

5.3 Object detection with YOLOv3 and OpenCV

In YOLO we forward the image only once through the network and detect objects in it, which is why it’s called You Only Look Once in short YOLO. First, it divides the image into a 13×13 grid of cells. The size of these 169 cells vary depending on the size of the input. For a 416×416 input size that we used in our experiments, the cell size was 32×32. Each cell is then responsible for predicting a number of boxes in the image. For each bounding box, the network also predicts the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being a particular class. Most of these bounding boxes will be eliminated as it would detect already detected objects. This technique is called non maximum suppression.

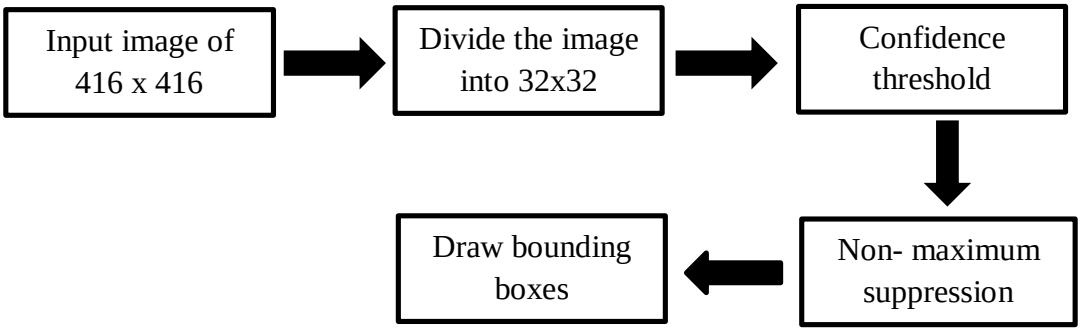

Figure 5.1: Overview of YOLOv3 object detection

Implementation is done with OpenCV libraries. First, we need to download the models and the weights file. Then we have to initialize the parameters. YOLOv3 outputs the bounding boxes for predicted detection and every such bounding box will have its confidence score. Based on this confidence score we have to get rid of less confident detections, in order to accomplish that we initialize a confidence threshold value. As stated above multiple

bounding boxes may detect the same object. In that case we have to find the best fit and eliminate the rest. This technique is called no-maximum suppression. Therefor we have to initiate a non-maximum threshold value to get rid of overlapping bounding boxes. Next we define the default input width and height.

Now the models and classes are loaded. We set the DNN backend to OpenCV and target to CPU or GPU based on the availability. As default the program will run in the GPU but if its unavailable it will automatically switch to the CPU. And now we read the input image. The input image to a neural network needs to be in a certain format called a blob. In this process, it scales the image pixel values to a target range of 0 to 1 using a scale factor of 1/255. It also resizes the image to the given size of (416, 416) without cropping. The forward function in OpenCV's Net class needs the ending layer till which it should run in the network. Since we want to run through the whole network, we need to identify the last layer of the network. Then we run the forward pass of the network to get output from the output layers. The network outputs bounding boxes are each represented by a vector of number of classes + 5 elements. The first 4 elements represent the center of x, center of y, width and height. The fifth element represents the confidence that the bounding box encloses an object. Rest of the elements are the confidence associated with each class. The box is assigned to the class corresponding to the highest score for the box. The highest score for a box is also called its confidence. If the confidence of a box is less than the given threshold, the bounding box is dropped and not considered for further processing. The boxes with their confidence equal to or greater than the confidence threshold are then subjected to Non Maximum Suppression. This would reduce the number of overlapping boxes this done with the non-maximum threshold we initialized. Finally, we draw the boxes that were filtered through the non-maximum suppression, on the input frame with their assigned class label and confidence scores.

5.4 Object detection with Mask R-CNN and OpenCV

Mask R-CNN is an improvement over Faster RCNN by including a mask predicting branch parallel to the class label and bounding box prediction branch as shown in Figure 5.2. It adds only a small overhead to the Faster R-CNN network and hence can still run at 5 fps on a GPU.

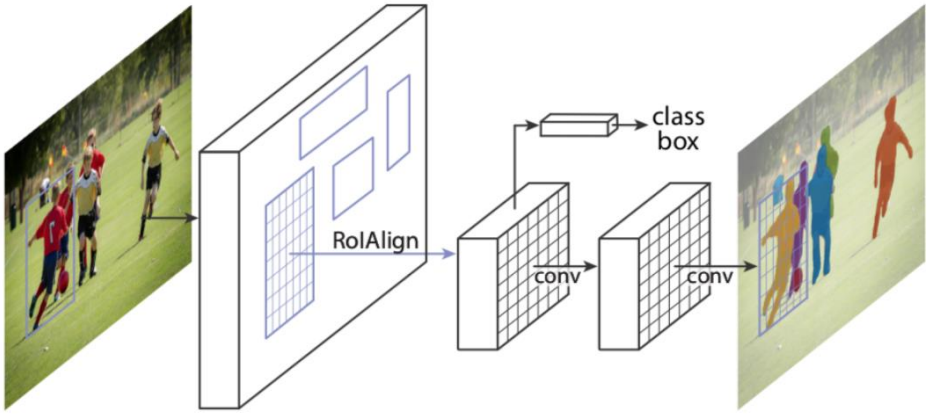

Figure 5.2: Architecture of Mask R-CNN object detection

Mask R-CNN has two major parts where The first one is the Region Proposal Network which generates around 300 region proposals per image. During training, each of these proposals (ROIs) go through the second part which is the object detection and mask prediction network. During inference, the region proposals go through Non-Maximum Suppression and only the top scoring 100 detection boxes are processed by the mask prediction branch. For each object it outputs an array containing the predicted class score, left, top, right and bottom coordinates of the bounding box of the detected object in the frame. These masks can then be threshold to get a completely binary mask

Implementation is done with OpenCV libraries. First, we need to download the models and the weights file. Then we have to initialize the parameters. Each bounding box is associated with a confidence score and the mask threshold. All the boxes below the confidence threshold parameter are ignored for further processing. Similar to the YOLOv3 implementation next models and classes are loaded. We set the DNN backend to OpenCV and target to CPU or GPU based on the availability. As default the program will run in the GPU but if its unavailable it will automatically switch to the CPU. And now we read the input image and transform it to blob format. The blob is then passed in to the network as its input and a forward pass is run to get a list of predicted bounding boxes and the object masks from the output layers. These boxes go through a post-processing step in order to filter out the ones with low confidence scores. In the post process the network's output masks object is a 4-dimensional object, where the first dimension represents the number of detected boxes in the frame, the second dimension represents the number of classes in the model and the third and fourth dimensions represent the mask shape. If the confidence of a box is less than the given threshold, the bounding box is dropped and not considered for further processing.

Finally, we draw the boxes that were filtered through the post-processing step, on the input frame with their assigned class label and confidence scores. We also overlay the colored masks along with their contours inside the boxes.

5.5 Counting Algorithm for multiclass object detection

We apply a common algorithm for counting the multiclass objects and to display and store them. Here we use the object oriented principle to create an object for every class with two attributes name and the count. Both python and c++ commonly used programming languages for OpenCV detection supports this. Along with the object detection counting should be done. As stated above we use either YOLOv3 or Mask R-CNN or any other available object detection algorithm to detect the bounding boxes. Then we perform counting as shown in Figure 5.2.

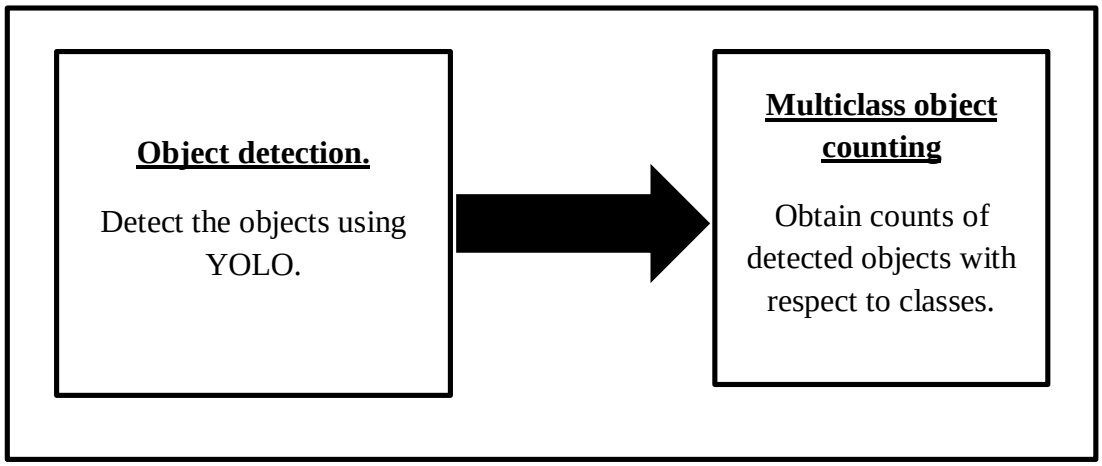

Figure 5.3: Overview of detecting and counting multiclass objects

We begin the counting by creating a class of object in general with two attributes, class-label and the count. We can define more than two attributes based on the need, for an instance if we want to perform a conditional counting we can store the confidence score as a variable. Next we create a list of such objects where the number of elements in the list is equal to the number of classes in the dataset. In the next step we have to assign values for the attributes of the objects in the list so, we load the names of the labels to a text file and read the file in order to create a list of class names. Now we traverse through the objects list and the class name list and assign the values of class name list to objects class-label. At the beginning we assign the count to be zero in all the objects.

Next simultaneously when the object detection algorithm detects a bounding box traverse through the objects list and check for the condition and select the object where the class label

is equals to the detected class. Then increment the count variable by one and add to the label which is displayed in the bounding box. Now each bounding box will have a counting index and that will be updated based on the class detected. Finally, to get the summery of total counts meaning that the counts of each class separately we traverse through the objects array a check for the objects with count equals to a positive value. Print the count of each such classes which will total of detected objects in that particular class. Figure 5.4 shows the steps of the counting algorithm.

Counting Multiclass Objects

1. Create an object class with two attributes, class-name and the count.
2. Create a list of objects where the initial length of the list is equal to number of classes.
3. Load the list of classes in the dataset as text file.
4. Create a list of classes for the classes in the dataset.
5. Traverse through the object list and the classes list.
    a) assign the objects class-name attribute the name in the class list.
    b) Initialize the count parameter with zero for all the objects in the list.
6. Draw the predicted bounding box.
    a) Traverse through the objects list.
    b) If the detected class name is equal to the objects class name increment the count of the detected class by one.
    c) Create a label with the class name, confidence score and the count.
    d) Display the label on top of the bounding box.
7. Get a command line print and display the count summery (number of objects in each class)
        a) Traverse through the objects list
        b) Check for the classes with non-zero count.
        c) Print the count of the class and write it on the image.

Figure 5.4: Algorithm for counting multiclass objects.

5.6 Testing Environment

Software specification

- Operating System: Ubuntu 18.04

Hardware specifications

- Processor: - Intel core i5
- RAM: - 4 GB
- Disk Space: - 1 TB
- GPU: - NVIDIA® GEFORCE 940MX
- GPU RAM: - 2 GB

Programming Technologies

- Python 3.6
- OpenCV
- Nunpy
- Tensor Flow

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Overview of the Results and Discussion.

This chapter highlights the results of the study including snapshots. Results for the experiments carried out for out of distribution problem and multiclass counting problem are presented separately. All results are followed by a discussion which describe how it works.

6.1 Detecting out of distribution samples

To test the methodology in this problem we use a ResNet trained with 100 layers and DenseNet trained with 100 layers on datasets CIFAR-10, CIFAR-100 and SVHN datasets. We use TinyImageNet and LSUN datasets as out of distribution samples for evaluation purpose. We use a threshold based confidence score for which classifies the sample as in-distribution if the confidence score is greater than the threshold. We measure the following matrices.

- TNR: - True Negative Rate.
- TPR: - True Positive Rate.
- AUROC: - Area Under the Receiver Operating Characteristic Curve.
- AUPR: - Area Under Precision Recall Curve.

In the proposed methodology we use input pre-processing which done by adding a small controlled noise to the input sample in order to make in distribution and out of distribution data samples more separable. In feature ensemble we consider the low level features in order to compute the Mahalanobis distance based confidence score rather than just using the high level features. Here we compute the weighted average of the Mahalanobis based distance score of all layers in the network.

First in order to check the impact of the input pre-processing and feature ensemble we use the model ResNet trained on CIFAR-10 and take SVHN as out of distribution dataset. We try this model without both pre-processing and feature ensemble, with one of the technique used and with both techniques used. The results of this experiment is shown in Table 6.1.

Table 6.1: Results of the impact of input pre-processing and feature ensemble.

Feature ensemble	Input pre-processing	TNR at TPR 95%	AUROC	Detection Accuracy	AUPR in	AUPR out
No	No	54.51	93.92	89.13	91.56	95.95
No	Yes	92.26	98.30	93.72	96.01	99.28
Yes	No	91.45	98.37	93.55	96.43	99.35
Yes	**Yes**	**96.42**	**99.14**	**95.75**	**98.26**	**99.60**

From Table 6.1 it shows the when both input pre-processing and feature ensemble is not used the result matrix values are comparatively low. The TNR at TPR 95% increases from 54% to almost 92% when one of the technique is used. When one technique is used the results are almost same. All the maximum values are bold in the table and it shows that the maximum accuracy in all measurement matrices when both input pre-processing and feature ensemble is used. This shows that adding a small noise to pre-process and considering low level features results in higher accuracies. Therefor we will be using both calibration techniques in the following experiments.

Now we will test the model on ResNet-34 trained on CIFAR-10 where CIFAR-10 is the in distribution dataset and SVHN, TinyImageNet and LSUN are used as out of distribution datasets for evaluation.

Table 6.2 Results of ResNet trained CIFAR-10

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
ResNet trained with CIFAR-10	SVHN	96.4	99.1	95.8
	TinyImageNet	97.1	99.5	96.3
	LSUN	98.9	99.7	97.7

From Table 6.2 it shows that the model detects out of distribution samples with above 96% TNR for all three datasets which are used as out of distribution. Based on how much your dataset vary from in distribution the accuracy rates are high.

Similarly, we use ResNet trained on CIFAR-100 and SVHN to identify the impact of using different datasets as in distribution to further validate the method. Table 6.2 and Table 6.3 shows the results of using

Table 6.3 Results of ResNet-34 with CIFAR 100

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
ResNet trained with CIFAR-100	SVHN	91.9	98.4	93.7
	TinyImageNet	90.9	98.2	93.3
	LSUN	90.9	98.2	93.5

Table 6.4 Results of ResNet-34 with SVHN

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
ResNet trained with SVHN	CIFAR-10	98.4	99.3	96.9
	TinyImageNet	99.9	99.9	99.1
	LSUN	99.9	99.9	99.5

Table 6.2 shows the results when ResNet trained CIFAR-100 is used as in distribution and SVHN, TinyImageNet and LSUN used as the out of distribution datasets. Since the number of classes are increased compared to CIFAR-10 the accuracy compared to CIFAR-10 it has gone down to 90%. Table 6.3 shows the results when ResNet trained on SVHN is used as in distribution and CIFAR-10, TinyImageNet and LSUN used as the out of distribution. Here the accuracy is almost 99%. As SVHN is a dataset of street house numbers and other out distribution data are from an entire different domain the accuracy has increased than using CIFAR-10 or CIFAR-100.

Next we will test the model with a different architecture to train the in-distribution model. Here we use DenseNet trained with 100 layers on CIFAR-10, CIFAR-100 and SVHN and LSUN and TinyImageNet as the out of distribution dataset. Table 6.4, Table 6.5 and Table 6.6 shows the results when DenseNet on CIFAR-10, CIFAR-100 and SVHN is used as in-distribution respectively.

Table 6.5 Results of DenseNet Trained with CIFAR-10

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
DenseNet trained with CIFAR-10	SVHN	90.8	98.1	93.9
	TinyImageNet	95.0	98.8	95.0
	LSUN	97.2	99.3	96.3

Table 6.6 Results of DenseNet Trained on CIFAR-100

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
DenseNet trained with CIFAR-100	SVHN	82.5	97.2	91.5
	TinyImageNet	86.6	97.4	92.2
	LSUN	91.4	98.0	93.9

Table 6.7 Results of DenseNet trained on SVHN

	Out of distribution datasets	TNR at TPR 95%	AUROC	Detection Accuracy
DenseNet trained with SVHN	CIFAR-10	96.8	98.9	95.9
	TinyImageNet	99.9	99.9	98.9
	LSUN	98.9	99.9	99.3

From the results in Table 6.4, Table 6.5 and Table 6.6 we can shows that the method gives high accuracies for a different pre trained architectures. Here the same datasets trained with ResNet are used with DenseNet as well. The relationship of datasets which was discussed above with ResNet trained models can be verified here, DenseNet with CIFAR-10 gives more accuracy than CIFAR-100. Therefore, the method is robust on the dataset as well as the trained network architecture. This method can be extended to address adversarial attacks [16] with a similar approach. which is another challenge that all object classification and detection algorithms face.

Further, as we use Mahalanobis distance as the confidence score to detect out of distribution samples with class conditional Gaussian distributions we can extent this method to perform class incremental learning. Class incremental learning means that the model will learn on the run. When out of class distributions with similar Mahalanobis distance we can a do a post process and assign the similar scores to a new class and learn the model with new examples. This can be useful in real world applications. For an instance we can consider a vehicle detection system, once we train and deploy the model there will be new models of vehicles which the system wasn't train. In order to address this issue class incremental learning would be an ideal solution as the model will evolve and increment classes and learn by itself to detect new classes.

Here the pre-trained networks final layer should be softmax in order to use this method. As the concept of Gaussian Discriminant analysis used here for the final layer. Here it works due to the theoretical relation between softmax and Gaussian Discriminant analysis. So if the pre-trained network has any other activation function like sigmoid for the final layer this approach will not work. To handle such a case, we can use a binary classifier before the final layer to detect whether the test samples are in or out of the distribution. Existing works in this problem mostly focus on the softmax distribution. One of the main reason for this is that in most of the object detection algorithms we use softmax to be the final layer as it gives better results.

6.3 Multiclass Object Detection and Counting.

We have implemented multiclass object detection with two popular algorithms YOLOv3 and Mask R-CNN on COCO dataset. YOLO and Mask R-CNN both uses different techniques to detect objects. Based on the domain the best fitting algorithm may be different. First we will compare the detection accuracy of YOLOv3 and Mask R-CNN on same images in simple environment as in general.

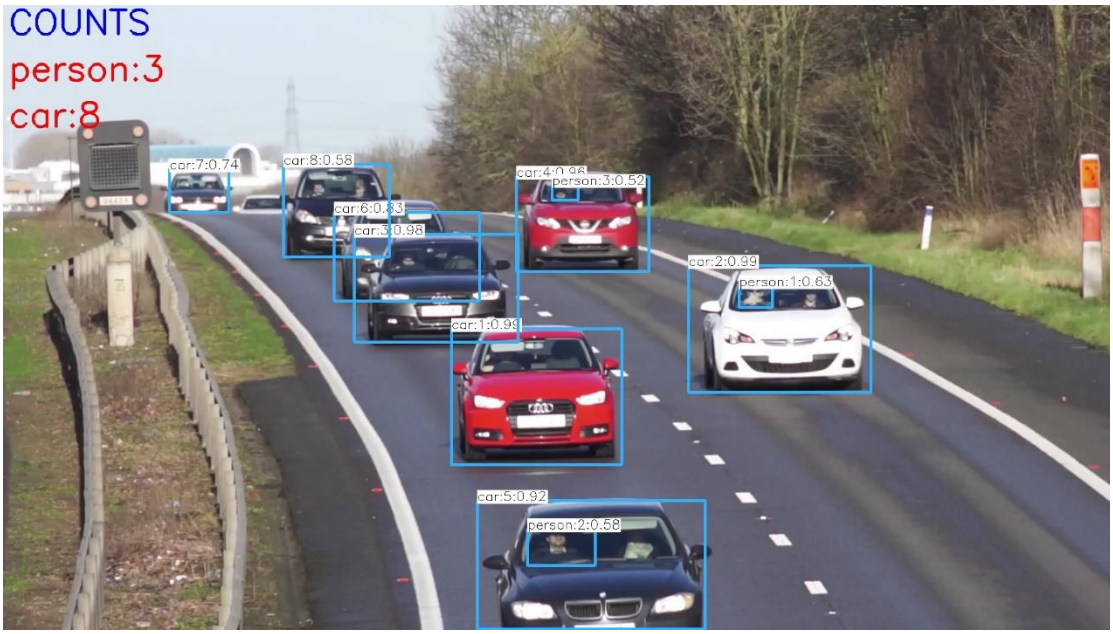

Figure 6.1: Image tested with cars and persons in YOLO

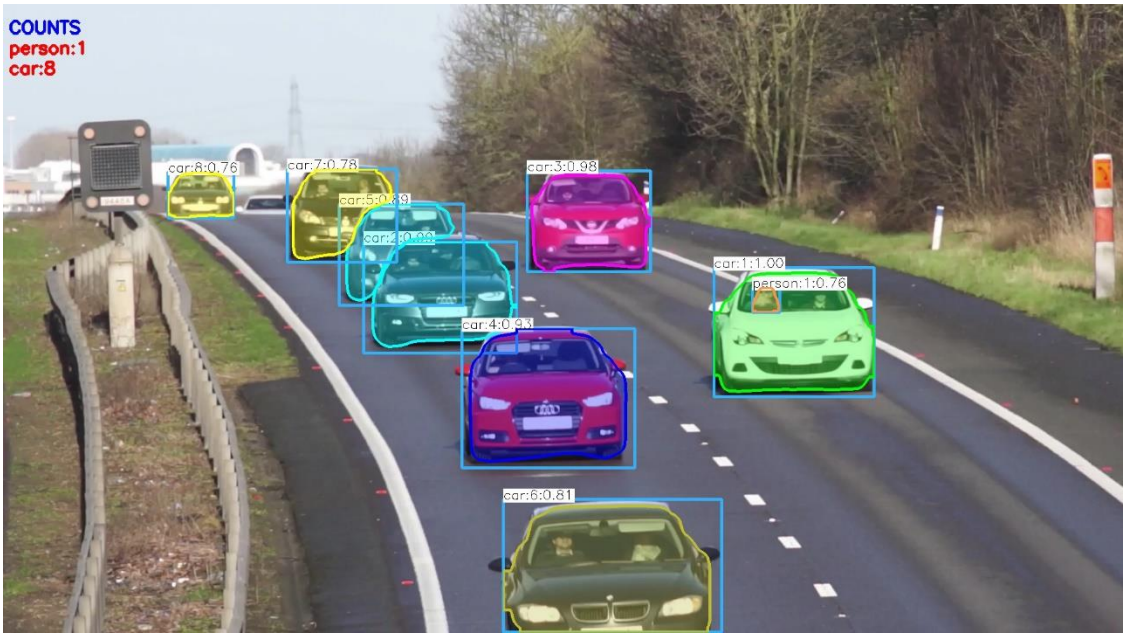

Figure 6.2: Image tested with cars and persons in Mask R-CNN

Figure 6.4 and Figure 6.5 shows the results when an image with 8 visible cars are tested with YOLO and Mask R-CNN. Both algorithms detect the number of cars 100% accurately where number of person inside cars are detected as 3 and 1 in in both algorithms respectively.

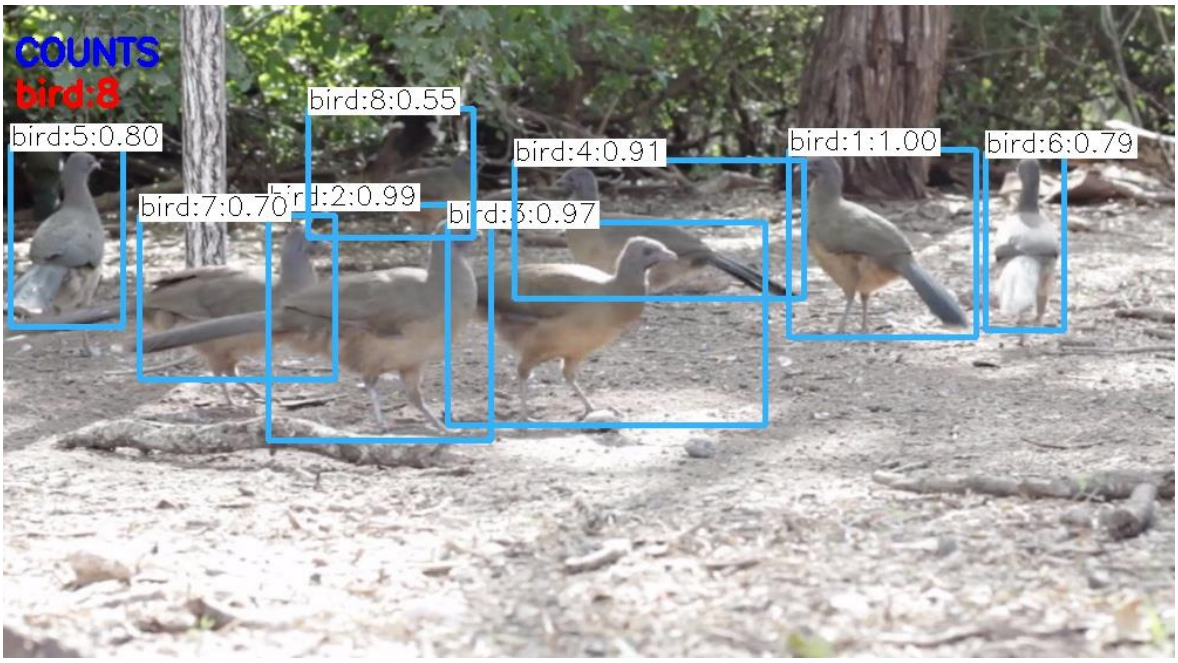

Figure 6.3: Image tested with birds in YOLOv3

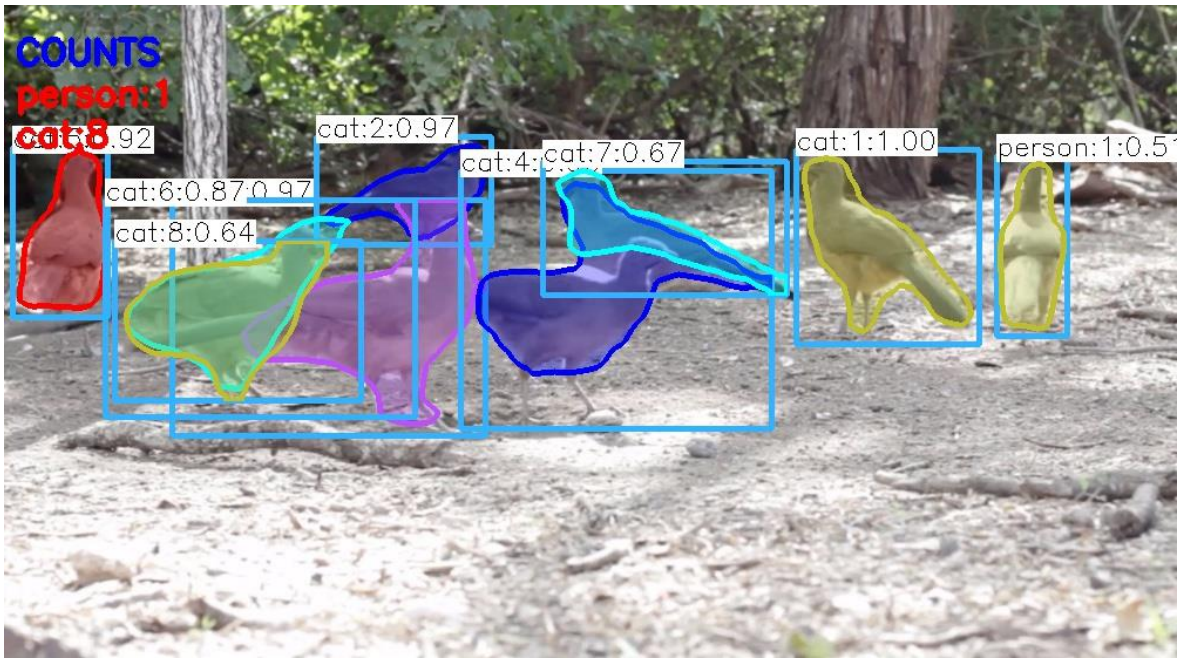

Figure 6.4: Image tested with birds and in Mask R-CNN

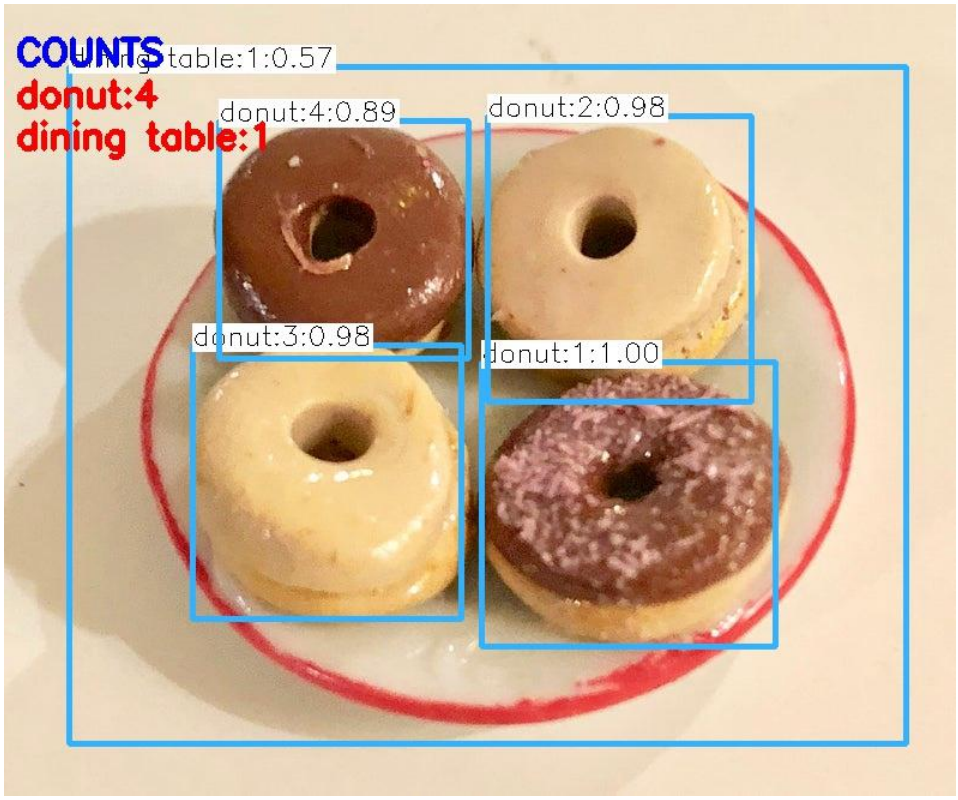

Figure 6.5: Image tested with doughnuts in YOLOv3

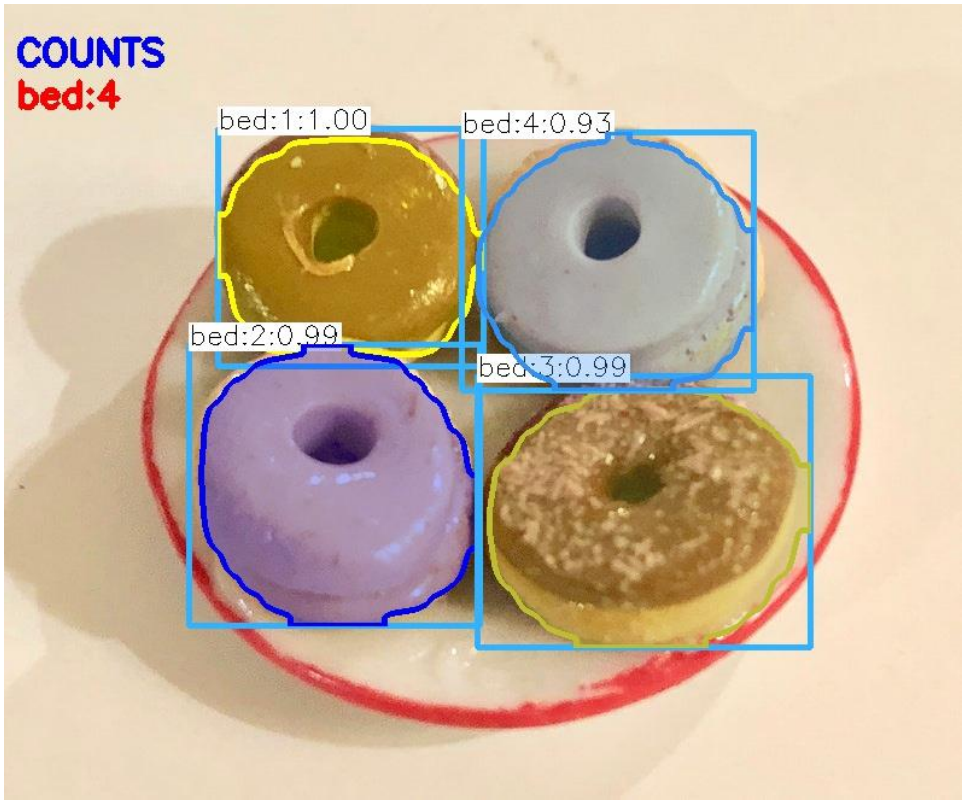

Figure 6.6: Image tested with doughnuts in Mask R-CNN

In Figure 6.6 and Figure 6.7 we testing an image with 8 birds. As shown in Figure 6.6 YOLOv3 detects all birds and gives out the correct counting output with a 100% accuracy but Mask R-CNN gives completely wrong labels. Even though all objects are detected their accuracy is very low. In order to improve the mask R-CNN to detect objects more accurately we have to retrain the network and fine tune hyper parameters.

To analyze this further, we take a simple image with 4 doughnuts and test with both algorithms. As Shown in Figure 6.8 YOLO detects and counts all the objects with 100% accuracy. Mask R-CNN again fails to detect as in Figure 6.9 it doesn't even get detect one doughnut properly.

Now for further analysis we use complex images where multiple objects belonging to different classes are used to test. Here we test these images with YOLOv3. Following figures shows the results of this test case.

Figure 6.7: Image tested with pizzas and cake

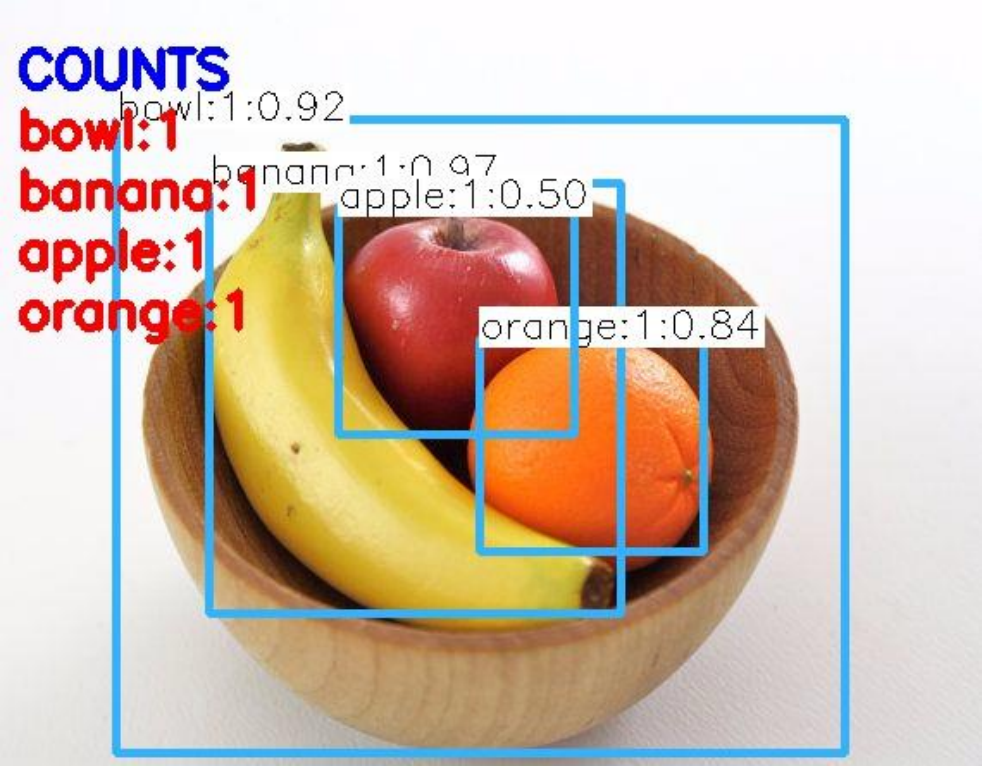

Figure 6.8: Image tested with three fruit classes.

In Figure 6.10 the detection and counts are 100% accurate where YOLOv3 detects 3 pizzas and the cake with the knife with correct counts. In Figure 6.11 We use an image with 3 types of fruits inside a bowl. The model detects all the counts 100% accurately along with the bowl. In the next case we are trying to test our models counting accuracy with the food classes in a more complex images where the food items are kept in different orientations and different camera angles.

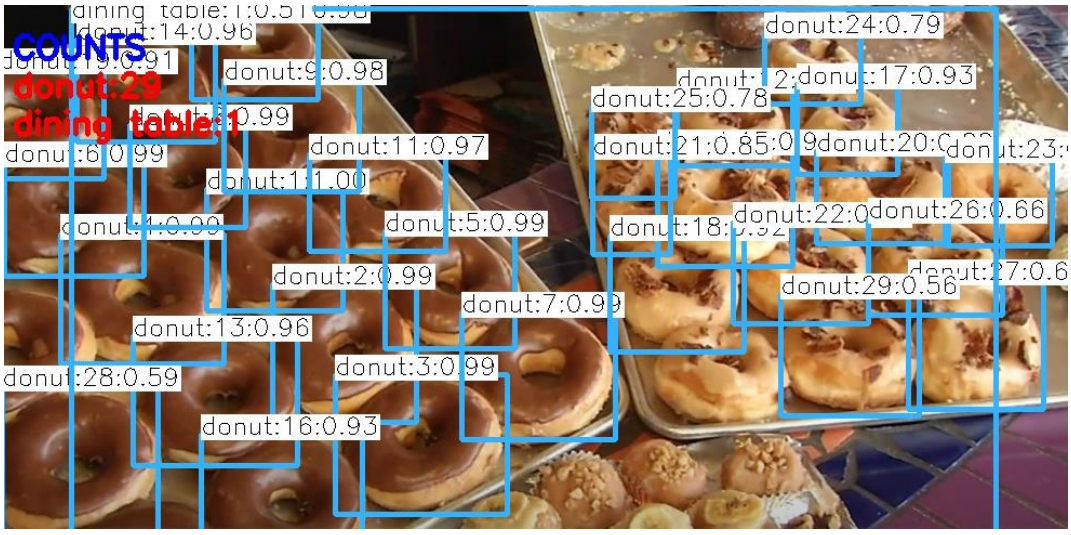

Figure 6.9: Image tested large number of doughnuts

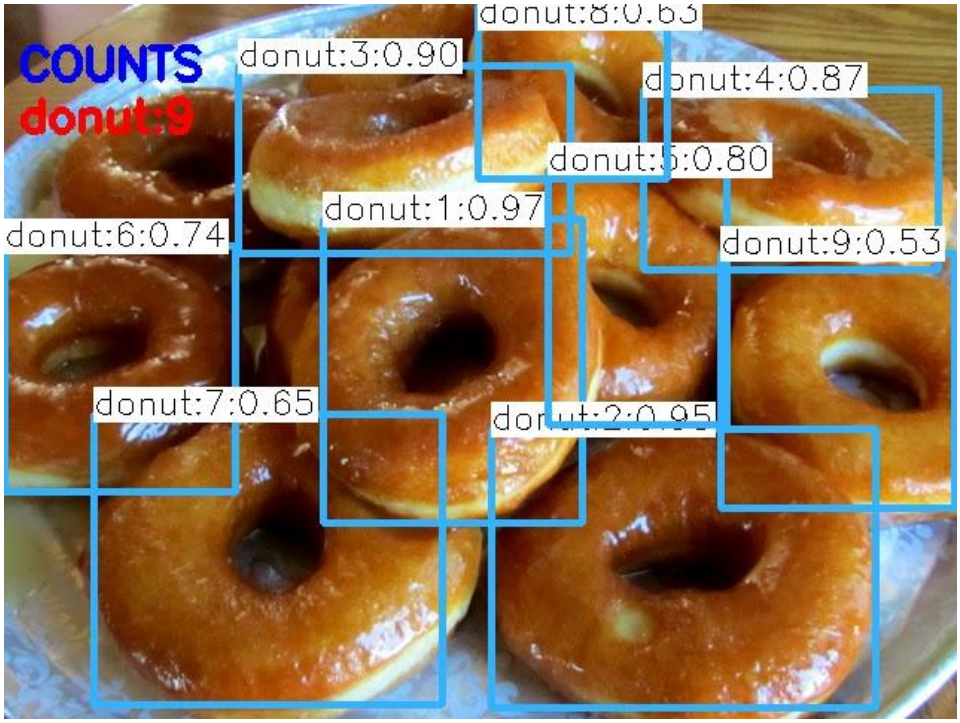

Figure 6.10: Image tested large number of doughnuts in different orientations

Figure 6.11: Image tested with doughnuts in different orientations

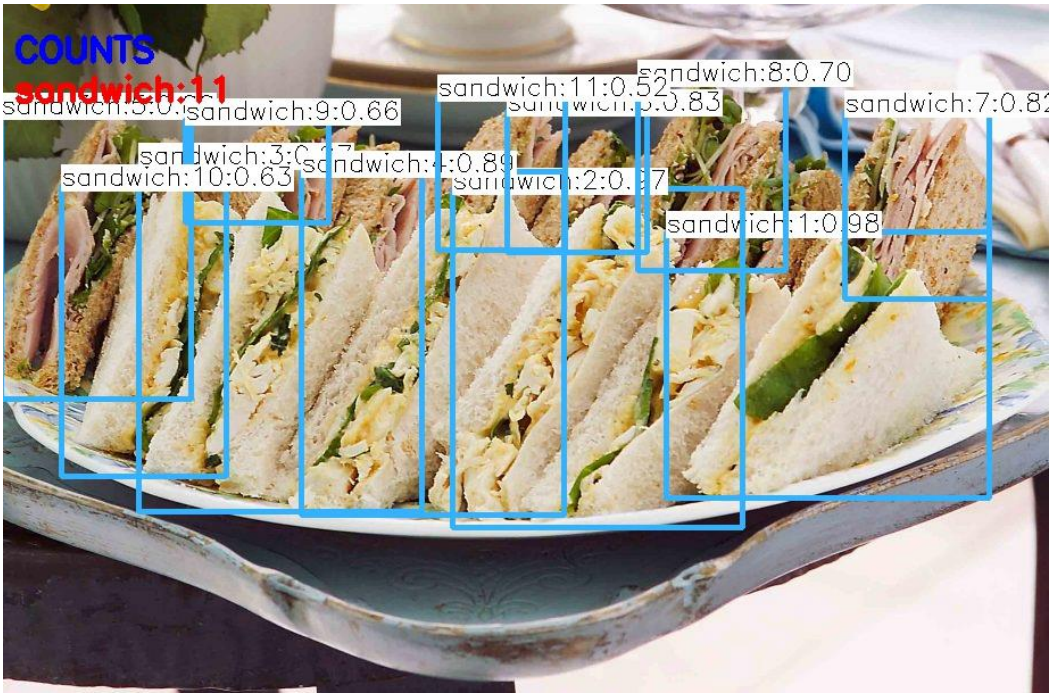

Figure 6.12: Image tested tray of sandwiches.

In Figure 6.12 two large trays of doughnuts in different flavors (colours) are tested the model gives a counting and detecting accuracy of 85% accuracy. And in figure 6.13 a plate full of doughnuts are kept with overlaps. The model does not detect the objects where not even a half of the object is visible. The accuracy of detection and counting in Figure 6.13 is 75%. In image 6.14 an image with only 7 doughnuts but the image is taken from a different camera perspective here the accuracy is 30% which is very low. Figure 6.15 shows a tray full of sandwiches which are overlapping and also too tightly packed. In this case the accuracy is 68%

With above results it shows that the YOLOv3 model gives high accuracy in detecting and counting multiclass objects which are clearly organized with less overlapping. When the domain gets complex with overlapping and camera directions the accuracy goes down depending on the complexity of the image. Figure 6.13 shows example where YOLOv3 fails to detect. The orientation of the cake pieces and the overlaps is the reason for the fall of in the accuracy.

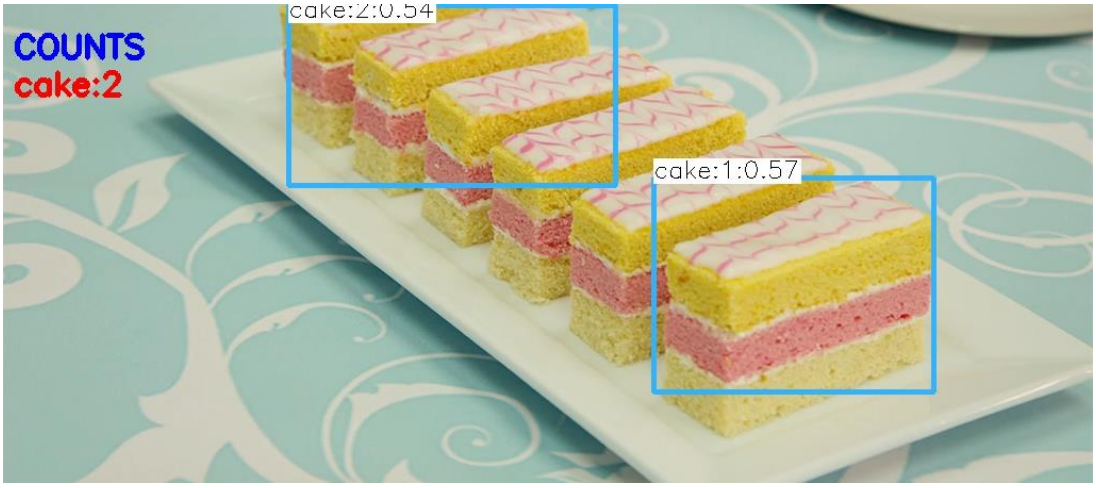

Figure 6.13: Image tested with cake pieces.

We have tried the same models with video inputs with both YOLOv3 and Mask R-CNN and the snapshots of the results are shown in Figure 6.14 and Figure 6.15. Here we measure the accuracy for real time object detection. Both the methods are tested under same environment.

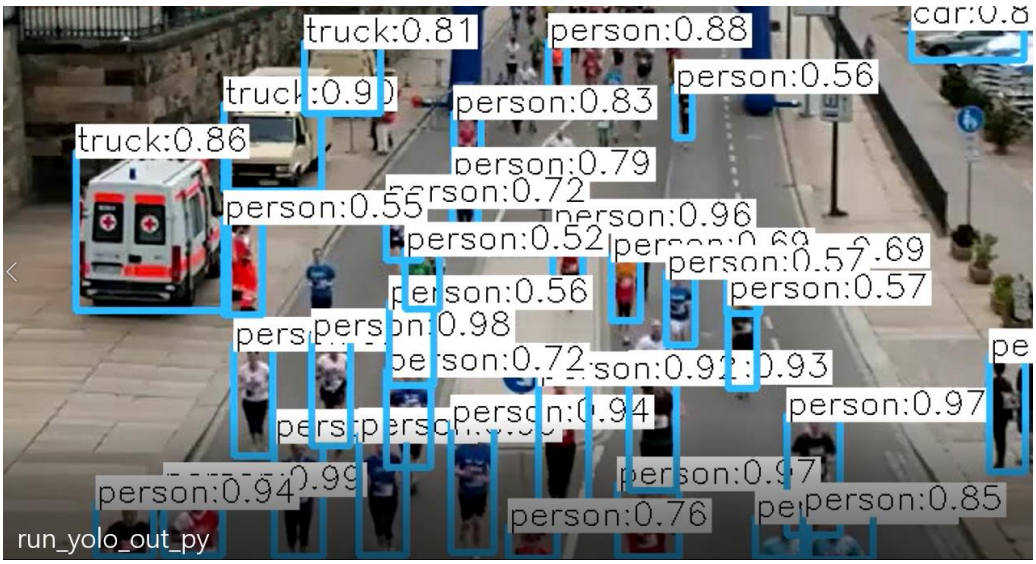

Figure 6.14: Snapshot of running video detection with YOLOv3.

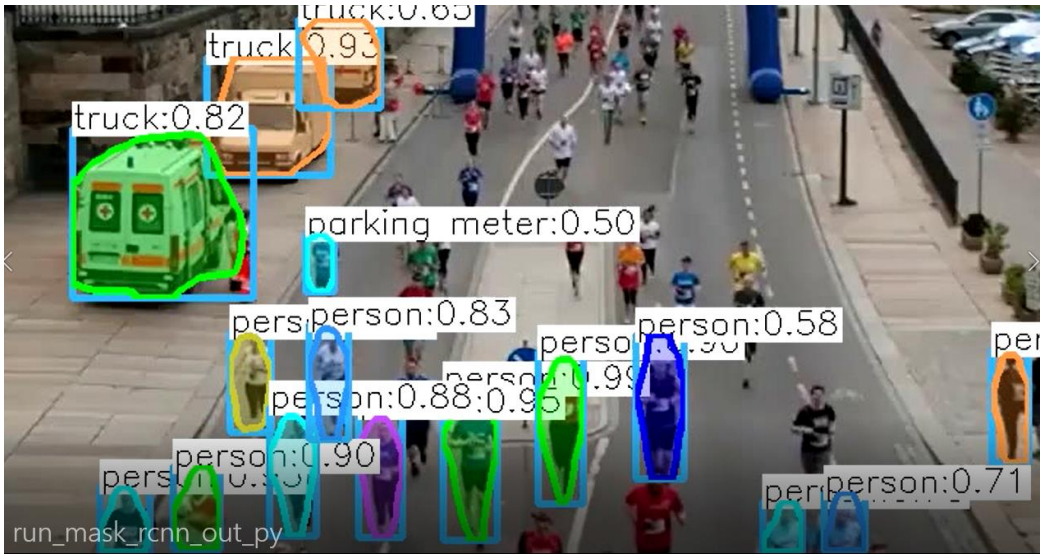

Figure 6.15: Snapshot of running video detection with Mask R-CNN.

YOLOv3 works with high accuracy compared to Mask R-CNN in real time object detection. YOLOv3 execution frames per second is far better than Mask R-CNN. One main reason for YOLO working better with real time object detection is that it looks at the image only once in order to process and detect objects whereas in R-CNN based object detection the process is little complex which reduces the frame rate.

As we can see in Figure 6.14 YOLO detects all the objects in the video frame. But if we want to maintain a count people in the clip we can adjust the proposed algorithms parameters and perform the conditional count. The multiclass counting algorithm is implemented and compared only with YOLOv3 and Mask R-CNN. Here since Mask R-CNN fails to detect simple scenarios where yolo is 99% accurate, therefor only YOLO is tested in complex scenarios. We can extend the same algorithm and implement it with SSD, Faster R-CNN and other object detection algorithms.

CHAPTER 7

CONCLUSION

Detecting out of distribution samples and counting multiclass objects were the main two objectives of this research. In this study we have proposed a method for detecting out of distribution samples. In a convolutional neural network. The main idea is including a confidence score before the final layer for any pre trained softmax classifier based which we decide whether the sample is in distribution or out of distribution. The confidence score is computed as the Mahalanobis distance. We perform two calibration techniques feature ensemble and input pre-processing to improve the accuracy. We also found that the method is robust to any pre trained softmax neural classifier. We can extend the work to detect adversarial samples and also to perform class incremental learning.

In multiclass object detection and counting we used two popular object detection algorithms YOLOv3 and Mask R-CNN to detect the objects. Then we proposed an algorithm for counting multiclass object which can be implemented in both YOLOv3 and Mask R-CNN. The counting algorithm works basically by crating objects to each class in the dataset with a count attribute where the count is incremented whenever an object is detected in a particular class. This can be extended to perform conditional counting where necessary parameters can be assigned to the object class and check for the condition when detection is performed.

CHAPTER 8

FUTURE WORK

We can extend this method to detect adversarial attacks as well. In 2014, a group of researchers at Google and NYU found that it was far too easy to fool ConvNets with an imperceivable, but carefully constructed nudge in the input [17]. Therefor to overcome this issue we can compute the Mahalanobis distance based confidence score with class conditional Gaussian distributions and implement a method similar to out of distribution problem. We can use Adversarial attack methods like FGSM, BIM, DeepFool, CW to validate this method.

Class incremental learning is one important aspect in deep learning models. When the environment changes and when new classes are introduced the models should be robust. But with most of the current models used you have to retrain the networks when new classes are introduced. In class incremental learning we let the model to learn on its own based on the new samples. Since we use Mahalanobis distance based score to detect objects we can use this distance and post process the image and create new labels for detected out of distribution samples with similar features.

CHAPTER 9

REFERANCES

[1] H. Dan and G. Kevin, “A baseline for detecting misclassified and out-of-distribution in neural networks”, in ICRL, 2017.

[2] S. Liang, Y. Li, R. Srikant. “Enhancing the reliability of out-of-distribution image detection in neural networks”, in ICRL, 2018.

[3] K. Lee, K. Lee, H. Lee and J. Shin, “A simple unified framework for detecting out-of-distribuion samples and adversarial attacks”, in NIPS, 2018.

[4] M. Masana, I. Ruiz, J. Serrat, J. van de Weijer, and A. M. Lopez, “Metric Learning for Novelty and Anomaly Detection”, in 2018.

[5] S. Chopra, R. Hadsell, and Y. LeCun. “Learning a similarity metric discriminatively”, with application to face verification. In IEEE conference on Computer Vision and pattern recognition.

[6] K. Lee, H. Lee, K. Lee, and J. Shin, “Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples”, in ICLR, 2018.

[7] M. Kliger and S. Fleishman, “Novelty Detection with GAN”.

[8] D. Hendrycks, M. Mazeika, and T. G. Dietterich, “Deep anomaly detection with outlier exposure”, in ICLR 2019.

[9] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, in 2015

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, real-time object detection. arXivpreprint arXiv:1506.02640, in 2015

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European Conference on Computer Vision, pages 21–37. Springer, 2016

[12] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z.W.Y. Song, S. Guadarrama, K. Murphy in Google Research, “Speed/accuracy tradeoffs for modern convolutional object detectors”

[13] Kaiming He, G. Gkioxari, P. Dollár, R. Girshick, “Mask R-CNN”, in 2017.

[14] A. Krizhevsky, I. Sutskever and G. Hinton, “ImageNet classification with deep convolutional neural networks”, in 2012

[15] J. Redmon, A. Farhadi, YOLO9000: Better, Faster, Stronger, in 2016.

[16] J. Redmon, A. Farhadi, YOLOv3: An Incremental Improvement, in2018.

[17] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Networks, in 2014.

APPENDIX

APPEXDIX A: - Snapshot of results with ResNet on CIFAR-10 as in distribution

```
Out-of-distribution:  lsun_resize
in_distribution: cifar10==========
out_distribution: svhn
 TNR    AUROC  DTACC  AUIN   AUOUT
 96.41  99.14  95.75  98.26  99.60
Input noise: Mahalanobis_0.01

out_distribution: imagenet_resize
 TNR    AUROC  DTACC  AUIN   AUOUT
 97.32  99.44  96.35  99.42  99.39
Input noise: Mahalanobis_0.002

out_distribution: lsun_resize
 TNR    AUROC  DTACC  AUIN   AUOUT
 99.14  99.67  98.05  99.60  99.57
Input noise: Mahalanobis_0.0
```

APPEXDIX B: - Snapshot of results with ResNet on CIFAR-100 as in distribution

```
in_distribution: cifar100==========
out_distribution: svhn
 TNR    AUROC  DTACC  AUIN   AUOUT
 91.94  98.37  93.66  96.41  99.34
Input noise: Mahalanobis_0.01

out_distribution: imagenet_resize
 TNR    AUROC  DTACC  AUIN   AUOUT
 91.39  98.18  93.57  98.22  97.90
Input noise: Mahalanobis_0.001
out_distribution: imagenet_resize
 TNR    AUROC  DTACC  AUIN   AUOUT
 91.39  98.18  93.57  98.22  97.90
Input noise: Mahalanobis_0.001

out_distribution: lsun_resize
 TNR    AUROC  DTACC  AUIN   AUOUT
 89.66  98.09  93.07  98.83  96.79
Input noise: Mahalanobis_0.002
```

APPEXDIX C: - Snapshot of results with ResNet on SVHN as in distributions