UNIVERSITY OF CALCUTTA

2-YEAR MASTER OF TECHNOLOGY 4th SEMESTER EXAMINATION 2020

COMPUTER SCIENCE AND ENGINEERING

Object Classification using Deep Reinforcement Learning

Submitted by

Suvarthi Sarkar
University of Calcutta
Exam Roll Number: 97/CSM/19/1014
Exam Roll Number: 02
Registration Number: A01-1112-0898-14

Under the supervision of

PROF. RAJAT KUMAR PAL
Department of Computer Science and Engineering

UNIVERSITY OF CALCUTTA

CERTIFICATE

This is to certify that the thesis entitled **"Image Classification using Reinforcement Learning"** being submitted by **Suvarthi Sarkar** (Roll No. – 97/CSM/19/1014, Registration No. – A01-1112-0898-14) for the award of the degree of **Master of Technology** in **Computer Science and Engineering** from **University of Calcutta**, is a record of research work carried out by him under my supervision and guidance. **Suvarthi Sarkar** has worked for one year on the above problem at the Department of Computer Science and Engineering, University of Calcutta, and this has reached the standard fulfilling the requirements and the regulations relating to the degree.

[illegible] 7/7/21

....

Signature of Supervisor

Professor
Department of Computer Science and Engineering
University of Calcutta
JD-2, Sector-III, Saltlake, Kolkata - 700106

Prof. Rajat Kumar Pal

Department of Computer Science and Engineering

University of Calcutta

....

Head of the Department

Department of Computer Science and Engineering

University of Calcutta

....

External Professor/ Examiner

ACKNOWLEDGEMENT

I express my sincere thanks to my project guide, **Prof. Dr. Rajat Kumar Pal** of the Department of Computer Science and Engineering, University of Calcutta, **Ranjan Mehra** of Subex Inc., 12303 Airport Way, Suite 390, Broomfield, CO 80021, United States and **Sunita Roy**, PhD Scholar, Department of Computer Science and Engineering, University of Calcutta for giving me their valuable time from their busy schedule, the vital advice, resourceful guidance, inspiring instructions, active supervision. I would like to say a more public thank you to all of them. I would like to pay my sincere thanks other professors of our department whom I approached for help. With my best knowledge and capacity, I hereby, declare that all statements and/or information are true and complete.

Suvarthi Sarkar

____

SUVARTHI SARKAR
M.Tech, 4th Semester
Department of Computer Science and Engineering
University of Calcutta
Roll no.: 97/CSM/191014
Registration No.: A01-1112-0898-14

ABSTRACT

Reinforcement Learning is a new trend in modern machine learning techniques. The traditional supervised approach suffers from a big problem. The problem is that the model's prediction is dependent on the data distribution and frequency of different classes of the training data. This should not be the case ideally. Because we usually use known data to train the model to predict unknown data. This big loop hole in supervised learning is the reason the modern researches is shifting towards reinforcement learning and supervised learning. We have used reinforcement learning to train our model for image classification problem.

Contents

1. Introduction.. 1
2. Problem Definition.. 1
   - 2.1 R-CNN Model Family.. 3
     - 2.1.1 R-CNN.. 3
     - 2.1.2 Fast R-CNN... 5
     - 2.1.3 Faster R-CNN.. 7
   - 2.2 YOLO Model Family... 9
     - 2.2.1 YOLO... 10
     - 2.2.2 YOLOv2 (YOLO9000) and YOLOv3.. 11
3. Reinforcement Learning.. 12
   - 3.1 Q-Learning... 12
   - 3.2 Deep Q-Learning..
4. Research Design..
5. Experimental Results..
6. Graphical Results...
7. Conclusion...
8. References...

List of Figures

Sl no.	Description	Page number
1	Summary of the R-CNN Model ArchitectureTaken from Rich feature hierarchies for accurate object detection and semantic segmentation.	4
2	RCNN Architecture	5
3	Summary of the Fast R-CNN Model Architecture. Taken from: Fast R-CNN	6
4	Fast R-CNN Architecture	7
5	Summary of the Faster R-CNN Model Architecture.Taken from: Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks	8
6	Summary of Predictions made by YOLO Model Taken from: You Only Look Once: Unified, Real-Time Object Detection	10
7	Example of the Representation Chosen when Predicting Bounding Box Position and ShapeTaken from: YOLO9000: Better, Faster, Stronger	11
8	Deep Q Learning	15
9	Episode V/s % of time exploring graph for training MNIST(32 Layers) graph	26
10	Episode V/s mean rewards graph for training MNIST(32 Layers) graph	26
11	Accuracy V/s Training Frequency Graph for MNIST(32 Layers)	26
12	Accuracy V/s Images per Episode for MNIST(32 Layers)	26
13	Accuracy V/s Learning Rate for MNIST(32 Layers)	26
14	Accuracy V/s Total number of Episodes for MNIST(32 Layers)	26
15	Episode V/s % of time exploring graph for training MNIST(64 Layers) graph	27
16	Episode V/s mean rewards graph for training MNIST(64 Layers) graph	27
17	Accuracy V/s Training Frequency Graph for MNIST(64 Layers)	27
18	Accuracy V/s Images per Episode for MNIST(64 Layers)	27
19	Accuracy V/s Learning Rate for MNIST(64 Layers)	27
20	Accuracy V/s Total number of Episodes for MNIST(64 Layers)	27
21	Episode V/s % of time exploring graph for training MNIST(128 Layers) graph	28
22	Episode V/s mean rewards graph for training MNIST(128 Layers) graph	28
23	Accuracy V/s Training Frequency Graph for MNIST(128 Layers)	28
24	Accuracy V/s Images per Episode for MNIST(128 Layers)	28
25	Accuracy V/s Learning Rate for MNIST(128 Layers)	28
26	Accuracy V/s Total number of Episodes for MNIST(128 Layers)	28
27	Episode V/s % of time exploring graph for training Fashion-MNIST(32 Layers) graph	29
28	Episode V/s mean rewards graph for training Fashion-MNIST(32 Layers) graph	29
29	Accuracy V/s Training Frequency Graph for Fashion-MNIST(32 Layers)	29
30	Accuracy V/s Images per Episode for Fashion-MNIST(32 Layers)	29
31	Accuracy V/s Learning Rate for Fashion-MNIST(32 Layers)	29
32	Accuracy V/s Total number of Episodes for Fashion-MNIST(32 Layers)	29
33	Episode V/s % of time exploring graph for training Fashion-MNIST(64	30

	Layers) graph	
34	Episode V/s mean rewards graph for training Fashion-MNIST(64 Layers) graph	30
35	Accuracy V/s Training Frequency Graph for Fashion-MNIST(64 Layers)	30
36	Accuracy V/s Images per Episode for Fashion-MNIST(64 Layers)	30
37	Accuracy V/s Learning Rate for Fashion-MNIST(64 Layers)	30
38	Accuracy V/s Total number of Episodes for Fashion-MNIST(64 Layers)	30
39	Episode V/s % of time exploring graph for training Fashion-MNIST(128 Layers) graph	31
40	Episode V/s mean rewards graph for training Fashion-MNIST(128 Layers) graph	31
41	Accuracy V/s Training Frequency Graph for Fashion-MNIST(128 Layers)	31
42	Accuracy V/s Images per Episode for Fashion-MNIST(128 Layers)	31
43	Accuracy V/s Learning Rate for Fashion-MNIST(128 Layers)	31
44	Accuracy V/s Total number of Episodes for Fashion-MNIST(128 Layers)	31
45	Episode V/s % of time exploring graph for training CIFAR 10(32 Layers) graph	32
46	Episode V/s mean rewards graph for training CIFAR 10 (32 Layers) graph	32
47	Accuracy V/s Training Frequency Graph for CIFAR 10 (32 Layers)	32
48	Accuracy V/s Images per Episode for CIFAR 10 (32 Layers)	32
49	Accuracy V/s Learning Rate for CIFAR 10 (32 Layers)	32
50	Accuracy V/s Total number of Episodes for CIFAR 10 (32 Layers)	32
51	Episode V/s % of time exploring graph for training CIFAR 10 (64 Layers) graph	33
52	Episode V/s mean rewards graph for training CIFAR 10 (64 Layers) graph	33
53	Accuracy V/s Training Frequency Graph for CIFAR 10 (64 Layers)	33
54	Accuracy V/s Images per Episode for CIFAR 10 (64 Layers)	33
55	Accuracy V/s Learning Rate for CIFAR 10 (64 Layers)	33
56	Accuracy V/s Total number of Episodes for CIFAR 10 (64 Layers)	33
57	Episode V/s % of time exploring graph for training CIFAR 10 (128 Layers) graph	24
58	Episode V/s mean rewards graph for training CIFAR 10 (128 Layers) graph	34
59	Accuracy V/s Training Frequency Graph for CIFAR 10 (128 Layers)	34
60	Accuracy V/s Images per Episode for CIFAR 10 (128 Layers)	34
61	Accuracy V/s Learning Rate for CIFAR 10 (128 Layers)	34
62	Accuracy V/s Total number of Episodes for CIFAR 10 (128 Layers)	34

List of Tables

Sl no.	Description	Page number
1	Episode V/s % of time exploring graph for training MNIST(32 Layers) graph	16
2	Episode V/s mean rewards graph for training MNIST(32 Layers) graph	16
3	Accuracy V/s Training Frequency Graph for MNIST(32 Layers)	16
4	Accuracy V/s Images per Episode for MNIST(32 Layers)	16
5	Accuracy V/s Learning Rate for MNIST(32 Layers)	16
6	Accuracy V/s Total number of Episodes for MNIST(32 Layers)	16
7	Episode V/s % of time exploring graph for training MNIST(64 Layers) graph	17

8	Episode V/s mean rewards graph for training MNIST(64 Layers) graph	17
9	Accuracy V/s Training Frequency Graph for MNIST(64 Layers)	17
10	Accuracy V/s Images per Episode for MNIST(64 Layers)	17
11	Accuracy V/s Learning Rate for MNIST(64 Layers)	17
12	Accuracy V/s Total number of Episodes for MNIST(64 Layers)	17
13	Episode V/s % of time exploring graph for training MNIST(128 Layers) graph	18
14	Episode V/s mean rewards graph for training MNIST(128 Layers) graph	18
15	Accuracy V/s Training Frequency Graph for MNIST(128 Layers)	18
16	Accuracy V/s Images per Episode for MNIST(128 Layers)	18
17	Accuracy V/s Learning Rate for MNIST(128 Layers)	18
18	Accuracy V/s Total number of Episodes for MNIST(128 Layers)	18
19	Episode V/s % of time exploring graph for training Fashion-MNIST(32 Layers) graph	19
20	Episode V/s mean rewards graph for training Fashion-MNIST(32 Layers) graph	19
21	Accuracy V/s Training Frequency Graph for Fashion-MNIST(32 Layers)	19
22	Accuracy V/s Images per Episode for Fashion-MNIST(32 Layers)	19
23	Accuracy V/s Learning Rate for Fashion-MNIST(32 Layers)	19
24	Accuracy V/s Total number of Episodes for Fashion-MNIST(32 Layers)	19
25	Episode V/s % of time exploring graph for training Fashion-MNIST(64 Layers) graph	20
26	Episode V/s mean rewards graph for training Fashion-MNIST(64 Layers) graph	20
27	Accuracy V/s Training Frequency Graph for Fashion-MNIST(64 Layers)	20
28	Accuracy V/s Images per Episode for Fashion-MNIST(64 Layers)	20
29	Accuracy V/s Learning Rate for Fashion-MNIST(64 Layers)	20
30	Accuracy V/s Total number of Episodes for Fashion-MNIST(64 Layers)	20
31	Episode V/s % of time exploring graph for training Fashion-MNIST(128 Layers) graph	21
32	Episode V/s mean rewards graph for training Fashion-MNIST(128 Layers) graph	21
33	Accuracy V/s Training Frequency Graph for Fashion-MNIST(128 Layers)	21
34	Accuracy V/s Images per Episode for Fashion-MNIST(128 Layers)	21
35	Accuracy V/s Learning Rate for Fashion-MNIST(128 Layers)	21
36	Accuracy V/s Total number of Episodes for Fashion-MNIST(128 Layers)	21
37	Episode V/s % of time exploring graph for training CIFAR 10(32 Layers) graph	22
38	Episode V/s mean rewards graph for training CIFAR 10 (32 Layers) graph	22
39	Accuracy V/s Training Frequency Graph for CIFAR 10 (32 Layers)	22
40	Accuracy V/s Images per Episode for CIFAR 10 (32 Layers)	22
41	Accuracy V/s Learning Rate for CIFAR 10 (32 Layers)	22
42	Accuracy V/s Total number of Episodes for CIFAR 10 (32 Layers)	22
43	Episode V/s % of time exploring graph for training CIFAR 10 (64 Layers) graph	23
44	Episode V/s mean rewards graph for training CIFAR 10 (64 Layers) graph	23
45	Accuracy V/s Training Frequency Graph for CIFAR 10 (64 Layers)	23
46	Accuracy V/s Images per Episode for CIFAR 10 (64 Layers)	23
47	Accuracy V/s Learning Rate for CIFAR 10 (64 Layers)	23

48	Accuracy V/s Total number of Episodes for CIFAR 10 (64 Layers)	23
49	Episode V/s % of time exploring graph for training CIFAR 10 (128 Layers) graph	24
50	Episode V/s mean rewards graph for training CIFAR 10 (128 Layers) graph	24
51	Accuracy V/s Training Frequency Graph for CIFAR 10 (128 Layers)	24
52	Accuracy V/s Images per Episode for CIFAR 10 (128 Layers)	24
53	Accuracy V/s Learning Rate for CIFAR 10 (128 Layers)	24
54	Accuracy V/s Total number of Episodes for CIFAR 10 (128 Layers)	24
55	Accuracy V/s Number of steps for MNIST	25
56	Accuracy V/s Number of steps for Fashion-MNIST	25
57	Accuracy V/s Number of steps for CIFAR10	25

INTRODUCTION

In real life, all the data we collect are in large amounts. The data is useless unless it is processed. It is impossible to process them manually. Here's when the concept of feature extraction comes in.

We can use machine learning for object detection on images. To work with them, we have to go for feature extraction procedure which will make your life easy.

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when we want to process it will be easier. The most important characteristic of these large data sets is that it has a large number of variables. These variables require a lot of computing resources to process them. So Feature extraction helps to get the best feature from those big data sets by select and combine variables into features, thus, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with the accuracy and originality.

The possibilities of working with images using computer vision techniques are endless. There's a strong belief that when it comes to working with unstructured data, especially image data, deep learning models are the way forward. Deep learning techniques undoubtedly perform extremely well.

For our particular problem, we have concentrated the task to human face detection. It is one of the particular instance of object. So, we have divided our task into two independent parts – feature recognition and classification. In the first part, our primary task is to identify the location of faces. By feature recognition, we will try to find the important factors (features) of human faces to ensure their position.

2. Problem Definition

Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs.
Image classification involves predicting the class of one object in an image. *Object localization* refers to identifying the location of one or more objects in an image and drawing

abounding box around their extent. *Object detection* combines these two tasks and localizes and classifies one or more objects in an image.

As such, we can distinguish between these three computer vision tasks:

- **Image Classification**: Predict the type or class of an object in an image.
  Input: An image with a single object, such as a photograph.
  Output: A class label (e.g. one or more integers that are mapped to class labels).
- **Object Localization**: Locate the presence of objects in an image and indicate their location with a bounding box.
  Input: An image with one or more objects, such as a photograph.
  Output: One or more bounding boxes (e.g. defined by a point, width, and height).
- **Object Detection**: Locate the presence of objects with a bounding box and types or classes of the located objects in an image.
  Input: An image with one or more objects, such as a photograph.
  Output: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.
- **Image classification**: Algorithms produce a list of object categories present in the image.
- **Single-object localization**: Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of one instance of each object category.
- **Object detection**: Algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of every instance of each object category.

We can see that "*Single-object localization*" is a simpler version of the more broadly defined "*Object Localization*," constraining the localization tasks to objects of one type within an image, which we may assume is an easier task.
The main problem is to classify the object. The input image will be fed to the model and the agent will make a prediction. The agent acts like a black box.

The performance of a model for image classification is evaluated using the mean classification error across the predicted class labels. The performance of a model for single-object localization is evaluated using the distance between the expected and predicted bounding box for the expected class. Whereas the performance of a model for object

recognition is evaluated using the precision and recall across each of the best matching bounding boxes for the known objects in the image.
Now that we are familiar with the problem of object localization and detection, let’s take a look at some recent top-performing deep learning models.

2.1 R-CNN Model Family

The R-CNN family of methods refers to the R-CNN, which may stand for “*Regions with CNN Features*” or “*Region-Based Convolutional Neural Network*,” developed by Ross Girshick, et al.
This includes the techniques R-CNN, Fast R-CNN, and Faster-RCNN designed and demonstrated for object localization and object recognition.
Let’s take a closer look at the highlights of each of these techniques in turn.

2.1.1 R-CNN

The R-CNN was described in the 2014 paper by Ross Girshick, et al. from UC Berkeley titled “Rich feature hierarchies for accurate object detection and semantic segmentation.”
It may have been one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation. The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset.
Their proposed R-CNN model is comprised of three modules; they are:

- **Module 1: Region Proposal**. Generate and extract category independent region proposals, e.g. candidate bounding boxes.
- **Module 2: Feature Extractor**. Extract feature from each candidate region, e.g. using a deep convolutional neural network.
- **Module 3: Classifier**. Classify features as one of the known class, e.g. linear SVM classifier model.

The architecture of the model is summarized in the image below, taken from the paper.

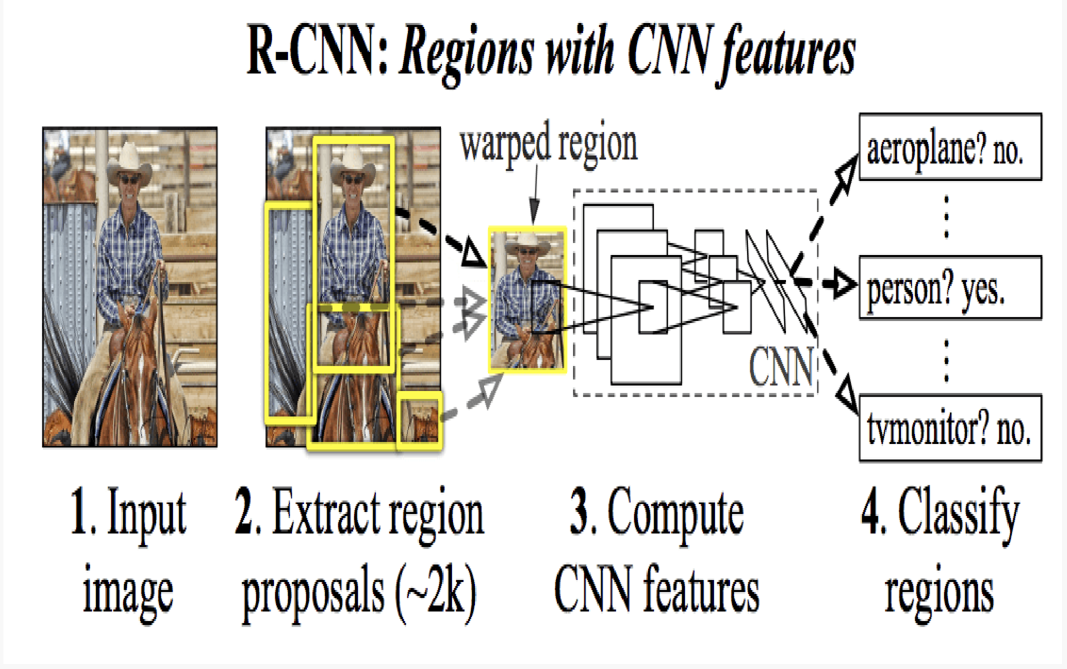


Fig 1: Summary of the R-CNN Model ArchitectureTaken from Rich feature hierarchies for accurate object detection and semantic segmentation.

A computer vision technique is used to propose candidate regions or bounding boxes of potential objects in the image called "*selective search*," although the flexibility of the design allows other region proposal algorithms to be used.

The feature extractor used by the model was the AlexNet deep CNN that won the ILSVRC-2012 image classification competition. The output of the CNN was a 4,096 element vector that describes the contents of the image that is fed to a linear SVM for classification, specifically one SVM is trained for each known class.

It is a relatively simple and straightforward application of CNNs to the problem of object localization and recognition. A downside of the approach is that it is slow, requiring a CNN-based feature extraction pass on each of the candidate regions generated by the region proposal algorithm. This is a problem as the paper describes the model operating upon approximately 2,000 proposed regions per image at test-time.

Python (Caffe) and MatLab source code for R-CNN as described in the paper was made available in the R-CNN GitHub repository.

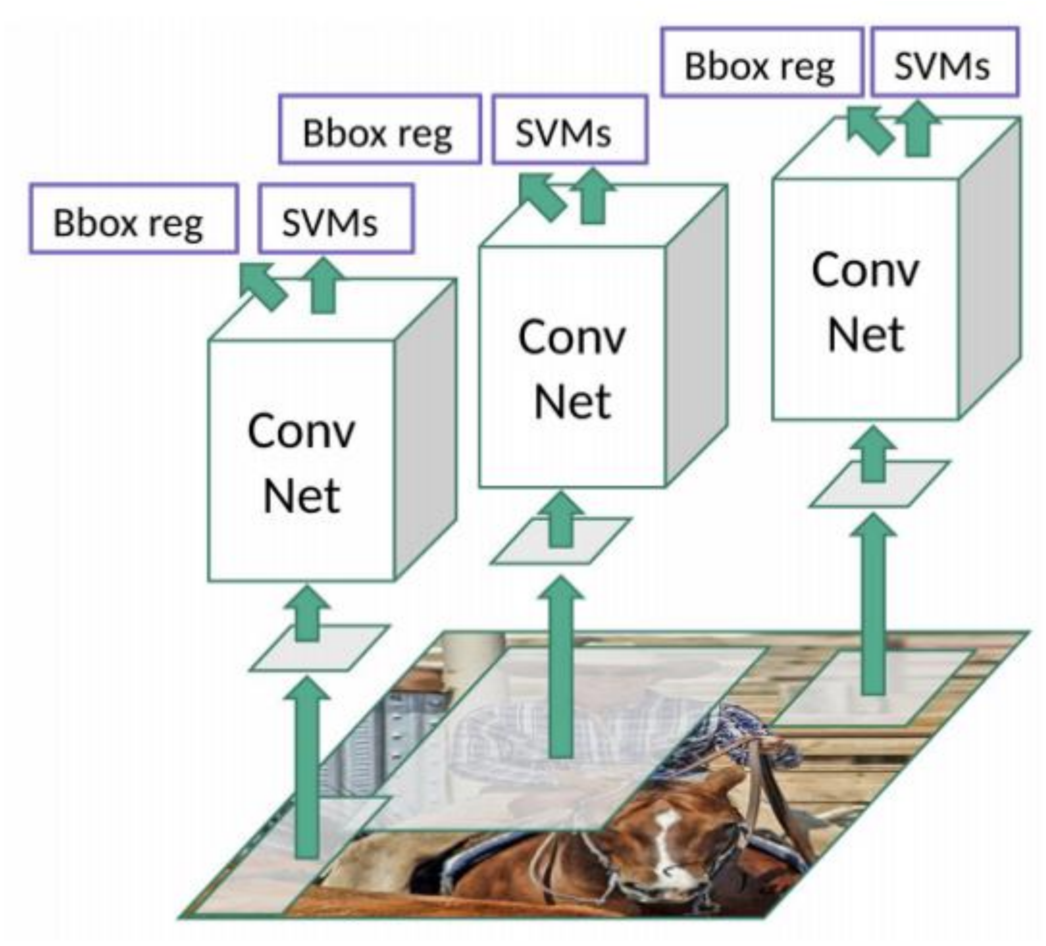


Fig 2: RCNN Architecture

2.1.2 Fast R-CNN

Given the great success of R-CNN, Ross Girshick, then at Microsoft Research, proposed an extension to address the speed issues of R-CNN in a 2015 paper titled "Fast R-CNN."
The paper opens with a review of the limitations of R-CNN, which can be summarized as follows:

- **Training is a multi-stage pipeline**. Involves the preparation and operation of three separate models.
- **Training is expensive in space and time**. Training a deep CNN on so many region proposals per image is very slow.
- **Object detection is slow**. Make predictions using a deep CNN on so many region proposals is very slow.

A prior work was proposed to speed up the technique called spatial pyramid pooling networks, or SPPnets, in the 2014 paper "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." This did speed up the extraction of features, but essentially used a type of forward pass caching algorithm.

Fast R-CNN is proposed as a single model instead of a pipeline to learn and output regions and classifications directly.

The architecture of the model takes the photograph a set of region proposals as input that are passed through a deep convolutional neural network. A pre-trained CNN, such as a VGG-16, is used for feature extraction. The end of the deep CNN is a custom layer called a Region of Interest Pooling Layer, or RoI Pooling, that extracts features specific for a given input candidate region.

The output of the CNN is then interpreted by a fully connected layer then the model bifurcates into two outputs, one for the class prediction via a softmax layer, and another with a linear output for the bounding box. This process is then repeated multiple times for each region of interest in a given image.

The architecture of the model is summarized in the image below, taken from the paper.

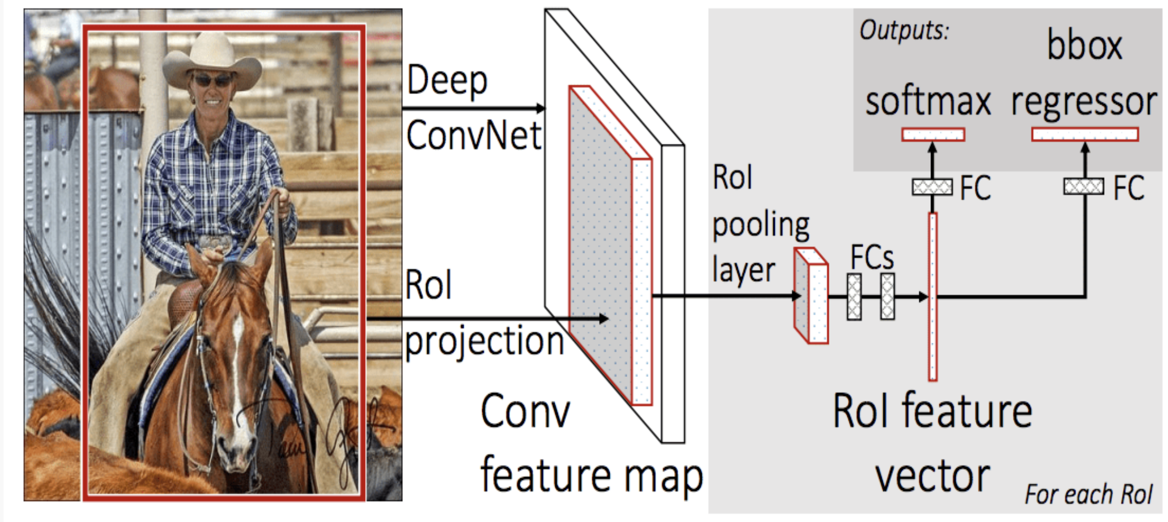


Fig 3: Summary of the Fast R-CNN Model Architecture.

Taken from: Fast R-CNN.

The model is significantly faster to train and to make predictions, yet still requires a set of candidate regions to be proposed along with each input image.

Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

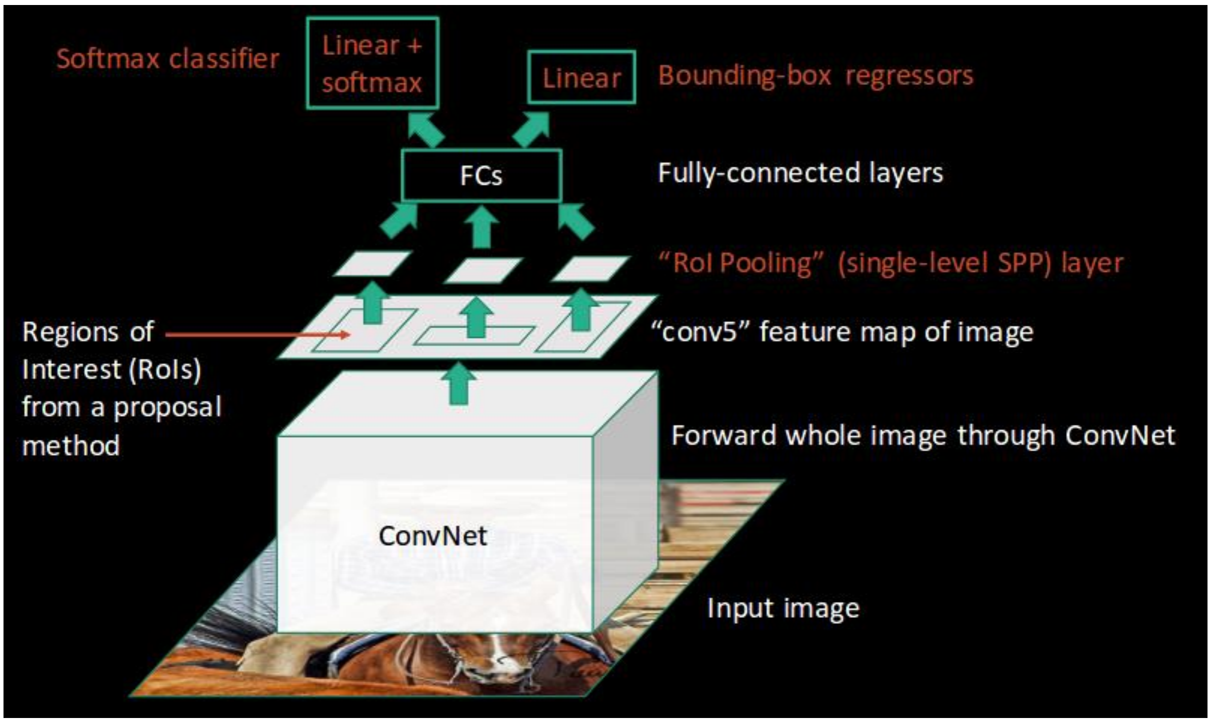

Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" (single-level SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image

Fig 4: Fast R-CNN Architecture

2.1.3. Faster R-CNN

The model architecture was further improved for both speed of training and detection by Shaoqing Ren, et al. at Microsoft Research in the 2016 paper titled "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."

The architecture was the basis for the first-place results achieved on both the ILSVRC-2015 and MS COCO-2015 object recognition and detection competition tasks.

The architecture was designed to both propose and refine region proposals as part of the training process, referred to as a Region Proposal Network, or RPN. These regions are then used in concert with a Fast R-CNN model in a single model design. These improvements both reduce the number of region proposals and accelerate the test-time operation of the model to near real-time with then state-of-the-art performance.

… our detection system has a frame rate of 5fps (including all steps) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks

— Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.

Although it is a single unified model, the architecture is comprised of two modules:

- **Module 1: Region Proposal Network**. Convolutional neural network for proposing regions and the type of object to consider in the region.
- **Module 2: Fast R-CNN**. Convolutional neural network for extracting features from the proposed regions and outputting the bounding box and class labels.

Both modules operate on the same output of a deep CNN. The region proposal network acts as an attention mechanism for the Fast R-CNN network, informing the second network of where to look or pay attention.

The architecture of the model is summarized in the image below, taken from the paper.

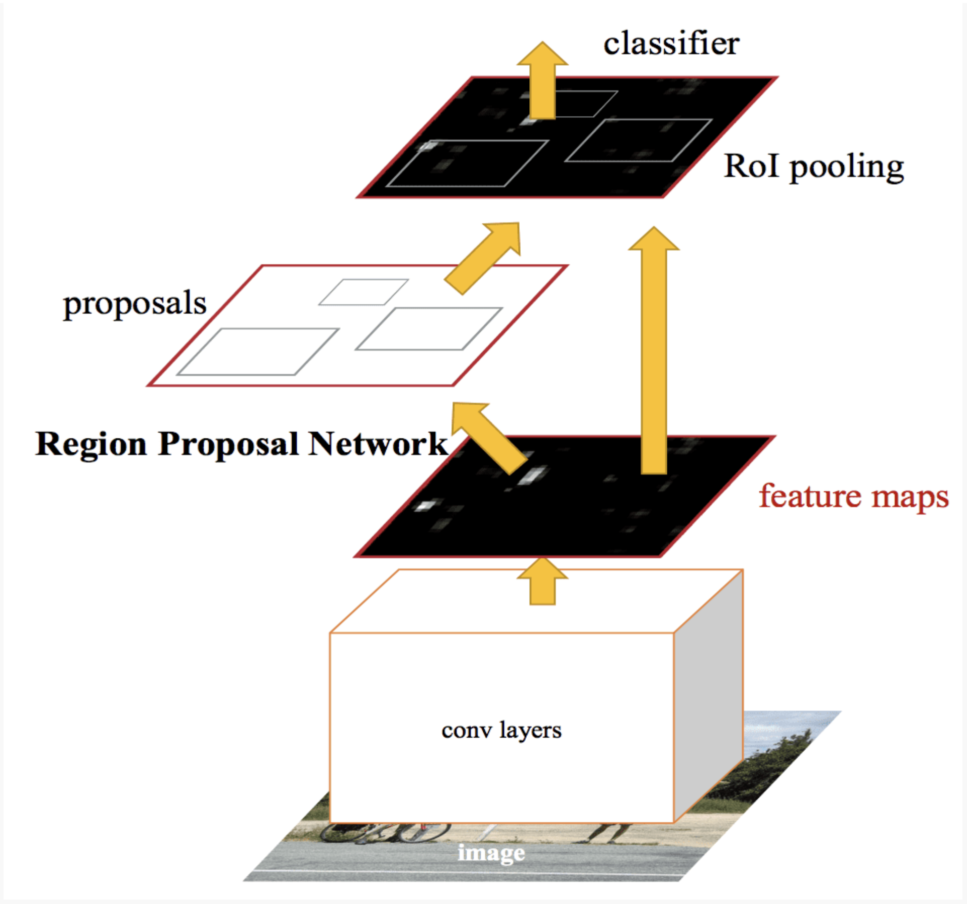


Fig 5: Summary of the Faster R-CNN Model Architecture.Taken from: Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks.

The RPN works by taking the output of a pre-trained deep CNN, such as VGG-16, and passing a small network over the feature map and outputting multiple region proposals and a class prediction for each. Region proposals are bounding boxes, based on so-called anchor boxes or pre-defined shapes designed to accelerate and improve the proposal of regions. The class prediction is binary, indicating the presence of an object, or not, so-called “*objectness*” of the proposed region.
A procedure of alternating training is used where both sub-networks are trained at the same time, although interleaved. This allows the parameters in the feature detector deep CNN to be tailored or fine-tuned for both tasks at the same time.
At the time of writing, this Faster R-CNN architecture is the pinnacle of the family of models and continues to achieve near state-of-the-art results on object recognition tasks. A further extension adds support for image segmentation, described in the paper 2017 paper “Mask R-CNN.”
Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

2.2 YOLO Model Family

Another popular family of object recognition models is referred to collectively as YOLO or “*You Only Look Once*,” developed by Joseph Redmon, et al.
The R-CNN models may be generally more accurate, yet the YOLO family of models are fast, much faster than R-CNN, achieving object detection in real-time.

2.2.1YOLO

The YOLO model was first described by Joseph Redmon, et al. in the 2015 paper titled “You Only Look Once: Unified, Real-Time Object Detection.” Note that Ross Girshick, developer of R-CNN, was also an author and contributor to this work, then at Facebook AI Research.
The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second …

— You Only Look Once: Unified, Real-Time Object Detection, 2015.

The model works by first splitting the input image into a grid of cells, where each cell is responsible for predicting a bounding box if the center of a bounding box falls within the cell. Each grid cell predicts a bounding box involving the x, y coordinate and the width and height and the confidence. A class prediction is also based on each cell.

For example, an image may be divided into a 7×7 grid and each cell in the grid may predict 2 bounding boxes, resulting in 94 proposed bounding box predictions. The class probabilities map and the bounding boxes with confidences are then combined into a final set of bounding boxes and class labels. The image taken from the paper below summarizes the two outputs of the model.

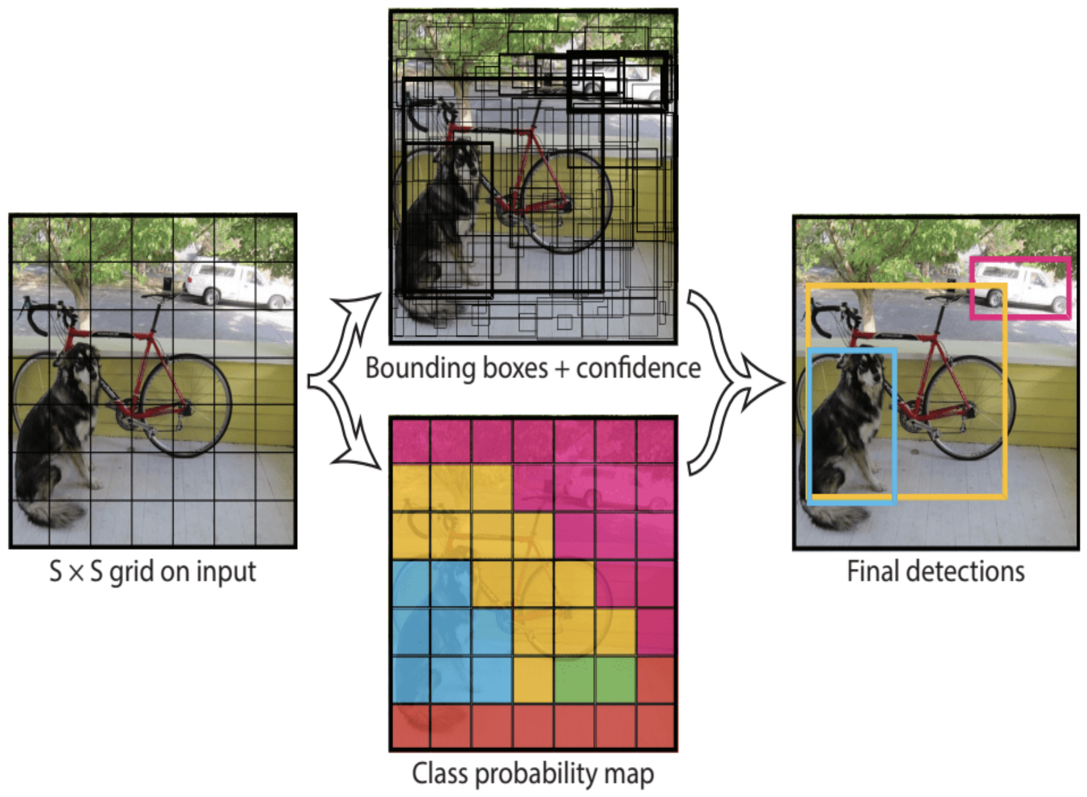


Fig 6: Summary of Predictions made by YOLO Model Taken from: You Only Look Once: Unified, Real-Time Object Detection

2.2.2 YOLOv2 (YOLO9000) and YOLOv3

The model was updated by Joseph Redmon and Ali Farhadi in an effort to further improve model performance in their 2016 paper titled "YOLO9000: Better, Faster, Stronger."

Although this variation of the model is referred to as YOLO v2, an instance of the model is described that was trained on two object recognition datasets in parallel, capable of predicting 9,000 object classes, hence given the name "*YOLO9000*."

A number of training and architectural changes were made to the model, such as the use of batch normalization and high-resolution input images.

Like Faster R-CNN, YOLOv2 model makes use of anchor boxes, pre-defined bounding boxes with useful shapes and sizes that are tailored during training. The choice of bounding boxes for the image is pre-processed using a k-means analysis on the training dataset.

Importantly, the predicted representation of the bounding boxes is changed to allow small changes to have a less dramatic effect on the predictions, resulting in a more stable model. Rather than predicting position and size directly, offsets are predicted for moving and reshaping the pre-defined anchor boxes relative to a grid cell and dampened by a logistic function.

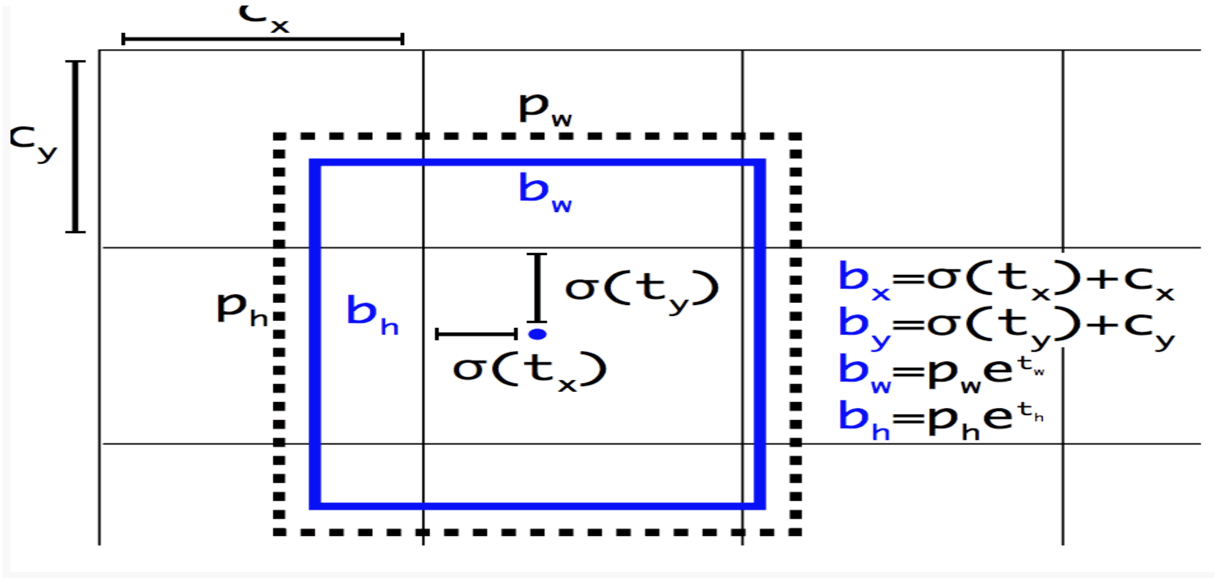


Fig 7: Example of the Representation Chosen when Predicting Bounding Box Position and ShapeTaken from: YOLO9000: Better, Faster, Stronger

Further improvements to the model were. The improvements were reasonably minor, including a deeper feature detector network and minor representational changes.

3. Reinforcement Learning

3.1 Q-Learning

One of my favorite algorithms that I learned while taking a reinforcement learning course was q-learning. Probably because it was the easiest for me to understand and code, but also because it seemed to make sense. In this quick post I'll discuss q-learning and provide the basic background to understanding the algorithm.

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

When q-learning is performed we create what's called a *q-table* or matrix that follows the shape of [state, action] and we initialize our values to zero. We then update and store our *q-values* after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value.

The next step is simply for the agent to interact with the environment and make updates to the state action pairs in our q-table Q[state, action].

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as ***exploiting*** since we use the information we have available to us to make a decision.

The second way to take action is to act randomly. This is called ***exploring***. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process. You can balance exploration/exploitation using epsilon (ε) and setting the value of how often you want to explore vs exploit. Here's some rough code that will depend on how the state and action space are setup.

The updates occur after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent. A terminal state for example can be anything like landing on a checkout page, reaching the end of some game, completing some desired objective, etc. The agent will not learn much after a single episode, but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values or q-star (Q∗).

Here are the 3 basic steps:

1. Agent starts in a state (s1) takes an action (a1) and receives a reward (r1)
2. Agent selects action by referencing Q-table with highest value (max) **OR** by random (epsilon, ε)
3. Update q-values

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}}\bigg)}^{\text{temporal difference}}$$

In the update above there are a couple variables that we haven’t mentioned yet. Whats happening here is we adjust our q-values based on the difference between the discounted new values and the old values. We discount the new values using gamma and we adjust our step size using learning rate (lr). Below are some references.

Learning Rate: lr or learning rate, often referred to as *alpha* or α, can simply be defined as how much you accept the new value vs the old value. Above we are taking the difference between new and old and then multiplying that value by the learning rate. This value then gets added to our previous q-value which essentially moves it in the direction of our latest update.

Gamma: gamma or γ is a discount factor. It’s used to balance immediate and future reward. From our update rule above you can see that we apply the discount to the future reward. Typically this value can range anywhere from 0.8 to 0.99.

Reward: reward is the value received after completing a certain action at a given state. A reward can happen at any given time step or only at the terminal time step.

Max: np.max() uses the numpy library and is taking the maximum of the future reward and applying it to the reward for the current state. What this does is impact the current action by the possible future reward. This is the beauty of q-learning. We're allocating future reward to current actions to help the agent select the highest return action at any given state.

3.2. Deep Q- Learning

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our agent. This helps the agent figure out exactly which action to perform.

Imagine an environment with 10,000 states and 1,000 actions per state. This would create a table of 10 million cells. Things will quickly get out of control!

It is pretty clear that we can't infer the Q-value of new states from already explored states. This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

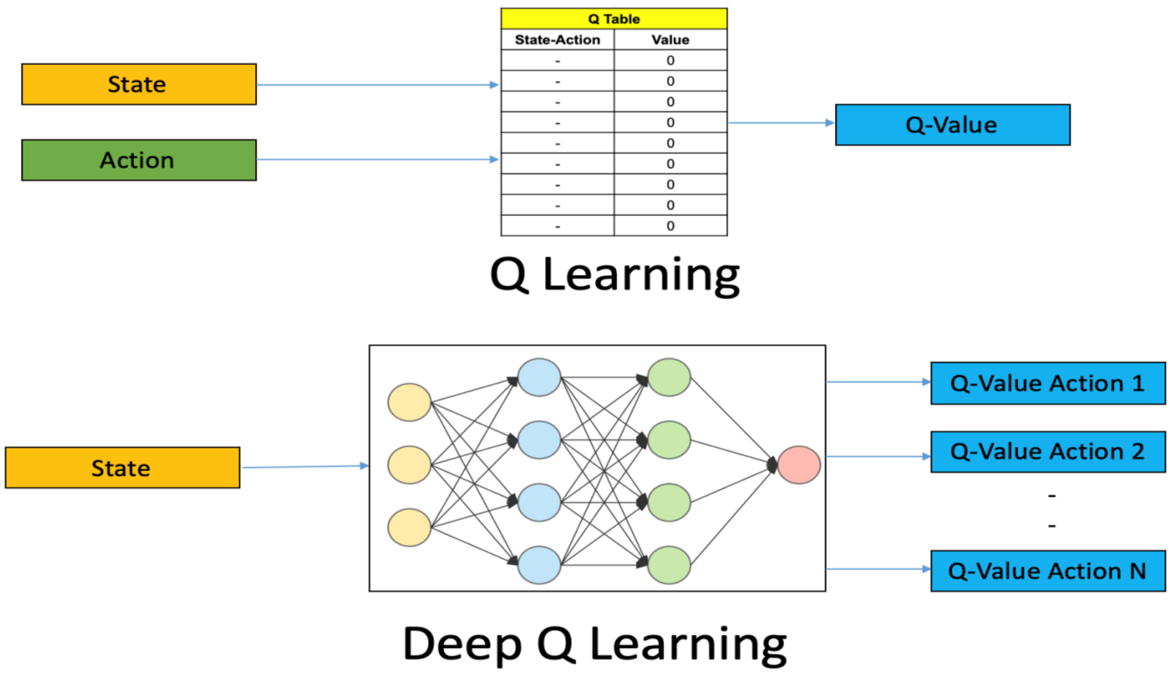


Fig 8: Deep Q- Network

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our agent. This helps the agent figure out exactly which action to perform.

Imagine an environment with 10,000 states and 1,000 actions per state. This would create a table of 10 million cells. Things will quickly get out of control!

It is pretty clear that we can’t infer the Q-value of new states from already explored states. This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

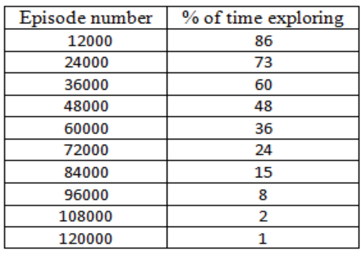
4. Research Design

The model we are trying to build will take in image as inputs. Our first job is to segment out the important parts from the image. Our objective is to classify the image. The elementary idea is to make the important features as rewards in such a way that the agent can learn to differentiate among them. Our objective is to train the agent to learn the distribution of rewards before making any prediction. Different objects have different features. Our aim is to assign different reward distribution on different object features so that the agent can distinguish between them.

The datasets used are as follows:

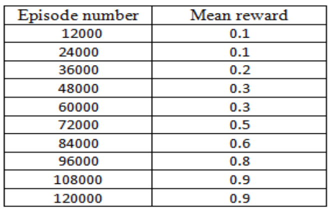
- MNIST- It consisting of training set of 60,000 examples and a test set of 10,000 examples. The sizes are 28*28 in dimension. It consists of hand written decimal digits. These are grey scale images.
- Fashion-MNIST-consisting of a training set of 60,000 examples and a test set of 10,000 examples. The sizes are 28*28 in dimension. It consists of many grey scale images of different cloths.
- CIFAR 10-consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

5. Experimental Results:



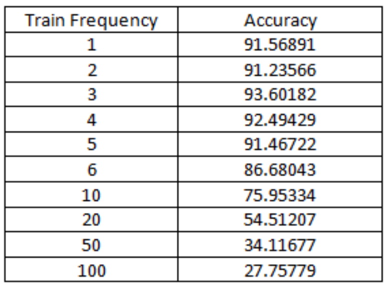
Episode number	% of time exploring
12000	86
24000	73
36000	60
48000	48
60000	36
72000	24
84000	15
96000	8
108000	2
120000	1

Table 1: % of time exploring to Episode number having 32 layers for MNIST

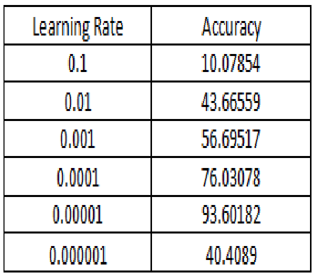


Episode number	Mean reward
12000	0.1
24000	0.1
36000	0.2
48000	0.3
60000	0.3
72000	0.5
84000	0.6
96000	0.8
108000	0.9
120000	0.9

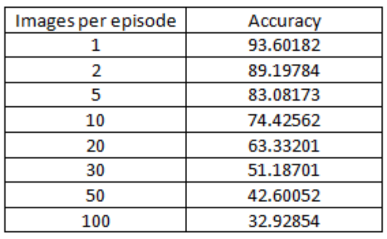
Table 2: Mean reward to Episode number having 32 layers for MNIST



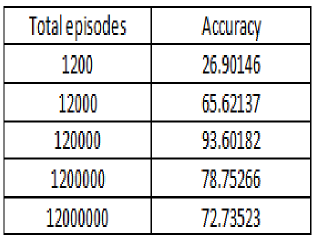
Train Frequency	Accuracy
1	91.56891
2	91.23566
3	93.60182
4	92.49429
5	91.46722
6	86.68043
10	75.95334
20	54.51207
50	34.11677
100	27.75779



Learning Rate	Accuracy
0.1	10.07854
0.01	43.66559
0.001	56.69517
0.0001	76.03078
0.00001	93.60182
0.000001	40.4089



Images per episode	Accuracy
1	93.60182
2	89.19784
5	83.08173
10	74.42562
20	63.33201
30	51.18701
50	42.60052
100	32.92854



Total episodes	Accuracy
1200	26.90146
12000	65.62137
120000	93.60182
1200000	78.75266
12000000	72.73523

Table 3, 4, 5, 6: Tuning hyper parameters 32 layers for MNIST

Episode number	% of time exploring
12000	86
24000	73
36000	60
48000	48
60000	36
72000	24
84000	15
96000	8
108000	2
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.2
48000	0.4
60000	0.5
72000	0.5
84000	0.7
96000	0.8
108000	0.9
120000	0.9

Table 7, 8: Training MNIST dataset for 64 layers

Train Frequency	Accuracy
1	91.78163
2	92.38017
3	93.70136
4	94.85517
5	90.67432
6	82.79151
10	71.06327
20	42.79238
50	32.62619
100	25.00445

Learning Rate	Accuracy
0.1	19.83421
0.01	56.59321
0.001	59.51245
0.0001	78.1715
0.00001	94.85517
0.000001	56.41601

Images per episode	Accuracy
1	94.85517
2	87.80471
5	81.89763
10	78.36173
20	66.93662
30	59.04314
50	55.66792
100	32.24361

Total episodes	Accuracy
1200	56.6024
12000	72.31415
120000	94.85517
1200000	72.52651
12000000	64.71557

Table 9, 10, 11, 12: Tuning hyper parameters 32 layers for MNIST

Episode number	% of time exploring
12000	86
24000	73
36000	60
48000	48
60000	36
72000	24
84000	15
96000	8
108000	2
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.3
48000	0.5
60000	0.5
72000	0.6
84000	0.8
96000	0.9

Table 13, 14: Training MNIST dataset for 128 layers

Train Frequency	Accuracy
1	91.38155
2	93.05276
3	93.6702
4	94.32909
5	91.36747
6	87.80681
10	78.93591
20	44.57337
50	34.91581
100	21.99582

Learning Rate	Accuracy
0.1	26.17196
0.01	65.0098
0.001	69.73393
0.0001	81.64615
0.00001	94.32909
0.000001	66.98808

Images per episode	Accuracy
1	94.32909
2	89.74553
5	71.95981
10	69.39567
20	66.32291
30	63.00537
50	62.3289
100	61.6135

Total episodes	Accuracy
1200	45.41946
12000	71.14696
120000	94.32909
1200000	71.34724
12000000	48.27453

Table 15, 16, 17, 18: Tuning hyper parameters 128 layers for MNIST

Episode number	% of time exploring
12000	89
24000	77
36000	65
48000	52
60000	40
72000	27
84000	14
96000	6
108000	1
120000	1

Episode number	Mean reward
12000	0.1
24000	0.1
36000	0.1
48000	0.2
60000	0.3
72000	0.5
84000	0.7
96000	0.8
108000	0.8
120000	0.8

Table 19, 20: Training Fashion MNIST dataset for 32 layers

Train Frequency	Accuracy
1	88.46856
2	89.9884
3	90.21101
4	90.6871
5	91.54747
6	86.45962
10	71.22571
20	67.63211
50	34.50285
100	13.14966

Learning Rate	Accuracy
0.1	10.79407
0.01	32.90121
0.001	43.80501
0.0001	69.12081
0.00001	91.54747
0.000001	50.52012

Images per episode	Accuracy
1	91.54747
2	88.14966
5	82.1911
10	76.77927
20	71.74574
30	51.25131
50	47.49979
100	43.36186

Total episodes	Accuracy
1200	12.39733
12000	51.8221
120000	91.54747
1200000	71.73707
12000000	42.45095

Table 21, 22, 23, 24: Tuning hyper parameters 32 layers for Fashion MNIST

Episode number	% of time exploring
12000	89
24000	77
36000	65
48000	52
60000	40
72000	27
84000	14
96000	6
108000	1
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.2
48000	0.4
60000	0.4
72000	0.5
84000	0.6
96000	0.8
108000	0.8
120000	0.9

Table 25, 26: Training Fashion MNIST dataset for 64 layers

Train Frequency	Accuracy
1	82.98905
2	86.30369
3	90.15648
4	91.00464
5	92.05257
6	81.17799
10	61.78799
20	52.07123
50	32.63339
100	21.36565

Learning Rate	Accuracy
0.1	12.98702
0.01	31.32639
0.001	45.29092
0.0001	76.56899
0.00001	92.05257
0.000001	59.73936

Images per episode	Accuracy
1	92.05257
2	88.67527
5	82.89589
10	78.33735
20	66.86143

Total episodes	Accuracy
1200	25.14687
12000	61.78725
120000	92.05257
1200000	69.39721
12000000	54.60609

Table 27, 28, 23, 30: Tuning hyper parameters 64 layers for Fashion MNIST

Episode number	% of time exploring
12000	89
24000	77
36000	65
48000	52
60000	40
72000	27
84000	14
96000	6
108000	1
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.3
48000	0.4
60000	0.5
72000	0.7
84000	0.8
96000	0.8
108000	0.8
120000	0.9

Table 31, 32: Training Fashion MNIST dataset for 128 layers

Train Frequency	Accuracy
1	81.99049
2	83.02519
3	86.28041
4	91.06244
5	91.45475
6	87.151
10	78.77334
20	44.40218
50	34.73898
100	18.57338

Learning Rate	Accuracy
0.1	23.09098
0.01	65.71937
0.001	71.74383
0.0001	83.49622
0.00001	91.45475
0.000001	76.07718

Images per episode	Accuracy
1	91.45475
2	88.1827
5	87.50318
10	79.86987
20	66.53733
30	61.66429
50	59.85004
100	51.27123

Total episodes	Accuracy
1200	31.50948
12000	62.89095
120000	91.45475
1200000	61.98145
12000000	31.40575

Table 33, 34, 35, 36: Tuning hyper parameters 128 layers for Fashion MNIST

Episode number	% of time exploring
12000	96
24000	87
36000	75
48000	65
60000	50
72000	34
84000	24
96000	12
108000	8
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.2
48000	0.2
60000	0.3
72000	0.4
84000	0.8
96000	0.8
108000	0.8
120000	0.9

Table 37, 38: Training CIFAR 10 dataset for 32 layers

Train Frequency	Accuracy
1	69.36852
2	71.79803
3	82.50724
4	86.55666
5	88.98145
6	85.30558
10	79.19763
20	54.4877
50	32.94799
100	15.1503

Learning Rate	Accuracy
0.1	10.79407
0.01	32.90121
0.001	43.80501
0.0001	69.12081
0.00001	91.54747
0.000001	50.52012

Images per Episode	Accuracy
1	88.98145
2	85.52223
5	81.76612
10	75.13189
20	64.2199
30	51.25844
50	31.7128
100	17.87742

Number of Episodes	Accuracy
1200	12.39733
12000	51.8221
120000	91.54747
1200000	71.73707
12000000	42.45095

Table 39, 40, 41, 42: Tuning hyper parameters 32 layers for CIFAR 10

Train Frequency	Accuracy
1	70.64389
2	81.73339
3	85.08333
4	88.95083
5	89.9757
6	73.53051
10	52.39667
20	48.21173
50	31.28131
100	9.993952

Images per Episode	Accuracy
1	89.9757
2	86.41772
5	81.09521
10	73.54515
20	67.39791
30	51.66275
50	41.41721
100	15.41427

Table 43, 44: Training CIFAR 10 dataset for 64 layers

Train Frequency	Accuracy
1	70.64389
2	81.73339
3	85.08333
4	88.95083
5	89.9757
6	73.53051
10	52.39667
20	48.21173
50	31.28131
100	9.993952

Learning Rate	Accuracy
0.1	11.74753
0.01	32.9901
0.001	43.43693
0.0001	51.67746
0.00001	89.9757
0.000001	57.13164

Images per Episode	Accuracy
1	89.9757
2	86.41772
5	81.09521
10	73.54515
20	67.39791
30	51.66275
50	41.41721
100	15.41427

Number of Episodes	Accuracy
1200	6.711638
12000	21.35
120000	37.44996
1200000	89.9757
12000000	47.28665

Table 45, 46, 47, 48: Tuning hyper parameters 64 layers for CIFAR 10

Episode number	% of time exploring
12000	96
24000	87
36000	75
48000	65
60000	50
72000	34
84000	24
96000	12
108000	8
120000	1

Episode number	Mean reward
12000	0.1
24000	0.2
36000	0.3
48000	0.3
60000	0.6
72000	0.7
84000	0.8
96000	0.9
108000	0.9
120000	0.9

Table 49, 50: Training CIFAR 10 dataset for 128 layers

Train Frequency	Accuracy
1	69.75429
2	74.41851
3	83.88016
4	87.93582
5	90.40575
6	85.29172
10	72.74997
20	61.74997
50	21.85647
100	8.663291

Learning Rate	Accuracy
0.1	16.51081
0.01	34.0425
0.001	38.12964
0.0001	55.49151
0.00001	90.40575
0.000001	50.40953

Images per Episode	Accuracy
1	90.40575
2	85.10805
5	73.77207
10	54.69588
20	41.84165
30	35.54282
50	31.81977
100	17.75829

Number of Episodes	Accuracy
1200	8.283292
12000	28.7285
120000	43.9814
1200000	90.40575
12000000	64.20678

Table 51, 52, 53, 54: Tuning hyper parameters 128 layers for CIFAR 10

Steps	Accuracy
10	45.76105
20	78.95763
30	96.50184
40	56.50947
50	38.88259

Table 55: Result of POP approach with various number of steps and accuracy in MNIST Dataset

Steps	Accuracy
10	55.29492
20	88.27731
30	73.67296
40	41.42803
50	25.41018

Table 56: Result of POP approach with various number of steps and accuracy in Fashion MNIST Dataset

Steps	Accuracy
10	41.24125
20	83.36331
30	81.43762
40	76.35587
50	63.65443

Table 57: Result of POP approach with various number of steps and accuracy in CIFAR 10 Dataset

6. Graphical Results:

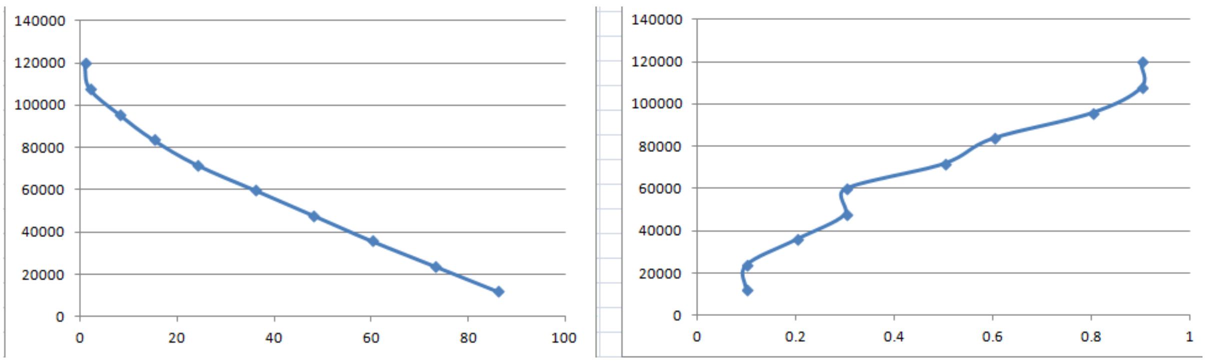


Fig 9, 10: Graph for Table 1, 2. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively

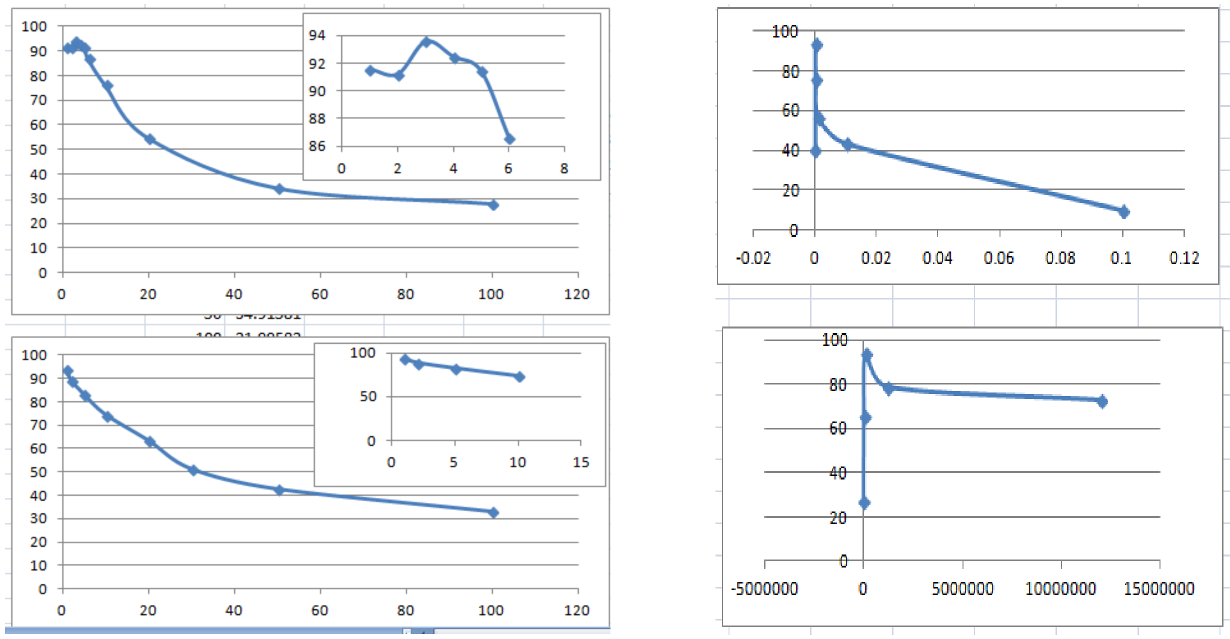
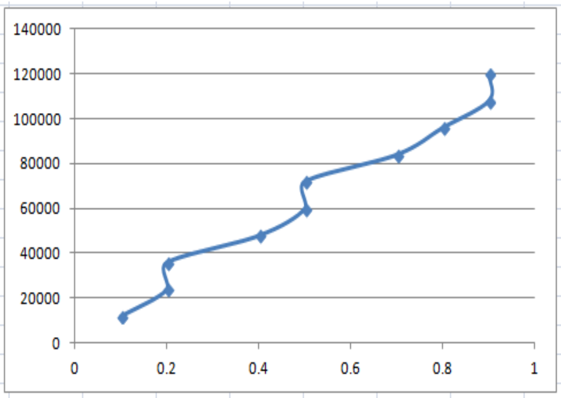


Fig 11, 12, 13, 14: Graph for Table 3, 4, 5, 6. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



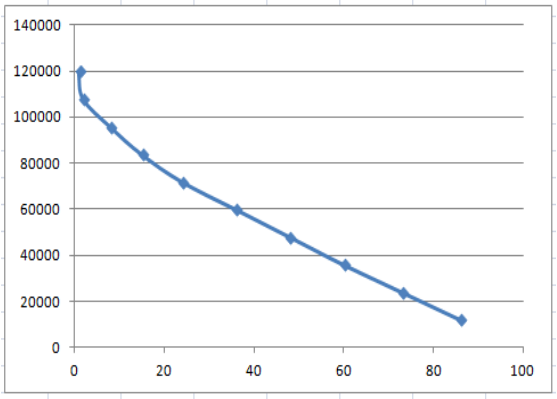
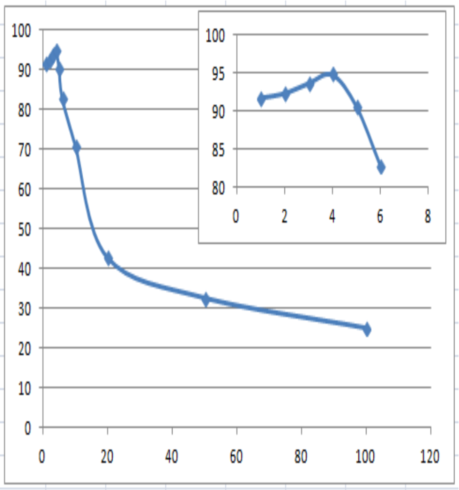
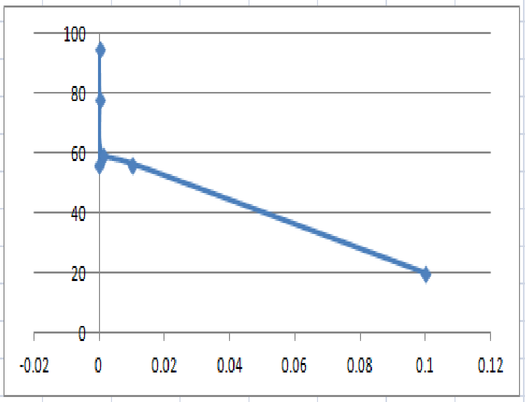
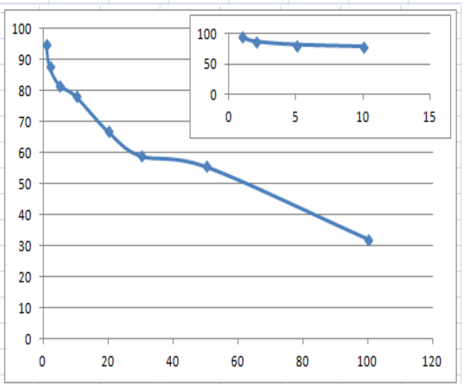


Fig 15, 16: Graph for Table 7, 8. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







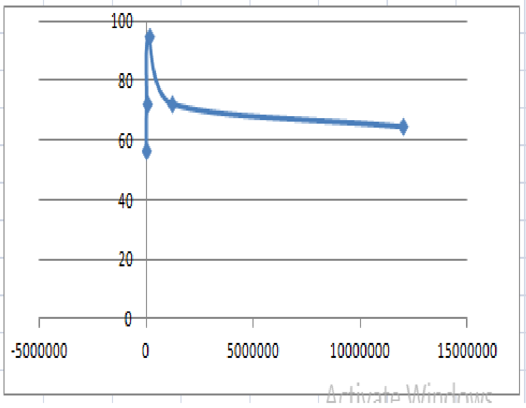
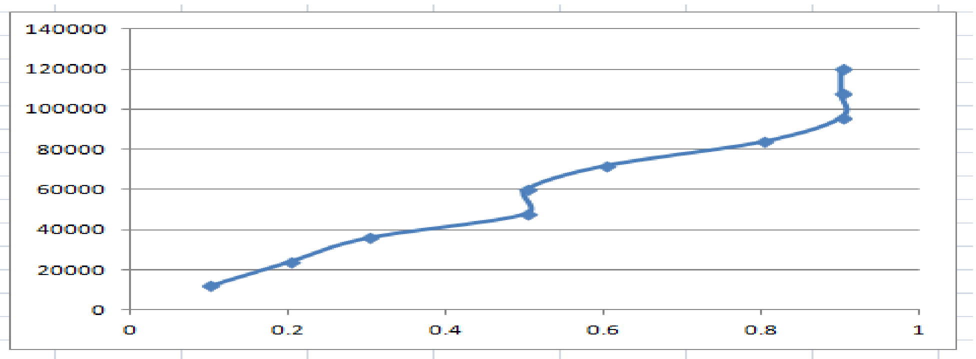


Fig 17, 18, 19, 20: Graph for Table 9, 10, 11, 12. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes



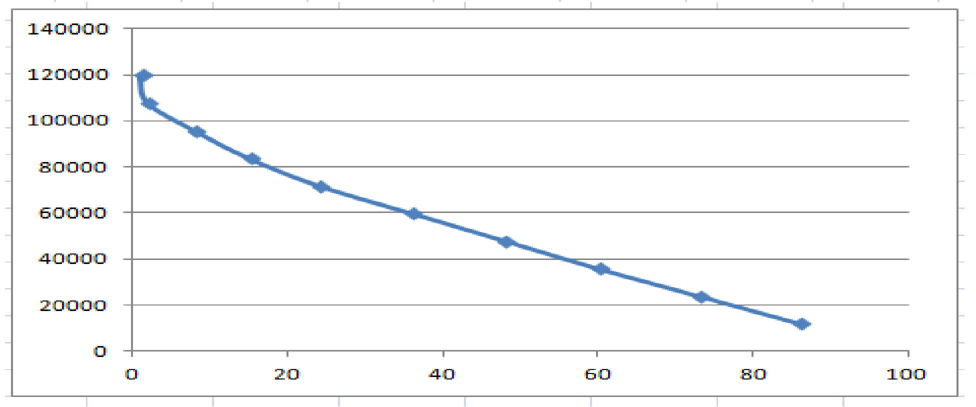
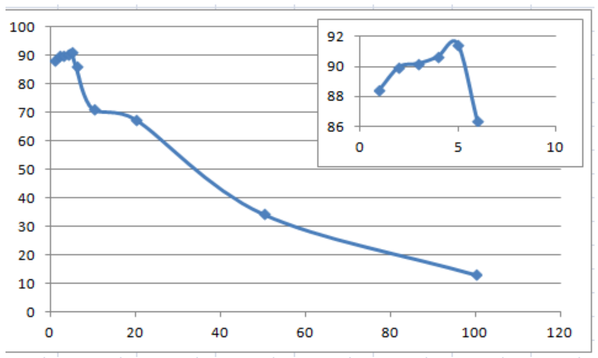
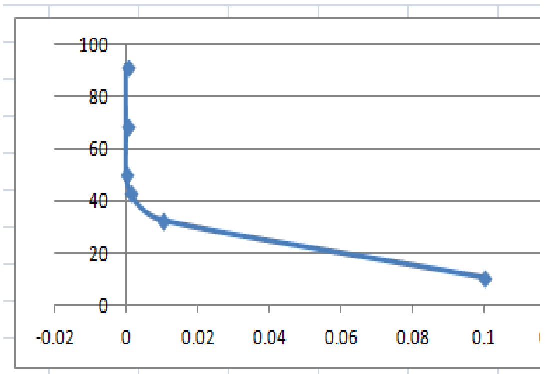
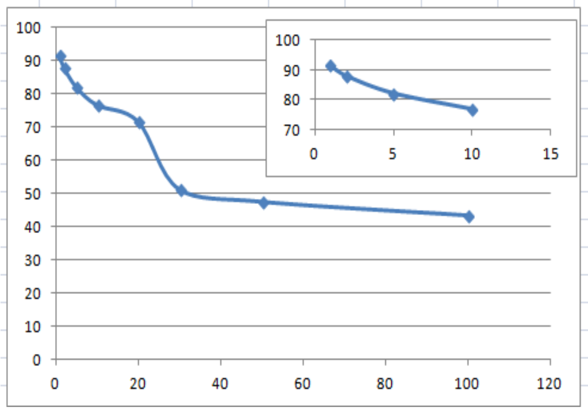


Fig 21, 22: Graph for Table 13, 14. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







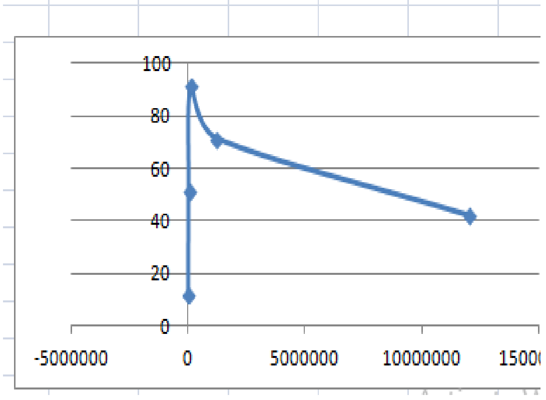
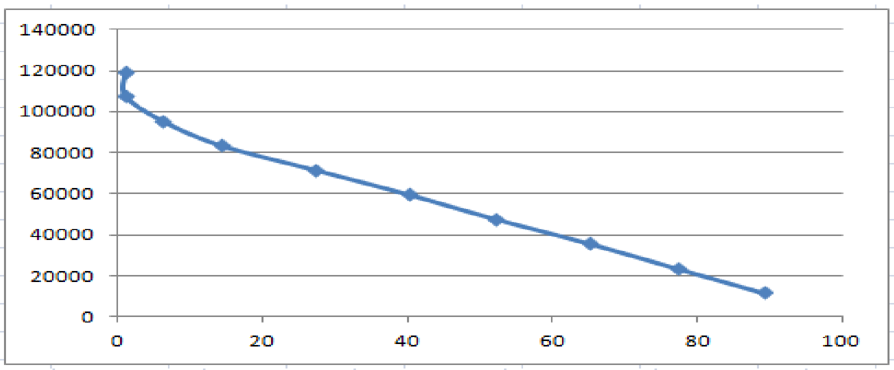


Fig 23, 24, 25, 26: Graph for Table 15, 16, 17, 18. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



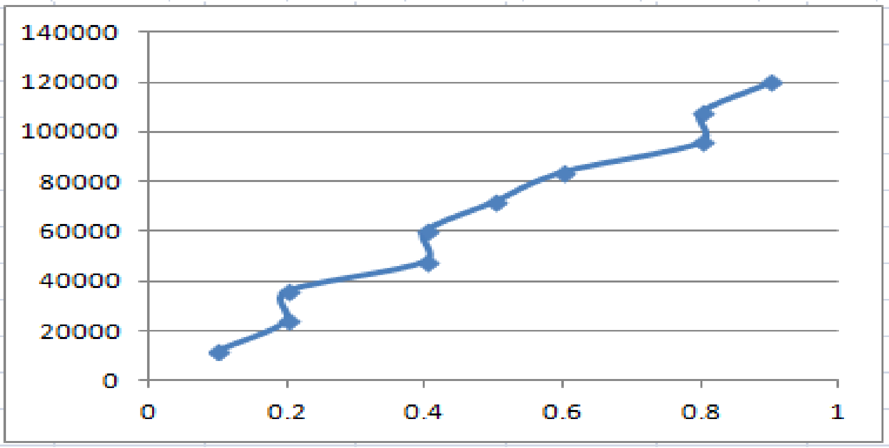
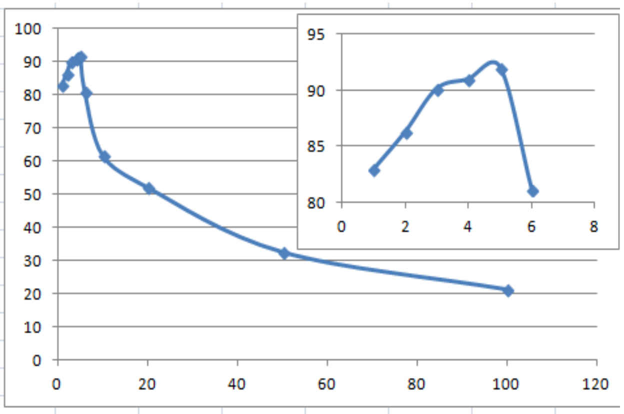
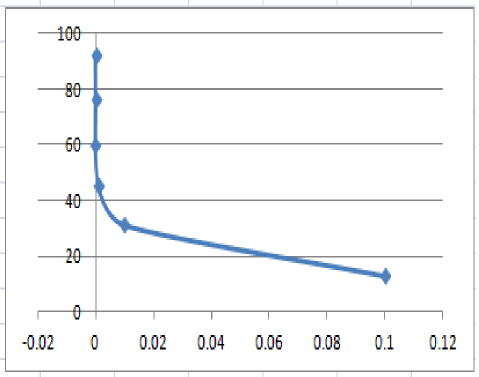
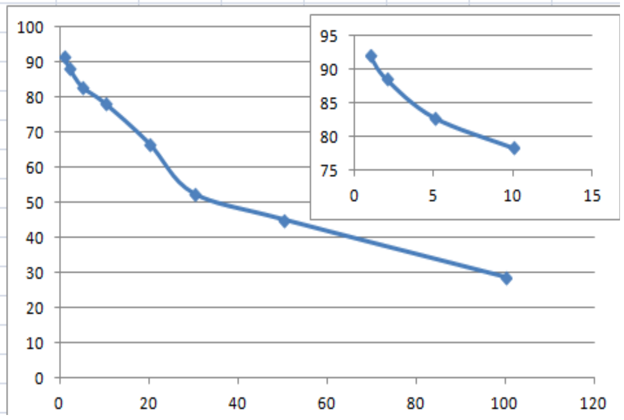


Fig 27, 28: Graph for Table 19, 20. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







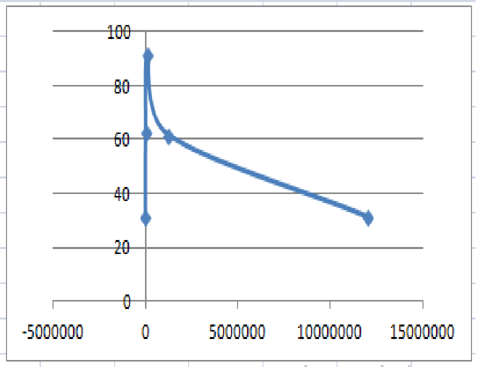
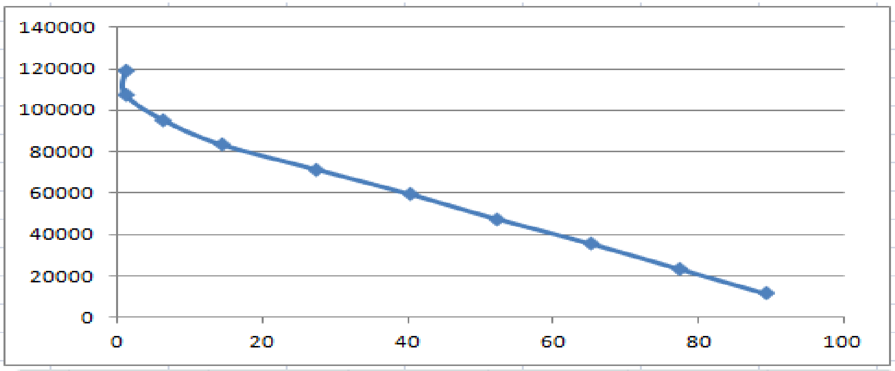


Fig 29, 30, 31, 32: Graph for Table 21, 22, 23, 24. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



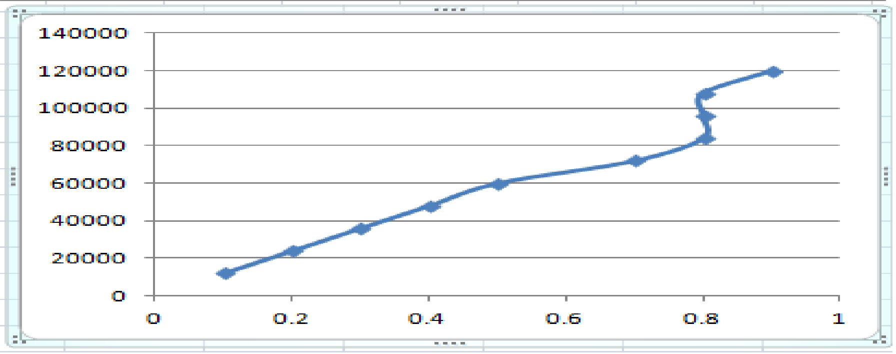
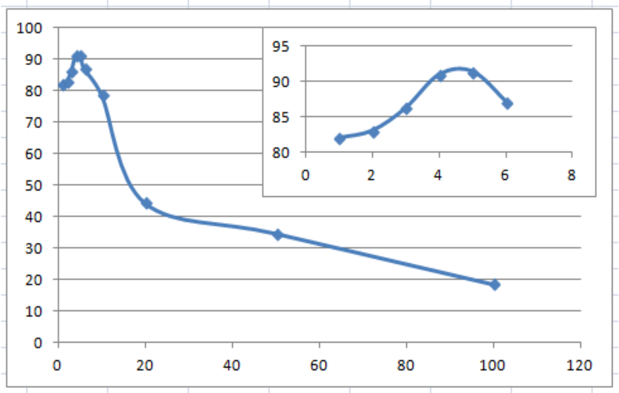
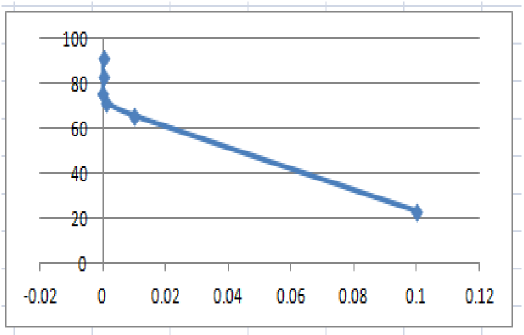
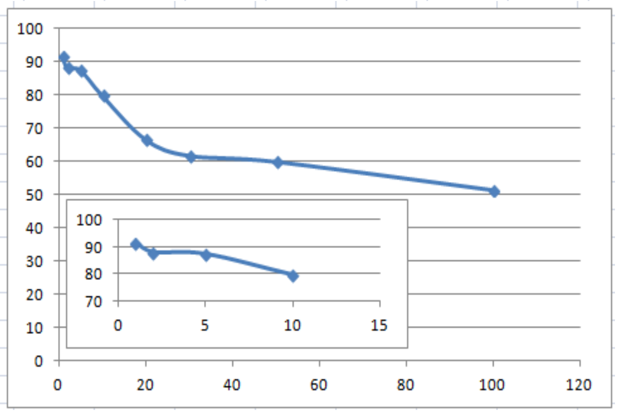


Fig 33, 34: Graph for Table 25, 26. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







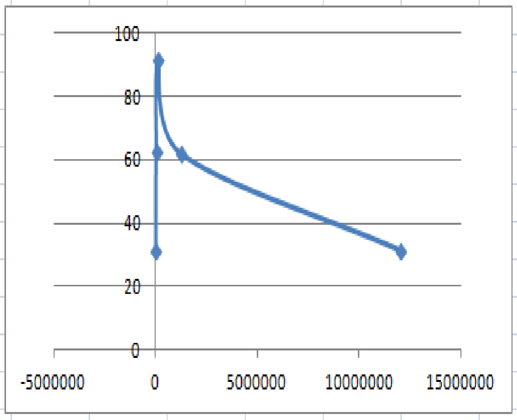
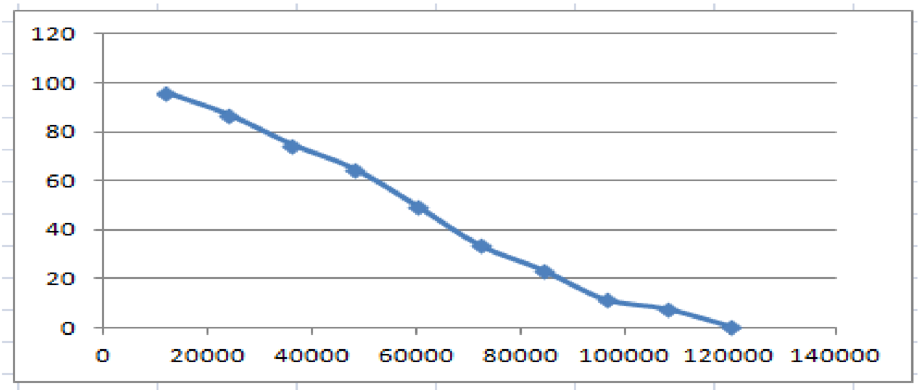


Fig 35, 36, 37, 38: Graph for Table 27, 28, 29, 30. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



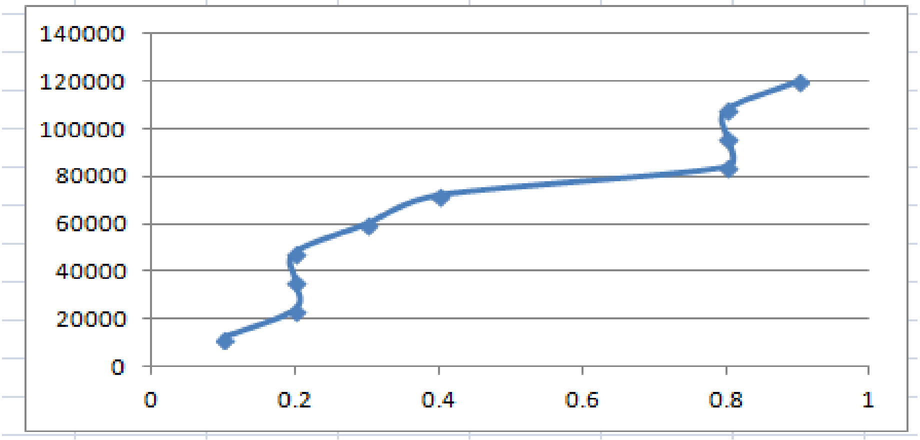
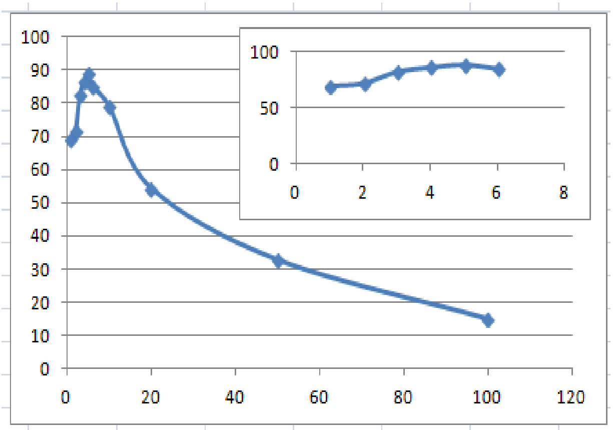
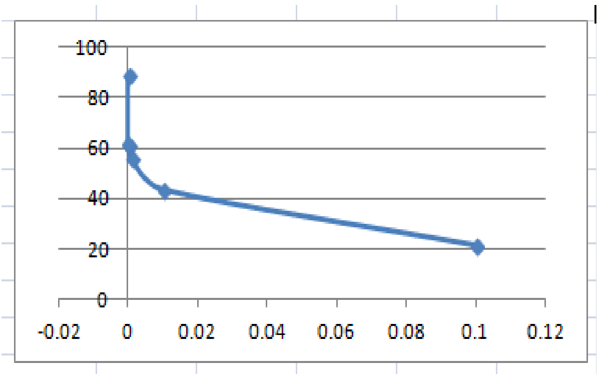
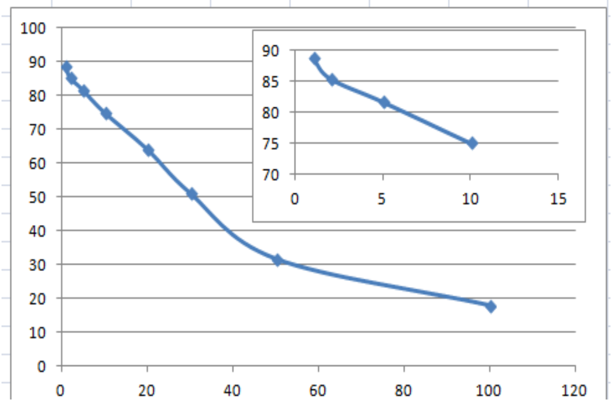


Fig 39, 40: Graph for Table 31, 32. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







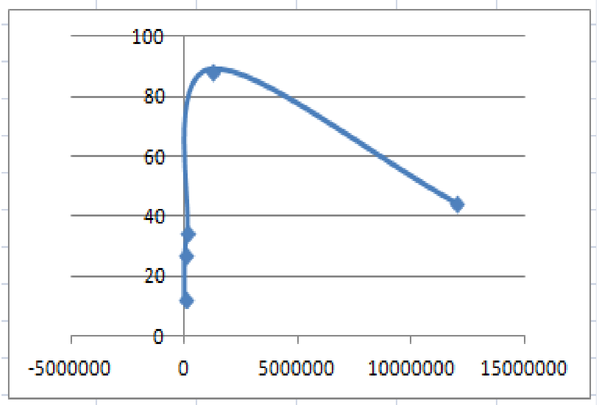
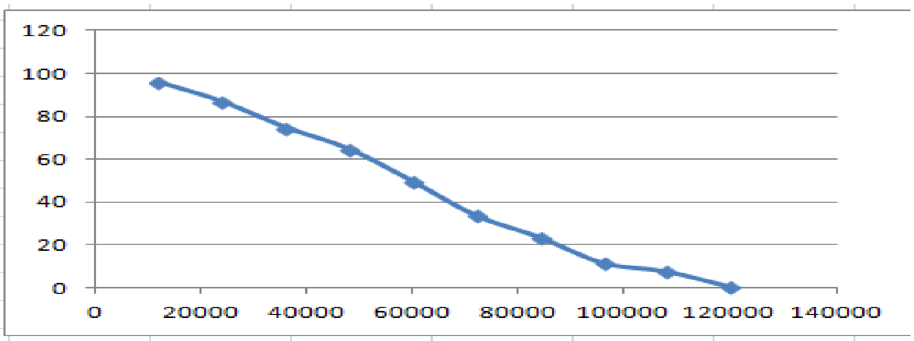


Fig 41, 42, 43, 44: Graph for Table 33, 34, 35, 36. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



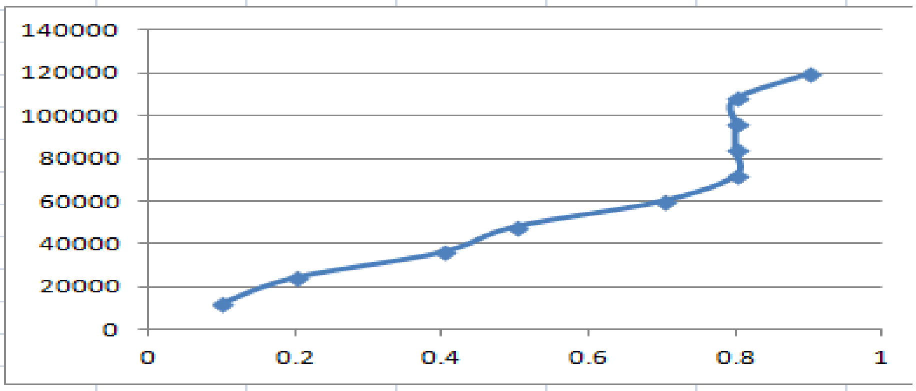
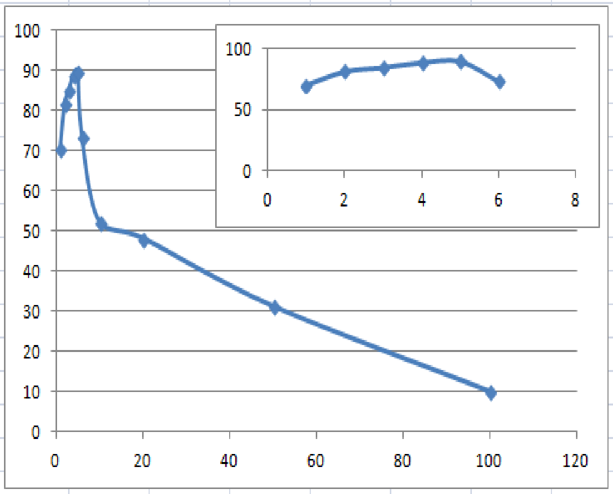
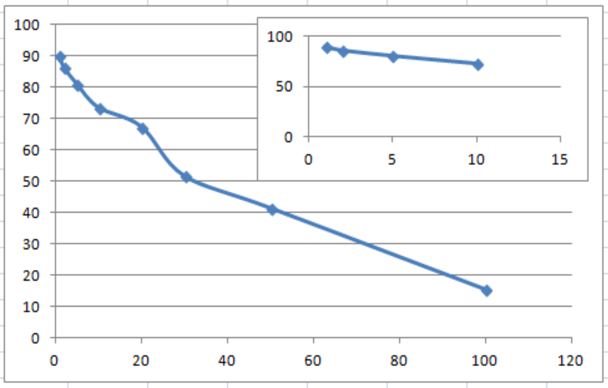
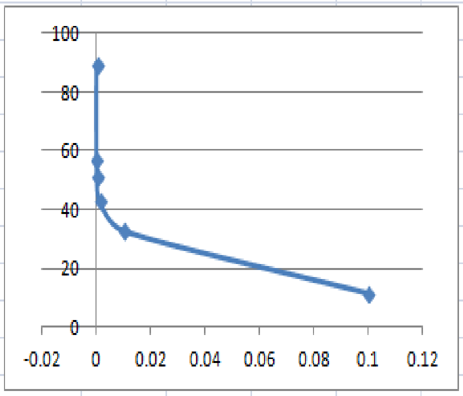


Fig 45, 46: Graph for Table 37, 38. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







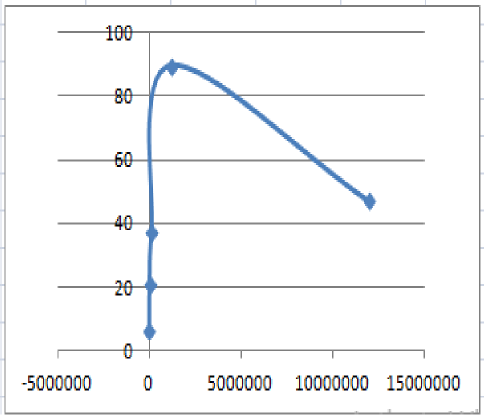
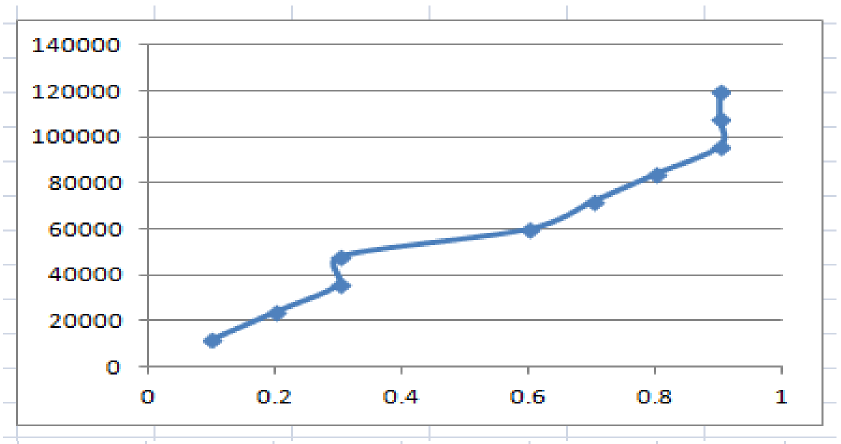


Fig 47, 48, 49, 50: Graph for Table 39, 40, 41, 42. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively



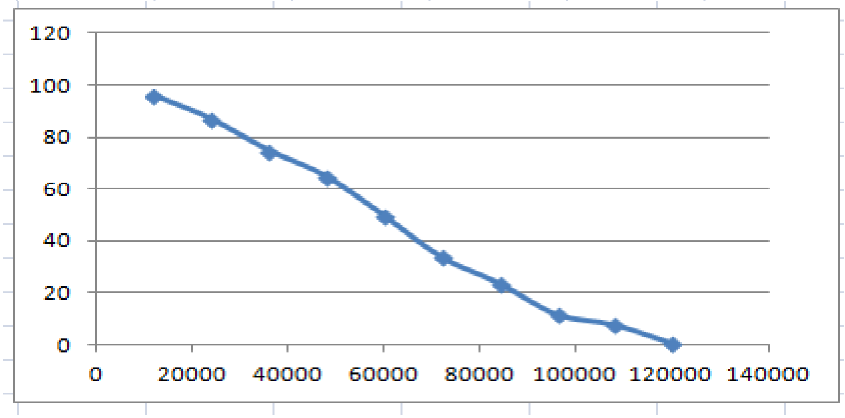
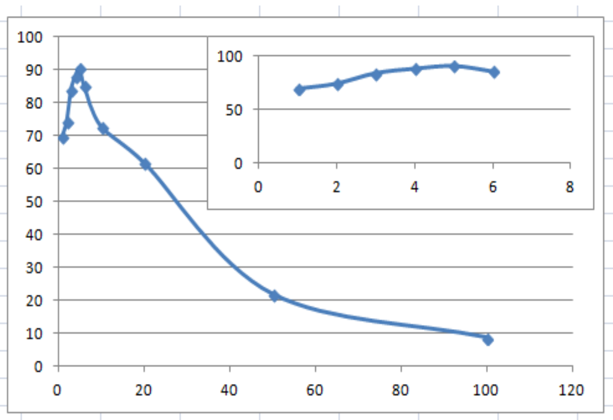
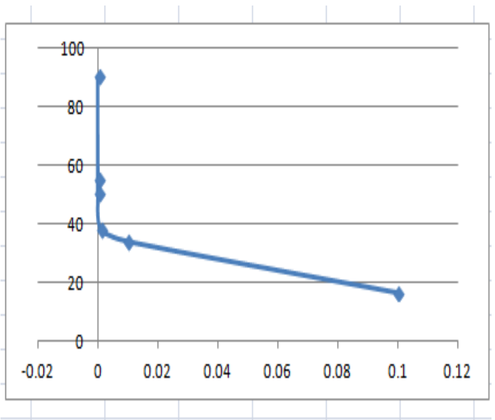
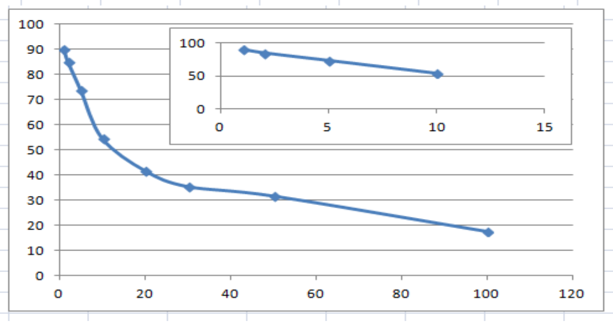


Fig 51, 52: Graph for Table 43, 44. Y Axis represents the number of Episodes and X axis represents the % of time exploring by the system and Mean Reward per episode respectively







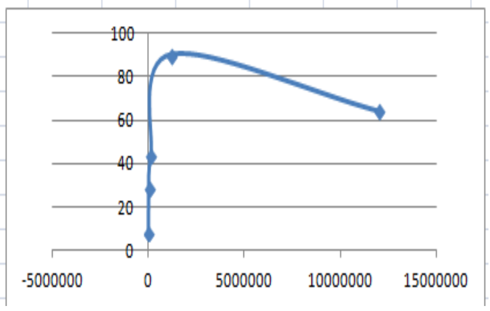


Fig 53, 54, 55, 56: Graph for Table 45, 46, 47, 48. Y Axis represents the Accuracy and X axis represents the training frequency, images per episode, learning rate, number of episodes respectively

7. Conclusion:

Since very less research has been done regarding this topic and the results we have achieved, we can say that the work was a success. We plan on to tune the various hyper parameters more in order to improve the results. Also we need to run our model on more datasets. This is a part of our roadmap.

References

- https://machinelearningmastery.com/object-recognition-with-deep-learning/

- https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/

- Active Learning for Image Classification: A Deep Reinforcement Learning Approach
 https://ieeexplore.ieee.org/abstract/document/8901911
Le Sun, Yihong Gong

- Multi-Agent Image Classification via Reinforcement Learning

  https://engineering.lehigh.edu/sites/engineering.lehigh.edu/files/_DEPARTMENTS/ise/pdf/tech-papers/19/19T_007.pdf
Hossein K. Mousavi , Mohammadreza Nazari , Martin Taka´cˇ , and Nader Motee, Lehigh University

- Outline Objects using Deep Reinforcement Learning
 https://arxiv.org/pdf/1804.04603.pdf
Zhenxin Wang, Sayan Sarcar, Kochi University of Technology

- Selective Search for Object Recognition
https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf
J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers , and A.W.M. Smeulders, University of Trento,2012, submitted to IJCV