8-2018

A Deep Learning Approach to Recognizing Bees in Video Analysis of Bee Traffic

Astha Tiwari
Utah State University

A DEEP LEARNING APPROACH TO RECOGNIZING BEES IN VIDEO ANALYSIS OF BEE TRAFFIC

by

Astha Tiwari

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	Xiaojun Qi, Ph.D. Committee Member
Nicholas Flann, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

A Deep Learning Approach to Recognizing Bees in Video Analysis of Bee Traffic

by

Astha Tiwari, Master of Science

Utah State University, 2018

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

This thesis presents a Deep Learning (DL) approach to recognize bees in videos for the analysis of bee traffic in real time. Bee traffic will further help in analysis of forager traffic to monitor bee colony behavior. Various Convolutional Neural Networks (ConvNet) and Multi Layer Perceptron (MLP) models have been designed and trained to experiment, compare and analyze different factors that can affect the performance of the model. The best model then is integrated with a motion detection algorithm that is made to work on a Raspberry Pi 3 computer in collaboration with Electronic Beehive Monitoring System. Modern computer vision tasks are dominated by ConvNets due to their superior performance. Many aspects of ConvNets have been examined in various publications but literature about the analysis and construction of neural network architectures are rare. This work is an attempt to fill this gap and closely analyze and compare the difference in performance of MLPs and ConvNets.

(47 pages)

PUBLIC ABSTRACT

A Deep Learning Approach to Recognizing Bees in Video Analysis of Bee Traffic

Astha Tiwari

Colony Collapse Disorder (CCD) has been a major threat to bee colonies around the world which affects vital human food crop pollination. The decline in bee population can have tragic consequences, for humans as well as the bees and the ecosystem. Bee health has been a cause of urgent concern for farmers and scientists around the world for at least a decade but a specific cause for the phenomenon has yet to be conclusively identified.

This work uses Artificial Intelligence and Computer Vision approaches to develop and analyze techniques to help in continuous monitoring of bee traffic which will further help in monitoring forager traffic. Bee traffic is the number of bees moving in a given area in front of the hive over a given period of time. And, forager traffic is the number of bees entering and/or exiting the hive over a given period of time. Forager traffic is an important variable to monitor food availability, food demand, colony age structure, impact of pesticides, etc. on bee hives. This will lead to improved remote monitoring and general hive status and improved real time detection of the impact of pests, diseases, pesticide exposure and other hive management problems.

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor, Dr. Vladimir Kulyukin. The door to Prof. Kulyukin's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently supported me and steered me in the right the direction whenever he thought I needed it. I acknowledge my gratitude to my committee members, Dr. Xiaojun Qi and Dr. Nicholas Flann for continuous support, I would like to thank my beloved family and friends for cheering me up and encouraging me throughout this academic pursuit.

Astha Tiwari

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ANN	Artificial Neural Network
ConvNet	Convolutional Neural Network
CCD	Colony Collapse Disorder
DL	Deep Learning
EBM	Electronic Beehive Monitoring
FC	Fully Connected
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
SGD	Stochastic Gradient Descent
POOL	Maxpooling Layer
CONV	Convolutional layer

CHAPTER 1

INTRODUCTION

1.1 Background

The Western honey bee (Apis mellifera L.) is the world's premier managed pollinator species. Demand for its services has soared from fruits, nuts (especially almonds) and vegetable farmers. That represents almost 100 crop species, making up one-third of the average diet. In the United States, honey bees pollinate an estimated $15 billion of crops each year [1].

Apiculture has been on the decline in both the USA and Europe over recent decades [2]. It is therefore crucial to make beekeeping a more attractive hobby and a less laborious profession, in order to encourage local apiculture and pollination. Sudden losses of honey bee colonies have occurred, and have received considerable public attention. Colony Collapse Disorder (CCD) in the USA has attracted great attention, and scientists here and in Europe are working hard to provide explanations for these extensive colony losses. Disruption of the honey bee supply raised prices for domestically grown nuts, fruits and vegetables.

These factors has lead bee health to be a cause of urgent concern for scientists and farmers around the world for at least a decade and a lot of work has been done in past for data collection and analysis for its research. One of the important variable in order to monitor food availability, food demand, colony age structure, impact of pesticides, etc. in bee-hives is forager traffic [3]. Forager activity is an important variable to monitor when evaluating the impact of pesticides on honey bee colony health [4]. Monitoring forger traffic will lead to improved remote monitoring of general hive status and improved real time detection of the impact of pests, diseases, pesticide exposure and other hive management problems. Since forager traffic can be affected by food availability, food demand, and colony age structure. Thus, sudden changes in that traffic may indicate acute changes on the colony level. Forager activity is described in terms of the number of bees entering and/or exiting the hive over a given time period, data can be collected, if need be, without the use

of equipment more sophisticated than an observer and a stopwatch. The use of human observation, while likely to be accurate, clearly limits due to fatigue and the amount of time that the hive can be observed.

Thus automation of this task has gained popularity and Electronic Beehive Monitoring Systems (EBM) came into picture. EBM makes it possible to capture large amounts of useful information about the behavior of the bees without interrupting the life cycles of honey bee colonies [5]. It uses video, audio, and temperature data to estimate the state of a bee colony.

1.2 Related Work

The importance of forager traffic in estimating the health of bees demands research and advancement over the span of several years in this area. EBM systems help in automating the task of collecting large amount of useful information on behavior without invasive and disruptive colony inspections. As EBMs gained popularity, it became important to focus on the design of EBMs as well as incorporating modern software solutions to aid and better utilize them. I have described such related work below.

One such work addresses the hardware design of EBM systems addressing which sensors to include and how to power the system. It uses the power analysis of the EBM system. The EBM system made to work either with a UB12120 12V 12Ah standard lead-acid battery or an Anker^{TM} Astro E7 5V lithium-ion battery to analyze the power requirements. These batteries are recharged by Renogy 50W 12V Monocrystalline Solar Panel. Power analysis is performed using both batteries to calculate systems efficiency. The performed power analysis indicates that the Anker (TM) Astro E7 26800mAh 5V lithium-ion battery runs approximately 6 hours more than the lead acid battery. Moreover, the lithium-ion battery is compact, light weight, more efficient and has a longer cycle life. Using lithium-ion batteries will likely result in fewer hardware components and a smaller environmental footprint [6].

Apart from the hardware, there have been algorithms proposed to contribute to the off-the-grid EBM. Specifically, the algorithms that can use computer vision to contribute towards solving the bee counting problem which in turn provides solutions to estimate forager traffic. Two algorithms are presented for omni-directional bee counting from images to estimate forager traffic levels [7].

The first bee identification method is based on a contour detection algorithm. An image is binarized and a list of contours, connected components of pixels are computed. Contours with fewer than 30 or more than 50 pixels are removed. The number of found contours is an estimate of the number of bees on the pad. The second bee identification algorithm is based on binary pixel separation of the cropped landing pad. Each pixel is inspected for the presence of green color. If the presence of the green color exceeds a threshold, the pixel is labeled as a pad pixel. Otherwise, the pixel is labeled as a bee pixel. An estimate of the number of bees on the landing pad is then obtained by dividing the number of detected bee pixels by 30, which is the average number of pixels, experimentally found, in an individual bee.

As an improvement of the previous mentioned work, an algorithm was presented for in situ omni-directional bee counting on Langstroth landing pads. Unlike this algorithm, previous two algorithms were not in situ in that they were implemented in JAVA with OpenCV v2 and evaluated on a system with 10GB RAM running Ubuntu 12.02 LTS. The concept of the 1D Haar Wavelet Spike was developed and the performance of the algorithm was tested in situ. The algorithm is more accurate than the previous two algorithms for omni-directional bee counting. [8]

Deep Learning (DL) is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. It allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains.

Image classification in general is a challenging task due to the variabilities within images. These include but are not limited to the lighting condition, scaling, rotation and various other sources of corruption. Numerous algorithms exist that try to minimize these effects by listing out the low level of features to be used for classifying an image. However, these may or may not be the best solution depending on the application and it is difficult to list a general set of features. Learning through Deep Neural Networks (DNNs) has gained significant attention in recent years due to their classification accuracies. These networks are often complex and involve high level features to represent more abstract patterns in the images. The generated features are often more

generalized and unrecognizable to humans but add robustness to the classification. DNNs usually involve hierarchical layers of patterns through hidden layers and evolutionary algorithms to select the highest probable class.

With DL models for image classification, an exponential decline of error rate has been achieved through last few years. Yann Lecun et. al., presented a paper pioneering the Convolutional Neural Network (ConvNets) in 1998, where the authors explain how ConvNets have been shown to eliminate the need of hand-crafted feature extractors [9]. However, DL really took off recently and the main reasons are the increased processing power (GPUs), the availability of abundant data, and new algorithms and techniques. The major breakthrough was AlexNet, a large, deep ConvNet which won the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which is a competition where research teams evaluate their algorithms on the given data set and compete to achieve higher accuracy on several visual recognition tasks. In the paper, the group discussed the architecture of the network AlexNet. The network was made up of 5 CONV layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories [10]. With AlexNet's success, more models were made and achieved better performance, one such is ZF Net [11]. They introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Similarly, a lot of work has been done in the past using different datasets and achieving higher accuracy on image classification on them by using various neural network architectures on handwritten MNIST dataset [12, 13], where new modifications in the architectures and algorithms were proposed to decrease the error rates in existing solutions.

1.3 Current Work

In this thesis, several contributions of DL to the off-the-grid EBM are investigated. Specifically, how DL can be used to develop computer vision based techniques in order to estimate bee traffic in video analysis that can help in future for forager traffic analysis. This thesis is organized as follows; Chapter 2 discusses the details about DL, MLP and ConvNets, focusing on the mathematics behind the Artificial Neural Network (ANN) architecture and how a model learns. Chapter 3 describes about the dataset used for training the ANN models and the data preprocessing process, it also

describes the algorithm used in order to recognize bees from a video on the Raspberry Pi. Chapter 4 details the experiment setup and the analysis of various parameter and hyper-parameters on the performance of a neural network. It explains the architecture of the models used in experiments. Chapter 5 draws conclusions from the experiments and discusses the future work.

CHAPTER 2

Deep Learning Models

Over last few years, DL methods have appeared to outperform previous state of the art machine learning (ML) methods in several fields, with computer vision being a standout amongst the most noticeable cases. Conventional ML techniques are limited in their ability to process natural raw data. For decades, constructing a pattern recognition or ML system required careful feature engineering and considerable domain expertise to design a feature extractor that transformed the raw data into a suitable internal representation or feature vector from which the learning system such as a classifier could detect or classify patterns in the input. However, the performance of ML algorithms can suffer substantially when the information is buried in meaningless features. The goal behind DL is to automatically learn the features from data; here the algorithms will do the feature engineering so as to provide deep neural network structures with meaningful information that it can learn more effectively [14].

The next sections provide detailed explanations of various DL models such as MLP and ConvNet.

2.1 Artificial Neural Network Model

The idea behind ANN is to simulate the working of human brain. John J. Hopfield [15] describes how ANN tries to fabricate networks and devices in the spirit of neurobiology and how they are designed to solve problems like pattern recognition, etc. Any neural network model, MLP or ConvNet consists of common basic elements; the neurons which are the basic unit of a neural network, several layers including input, hidden and output layers and an activation function that ensures that the representation in the input space is mapped to a different space in the output. Following subsections describe each component in detail.

Neurons

Neurons are the basic unit of a neural network, as the ANN copies the functionality and structure of biological neural networks where the neurons have a number of dendrites, a cell nucleus and an axon in our brain, similarly the ANNs have inputs, processor and output respectively. There are several inputs to every neuron, for each input there is a weight of that specific connection. When the artificial neuron activates, it computes its state, by adding all the incoming inputs multiplied by its corresponding connection weight. But neurons always have one extra input, the bias, which is always 1, and has its own connection weight. This makes sure that even when all the inputs are all zeros, there is going to be an activation in the neuron. Fig 2.1 shows the components of a Perceptron/Neuron.

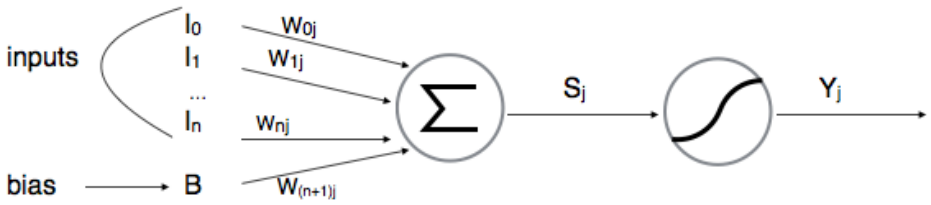


Fig. 2.1: Perceptron model with inputs, outputs, weights and biases.

Layers

There are 3 types of layers in Neural Network Models; Input layer, Hidden layer and Output layer. Fig 2.2 shows such a simple model

1. **Input Layer:** The Input layer communicates with the external environment that presents a pattern to the neural network. It deals with all the inputs only. This input gets transferred to the hidden layers. Every input neuron should represent some independent variable that has an influence over the output of the neural network.

2. **Hidden Layer:** The hidden layer is a collection of neurons which has activation function applied on it and it is an intermediate layer found between the input layer and the output

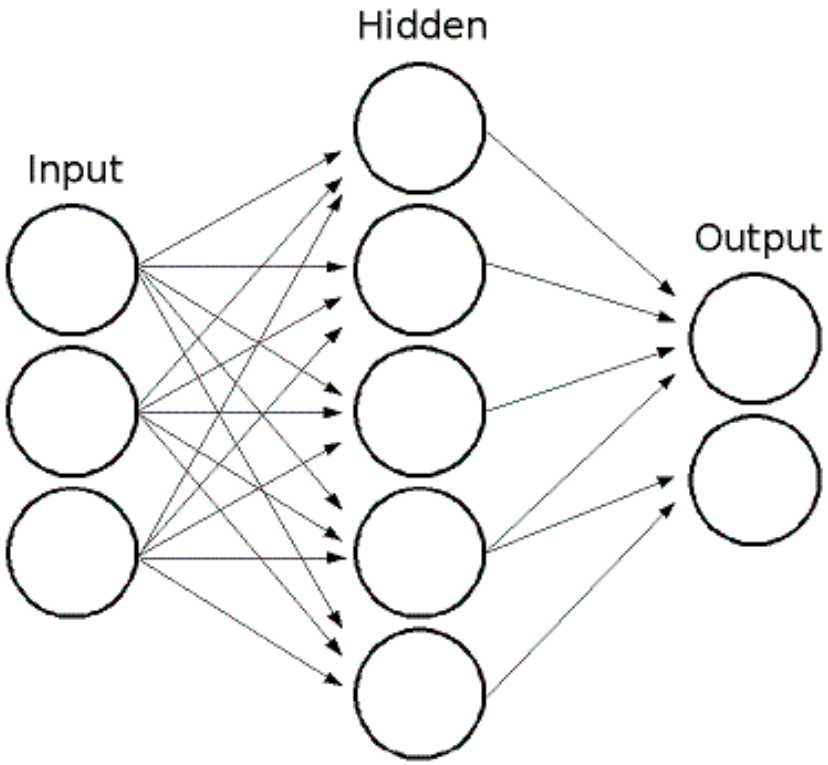


Fig. 2.2: Basic ANN architecture with an Input Layer, One Hidden Layer and an Output Layer

layer. It processes the inputs obtained by its previous layer. It is the layer which is responsible for extracting the required features from the input data.

3. **Output Layer:** The output layer of the neural network collects and transmits the information accordingly in way it has been designed to give. The number of neurons in output layer are the number of outcomes we want from it, for example for a binary classifier as in our case it should be 2.

Activation Function

Activation functions are really important for a Artificial Neural Network to learn complicated and non-linear complex functional mappings between the inputs and target variable. They introduce non-linear properties to the Network. The reason ANN need non-linearity so that the output can't be reproduced from a linear combination of the inputs. Without a non-linear activation function in the network, an ANN would behave just like a single-layer perceptron because summing

these layers would give us just another linear function. In ANN as in equation 2.2 we do the sum of products of inputs(X) and their corresponding Weights(W) and apply a Activation function φ to it to get the output of that layer and feed it as an input to the next layer.

In my architectures I have used Softmax function as the Activation Function in the Output layer of the MLP and CNN to ensure that the outputs are probabilities and they add up to 1. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one. So, in this case, Probability(Bee) + Probability(NoBee) = 1

Mathematically the softmax function is shown below, where a_k is the activation value of output k, and K is the number of classes:

$$Y_k = \frac{e^{a_k}}{\sum_{l=1}^{K} e^{a_l}} \tag{2.1}$$

2.1.1 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is perhaps one of the most traditional type of DL architectures. An MLP is a network of simple neurons called perceptrons. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then putting the output through some nonlinear activation function. The development of the back-propagation learning algorithm for determining weights in an MLP has made these networks very popular and perform better [16]. A typical MLP network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer. Here every element of a previous layer, is connected to every element of the next layer.

The computations performed by such a network with nonlinear activation functions and a linear output layer can be written mathematically as:

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b) = \phi(\sum_{i=1}^{n} W^T X + b) \tag{2.2}$$

where $\mathbf{w}$ denotes the vector of weights, $\mathbf{x}$ is the vector of inputs, b is the bias and φ is the activation function.

The algorithm for MLP consists of two steps. In the forward pass, the predicted outputs corresponding to the given inputs are evaluated as in equation 2.2. In the backward pass, partial derivatives of the cost function with respect to the different parameters are propagated back through the network. The chain rule of differentiation gives very similar computational rules for the backward pass as the ones in the forward pass. The network weights can then be adapted using any gradient-based optimization algorithm. The whole process is iterated until the weights have converged.

A simple MLP could look like shown in Fig 2.3 as explained in [17].

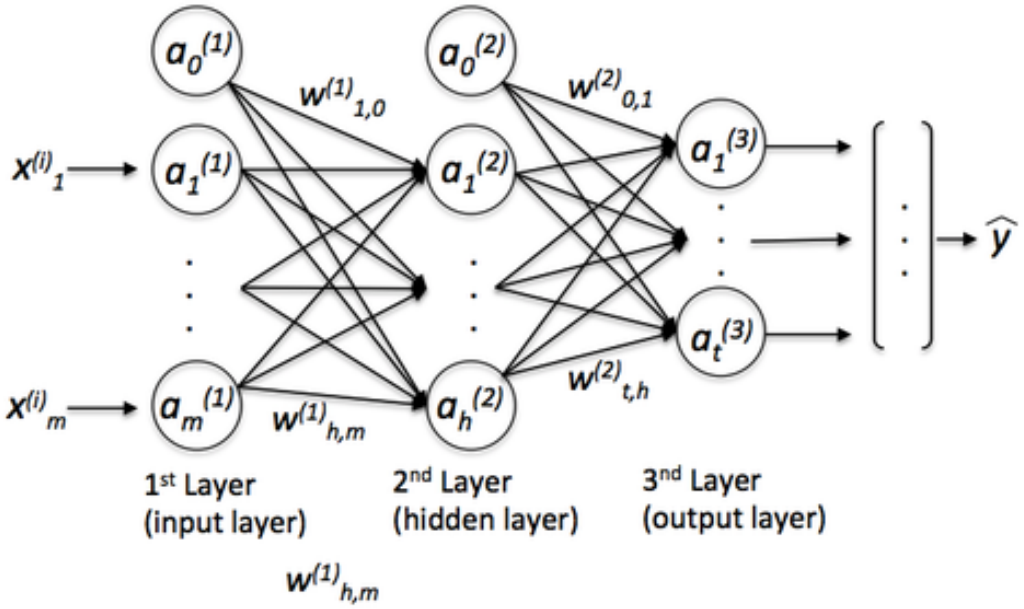


Fig. 2.3: A generic Multilayer Perceptron with one hidden layer

2.1.2 Limitations of MLP

If multiple hidden layers are added to a MLP, it will still be a deep network, but the problem with such networks is that it becomes tougher and tougher to learn good weights for this network. Initially while starting training of this network, typically random values are assigned as initial weights, which can be way off from the optimal solution but during training, back-propagation

algorithm is used. Now, the problem with deep MLP networks is the vanishing gradient [18, 19], the more layers are added, the harder it becomes to update the weights because the signal becomes weaker and weaker. Since the networks weights can be terribly off in the beginning due to random initialization, it can become almost impossible to parameterize a deep MLP with back-propagation.

2.1.3 Convolutional Neural Network

ConvNets are a derivative of standard MLP neural networks optimized for two-dimensional pattern recognition problems. Instead of using fully connected hidden layers as described in the preceding section, the ConvNet introduces a special network structure, which consists of convolution and sub-sampling layers. Feature maps generated by convolution layers, contain neurons that take their synaptic inputs from a local receptive field. The weights of neurons within the same feature map are shared. It allows to have replicated units sharing the same configuration, thereby features can be detected regardless of their position in the visual field. Moreover, the fact that weights are shared increases learning efficiency by reducing the number of parameters being learned. In order to have a data reduction, a sub-sampling operation called pooling is performed. This data reduction operation is applied to the predecessor convolution result by a local averaging over a predefined window. It partitions the input image into a set of non-overlapping windows and then for each sub-region outputs the maximum value. This step is important because it helps to eliminate non-maximal values. The output layer ensures the classification of the input image. In this layer all neurons are fully connected and have a unique set of weights so they can detect complex features and perform classification. Apart from input and output layer, what makes ConvNets different from MLPs are Convolutional Layer, Pooling Layer, and Fully-Connected Layer. We will describe each layer and its working in details in below sections.

Convolutional Layer

The main task of the convolutional layer is to detect local conjunctions of features from the previous layer and mapping their appearance to a feature map. Figure 2.4 shows the feature map of an images while performing convolution. As a result of convolution in neuronal networks, the image is split into perceptrons, creating local receptive fields and finally compressing the perceptrons in

feature maps of size m_2 x m_3.

Thus, this map stores the information where the feature occurs in the image and how well it corresponds to the filter. Hence, each filter is trained spatial in regard to the position in the volume it is applied to.

In each layer, there is a bank of m_1 filters. The number of how many filters are applied in one stage is equivalent to the depth of the volume of output feature maps. Each filter detects a particular feature at every location on the input.

The output $Y_i^{(l)}$ of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l)} m_2^{(l)}$. The i^{th} feature map, denoted $Y_i^{(l)}$, is computed as.

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)}$$

The result of staging these convolutional layers in conjunction with the following layers is that the information of the image is classified like in vision. That means that the pixels are assembled into edglets, edglets into motifs, motifs into parts, parts into objects, and objects into scenes [20].

ReLU

The rectified linear units (ReLUs) are a special implementation that combines non-linearity and rectification layers in convolutional neural networks. A rectified linear unit (i.e. thresholding at zero) is a piecewise linear function defined as:

$$Y_i^{(l)} = max(0, Y_i^{(l-1)})$$

Pooling Layer

The pooling or downsampling layer is responsible for reducing the spacial size of the activation maps. In general, they are used after multiple stages of other layers (i.e. convolutional and non-linearity layers) in order to reduce the computational requirements progressively through the network as well as minimizing the likelihood of overfitting. The pooling layer l has two hyperparameters, the spatial extent of the filter $F^{(l)}$ and the stride $S^{(l)}$. It takes an input volume of size

$m_1^{(l-1)}$ x $m_2^{(l-1)}$ x $m_3^{(l-1)}$ and provides the volume of size $m_1^{(l)}$ x $m_2^{(l)}$ x $m_3^{(l)}$ where;

$$m_1^{(l)} = m_1^{(l-1)}$$

$$m_2^{(l)} = (m_2^{(l-1)} - F^{(l)})/S^{(l)}) + 1$$

$$m_3^{(l)} = (m_3^{(l-1)} - F^{(l)})/S^{(l)}) + 1$$

The key concept of the pooling layer is to provide translational invariance since particularly in image recognition tasks, the feature detection is more important compared to the feature's exact location. Therefore the pooling operation aims to preserve the detected features in a smaller representation and does so, by discarding less significant data at the cost of spatial resolution.

The pooling layer operates by defining a window of size $F^{(l)}F^{(l)}$ and reducing the data within this window to a single value. The window is moved by $S^{(l)}$ positions after each operation similarly to the convolutional layer and the reduction is repeated at each position of the window until the entire activation volume is spatially reduced.

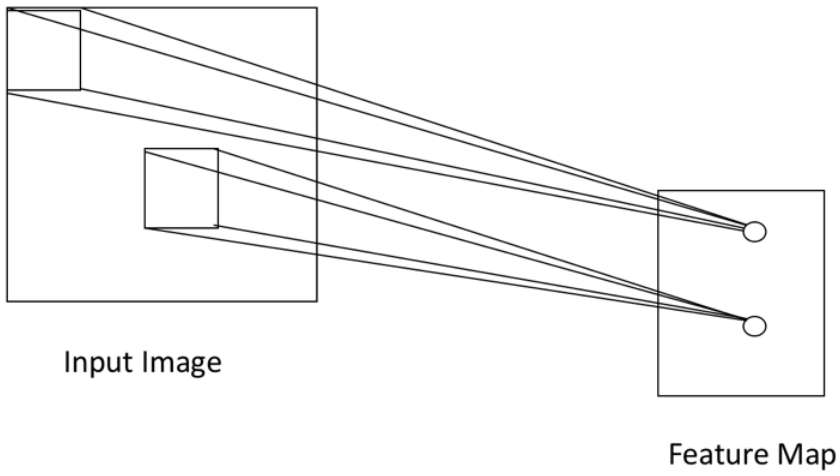


Fig. 2.4: Feature Map on an image while performing convolutions

Dropout

At each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

Fully Connected Layer

The fully connected layers in a ConvNet are practically a multilayer perceptron (generally a two or three layer MLP) that aims to map the $m_1^{(l-1)}$ x $m_2^{(l-1)}$ x $m_3^{(l-1)}$ activation volume from the combination of previous different layers into a class probability distribution. Thus, the output layer of the multilayer perceptron will have $m_1^{(l-i)}$ outputs, i.e. output neurons where i denotes the number of layers in the multilayer perceptron.

The key difference from a standard multilayer perceptron is the input layer where instead of a vector, an activation volume is taken as the input. As a result the fully connected layer is defined as:

If l-1 is a fully connected layer

$$Y_i^{(l)} = f(Z_i^{(l)}) = \sum_{j=1}^{m_1^{(l-)}} w_{i,j}^{(l)} y_i^{(l-1)}$$

otherwise,

$$Y_i^{(l)} = f(Z_i^{(l)}) = \sum_{j=1}^{m_1^{(l-)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} (Y_i^{(l-1)})_{r,s}$$

The goal of the complete fully connected structure is to tune the weight parameters $w_{i,j}^{(l)}$ or $w_{i,j,r,s}^{(l)}$ to create a stochastic likelihood representation of each class based on the activation maps generated by the concatenation of convolutional, non-linearity, rectification and pooling layers. Individual fully connected layers operate identically to the layers of the multilayer perceptron with the only exception being the input layer.

Figure 2.5 shows the internal architecture of a ConvNet depicting convolutional layer that shows the convolution on an input image resulting into feature maps. Next, pooling layer down-samples it into further smaller feature maps which is then sent to fully connected MLP. The output is sent to the classifier which classifies the output into respective categories.

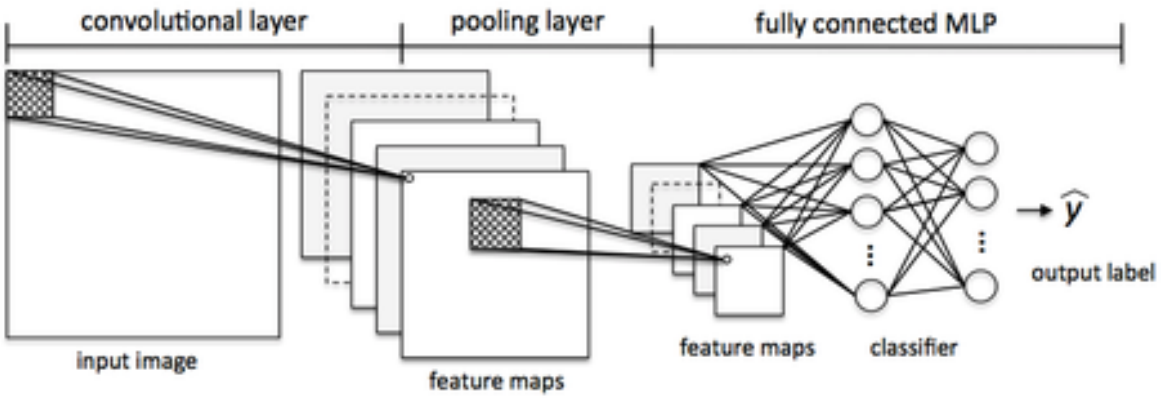


Fig. 2.5: Convolutional Neural Network Architecture

CHAPTER 3

Bee Classification Using Deep Learning

3.1 Overview

To monitor forager traffic in videos, Computer Vision based techniques using DL are developed that can recognize individual bees in videos to automate bee traffic assessment and, eventually, forager traffic assessment.

This section will cover the description of our Electronic Beehive Monitoring System (BeePi), data collection, data preprocessing, and the algorithms for ultimate bee recognition in real time.

3.2 BeePi and Data Collection

For continuous integration with the BeePi system, I chose to work with the Raspberry Pi 3 that is already part of BeePi. The pi-3 has 1Gb of RAM and 4 cores that are more than capable of processing videos with ConvNets. The data used for training and testing the Neural Network models is taken from the videos collected by the solar-powered, EBM system, called BeePi [21]. The BeePi is designed for the Langsthroth hive used by many beekeepers worldwide. Four BeePi EBM systems were assembled and deployed at two Northern Utah apiaries to collect 27.0 GB of audio, temperature, and video data in different weather conditions.

.A fundamental objective of the BeePi design is reproducibility: other researchers and practitioners should be able to replicate our results at minimum cost and time commitments. Each BeePi consists of a Raspberry Pi computer, a miniature camera, a solar panel, a temperature sensor, a battery, a hardware clock, and a solar charge controller. Figure 3.1 shows a picture of the BeePi system.

3.3 Data Preprocessing for Neural Network

Every video is composed of 749 frames of size (360 x 240). To get 32x32 size images from these frames, I have applied random sampling to pick 400 frames from the video. Every such frame has

Fig. 3.1: Components of BeePi

been again subjected to random sampling to pick start and end coordinates randomly to generate 2500-3000 images per frame. This results in 3980, 32x32 images from one video.

After all the data is prepared into 32x32 images, we were able to generate a total of 54000 images. This data then was manually cleaned and labeled into two categories: BEE and NO-BEE. Figure 3.2 shows the sample images from bee category and 3.3 shows the sample images from the no-bee category.

Figure 3.4 describes a quick pictorial representation of data preprocessing and bee classification process. From a mp4 video 360x240 size frames are extracted which are further processed to get 32x32 size of images. These images then manually labeled into bee and no-bee category to train the neural network models which will then classify if an image has a bee in it or not.

For training the Neural network we used data taken from Craig's East and West hives which is 50,000 images in total. For testing the performance of the model, we used separate data taken from Richard's east and West hives, which is 4000 images in total.

Fig. 3.2: Sample images from bee category used for training and testing

Fig. 3.3: Sample images from no-bee category used for training and testing

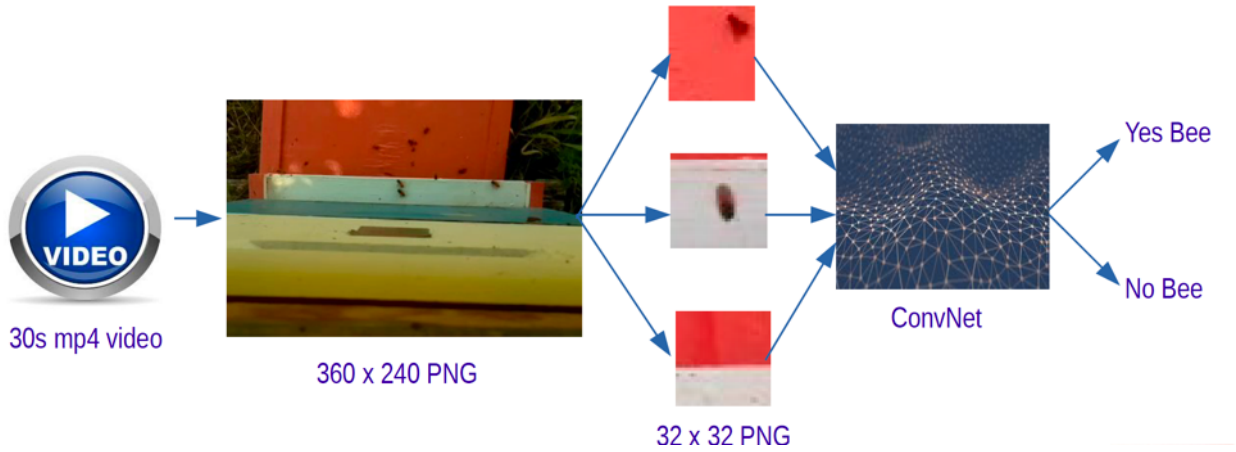


Fig. 3.4: Data Preprocessing and bee-classification process

3.4 Algorithm

Figure 3.5 describes the flowchart of the algorithm used finally on the Raspberry Pi. It is working with a motion detection algorithm. The input to the algorithm is a 30s mp4 video from a hive taken with the help of BeePi. This video is fed to the algorithm and the motion detection algorithm checks for any motion in the video. If the motion detection algorithm doesn't encounter any motion in the video then the algorithm terminates after a complete video is processed. If it encounters any motion, it draws a contour where the motion is detected and crops a 32x32 size image across it. That 32x32 image is fed into the neural network to check whether the detected motion image has a bee in it or not and then classify it accordingly into a bee or no-bee category.

3.5 Bee Recognition DL Models

With the knowledge of MLP and ConvNets as described in earlier sections, we designed various architectures by feeding our labeled data into the neural networks for training and testing the performance of each model for comparison. The details of all the models and the experiments are described in the next section. All this training has been performed on a PC with Ubuntu 14.04 and 14GB RAM. These models have been tested on the PC and the best 4 models were picked and tested on the Raspberry Pi.

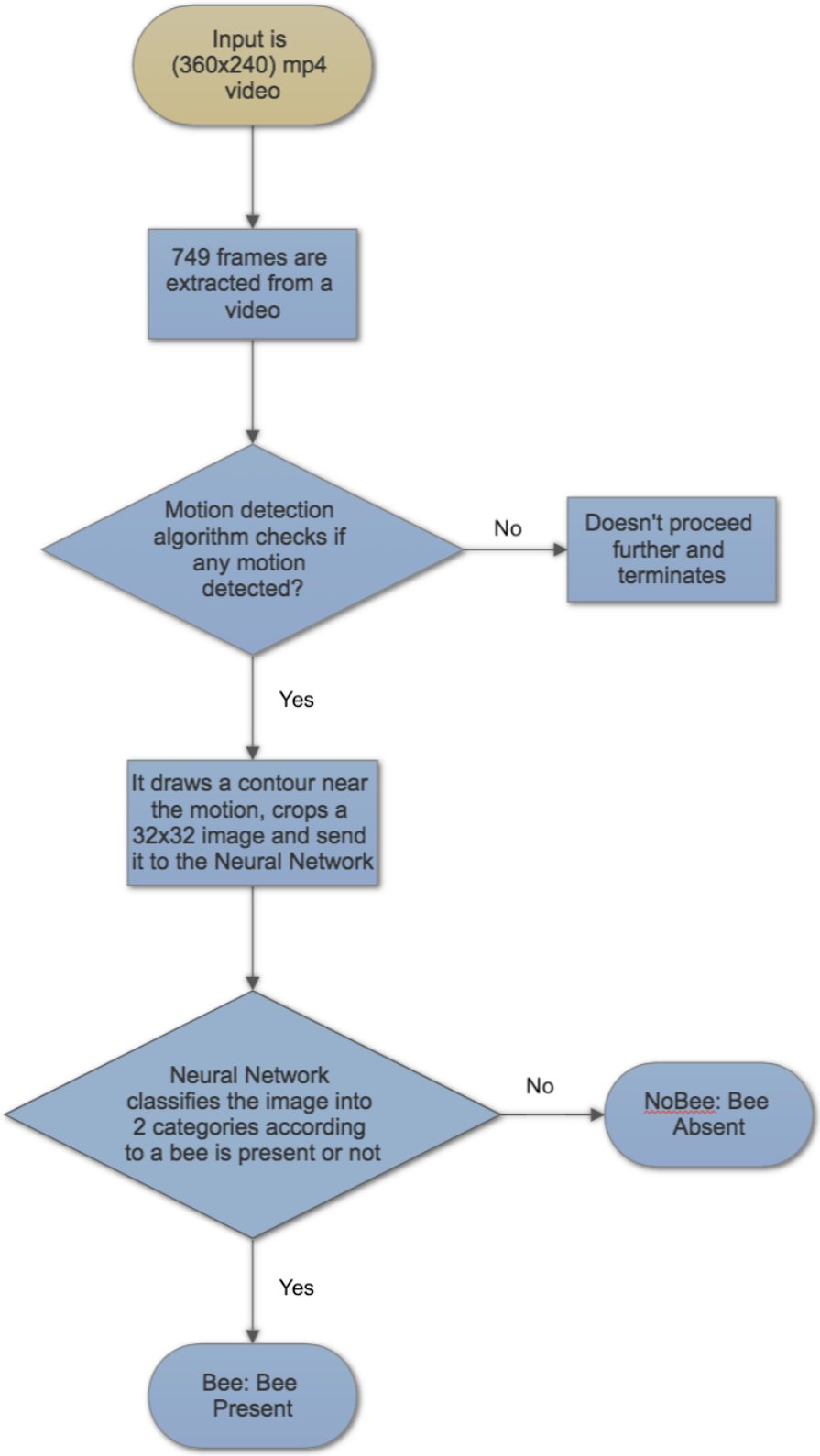


Fig. 3.5: Algorithm for the bee classifier using motion detection a CNN on a Raspberry Pi

CHAPTER 4

Experiments and Results

The input to our neural network models consists of images of 3 order tensor, (an image with 32 rows and 32 columns and 3 color channels (R, G and B)). For better numerical stability, instead of representing data in [0, 255] range, the data is scaled down to [0, 1]. These tensors are then fed into neural network as 75% of the data for training and 25% for testing from 50000 images. 4000 images are used for validation data to see the accuracy of the model. These 4000 images are from the different hive and data from this hive is not used at all while training the model. Also, we added our false positives and false negative data collected from running experiments into this validation dataset to check the performance of the models. The accuracy and loss on training and testing is verified by using the model on our validation dataset of 4000 labeled images. It's divided into two categories of validation accuracy of bees and validation accuracy of no-bees.

4.1 Multilayer Perceptron Design

We started by training simplest type of DL models, MLP. We trained four different MLP networks and tuned them with various hyper-parameters. All the networks are trained for 100 epochs. Activation function REctified Linear Unit (RELU) is used on the hidden layer and Softmax function on the output layer with loss function as categorical-cross entropy. An efficient ADAM gradient descent algorithm is used to learn the weights. Also, a dropout of 0.5 is used after every fully-connected hidden layer. The results from the output layer are probability-like values and allow one class of the 2 to be selected as the models output prediction.

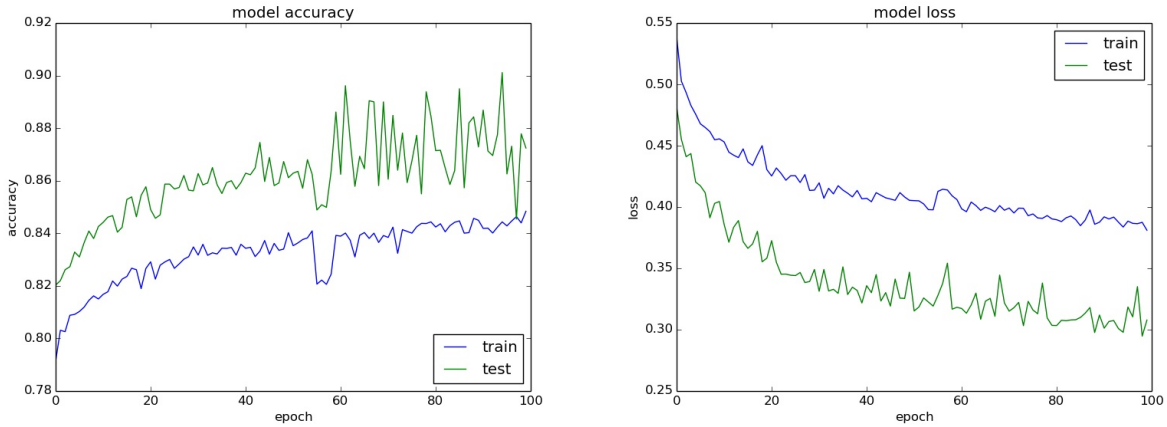
The first network consists of one hidden layer with the 64 number of neurons. To see the effect on performance by changing the number of neurons in hidden layer the second network is trained with 512 number of neurons. Similarly, the other two models are trained by changing the architecture including 2 hidden layers of 64 neurons each and 512 neurons each respectively.

Figure 4.1 shows the results of the accuracy of training and testing after every 10 epochs and

table 4.1 gives the final accuracy and loss of training, testing and validation for all models. The results of validating the models on the validation dataset shows that all the MLP models gave a very poor accuracy on identifying bee. MLP with one hidden layer and 64 neurons gave an accuracy of 35.34% on identifying bee, MLP with one hidden layer and 512 neurons increased it to 39.79% whereas MLP with two hidden layers having 64 neurons each increased it further to 41.74%. But, MLP with two hidden layers with 512 neurons each decreased it again to 29.78%. Accuracy in identifying no-bee images is still pretty high with all the four models and the maximum is achieved to be 83.30% on the MLP with two hidden layers and 512 neurons each. We can conclude that for pattern recognition in bee images, MLP didn't perform very well.

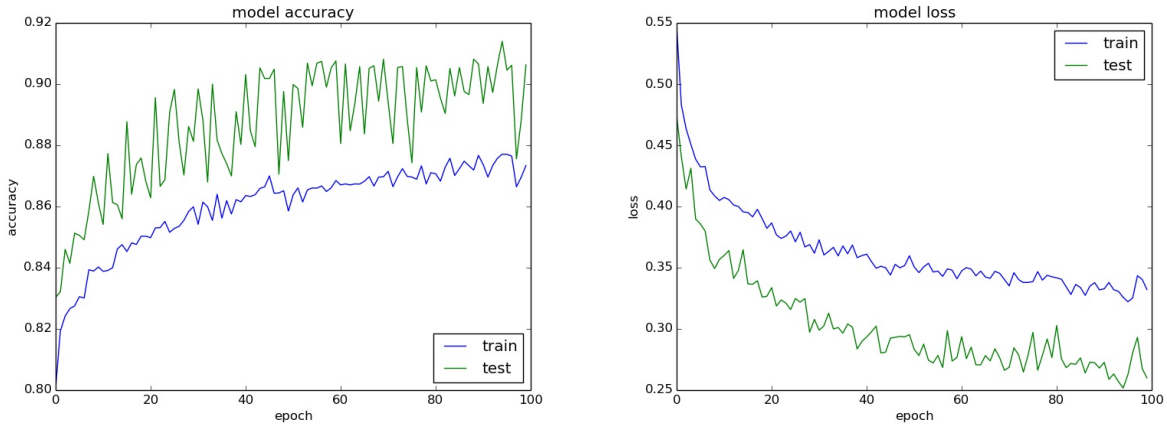
Table 4.1: Different MLP Results

Network	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
1 hidden layer with 64 neurons	84.84%	0.3810	87.24%	0.3078	35.34 %	79.56%
2 hidden layers with 64 neurons	83.78%	0.4421	85.06%	0.3957	41.74 %	73.83%
1 hidden layer with 512 neurons	87.34%	0.3322	90.63%	0.2600	39.79 %	80.17%
2 hidden layers with 512 neurons	87.40%	0.3562	86.47%	0.3291	29.78 %	83.30%



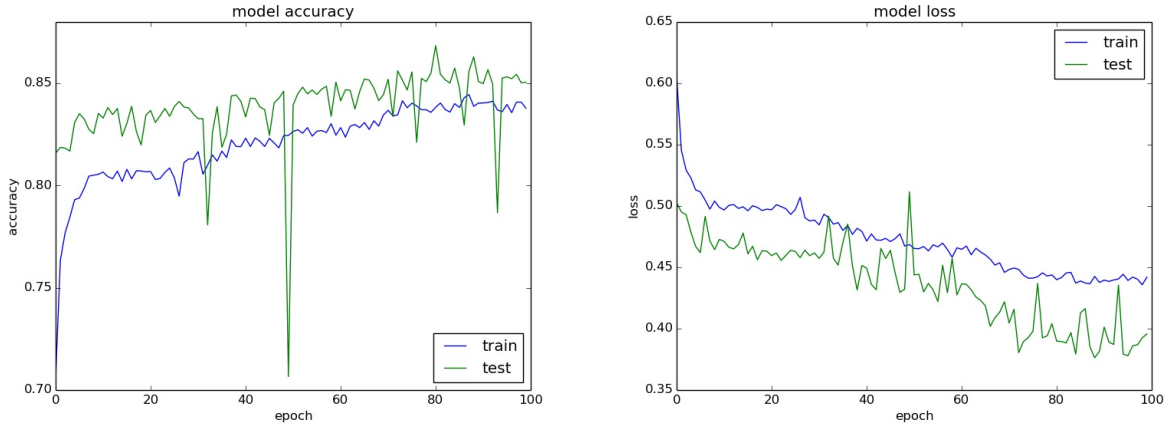
(a) Accuracy (b) Loss

Fig. 4.1: MLP: One hidden layer with 64 neurons



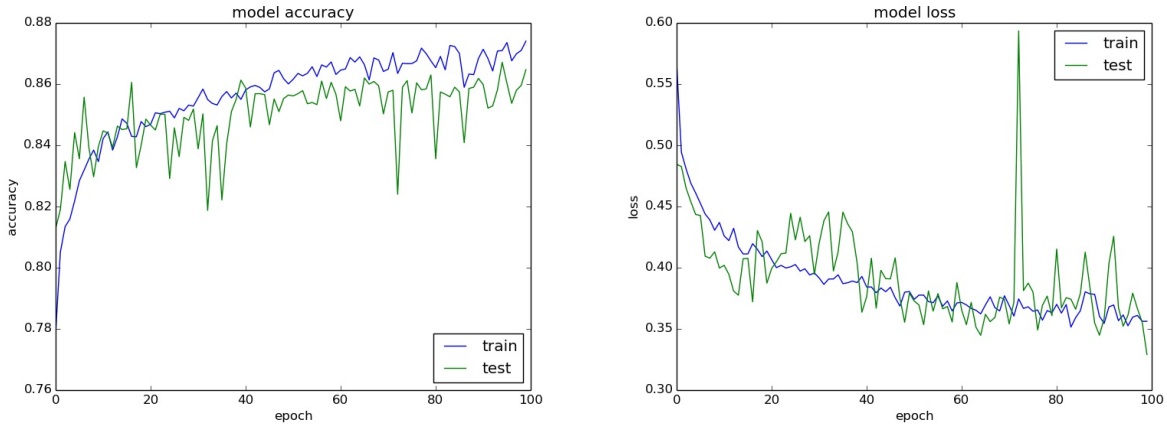
(a) Accuracy (b) Loss

Fig. 4.2: MLP: One hidden layer with 512 neurons



(a) Accuracy (b) Loss

Fig. 4.3: MLP: Two hidden layers with 64 neurons



(a) Accuracy (b) Loss

Fig. 4.4: MLP: Two hidden layers with 512 neurons

4.2 Convolution Neural Network Design

We designed ConvNet models to classify the images in our dataset using the basic CNN architecture model that is tuned by changing parameter and hyper parameter values. The models are categorized as shallow, deep and deeper models, named according to the complexity of the models. Architecture of these models are described in below sections.

4.2.1 Shallow Network

The input layer is same as described above for MLP. The first hidden layer is a convolutional layer called a Convolution2D. The layer has 32 feature maps, with the size of 3x3 and the stride of size 1 and RELU activation function. We added a flatten layer before the dense layer to convert the output of the convolutional part of the ConvNet into a 1D feature vector, to be used by the dense layer. The output layer is a fully-connected layer with 2 neurons and uses softmax activation function. Finally, the output layer has 2 neurons for the 2 classes and a softmax activation function to output probability-like predictions for each class. The model is trained using logarithmic loss and experimented with 3 optimizers; ADAM, RMSprop, Stochastic Gradient Descent. The ConvNet is fit over various epochs with a batch size of 50. The second network then is modified by adding a pooling layer after the Convolution2D layer, called MaxPooling2D which is configured with a pool

size of 2x2. The third network is modified by adding a dense layer with 256 neurons to see the importance of a fully connected layer before the output layer. Above architecture is tuned further by changing the number of neurons in the dense layer, we experimented with 256, 512 and 1024 number of neurons. This layer uses RELU activation function. The fourth model is trained by adding a dropout layer after fully connected layer by using a dropout of 0.5.

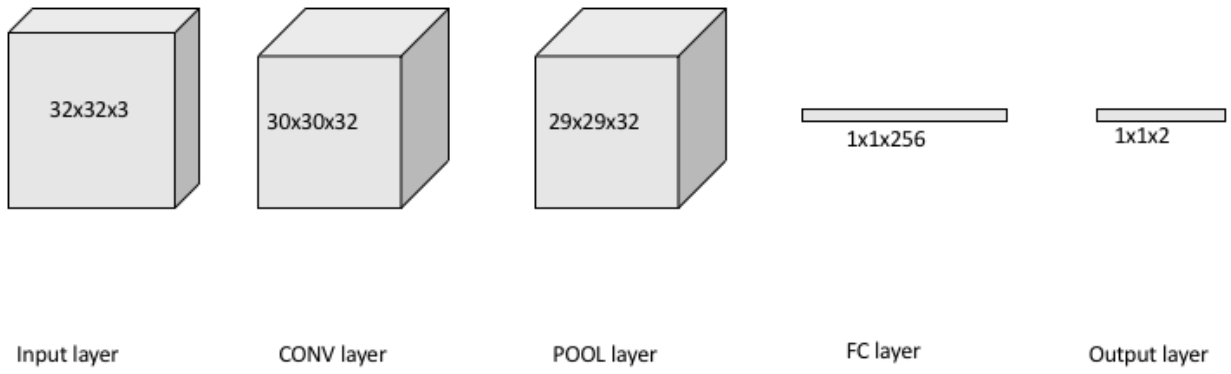
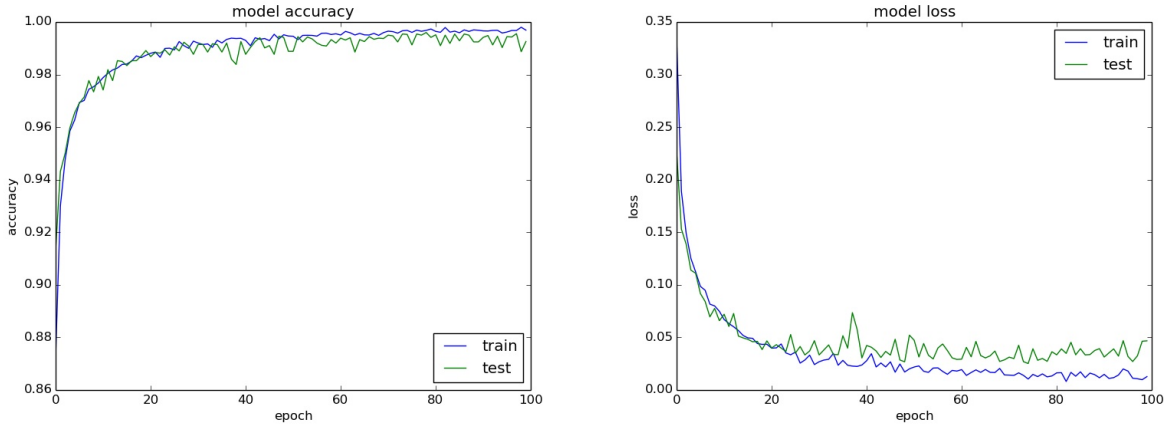


Fig. 4.5: Convolution layer with 1 CONV layer, POOL layer, a dense layer with 256 neurons and an output layer with 2 neurons

Figure 4.5 shows the final shallow network model with INPUT [32x32x3] will hold the raw pixel values of the image. CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [30x30x32] for 32 filters and kernel size 3x3. RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged [30x30x32]. POOL layer will perform a downsampling operation along the spatial dimensions resulting in volume such as [29x29x32]. A dense layer with 256, 512 and 1024 neurons in different networks. FC layer

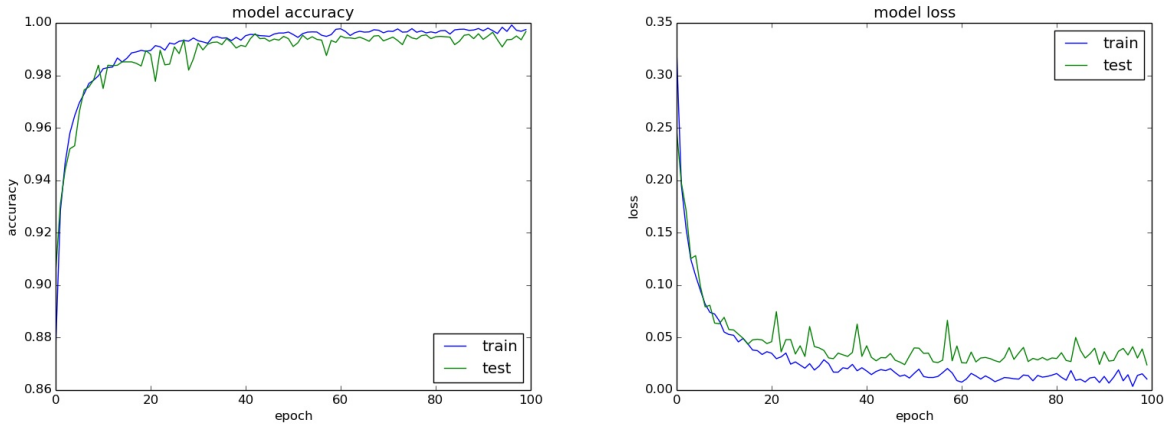
will compute the class scores, resulting in volume of size [1x1x2], where each of the 2 numbers correspond to a class score, such as bee and no-bee categories.

Figure 4.6 and figure 4.7 shows the results of the accuracy of training and testing after every 10 epochs for shallow networks with one CONV layer, one POOL layer with kernel size of 3x3 and 256 and 512 neurons in the FC layer respectively. Table 4.2 gives the final accuracy and loss of training, testing and validation for all the models starting with shallow network with only one CONV layer, one CONV layer with POOL layer, one CONV layer with POOL layer and FC layer having 256 or 512 neurons and one CONV layer with POOL layer and FC layer having 256 or 512 neurons with dropout layer added after FC layer with a dropout of 0.5. It can be observed that by adding a POOL layer after CONV layer there is a significant increase in accuracy of identifying bee on validation dataset. The accuracy increased from 58.08% to 77.08%. Further by adding a FC layer with 256 neurons increased the accuracy to 87.69%. After adding a dropout of 0.5 after FC layer improved the accuracy to 90.38%. Experiments with changing the number of neurons on the FC layer from 256 to 512 on the final shallow network improved accuracy to identify bee slightly to 90.82% but the accuracy for no-bee decreased from 64.12% to 63.96%. Also, there is no improvement noticed after increasing number of neurons to 1024. Hence, it can be concluded that considering the best accuracy in both categories of identifying bee and no-bee is achieved by the shallow ConvNet with 256 neurons and dropout.



(a) Accuracy (b) Loss

Fig. 4.6: One convolutional layer with maxpooling and dense layer with 256 neurons with dropout



(a) Accuracy (b) Loss

Fig. 4.7: One convolutional layer with maxpooling and dense layer with 512 neurons with dropout

Table 4.2: 1-convolutional Layer, different hyper-parameters

Network	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
One conv layer	99.72%	0.0126	97.23%	0.1410	58.08 %	55.11%
with maxpooling	99.79%	0.0085	98.49%	0.0655	77.68 %	36.09%
dense256	99.99%	0.0294	99.34%	0.0453	87.69 %	64.06%
dense256 dropout	99.69%	0.0127	99.26%	0.0468	90.38 %	64.12%
dense512	99.97%	0.0014	99.34%	0.0504	83.52%	64.78 %
dense512 with dropout	99.75%	0.0105	99.67%	0.0240	90.82 %	63.96%
dense1024	99.99%	0.0115	99.27%	0.0526	84.45 %	68.19%
dense1024 with dropout	99.97%	0.0014	99.34%	0.0504	84.23 %	61.04%

4.2.2 Different Optimizer Functions

The basic shallow ConvNet with 1 CONV layer, 1 POOL layer and a dense layer with 256 layers is experimented with 3 different optimizer functions, ADAM, RMSProp, SGD. Among these SGD has emerged as one of the most used training algorithms for deep neural networks. It's

very simple and yet performs great across a variety of applications but also has strong theoretical foundations [22]. But, the problem with SGD is that it scales the gradient uniformly in all directions which this can be a unfavorable for ill-scaled problems and also makes the process of tuning the learning rate circumstantially laborious. ADAM is a method for efficient stochastic optimization. It only requires first-order gradients and computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. That's how the name Adam is derived from adaptive moment estimation [23]. RMSProp keeps a moving average of the squared gradient for each weight meaning it divided the learning rate by an exponentially decaying average of squared gradients [24]. The results are shown in table 4.3, the accuracy of validating these 3 models on our validation dataset shows that performance of SGD is the poorest in both categories with only 61.59% accuracy is identifying bee and 51.70% accuracy in classifying no-bee images. RMSProp worked better than SGD but ADAM performs best on our models achieving an accuracy of 90.82% and 64.96% in bee and no-bee categories respectively. So, rest of the models were trained using ADAM optimizer.

Table 4.3: 1-convolutional Layer, different hyper-parameters

Optimizer	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
SGD	99.43%	0.0321	99.01%	0.1023	61.59 %	51.70%
RMSProp	99.67%	0.0203	98.83.34%	0.0961	82.36%	60.11%
ADAM	99.75%	0.0105	99.67%	0.0240	90.82 %	63.96%

4.2.3 Deep Network

To increase the complexity I added more convolutional layers and FC layers to the network, A deeper ConvNet as shown in Figure 4.8, is trained with 2 convolutional layers and 1 FC layer. The first convolutional layer has 32 3x3 filters, the second one has 64 3x3 filters. In all the convolutional layers, we have a stride of size 1, dropout, max-pooling and ReLU as the activation function. The

hidden layer in the FC layer has 512 neurons. In FC layer, same as in the convolutional layers, ReLU is used. Rest of the network is same as shallow ConvNet model.

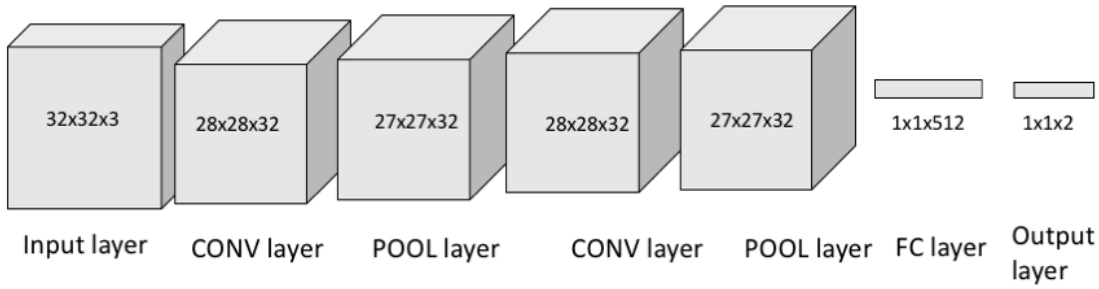
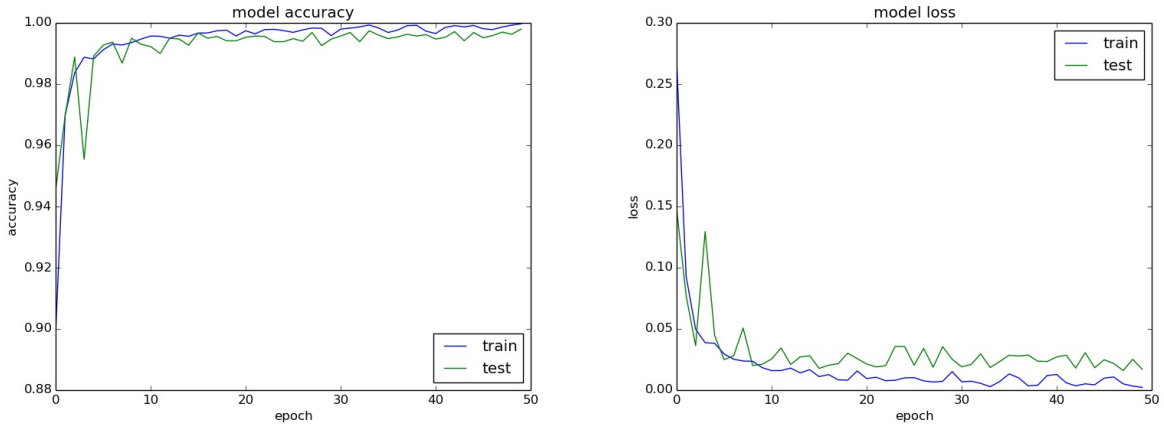


Fig. 4.8: Convolution layer with 2 CONV layer, 2 POOL layer, a dense layer with 512 neurons and an output layer with 2 neurons



(a) Accuracy (b) Loss

Fig. 4.9: Deep Network: Two convolutional layer with maxpooling and dense layer with 512 neurons with dropout

Figure 4.9 shows the results of the accuracy of training and testing after every 10 epochs for deep model described above. Table 4.4 gives the final accuracy and loss of training, testing and validation for all the models starting with deep network with just two CONV layers, then adding

two POOL layers after each CONV layer, further adding a FC layer having 512 neurons and finally adding a dropout of 0.5. It can be noticed that the performance of the model improves as we add POOL, FC and dropout layers and the highest accuracy is achieved with the final Deep model which is 96.42% and 82.19% for bee and no-bee categories respectively.

Table 4.4: Deep CNN with 2 convolutional layer with different hyper parameters

Network	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
two conv layer	99.93%	0.0036	99.16%	0.0488	89.61 %	71.15%
with maxpooling	99.99%	0.1362	98.63%	0.0258	93.29 %	73.08%
dense512	99.96%	0.0016	99.66%	0.0205	95.71 %	82.53%
dense512 and dropout	99.96%	0.0016	99.66%	0.0205	96.42 %	82.19%

4.2.4 Different Kernel Size on Convlolutional Layers

To see the effect of changing the kernel (filter) size of the CONV layer, several experiments were performed on shallow and deep models. Table 4.5 shows the results of the experiments. Accuracy of shallow model with 256 neurons using 3x3 kernel size is 87.69% and 64.06% in bee and no-bee category respectively. The performance degraded to 84.07% and 61.48% in bee and no-bee category respectively by using 5x5 kernel size. Similarly for the case of shallow model with 512 neurons. However, for the deep model we got a better performance with 5x5 kernel size which is 97.91% and 83.46% in bee and no-bee categories respectively whereas accuracy by using 3x3 kernel size on deep model is 95.71% and 82.53% in bee and no-bee category respectively. Hence, it can not be concluded which kernel size works the best on a particular model.

Table 4.5: Effect of different filter size on 1-convolutional layer model

Network	Number of Neurons	Val Acc Bee	Val Acc NoBee
shallow k=3	256	87.69 %	64.06%
shallow k=5	256	84.07%	61.48%
shallow k=3	512	90.82 %	63.96%
shallow k=5	512	86.97%	62.66%
deep k=3	512	96.42%	82.19%
deep k=5	512	97.91%	83.46%

4.2.5 Deeper Network

To explore the deeper CNNs, we trained our network with 3 conv layer and 1 FC layers 4.10. The first convolutional layer had 32 5x5 filters, the second one had 32 5x5 filters and the third had 64 5x5 filters. In all the convolutional layers, we have a stride of size 1, dropout, max-pooling and ReLU as the activation function.The hidden layer in the FC layers has 512 neurons. In FC layer, same as in the convolutional layers, we used dropout and ReLU.

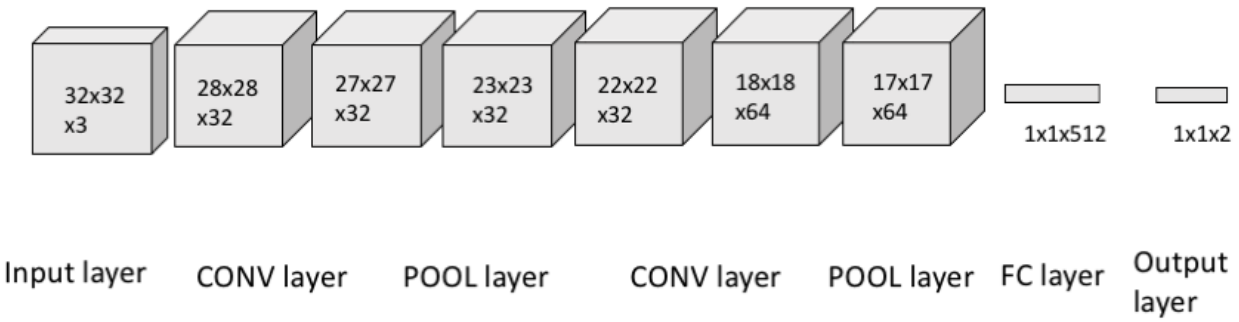


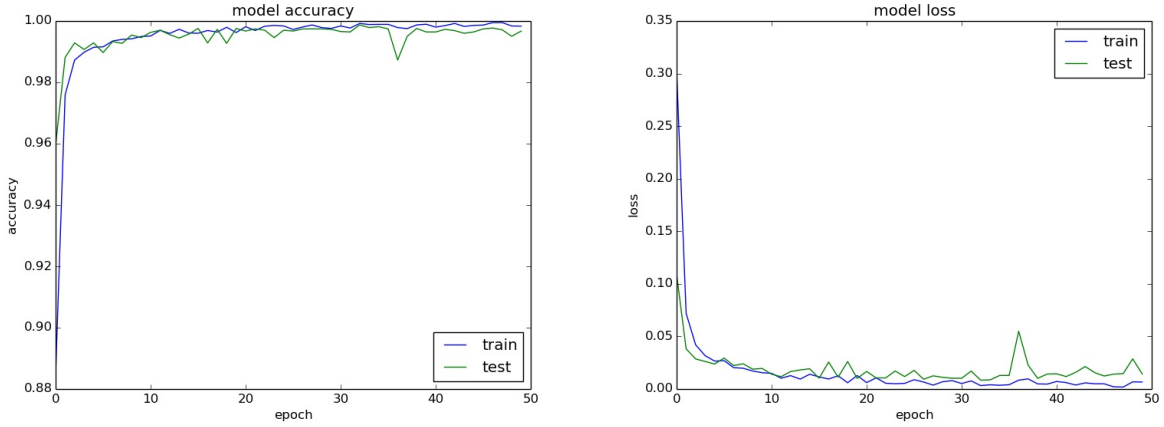
Fig. 4.10: Convolution layer with 3 CONV layer, 3 POOL layer, a dense layer with 512 neurons and an output layer with 2 neurons

Figure 4.11 shows the results of the accuracy of training and testing after every 10 epochs for deeper model described above. Table 4.7 gives the final accuracy and loss of training, testing and

performance of the model on validation dataset. The accuracy of the deeper model is is 97.58% and 83.84% for bee and no-bee categories respectively. As it can be observed that there is no noticeable improvement from deep to deeper network in terms of performance of the model on validation dataset. Different architectures were trained to observe the performance but the accuracy became stagnant after deep model.

Table 4.6: Deeper model results

Network	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
3 conv layer 1 FC layer	99.83%	0.0066	99.67%	0.0143	97.58 %	83.84%
4 conv layer 2 FC layers	99.99%	0.1362	98.63%	0.0258	97.46 %	80.09%



(a) Accuracy (b) Loss

Fig. 4.11: Deeper Network: Three convolutional layer, 1 FC layer

4.3 Effect of Dataset

To see the effect of increasing the number of images for training the neural network we experimented by having 3 different datasets with the deep model with 2 CONV layers and 1 FC layer.

The first CONV layer has 32 5x5 filters, the second one has 64 5x5 filters. FC layer has 512 neurons. This model is same as described in section 4.2.4. With this model 3 experiments were performed.

The first experiment has 10,000 images with 5000 images in bee category and 5000 images in no-bee category. The data from each category is split into training and testing set with 75% images in training set and 25% images in test set. The second experiment has 28,000 images in total. And, the third experiment has 50,000 images in total.

Table 4.7: Deeper model results

Network	Train Acc	Train Loss	Test Acc	Test Loss	Val Acc Bee	Val Acc NoBee
10k images	99.91%	0.0029	99.40%	0.0205	97.41 %	83.29%
28k images	99.83%	0.0066	99.67%	0.0143	97.58 %	83.84%
50k images	99.90%	0.0054	99.61%	0.0307	96.48 %	85.65%

4.4 Experiments on Raspberry Pi

After analyzing the performance of models on our validation dataset, we picked the best 2 models and used them with our motion detection algorithm on the Raspberry Pi. On the Raspberry Pi, 3 different videos are experimented on each model and the results of those models are manually analyzed. 2950 images were collected by the motion detection algorithm from 3 videos. Model 1 is the simplest shallow model with kernel size on CONV layer as 3x3 and 256 neurons in the FC layer. Model 2 is a deeper model with kernel size on CONV layer as 3x3 and 512 neurons in the FC layer.

As we can see in the table 4.8, both of our models performed really well on the raspberry pi experiments and since the input data to the neural network is mostly a detected motion image, so there is high probability of having bee images and our models give a good performance in classifying the images that has bees in bee category and images without bees in no-bee category.

Table 4.8: Results of testing models on Raspberry Pi

Network	Video	True Bee	False Bee	True No-Bee	False No-Bee
model1	1	927	7	20	15
model1	2	948	48	378	7
model1	3	428	4	157	11
model2	1	933	1	21	14
model2	2	990	6	382	3
model2	3	429	3	156	12

CHAPTER 5

Conclusion and Future Work

For forager traffic analysis, this thesis investigates various methods for object recognition. Taking advantage of DL to work with images without extracting features manually unlike in traditional ML methods, this thesis analyzes various MLP and ConvNet models and pick the best 2 to experiment further on raspberry pi to work in real time.

To recognize bees, different ANN models have been trained with a manually labeled dataset of 50,000 images and tuned to improve performance. The trained models were then tested on a validation dataset of 4000 images and best picked 2 models from there were tested on Raspberry Pi on 3 different videos.

From the experiments and results present in Chapter 4, several conclusions can be drawn. First, the simple MLP performed really poor as compared with a simple CNN model on our dataset. Second, all the models seem to stabilize in terms of accuracy and loss around 40-50 epochs, that's why after shallow models, other models were trained for 50 epochs. Third, the performance with ADAM optimizer turned to be better that the others. Fourth, with kernel size as 5 on convolutional layers, the models performed better. Fifth, number of neurons in the fully connected layer seemed to have an effect on performance, and in our case 512 worked the best. Increasing numbers of hidden layers also had an effect on performance, deep model perform better than shallow and deeper model perform better than deep. But, after that there was no improvement in performance.

Undoubtedly with the help of ConvNets, an accuracy of 97 % is achieved on our dataset but there are several classification problems with ConvNets observed after analyzing the results. One of them was classifying images with shadows as bees due to very slight difference between an image with bee's shadow and a real bee. Also, in few cases when a bird's part was captured in a 32x32 image it looked very much like a bee and ConvNet classified it as a bee. These were the reasons for dropped accuracy of NoBee category in most of the results.

In our future work, we propose to improve the performance of the models and working on

the shortcoming of the present models, like detecting shadows and bird's tail as bees. To solve the shadow problem in current models, more shadow data can be included and then new models can be trained with 3 classes i.e. bee, no-bee and shadow. Overall performance of the models can be improved by adding more data to the training dataset. Also, the quality of the data can be improved by using high resolution videos. Another area of improvement can be experimenting with different architectures and hyper parameters, such as using various other activation functions on each layer and going for deeper architectures, or defining custom deep learning layers instead of using CONV layers. There is a scope of improvement in motion detection algorithm also that can help in capturing a whole bee in a 32x32 image for better image recognition.

REFERENCES

[1] [Online]. Available: https://www.usda.gov/media/press-releases/2016/05/12/usda-releases-results-new-survey-honey-bee-colony-health

[2] S. G. Potts, S. P. M. Roberts, R. Dean, G. Marris, M. A. Brown, R. Jones, P. Neumann, and J. Settele, "Declines of managed honey bees and beekeepers in europe," *Journal of Apicultural Research*, vol. 49, no. 1, pp. 15–22, 2010. [Online]. Available: https://doi.org/10.3896/IBRA.1.49.1.02

[3] A. R. Mclellan, "Honeybee colony weight as an index of honey production and nectar flow: A critical evaluation.SNAP-1," 1977, pp. 401–408.

[4] D. A. K. L. D. J. Pham-Delegue, M.-H., "Behavioural methods to assess the effects of pesticides on honey bees," in *Apidologie,33*, 2002.

[5] M. Sanford, "2nd international workshop on hive and bee monitoring," *American Bee Journal*, pp. 1351–1353, 2014.

[6] K. Shah, "Power analysis of continuous data capture in beepi, a solar- powered multi-sensor electronic beehive monitoring system for langstroth beehives," 2017. [Online]. Available: https://digitalcommons.usu.edu/etd/6507

[7] V. A. Kulyukin and S. K. Reka, "Toward sustainable electronic beehive monitoring: Algorithms for omnidirectional bee counting from images and harmonic analysis of buzzing signals." *Engineering Letters*, vol. 24, no. 3, 2016.

[8] V. A. Kulyukin, "In situ omnidirectional vision-based bee counting using 1d haar wavelet spikes," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2017.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

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

[11] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[12] P. Y. Simard, D. Steinkraus, J. C. Platt *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[13] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature Publishing Group, a division of Macmillan Publishers Limited.*, 2015.

[15] J. J. Hopfield, "Artificial neural networks," *IEEE Circuits and Devices Magazine*, vol. 4, no. 5, pp. 3–10, Sept 1988.

[16] M. Minsky, S. A. Papert, and L. Bottou, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[17] "Back propogation algorithm."

[18] G. E. Hinton, "To recognize shapes, first learn to generate images," *Progress in brain research*, vol. 165, pp. 535–547, 2007.

[19] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[20] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 253–256.

[21] V. Kulyukin, M. Putnam, and S. Kiran Reka, "Digitizing buzzing signals into a440 piano note sequences and estimating forage traffic levels from images in solar-powered, electronic beehive monitoring." 03 2016.

[22] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.