8-2021

An Empirical and Theoretical Investigation of Random Reinforced Forests and Shallow Convolutional Neural Networks

Nikhil Ganta
Utah State University

AN EMPIRICAL AND THEORETICAL INVESTIGATION OF RANDOM REINFORCED FORESTS AND SHALLOW CONVOLUTIONAL NEURAL NETWORKS

by

Nikhil Ganta

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Vladimir Kulyukin, Ph.D. Major Professor	Nicholas Flann, Ph.D. Committee Member
Kevin Moon, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

ABSTRACT

An Empirical and Theoretical Investigation of Random Reinforced Forests and Shallow Convolutional Neural Networks

by

Nikhil Ganta, Master of Science

Utah State University, 2021

Major Professor: Vladimir Kulyukin, Ph.D.
Department: Computer Science

We perform an investigation comparing random forests generated by adding the decision trees trained on local receptive fields of a given image dataset through reinforcement learning (based on Q-value) or performance-based (based on individual accuracy) with shallow convolutional neural networks of a similar memory footprint. The comparison between them is based on their classification performance (validation accuracies), the number of parameters and basic operations performed for classifying a single image. Two types of Q-Learning algorithms are discussed, named *Hidden* with pre-defined states and *Dynamic* with zero initial states. Furthermore, a theorem is proved, why Dynamic Q-Learning is unlikely to find an optimal solution. Finally, all the methods mentioned are trained, tested, and validated across five honeybee image datasets (total size of 232,293 images) and CIFAR-10 aiming to perform an efficient and accurate classification when deployed onto a Raspberry Pi platform in comparison to the Deep Neural Networks without sacrificing much on the accuracies.

(75 pages)

PUBLIC ABSTRACT

An Empirical and Theoretical Investigation of Random Reinforced Forests and Shallow Convolutional Neural Networks

Nikhil Ganta

For many years, the global population of honey bees has been decreasing due to inconclusive reasons resulting in the syndrome Colony Collapse Disorder (CCD). This syndrome has been plaguing bees and affecting commercial agriculture pollination since 1998. Many researchers have suggested that pesticides, in-hive chemicals, pathogens, etc., might be the causes of CCD. Researchers also believe that any changes in a beehive can disturb the bees, which may negatively affect their health. Honey bees are the most vital among all the animal pollinators contributing to approximately 30% of the world's commercial pollination services. As they are of keystone importance to their respective ecosystems, monitoring their hives is crucial for understanding the effects of CCD and enabling beekeepers to maintain the health of their hives.

As beekeepers cannot monitor their hives continuously, electronic beehive monitoring (EBM) can help them keep an eye on their hives. EBM extracts the videos, audios, temperature using cameras, microphones, sensors for observing the forager traffic (incoming and outgoing flow of the bees through the hive) to track food and nectar availability, following the sounds of the buzzing, and monitoring the abrupt temperature changes. EBM reduces the number of invasive inspections and transportation costs incurred for traveling to the beehive location. This research proposes a new technique using reinforcement learning, a method based on a reward/punishment strategy and aims at providing both accurate and energy efficient classification techniques to improve individual bee recognition in bee traffic videos.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to my major professor Dr. Vladimir Kulyukin for his continuous guidance and support for this accomplishment as a thesis. His immense knowledge and suggestions helped me all the time during my research and thesis documentation. I could not have imagined having a better advisor and mentor for my research work.

I would like to thank the rest of my thesis committee members Dr. Nicholas Flann and Dr. Kevin Moon, for providing valuable suggestions and comments in my research.

I would like to thank Anastasiia Tkachenko for the verifying and contributing information into RRF-Theorem, Complexity analysis.

I would like to thank my beloved family and friends for their continuous support and encouragement in my difficult times.

Finally, I would like to thank all the kickstarter backers and especially our BeePi Angel Backers (in alphabetical order): Prakhar Amlathe, Ashwani Chahal, Trevor Landeen, Felipe Queiroz, Dinis Quelhas, Tharun Tej Tammineni, and Tanwir Zaman. I would also like to express my gratitude to Gregg Lind, who backed both fundraisers and donated hardware for the BeePi project and Richard Waggstaff, Craig Huntzinger, and Richard Mueller for letting us use their property for deploying the hives with the BeePi monitors.

Nikhil Ganta

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

CCD	Colony Collapse Disorder
EBM	Electronic Beehive Monitoring
SHR	Single Honey Bee Region
PHR	Plural Honey Bee Region
RFID	Radio-Frequency Identification
SVM	Support Vector Machine
RF	Random Forest
HQL	Hidden Q-Learning
DQL	Dynamic Q-Learning
LRF	Local Receptive Field
CNN	Convolutional Neural Network
RRF	Random Reinforced Forest
SCNN	Shallow Convolutional Neural Network

CHAPTER 1

INTRODUCTION

1.1 Background

Honey Bees (*Apis Mellifera*) contribute to approximately 30% of the world's commercial pollination, making them the most vital among the animal pollinators [4].

A beehive population consists of a single queen bee, hundreds of drone bees responsible for mating with the queen, and thousands of worker bees accountable for collecting the food, feeding the larvae, building the honeycomb, etc. The worker bees feed on the nectar and pollen of different plants for their food resulting in the transfer of pollen.

A large-scale loss of honeybees, especially the adult worker bees, has been plaguing since 1998, resulting in the syndrome named Colony Collapse Disorder (CCD). Researchers have not yet proposed the particular cause for CCD but have suggested some of the various possibilities such as parasites, in-hive chemicals, or agricultural insecticides [5, 6]. Additionally, any manipulations in the physical environment of the beehive, such as frequent examinations, can disturb the bees leading to the weakening of the beehive colony [7]. Some of the other reasons for the decrements of the bee colonies are unusual low temperatures, pesticide exposure, or internal beehive events such as the death of the queen can abruptly decrease the forager traffic [8]. Due to its significant contribution to agricultural farming, saving them from CCD has become a higher priority.

Continuous monitoring of the beehive gives us information about any changes in the forager traffic, representing the number of incoming and outgoing bees from the beehive, queen status, hive health, abrupt changes in the temperature, food and nectar flow, etc. The beekeeper cannot always be available near the hive monitoring the changes in the forager traffic, checking for any abnormalities by performing invasive inspections.

Thus, electronic beehive monitoring comes into action, tracking the beehive continuously with minimal human support and extracting lots of information. EBM monitors the hive using cameras, microphones, and other sensors extracting valuable information like colony temperature, videos for observing the forager traffic, audios for following the buzzing sounds, and various others like humidity, radiations, etc, depending on the additional equipment used [8]. EBM can reduce the number of invasive hive inspections and reduce the transportation costs incurred for traveling to the beehive as they are generally far away from the beekeepers.

1.2 Related Work

Estivill-Castro et al. [9] present research about tracking the bees flying around macadamia trees in an outdoor environment with uncontrolled illumination using inexpensive equipment and color segmentation. The research presents a change detection technique called frame differencing based on color, assuming that the camera stays in the same position and not moving frame to frame due to external conditions. Instead of using simple difference on the grayscale images, this study performs absolute differences and Minkowsky metric (M) between the three channels (R, G, B) between consecutive frames. The values of M exhibit high differences when the bees are with a clear background and are similar when the bees fly behind the flowers. The RGB difference detects the bees flying in the background of the flowers due to its transparency. The research mentions that the algorithm uses threshold values of 105 and 75 and detects a bee flying with a clear background but not the bees flying on the flowers. Using a lower threshold detects movements of leaves and flowers, which was misleading for detecting the bees. Placing the camera near the hive (top or front) can avoid unnecessary tracking of leaves/flowers and focus only on the bees.

Kimura et al. [10] performs a research on extracting the behavioral data of the honey bees using the vector quantization method, which enabled the segmentation of the bee bodies in frames of the video recordings. This research was able to identify 72% of the bees and the active regions using the trajectories in the hive. First, the bee body regions are extracted using vector quantization called the honeybee-code image. Two types of regions, named single honeybee region (SHR) and plural honeybee region (PHR), are extracted using morphological information like body size and shape of the bees. The behavior is identified on 30fps videos of resolution 720 x 480 pixels detecting more than 500 bees in every frame.

The in-and-out activity of the beehive is monitored using an imaging system developed by C. Chen et al. [11] use an infrared LED light source, an infrared CCD camera, and a personal computer to collect and process the images. Each bee is attached with circular character-encoding tags at the dorsal part of the bee's thoraxes. The labels have a black dot to recognize the orientation of the encoding and are segmented in the frames using hough transform and SVM. This method avoids the downsides of the RFID tracking that weighed 2.5mg and required a power source produced by electromagnetic waves emitted by the reader, which might affect the bee behavior. This study replaced the tags with a circular tag that did not need any power source and weighed 1.0mg. The character segmentation and recognition were done instead of classifying the motion regions into bees or no-bees. The bees are placed in a freezer to temporarily lose their mobility and remove the fine hairs on the back to apply glue tags. Though the classification accuracy is around 98%, this process seems to be both time-consuming and might affect the behavior of the bees.

To monitor the bee traffic, the research by Ghadiri [12] placed the camera in front and top of the hive and recorded their movements which are transferred to the server directly. The study estimates the bee traffic by using the change detection algorithm on the current image with an average of 11 images that appear before the current image. The change mask generated from the change detection algorithm shows the number of segments represented as the bee counts. Since there can be more than one bee in each segment, the average bee size divides the segment size to estimate the bee motions better.

Ghadiri's [12] study also mentions that the background image generation can be more accurate by capturing images in faster intervals. In change detection, the lower the threshold, the lesser the changes detected. The higher the threshold, the more sensitive the detection becomes, resulting in detecting many changes than required. The value is tuned accordingly before fixing the value that can detect changes as per the requirement.

1.3 Our Contribution

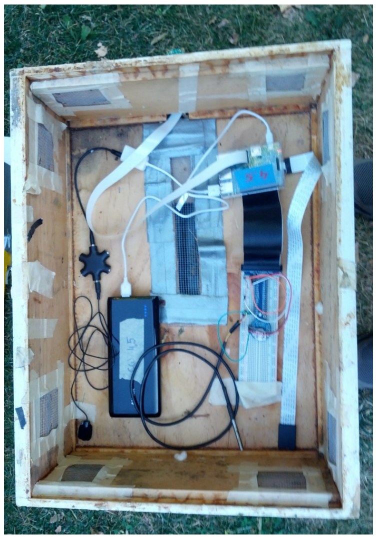
In this thesis, we present a few techniques to generate shallow convolutional neural networks and random reinforced forests that can be deployed onto a raspberry pi 3 and will be able to perform classification efficiently in comparison to the deep neural networks. First, we explain the data collection and processing using the BeePi monitors mounted on the beehive. Second, we introduce new techniques like reinforcement learning and performance-based to generate random forests using decision trees trained on different local receptive fields of the image dataset. We also present a RRF-theorem (Random Reinforced Forest Theorem) to prove that one reinforcement learning algorithm is unlikely to find a best-performing random forest. Third, we describe the shallow convolutional neural networks and how they are generated. Fourth, the number of operations and parameters used to classify a single image in both neural networks and random forests is formulated. Fifth, all the classifiers are compared based on the validation accuracies, number of operations, and parameters. Finally, we conclude about the experiments and present the future work.

CHAPTER 2

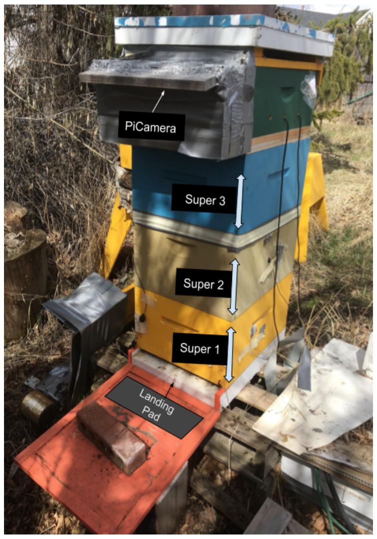
MATERIALS AND METHODS

2.1 Hardware and Data Collection

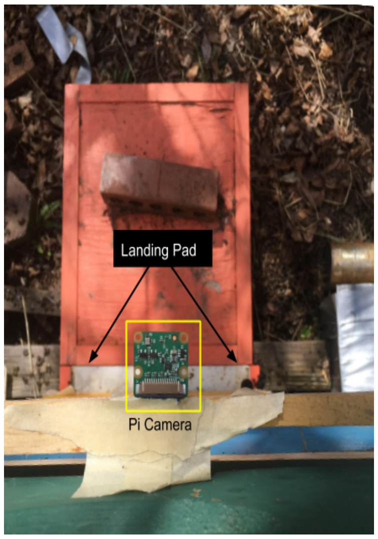
EBM is performed through a BeePi monitor (See Figure 2.1a) which consists of multiple sensors and other hardware components together mounted on the top of the beehive (See Figure 2.1b) [13,14]. This BeePi monitor has a raspberry pi computer connected to a power supply, a miniature camera facing down towards the landing pad of the hive (See Figure 2.1c, 2.1d) for collecting the video samples to monitor the forager traffic, an audio hub connected to microphones (See Figure 2.1e) hanging just above the landing pad for collecting the audio samples, a waterproof temperature sensor for collecting the surrounding temperatures. It is also connected to a hardware clock for embedding the timestamp during data collection, an SD card containing the system software and fused samples of video, audio, temperature, and respective timestamps using sensor fusion breadboard for integrating all the hardware mentioned.



(a) Inside the BeePi and it's hardware

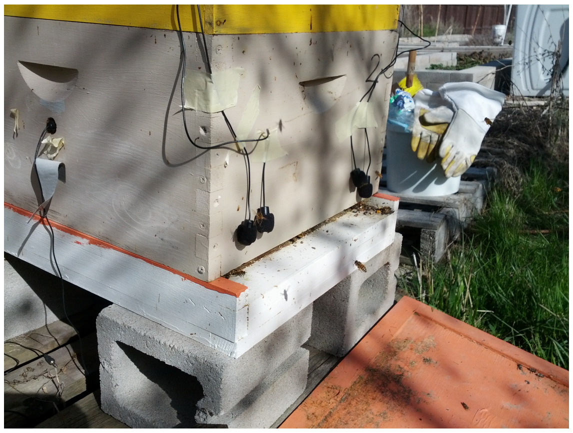


(b) Beehive of 3 supers with BeePi on the top



(c) Miniature camera faced down towards the landing pad

(d) View of the miniature camera from the landing pad



(e) Four microphones hanging near to the beehive entrance

Fig. 2.1: Beehive and BeePi Monitor [1, 2]

Each video sample is run through a motion detection algorithm to capture the motion regions, which are then classified into bee or no-bee through a trained convolutional neural network or machine learning techniques like Random Forests (See Figure 2.2 for an overview of this process).

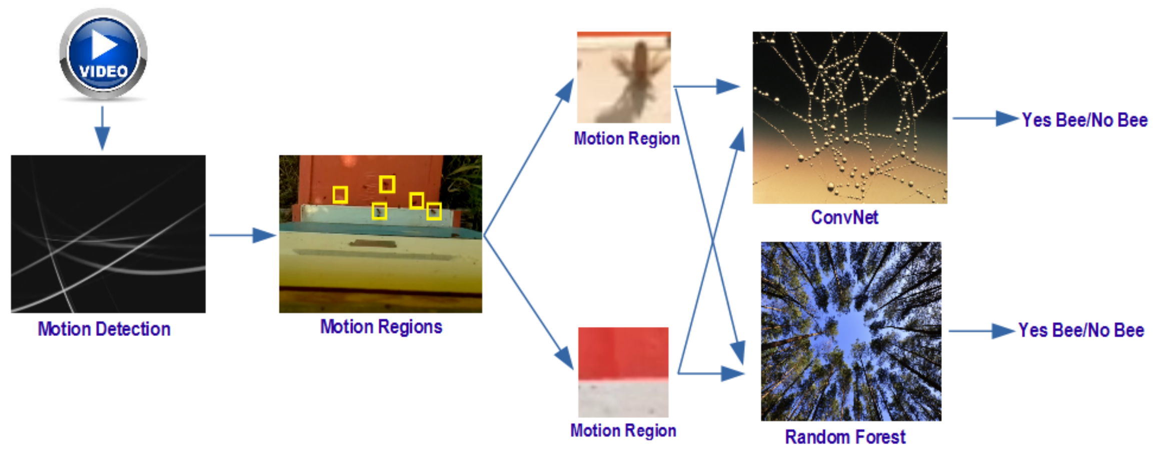


Fig. 2.2: An overview of the video processing and classification process [1]

2.2 Datasets

In this thesis, all the methods have been trained, tested, and validated across five different image datasets. Each dataset contains three divisions of training, testing and validation which are labeled and divided manually. The training and testing divisions are used for model fitting and the validation dataset for model selection ensuring that the training and testing images are from a different hive as of validation images for better generalization of the model. The tables 2.1 - 2.5 show the distribution of data into respective divisions and the figures 2.3 - 2.7 show the image samples of BEE, NO_BEE and SHADOW_BEE (BEE3 Only) for the following datasets.

1. BEE1

   This dataset consists of 54,391 32 × 32 images obtained by selecting 40 videos randomly from approximately 3000 videos dataset collected from the BeePi monitors with a frame rate of ≈ 25fps and a resolution of 360 × 240. The videos are collected from four BeePi monitors deployed in four Langstroth hives with Italian colonies.

	BEE	**NO_BEE**	**Total**
Train	19082	19057	38139
Test	6362	6362	12724
Valid	1810	1718	3528
Total	27254	27137	54391

Table 2.1: BEE1 sample distribution of image samples

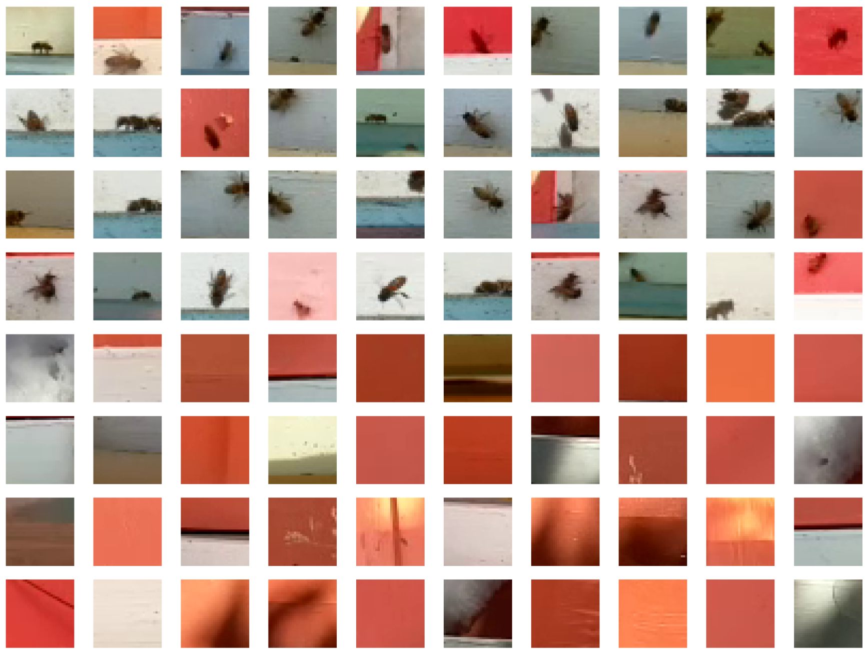


Fig. 2.3: Sample of images from BEE1. The first four rows include images labeled as BEE. The last four rows include images labeled as NO_BEE [2]

2. BEE2

   This second dataset, named BEE2, contains a total of 112,879 images that are extracted from the videos recorded by the BeePi monitors of four Langstroth hives with Carniolan bee colonies. A total of 5509 one-super and 5460 two-super videos of resolution 1920 $\times$ 1080 are used to select 50 videos from each super randomly for the generation of the datasets. One-super videos are collected when the BeePi monitor is placed on top of one deep Langstroth hive and two-super videos from two deep Langstroth hives. The dataset named BEE2_1S is generated from the one-super videos contains 58,201 150 $\times$ 150 images. Similarly, the dataset called BEE2_2S is formed from the two-super videos consists of 54,768 90 $\times$ 90 images.

	BEE	**NO_BEE**	**Total**
Train	8266	27108	35374
Test	2828	9035	11863
Valid	8298	2666	10964
Total	19392	38809	58201

Table 2.2: BEE2_1S sample distribution of image samples

	BEE	**NO_BEE**	**Total**
Train	12982	15983	28965
Test	4194	5327	9521
Valid	6823	9369	16192
Total	23999	30679	54678

Table 2.3: BEE2_2S sample distribution of image samples

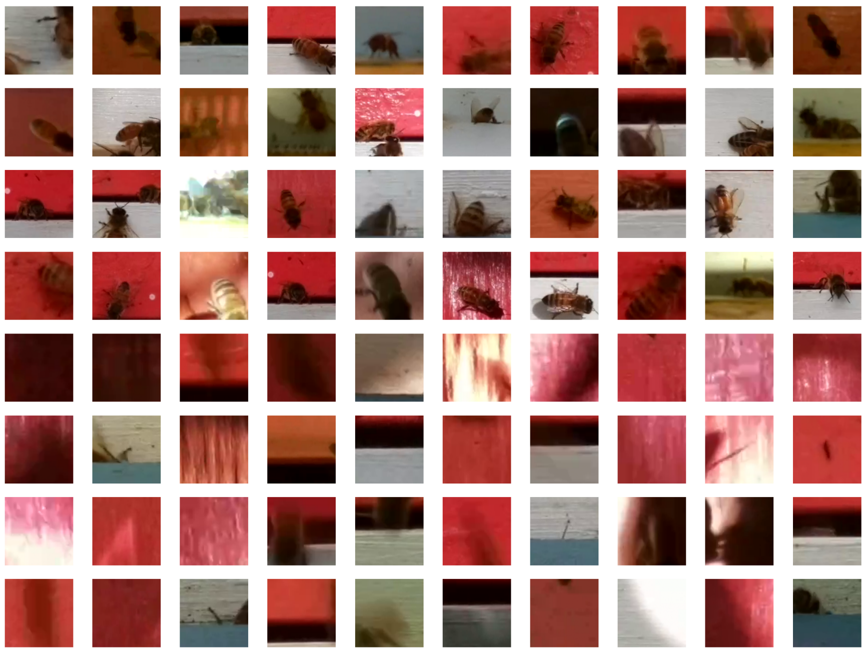


Fig. 2.4: Sample of images from BEE2_1S. The first four rows include images labeled as BEE. The last four rows include images labeled as NO_BEE [2]

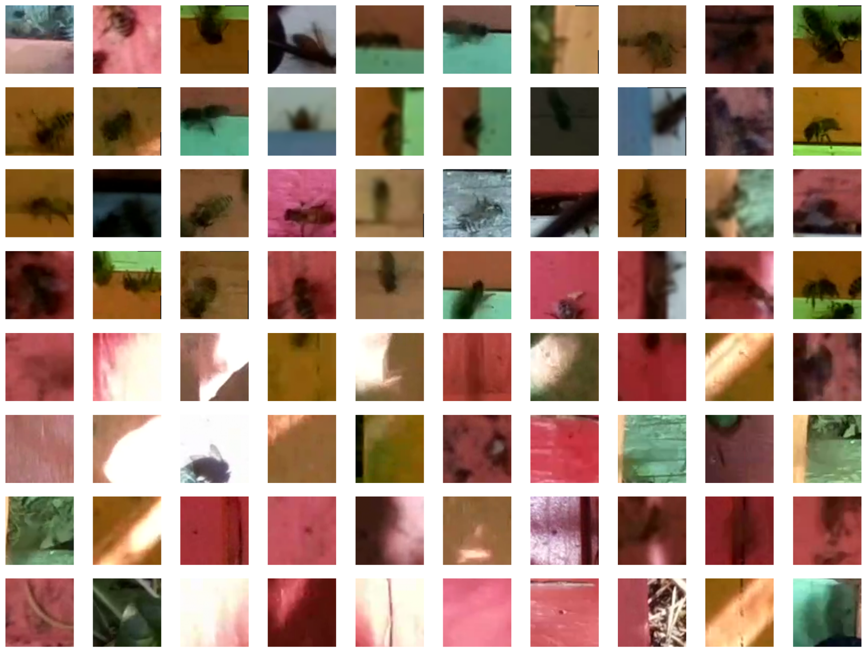


Fig. 2.5: Sample of images from BEE2_2S. The first four rows include images labeled as BEE. The last four rows include images labeled as NO_BEE [2]

3. BEE3

   It is observed that the shadow_bees are classified as bees while estimating the forager traffic of the beehive. Therefore, a new class of shadow_bee is introduced to avoid these counts as well. This dataset contains 65,023 64 $\times$ 64 images collected from videos recorded during the noon when the majority of shadows can occur. The images are then manually labeled into BEE, NO_BEE, SHADOW_BEE.

	BEE	**NO_BEE**	**SHADOW_BEE**	**Total**
Train	12948	12857	12006	37811
Test	4236	4242	3993	12471
Valid	5187	5204	4350	14741
Total	22371	22303	20349	65023

Table 2.4: BEE3 sample distribution of image samples

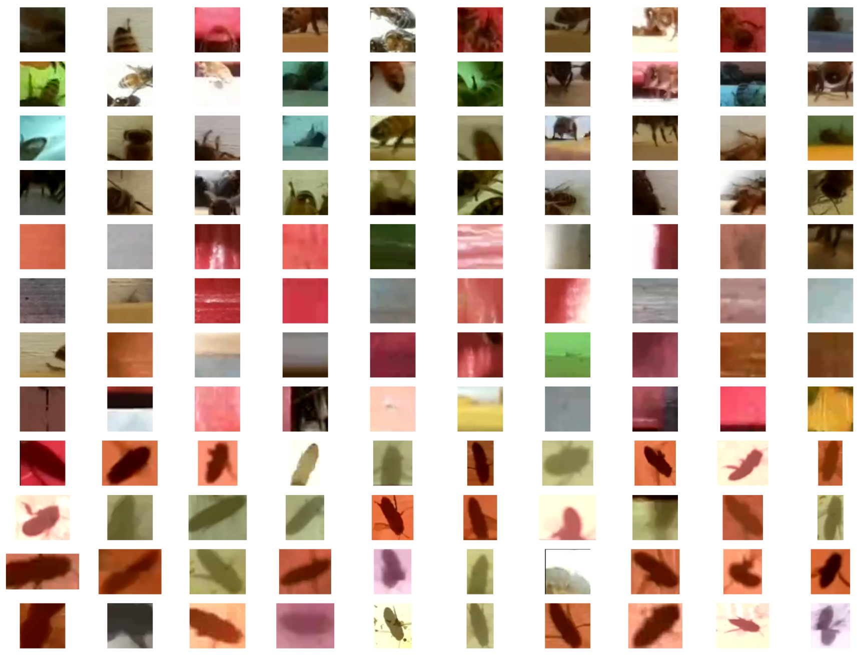


Fig. 2.6: Sample of images from BEE3. The first four rows include images labeled as BEE. The second four rows include images labeled as NO_BEE. The last four rows include images labeled as SHADOW-BEE [3]

4. BEE4

   A total of 996 images in the BEE2_2S dataset have been mislabeled and needed to be moved into their respective classes. Therefore, we curated, verified the images that are being transferred, and renamed the dataset to BEE4.

	BEE	**NO_BEE**	**Total**
Train	12696	16269	28965
Test	4142	5379	9521
Valid	7243	8949	16192
Total	24081	30597	54678

Table 2.5: BEE4 sample distribution of image samples

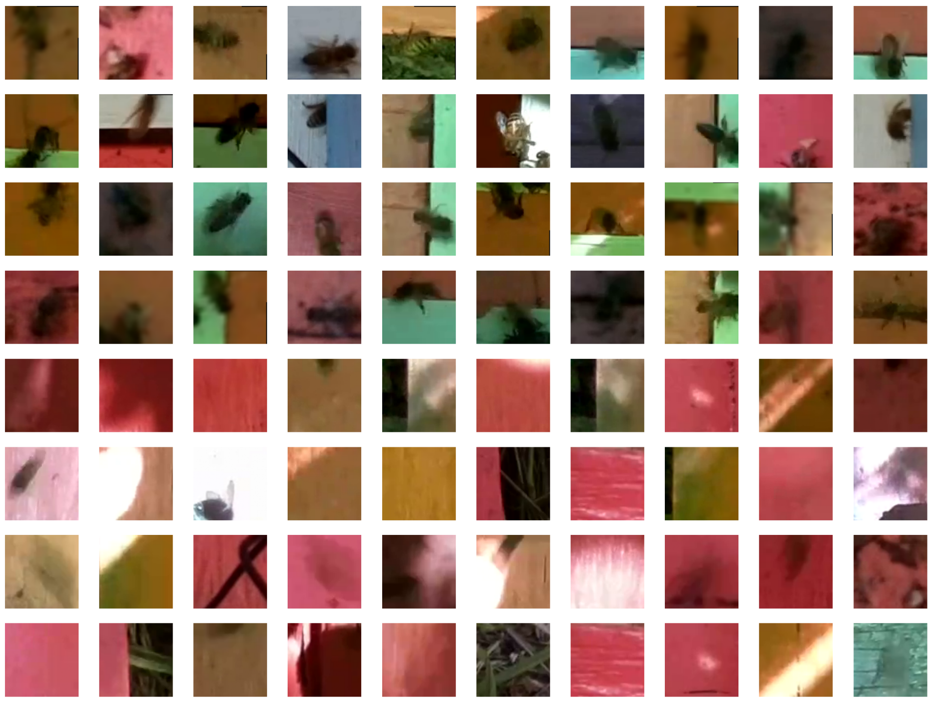


Fig. 2.7: Sample of images from BEE4. The first four rows include images labeled as BEE. The last four rows include images labeled as NO_BEE

In addition to these generated datasets, the comparison is also performed on a well-known challenging dataset CIFAR-10 [15], a publicly available dataset that consists of 60,000 color images of size 32×32 with ten different classes. The set named testing of size 10000 (100 per class) in this thesis is called validation, which is used for model selection at the end. The training set is randomly divided into 40000 training images (400 per class) and 10000 testing images (100 per class) used for model fitting.

CHAPTER 3
RANDOM REINFORCED FOREST

3.1 Local Receptive Field Overlap

All the images of the dataset are 3-channel RGB images which are converted to grayscale and resized to be 64 x 64. Further, these images are processed through a set of pre-defined local receptive fields (LRFs) (See Figure 3.1) and then used for training the decision trees. This is done by creating all the shapes of the LRFs and then overlapping the images onto these LRFs and accessing the pixels that have been matched in the position.

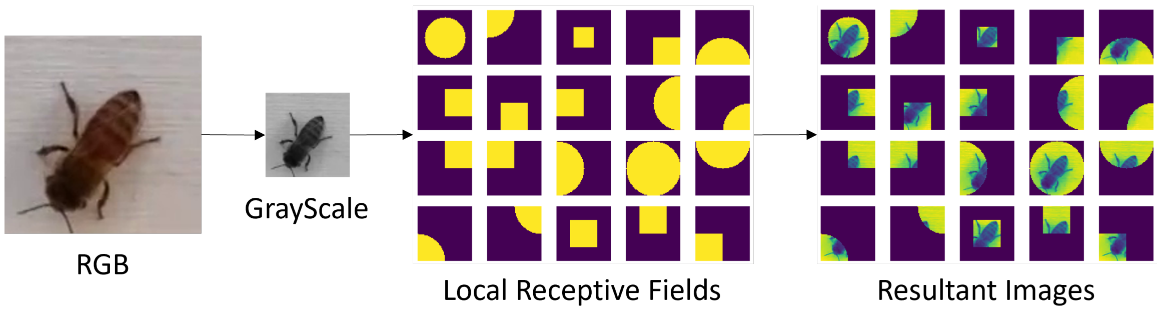


Fig. 3.1: LRF Overlap for Decision Trees

3.2 Decision Trees

A decision tree is a tree-based machine learning technique that is used for both classification and regression modeling. Each sample in the data consists of attribute-value pairs (pixel position and pixel value in the case of an image), which are used for splitting the dataset into subgroups based on either **Information Gain** (based on Entropy which measures the randomness), which reduces the uncertainty [16] or **Gini Index** (which measures the impurity) which minimizes the impurity [17]. The more diverse the dataset, the more its randomness or impurity, and vice versa.

The entropy (H) of the dataset (S) with respect to an attribute A with C set of possible values of $\{v_1, v_2, v_3, \ldots, v_C\}$ is defined as:

$$H(S, A) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

where p_i is the proportion of the samples for which $A = v_i$

The information gain based on entropy allows us to measure the increase in certainty or reduction in the entropy. The information gain for the dataset (S) with respect to an attribute A with C set of possible values of $v_1, v_2, v_3, \ldots, v_C$ and target attribute T is defined as:

$$Gain(S, A) = H(S, T) - \sum_{v \in Values(A)} \frac{|S_{A=v}|}{|S|} H(S_{A=v}, T)$$

where $|S_{A=v}|$ is the cardinality (number of elements) of the set of examples for which $A = v$.

Similarly, the gini index computes the probability of a randomly chosen variable being classified wrongly. The gini of the dataset (S) with respect to an attribute A with C set of possible values of $\{v_1, v_2, v_3, \ldots, v_C\}$ is defined as:

$$Gini(S, A) = \sum_{i=1}^{C} p_i(1 - p_i) = 1 - \sum_{i=1}^{C} (p_i)^2$$

where p_i is the proportion of the samples for which $A = v_i$

The decision tree grows by recursively splitting the subgroups until the leaf nodes become pure (subgroups contain data from the same class) or the tree reaches the maximum depth, which is one of the input parameters. After the generation of the decision tree, each leaf node now contains a sub-group of samples with class probabilities. When a new sample (out-of-bag) is fed into the trained decision tree for the classification, it traverses from the root node to the leaf node. The class with the highest probability in the respective leaf node represents the class of the new sample. See Figure 3.2 for a sample decision tree with a max-depth of 2.

At each node in the decision tree in figure 3.2, the splitting of the samples happens by choosing the pixel position with the lowest Gini index (e.g., at root node #0, split happens based on pixel value at position 1009 compared to 149.5). The *gini* mentioned is based on the target attribute, which is calculated after each split. Thus, the lower the value of the gini, the purer the nodes are. The *value* represents the list of probabilities of each class of samples present at the particular node. Finally, the *class* is assigned based on the highest chance of all the categories.

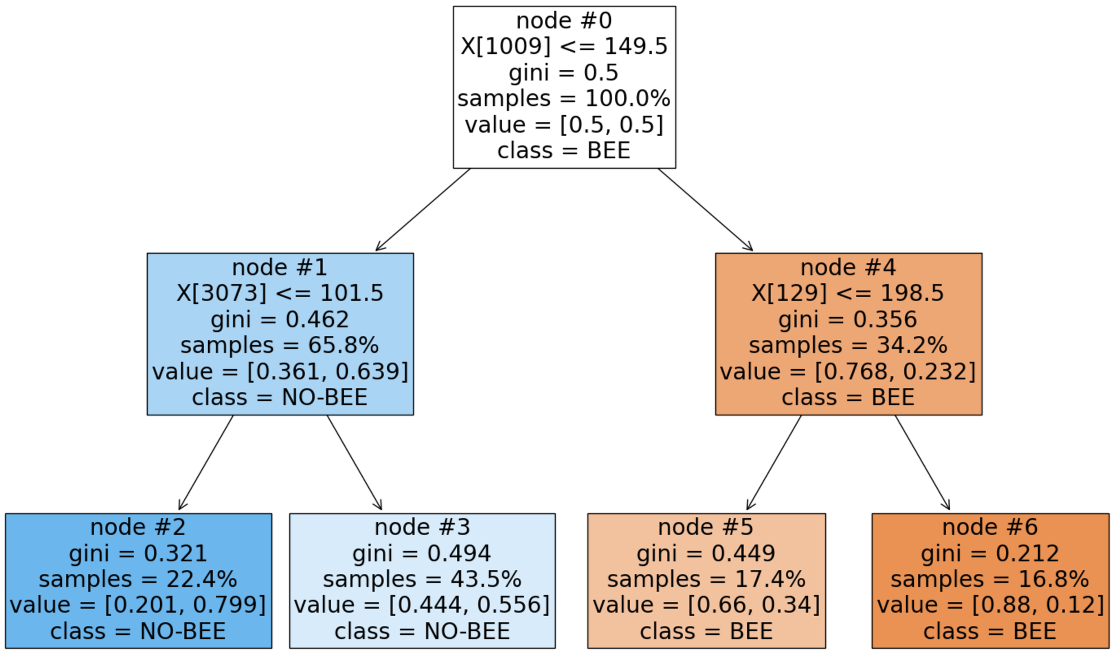


Fig. 3.2: Sample Decision Tree

3.3 Random Forests

Bootstrap aggregation is a process of randomly picking **n** samples with replacement from the whole dataset. These **n** samples are used to train one decision tree. An ensemble of multiple decision trees through bootstrap aggregation (also known as bagging) of samples is known as a random forest [18]. For a single decision tree, the samples that are used for the training are called in-bag samples and the ones that are not used are called out-of-bag samples. Through this procedure, many different decision trees are generated and then used for decision-making based on the mean of the probabilities provided by each decision tree.

3.4 Random Forest Generation

To mimic the working of the random forests as explained in the section 3.3, the set of bootstrap samples $\{B_1, B_2, \ldots, B_{200}\}$ of the dataset are fed to the decision trees instead of the complete data. With a set of 10 pre-defined maximum depths $\{D_1, D_2, \ldots, D_{10}\}$ and 20 local receptive fields $\{S_1, S_2, S_3, \ldots, S_{19}, S_{20}\}$ as mentioned in section 3.1, a total of 200 different decision trees $\{T_{S_1,D_1}, T_{S_1,D_2}, T_{S_1,D_3}, \ldots, T_{S_{20},D_9}, T_{S_{20},D_{10}}\}$ (see section 3.2 for understanding the decision tree working) have been trained on the bootstrap samples of the processed dataset which can now be combined as random forests. The total number of various possibilities of random forests that can exist from the 200 decision trees is $2^{200} - 1$ (without empty random forest). The fig 3.3 shows the overview of this process of generating random forests.

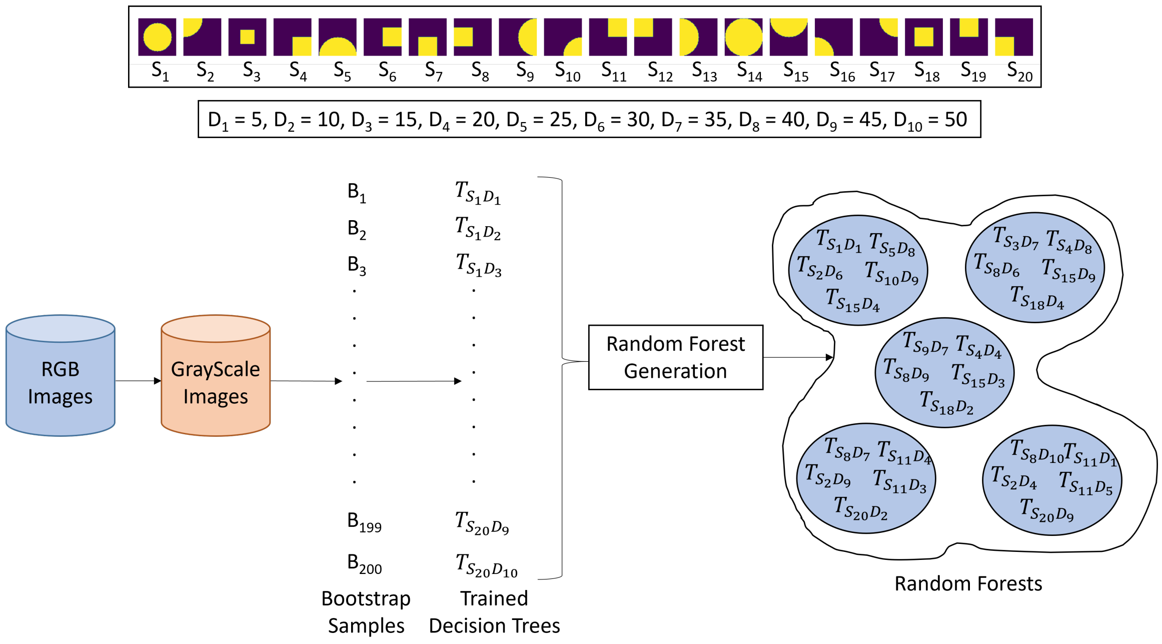

S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10} S_{11} S_{12} S_{13} S_{14} S_{15} S_{16} S_{17} S_{18} S_{19} S_{20}
$D_1 = 5, D_2 = 10, D_3 = 15, D_4 = 20, D_5 = 25, D_6 = 30, D_7 = 35, D_8 = 40, D_9 = 45, D_{10} = 50$
RGB Images
GrayScale Images
B_1 B_2 B_3 B_{199} B_{200}
Bootstrap Samples
$T_{S_1D_1}$ $T_{S_1D_2}$ $T_{S_1D_3}$ $T_{S_{20}D_9}$ $T_{S_{20}D_{10}}$
Trained Decision Trees
Random Forest Generation
$T_{S_1D_1}$ $T_{S_5D_8}$ $T_{S_2D_6}$ $T_{S_{10}D_9}$ $T_{S_{15}D_4}$
$T_{S_3D_7}$ $T_{S_4D_8}$ $T_{S_8D_6}$ $T_{S_{15}D_9}$ $T_{S_{18}D_4}$
$T_{S_9D_7}$ $T_{S_4D_4}$ $T_{S_8D_9}$ $T_{S_{15}D_3}$ $T_{S_{18}D_2}$
$T_{S_8D_7}$ $T_{S_{11}D_4}$ $T_{S_2D_9}$ $T_{S_{11}D_3}$ $T_{S_{20}D_2}$
$T_{S_8D_{10}}$ $T_{S_{11}D_1}$ $T_{S_2D_4}$ $T_{S_{11}D_5}$ $T_{S_{20}D_9}$
Random Forests

Fig. 3.3: Process of training decision trees and generating random forests using reinforcement learning and performance-based technique

3.5 Reinforcement Learning

Reinforcement learning is a process where in an environment, the agent is given a set of actions to pick which can result in either a positive reward or a negative reward. The main motive of the agent is to find the actions that can result in the maximization of the reward [19].

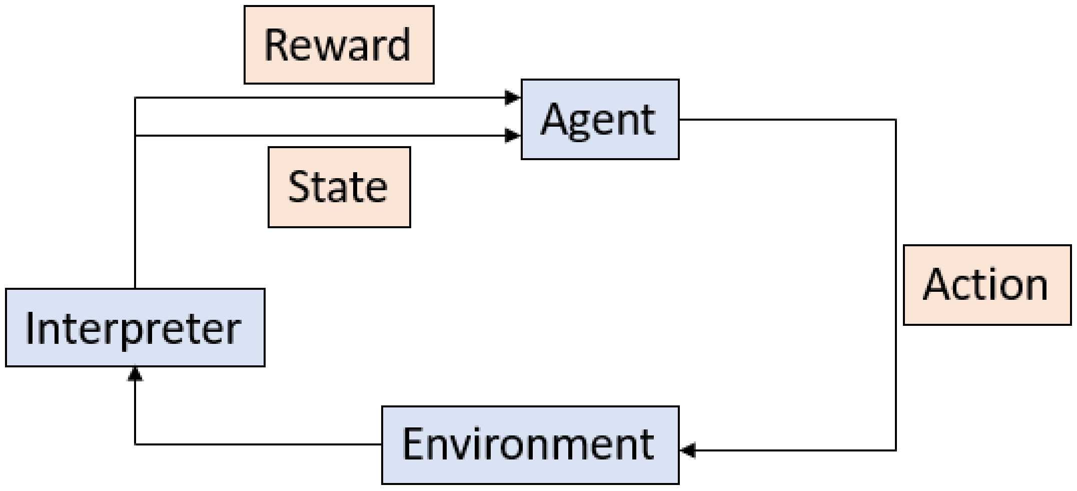


Fig. 3.4: Reinforcement Learning

3.5.1 Q-Learning

Q-Learning is one of the basic reinforcement learning algorithms that is used for many applications [19]. This algorithm starts with the initialization of Q-Matrix/Q-Table with zeros of size number of states $\times$ number of actions. At every step of the learning, the agent picks the action in two different ways:

1. Exploitation: Given a state (s), an action (a) is picked which has the maximum q-value
2. Exploration: Given a state (s), an action (a) is picked randomly from the available actions

The exploration and exploitation is controlled by the epsilon value. The higher the epsilon value, the higher the exploration and lower the exploitation and vice versa. The algorithm should explore completely during the start to completely exploit towards the end of the episodes. Therefore, it's a good practice to decay the epsilon from 1 to 0 using an epsilon decay.

Based on the action, the assignment of reward (r) and generation of the next state (s') and used for updating the q-value for the respective state and action. The update equation [19] is:

$$Q(s,a) = (1-\alpha) \times Q(s,a) + \alpha \times (r + \gamma \times max(Q(s',a'))) \tag{3.1}$$

where α is the learning rate and γ is the discount factor

3.6 Random Reinforced Forests

A Random Reinforced Forest (RRF) is a random forest generated using reinforcement learning (Q-Learning) where the agent is the python script performing an action of adding a tree to the random forest environment and the interpreter calculates the new state and the reward which is fed back to the agent. There are two different types of RRFs that are discussed which are *Hidden Q-Learning* and *Dynamic Q-Learning*.

3.7 Hidden Q-Learning (HQL)

This method follows the Q-learning algorithm (refer section 3.5.1) by maintaining a hidden state (random forest) for each episode of learning on which the action is taken. The below are the specifications of the algorithm.

1. *Action*: Add a decision tree to the random forest (200 actions)

2. *State*: Represents the last addition of a decision tree in the random forest (200 states)

3. *Initial State*: A random tree is selected to be in the random forest from the available 200 decision trees (refer figure 3.3) which represent the last addition.

4. $Reward = valid\_acc_{current} - valid\_acc_{previous}$

where $valid\_acc_{current}$ indicates the validation accuracy of the random forest after performing the action (addition of a tree) and $valid\_acc_{previous}$ indicates the validation accuracy before performing the action (addition of a tree).

5. Learning takes place for 1000 *episodes* with 100 steps each

Below is an example for the HQL method on four trees. Each episode of learning starts with an empty random forest $RF = []$, with the parameters, $\alpha = 0.9$, $\gamma = 0.1$. The initial Q-Matrix will be a zeros matrix of shape 4×4 representing the 4 actions and 4 states respectively. Assuming the initial state for the episode is *last_added_1* which represents the tree that has been last added to the random forest.

Step 0: Initial State: last_added_1, Hidden State RF = [1], $valid\_acc_{current} = 68$

	add_1	add_2	add_3	add_4
last_added_1	0	0	0	0
last_added_2	0	0	0	0
last_added_3	0	0	0	0
last_added_4	0	0	0	0

Step 1: Action Taken: add_2, Hidden State RF = [1, 2], $valid\_acc_{current} = 70$, $Reward = 70 - 68 = 2$, New State: last_added_2

$$Q(1,2) = (1 - 0.9) \times 0 + 0.9 \times (2 + 0.1 \times 0) = 1.8$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
last_added_1	0	1.8	0	0
last_added_2	0	0	0	0
last_added_3	0	0	0	0
last_added_4	0	0	0	0

Step 2: Action Taken: add_3, Hidden State RF = [1, 2, 3], $valid\_acc_{current} = 72$, $Reward = 72 - 70 = 2$, New State: last_added_3

$$Q(2, 3) = (1 - 0.9) \times 0 + 0.9 \times (2 + 0.1 \times 0) = 1.8$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
last_added_1	0	1.8	0	0
last_added_2	0	0	1.8	0
last_added_3	0	0	0	0
last_added_4	0	0	0	0

Step 3: Action Taken: add_1, Hidden State RF = [1, 2, 3, 1], $valid\_acc_{current} = 71$, $Reward = 71 - 72 = -1$, New State: last_added_1

$$Q(3, 1) = (1 - 0.9) \times 0 + 0.9 \times (-1 + 0.1 \times 0) = -0.9$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
last_added_1	0	1.8	0	0
last_added_2	0	0	1.8	0
last_added_3	-0.9	0	0	0
last_added_4	0	0	0	0

Step 4: Action Taken: add_2, Hidden State RF = [1, 2, 3, 1, 2], $valid\_acc_{current} = 72$, $Reward = 72 - 71 = 1$, New State: last_added_2

$$Q(1,2) = (1 - 0.9) \times 1.8 + 0.9 \times (1 + 0.1 \times 1.8) = 1.242$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
last_added_1	0	1.242	0	0
last_added_2	0	0	1.8	0
last_added_3	0.9	0	0	0
last_added_4	0	0	0	0

3.8 Dynamic Q-Learning (DQL)

This method also follows the Q-learning algorithm (refer section 3.5.1) but the learning starts with 0 states. The new state calculated by the interpreter is added to the current list of states if it is not present.

1. *Action*: Add a decision tree to the random forest (200 actions)

2. *State*: A vector of length 200 representing the presence of all the trees in the forest (e.g., $[1, 0, 1, 0, 0, \ldots, 0, 1]$)

3. *Initial State*: Random forest with a single tree that is randomly selected from the available 200 decision trees (refer figure 3.3). Therefore, the initial state would be binary vector of length 200 with only a single one and the rest will be zeros ($[1, 0, 0, 0, 0, \ldots, 0, 0, 0]$)

4. DQL1 $Reward = valid\_acc_{current} - valid\_acc_{previous}$

5. DQL2 $Reward = valid\_acc_{current} - valid\_acc_{threshold}$

   where $valid\_acc_{current}$ indicates the validation accuracy of the random forest after the action (addition of a tree), $valid\_acc_{previous}$ indicates the validation accuracy before the action (addition of a tree), $valid\_acc_{threshold}$ indicates the accuracy value that is tough for every random forest to achieve. The threshold accuracies are 85% for BEE1, 75% for BEE2_1S, BEE2_2S, BEE4, 85% for BEE3 (No Shadow), 80% for BEE3 (with shadow) and 40% for CIFAR-10.

6. Learning takes place for 1000 *episodes* with 100 steps each

In the worst case, the total number of states would be 10^5 but there would be no reinforcement learning in this case. But to prove that there is re-visitation of states happening in this method, the below statistics in table 3.1 have been calculated based on 4 experiments each done on 7 different datasets with thereby repeated on 2 reward structures resulting in 56 experiments. Also, See figure 3.5 for the distribution of the episodes in the DQL1, DQL2 experiments and also the distribution of the total experiments. Therefore, the minimum probability of the re-visitation of the episodes that are already generated is $\approx 21\%$.

	Minimum	**Maximum**	**Mean**	**Number of Experiments**
DQL1	78352	78779	78630	28
DQL2	78480	78834	78633	28
Total	78352	78834	78632	56

Table 3.1: Statistics of Number of Episodes Generated using DQL

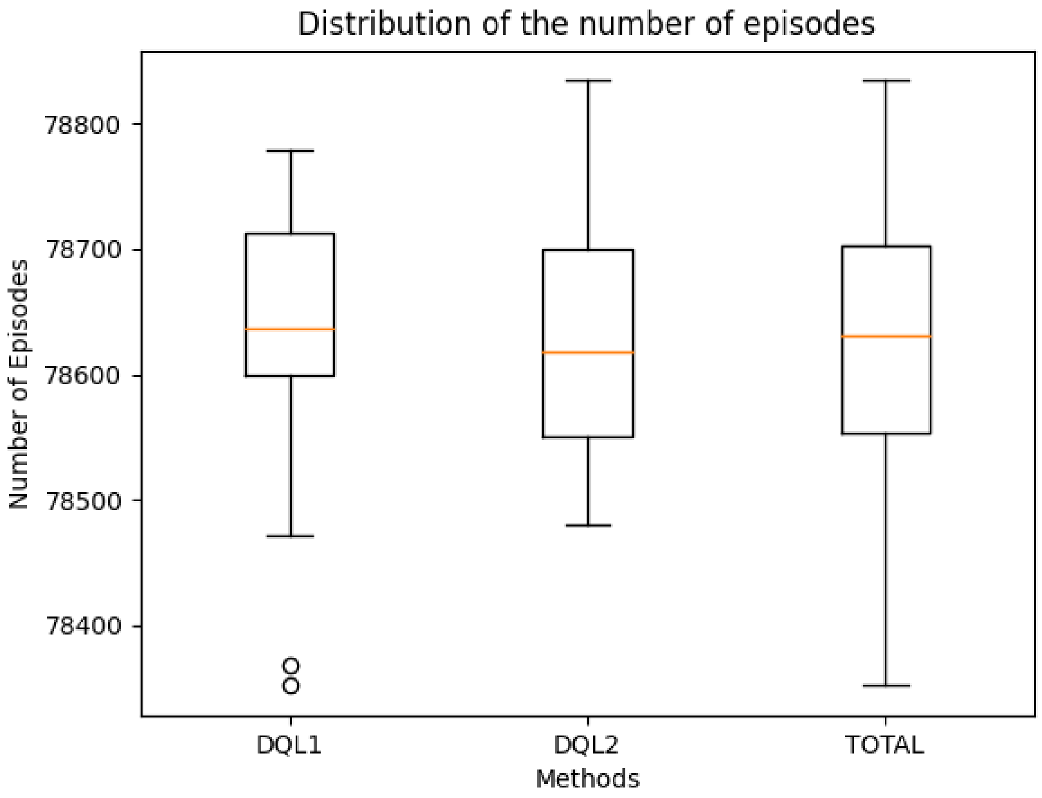


Fig. 3.5: Boxplot representing the distribution of the number of episodes in each method

Below is an example for the DQL method on four trees. Each episode of learning starts with an empty random forest $RF = []$, with the parameters, $\alpha = 0.9$, $\gamma = 0.1$. The initial Q-Matrix will be with four actions of adding a tree and zero states. Each upcoming state represents the presence and absence of all the trees using binary notation. Assuming the initial state for the episode learning containing 4 steps is $[1, 0, 0, 0]$ which represents the presence of the first tree and absence of other 3 trees.

Episode 1

Step 0: Initial State: [1, 0, 0, 0], RF = [1], $valid\_acc_{current} = 68$

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	0	0	0

Step 1: Action Taken: add_2, RF = [1, 2], $valid\_acc_{current} = 70$, $Reward = 70 - 68 = 2$, New State: [1, 1, 0, 0]

$$Q(1,2) = (1 - 0.9) \times 0 + 0.9 \times (2 + 0.1 \times 0) = 1.8$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	0	0

Step 2: Action Taken: add_3, RF = [1, 2, 3], $valid\_acc_{current} = 72$, $Reward = 72 - 70 = 2$, New State: [1, 1, 1, 0]

$$Q(2,3) = (1 - 0.9) \times 0 + 0.9 \times (2 + 0.1 \times 0) = 1.8$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	1.8	0
[1, 1, 1, 0]	0	0	0	0

Step 3: Action Taken: add_1, RF = [1, 2, 3, 1], $valid\_acc_{current} = 71$, $Reward = 71 - 72 = -1$, New State: [1, 1, 1, 0]

$$Q(3,1) = (1 - 0.9) \times 0 + 0.9 \times (-1 + 0.1 \times 0) = -0.9$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	1.8	0
[1, 1, 1, 0]	-0.9	0	0	0

Step 4: Action Taken: add_4, RF = [1, 2, 3, 1, 4], $valid\_acc_{current} = 72$, $Reward = 72 - 71 = 1$, New State: [1, 1, 1, 1]

$$Q(3, 4) = (1 - 0.9) \times 0 + 0.9 \times (1 + 0.1 \times 0) = 0.9$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	1.8	0
[1, 1, 1, 0]	-0.9	0	0	0.9
[1, 1, 1, 1]	0	0	0	0

Episode 2: This episode learning is run for only one step to show the learning transfer.

Step 0: Initial State: [0, 1, 0, 0], RF = [2], $valid\_acc_{current} = 69$

The updated Q-matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	1.8	0
[1, 1, 1, 0]	-0.9	0	0	0.9
[1, 1, 1, 1]	0	0	0	0
[0, 1, 0, 0]	0	0	0	0

Step 1: Action Taken: add_1, RF = [2, 1], $valid\_acc_{current} = 70$, $Reward = 70 - 69 = 1$, New State: [1, 1, 0, 0]

$$Q(5,1) = (1 - 0.9) \times 0 + 0.9 \times (1 + 0.1 \times 1.8) = 1.062$$

The updated Q-value matrix is:

	add_1	add_2	add_3	add_4
[1, 0, 0, 0]	0	1.8	0	0
[1, 1, 0, 0]	0	0	1.8	0
[1, 1, 1, 0]	-0.9	0	0	0.9
[1, 1, 1, 1]	0	0	0	0
[0, 1, 0, 0]	1.062	0	0	0

3.9 RRF-Theorem

Let $\{T_1, T_2, T_3, \ldots, T_k\}$ be the set of trained decision trees using k bootstrap samples generated from a dataset. Therefore, there exists $2^k - 1$ possible random forests. Each random forest is considered to be a possible explored *State* in the DQL.

The maximum number of possible states is:

$$M = 2^k - 1$$

A random forest is said to be a *Best State* random forest if and only if its performance (validation accuracy) is greater than a pre-defined value of θ.

Theorem: Let M be a total number of possible random forests, of which B are best state. If $B \lll M$ (considerably small), then the probability of not finding any best state random forest with DQL is 1.

Proof: The maximum number of possible states explored in the DQL technique is

$$N = n \cdot \epsilon \cdot s$$

where ϵ epsilon or the percentage of exploration, n is number of episodes, s is number of steps per episode of learning
The probability of not exploring a best state for the first trial is:

$$P_{1,B} = \left(1 - \frac{B}{M}\right)$$

Similarly, the probability of not exploring a best state for two trials is:

$$P_{2,B} = \left(1 - \frac{B}{M}\right)\left(1 - \frac{B}{M-1}\right)$$

Extending this behavior to the N trials, we obtain:

$$P_{N,B} = \left(1 - \frac{B}{M}\right)\left(1 - \frac{B}{M-1}\right)\dots\left(1 - \frac{B}{M-N+1}\right)$$

$$= \left(\frac{M-B}{M}\right)\left(\frac{M-B-1}{M-1}\right)\dots\left(\frac{M-B-N+1}{M-N+1}\right)$$

$$= \left(\frac{(M-B)!}{(M-B-N)!}\right) \Big/ \left(\frac{M!}{(M-N)!}\right) = \frac{\binom{M-B}{N}}{\binom{M}{N}}$$

Therefore, the probability of not exploring any best state for N trials is:

$$\boxed{P_{N,B} = \frac{\binom{M-B}{N}}{\binom{M}{N}}}$$

If we assume that there is at least one best state for a set of decision trees, the probability value $P_{N,1}$ will be:

$$P_{N,1} = \frac{\binom{M-1}{N}}{\binom{M}{N}} = \frac{M-N}{M} = \left(1 - \frac{N}{M}\right)$$

For $k = 200$, $M = 2^{200} - 1$ and with $N = 10^5$, we get $P_{N,1} = 0.\overline{9} \approx 1$

Since B is a fraction of M, let the fraction be represented as α

$$B = \alpha M \implies \alpha = \frac{B}{M}$$

As the number of best states decreases, the probability term will tend to become 1

$$\boxed{\lim_{\alpha \to 0} P_{N,B} = \lim_{\alpha \to 0} \frac{\binom{M-\alpha M}{N}}{\binom{M}{N}} = 1}$$

CHAPTER 4
SHALLOW NEURAL NETWORKS

4.1 Multi-Layer Perceptron

MLP is a type of neural network with three layers: the input layer, hidden layer, and output layer. In a classification problem, the size input layer matches the flattened 1-dimension size of the input of any shape. The output layer would be the number of classes into which the input data needs to be classified. The hidden layers contain an arbitrary number of nodes that connect the input and the output layers through the edges (weights). The set of edges connect each pair of nodes between two layers (input-hidden, hidden-hidden, hidden-output), representing the weights multiplied by the inputs [20]. A bias adds to the product, which illustrates how easily we can activate the output. More the bias, the higher the activation of the weighted output (sum of bias and the product of weights and inputs). Once data is fed forward through the MLP, the weights and biases are then tweaked by back-propagating the loss calculated by comparing the output and the ground truth using various functions like mean squared error, cross-entropy. The main motive of the backpropagation is to minimize the loss/cost and get the classifier to learn the input data and get closer to the ground truths as much as possible [21]. See Figure 4.1 for the visual representation of multi-layer perceptrons [22]. MLPs are good for numerical data to perform both classification and regression modeling. These neural networks do not consider the spatial orientation of the values in the data. For example, in the images, the position of the pixel values can add information for the classification.

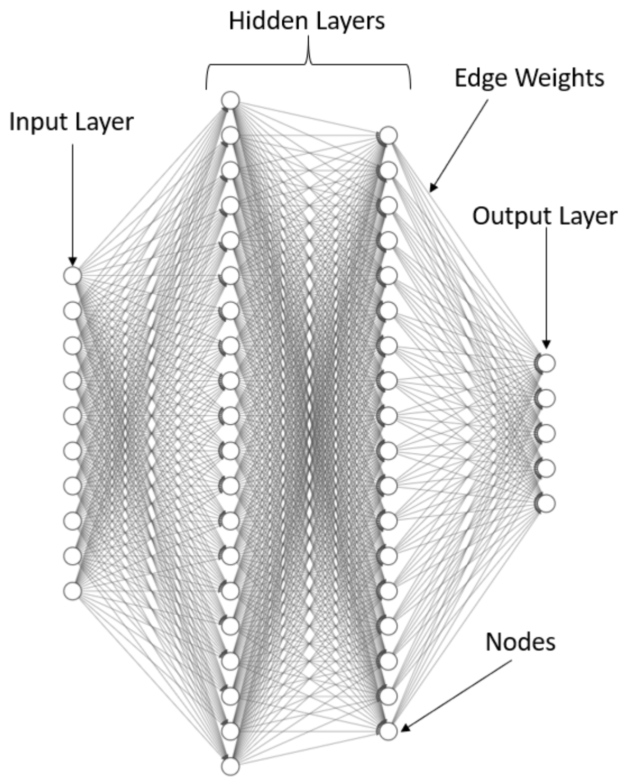


Fig. 4.1: Multi-Layer Perceptron

4.2 Convolutional Neural Networks

When compared to MLPs, Convolutional neural networks perform well on images or data where the value positions matter (e.g., Time series classification) as they consider the spatial orientation of the data [23]. In addition to the layers mentioned in the MLP, CNNs use convolutional layers and pooling layers.

Convolutional Layer

Most of the behavior is the same as mentioned in MLP. Here the input data is not flattened, and its shape is as it should be. The input image gets converted to a matrix of pixel values which consists of 3 dimensions (height, width, and depth). This matrix is connected to the hidden layer using shared weights, also known as filters of a fixed size with a depth dimension like the input instead of different weights for every edge as in MLP [23].

It also has a shared bias instead of a different bias for each neuron in the hidden layer. Each node in the hidden layer connects to a region of the image through the shared weights and performs a convolution operation. This region of the image is called the local receptive field [21]. The exact weights connect all the local receptive fields and the neurons in the hidden layer. If there are n filters used for the convolution, the output will contain n feature maps. Each feature map is the output of each filter performing convolution operation on the local receptive fields. See Figures 4.2, 4.3 for visualizing the convolution operations happening between the input layer and the hidden layer using a formula of convolution as mentioned in the equation 4.1.

The convolution output [21] at the position (i, j) of feature map is:

$$C_{i,j} = \sum_{l=0}^{f_x-1} \sum_{m=0}^{f_y-1} w_{l,m} p_{i+l,j+m} \tag{4.1}$$

where the size of the filter is $f_x \times f_y$, l and m represent the positions of filter, $i + l$ and $j + m$ for the image, w is the shared weights and p is image as pixel matrix.

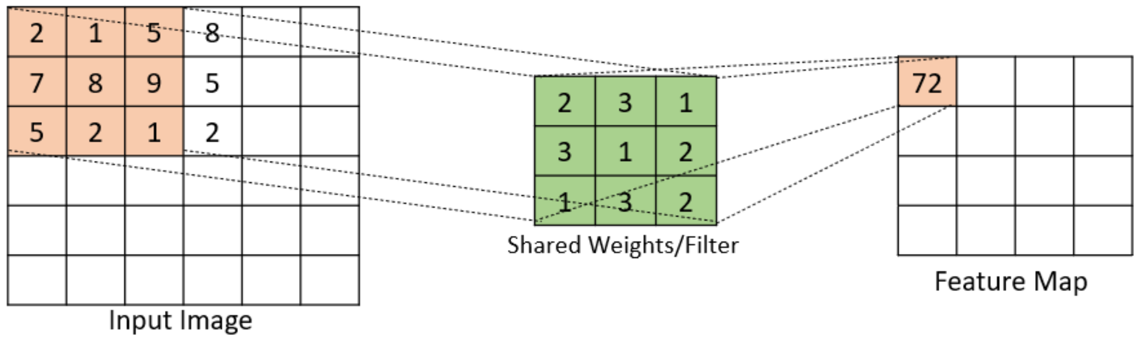


Fig. 4.2: Convolution operation for the first cell of feature map

$$C_{0,0} = \sum_{l=0}^{2} \sum_{m=0}^{2} w_{l,m} p_{0+l,0+m}$$

$$= 2 \times 2 + 1 \times 3 + 5 \times 1 + 7 \times 3 + 8 \times 1 + 9 \times 2 + 5 \times 1 + 2 \times 3 + 1 \times 2 = 72$$

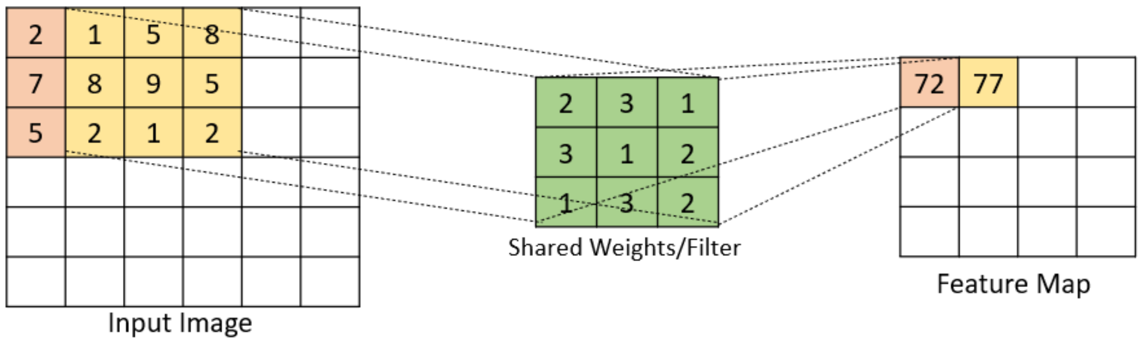


Fig. 4.3: Convolution operation for the second cell of feature map

$$C_{0,1} = \sum_{l=0}^{2} \sum_{m=0}^{2} w_{l,m} p_{0+l,1+m}$$

$$= 1 \times 2 + 5 \times 3 + 8 \times 1 + 8 \times 3 + 9 \times 1 + 5 \times 2 + 2 \times 1 + 1 \times 3 + 2 \times 2 = 77$$

Pooling Layer

The pooling layer is usually present right after the convolutional layer to reduce the dimensionality of the feature maps. Pooling converts the output by applying statistical functions to a set of values in a region from the feature map and obtaining a single value [24]. Some examples of the functions are maximum, L2 (square root of the sum of squares), average, global maximum, global average. See Figure 4.4 for visualizing some examples of the conversion of the output from convolutional layer (feature maps) to low-dimensional feature maps.

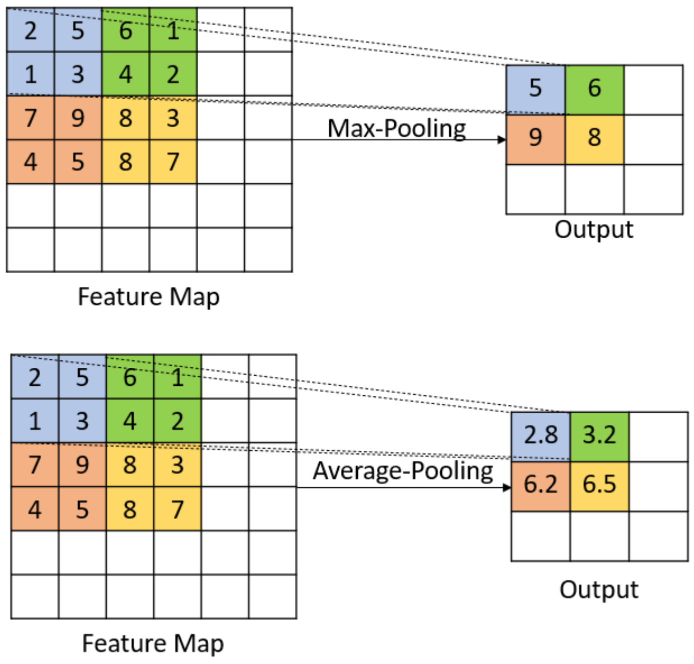


Fig. 4.4: Types of Pooling

Dropout

Neural networks use many techniques to avoid overfitting the training data. Dropout is one technique during which the feedforward and the backpropagation passes happen by dropping a percentage of nodes in the layer mentioned. We repeat this process by restoring and dropping another set of nodes from the same layer. This technique makes the neurons learn the features robustly by avoiding the dependency on the other neurons [21, 25].

4.3 Shallow CNNs

A convolutional neural network is said to be shallow if the size on disk of the persisted network is $\leq \theta$ which is application dependent. In our case, the θ refers to the memory on disk of the random forests generated using the methods HQL (See Section 3.7), DQL (See Section 3.8). The architecture is developed by starting with basic architecture containing a convolutional layer, max-pooling layer, fully connected layer apart from the input and the output layers. The number of filters and nodes is then incremented until the network shares a similar footprint as the random forests. See Figure 4.5 for the generated architecture as mentioned.

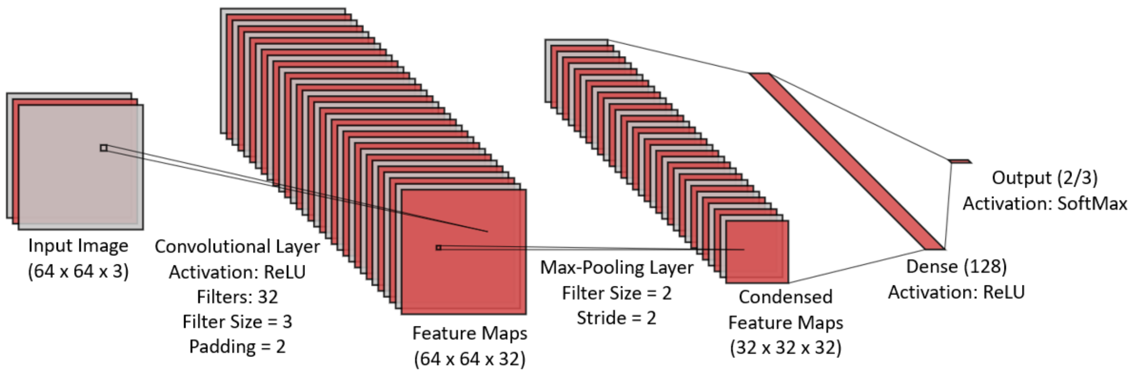


Fig. 4.5: Architecture for the Shallow Convolutional Network

The main advantage behind choosing shallow CNNs is that they can perform the classification much faster when compared to deep neural networks due to the small number of parameters and operations used by the shallow CNNs. The only disadvantage that we may face is low performance in comparison with the deep CNNs.

CHAPTER 5
COMPLEXITY ANALYSIS

Classifiers are generally evaluated based on the performance metrics like accuracy, precision, recall, f1-score, roc curve. This chapter defines how to calculate the number of parameters used and the number of operations performed by the neural networks and random forests to classify an input image. Comparison based on the number of parameters and operations gives us insight into the cost incurred for the model to classify the images or any other input data.

5.1 Number of Parameters

The number of parameters used while classifying a single image using neural networks or random forests is:

Neural Networks

1. Convolutional Layer:

   Let the input dimension to the convolutional layer be $R_I \times C_I \times L_I$ and use N_F filters of dimensions $R_F \times C_F$. The total number of parameters (P_{CL}) is the number of shared weights/filters and number of biases. This is expressed as follows:

$$P_{CL} = (R_F \times C_F \times L_I)N_F + N_F$$

2. Max-pooling Layer:

   The max-pooling layers does not have any additional parameters like other layers as the feature maps are converted into the low-dimension output using a maximum function within a region of fixed dimensions.

3. Fully-Connected Layer:

   Let the number of input nodes be N_I and the number of nodes in the fully-connected layer be N_{FC}. The total number of parameters (P_{FC}) is number of weights connected between the layers and number of biases (as observed in 5.1). This is expressed as follows:

$$P_{FC} = N_I N_{FC} + N_{FC}$$

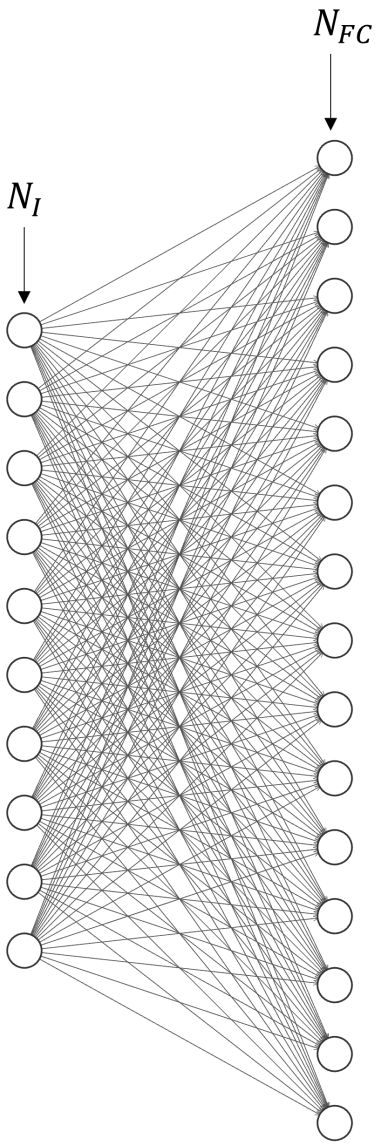


Fig. 5.1: Parameters involved in the classification of an image in a fully connected layer

Decision Trees

Assuming the decision tree generated is a full binary tree of height H, the maximum number of nodes (if each node is considered to be a parameter) used for the classification of single image is equal to H. The image is either traversed either to the left or right at a particular node based on a decision function as shown in figure 5.2

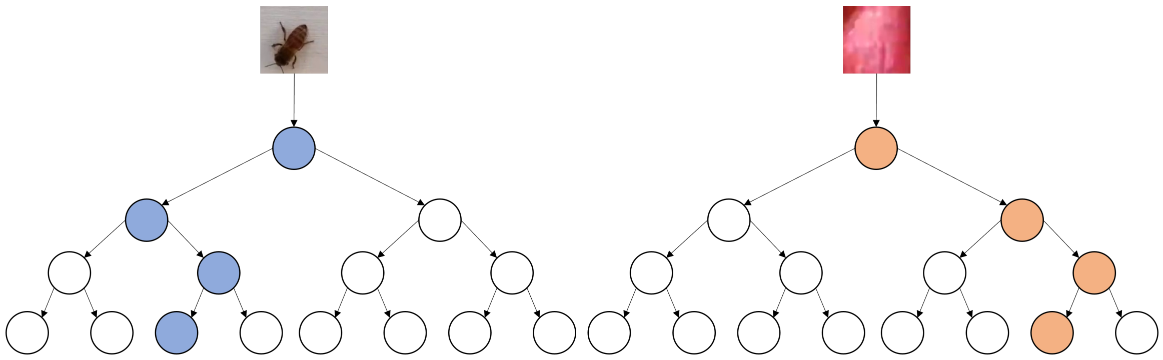


Fig. 5.2: Parameters involved in the classification of an image in a decision tree

LRF Overlap in RRF

Using the centre and dimension length (radius for circle and side length for square), the circles and square LRFs are generated when necessary and these LRFs are then used for the overlapping over the image. For each LRF there are $IMG_{width} \times IMG_{height}$ number of parameters that are used for the classification process. Therefore, for m LRFs, there are $m \times IMG_{width} \times IMG_{height}$ parameters.

5.2 Number of Operations

Operations such as addition, subtraction, multiplication, division, comparison, returning a value are considered as basic operations which can performed using a constant time. Assuming that an if-else operation takes 2 operations, the activation of output using ReLU which is maximum function between 0 and output, performs 2 operations. The following formulae are generated to represent the total number of basic operations performed while classifying a single image using neural networks and random forests.

Neural Networks

1. Convolutional Layer: Let the input dimension to the convolutional layer be $R_I \times C_I \times L_I$ and use N_F filters of dimensions $R_F \times C_F$. In a single convolution operation:

   (a) $R_F \times C_F \times L_I$ multiplications while the filter is overlapped onto the local receptive field

   (b) $R_F \times C_F \times L_I - 1$ additions while summing the multiplications

   (c) 1 bias addition

   (d) 2 operations for ReLU activation

   Together, there are $2 \times R_F \times C_F \times L_I + 2$ operations in a single convolution operation (See Figure 5.3). Therefore, the total number of operations happening in a convolutional layer (O_{CL}) for an output size of $R_o \times C_o$ is expressed as follows:

$$O_{CL} = (2 \times R_F \times C_F \times L_I + 2) \times R_o \times C_o \times N_F$$

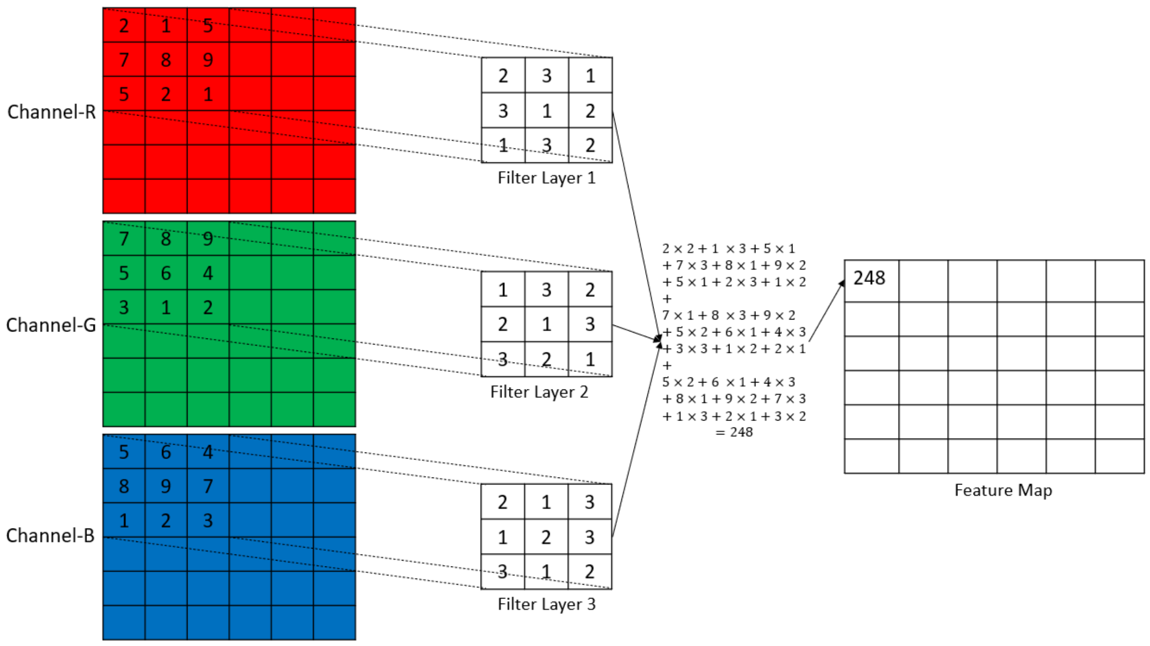


Fig. 5.3: Convolution Operation using the equation 4.1

2. Max-Pooling Layer:

   Let the input dimension to Max-Pooling layer be $R_I \times C_I \times L_I$ and filter dimensions be $R_F \times C_F$. In a single max-pooling operation, there are a maximum of $2 \times R_F \times C_F$ operations (See Figure 5.4).

   Therefore, the total number of operations happening in a max-pooling layer (O_{MP}) for an output size of $R_o \times C_o$ is expressed as follows:

$$O_{MP} = (2 \times R_F \times C_F) \times R_o \times C_o \times L_I$$

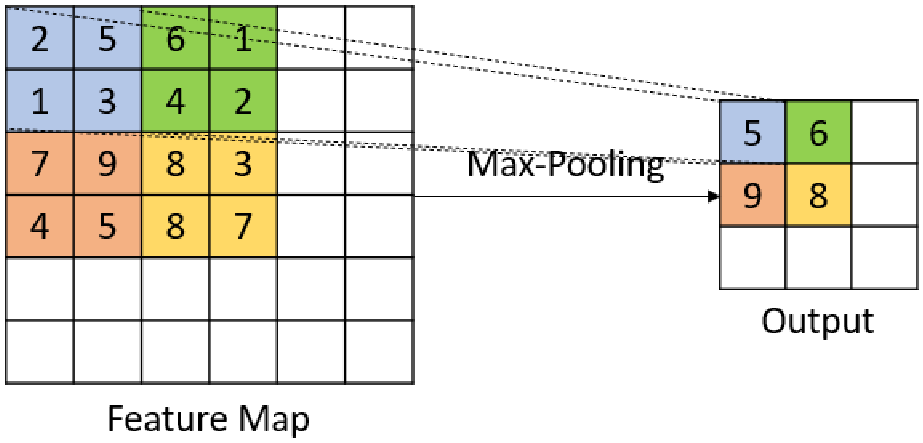


Fig. 5.4: Max-Pooling Operation

3. Fully-Connected Layer

   Let the number of input nodes be N_I and the number of nodes in the fully-connected layer be N_{FC}. For a single node in the fully connected layer, there are:

   (a) N_I multiplication while multiplying the input to the respective weight

   (b) $N_I - 1$ additions while computing sum of the products

   (c) 1 bias addition

   (d) A maximum of 2 operations for ReLU activation

Together, there are $2N_I + 2$ operations for a single node in the fully connected layer. Therefore, the total number of operations happening in a fully connected layer (O_{FCL}) is expressed as:

$$O_{FCL} = (2N_I + 2) \times N_{FC}$$

4. Output Layer

Let the number of nodes in the fully connected layer be N_{FC} and number of nodes in the output layer (number of classes) be N_{OL}. The activation used in the output layer is softmax which calculates the probabilities based on the weighted output calculated before the activation. In addition to the operations happening for the multiplications and additions as mentioned in previously, there are N_{OL} operations for calculating the sum of all weighted outputs, N_{OL} operations for normalizing the weighted outputs and $2N_{OL}$ operations for calculating the maximum activation for defining the class. Therefore, the total number of operations happening in a output layer (O_{OL}) is expressed as follows:

$$O_{OL} = 2N_{FC}N_{OL} + 4N_{OL}$$

Decision Trees

A decision tree classifier for an image uses the pixel values to threshold between the classes. Therefore, for a given height H, the image is traversed through every node by comparing the pixel value to the threshold making 2 operations at maximum. It is traversed until the image reaches the leaf node. Therefore, the total operations is at most $2H$.

Random Forests

In addition to the operations counted for each tree in the random forest using the method mentioned in section 5.2 for decision trees, the classification from the decisions of trees is also counted for the random forests.

After the generation of the trees and image being traversed through each decision tree of the random forest, each leaf node contains the class probabilities of all the classes which is calculated based on the proportion of the training of the data residing in this leaf node.

Assuming there are n trees with the data divided into C classes, there will be n lists of length C stating the probabilities of each of the C classes. Now the mean probability for each class is calculated. For each class, there are:

1. n - 1 additions of the probabilities

2. 1 division for calculating the mean probability

Together, there are n operations being performed for each class. The total number of operations for calculating all the C mean probabilities is nC. An additional $2C$ operations are performed for deciding the class based on maximum probability. Therefore, the total number of operations happening in random forest (O_{RF}) with decision trees of maximum depth of H is expressed as follows:

$$O_{RF} = 2nH + nC + 2C$$

LRF Overlap in RRF

The input image is overlapped onto the LRF before it is passed through a decision in the RRF. To perform the overlap, the input image needs to iterated over each pixel and be checked whether the LRF has a $value > 0$. For each LRF overlap, there are $IMG_{width} \times IMG_{height}$ number of operations during the classification process. Therefore, if there are n decision trees involved in the RRF, the total number of operations is equal to $n \times IMG_{width} \times IMG_{height}$.

5.2.1 Complexity Analysis Results

A comparison is performed on the shallow neural network that has been generated in 4.3, random reinforced forests, and standard random forests generated using the scikit-learn package. Assuming there are 200 decision trees of max height 50 in all the random forests and all the below models are classifying images into 2 classes, the numbers have been calculated as shown in table 5.1.

	RFs	**RRFs**	**SCNNs**
Parameters	10,000	91,920	4,195,328
Operations	20,404	839,604	15,991,560

Table 5.1: Total number of Parameters and Operations in Random Forests, Random Reinforced Forests and Shallow CNNs

CHAPTER 6
EXPERIMENTS AND RESULTS

6.1 Overview

In this chapter, we present the performances of each model discussed in chapters 3 and 4 and compare them based on the validation accuracies on various bee datasets, number of operations, and number of parameters used for classifying a single image.

6.2 Classification Models

Six different models are being compared based on the validation accuracies of the datasets. The best accuracies on each of the datasets on the various methods can are reported in the table 6.1. The following are the methods explained and used for the classification problem:

HQL Random Forests

The Q-Matrix obtained from using the HQL method (See Section 3.7) is used for the generation of random forest. Initially, the forest contains the two trees corresponding to the state and action of the highest Q-value in the Q-matrix. After the addition of the trees, the respective Q-value equates to zero to avoid recursion error. From the last action performed, we go to the respective state and perform the action with the highest q-value, and after the addition of the tree, this value again equates to zero. We repeat the process until the number of trees reaches 200. At every addition, the validation accuracy is recorded and this can be visualized in figures 6.1 - 6.7. We slice the forest at the maximum validation performance and persist it.

DQL Random Forests

Two Q-matrices are obtained using the DQL method following the two reward structures (See Section 3.8). From each of the Q-matrix, based on the highest Q-value, the state and action are retained. We add all the trees corresponding to the presence of trees from the state vector as mentioned (e.g., vector of length 200 [1, 0, 0, 1, 0, ..., 0, 1]) and then perform the action of adding the tree resulting in the best performing random forest according to the DQL method.

Performance Random Forests

As seen in HQL and DQL methods, the random forest is generated based on the highest Q-value. In the performance random forest (PRF), first, the trees are sorted in the decreasing order of the validation accuracies, and then the trees are added to the forest in this order. At every addition, the validation accuracy is recorded and this can be visualized in figures 6.1 - 6.7. We slice the forest at the maximum validation accuracy and persist it.

Scikit-Learn Random Forests

Using the Scikit-Learn's [26] "RandomForestClassifier" class in the ensemble module, multiple random forests with 50, 100, 150 decision trees are generated using "Gini" criterion and max_features argument as "sqrt" on the grayscale images of all the datasets.we

Shallow Convolutional Neural Networks

All the datasets have been trained, tested, and validated using the neural network architecture generated in the figure 4.5, by performing a grid search of the parameters learning rate, weight decay, and dropout percentage. The training model also uses model checkpoint to save the best model based on the lowest testing loss in all the epochs. The validation accuracies for the overall best models for each dataset have been reported in the table 6.1.

Notation: The **ALPHA** and **GAMMA** in figures 6.1 - 6.7 represent the learning rate (α) and the discount factor (γ) in the equation 3.1

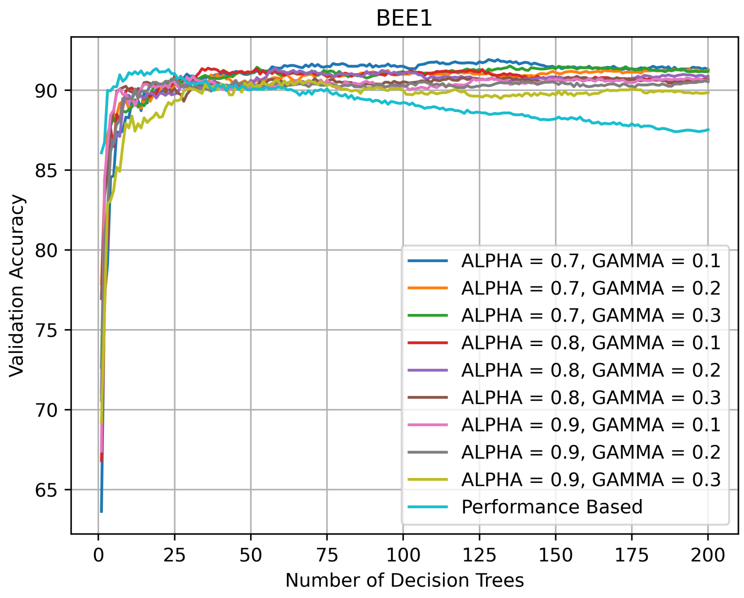


Fig. 6.1: Performances of different HQL RFs and PRF on the validation set of BEE1

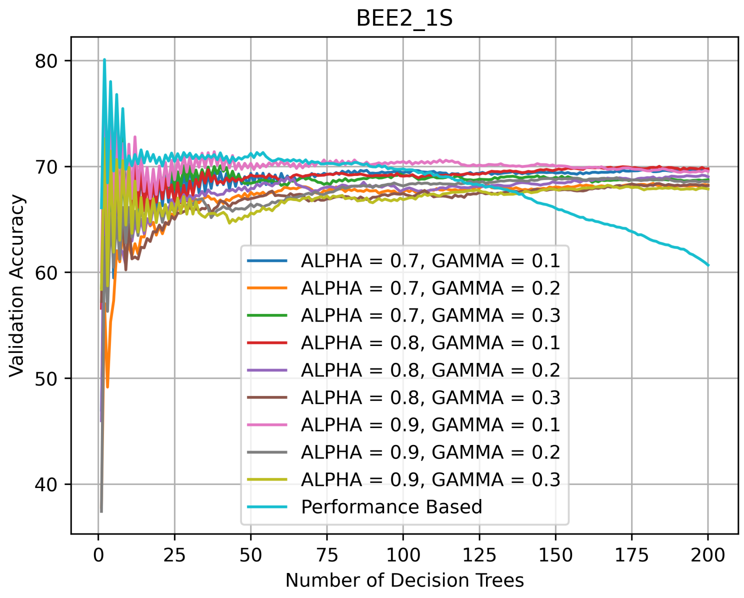


Fig. 6.2: Performances of different HQL RFs and PRF on the validation set of BEE2_1S

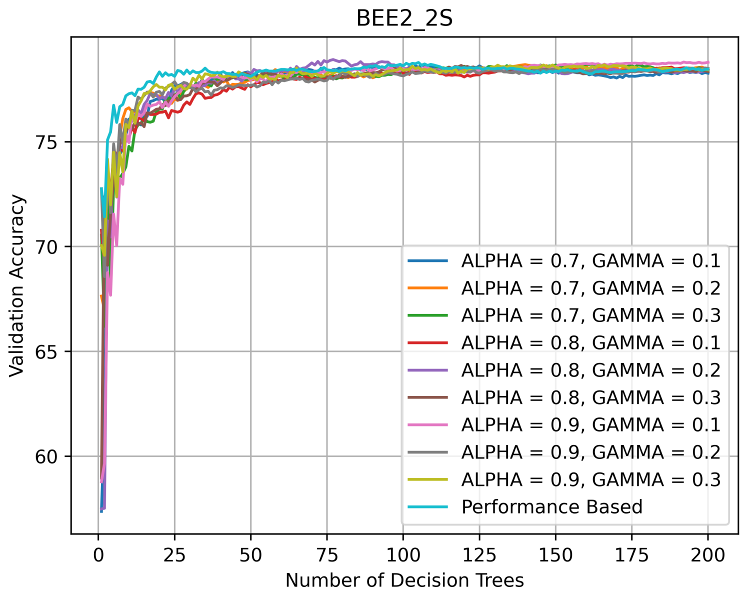


Fig. 6.3: Performances of different HQL RFs and PRF on the validation set of BEE2_2S

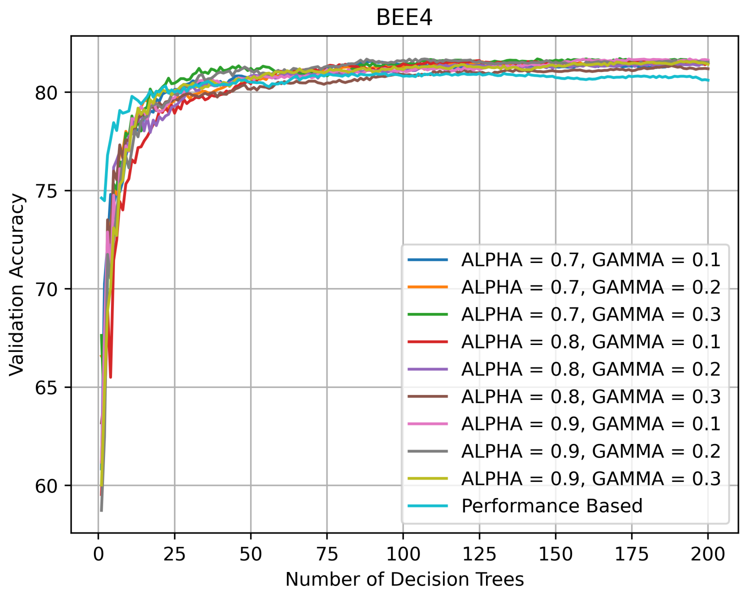


Fig. 6.4: Performances of different HQL RFs and PRF on the validation set of BEE4

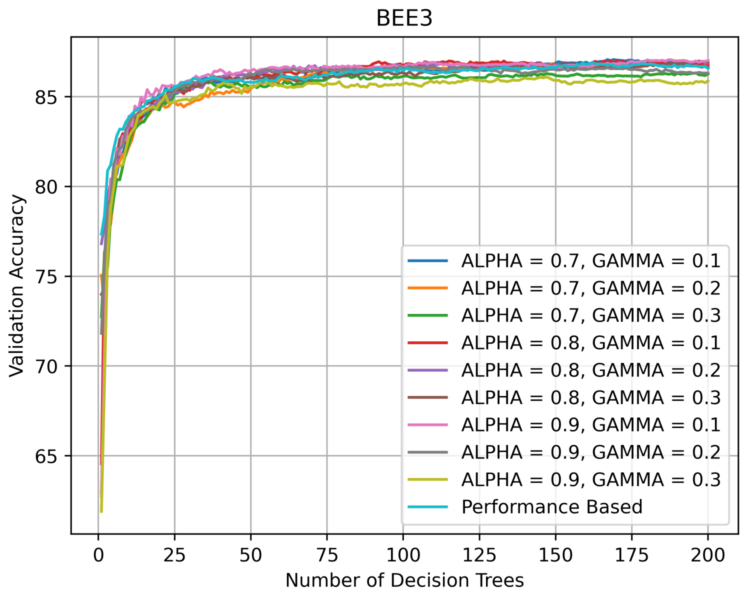


Fig. 6.5: Performances of different HQL RFs and PRF on the validation set of BEE3 No Shadow

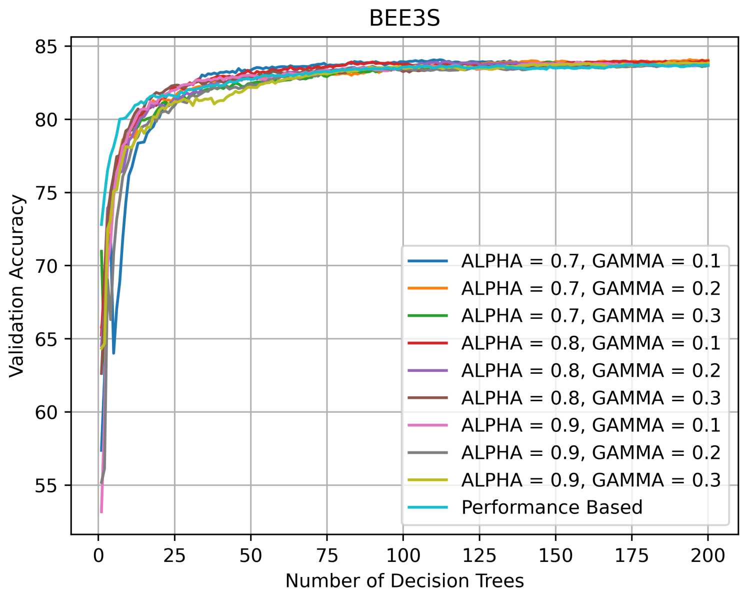


Fig. 6.6: Performances of different HQL RFs and PRF on the validation set of BEE3

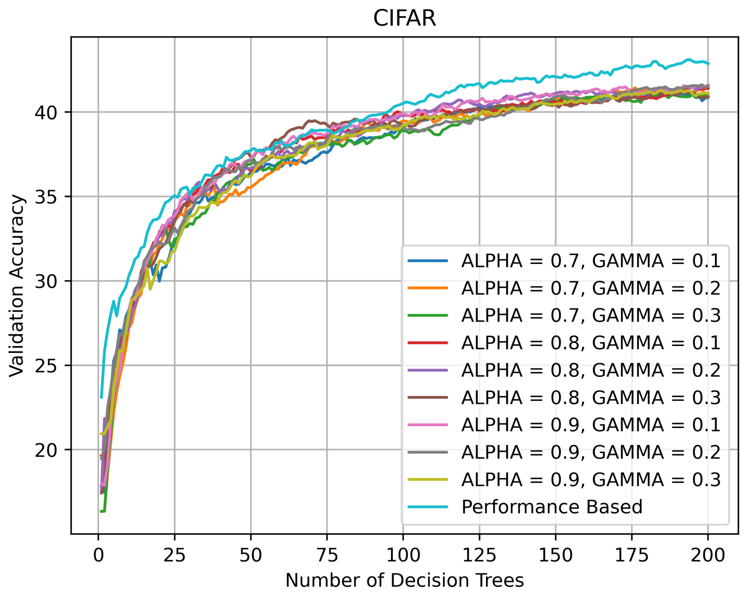


Fig. 6.7: Performances of different HQL RFs and PRF on the validation set of CIFAR-10

	HQL	**DQL1**	**DQL2**	**PRF**	**Skl_RF**	**Skl_RF (Color)**	**SCNN**
BEE1	91.92	84.89	89.88	91.35	93.31	93.79	97.00
BEE2_1S	78.12	68.93	79.68	80.10	76.09	63.00	89.16
BEE2_2S	78.92	71.63	79.05	78.78	77.76	74.46	76.29
BEE4	81.71	73.72	81.10	80.95	79.68	74.61	77.85
BEE3NS	87.07	77.77	87.22	86.91	87.54	81.74	90.52
BEE3S	84.08	70.97	83.35	83.77	83.83	83.35	88.08
CIFAR-10	41.62	25.45	40.56	43.11	41.90	47.07	65.56

Table 6.1: Validation Accuracies on different datasets using random forests generated using various methods explained and shallow convolutional neural networks

CHAPTER 7
CONCLUSION

This thesis performs an empirical and theoretical investigation by comparing various types of random forests and shallow convolutional neural networks. The comparison is based on validation accuracies and the number of parameters and operations used for the classification. We also present a new dataset called BEE4, a newly curated dataset generated by correcting 996 mislabeled images into their suitable classes. Shallow CNNs perform the best on four bee datasets BEE1, BEE2_1S, BEE3 No Shadow, BEE3 with Shadow with validation accuracies of 97.0%, 89.16%, 90.52%, 88.08% respectively. BEE2_2S and BEE4 are the most challenging datasets among the bee datasets due to the positioning of the camera at the second super. These two datasets, although being difficult, the random forests of all kinds except DQL1 perform close enough and perform better than the shallow CNNs. The best validation accuracies on BEE2_2S and BEE4 using shallow CNNs are 79.05% and 81.71%, respectively. The process has been repeated on a publicly available dataset, CIFAR-10, to check the performance behavior on a NON-BEE dataset. Though all the random forests (except DQL1) perform close enough with the best validation accuracy of 43.11%, shallow CNNs perform far better with an accuracy of 65.56%. Additionally, the random forests generated through the DQL1 method are the lowest in the performance on all of the datasets. One of the reasons for the shallow CNNs better performance might be the over-parameterization for the classification [27].

During the training of the shallow CNNs, we performed manual tuning of the hyper-parameters, the results of which were not as good as the ones achieved through grid search. We experimented with the grid-search containing multiple learning rate values, weight decay, and dropout, which gave far better performances as reported. Finally, we observed that the different combinations of all the hyper-parameters had made all the massive differences in performances with the same shallow networks. This observation supports the *Universal*

Approximation Theorem that neural networks can closely approximate any function *f(x)* [28] and also conveys the importance of hyper-parameters for a given architecture.

We performed a complexity analysis to compare the random forests and shallow CNNs based on the number of parameters used and operations performed for classifying a single image. The training time for shallow CNNs was approximately 30 - 40 minutes per classifier per dataset. The overall time taken for generating a random reinforced forest is about 10 hours, out of which the reinforcement learning process consumed the significant time of ≈ 9 hours. The reinforcement learning time depends on the validation data size because, at each step of the learning, the reward calculation is based on the validation accuracy. Although the RRFs, take a considerable time generating a classifier, the complexity involved in the classification process is significantly lower when compared to the shallow CNNs. The RRFs perform only 5% of the number of operations performed in the SCNNs and considers only about 2% of the number of parameters considered in the SCNNs. This low complexity is beneficial when the machine learning models are deployed onto embedded platforms like the Raspberry Pi.

We also formulated a theorem to support that random forests generated from the DQL method are unlikely to generate a best random forest than the other methods and do not say anything about reward strategies. The theorem basically proves that when number of best random forests that are possible tends to become low, the chances of not finding it through DQL method becomes higher. Another reason for the low performance of the random reinforced forests and performance random forests might be due to invaluable local receptive fields. For example, the corner LRFs mainly do not capture the bee, resulting in a poor-performing decision tree.

CHAPTER 8
FUTURE WORK

Selecting more valuable LRFs might increase the quality of the random forest as a whole because some of the trees with certain LRFs seem misguided. LRFs of different shapes that can focus more at the center of images might increase the performance and hold only if most images contain bee at the central region. Another way that might improve the performance of the random reinforced forests would be to remove the trees that are not performing well on the dataset and use the leftover trees. We can do this by choosing a threshold validation accuracy or a threshold number of top-performing trees that can be used for further learning. The random forest generation from the order of trees in HQL and PRF can be optimized using algorithm that is similar to *Maximum Sum Sub-sequence algorithm.*

Reverse RRF can be also experimented by starting with a random forest containing all the 200 trees and use the reinforcement learning methods to delete the trees that can increase the performance of the overall random forest.

More experiments can be done on shallow CNNs by increasing the number of filters and nodes instead of the layers, which would not drastically increase the number of parameters and operations. Early stopping is one of the callback techniques which stops the training when the network is either stuck at a local minimum or when the performance is not increasing for a couple of epochs. Using this callback function can reduce the training time and helps explore more combinations of the hyper-parameters.

We can also perform this comparison with other challenging image datasets like EMNIST, ImageNet, Chest X-rays, etc., and beehive audio datasets by choosing linear LRFs instead of various 2D shapes. Additionally, the comparison of the neural networks and random forests can also be based on the energy consumed (power usage) for classifying the images when deployed onto a raspberry pi 3. The energy consumed can be calculated by monitoring the power usage before and during the classification process.

REFERENCES

[1] V. Kulyukin and S. Mukherjee, "On video analysis of omnidirectional bee traffic: Counting bee motions with motion detection and image classification," *Applied Sciences*, vol. 9, no. 18, p. 3743, 2019.

[2] V. Kulyukin, "Audio, image, video, and weather datasets for continuous electronic beehive monitoring," *Applied Sciences*, vol. 11, no. 10, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/10/4632

[3] L. Alavala, "Bee shadow recognition in video analysis of omnidirectional bee traffic," 2019. [Online]. Available: https://digitalcommons.usu.edu/etd/7624

[4] E. Genersch, "Honey bee pathology: current threats to honey bees and beekeeping," *Applied microbiology and biotechnology*, vol. 87, no. 1, pp. 87–97, 2010.

[5] B. P. Oldroyd, "What's killing american honey bees?" *PLoS Biol*, vol. 5, no. 6, p. e168, 2007.

[6] J. D. Evans, C. Saegerman, C. Mullin, E. Haubruge, B. K. Nguyen, M. Frazier, J. Frazier, D. Cox-Foster, Y. Chen, R. Underwood *et al.*, "Colony collapse disorder: a descriptive study," *PloS one*, vol. 4, no. 8, p. e6481, 2009.

[7] S. K. Evans *et al.*, "Electronic beehive monitoring-applications to research." *Julius-Kühn-Archiv*, no. 450, pp. 121–129, 2015.

[8] W. Meikle and N. Holst, "Application of continuous monitoring of honeybee colonies," *Apidologie*, vol. 46, no. 1, pp. 10–22, 2015.

[9] V. Estivill-Castro, D. Lattin, F. Suraweera, and V. Vithanage, "Tracking bees-a 3d, outdoor small object environment," in *Proceedings 2003 International Conference on Image Processing (Cat. No. 03CH37429)*, vol. 3. IEEE, 2003, pp. III–1021.

[10] T. Kimura, M. Ohashi, R. Okada, and H. Ikeno, "A new approach for the simultaneous tracking of multiple honeybees for analysis of hive behavior," *Apidologie*, vol. 42, no. 5, p. 607, 2011.

[11] C. Chen, E.-C. Yang, J.-A. Jiang, and T.-T. Lin, "An imaging system for monitoring the in-and-out activity of honey bees," *Computers and electronics in agriculture*, vol. 89, pp. 100–109, 2012.

[12] A. Ghadiri, "Implementation of an automated image processing system for observing the activities of honey bees," Ph.D. dissertation, Appalachian State University, 2013.

[13] V. Kulyukin. Beepi: A multisensor electronic beehive monitor. [Online]. Available: https://www.kickstarter.com/projects/beepihoneybeesmeetai/beepi-a-multisensor-electronic-beehive-monitor

[14] ——. Beepi: Honeybees meet ai: Stage 2. [Online]. Available: https://www.kickstarter.com/projects/beepihoneybeesmeetai/beepi-honeybees-meet-ai-stage-2

[15] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[16] B. Charbuty and A. Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, no. 01, pp. 20–28, 2021.

[17] K. Mathan, P. M. Kumar, P. Panchatcharam, G. Manogaran, and R. Varadharajan, "A novel gini index decision tree data mining method with neural network classifiers for prediction of heart disease," *Design automation for embedded systems*, vol. 22, no. 3, pp. 225–242, 2018.

[18] D. M. Farid, M. Z. Rahman, and C. M. Rahman, "An ensemble approach to classifier construction based on bootstrap aggregation," *International Journal of Computer Applications*, vol. 25, no. 5, pp. 30–34, 2011.

[19] R. S. Sutton and A. Barto. "Reinforcement Learning: An introduction, volume 1. MIT press Cambridge, 1998.".

[20] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[21] M. A. Nielsen, *Neural networks and deep learning.* Determination press San Francisco, CA, 2015, vol. 25.

[22] A. LeNail, "Nn-svg: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[23] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017, pp. 1–6.

[24] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE, 2011, pp. 342–347.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, “The computational limits of deep learning,” *arXiv preprint arXiv:2007.05558*, 2020.

[28] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

APPENDICES

APPENDIX A

Detailed Results of the HQL and DQL Experiments

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.7	0.1	91.92	128
0.7	0.3	91.52	135
0.8	0.1	91.38	36
0.8	0.2	91.38	58
0.7	0.2	91.3	189
0.9	0.1	90.87	31
0.8	0.3	90.87	157
0.9	0.2	90.73	25
0.9	0.3	90.62	58

Table A.1: Validation Accuracies on BEE1 Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.2	84.89	2
0.9	0.2	81.66	2
0.8	0.1	80.67	2
0.9	0.1	79.34	2

Table A.2: Validation Accuracies on BEE1 Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	89.88	6
0.9	0.2	88.95	36
0.8	0.1	88.75	35
0.8	0.2	88.29	68

Table A.3: Validation Accuracies on BEE1 Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	78.13	2
0.8	0.1	77.19	2
0.7	0.3	75.35	2
0.8	0.3	73.01	2
0.9	0.3	72.63	2
0.8	0.2	72.56	4
0.7	0.1	70.34	12
0.9	0.2	68.95	170
0.7	0.2	68.41	182

Table A.4: Validation Accuracies on BEE2_1S Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	68.93	2
0.9	0.2	68.66	2
0.8	0.2	65.09	2
0.8	0.1	61.26	2

Table A.5: Validation Accuracies on BEE2_1S Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	79.68	2
0.8	0.2	79.23	2
0.8	0.1	79.14	2
0.9	0.2	78.63	2

Table A.6: Validation Accuracies on BEE2_1S Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.2	78.92	77
0.9	0.1	78.8	192
0.7	0.3	78.71	162
0.7	0.2	78.68	140
0.9	0.3	78.67	166
0.7	0.1	78.66	106
0.8	0.3	78.64	119
0.8	0.1	78.56	188
0.9	0.2	78.52	199

Table A.7: Validation Accuracies on BEE2_2S Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.2	71.63	3
0.8	0.1	70.91	3
0.9	0.2	70.86	3
0.9	0.1	69.19	3

Table A.8: Validation Accuracies on BEE2_2S Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	79.05	61
0.8	0.1	78.65	63
0.9	0.2	78.61	54
0.8	0.2	78.38	77

Table A.9: Validation Accuracies on BEE2_2S Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.7	0.3	81.71	171
0.9	0.2	81.69	107
0.9	0.1	81.69	174
0.8	0.1	81.61	125
0.7	0.1	81.58	90
0.9	0.3	81.57	191
0.7	0.2	81.55	126
0.8	0.2	81.51	200
0.8	0.3	81.35	188

Table A.10: Validation Accuracies on BEE4 Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	73.72	3
0.8	0.1	73.67	3
0.8	0.2	73.07	3
0.9	0.2	72.71	3

Table A.11: Validation Accuracies on BEE4 Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.1	81.10	74
0.8	0.1	80.86	74
0.9	0.2	80.85	47
0.8	0.2	80.56	58

Table A.12: Validation Accuracies on BEE4 Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.7	0.1	87.07	167
0.9	0.1	87.07	187
0.8	0.1	87	115
0.8	0.3	86.99	197
0.7	0.2	86.93	181
0.8	0.2	86.9	178
0.9	0.2	86.69	102
0.7	0.3	86.33	196
0.9	0.3	86.1	147

Table A.13: Validation Accuracies on BEE3 (No Shadow) Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.2	77.77	3
0.8	0.2	76.76	3
0.9	0.1	76.50	3
0.8	0.1	75.86	3

Table A.14: Validation Accuracies on BEE3 (No Shadow) Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.2	87.22	74
0.9	0.1	87.04	82
0.8	0.1	86.78	80
0.8	0.2	86.60	75

Table A.15: Validation Accuracies on BEE3 (No Shadow) Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.7	0.2	84.08	194
0.7	0.1	84.07	112
0.8	0.1	84.02	198
0.8	0.2	83.93	114
0.7	0.3	83.92	147
0.9	0.1	83.92	157
0.9	0.3	83.91	187
0.8	0.3	83.81	190
0.9	0.2	83.78	196

Table A.16: Validation Accuracies on BEE3 Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.1	70.97	3
0.9	0.1	70.35	3
0.9	0.2	70.28	3
0.8	0.2	66.44	2

Table A.17: Validation Accuracies on BEE3 Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.2	83.35	79
0.9	0.1	83.31	77
0.9	0.2	83.31	77
0.8	0.1	83.11	85

Table A.18: Validation Accuracies on BEE3 Using DQL2

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.9	0.2	41.62	198
0.8	0.2	41.57	200
0.9	0.1	41.56	197
0.8	0.1	41.46	196
0.7	0.2	41.44	176
0.9	0.3	41.29	191
0.7	0.1	41.21	188
0.8	0.3	41.2	191
0.7	0.3	41.06	185

Table A.19: Validation Accuracies on CIFAR-10 Using HQL

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.1	25.45	2
0.9	0.2	24.65	6
0.9	0.1	23.08	3
0.8	0.2	22.97	3

Table A.20: Validation Accuracies on CIFAR-10 Using DQL1

ALPHA	GAMMA	Validation Accuracy	Number of Decision Trees
0.8	0.2	40.56	83
0.9	0.2	40.49	83
0.8	0.1	40.19	86
0.9	0.2	17.57	2

Table A.21: Validation Accuracies on CIFAR-10 Using DQL2