**Insect Detection in Chilli/Cotton crop using**

**convolutional neural network (CNN)**

**A Project Report Submitted**

**in Partial Fulfillment of the Requirements for the award of the Degree**

**Bachelor of Technology**

in

**Computer Science and Engineering**

by

**V. Rahul Krishna** 18761A0555

**B. V. Anusha** 18761A0505

**A. Ajay** 18761A0501

**K. Yohan Siddartha** 18761A0530

*under the guidance of*

**Mr. T. Udaya Kumar**

**Sr. Assistant Professor**



**Department of Computer Science and Engineering Lakireddy Bali Reddy College of Engineering (Autonomous)**

**Accredited by NAAC & NBA(Under Tier-1) and ISO 9001:2015 Certified Institution Approved by AICTE, New Delhi & Aﬀiliated to JNTUK, Kakinada LBReddy Nagar, Mylavaram, Krishna(Dt.), Andhra Pradesh, India-521230**

**2017 – 2021**

**Lakireddy Bali reddy College of Engineering (Autonomous)**

**Accredited by NAAC & NBA(Under Tier-1) and ISO 9001:2015 Certified Institution Approved by AICTE, New Delhi & Aﬀiliated to JNTUK, Kakinada LBReddy Nagar, Mylavaram, Krishna(Dt.), Andhra Pradesh, India-521230**

**2017 – 2021**

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the project entitled “**Insect Detection in Chilli/Cotton crop using**

**convolutional neural network (CNN)**” is a bonafide work of

**V. Rahul Krishna 18761A0555**

**B. V. Anusha 18761A0505**

**A. Ajay 18761A0501**

**K. Yohan Siddartha 18761A0530**

in partial fulfillment of the requirements for the award of degree of **B. Tech.** in **Computer Science Engineering** from **Jawaharlal Nehru Technological University Kakinada** is a record of bonafide work carried out by them at **Lakireddy Bali Reddy College of Engineering(A).**

The results embodied in this have not been submitted elsewhere for a degree or diploma.

Supervisor Head of the Department

**Mr. T. Udaya Kumar Dr. D. Veeraiah**

EXTERNAL EXAMINER

**Declaration**

We are here by declaring that the project entitled “**Insect Detection in Chilli/Cotton crop using convolutional neural network (CNN)**” work done by us. We certify that the work contained in the report is original and has been done by me under the guidance of supervisor. The work has not been submitted to any other institute in preparing for any degree or diploma. We have followed the guidelines provided by the institute in preparing the report. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the institute. Whenever we have used materials (data, theoretical analysis, figures and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references. Further, we have taken permission from the copyrights owner of the sources, whenever necessary.

Name of the Student Signature of the student

**V. Rahul Krishna (18761A0555)**

**B. V. Anusha (18761A0505)**

**A. Ajay (18761A0501)**

**K. Yohan Siddhartha (18761A0530)**

**Acknowledgements**

We take great pleasure to express our deep sense of gratitude to our project guide **Mr. T. Udaya Kumar**, Associate Professor, for his valuable guidance during the course of our project work. We would like to thank **Dr. D. Veeraiah**, Professor & Head of the Department of Computer Science & Engineering for his encouragement. We would like to express our heart-felt thanks to **Dr. K. Appa Rao**, Principal, Lakireddy Bali Reddy College of Engineering for providing all the facilities for our project. Our utmost thanks to all the Faculty members and Non Teaching Staff of the Department of Computer Science & Engineering for their support throughout our project work. Our Family Members and Friends receive our deepest gratitude and love for their support throughout our academic year.

**V. Rahul Krishna 18761A0555**

**B. V. Anusha 18761A0505**

**A. Ajay 18761A0501**

**K. Yohan Siddartha 18761A0530**

**Abstract**

Insects are major trouble makers for Farmers as it affects quality of crops. Most of the Farmers are not aware of standard insecticides suitable for each type of Insect. That’s why insecticides or pesticides are not working effectively every time. This study aims to classify and detect the insects in cotton and chilli crops using machine learning and insect detection algorithm at the early stage of crop growth. This project uses local dataset of three classes of insects called ‘Ladha’, ‘Black pest’ and ‘leaf’ in local language Telugu. This project suggests a mobile application which uses a camera installed in the field, which will capture random images from the crop field every day in the morning, and will process those images using computer vision algorithms and modern machine learning techniques to detect the insects for suggesting suitable insecticide to the farmer.

**Contents**

1. **Introduction**……………………………………………………………………..1
2. **Review of Prior Work** ………………………………………………………….2
   1. Insect Detection using Machine Learning Algorithms……………………….2
   2. Conclusion…………………………………………………………………....3
3. **Related Work**……………………………………………………………………4
   1. Artificial Neural Networks…………………………………………………...4
      1. Training………………………………………………………………4
      2. Models………………………………………………………………..4
      3. Artificial Neurons…………………………………………………….5
      4. Organization………………………………………………………….5
      5. Hyperparameter…..…………………………………………………..5
      6. Learning………………………………………………………………5
      7. Learning rate………………………………………………………….6
      8. Cost Function…………………………………………………………6
      9. Back Propagation……………………………………………………..6
      10. Types………………………………………………………………….6
   2. Convolutional Neural Networks………………………………………………7
      1. Definition……………………………………………………………...7
      2. Architecture……………………………………………………………7
      3. Convolutional layers…………………………………………………...8
      4. Pooling layers………………………………………………………….8
      5. Fully Connected layers…………………………………………………8
      6. Receptive field………………………………………………………….8
      7. Weights…………………………………………………………………8
      8. Building Blocks…………………………………………………………9
      9. Explicit…………………………………………………………………14
      10. Materials and Methods ………………………………………………...15
4. **System Design**……………………………………………………………………...21
5. **Implementation Details and Results**……………………………………………...22
   1. Dataset Description……………………………………………………………..22
   2. Implementation Details…………………………………………………………23
   3. Implementation of Insect Detection………………………………………….23
   4. Work for Insect Classification……………………………………………….24
      1. TensorFlow………………………………………………………….24
      2. Keras…………………………………………………………………24
      3. Numpy……………………………………………………………….25
      4. Flask…………………………………………………………………25
   5. Code & Result……………………………………………………………….25
      1. Artificial Neural Network……………………………………………26
      2. Convolutional Neural Network………………………………………28
      3. Web App……………………………………………………………...31
   6. Results………………………………………………………………………..34
6. **Conclusion and Future Work**…………………………………………………..44
   1. Conclusion……………………………………………………………………44
   2. Future Work…………………………………………………………………..44

**Bibiliography**

**List of Figures**

3.1 Artificial Neural Networks ………………………………………………………….. 14

3.2 Convolutional Neural Networks …………………………………………………….. 17

3.3 Black Pest …………………………………………………………………………… 24

3.4 Ladha ………………………………………………………………………………... 24

3.5 Leaf ………………………………………………………………………………….. 24

3.6 Data Agumentation ………………………………………………………………….. 26

3.7 Flowchart of insect classification ……………………………………………………. 27

4.1 Block Diagram of the System ……………………………………………………….. 29

5.1 Ladha, Black pest, Leaf ……………………………………………………………… 31

5.2 Framework for insect classification ………………………………………………….. 32

5.3 Output ………………………………………………………………………………… 48

**List of Tables**

3.1 Dataset ……………………………………………………………………….. 23

5.1 Dataset ……………………………………………………………………….. 30

5.2 Classwise accuracy …………………………………………………………… 47

**List of Graphs**

5.1 Change in Accuracy of ANN model ……………………………………………………. 44

5.2 Change in accuracy of every epoch (CNN model) ……………………………………… 46

5.3 Classwise accuracy (CNN & ANN model) ……………………………………………... 47

5.4 Accuracy of CNN and ANN model ……………………………………………………...48

**Chapter 1**

**Introduction**

Agriculture, which is considered the backbone of the economy, contributes to the country’s economic growth and determines the standard of life. The agriculture and food processing industry is among the major sectors in any country and plays an essential role in expanding the export quality of agricultural and food products. In developing countries, the increase in food processing transformations is mainly due to the impact of export earnings and domestic market demands. In specific conditions, it requires storage, constant maintenance of equipment, and workspaces very frequently. Pest attack is one of the significant problems in the agriculture sector that results in degradation of crop quality. Pests, germs, and weeds cause massive loss to crops and results in a low market for the final products. Finding new ways to gain even small increases in efficiency can make the difference between turning them into a profit or a loss. It has to take care of the pest attack on crops that affects the growth of the field crops. The highly essential cash crops mostly contribute to the vast quantities of production. The insects are the main reason behind crop quality degradation and reduce the productivity of crops, therefore. Hence, monitoring and evaluating the losses due to insects is necessary to ensure crop quality and safety in agriculture.

Machine vision applied in monitoring of crop and soil, fruit grading, plant disease detection, insect pest recognition, and detection. Recently, many developments have been made in the agriculture sector, using machine learning to detect and classify the insects under stored grain conditions [1]. Fruits and vegetables quality evaluation performed using computer vision-based quality inspection comprising four main steps, such as acquisition, segmentation, feature extraction, and classification [2]. Moment invariant techniques applied for extracting shape features and neural networks were developed to classify 20 types of insect images [3]. Yue et al. [4] proposed a super-resolution model based on a deep recursive residual network for agricultural pest surveillance and detection. Pest identification in the complex background using deep residual learning was developed to improve the recognition performance for ten classes of crop insects [5]. Various unsupervised feature learning methods and multi-level classification frameworks were developed for the automatic classification of field crop pests [6]. More recent studies [7,8] reported that image processing applied effectively for insect detection due to less computation cost, fast detection, and easy to distinguish insects with similar color and shape. In [9], clustering segmentation with descriptors is implemented to detect the pests in grapevine with different orientations and lighting environments. The contour-based and region-based segmentation are combined and applied for detecting individual moths and touching insects [10].

Various studies are made to detect the insect using machine learning algorithms. These studies prove that Machine learning classification algorithms can detect the insect on crop. Each of these studies are able to detect and classify their local insect present in the crop. As we know that insects present in a geographical area will differ in some feature from insects present in other geographical area. So, when we apply this detection algorithm on insects present in India especially on insect present on crop of Andhra Pradesh and Telangana it might not give good result. So, we come up with a study to develop the computer vision algorithm to detect insects present in crop of Andhra Pradesh and Telangana. In our study we use Artificial neural networks and Convolutional neural network to perform the detection process. This work implemented on Google colab. The work is implanted in Python and supportive frameworks such as Keras and Tensorflow.

**Chapter 2**

**Review of Prior Works**

**2.1 Insect Detection using Machine Learning algorithms**

In recent years, advanced models in machine learning were successfully achieved the best performance in pest classification and detection [3–6]. Among these works, the various models were trained by using extracted features from the insects and different categories of insect images were classified. It is very difficult to classify and detect insects with similar feature types and different positions in the natural environment. Wang [11] and Xie [12] used ANN and SVM model for the classification of insects in the crops. Wang et al. reported that ANN performed with good stability, and the results of SVM were showed better classification for seven geometrical features. Recently, Yang et al. [13] used SVM based method for the identification of insects with different proportions of the wings. Xie et al. [12] used the SVM model for 24 common pest species of field crops for color, texture, and shape and proposed the effective feature description for insect images. It is well known that ANN and SVM model are provided better classification results that can be applied for insect classification. In contrast, the parameters in existing insect-classification methods influence the process of low-level features [14] and increase the computation time [15]. Hence, the different parameter sets for SVM and ANN algorithms applied successfully to classify insects to achieve state-of-the-art performance. The deep learning model with low classifier has poor performance in image classification applications; hence, it is needed to improve the performance. Liu et al. [16] systematically applied state-of-the-art methods such as VGG, GoogleNet, ResNet, OverFeat, and compare the classification performance with their stack sparse autoencoder model for the improvement of the classification effect of the deep learning model. Nanni et al. [17] also investigated handcrafted and learned descriptors for data augmentation to improve the performance of CNNs. The improvement was obtained by combining local features, dense sampling features, and deep learning approaches using augmented images. This study aims to classify and detect the insects in corn, soybean, and wheat, etc. using machine learning and insect pest detection algorithm at the early stage of crop growth. The different shape features were used for insect classification by applying ANN, SVM, KNN, NB, and CNN models. The performances of machine algorithms for two datasets were compared to provide insect class information and detection of insects performed with four datasets. The insect pest detection algorithm is simple and efficient in terms of computation time for detecting insects in agriculture fields. Image processing techniques applied to segment the foreground insect and locating the position of the insect in the image with a bounding box. The experiments were conducted on a 2.3 GHz Intel corei5 processor with 16 GB of RAM. The work is implemented in MATLAB2018a, Python, and other supporting frameworks such as Keras and Tensorflow for the analysis of detection and classification of insect images in the agriculture field.

All the above studies are able to detect and classify the local insect present on the crop. These studies might not work better to detect & classify the insects present on crops of Andhra Pradesh and Telangana states. So, we made study to classify the insect present on crops of Andhra Pradesh and Telangana state.

In our study we are using three different classes.

1. Ladha
2. Black pest
3. Leaf

Ladha is an insect; black pest is a pest and leaf represents normal leaf without pest and insect.

Set of images are collected for each class from local agricultural fields (cotton and chili plants).

**2.2 Conclusion**

The main objective of our project is to detect local insects, pest or leaf present on crops using various machine learning algorithms. In this study, we are going to detect ladha, black pest and normal leaf from cotton and chili plant of Andhra Pradesh and Telangana.

**Chapter 3**

**Related Work**

**3.1 Artificial neural network (Wikipedia)**

**Artificial neural networks** (**ANNs**), usually simply called **neural networks** (**NNs**), are computing systems inspired by the biological neural networks that constitute animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the neurons in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a [real number](https://en.wikipedia.org/wiki/Real_number), and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called *edges*. Neurons and edges typically have a [*weight*](https://en.wikipedia.org/wiki/Weighting) that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

**3.1.1 Training**

Neural networks learn (or are trained) by processing examples, each of which contains a known "input" and "result," forming probability-weighted associations between the two, which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This difference is the error. The network then adjusts its weighted associations according to a learning rule and using this error value. Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output. After a sufficient number of these adjustments the training can be terminated based upon certain criteria. This is known as [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning).

Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in [image recognition](https://en.wikipedia.org/wiki/Image_recognition), they might learn to identify images that contain cats by analyzing example images that have been manually [labeled](https://en.wikipedia.org/wiki/Labeled_data) as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers, and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

**3.1.2 Models**

ANNs began as an attempt to exploit the architecture of the human brain to perform tasks that conventional algorithms had little success with. They soon reoriented towards improving empirical results, mostly abandoning attempts to remain true to their biological precursors. Neurons are connected to each other in various patterns, to allow the output of some neurons to become the input of others. The network forms a [directed](https://en.wikipedia.org/wiki/Directed_graph), [weighted graph](https://en.wikipedia.org/wiki/Weighted_graph).

An artificial neural network consists of a collection of simulated neurons. Each neuron is a [node](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) which is connected to other nodes via [links](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms#edge) that correspond to biological axon-synapse-dendrite connections. Each link has a weight, which determines the strength of one node's influence on another.

**3.1.3 Artificial neurons**

ANNs are composed of [artificial neurons](https://en.wikipedia.org/wiki/Artificial_neurons) which are conceptually derived from biological [neurons](https://en.wikipedia.org/wiki/Neuron). Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons. The outputs of the final *output neurons* of the neural net accomplish the task, such as recognizing an object in an image.

To find the output of the neuron, first we must take the weighted sum of all the inputs, weighted by the *weights* of the *connections* from the inputs to the neuron. We add a *bias* term to this sum. This weighted sum is sometimes called the *activation*. This weighted sum is then passed through a (usually nonlinear) [activation function](https://en.wikipedia.org/wiki/Activation_function) to produce the output. The initial inputs are external data, such as images and documents. The ultimate outputs accomplish the task, such as recognizing an object in an image.

**3.1.4 Organization**

The neurons are typically organized into multiple layers, especially in [deep learning](https://en.wikipedia.org/wiki/Deep_learning). Neurons of one layer connect only to neurons of the immediately preceding and immediately following layers. The layer that receives external data is the *input layer*. The layer that produces the ultimate result is the *output layer*. In between them are zero or more *hidden layers*. Single layer and unlayered networks are also used. Between two layers, multiple connection patterns are possible. They can be 'fully connected', with every neuron in one layer connecting to every neuron in the next layer. They can be *pooling*, where a group of neurons in one layer connect to a single neuron in the next layer, thereby reducing the number of neurons in that layer. Neurons with only such connections form a [directed acyclic graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph) and are known as [feedforward networks](https://en.wikipedia.org/wiki/Feedforward_neural_network). Alternatively, networks that allow connections between neurons in the same or previous layers are known as [recurrent networks](https://en.wikipedia.org/wiki/Recurrent_neural_network)*.*

**3.1.5 Hyperparameter**

A hyperparameter is a constant [parameter](https://en.wikipedia.org/wiki/Parameter) whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include [learning rate](https://en.wikipedia.org/wiki/Learning_rate), the number of hidden layers and batch size. The values of some hyperparameters can be dependent on those of other hyperparameters. For example, the size of some layers can depend on the overall number of layers.

**3.1.6 Learning**

Learning is the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result. This is done by minimizing the observed errors. Learning is complete when examining additional observations does not usefully reduce the error rate. Even after learning, the error rate typically does not reach 0. If after learning, the error rate is too high, the network typically must be redesigned. Practically this is done by defining a [cost function](https://en.wikipedia.org/wiki/Loss_function) that is evaluated periodically during learning. As long as its output continues to decline, learning continues. The cost is frequently defined as a [statistic](https://en.wikipedia.org/wiki/Statistic) whose value can only be approximated. The outputs are actually numbers, so when the error is low, the difference between the output (almost certainly a cat) and the correct answer (cat) is small. Learning attempts to reduce the total of the differences across the observations. Most learning models can be viewed as a straightforward application of [optimization](https://en.wikipedia.org/wiki/Mathematical_optimization) theory and [statistical estimation](https://en.wikipedia.org/wiki/Statistical_estimation).

**3.1.7 Learning rate**

The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A high learning rate shortens the training time, but with lower ultimate accuracy, while a lower learning rate takes longer, but with the potential for greater accuracy. Optimizations such as [Quickprop](https://en.wikipedia.org/wiki/Quickprop" \o "Quickprop) are primarily aimed at speeding up error minimization, while other improvements mainly try to increase reliability. In order to avoid oscillation inside the network such as alternating connection weights, and to improve the rate of convergence, refinements use an [adaptive learning rate](https://en.wikipedia.org/wiki/Adaptive_learning_rate) that increases or decreases as appropriate. The concept of momentum allows the balance between the gradient and the previous change to be weighted such that the weight adjustment depends to some degree on the previous change. A momentum close to 0 emphasizes the gradient, while a value close to 1 emphasizes the last change.

**3.1.8 Cost function**

While it is possible to define a cost function [ad hoc](https://en.wikipedia.org/wiki/Ad_hoc), frequently the choice is determined by the function's desirable properties (such as [convexity](https://en.wikipedia.org/wiki/Convex_function)) or because it arises from the model (e.g. in a probabilistic model the model's [posterior probability](https://en.wikipedia.org/wiki/Posterior_probability) can be used as an inverse cost).

**3.1.9 Backpropagation**

Backpropagation is a method used to adjust the connection weights to compensate for each error found during learning. The error amount is effectively divided among the connections. Technically, backprop calculates the [gradient](https://en.wikipedia.org/wiki/Gradient) (the derivative) of the [cost function](https://en.wikipedia.org/wiki/Loss_function) associated with a given state with respect to the weights. The weight updates can be done via [stochastic gradient descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent) or other methods, such as [Extreme Learning Machines](https://en.wikipedia.org/wiki/Extreme_Learning_Machines), "No-prop" networks, training without backtracking, "weightless" networks, and [non-connectionist neural networks](https://en.wikipedia.org/wiki/Holographic_associative_memory).

**3.1.10Types**

ANNs have evolved into a broad family of techniques that have advanced the state of the art across multiple domains. The simplest types have one or more static components, including number of units, number of layers, unit weights and [topology](https://en.wikipedia.org/wiki/Topology). Dynamic types allow one or more of these to evolve via learning. The latter are much more complicated, but can shorten learning periods and produce better results. Some types allow/require learning to be "supervised" by the operator, while others operate independently. Some types operate purely in hardware, while others are purely software and run on general purpose computers.

Some of the main breakthroughs include: [convolutional neural networks](https://en.wikipedia.org/wiki/Convolutional_neural_network) that have proven particularly successful in processing visual and other two-dimensional data; long short-term memory avoid the [vanishing gradient problem](https://en.wikipedia.org/wiki/Vanishing_gradient_problem) and can handle signals that have a mix of low and high frequency components aiding large-vocabulary speech recognition,text-to-speech synthesis, and photo-real talking heads; competitive networks such as [generative adversarial networks](https://en.wikipedia.org/wiki/Generative_adversarial_network) in which multiple networks (of varying structure) compete with each other, on tasks such as winning a game or on deceiving the opponent about the authenticity of an input.

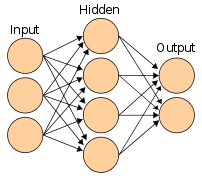


Figure 3.1 Artificial Neural Networks

**3.2 Convolutional neural network**

In [deep learning](https://en.wikipedia.org/wiki/Deep_learning), a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of [Artificial Neural Network](https://en.wikipedia.org/wiki/Artificial_Neural_Network) (**ANN**), most commonly applied to analyze visual imagery. They are also known as **Shift Invariant** or **Space Invariant Artificial Neural Networks** (**SIANN**), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation [equivariant](https://en.wikipedia.org/wiki/Equivariant_map) responses known as feature maps. Counter-intuitively, most convolutional neural networks are only [equivariant](https://en.wikipedia.org/wiki/Equivariant_map), as opposed to [invariant](https://en.wikipedia.org/wiki/Translation_invariant), to translation. They have applications in [image and video recognition](https://en.wikipedia.org/wiki/Computer_vision), [recommender systems](https://en.wikipedia.org/wiki/Recommender_system), [image classification](https://en.wikipedia.org/wiki/Image_classification), [image segmentation](https://en.wikipedia.org/wiki/Image_segmentation), [medical image analysis](https://en.wikipedia.org/wiki/Medical_image_computing), [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing), [brain-computer interfaces](https://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface), and financial [time series](https://en.wikipedia.org/wiki/Time_series).

CNNs are [regularized](https://en.wikipedia.org/wiki/Regularization_(mathematics)) versions of [multilayer perceptrons](https://en.wikipedia.org/wiki/Multilayer_perceptron). Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one [layer](https://en.wikipedia.org/wiki/Layer_(deep_learning)) is connected to all neurons in the next [layer](https://en.wikipedia.org/wiki/Layer_(deep_learning)). The "full connectivity" of these networks make them prone to [overfitting](https://en.wikipedia.org/wiki/Overfitting) data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were [inspired](https://en.wikipedia.org/wiki/Mathematical_biology) by [biological](https://en.wikipedia.org/wiki/Biological) processes in that the connectivity pattern between [neurons](https://en.wikipedia.org/wiki/Artificial_neuron) resembles the organization of the animal [visual cortex](https://en.wikipedia.org/wiki/Visual_cortex). Individual [cortical neurons](https://en.wikipedia.org/wiki/Cortical_neuron) respond to stimuli only in a restricted region of the [visual field](https://en.wikipedia.org/wiki/Visual_field) known as the [receptive field](https://en.wikipedia.org/wiki/Receptive_field). The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other [image classification algorithms](https://en.wikipedia.org/wiki/Image_classification). This means that the network learns to optimize the [filters](https://en.wikipedia.org/wiki/Filter_(signal_processing)) (or kernels) through automated learning, whereas in traditional algorithms these filters are [hand-engineered](https://en.wikipedia.org/wiki/Feature_engineering). This independence from prior knowledge and human intervention in feature extraction is a major advantage.

**3.2.1 Definition**

The name "convolutional neural network" indicates that the network employs a mathematical operation called [convolution](https://en.wikipedia.org/wiki/Convolution). Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers. or in other words "A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data."

**3.2.2 Architecture**

A convolutional neural network consists of an input layer, [hidden layers](https://en.wikipedia.org/wiki/Multilayer_perceptron#Layers) and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final [convolution](https://en.wikipedia.org/wiki/Convolution). In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically this includes a layer that performs a [dot product](https://en.wikipedia.org/wiki/Dot_product) of the convolution kernel with the layer's input matrix. This product is usually the [Frobenius inner product](https://en.wikipedia.org/wiki/Frobenius_inner_product" \o "Frobenius inner product), and its activation function is commonly [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)" \o "Rectifier (neural networks)). As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers.

### 3.2.3 Convolutional layers

In a CNN, the input is a [tensor](https://en.wikipedia.org/wiki/Tensor) with a shape: (number of inputs) x (input height) x (input width) x (input [channels](https://en.wikipedia.org/wiki/Channel_(digital_image))). After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape: (number of inputs) x (feature map height) x (feature map width) x (feature map [channels](https://en.wikipedia.org/wiki/Channel_(digital_image))).

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its [receptive field](https://en.wikipedia.org/wiki/Receptive_field). Although [fully connected feedforward neural networks](https://en.wikipedia.org/wiki/Multilayer_perceptron) can be used to learn features and classify data, this architecture is generally impractical for larger inputs such as high resolution images. It would require a very high number of neurons, even in a shallow architecture, due to the large input size of images, where each pixel is a relevant input feature. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for *each* neuron in the second layer. Instead, convolution reduces the number of free parameters, allowing the network to be deeper. For example, regardless of image size, using a 5 x 5 tiling region, each with the same shared weights, requires only 25 learnable parameters. Using regularized weights over fewer parameters avoids the vanishing gradients and exploding gradients problems seen during [backpropagation](https://en.wikipedia.org/wiki/Backpropagation) in traditional neural networks. Furthermore, convolutional neural networks are ideal for data with a grid-like topology (such as images) as spatial relations between separate features are taken into account during convolution and/or pooling.

### 3.2.4 Pooling layers

Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2 x 2 are commonly used. Global pooling acts on all the neurons of the feature map. There are two common types of pooling in popular use: max and average. *Max pooling* uses the maximum value of each local cluster of neurons in the feature map, while *average pooling* takes the average value.

### 3.2.5 Fully connected layers

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is the same as a traditional [multi-layer perceptron](https://en.wikipedia.org/wiki/Multi-layer_perceptron) neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

### 3.2.6 Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a convolutional layer, each neuron receives input from only a restricted area of the previous layer called the neuron's *receptive field*. Typically the area is a square (e.g. 5 by 5 neurons). Whereas, in a fully connected layer, the receptive field is the *entire previous layer*. Thus, in each convolutional layer, each neuron takes input from a larger area in the input than previous layers. This is due to applying the convolution over and over, which takes into account the value of a pixel, as well as its surrounding pixels. When using dilated layers, the number of pixels in the receptive field remains constant, but the field is more sparsely populated as its dimensions grow when combining the effect of several layers.

### 3.2.7 Weights

Each neuron in a neural network computes an output value by applying a specific function to the input values received from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning consists of iteratively adjusting these biases and weights.

The vector of weights and the bias are called *filters* and represent particular [features](https://en.wikipedia.org/wiki/Feature_(machine_learning)) of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces the [memory footprint](https://en.wikipedia.org/wiki/Memory_footprint) because a single bias and a single vector of weights are used across all receptive fields that share that filter, as opposed to each receptive field having its own bias and vector weighting.



Figure 3.2 Convolutional Neural Networks

**3.2.8 Building blocks**

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function. A few distinct types of layers are commonly used. These are further discussed below.

**Convolutional layer**

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable [filters](https://en.wikipedia.org/wiki/Filter_(signal_processing)) (or [kernels](https://en.wikipedia.org/wiki/Kernel_(image_processing))), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is [convolved](https://en.wikipedia.org/wiki/Convolution) across the width and height of the input volume, computing the [dot product](https://en.wikipedia.org/wiki/Dot_product) between the filter entries and the input, producing a 2-dimensional [activation map](https://en.wikipedia.org/wiki/Activation_function) of that filter. As a result, the network learns filters that activate when it detects some specific type of [feature](https://en.wikipedia.org/wiki/Feature_(machine_learning)) at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

**Local connectivity**

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such a network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a [sparse local connectivity](https://en.wikipedia.org/wiki/Sparse_network) pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume.

The extent of this connectivity is a [hyperparameter](https://en.wikipedia.org/wiki/Hyperparameter_optimization) called the [receptive field](https://en.wikipedia.org/wiki/Receptive_field) of the neuron. The connections are [local in space](https://en.wikipedia.org/wiki/Spatial_Locality) (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

**Parameter sharing**

A parameter sharing scheme is used in convolutional layers to control the number of free parameters. It relies on the assumption that if a patch feature is useful to compute at some spatial position, then it should also be useful to compute at other positions. Denoting a single 2-dimensional slice of depth as a *depth slice*, the neurons in each depth slice are constrained to use the same weights and bias.

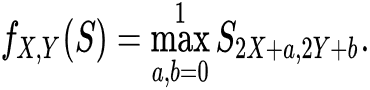
Since all neurons in a single depth slice share the same parameters, the forward pass in each depth slice of the convolutional layer can be computed as a [convolution](https://en.wikipedia.org/wiki/Convolution) of the neuron's weights with the input volume. Therefore, it is common to refer to the sets of weights as a filter (or a [kernel](https://en.wikipedia.org/wiki/Kernel_(image_processing))), which is convolved with the input. The result of this convolution is an [activation map](https://en.wikipedia.org/wiki/Activation_function), and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. Parameter sharing contributes to the [translation invariance](https://en.wikipedia.org/wiki/Translational_symmetry) of the CNN architecture.

Sometimes, the parameter sharing assumption may not make sense. This is especially the case when the input images to a CNN have some specific centered structure; for which we expect completely different features to be learned on different spatial locations. One practical example is when the inputs are faces that have been centered in the image: we might expect different eye-specific or hair-specific features to be learned in different parts of the image. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a "locally connected layer".

**Pooling layer**

Another important concept of CNNs is pooling, which is a form of non-linear [down-sampling](https://en.wikipedia.org/wiki/Downsampling_(signal_processing)). There are several non-linear functions to implement pooling, where *max pooling* is the most common. It [partitions](https://en.wikipedia.org/wiki/Partition_of_a_set) the input image into a set of rectangles and, for each such sub-region, outputs the maximum.

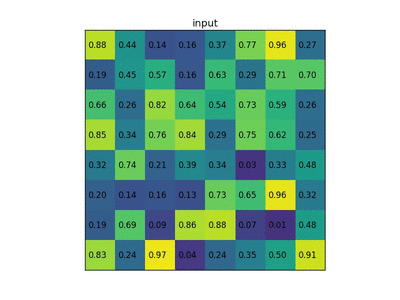
Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, [memory footprint](https://en.wikipedia.org/wiki/Memory_footprint) and amount of computation in the network, and hence to also control [overfitting](https://en.wikipedia.org/wiki/Overfitting). This is known as down-sampling. It is common to periodically insert a pooling layer between successive convolutional layers (each one typically followed by an activation function, such as a [ReLU layer](https://en.wikipedia.org/wiki/Convolutional_neural_network" \l "ReLU_layer) in a CNN architecture. While pooling layers contribute to local translation invariance, they do not provide global translation invariance in a CNN, unless a form of global pooling is used. The pooling layer commonly operates independently on every depth, or slice, of the input and resizes it spatially. A very common form of max pooling is a layer with filters of size 2×2, applied with a stride of 2, which subsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations:



n this case, every [max operation](https://en.wikipedia.org/wiki/Maximum) is over 4 numbers. The depth dimension remains unchanged (this is true for other forms of pooling as well).

In addition to max pooling, pooling units can use other functions, such as [average](https://en.wikipedia.org/wiki/Average) pooling or [ℓ2-norm](https://en.wikipedia.org/wiki/Euclidean_norm) pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which generally performs better in practice.

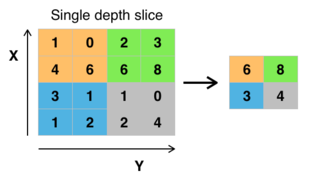
Due to the effects of fast spatial reduction of the size of the representation, there is a recent trend towards using smaller filters or discarding pooling layers altogether.

[](https://en.wikipedia.org/wiki/File:RoI_pooling_animated.gif)

RoI pooling to size 2x2. In this example region proposal (an input parameter) has size 7x5.

"[Region of Interest](https://en.wikipedia.org/wiki/Region_of_interest)" pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter.

Pooling is an important component of convolutional neural networks for [object detection](https://en.wikipedia.org/wiki/Object_detection) based on the Fast R-CNN architecture



**ReLU layer**

ReLU is the abbreviation of [rectified linear unit](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)), which applies the non-saturating [activation function](https://en.wikipedia.org/wiki/Activation_function) {\textstyle f(x)=\max(0,x)}

f(x) = max(0, x)

It effectively removes negative values from an activation map by setting them to zero. It introduces [nonlinearities](https://en.wikipedia.org/wiki/Nonlinearity_(journal)) to the [decision function](https://en.wikipedia.org/wiki/Decision_boundary) and in the overall network without affecting the receptive fields of the convolution layers.

Other functions can also be used to increase nonlinearity, for example the saturating [hyperbolic tangent](https://en.wikipedia.org/wiki/Hyperbolic_tangent) {\displaystyle f(x)=\tanh(x)} f(x) = tanh(x)

f(x) = |tanh(x)|

 and the [sigmoid function](https://en.wikipedia.org/wiki/Sigmoid_function). ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to [generalization](https://en.wikipedia.org/wiki/Generalization_(learning)) accuracy.

**Fully connected layer**

After several convolutional and max pooling layers, the final classification is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network). Their activations can thus be computed as an [affine transformation](https://en.wikipedia.org/wiki/Affine_transformation), with [matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication) followed by a bias offset ([vector addition](https://en.wikipedia.org/wiki/Vector_addition) of a learned or fixed bias term.

**Loss layer**

The "loss layer", or "[loss function](https://en.wikipedia.org/wiki/Loss_function)", specifies how [training](https://en.wikipedia.org/wiki/Training) penalizes the deviation between the predicted output of the network, and the [true](https://en.wikipedia.org/wiki/Ground_truth) data labels (during supervised learning). Various [loss functions](https://en.wikipedia.org/wiki/Loss_function) can be used, depending on the specific task.

The [Softmax](https://en.wikipedia.org/wiki/Softmax_function" \o "Softmax function) loss function is used for predicting a single class of *K* mutually exclusive classes. [Sigmoid](https://en.wikipedia.org/wiki/Sigmoid_function) [cross-entropy](https://en.wikipedia.org/wiki/Cross_entropy) loss is used for predicting *K* independent probability values in [0,1]{\displaystyle [0,1]}. [Euclidean](https://en.wikipedia.org/wiki/Euclidean_distance) loss is used for [regressing](https://en.wikipedia.org/wiki/Regression_(machine_learning)) to [real-valued](https://en.wikipedia.org/wiki/Real_number) labels (-inf, +inf){\displaystyle (-\infty ,\infty )}.

Hyperparameters

Hyperparameters are various settings that are used to control the learning process. CNNs use more [hyperparameters](https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)) than a standard multilayer perceptron (MLP).

**Kernel size**

The kernel is the number of pixels processed together. It is typically expressed as the kernel's dimensions, e.g., 2x2, or 3x3.

**Padding**

Padding is the addition of (typically) 0-valued pixels on the borders of an image. This is done so that the border pixels are not undervalued (lost) from the output because they would ordinarily participate in only a single receptive field instance. The padding applied is typically one less than the corresponding kernel dimension. For example, a convolutional layer using 3x3 kernels would receive a 2-pixel pad on all sides of the image.[[71]](https://en.wikipedia.org/wiki/Convolutional_neural_network#cite_note-74)

**Stride**

The stride is the number of pixels that the analysis window moves on each iteration. A stride of 2 means that each kernel is offset by 2 pixels from its predecessor.

**Number of filters**

Since feature map size decreases with depth, layers near the input layer tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the product of feature values *va* with pixel position is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next.

The number of feature maps directly controls the capacity and depends on the number of available examples and task complexity.

**Filter size**

Common filter sizes found in the literature vary greatly, and are usually chosen based on the data set.

The challenge is to find the right level of granularity so as to create abstractions at the proper scale, given a particular data set, and without [overfitting](https://en.wikipedia.org/wiki/Overfitting).

**Pooling type and size**

[Max pooling](https://en.wikipedia.org/wiki/Max_pooling) is typically used, often with a 2x2 dimension. This implies that the input is drastically [downsampled](https://en.wikipedia.org/wiki/Downsampling_(signal_processing)" \o "Downsampling (signal processing)), reducing processing cost.

Large input volumes may warrant 4×4 pooling in the lower layers. Greater pooling [reduces the dimension](https://en.wikipedia.org/wiki/Dimensionality_reduction) of the signal, and may result in unacceptable [information loss](https://en.wikipedia.org/wiki/Data_loss). Often, non-overlapping pooling windows perform best.

**Dilation**

Dilation involves ignoring pixels within a kernel. This reduces processing/memory potentially without significant signal loss. A dilation of 2 on a 3x3 kernel expands the kernel to 7x7, while still processing 9 (evenly spaced) pixels. Accordingly, dilation of 4 expands the kernel to 15x15.

## 3.2.8 Regularization methods

[Regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)) is a process of introducing additional information to solve an [ill-posed problem](https://en.wikipedia.org/wiki/Ill-posed_problem) or to prevent [overfitting](https://en.wikipedia.org/wiki/Overfitting). CNNs use various types of regularization.

### Empirical

#### Dropout

Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is [dropout](https://en.wikipedia.org/wiki/Dropout_(neural_networks)). At each training stage, individual nodes are either "dropped out" of the net (ignored) with probability {\displaystyle 1-p} or kept with probability {\displaystyle p}, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

In the training stages, {\displaystyle p} is usually 0.5; for input nodes, it is typically much higher because information is directly lost when input nodes are ignored.

At testing time after training has finished, we would ideally like to find a sample average of all possible {\displaystyle 2^{n}} dropped-out networks; unfortunately this is unfeasible for large values of {\displaystyle n}. However, we can find an approximation by using the full network with each node's output weighted by a factor of {\displaystyle p}, so the [expected value](https://en.wikipedia.org/wiki/Expected_value) of the output of any node is the same as in the training stages. This is the biggest contribution of the dropout method: although it effectively generates {\displaystyle 2^{n}} neural nets, and as such allows for model combination, at test time only a single network needs to be tested.

By avoiding training all nodes on all training data, dropout decreases overfitting. The method also significantly improves training speed. This makes the model combination practical, even for [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network). The technique seems to reduce node interactions, leading them to learn more robust features that better generalize to new data.

#### DropConnect

DropConnect is the generalization of dropout in which each connection, rather than each output unit, can be dropped with probability {\displaystyle 1-p}. Each unit thus receives input from a random subset of units in the previous layer.

DropConnect is similar to dropout as it introduces dynamic sparsity within the model, but differs in that the sparsity is on the weights, rather than the output vectors of a layer. In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at random during the training stage.

#### Stochastic pooling

A major drawback to Dropout is that it does not have the same benefits for convolutional layers, where the neurons are not fully connected.

In stochastic pooling, the conventional [deterministic](https://en.wikipedia.org/wiki/Deterministic_algorithm) pooling operations are replaced with a stochastic procedure, where the activation within each pooling region is picked randomly according to a [multinomial distribution](https://en.wikipedia.org/wiki/Multinomial_distribution), given by the activities within the pooling region. This approach is free of hyperparameters and can be combined with other regularization approaches, such as dropout and [data augmentation](https://en.wikipedia.org/wiki/Data_augmentation).

An alternate view of stochastic pooling is that it is equivalent to standard max pooling but with many copies of an input image, each having small local [deformations](https://en.wikipedia.org/wiki/Deformation_theory). This is similar to explicit [elastic deformations](https://en.wikipedia.org/wiki/Elastic_deformation) of the input images, which delivers excellent performance on the [MNIST data set](https://en.wikipedia.org/wiki/MNIST_database). Using stochastic pooling in a multilayer model gives an exponential number of deformations since the selections in higher layers are independent of those below.

#### Artificial data

Because the degree of model overfitting is determined by both its power and the amount of training it receives, providing a convolutional network with more training examples can reduce overfitting. Because these networks are usually trained with all available data, one approach is to either generate new data from scratch (if possible) or perturb existing data to create new ones. For example, input images can be cropped, rotated, or rescaled to create new examples with the same labels as the original training set.

### 3.2.9 Explicit

One of the simplest methods to prevent overfitting of a network is to simply stop the training before overfitting has had a chance to occur. It comes with the disadvantage that the learning process is halted.

#### Number of parameters

Another simple way to prevent overfitting is to limit the number of parameters, typically by limiting the number of hidden units in each layer or limiting network depth. For convolutional networks, the filter size also affects the number of parameters. Limiting the number of parameters restricts the predictive power of the network directly, reducing the complexity of the function that it can perform on the data, and thus limits the amount of overfitting. This is equivalent to a "[zero norm](https://en.wikipedia.org/wiki/Zero_norm)".

#### Weight decay

A simple form of added regularizer is weight decay, which simply adds an additional error, proportional to the sum of weights ([L1 norm](https://en.wikipedia.org/wiki/L1-norm)) or squared magnitude ([L2 norm](https://en.wikipedia.org/wiki/L2_norm)) of the weight vector, to the error at each node. The level of acceptable model complexity can be reduced by increasing the proportionality constant('alpha' hyperparameter), thus increasing the penalty for large weight vectors.

L2 regularization is the most common form of regularization. It can be implemented by penalizing the squared magnitude of all parameters directly in the objective. The L2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors. Due to multiplicative interactions between weights and inputs this has the useful property of encouraging the network to use all of its inputs a little rather than some of its inputs a lot.

L1 regularization is also common. It makes the weight vectors sparse during optimization. In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the noisy inputs. L1 with L2 regularization can be combined; this is called [Elastic net regularization](https://en.wikipedia.org/wiki/Elastic_net_regularization).

#### Max norm constraints**:**

Another form of regularization is to enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use [projected gradient descent](https://en.wikipedia.org/wiki/Sparse_approximation#Projected_Gradient_Descent) to enforce the constraint. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector {\displaystyle {\vec {w}}} of every neuron to satisfy {\displaystyle \|{\vec {w}}\|\_{2}<c}. Typical values of {\displaystyle c}are order of 3–4. Some papers report improvements when using this form of regularization.

**3.2.10 Materials and Methods**

In this study we have three classes. They are

1. Ladha
2. Black pest
3. Leaf

For each and Every class we have collected the images from local agricultural fields of Andhra Pradesh and Telangana. Totally we have collected 1503 images. The below table represent the no of images we have collected for each class.

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **No of images** | **Train** | **Test** |
| Ladha | 736 | 686 | 50 |
| Black Pest | 512 | 462 | 50 |
| Leaf | 255 | 205 | 50 |
| **Total** | **1503** | **1353** | **150** |

Table 3.1 Dataset

Some of the images are shown below

**Black Pest:**

Figure 3.3 Black Pest

**Ladha:**

Figure 3.4 Ladha

**Leaf:**

**   **

Figure 3.5 Leaf

**Image Preprocessing:**

In image pre-processing, image enhancement techniques applied to reduce noise in the images and sharpen the images for better accuracy. It improves the quality of the image for better detection and classification of insects. The datasets used here are already pre-processed and segmented.

**Image Agumentation:**

Since fewer insect images are available in both Wang and Xie datasets, image augmentation is applied. The insect images were rescaled to the size of 227 \* 227 pixels. In our study 514\*514 pixels. Image data augmentation techniques such as rotation, flipping, and cropping operators are used to increase the training set for achieving improved accuracy and eliminate the problems of overtraining [21,22]. As shown in Fig. 1,.





2700 rotation

1800 rotation

900 rotation

Original Image





2700 rotation with Flipping

Right to left

1800 rotation with Flipping

right to left

900 rotation with Flipping

right to left

Figure 3.6 Data Augumentation

**Image Classification Methodology**

Insect classification involves various steps to be performed. The flow of steps for insect classification is illustrated in Fig. 2. Image augmentation is applied to insect dataset images to expand the training dataset. Then, shape features extracted from the insect images and ANN machine learning algorithm are applied to classify the insect classes. CNN based insect classification adapted for comparison performance.

2.2.1. Insect classification with image processing and machine learning techniques

Shape feature extraction. Shape features are the essential features which are not affected due to scaling, rotation, and translation and applied in computer vision and automatic object recognition systems. Classification of insects is performed based on the finite shape features extracted from the insect images. The insect images in the form of RGB converted to grayscale images for further feature extraction. Image processing techniques are applied to extract the shape features using the Sobel edge detection algorithm and morphological operations [23]. The nine shape features include area, perimeter, major axis length, minor axis length, eccentricity, circularity, solidity, form factor, and compactness [23] stored in feature vectors and then applied to the classifier models

2.2.1.2. Insect classification with ANN. The insects are classified into various classes using the four machine learning techniques such as ANN classifier and described as follows.

Collecting

Images of Insects (Dataset)

Image

pre-processing

Apply

Image augmentation

Shape Feature Extraction

Insect Classification using ANN

CNN Insect Classification

Figure 3.7 Flowchart

**ANN classifier**

A simple neural network contains an input, hidden, and output layer with linkages. Initially, random weights are assigned. The final linkage weights determined, and activation rates of the output layer were calculated. A feed-forward multi-layer artificial neural network is designed to automatically identify and classify adult stage whiteflies and thrip in greenhouses [24]. The insect recognition model’s ability was improved using a backpropagation ANN model for identifying Beet armyworm (Spodoptera exigua) from other species [25]

**CNN Classifier –**

The CNN model developed to train with RGB insect images from the dataset. The CNN model comes under the class of deep, feed-forward neural networks applied to analyze visual imagery of insect images and computationally efficient due to automatic feature learning and weight sharing [30–33]. The CNN model contains two convolutional layers and two max-pooling layers, a flatten layer, a fully connected layer, and a softmax output layer to classify insect images. The size of the insect image is rescaled to 64 \* 64. The CNN model can run over each insect image pretty fast and reduce the computational operations per layer and memory requirements. Each convolution layer and max-pooling layer use 3 \* 3 and 5 \* 5 filter sizes, respectively. A fully connected layer is designed in such a way to learn high-level features for final insect classification.

**Chapter 4**

**System Design**

Upload Image

**USER**

**Detected class**

**CNN MODEL WEIGHTS**

**WEB APP**

Result

Classify the image

Figure 4.1 Block Diagram of the System

This chapter discusses the modules in the system. First the user will upload the image in web app using web application that we developed. The web app uses the weights of cnn model and using the weights the web app will classify the image. The result will be displayed on web page.

The system consists of the following modules:

1. Web Interface
2. Image Processing engine

**Web Interface**

Web Interface in this project is used to accept an image from the user and the image will be given as input to the image processing. This web interface will also used to display the output of the image processing engine. This web interface is developed by using flask web development framework. Here it will have home.html file which will have an interface to accept image from the user. This image will be processed and the output will be displayed on predict.html page.

**Image Processing Engine**

Image processing engine is the main part of the web app. This engine will accept the image from home.html page and using the weight from cnn model & image processing libraries this image will be classified into a class.

The ouput for this will be displayed on predict,html page

**Chapter 5**

**Implementation Details And Results**

This chapter discusses the details of the project implementation along with the obtained results. The details presented in this chapter are categorized as follows:

1. Data set description
2. Python Libraries
3. Image Processing Tools
4. Implementation of Image Processing module

**5.1 Data set Description**

The Insect detection project is a simple and easy to recognize the insects in a image that is given as input to the model. This project is implemented on real world images that are collected from fields of Andhra Pradesh and Telangana. This Data set consists of three different classes ladhe, blackpest and normal leaf. Total of we have 1500 images. Each image is in .jpg format and every image is proprocessed to 512\*512 pixel images.

To avoid any unwanted noise, we have made processing to eliminate the noise

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **No of images** | **Train** | **Test** |
| Ladha | 736 | 686 | 50 |
| Black Pest | 512 | 462 | 50 |
| Leaf | 255 | 205 | 50 |
| **Total** | **1503** | **1353** | **150** |

Table 5.1 Dataset

Figure 5.1 Ladha, Blackpest, Leaf

**5.2 Implementation Details**

This section gives a broad view of the libraries used for implementing the modules in the system with the issues that were handled. The following modules were used for implementing the image recognition and Web application.

1. TensorFlow
2. OpenCV
3. Matplotlib
4. Flask

The Image recognition for this project is implemented by using two algorithms

* Initially, an Artificial neural network is used to implement the model for our project using keras library
* Later Convolution neural network used to implement the model using keras.

The model developed by using Artificial neural network was not able to give the best performance. However, Convolution neural network gives us the best performance.

**5.3 Implementation of Insect Detection**

ANN and CNN are used to implement insect detection model. Artificial neural network (ANN) uses simple neural network structure for classification problem. Convolution neural network (CNN) uses additional conv layers to classification problem.

Insect Data

Image Augmentation

Shape Feature Extraction

Insect Classification using ANN

Convolution neural network (CNN)

CNN insect classification

CNN insect classification

Figure 5.2 Framework for insect classification.

**5.4 work for insect classification**.

The libraries we have used in the code are python, Tensorflow, openCV, NumPy, Flask.

**5.4.1 Tensorflow**

**TensorFlow** is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [software library](https://en.wikipedia.org/wiki/Library_(computing)) for [machine learning](https://en.wikipedia.org/wiki/Machine_learning) and [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence). It can be used across a range of tasks but has a particular focus on [training](https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks#Training) and [inference](https://en.wikipedia.org/wiki/Statistical_inference) of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_networks).

TensorFlow was developed by the [Google Brain](https://en.wikipedia.org/wiki/Google_Brain) team for internal [Google](https://en.wikipedia.org/wiki/Google) use in research and production. The initial version was released under the [Apache License 2.0](https://en.wikipedia.org/wiki/Apache_License_2.0) in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019.

TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as Javascript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

**5.4.2 Keras** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [software](https://en.wikipedia.org/wiki/AI_software) library that provides a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) interface for [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network). Keras acts as an interface for the [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow) library.

Up until version 2.3, Keras supported multiple backends, including [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), [Microsoft Cognitive Toolkit](https://en.wikipedia.org/wiki/Microsoft_Cognitive_Toolkit), [Theano](https://en.wikipedia.org/wiki/Theano_(software)), and [PlaidML](https://en.wikipedia.org/wiki/PlaidML" \o "PlaidML). As of version 2.4, only [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow) is supported. Designed to enable fast experimentation with [deep neural networks](https://en.wikipedia.org/wiki/Deep_learning), it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is [François Chollet](https://en.wikipedia.org/wiki/Fran%C3%A7ois_Chollet), a [Google](https://en.wikipedia.org/wiki/Google) engineer. Chollet is also the author of the XCeption deep neural network model.

**Command:** pip install tensorflow

**OpenCV**

**OpenCV** (*Open Source Computer Vision Library*) is a [library of programming functions](https://en.wikipedia.org/wiki/Library_(computing)) mainly aimed at real-time [computer vision](https://en.wikipedia.org/wiki/Computer_vision). Originally developed by [Intel](https://en.wikipedia.org/wiki/Intel_Corporation), it was later supported by [Willow Garage](https://en.wikipedia.org/wiki/Willow_Garage) then Itseez (which was later acquired by Intel). The library is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and free for use under the [open-source](https://en.wikipedia.org/wiki/Open-source_software) [Apache 2 License](https://en.wikipedia.org/wiki/Apache_License). Starting with 2011, OpenCV features GPU acceleration for real-time operations.

**Command:** pip install opencv-python

**5.4.3 NumPy**

NumPy is a Python library used for working with arrays. It also has functions for working in the

domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis

Oliphant. It is an open-source project and you can use it freely. NumPy stands for Numerical

Python. We can install NumPy as:

Command: pip install numpy

**5.4.4 Flask**

**Flask** is a micro [web framework](https://en.wikipedia.org/wiki/Web_framework) written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It is classified as a [microframework](https://en.wikipedia.org/wiki/Microframework) because it does not require particular tools or libraries. It has no [database](https://en.wikipedia.org/wiki/Database) abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Command: pip install Flask

Python >= 3.5

TensorFlow >= 2.2.0

NumPy >= 1.10

Flask >= 2.0.3

scikit-learn >= 0.16

And we also require

Matplotlib >= 1.1.1

**5.5 Code & Result**

**Image preprocessing:**

Eliminated unnecessary noise from the dataset

Code:

import cv2

import glob

import os

inputFolder ='leaf'

os.mkdir('leaf Resized')

i = 0

for img in glob.glob(inputFolder+ '/\*.jpg'):

image = cv2.imread(img)

imgResized = cv2.resize(image, (512, 512))

cv2.imwrite("Leaf Resized\image%04i.jpg" %i, imgResized)

i+=1

cv2.destroyAllWindos()

result:

images will be rescaled to 512\*512 pixel images

* + 1. **Artificial neural network**

Code:

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#import utils

import os

%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D

from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

#from tensorflow.keras.utils import plot\_model

from IPython.display import SVG, Image

#from livelossplot import PlotLossesTensorFlowKeras()

import tensorflow as tf

base\_dir = '/dataset' #we have data set in dataset folder

train\_dir = os.path.join(base\_dir,'train')

test\_dir = os.path.join(base\_dir,'test')

#data augmentation

img\_size = 64

batch\_size = 64

train\_datagen = ImageDataGenerator(rescale=1/255,

                                  rotation\_range=40,

                                  width\_shift\_range=0.2,

                                  height\_shift\_range=0.2,

                                  shear\_range=0.2,

                                  zoom\_range=0.2,

                                  horizontal\_flip=True,

                                  fill\_mode='nearest')

valid\_datagen = ImageDataGenerator(rescale = 1/255)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode='grayscale',

                                                    batch\_size=batch\_size,

                                                    class\_mode='categorical',

                                                    shuffle=True)

validation\_generator = valid\_datagen.flow\_from\_directory(test\_dir,

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode='grayscale',

                                                    batch\_size=batch\_size,

                                                    class\_mode='categorical',

                                                    shuffle=False)

#ANN Model

model = Sequential()

model.add(Conv2D(64,(3,3),padding='same',input\_shape=(64,64,1)))

model.add(Flatten())

model.add(Dense(2048, input\_dim=8, activation='relu'))

model.add(Dense(1024, input\_dim=8, activation='relu'))

model.add(Dense(1024, input\_dim=8, activation='relu'))

model.add(Dense(128, activation='relu'))

model.add(Dense(4, activation='softmax'))

opt = Adam(lr=0.0005)

model.compile(optimizer=opt,loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| Layer (type) Output Shape Param # |
| ================================================================= |
| conv2d\_1 (Conv2D) (None, 64, 64, 64) 640 |
|  |
| flatten\_2 (Flatten) (None, 262144) 0 |
|  |
| dense\_9 (Dense) (None, 2048) 536872960 |
|  |
| dense\_10 (Dense) (None, 1024) 2098176 |
|  |
| dense\_11 (Dense) (None, 1024) 1049600 |
|  |
| dense\_12 (Dense) (None, 128) 131200 |
|  |
| dense\_13 (Dense) (None, 4) 516 |
|  |

Total params: 540,153,092

Trainable params: 540,153,092

Non-trainable params: 0

#model fitting

epochs = 15

steps\_per\_epoch = train\_generator.n//train\_generator.batch\_size

validation\_steps = validation\_generator.n//validation\_generator.batch\_size

checkpoint = ModelCheckpoint('model\_weights.h5',monitor='val\_accuracy',

                            save\_weights\_only=True,mode='max',verbose=1)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',factor=0.1,patience=2,min\_lr=0.00001,mode='auto')

callbacks = [checkpoint, reduce\_lr]

history = model.fit(x=train\_generator,

                   steps\_per\_epoch=steps\_per\_epoch,

                   epochs=epochs,

                   validation\_data=validation\_generator,

                   validation\_steps=validation\_steps,

                   callbacks = callbacks )

accuracy: 41.5%

* + 1. **Convolutional neural network (CNN)**

**Code:**

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#import utils

import os

%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D

from tensorflow.keras.layers import BatchNormal ization, Activation, MaxPooling2D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

#from tensorflow.keras.utils import plot\_model

from IPython.display import SVG, Image

#from livelossplot import PlotLossesTensorFlowKeras()

import tensorflow as tf

base\_dir = '/dataset' #insect data set is present in dataset folder

train\_dir = os.path.join(base\_dir,'train')

test\_dir = os.path.join(base\_dir,'test')

#data augmentation

img\_size = 64

batch\_size = 64

datagen\_train = ImageDataGenerator(horizontal\_flip=True)

train\_generator = datagen\_train.flow\_from\_directory(train\_dir,

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode='grayscale',

                                                    batch\_size=batch\_size,

                                                    class\_mode='categorical',

                                                    shuffle=True)

datagen\_validation = ImageDataGenerator(horizontal\_flip=True)

validation\_generator = datagen\_validation.flow\_from\_directory(test\_dir,

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode='grayscale',

                                                    batch\_size=batch\_size,

                                                    class\_mode='categorical',

                                                    shuffle=False)

#CNN model

model = Sequential()

#1 - conv

model.add(Conv2D(64,(3,3),padding='same',input\_shape=(64,64,1)))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

#2 - conv

model.add(Conv2D(128,(5,5),padding='same'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

#flatten

model.add(Flatten())

model.add(Dense(256))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.25))

model.add(Dense(512))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.25))

model.add(Dense(4,activation='softmax'))

opt = Adam(lr=0.0005)

model.compile(optimizer=opt,loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| Layer (type) Output Shape Param # |
|  |
| conv2d\_16 (Conv2D) ( None, 64, 64, 64) 640 |
|  |
| batch\_normalization\_32 (Bat ( None, 64, 64, 64) 256 |
| chNormalization) |
|  |
| activation\_32 (Activation) (None, 64, 64, 64) 0 |
|  |
| max\_pooling2d\_16 (MaxPoolin (None, 32, 32, 64) 0 |
| g2D) |
|  |
| dropout\_32 (Dropout) (None, 32, 32, 64) 0 |
|  |
| conv2d\_17 (Conv2D) (None, 32, 32, 128) 204928 |
|  |
| batch\_normalization\_33 (Bat (None, 32, 32, 128) 512 |
| chNormalization) |
|  |
| activation\_33 (Activation) (None, 32, 32, 128) 0 |
|  |
| max\_pooling2d\_17 (MaxPoolin (None, 16, 16, 128) 0 |
| g2D) |
|  |
| dropout\_33 (Dropout) (None, 16, 16, 128) 0 |
|  |
| flatten\_8 (Flatten) (None, 32768) 0 |
|  |
| dense\_24 (Dense) (None, 256) 8388864 |
|  |
| batch\_normalization\_34 (Bat (None, 256) 1024 |
| chNormalization) |
|  |
| activation\_34 (Activation) (None, 256) 0 |
|  |
| dropout\_34 (Dropout) (None, 256) 0 |
|  |
| dense\_25 (Dense) (None, 512) 131584 |
|  |
| batch\_normalization\_35 (Bat (None, 512) 2048 |
| chNormalization) |
|  |
| activation\_35 (Activation) (None, 512) 0 |
|  |
| dropout\_35 (Dropout) (None, 512) 0 |
|  |
| dense\_26 (Dense) (None, 4) 2052 |
|  |

Total params: 8,731,908

Trainable params: 8,729,988

Non-trainable params: 1,920

**#model fitting**

epochs = 15

steps\_per\_epoch = train\_generator.n//train\_generator.batch\_size

validation\_steps = validation\_generator.n//validation\_generator.batch\_size

checkpoint = ModelCheckpoint('model\_weights.h5',monitor='val\_accuracy',

                            save\_weights\_only=True,mode='max',verbose=1)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss',factor=0.1,patience=2,min\_lr=0.00001,mode='auto')

callbacks = [checkpoint, reduce\_lr]

history = model.fit(x=train\_generator,

                   steps\_per\_epoch=steps\_per\_epoch,

                   epochs=epochs,

                   validation\_data=validation\_generator,

                   validation\_steps=validation\_steps,

                   callbacks = callbacks )

**accuracy= 92%**

**5.5.3 WEB APP:**

Among ANN and CNN, CNN gives the best accuracy and performance. So the weights obtained from cnn is used in web app to recognize the image.

Web Pages:

home.html

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<title>Upload Image</title>

</head>

<body>

<div class="col-lg">

<h1><span class="badge badge-danger">Insect Detection</span></h1>

</div>

<div class="col-lg" style="border:thin">

<form action = "/predict" method = "post" enctype="multipart/form-data">

<input type="file" name="file" align="center"/>

<br>

<br>

<br>

<input type = "submit" value="Upload">

</form>

</div>

</body>

</html>

**predict.hmtl**

<!DOCTYPE html>

<html>

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<title>Quality Check</title>

</head>

<body>

<div class="container">

<div class="row">

<div class="col-sm">

<h1>Category <span class="badge badge-secondary">{{product}}</span></h1>

</div></div></div>

<br>

</head>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

</body>

</html>

**“\_\_init\_\_.py” for running web app**

import tensorflow as tf

from tensorflow import keras

from flask import Flask, redirect, url\_for, render\_template, request, flash

from flask import Flask

from tensorflow.keras.preprocessing import image

import numpy as np

import cv2

import os

print(tf.\_\_version\_\_)

model = keras.models.load\_model('model\_weights.h5')

def prepare(filepath):

IMG\_SIZE = 64

img\_array = cv2.imread(filepath, cv2.IMREAD\_GRAYSCALE)

new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))

return new\_array.reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)

def prediction(img\_path):

img = image.load\_img(img\_path, target\_size=(64, 64,1))

img\_array = image.img\_to\_array(img)

prediction = model.predict([prepare(img\_path)])

return prediction

app = Flask(\_\_name\_\_)

#get\_model()

@app.route("/", methods=['GET', 'POST'])

def home():

return render\_template('home.html')

@app.route("/predict", methods = ['GET','POST'])

def predict():

if request.method == 'POST':

file = request.files['file']

filename = file.filename

file\_path = os.path.join(r'E:\Data set\static', filename) #slashes should be handeled properly

file.save(file\_path)

print(filename)

print(file\_path)

product = prediction(file\_path)

classes = {1:'ladha',2:'leaf',3:'black pest'}

classes\_x=np.argmax(product,axis=1)

product = classes[int(classes\_x)]

return render\_template('predict.html', product = product)

app.run()

**5.6 Results:**

After running the ANN model we have got 41.5% accuracy and for CNN model we have got 92%accuracy

**5.6.1 ANN model:**

Epoch 1/15

20/20 [==============================] - ETA: 0s - loss: 7.1101 - accuracy: 0.4109

Epoch 1: saving model to model\_weights.h5

20/20 [==============================] - 147s 7s/step - loss: 7.1101 - accuracy: 0.4109 - val\_loss: 2.2769 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 2/15

20/20 [==============================] - ETA: 0s - loss: 1.2544 - accuracy: 0.4683

Epoch 2: saving model to model\_weights.h5

20/20 [==============================] - 149s 7s/step - loss: 1.2544 - accuracy: 0.4683 - val\_loss: 1.3257 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 3/15

20/20 [==============================] - ETA: 0s - loss: 1.0027 - accuracy: 0.4683

Epoch 3: saving model to model\_weights.h5

20/20 [==============================] - 143s 7s/step - loss: 1.0027 - accuracy: 0.4683 - val\_loss: 1.2200 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 4/15

20/20 [==============================] - ETA: 0s - loss: 0.9900 - accuracy: 0.5035

Epoch 4: saving model to model\_weights.h5

20/20 [==============================] - 150s 7s/step - loss: 0.9900 - accuracy: 0.5035 - val\_loss: 1.1859 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 5/15

20/20 [==============================] - ETA: 0s - loss: 1.0049 - accuracy: 0.4746

Epoch 5: saving model to model\_weights.h5

20/20 [==============================] - 149s 7s/step - loss: 1.0049 - accuracy: 0.4746 - val\_loss: 1.3327 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 6/15

20/20 [==============================] - ETA: 0s - loss: 0.9781 - accuracy: 0.5113

Epoch 6: saving model to model\_weights.h5

20/20 [==============================] - 145s 7s/step - loss: 0.9781 - accuracy: 0.5113 - val\_loss: 1.1132 - val\_accuracy: 0.3594 - lr: 5.0000e-04

Epoch 7/15

20/20 [==============================] - ETA: 0s - loss: 0.9629 - accuracy: 0.5496

Epoch 7: saving model to model\_weights.h5

20/20 [==============================] - 145s 7s/step - loss: 0.9629 - accuracy: 0.5496 - val\_loss: 1.0250 - val\_accuracy: 0.3906 - lr: 5.0000e-04

Epoch 8/15

20/20 [==============================] - ETA: 0s - loss: 0.9717 - accuracy: 0.5293

Epoch 8: saving model to model\_weights.h5

20/20 [==============================] - 156s 8s/step - loss: 0.9717 - accuracy: 0.5293 - val\_loss: 1.0586 - val\_accuracy: 0.3750 - lr: 5.0000e-04

Epoch 9/15

20/20 [==============================] - ETA: 0s - loss: 0.9557 - accuracy: 0.5504

Epoch 9: saving model to model\_weights.h5

20/20 [==============================] - 147s 7s/step - loss: 0.9557 - accuracy: 0.5504 - val\_loss: 1.3406 - val\_accuracy: 0.3984 - lr: 5.0000e-04

Epoch 10/15

20/20 [==============================] - ETA: 0s - loss: 0.9511 - accuracy: 0.5653

Epoch 10: saving model to model\_weights.h5

20/20 [==============================] - 153s 8s/step - loss: 0.9511 - accuracy: 0.5653 - val\_loss: 1.0832 - val\_accuracy: 0.4219 - lr: 5.0000e-05

Epoch 11/15

20/20 [==============================] - ETA: 0s - loss: 0.9322 - accuracy: 0.5668

Epoch 11: saving model to model\_weights.h5

20/20 [==============================] - 146s 7s/step - loss: 0.9322 - accuracy: 0.5668 - val\_loss: 1.0886 - val\_accuracy: 0.4141 - lr: 5.0000e-05

Epoch 12/15

20/20 [==============================] - ETA: 0s - loss: 0.9186 - accuracy: 0.5809

Epoch 12: saving model to model\_weights.h5

20/20 [==============================] - 158s 8s/step - loss: 0.9186 - accuracy: 0.5809 - val\_loss: 1.0874 - val\_accuracy: 0.4141 - lr: 1.0000e-05

Epoch 13/15

20/20 [==============================] - ETA: 0s - loss: 0.9206 - accuracy: 0.5731

Epoch 13: saving model to model\_weights.h5

20/20 [==============================] - 146s 7s/step - loss: 0.9206 - accuracy: 0.5731 - val\_loss: 1.0846 - val\_accuracy: 0.4141 - lr: 1.0000e-05

Epoch 14/15

20/20 [==============================] - ETA: 0s - loss: 0.9208 - accuracy: 0.5754

Epoch 14: saving model to model\_weights.h5

20/20 [==============================] - 149s 7s/step - loss: 0.9208 - accuracy: 0.5754 - val\_loss: 1.0865 - val\_accuracy: 0.4141 - lr: 1.0000e-05

Epoch 15/15

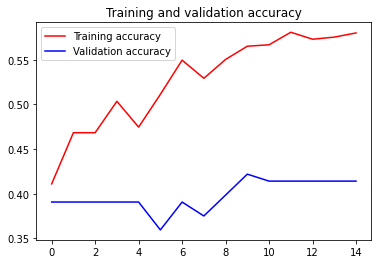
20/20 [==============================] - ETA: 0s - loss: 0.9175 - accuracy: 0.5801

Epoch 15: saving model to model\_weights.h5

20/20 [==============================] - 155s 8s/step - loss: 0.9175 - accuracy: 0.5801 - val\_loss: 1.0817 - val\_accuracy: 0.4141 - lr: 1.0000e-05

**Graph**

Below graph represents the change in accuracy for each and every epoch



epochs

acurracy

Graph 5.1 change in accuracy of ANN model

**5.6.2 Cnn Model**

Epoch 1/15

20/20 [==============================] - ETA: 0s - loss: 0.0301 - accuracy: 0.9937

Epoch 1: saving model to model\_weights.h5

20/20 [==============================] - 58s 3s/step - loss: 0.0301 - accuracy: 0.9937 - val\_loss: 5.5483 - val\_accuracy: 0.2344 - lr: 5.0000e-04

Epoch 2/15

20/20 [==============================] - ETA: 0s - loss: 0.0146 - accuracy: 0.9992

Epoch 2: saving model to model\_weights.h5

20/20 [==============================] - 58s 3s/step - loss: 0.0146 - accuracy: 0.9992 - val\_loss: 0.9817 - val\_accuracy: 0.6719 - lr: 5.0000e-04

Epoch 3/15

20/20 [==============================] - ETA: 0s - loss: 0.0111 - accuracy: 0.9977

Epoch 3: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0111 - accuracy: 0.9977 - val\_loss: 0.8678 - val\_accuracy: 0.6406 - lr: 5.0000e-04

Epoch 4/15

20/20 [==============================] - ETA: 0s - loss: 0.0108 - accuracy: 0.9977

Epoch 4: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0108 - accuracy: 0.9977 - val\_loss: 0.2765 - val\_accuracy: 0.9297 - lr: 5.0000e-04

Epoch 5/15

20/20 [==============================] - ETA: 0s - loss: 0.0105 - accuracy: 0.9977

Epoch 5: saving model to model\_weights.h5

20/20 [==============================] - 58s 3s/step - loss: 0.0105 - accuracy: 0.9977 - val\_loss: 0.2223 - val\_accuracy: 0.9062 - lr: 5.0000e-04

Epoch 6/15

20/20 [==============================] - ETA: 0s - loss: 0.0091 - accuracy: 0.9984

Epoch 6: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0091 - accuracy: 0.9984 - val\_loss: 0.3204 - val\_accuracy: 0.9062 - lr: 5.0000e-04

Epoch 7/15

20/20 [==============================] - ETA: 0s - loss: 0.0063 - accuracy: 0.9992

Epoch 7: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0063 - accuracy: 0.9992 - val\_loss: 0.0351 - val\_accuracy: 0.9922 - lr: 5.0000e-04

Epoch 8/15

20/20 [==============================] - ETA: 0s - loss: 0.0059 - accuracy: 0.9992

Epoch 8: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0059 - accuracy: 0.9992 - val\_loss: 0.0938 - val\_accuracy: 0.9375 - lr: 5.0000e-04

Epoch 9/15

20/20 [==============================] - ETA: 0s - loss: 0.0072 - accuracy: 0.9992

Epoch 9: saving model to model\_weights.h5

20/20 [==============================] - 56s 3s/step - loss: 0.0072 - accuracy: 0.9992 - val\_loss: 0.1148 - val\_accuracy: 0.9609 - lr: 5.0000e-04

Epoch 10/15

20/20 [==============================] - ETA: 0s - loss: 0.0126 - accuracy: 0.9945

Epoch 10: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0126 - accuracy: 0.9945 - val\_loss: 0.1348 - val\_accuracy: 0.9453 - lr: 5.0000e-05

Epoch 11/15

20/20 [==============================] - ETA: 0s - loss: 0.0044 - accuracy: 0.9984

Epoch 11: saving model to model\_weights.h5

20/20 [==============================] - 56s 3s/step - loss: 0.0044 - accuracy: 0.9984 - val\_loss: 0.3457 - val\_accuracy: 0.9141 - lr: 5.0000e-05

Epoch 12/15

20/20 [==============================] - ETA: 0s - loss: 0.0032 - accuracy: 0.9984

Epoch 12: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0032 - accuracy: 0.9984 - val\_loss: 0.3273 - val\_accuracy: 0.9141 - lr: 1.0000e-05

Epoch 13/15

20/20 [==============================] - ETA: 0s - loss: 0.0063 - accuracy: 0.9984

Epoch 13: saving model to model\_weights.h5

20/20 [==============================] - 56s 3s/step - loss: 0.0063 - accuracy: 0.9984 - val\_loss: 0.3411 - val\_accuracy: 0.9062 - lr: 1.0000e-05

Epoch 14/15

20/20 [==============================] - ETA: 0s - loss: 0.0033 - accuracy: 1.0000

Epoch 14: saving model to model\_weights.h5

20/20 [==============================] - 56s 3s/step - loss: 0.0033 - accuracy: 1.0000 - val\_loss: 0.3325 - val\_accuracy: 0.9062 - lr: 1.0000e-05

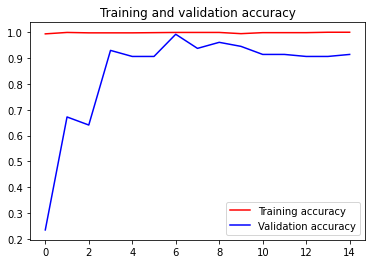
Epoch 15/15

20/20 [==============================] - ETA: 0s - loss: 0.0039 - accuracy: 1.0000

Epoch 15: saving model to model\_weights.h5

20/20 [==============================] - 57s 3s/step - loss: 0.0039 - accuracy: 1.0000 - val\_loss: 0.2615 - val\_accuracy: 0.9141 - lr: 1.0000e-05

Below graph represents the change in accuracy for each and every epoch



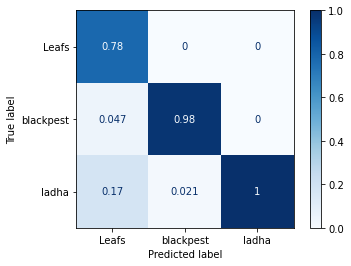
Epochs

accuracy

**Graph: change in accuracy for every epoch (CNN model)**

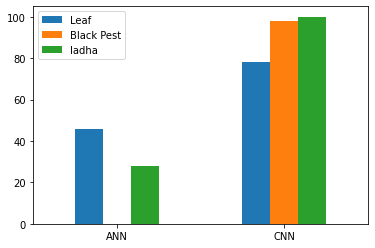
**Confusion matrix**

**This matrix represents the accuracy for each and every class**

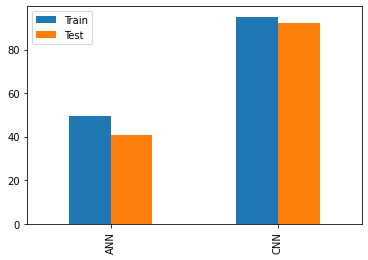
 **Table 5.7 classwise accuracy**

**5.7 ANN Vs CNN**

**Classwise accuracy comparsion for both ANN and CNN model**



**Graph 5.3: classwise accuracy (CNN & ANN model)**

****

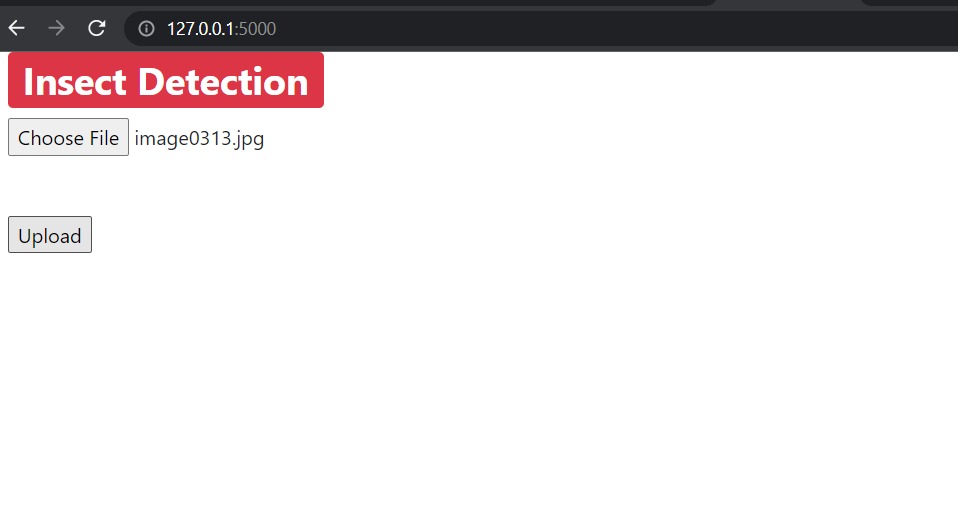
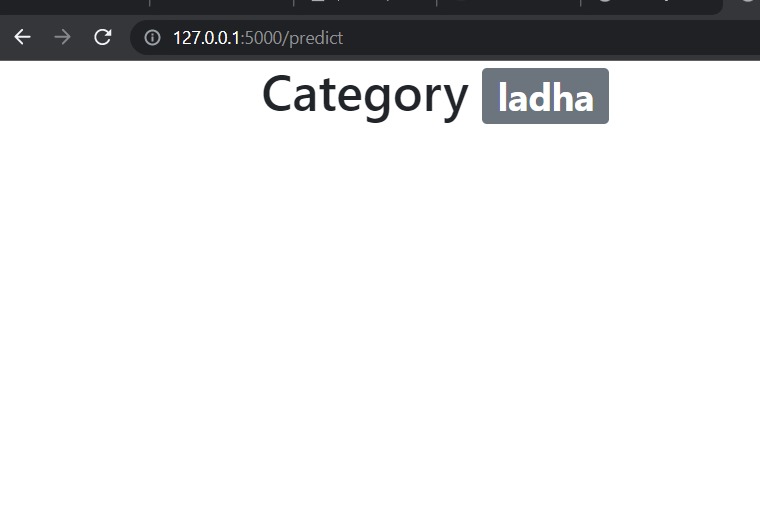
Graph 5.4 accuracy of CNN and ANN model

After comparing the results of CNN and ANN, we are concluded that CNN algorithm is working better for the insect detection, especially for our project. So we used the weights obtained from CNN model to build the web and detect the image.

**Web APP**

In this project we have also build a web app which accepts an image from the user and detect if any insect or pest present in the leaf.

This web app is developed by using Flask web development framework.

Figure 5.3

**Code:**

**home.html**

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<title>Upload Image</title>

</head>

<body>

<div class="col-lg">

<h1><span class="badge badge-danger">Insect Detection</span></h1>

</div>

<div class="col-lg" style="border:thin">

<form action = "/predict" method = "post" enctype="multipart/form-data">

<input type="file" name="file" align="center"/>

<br>

<br>

<br>

<input type = "submit" value="Upload">

</form>

</div>

</body>

</html>

**predict.html**

<!DOCTYPE html>

<html>

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<title>Quality Check</title>

</head>

<body>

<div class="container">

<div class="row">

<div class="col-sm">

<h1>Category <span class="badge badge-secondary">{{product}}</span></h1>

</div></div></div>

<br>

</head>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

</body>

</html>

**\_\_init\_\_.py**

import tensorflow as tf

from tensorflow import keras

from flask import Flask, redirect, url\_for, render\_template, request, flash

from flask import Flask

from tensorflow.keras.preprocessing import image

import numpy as np

import cv2

import os

print(tf.\_\_version\_\_)

model = keras.models.load\_model('model\_weights.h5')

def prepare(filepath):

IMG\_SIZE = 64

img\_array = cv2.imread(filepath, cv2.IMREAD\_GRAYSCALE)

new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))

return new\_array.reshape(-1, IMG\_SIZE, IMG\_SIZE, 1)

def prediction(img\_path):

img = image.load\_img(img\_path, target\_size=(64, 64,1))

img\_array = image.img\_to\_array(img)

prediction = model.predict([prepare(img\_path)])

return prediction

app = Flask(\_\_name\_\_)

#get\_model()

@app.route("/", methods=['GET', 'POST'])

def home():

return render\_template('home.html')

@app.route("/predict", methods = ['GET','POST'])

def predict():

if request.method == 'POST':

file = request.files['file']

filename = file.filename

file\_path = os.path.join(r'E:\Data set\static', filename) #slashes should be handeled properly

file.save(file\_path)

print(filename)

print(file\_path)

product = prediction(file\_path)

classes = {1:'ladha',2:'leaf',3:'black pest'}

classes\_x=np.argmax(product,axis=1)

product = classes[int(classes\_x)]

return render\_template('predict.html', product = product)

app.run()

**Directory Structure**

/

\_\_init\_\_.py

model\_weights.h5

static

home.html

predict.html

**Chapter 6**

**Conclusions and Future work**

**6.1 Conclusion:**

Our project main objective is to detect Ladha, Black Pest and Leaf present in an image which is given as input to web app. We can also increase the scope of this project by adding some more classes too. This project will helpful to the farmer to know the type of the insect present on their crop. So that they can use corresponding methodologies to get rid of them from the field if they are harmful to the crop. For this project we have collected 1503 images of ladha, Black pest and normal leaf from the field of cotton and chili in Andhra Pradesh and Telangana.

**6.2 Future Work:**

In the future, we are trying to extend the scope of our project by adding more data sets and different insects & pests classes from different crops. We are planning to collect more data sets. We will include some other insects & pest types in our project. So the our project will detect some more classes too. This project can also embedded a moving camera which will take images continuously in the field and detect the insects & pests if they are present. By this process, farmers will get an alert message whenever this moving camera detects an insect or pest in the field.

**Bibiliography**

[1] Shen Y, Zhou H, Li J, Jian F, Jayas DS. Detection of stored-grain insects using deep learning. Comput Electron Agric 2018;145:319–25.

[2] Bhargava A, Bansal A. Fruits and vegetables quality evaluation using computer vision: A review. J King Saud Univ Comput Inf Sci 2018. <https://doi.org/10.1016/j>. jksuci.2018.06.002 [in press].

[3] Yaakob SN, Jain L. An insect classification analysis based on shape features using quality threshold ARTMAP and moment invariant. Appl Intell 2012;37(1):12–30.

[4] Yue Yi, Cheng Xi, Zhang Di, Wu Y, Zhao Y, Chen Y, Fan G, Zhang Y. Deep recursive super resolution network with Laplacian Pyramid for better agricultural pest surveillance and detection. Comput Electron Agric 2018;150:26–32.

[5] Cheng X, Zhang Y, Chen Y, Wu Y, Yue Y. Pest identification via deep residual learning in complex background. Comput Electron Agric 2017;141:351–6.

[6] Xie C, Wang R, Zhang J, Chen P, Dong W, Li R, Chen T, Chen H. Multi-level learning features for automatic classification of field crop pests. Comput Electron Agric 2018;152:233–41.

[7] Nanni L, Maguolo G, Pancino F. Research on insect pest image detection and recognition based on bio inspired methods; 2019. arXiv: 1910.00296.

[8] Deng L, Wang Z, Wang C, He Y, Huang T, Dong Y, et al. Application of agricultural insect pest detection and control map based on image processing analysis. J Intell Fuzzy Syst 2020;38:379–89.

[9] Garcia J, Pope C, Altimiras F. A distributed-means segmentation algorithm applied to lobesia botrana

recognition. Complexity 2017;14.

[10] Bakkay MC, Chambon S, Rashwan HA, Lubat C, Barsotti S. Automatic detection of individual and touching moths from trap images by combining contour-based and region-based segmentation. IET Comput Vis 2017;12:138–45.

[11] Wang J, Lin C, Ji L, Liang A. A new automatic identification system of insect images at the order level. Knowl-Based Syst 2012;33:102–10.

[12] Xie C, Zhang J, Li R, Li J, Hong P, Xia J, Chen P. Automatic classification for field crop insects via multiple-task sparse representation and multiple-kernel learning. Comput Electron Agric 2015;119:123–32.

[13] Yang H-P, Ma C-S, Wen H, Zhan Q-B, Wang X-L. A tool for developing an automatic insect identification system based on wing outlines. Sci Rep 2015;5(1):1–11.

[14] Coates A, Ng AY. Selecting receptive fields in deep networks. Adv Neural Inf Process Syst 2011:2528–36.

[15] Lucas B, Shifaz A, Pelletier C, O’Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI. Proximity Forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Disc 2019;33(3):607–35.

[16] Liu J-e, An F-P. Image classification algorithm based on deep learning-kernel function. Sci Program 2020;2020.

[17] Nanni L, Brahnam S, Ghidoni S, Maguolo G. General purpose (GenP) bioimage ensemble of handcrafted and learned features with data augmentation; 2019. arXiv: 1904.08084.

[18] Makandar A, Halalli B. Image enhancement techniques using highpass and lowpass filters. Int J Comput Appl 2015;109:12–5.

[19] Hambal AM, Pei Z, Ishabailu FL. Image noise reduction and filtering techniques. Int J Sci Res 2017;6:2033–8.

[20] Wen C, Guyer D. Image-based orchard insect automated identification and classification method. Comput Electron Agric 2012;89:110–5.

[21] Mikołajczyk A, Grochowski M. Data augmentation for improving deep learning in image classification problem. In: Proc IIPhDW’ 18 proceedings of the 2018 international interdisciplinary PhD workshop. Swinoujscie, Poland; 2018. p. 117–22.

[22] Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. J Big Data 2019;6(1):60.

[23] Thenmozhi K, Reddy US. Image processing techniques for insect shape detection in field crops. In: IEEE 2017 international conference on inventive computing and informatics. Coimbatore, India; 2017; p. 699–704.

[24] Espinoza K, Valera DL, Torres JA, Lo´pez A, Molina-Aiz FD. Combination of image processing and artificial neural networks as a novel approach for the identification of Bemisia tabaci and Frankliniella occidentalis on sticky traps in greenhouse agriculture. Comput Electron Agric 2016;127:495–505.

[25] Asefpour Vakilian K, Massah J. Performance evaluation of a machine vision system for insect pests identification of field crops using artificial neural networks. Arch Phytopathol Pflanzenschutz 2013;46(11):1262–9.

[26] Fuchida M, Pathmakumar T, Mohan RE, Tan N, Nakamura A. Vision-based perception and classification of mosquitoes using support vector machine. Appl Sci. 2017;7:51.

[27] Favret C, Sieracki JM. Machine vision automated species identification scaled towards production levels. Syst Entomol 2016;41:133–43.

[28] Li F, Xiong Y. Automatic identification of butterfly species based on HoMSC and GLCMoIB. Vis Comput 2018;34 (11):1525–33.

[29] Antony JC, Pratheepa M. A Bayesian classification approach for predicting Gesonia gemma Swinhoe population on soybean crop in relation to abiotic factors based on economic threshold level. JBC 2018;32:68–73.

[30] Martineau M, Conte D, Raveaux R, Arnault I, Munier D, Venturini G. A survey on image-based insect classification. Pattern Recogn 2017;65:273–84.

[31] Zhu LQ, Ma MY, Zhang Z, Zhang PY, Wu W, Wang DD, et al. Hybrid deep learning for automated lepidopteran insect image classification. Orient Insects 2017;51:79–91.

[32] Martins VA, Freitas LC, de Aguiar MS, de Brisolara LB, Ferreira PR. Deep learning applied to the identification of fruit fly in intelligent traps. In: Proc SBESC 19 IX Brazilian symposium on computing systems engineering. Rio Grande do Norte, Brazil. IEEE; 2019. p. 1–8.

[33] Sharma P, Berwal YP, Ghai W. Performance analysis of deep learning CNN models for disease detection in plants using image segmentation. Inf Process Agric 2019. <https://doi.org/> 10.1016/j.inpa.2019.11.001 [in press].

[34] Deng L, Wang Y, Han Z, Yu R. Research on insect pest image detection and recognition based on bio-inspired methods. Biosyst Eng 2018;169:139–48.

[35] A large-scale benchmark dataset for insect pest recognition; 2019. Link: https://github.com/xpwu95/IP102.

[36] Rother C, Kolmogorov V, Blake A. ‘‘GrabCut” interactive foreground extraction using iterated graph cuts. ACM T Graphic 2004;23:309–14.

[37] Mantovani RG, Rossi AL, Vanschoren J, Bischl B, De Carvalho AC. Effectiveness of random search in SVM hyper-parameter tuning. In: Proc IJCNN’ 15 international joint conference on neural networks, Killarney, Ireland; 2015. p. 1–8.

[38] Cheng X, Zhang YH, Wu YZ, Yue Y. Agricultural pests tracking and identification in video surveillance based on deep learning. In: Proc Part III ICIC ’17 international conference on intelligent computing. Liverpool, UK; 2017. p. 58–70.

[39] Nanni L, Maguolo G, Pancino F. Insect pest image detection and recognition based on bio-inspired methods. Ecol Inform. 2020;101089.

[40] Abdullah MA, Dlay SS, Woo WL, Chambers JA. Robust iris segmentation method based on a new active contour force with a noncircular normalization. IEEE transactions on systems, man, and cybernetics. Systems 2016;47:3128–41