INSECTS & WEED DETECTION FROM RICE CROP USING FASTER R-CNN

BY

RADHE RAMAN TIWARI

Student at IIITMANIPUR

Supervised by

Dr. SHRIVISHAL TRIPATHI

Assistant Professor, ECE at IIITNaya Raipur

THESIS

SUBMITTED TO

DR. SHYAMA PRASAD MUKHERJEE INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY,NAYA RAIPUR

Summer Internship May - July 2K19

JULY, 2K19

Abstract

As the increase in the world population the demand of the rice is also increases. In order to increase the growth of rice in the rice crop it is necessary to detect the weed and insects in the rice crop to minimize the growth of weed and insects so that the growth of the rice can be increased.Insect and Weed detection is the important factor to be analyzed. Unmanned Air Vehicle (UAV) is used for data acquisition of rice crop in different phases and states so that high quality of RGB images can be captured. In which we have taken 15 different types of rice crop insects species images and different phases of weed images to train the model. The proposed method facilitates the extraction of weed and insects into the rice crop field using deep learning concept faster region-base convolutional neural networks(Faster R-CNNs) it is implemented using Python3 with the help of Tensorflow API. The result shows that Faster R-CNN method is the state of arts method for detection and classification of weed and insects with good accuracy rate.

Keywords - Computer vision, Deep learning, Weed detection, Insects detection, Rice detection, Convolutional neural networks, Region based convolutional neural networks

Declaration

I declare that this submission represents my idea in my own words and where others' idea or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/sources in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from proper permission has not been taken when needed.

Signatur

Radhe Raman Tiwari

Date:

Student at IIITMANIPUR

Acknowledgement

I extend my sincere thanks to the institute **Dr. Shyama Prasad Mukherjee International Institute of Information Technology Naya Raipur** who has provided me an opportunity to do this research project. I would like to express deep dept to Dr. Shrivishal Tripathi ,Project guide for his vital suggestion ,meticulous guidance and constant motivation which went a long way in successful completion of the project. I cannot move forward without thank to Dr. Pradeep Kumar Sinha ,Vice Chancellor & Director, IIIT-NR who has always been an inspiration to me. I would like to thanks Dr. Ramakrishna Bandi ,Internship co-ordinator who has been provide an opportunity to express my knowledge & dedication towards my work . On a moral personal note my deepest appreciation and gratitude to my beloved friends ,who has been an inspiration and have provided me a unrelenting encouragement and support

RADHE RAMAN TIWARI
Student at IIITMANIPUR

Contents

List of Figures

List of abbreviations

CPU	Central Processing Unit
GUP	Graphical Processing Unit
CNN	Convolutional Neural Network
R-CNN	Convolutional Neural Network with Region proposals
ROI	Region of Interest
RPN	Region Proposal Network
FC	Fully Connected (layer or network
FCN	Fully Convolutional Network
FPS	Frames Per Second
IPA	International Phonetics Alphabets
SVM	Support Vector Machine

Chapter 1

Introduction

Rice cultivation is one of the most important economical sectors for Indian economy. Hence, the need to develop technological tools to improve the management and productivity of the sector. One of the problems in rice fields is weed plants and rice insects, due to the fact that these plants compete for water and nutrients reducing up to 60-70% of the field yield if not control is carried out. Generally, it can be considered that critical periods for competence in any cultivation with weed are at initial of growing and development states, for rice as found in this period is between 0 and 75 days. Therefore the identification and removal of this plants are very important for rice growers..

Recent advances in object detection are driven by the success of region proposal methods and faster region-based convolutional neural networks (Faster R-CNNs). Although region-based CNNs were computationally expensive as originally developed in, their cost has been drastically reduced thanks to sharing convolutions across proposals. The latest incarnation, Faster R-CNN, achieves near real-time rates using very deep networks, when ignoring the time spent on region proposals. Now, proposals are the test-time computational bottleneck in state-of-the-art detection systems. Region proposal methods typically rely on inexpensive features and economical inference schemes. Selective Search, one of the most popular methods, greedily merges superpixels based on engineered low-level features. Yet when compared to efficient detection networks, Selective Search is an order of magnitude slower, at 2 seconds per image in a CPU implementation. EdgeBoxes currently provides the best tradeoff between proposal quality and speed, at 0.2 seconds per image. Nevertheless, the region proposal step still consumes as much running time as the detection network. One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable.An obvious way to accelerate proposal computation is to re-implement it for the GPU.This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses important opportunities for sharing computation.

In this project we focus on the weed and insects identification and classification problem, therefore, we propose to use aerial images, captured from a UAV at 10 meters over ground, and faster region-based convolutional neural networks (Faster R-CNNs), which is a mature technique for pattern recognition and classification to detect weed plants and rice insects into a rice field.

The project is organized as follows, first, we present the methodology where we describe the location where the study was done, then, how the images are acquired and how the Faster R-CNN is trained, tested and validated. Afterward the results are shown and discussed to finally make conclusion.

1.1 Problem Statement

Objects contained in image files can be located and identified automatically. This is called object detection and is one of the basic problems of computer vision. As we will demonstrate,faster region-base convolutional neural networks(Faster R-CNN) are currently the state-of-the-art solution for object detection. The main task of this thesis is to detection and classification of rice crop insects and weed in rice crop field with the help of images and videos in real-time

1.2 Outline of the report

This report is organised around five main parts....

Chapter 1 Introduction to the project

Chapter 2 Background Study.

Chapter 3 Convolutional Object Detection

Chapter 4 Implementation and Setup

Chapter 5 Result Analysis & Conclusion

1.3 Gantt chart

For the completion of any work we need to plan and divide the work in some time frame.I divided my work according to the following Gantt Chart.

The figure 1.1 chart describes the time frame of simultaneous phases of the project. The first phase is the analysis phase, in which I completed the study of research papers,

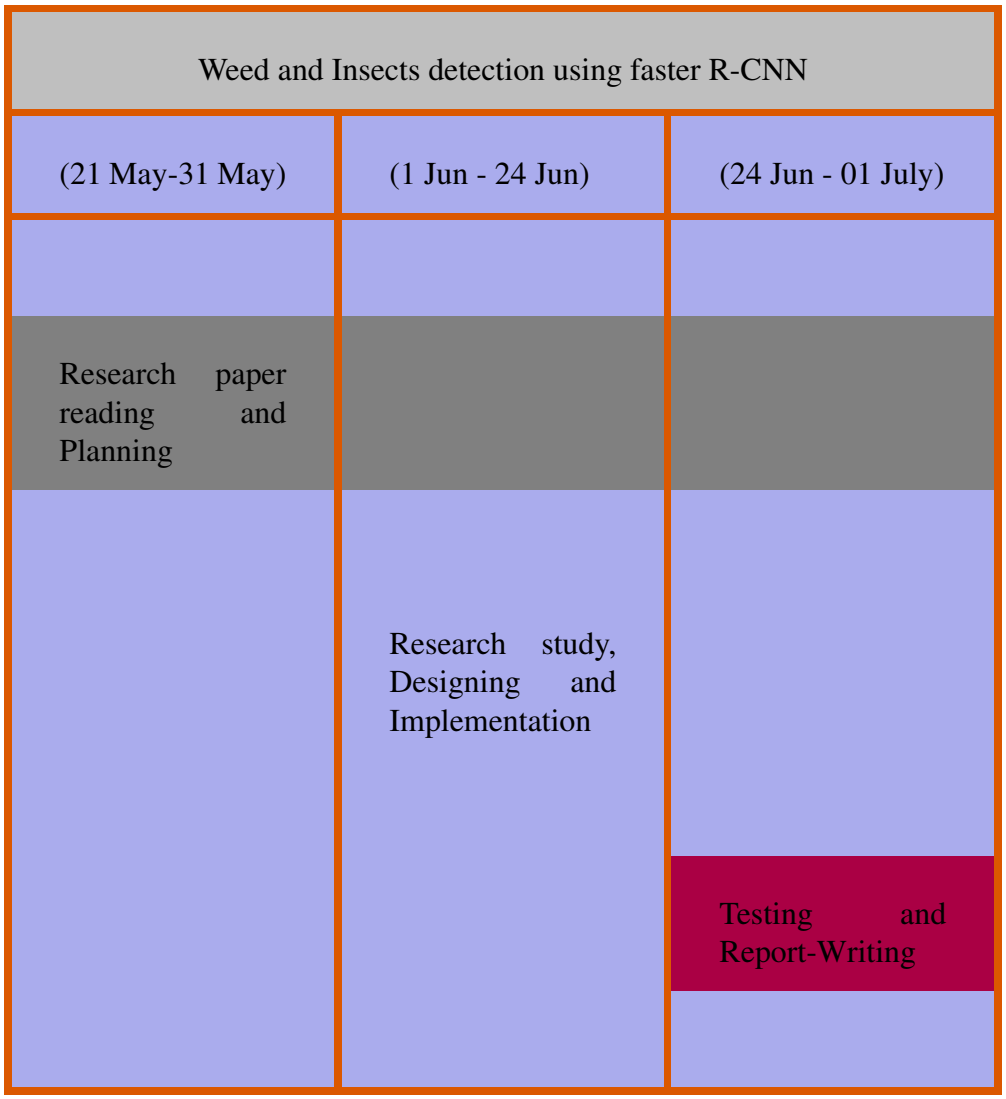

Figure 1.1: Gantt Chat

previous related works and their outcomes. The second phase is the design and implementation phase, in which I completed train of Faster R-CNN models with the help of rice crop insects and weed raw images data-set by Python3 programming language. The third phase is the testing and report-writing phase, in which I completed the testing of models with their predicted result and report-writing by Latex.

Chapter 2

Background Study

Outline: This chapter presents the following:

1. Introduction
2. Machine Learning
3. Neural Networks
4. Computer Vision
5. Convolutional neural networks

2.1 Introduction

In this chapter, we provide the theoretical background necessary for understanding the methods discussed in the next chapter. First, we discuss relevant details of machine learning, neural networks, and computer vision. Finally, we explain how these disciplines are combined in convolutional neural networks.

2.2 Machine Learning

Learning algorithms are widely used in computer vision applications. Before considering image related tasks, we are going to have a brief look at basics of machine learning.

Machine learning has emerged as a useful tool for modeling problems that are otherwise difficult to formulate exactly. Classical computer programs are explicitly programmed by hand to perform a task. With machine learning, some portion of the human contribution is replaced by a learning algorithm. As availability of computational capacity and data has increased, machine learning has become more and more practical over the years, to the point of being almost ubiquitous.

2.2.1 Types

A typical way of using machine learning is supervised learning. A learning algorithm is shown multiple examples that have been annotated or labeled by humans. For example, in the object detection problem we use training images where humans have marked the locations and classes of relevant objects. After learning from the examples, the algorithm is able to predict the annotations or labels of previously unseen data. Classification and regression are the most important task types. In classification, the algorithm attempts to predict the correct class of a new piece of data based on the training data. In regression, instead of discrete classes, the algorithm tries to predict a continuous output.

In unsupervised learning, the algorithm attempts to learn useful properties of the data without a human teacher telling what the correct output should be. Classical example of unsupervised learning is clustering. More recently, especially with the advent of deep learning technologies, unsupervised preprocessing has become a popular tool in supervised learning tasks for discovering useful representations of the data.

2.2.2 Features

Some kind of preprocessing is almost always needed. Preprocessing the data into a new, simpler variable space is called feature extraction. Often, it is impractical or impossible to use the full-dimensional training data directly. Rather, detectors are programmed to extract

interesting features from the data, and these features are used as input to the machine learning algorithm.

In the past, the feature detectors were often hand-crafted. The problem with this approach is that we do not always know in advance, which features are interesting. The trend in machine learning has been towards learning the feature detectors as well, which enables using the complete data.

2.2.3 Generalization

Since the training data cannot include every possible instance of the inputs, the learning algorithm has to be able to generalize in order to handle unseen data points. Too simple model estimate can fail to capture important aspects of the true model. On the other hand, too complex methods can overfit by modeling unimportant details and noise, which also leads to bad generalization. Typically, overfitting happens when a complex method is used in conjunction with too little training data. An overfitted model learns to model the known examples but does not understand what connects them.

The performance of the algorithm can be evaluated from the quality and quantity of errors. A loss function, such as mean squared error, is used to assign a cost to the errors. The objective in the training phase is to minimize this loss.

2.3 Neural Networks

Neural networks are a popular type of machine learning model. A special case of a neural network called the convolutional neural network (CNN) is the primary focus of this thesis. Before discussing CNNs, we will discuss how regular neural networks work.

2.3.1 Origins

Neural networks were originally called artificial neural networks, because they were developed to mimic the neural function of the human brain. Pioneering research includes the threshold logic unit by Warren McCulloch and Walter Pitts in 1943 and the perceptron by Frank Rosenblatt in 1957. Even though the inspiration from biology is apparent, it would be misleading to overemphasize the connection between artificial neurons and biological neurons or neuroscience. The human brain contains approximately 100 billion neurons operating in parallel. Artificial neurons are mathematical functions implemented on more-or-less serial computers. Research into neural networks is mostly guided by developments in engineering and mathematics rather than biology.

An artificial neuron based on the McCulloch-Pitts model is shown in Figure 2.1. The neuron k receives m input parameters x_j. The neuron also has m weight parameters w_{kj}. The weight parameters often include a bias term that has a matching dummy input with a

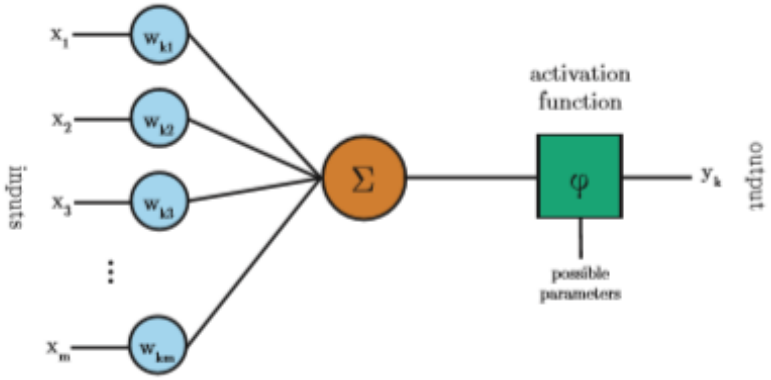

Figure 2.1: An artificial neuron.
link

fixed value of 1. The inputs and weights are linearly combined and summed. The sum is then fed to an activation function ϕ that produces the output y_k of the neuron:

$$y_k = \phi(s_k) = \phi(\sum_{j=0}^{m} w_{kj} x_j)$$

The neuron is trained by carefully selecting the weights to produce a desired output for each input.

2.3.2 Multi-layer networks

A neural network is a combination of artificial neurons. The neurons are typically grouped into layers. In a fully-connected feed-forward multi-layer network, shown in Figure 2.2, each output of a layer of neurons is fed as input to each neuron of the next layer. Thus, some layers process the original input data, while some process data received from other neurons. Each neuron has a number of weights equal to the number of neurons in the previous layer.

A multi-layer network typically includes three types of layers: an input layer, one or more hidden layers and an output layer. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activations to an output, such as a classification. A multilayer feed-forward network with at least one hidden layer can function as a universal approximator i.e. can be constructed to compute almost any function.

In this thesis, we will mostly discuss fully-connected networks and convolutional networks (see section 2.5). Convolutional networks utilize parameter sharing and have limited connections compared to fully-connected networks. Other network types, such as recurrent networks, are outside the scope of this thesis.

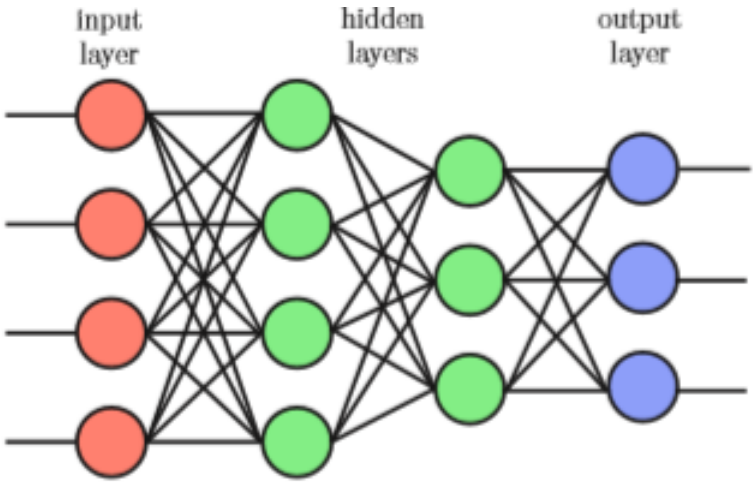

Figure 2.2: A fully-connected multi-layer neural network.
link

2.3.3 Back-propagation

A neural network is trained by selecting the weights of all neurons so that the network learns to approximate target outputs from known inputs. It is difficult to solve the neuron weights of a multi-layer network analytically. The back-propagation algorithm provides a simple and effective solution to solving the weights iteratively. The classical version uses gradient descent as optimization method. Gradient descent can be quite time-consuming and is not guaranteed to find the global minimum of error, but with proper configuration (known in machine learning as hyperparameters) works well enough in practice.

In the first phase of the algorithm, an input vector is propagated forward through the neural network. Before this, the weights of the network neurons have been initialized to some values, for example small random values. The received output of the network is compared to the desired output (which should be known for the training examples) using a loss function. The gradient of the loss function is then computed. This gradient is also called the error value. When using mean squared error as the loss function, the output layer error value is simply the difference between the current and desired output.

The error values are then propagated back through the network to calculate the error values of the hidden layer neurons. The hidden neuron loss function gradients can be solved using the chain rule of derivatives. Finally, the neuron weights are updated by calculating the gradient of the weights and subtracting a proportion of the gradient from the weights. This ratio is called the learning rate. The learning rate can be fixed or dynamic. After the weights have been updated, the algorithm continues by executing the phases again with different input until the weights converge. In the above description, we have described online learning that calculates the weight updates after each new input. Online learning can lead to “zig-zagging” behaviour, where the single data point estimate of the

gradient keeps changing direction and does not approach the minimum directly. Another way of computing the updates is full batch learning, where we compute the weight updates for the complete dataset. This is quite computationally heavy and has other drawbacks. A compromise version is mini-batch learning, where we use only some portion of the training set for each update. Mathematical descriptions of the algorithm are readily available in other works.

2.3.4 Deep learning

Modern neural networks are often called deep neural networks. Even though multi-layer neural networks have existed since the 1980s, several reasons prevented the effective training of networks with multiple hidden layers.

One of the main problems is the curse of dimensionality. As the number of variables increases, the number of different configurations of the variables grows exponentially. As the number of configurations increases, the number of training samples should increase in equal measure. Collecting a training dataset of sufficient size is time-consuming and costly or outright impossible.

Fortunately, real-world data is not uniformly distributed and often involves a structure, where the interesting information lies on a low-dimensional manifold. The manifold hypothesis assumes that most data configurations are invalid or rare. We can decrease dimensionality by learning to represent the data using the coordinates of the manifold. Another way to improve generalization is to assume local constancy. This means assuming that the function that the neural network learns to approximate should not change much within a small region.

In the past ten years, neural networks have had a renaissance, mainly because of the availability of more powerful computers and larger datasets. In early 2000s, it was discovered that neural networks could be trained efficiently using graphics processing units. GPUs are more efficient for the task than traditional CPUs and provide a relatively cheap alternative to specialist hardware. Today, researchers typically use high-end consumer graphic cards, such as NVIDIA Tesla K40.

Other more theoretical breakthroughs include replacing mean-squared error functions with cross-entropy based functions and replacing sigmoidal activation functions with rectified linear units.

With deep learning, there is less need for hand-tuned machine learning solutions that were used previously. A classical pattern detection system, for example, includes a hand-tuned feature detection phase before a machine learning phase. The deep learning equivalent consists of a single neural network. The lower layers of the neural network learn to recognize the basic features, which are then fed forward to higher layers of the network.

2.3.5 Activation function type

The activation function determines the final output of each neuron. It is important to select the function properly in order to create an effective network.

Early researchers found that perceptrons and other linear systems had severe drawbacks, being unable to solve problems that were not linearly separable, such as the XOR-problem. Sometimes, linear systems can solve these kinds of problems using hand-crafted feature detectors, but this is not the most advantageous use of machine learning. Simply adding layers does not help either, because a network composed of linear neurons remains linear no matter how many layers it has.

A light-weight and effective way of creating a non-linear network is using rectified linear units (ReLu). A rectified linear function generates the output using a ramp function such as:

$$\phi(s) = max(0, s)$$

.

This type of function is easy to compute and differentiate (for backpropagation). The function is not differentiable at zero, but this has not prevented its use in practice. ReLus have become quite popular lately, often replacing sigmoidal activation functions, which have smooth derivatives but suffer from gradient saturation problems and slower computation. For multi-class classification problems, the softmax activation function is used in the output layer of the network:

$$\phi(s) = \frac{exp(s_k)}{\sum_{k=1}^{K} exp(s_k)}$$

.

The softmax function takes a vector of K arbitrarily large values and outputs a vector of K values that range between 0...1 and sum to 1. The values output by the softmax unit can be utilized as class probabilities.

2.4 Computer vision

Next, we are going to discuss computer vision in general and explore the primary subject of this thesis, object detection, as a subproblem of computer vision.

2.4.1 Overview

Computer vision deals with the extraction of meaningful information from the contents of digital images or video. This is distinct from mere image processing, which involves manipulating visual information on the pixel level. Applications of computer vision include image classification, visual detection, 3D scene reconstruction from 2D images, image retrieval, augmented reality, machine vision and traffic automation.

Today, machine learning is a necessary component of many computer vision algorithms. Such algorithms can be described as a combination of image processing and machine learning. Effective solutions require algorithms that can cope with the vast amount of information contained in visual images, and critically for many applications, can carry out the computation in real time.

2.4.2 Object detection

Object detection is one of the classical problems of computer vision and is often described as a difficult task. In many respects, it is similar to other computer vision tasks, because it involves creating a solution that is invariant to deformation and changes in lighting and viewpoint. What makes object detection a distinct problem is that it involves both locating and classifying regions of an image. The locating part is not needed in, for example, whole image classification.

To detect an object, we need to have some idea where the object might be and how the image is segmented. This creates a type of chicken-and-egg problem, where, to recognize the shape (and class) of an object, we need to know its location, and to recognize the location of an object, we need to know its shape. Some visually dissimilar features, such as the clothes and face of a human being, may be parts of the same object, but it is difficult to know this without recognizing the object first. On the other hand, some objects stand out only slightly from the background, requiring separation before recognition.

Low-level visual features of an image, such as a saliency map, may be used as a guide for locating candidate objects. The location and size is typically defined using a bounding box, which is stored in the form of corner coordinates. Using a rectangle is simpler than using an arbitrarily shaped polygon, and many operations, such as convolution, are performed on rectangles in any case. The sub-image contained in the bounding box is then classified by an algorithm that has been trained using machine learning. The boundaries of the object can be further refined iteratively, after making an initial guess.

During the 2000s, popular solutions for object detection utilized feature descriptors, such as scale-invariant feature transform (SIFT) developed by David Lowe in 1999 and histogram of oriented gradients (HOG) popularized in 2005. In the 2010s, there has been a shift towards utilizing convolutional neural networks.

Before the widescale adoption of CNNs, there were two competing solutions for generating bounding boxes. In the first solution, a dense set of region proposals is generated and then most of these are rejected. This typically involves a sliding window detector. In the second solution, a sparse set of bounding boxes is generated using a region proposal method, such as Selective Search. Combining sparse region proposals with convolutional neural networks has provided good results and is currently popular.

2.5 Convolutional neural networks

Next, we are going to discuss why and how convolutional neural networks (CNN) are used and describe their history.

2.5.1 Justification

The problem with solving computer vision problems using traditional neural networks is that even a modestly sized image contains an enormous amount of information (see section 2.2.4 on deep learning and the curse of dimensionality).

A monochrome 620x480 image contains 297 600 pixels. If each pixel intensity of this image is input separately to a fully-connected network, each neuron requires 297 600 weights. A 1920x1080 full HD image would require 2,073,600 weights. If the images are polychrome, the amount of weights is multiplied by the amount of colour channels (typically three). Thus, we can see that the overall number of free parameters in the network quickly becomes extremely large as the image size increases. Too large models cause overfitting and slow performance.

Furthermore, many pattern detection tasks require that the solution is translation invariant. It is inefficient to train neurons to separately recognize the same pattern in the left-top corner and in the right-bottom corner of an image. A fully-connected neural network fails to take this kind of structure into account.

2.5.2 Basic structure

The basic idea of the CNN was inspired by a concept in biology called the receptive field. Receptive fields are a feature of the animal visual cortex. They act as detectors that are sensitive to certain types of stimulus, for example, edges. They are found across the visual field and overlap each other.

This biological function can be approximated in computers using the convolution operation. In image processing, images can be filtered using convolution to produce different visible effects. Figure 2.3 shows how a hand-selected convolutional filter detects horizontal edges from an image, functioning similarly to a receptive field.

The discrete convolution operation between an image f and a filter matrix g is defined as:

$$h[x,y] = f[x,y] * g[x,y] = \sum_n \sum_m f[n,m]g[x-n,y-m]$$

.

In effect, the dot product of the filter g and a sub-image of f (with same dimensions as g) centred on coordinates x,y produces the pixel value of h at coordinates x,y. The size of the receptive field is adjusted by the size of the filter matrix. Aligning the filter successively with every sub-image of f produces the of output pixel matrix h. In the case of neural networks, the output matrix is also called an feature map (or an activation map

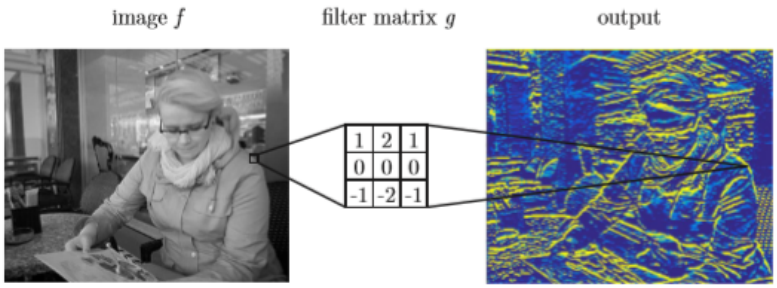

Figure 2.3: Detecting horizontal edges from an image using convolution filtering.
link

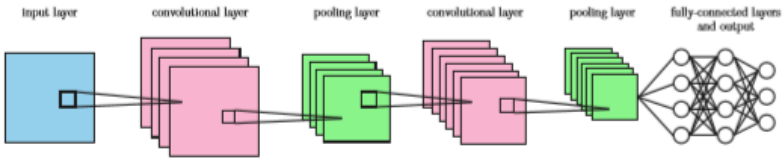

Figure 2.4: An example of a convolutional network.
link

after computing the activation function). Edges need to be treated as a special case. If image f is not padded, the output size decreases slightly with every convolution.

A set of convolutional filters can be combined to form a convolutional layer of a neural network. The matrix values of the filters are treated as neuron parameters and trained using machine learning. The convolution operation replaces the multiplication operation of a regular neural network layer. Output of the layer is usually described as a volume. The height and width of the volume depend on the dimensions of the activation map. The depth of the volume depends on the number of filters.

Since the same filters are used for all parts of the image, the number of free parameters is reduced drastically compared to a fully-connected neural layer. The neurons of the convolutional layer mostly share the same parameters and are only connected to a local region of the input. Parameter sharing resulting from convolution ensures translation invariance. An alternative way of describing the convolutional layer is to imagine a fully-connected layer with an infinitely strong prior placed on its weights. This prior forces the neurons to share weights at different spatial locations and to have zero weight outside the receptive field.

Successive convolutional layers (often combined with other types of layers, such as pooling described below) form a convolutional neural network (CNN). An example of a convolutional network is shown in figure 2.4. The backpropagation training algorithm, described in section 2.2.3, is also applicable to convolutional networks. In theory, the layers closer to the input should learn to recognize low-level features of the image, such

as edges and corners, and the layers closer to the output should learn to combine these features to recognize more meaningful shapes. In this thesis, we are interested in studying whether convolutional networks can learn to recognize complete objects.

2.5.3 Pooling and stride

To make the network more manageable for classification, it is useful to decrease the activation map size in the deep end of the network. Generally,the deep layers of the network require less information about exact spatial locations of features, but require more filter matrixes to recognize multiple high-level patterns. By reducing the height and width of the data volume, we can increase the depth of the data volume and keep the computation time at a reasonable level.

There are two ways of reducing the data volume size. One way is to include a pooling layer after a convolutional layer. The layer effectively down-samples the activation maps. Pooling has the added effect of making the resulting network more translation invariant by forcing the detectors to be less precise. However, pooling can destroy information about spatial relationships between subparts of patterns. Typical pooling method is max-pooling. Max-pooling simply outputs the maximum value within a rectangular neighbourhood of the activation map.

Another way of reducing the data volume size is adjusting the stride parameter of the convolution operation. The stride parameter controls whether the convolution output is calculated for a neighbourhood centred on every pixel of the input image (stride 1) or for every nth pixel (stride n). Research has shown that pooling layers can often be discarded without loss in accuracy by using convolutional layers with larger stride value. The stride operation is equivalent to using a fixed grid for pooling.

2.5.4 Additional layers

The convolutional layer typically includes a non-linear activation function, such as a rectified linear activation function (see subsection 2.2.5). Activations are sometimes described as a separate layer between the convolutional layer and the pooling layer.

Some systems, such as, also implement a layer called local response normalization, which is used as a regularization technique. Local response normalization mimics a function of biological neurons called lateral inhibition, which causes excited neurons to decrease the activity of neighbouring neurons. However, other regularization techniques are currently more popular and these are discussed in the next section.

The final hidden layers of a CNN are typically fully-connected layers. A fully-connected layer can capture some interesting relationships parameter-sharing convolutional layers cannot. However, a fully connected layer requires a sufficiently small data volume size in order to be practical. Pooling and stride settings can be used to reduce the size of the

data volume that reaches the fully-connected layers. A convolutional network that does not include any fully-connected layers, is called a fully convolutional network (FCN).

If the network is used for classification, it usually includes a softmax output layer (see also section 2.2.5). The activations of the topmost layers can also be used directly to generate a feature representation of an image. This means that the convolutional network is used as a large feature detector.

2.5.5 Regularization and data augmentation

Regularization refers to methods that are used to reduce overfitting by introducing additional constraints or information to the machine learning system. A classical way of using regularization in neural networks is adding a penalty term to the objective/loss function that penalizes certain types of weights. The parameter sharing feature of convolutional networks is another example of regularization.

There are several regularization techniques that are specific to deep neural networks. A popular technique called dropout attempts to reduce the co-adaptation of neurons. This is achieved by randomly dropping out neurons during training, meaning that a slightly different neural network is used for each training sample or minibatch. This causes the system not to depend too much on any single neuron or connection and provides an effective yet computationally inexpensive way of implementing regularization. In convolutional networks, dropout is typically used in the final fully-connected layers.

Overfitting can also be reduced by increasing the amount of training data. When it is not possible to acquire more actual samples, data augmentation is used to generate more samples from the existing data. For classification using convolutional networks, this can be achieved by computing transformations of the input images that do not alter the perceived object classes, yet provide additional challenge to the system. The images can be, for example, flipped, rotated or subsampled with different crops and scales. Also, noise can be added to the input images.

2.5.6 Development

Convolutional neural networks were one of the first successful deep neural networks. The Neocognitron, developed by Fukushima in 1980s, provided a neural network model for translation-invariant object recognition, inspired by biology. Le Cun et al. combined this method with a learning algorithm, i.e. back-propagation. These early solutions were mostly used for handwritten character recognition.

After providing some promising results, the neural network methods faded in prominence and were mostly replaced by support vector machines. Then, in 2012, Krizhevsky et al. achieved excellent results on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset by combining Le Cun's method with recent fine-tuning methods

for deep learning. These results popularised CNNs and led to the development of new powerful object detection methods described in chapter 3.

For the 2014 ImageNet challenge, Simonyan and Zisserman explored the effect of increasing the depth of a convolutional network on localisation and classification accuracy. The team achieved results that improved the then state-of-the-art by using convolutional networks 16 and 19 layers deep. The 16-layer architecture includes 13 convolutional layers (with 3x3 filters), 5 pooling layers (2x2 neighbourhood max-pooling) and 3 fully-connected layers. All hidden layers use rectified (ReLu) activations. The fully-connected layers reduce 4096 channels down to 1000 softmax outputs and are regularized using dropout. This form of network is referred to as VGG-16 later in this thesis.

The current (2016) winner of the object detection category in the ImageNet challenge is also CNN-based. The method uses a combination of CRAFT region proposal generation, gated bi-directional CNN, clustering, landmark generation and ensembling.

Chapter 3

Convolutional object detection

Outline: This chapter presents the following:

1. Introduction
2. R-CNN
3. Fast R-CNN
4. Region proposal generation and use
5. Faster R-CNN
6. SSD
7. Comparing the methods

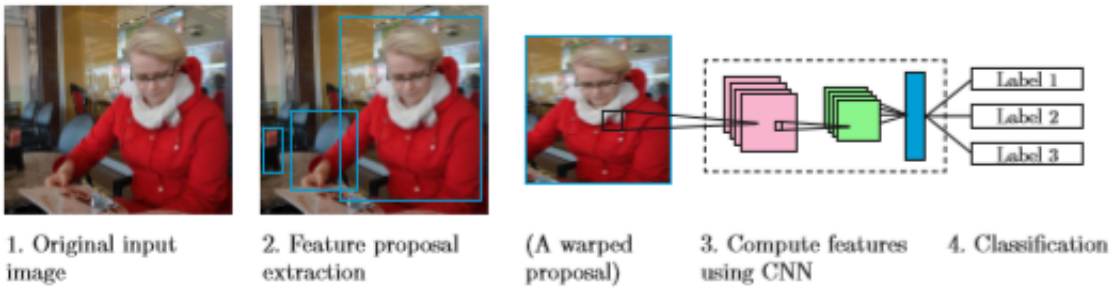

Figure 3.1: Stages of R-CNN forward computation.
link

3.1 Introduction

In this chapter, we discuss and compare different object detection methods that utilize convolutional neural networks. In particular, we are going to look at methods that combine CNNs with region proposal classification. We further discuss, how the region proposals, also called regions of interest (ROI), are generated.

3.2 R-CNN

In 2012, Krizhevsky et al. achieved promising results with CNNs for the general image classification task, as mentioned in section 2.4.6. In 2013, Girshick et al. published a method generalizing these results to object detection. This method is called R-CNN ("CNN with region proposals").

3.2.1 General description

R-CNN forward computation has several stages, shown in figure 3.1. First, the regions of interest are generated. The RoIs are category-independent bounding boxes that have a high likelihood of containing an interesting object. In the paper, a separate method called Selective Search, is used for generating these, but other region generation methods can be used instead. Selective Search, along with other region proposal generation techniques, is discussed in further detail in section 3.3.

Next, a convolutional network is used to extract features from each region proposal. The sub-image contained in the bounding-box is warped to match the input size of the CNN and then fed to the network. After the network has extracted features from the input, the features are input to support vector machines (SVM) that provide the final classification. The method is trained in multiple stages, beginning with the convolutional network.

After the CNN has been trained, the SVMs are fitted to the CNN features. Finally, the region proposal generating method is trained.

3.2.2 Drawbacks

R-CNN is an important method, because it provided the first practical solution for object detection using CNNs. Being the first, it has many drawbacks that have been improved upon by later methods.

In his 2015 paper for Fast R-CNN, Girshick lists three main problems of R-CNN:

First, training consists of multiple stages, as described above. Second, training is expensive. For both SVM and region proposal training, features are extracted from each region proposal and stored on disk. This requires days of computation and hundreds of gigabytes of storage space. Third, and perhaps most important, object detection is slow, requiring almost a minute for each image, even on a GPU. This is because the CNN forward computation is performed separately for every object proposal, even if the proposals originate from the same image or overlap each other.

3.3 Fast R-CNN

Fast R-CNN published in 2015 by Girshick provides a more practical method for object recognition. The main idea is to perform the forward pass of the CNN for the entire image, instead of performing it separately for each ROI.

3.3.1 General description

The general structure of Fast R-CNN is illustrated in figure 3.2. The method receives as input an image plus regions of interest computed from the image. As in R-CNN, the RoIs are generated using an external method. The image is processed using a CNN that includes several convolutional and max pooling layers. The convolutional feature map that is generated after these layers is input to a RoI pooling layer.

This extracts a fixed-length feature vector for each RoI from the feature map. The feature vectors are then input to fullyconnected layers that are connected to two output layers: a softmax layer that produces probability estimates for the object classes and a real-valued layer that outputs bounding box co-ordinates computed using regression (meaning refinements to the initial candidate boxes).

3.3.2 Classification performance

According to the authors, Fast R-CNN provides significantly shorter classification time compared to regular R-CNN, taking less than a second on a state-of-the-art GPU. This is

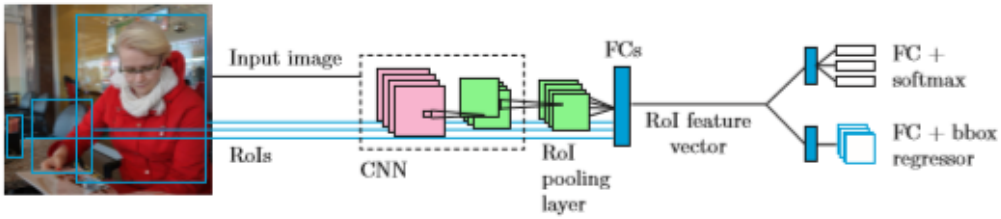

Figure 3.2: Stages of Fast R-CNN forward computation.
link

mainly due to using the same feature map for each RoI.

As the detection time decreases, the overall computation time begins to depend significantly on the performance of the region proposal generation method. The RoI generation can thus form a computational bottleneck. Additionally, when there are many RoIs, the time spent on evaluating the fully-connected layers can dominate the evaluation time of the convolutional layers. Classification time can be accelerated by approximately 30% if the fully-connected layers are compressed using truncated singular value decomposition. This results in a slight decrease in precision, however.

3.3.3 Training

According to the original publication, Fast R-CNN is more efficient to train than R-CNN, with nine-fold reduction in training time. The entire network (including the RoI pooling layer and the fully-connected layers) can be trained using the back-propagation algorithm and stochastic gradient descent. Typically, a pre-trained network is used as a starting point and then fine-tuned. Training is done in mini-batches of N images. R/N RoIs are sampled from each mini-batch image. The RoI samples are assigned to a class, if their intersection over union with a ground-truth box is over 0.5. Other RoIs belong to the background class.

As in classification, RoIs from the same image share computation and memory usage. For data augmentation, the original image is flipped horizontally with probability 0.5. The softmax classifier and the bounding box regressors are fine-tuned together using a multi-task loss function, which considers both the true class of the sampled RoI and the offset of the sampled bounding box from the true bounding box.

3.4 Region proposal generation and use

To use R-CNN and Fast R-CNN, we need a method for generating the classagnostic regions of interest. Next, we are going to discuss general principles of RoI generation, and have a closer look at two popular methods: Selective Search and Edge Boxes.

3.4.1 Overview

The aim of region proposal generation in object detection is to maximize recall i.e. to generate enough regions so that all true objects are recovered. The generator is less concerned with precision, since it is the task of the object detector to identify correct regions from the output of the region proposal generator.

However, the amount of proposals generated affects performance. As mentioned in section 2.3.2, there are two main approaches to region generation: dense set generation and sparse set generation.

Dense set solutions attempt to generate by brute force an exhaustive set of bounding boxes that includes every potential object location. This can be achieved by sliding a detection window across the image. However, searching through every location of the image is computationally costly and requires a fast object detector. Additionally, different window shapes and sizes need to be considered. Thus, most sliding window methods limit the amount of candidate objects by using a coarse step-size and a limited number of fixed aspect ratios.

Most region proposals in a dense set do not contain interesting objects. These proposals need to be discarded after the object detection phase. Detection results can be discarded, if they fall behind a predefined confidence threshold or if their confidence value is below a local maximum (non-maximum suppression).

Instead of discarding the regions after the object detection stage, the region proposal generator itself can rank the regions in a class-agnostic way and discard low-ranking regions. This generates a sparse set of object detections. Similarly to dense set methods, thresholding and non-maximum suppression can be implemented after the detection phase to further improve the detection quality. Sparse set solutions can be grouped into unsupervised and supervised methods.

One of the most popular unsupervised methods is Selective Search (see section 3.3.2), which utilizes an iterative merging of superpixels. There are also other methods that use the same approach. Another approach is to rank the objectness of a sliding window. A popular example of this is Edge Boxes (see section 3.3.3), which calculates the objectness score by calculating the number of edges within a bounding box and by subtracting the number of edges that overlap the box boundary. There is also a third group of methods based on seed segmentation.

Supervised methods treat region proposal generation as a classification or a regression problem. This means using a machine learning algorithm, such as a support vector machine. It is also possible to use a convolutional network to generate the regions of interest. An example of using a CNN for calculating the bounding boxes is Multi-Box.

Certain advanced object detection methods, such as Faster R-CNN described in 3.4.1, use parts of the same convolutional network both for generating the region proposals and for detection. We call these kinds of methods integrated methods.

3.4.2 Selective Search

Selective Search utilizes a hierarchical partitioning of an image to create a sparse set of object locations. The main design philosophy is not to use a single strategy, but to combine the best features of bottom-up segmentation and exhaustive search. The authors had three main design considerations: the search should capture all scales, be diverse i.e. not use any single strategy for grouping regions and be fast to compute.

The algorithm begins by creating a set of small initial regions using a method called Graph Based Image Segmentation designed by Felzenszwalb and Huttenlocher. The method creates a set of regions called superpixels. The superpixels are internally nearly uniform. Combined, they span the entire image, but individually they should not span different objects.

Selective Search then continues by iteratively grouping the regions together using a greedy algorithm, beginning with the two most similar regions. Many complimentary measures are used to compute the similarity. These measures consider colour similarity (by computing a colour histogram), texture similarity (by computing a SIFT-like measure), size of the regions (small regions should be merged earlier) and how well the regions fit together (gaps should be avoided). The grouping phase ends when every region has been combined.

The hypothetical object locations thus generated are then ordered by the likelihood of the location containing an object. In practice, the locations are ordered based on the order in which they were grouped together by the different measures. A certain element of randomness is added to prevent large objects from being favoured too much. Lower-ranking duplicates are removed.

Both the region generating method and the similarity measures were selected to be fast to compute, making the method fast in general. In addition to using diverse similarity measures, the search can be further diversified by using complementary colour spaces (to ensure lighting invariance) and using complementary starting regions.

3.4.3 Edge Boxes

As the name suggests, Edge Boxes is based on detecting objects from edge maps. The main contribution of the authors of the method is the observation that the number of edge contours wholly enclosed by a bounding box is correlated with the likelihood that the box contains an object.

First, the edge map is calculated using a method by the same authors called Structured Edge Detector. Then, thick edge lines are thinned using non-maximum suppression. Instead of operating on the edge pixels directly, the pixels are grouped using a greedy algorithm. An affinity measure is devised to calculate whether edge groups are part of the same contour.

The region proposals are found by scanning the image using the traditional sliding

window method and calculating an objectness score at each position, aspect ratio and scale. The score is calculated by summing the edge strength of edge groups that lie completely within the box and subtracting the strength of edge groups that are part of a contour that cross the box boundary. Promising regions are then further refined.

3.5 Faster R-CNN

In the experimental section of this thesis, we will focus mostly on Faster R-CNN. There are, however, several state-of-the-art algorithms with an improved computation time or accuracy. Next, we will describe two of these algorithms first faster R-CNN in this section and SSD (see section 3.6) in next section.

3.5.1 Overview

Faster R-CNN by Ren et al. is an integrated method. The main idea is to use shared convolutional layers for region proposal generation and for detection. The authors discovered that feature maps generated by object detection networks can also be used to generate the region proposals. The fully convolutional part of the Faster R-CNN network that generates the feature proposals is called a region proposal network (RPN). The authors used Fast R-CNN architecture for the detection network.

A Faster R-CNN network is trained by alternating between training for RoI generation and detection. First, two separate networks are trained. Then, these networks are combined and fine-tuned. During fine-tuning, certain layers are kept fixed and certain layers are trained in turn.

The trained network receives a single image as input. The shared fully convolutional layers generate feature maps from the image. These feature maps are fed to the RPN. The RPN outputs region proposals, which are input, together with the said feature maps, to the final detection layers. These layers include a RoI pooling layer and output the final classifications.

Using shared convolutional layers, region proposals are computationally almost cost-free. Computing the region proposals on a CNN has the added benefit of being realizable on a GPU. Traditional RoI generation methods, such as Selective Search, are implemented using a CPU.

For dealing with different shapes and sizes of the detection window, the method uses special anchor boxes instead of using a pyramid of scaled images or a pyramid of different filter sizes. The anchor boxes function as reference points to different region proposals centred on the same pixel.

3.5.2 General description

In this section, we briefy introduce the key aspects of the Faster R-CNN. We refer readers to the original paper ((Ren, He, Girshick, & Sun, 2015)) for more technical details.

In the RPN, the convolution layers of a pre-trained network are followed by a 3×3 convolutional layer. This corresponds to mapping a large spatial window or receptive field (e.g., 228×228 for VGG16) in the input image to a low-dimensional feature vector at a center stride (e.g., 16 for VGG16). Two 1×1 convolutional layers are then added for classification and regression branches of all spatial windows.

To deal with different scales and aspect ratios of objects, anchors are introduced in the RPN. An anchor is at each sliding location of the convolutional maps and thus at the center of each spatial window. Each anchor is associated with a scale and an aspect ratio. Following the default setting of, we use 3 scales (1282, 2562, and 5122 pixels) and 3 aspect ratios (1: 1, 1: 2, and 2: 1), leading to k = 9 anchors at each location. Each proposal is parameterized relative to an anchor. Therefore, for a convolutional feature map of size W $\times$H, we have at most W H k possible proposals. We note that the same features of each sliding location are used to regress k=9 proposals, instead of extracting k sets of features and training a single regressor.t Training of the RPN can be done in an end-to-end manner using stochastic gradient descent (SGD) for both classification and regression branches. For the entire system, we have to take care of both the RPN and Fast R-CNN modules since they share convolutional layers. In this paper, we adopt the approximate joint learning strategy proposed in ((Ren et al., 2015)). The RPN and Fast R-CNN are trained end-to-end as they are independent. Note that the input of the Fast R-CNN is actually dependent on the output of the RPN. For the exact joint training, the SGD solver should also consider the derivatives of the RoI pooling layer in the Fast R-CNN with respect to the coordinates of the proposals predicted by the RPN. However, as pointed out by ((Ren et al., 2015)), it is not a trivial optimization problem.

3.6 SSD

In the experimental section of this thesis, we will focus mostly on Faster R-CNN. There are, however, several state-of-the-art algorithms with an improved computation time or accuracy. Next, we will describe SSD in this section and faster R-CNN (see section 3.5) described in previous section.

3.6.1 Overview

The Single Shot MultiBox Detector(SSD) takes integrated detection even further. The method does not generate proposals at all, nor does it involve any resampling of image segments. It generates object detections using a single pass of a convolutional network.

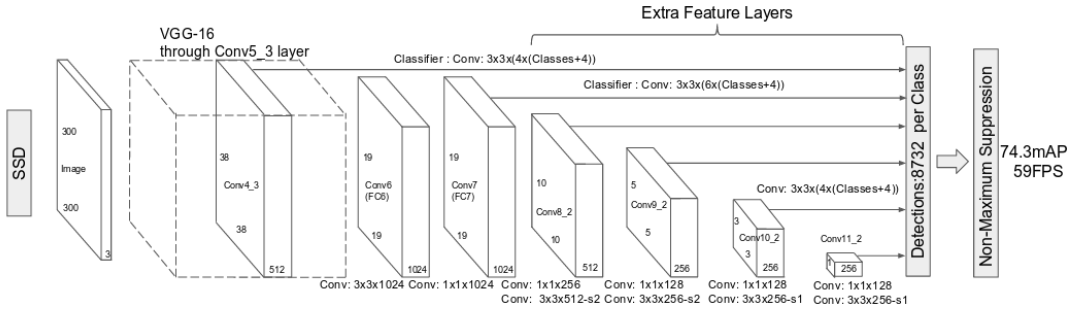

Figure 3.3: SSD Architecture
link

Somewhat resembling a sliding window method, the algorithm begins with a default set of bounding boxes. These include different aspect ratios and scales. The object predictions calculated for these boxes include offset parameters, which predict how much the correct bounding box surrounding the object differs from the default box.

The algorithm deals with different scales by using feature maps from many different convolutional layers (i.e. larger and smaller feature maps) as input to the classifier. Since the method generates a dense set of bounding boxes, the classifier is followed by a non-maximum suppression stage that eliminates most boxes below a certain confidence threshold.

3.7 Comparing the methods

Above, we described how Fast R-CNN is faster and more accurate than regular R-CNN. But how does Fast R-CNN perform compared to the above mentioned advanced methods?

Liu et al. [36] compared the performance of Fast R-CNN, Faster R-CNN and SSD on the PASCAL VOC 2007 test set (see section 4.5 for discussion of the standard benchmarks). When using networks trained on the PASCAL VOC 2007 training data, Fast R-CNN achieved a mean average precision (mAP) of 66.9. Faster R-CNN performed slightly better, with a mAP of 69.9. SSD achieved a mAP of 68.0 with input size 300 x 300 and 71.6 with input size 512 x 512. As the standard implementations of Fast R-CNN and Faster R-CNN use 600 as the length of the shorter dimension of the input image, SSD seems to perform better with similarly sized images. However, SSD requires extensive use of data augmentation to achieve this result. Fast R-CNN and Faster RCNN only use horizontal flipping, and it is currently unknown, whether they would benefit from additional augmentation.

While the advanced methods are more precise than Fast R-CNN, the real improvements

come from speed. When most of the detections with a low probability are eliminated using thresholding and non-maximum suppression, SSD512 can run at 19 FPS on a Titan X GPU. Meanwhile, Faster R-CNN with a VGG-16 architecture performs at 7 FPS. The original authors of Faster R-CNN report a running time of 5 FPS i.e. 0.2 s per image. Fast R-CNN has approximately the same evaluation speed, but requires additional time for calculating the region proposals. Region generation time depends on the method, with Selective Search requiring 2 seconds per image on a CPU and Edge Boxes requiring 0.2 seconds per image.

Chapter 4

Implementation and Setup

Outline: This chapter presents the following:

1. Introduction
2. Data Acquisition
3. Region Proposal Networks
4. Edge Boxes Parameters
5. Faster R-CNN Architecture Model

4.1 Introduction

In this chapter, we begin discussing the experimental part of the thesis. First, we will discuss selection criteria for models and datasets. Then we will describe the selected models, their parameters and the selected datasets. Finally, we will discuss postprocessing and evaluation. The implementation of the models is mostly discussed in the following chapter. However, some implementation details are also discussed in this chapter, since they influence method selection.

4.2 Data Acquisition

In this section, we will describe first, rice crop insect images dataset classification species and their labels and next, rice crop weed images dataset.

4.2.1 Insects Data Exploration

The dataset contains 814 images in which 723 training images and 91 test images, each of which belongs to one of fifteen species at several different growth stages(i.e Egg, Larva, Pupa and Adult.). We obtained the dataset through reference of Rice Knowledge Bank, but originally the dataset has been taken from different sites and google search by the reference of this mention link and we have taken raw images of rice crop insects for their Complete Metamorphosis(Holometalolous) type according to their 4 stages and they are Egg, Larva, Pupa and Adult. The training images include the label of insect species, while the test images also include labels. We have used only the 723 training images to train a model and test the model accuracy.

The input images are all square, but they have a wide range of pixel widths and heights, with some less than 100x100 pixels and some over 1000x1000 pixels, and each image has RGB colors. The figure 4.1 shows the distribution of pixel widths (also equal to the height distribution) for all input images.

I will need to rescale the input images to a uniform size to input into my neural network. A smaller uniform size makes the most sense, considering that many images are 100 pixels on a side, and it's easier to reduce the sizes of images than to reliably scale up the smaller images to a larger size. Also, using a smaller uniform size will help to save computational power. Therefore, I chose to rescale all images to the minimum size of the input data, 800 $\times$ 600.

The input images include rice crop insects species from the following: Armyworms, Brown Plant Leafhopper , Gall Midge, Grasshopper, Leaf Folder, Mealy Bug, Paddy Stemborer, Rice Case Worm, Rice Earhead Bug, Rice Horned Caterpillar, Rice Skipper, Spiny Beetle, Thrips, White Backed Plant Hopper, Whorl Maggot. The table shown in the figure 4.2 gives the total number of each species, which range from 32 - 85

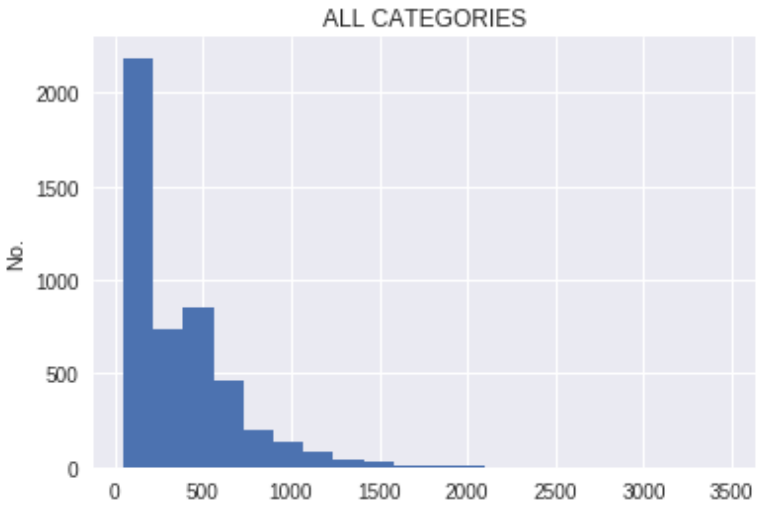

Figure 4.1: Histogram of insects dataset

I split the 814 images into two different datasets: training set (88%) and a test set (12%). The training set is used to train the model and the test set is used to adjust weight parameters to check the loss function after training the model. Finally, the test set and other image serves to check the model accuracy using our scoring metrics and accuracy score. The images below are examples of each insect species of the input images.

4.2.2 Weed Data Exploration

The dataset contains 224 images in which 194 training images and 30 test images, each of which belongs to one of different growth stages of rice crop weed. We obtained the dataset through following dataset link weed dataset, but originally the dataset has been taken from the Figshare website. The training images include the label of weed species, while the test images also include labels. We have used only the 194 training images to train a model and test the model accuracy.

I will need to rescale the input images to a uniform size to input into my neural network. A smaller uniform size makes the most sense, considering that many images are 100 pixels on a side, and it's easier to reduce the sizes of images than to reliably scale up the smaller images to a larger size. Also, using a smaller uniform size will help to save computational power. Therefore, I chose to rescale all images to the minimum size of the input data, 800 $\times$ 600.

Label	Species Class	Size	Totel
1	Armyworms	800 × 600	65
2	Brown Plant Leafhopper	800 × 600	73
3	Gall Midge	800 × 600	32
4	Grasshopper	800 × 600	74
5	Leaf Folder	800 × 600	45
6	Mealy Bug	800 × 600	68
7	Paddy Stemborer	800 × 600	43
8	Rice Case Worm	800 × 600	44
9	Rice Earhead Bug	800 × 600	48
10	Rice Horned Caterpillar	800 × 600	63
11	Rice Skipper	800 × 600	42
12	Spiny Beetle	800 × 600	85
13	Thrips	800 × 600	61
14	White Backed Plant Hopper	800 × 600	37
15	Whorl Maggot	800 × 600	36

Figure 4.2: Rice crop insects species list

4.3 Region Proposal Networks

A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score. We model this process with a fully convolutional network, which we describe in this section. Because our ultimate goal is to share computation with a Fast R-CNN object detection network, we assume that both nets share a common set of conv layers. In our experiments, we investigate the Zeiler and Fergus model(ZF), which has 5 shareable conv layers and the Simonyan and Zisserman model (VGG), which has 13 shareable conv layers.

To generate region proposals, we slide a small network over the conv feature map output by the last shared conv layer. This network is fully connected to an n × n spa-

Figure 4.3: Insects species name-I

Figure 4.4: Insects species name-II

Figure 4.5: Insects species name-III

Figure 4.6: Insects species name-IV

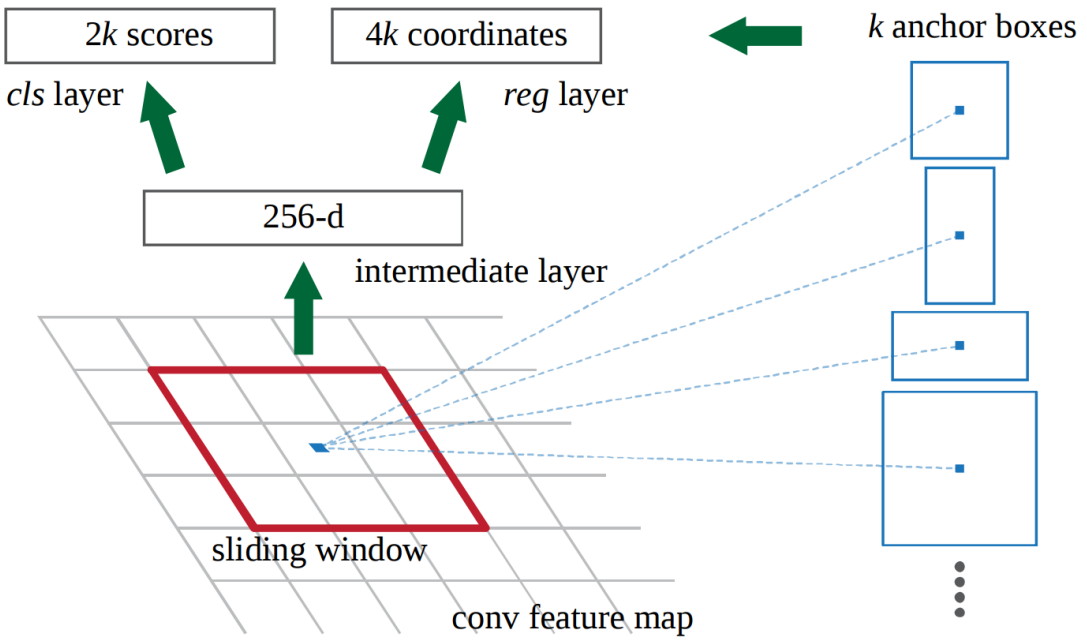

Figure 4.7: Region Proposal Network (RPN)
link

tial window of the input conv feature map. Each sliding window is mapped to a lower-dimensional vector (256-d for ZF and 512-d for VGG). This vector is fed into two sibling fully-connected layers—a box-regression layer (reg) and a box-classification layer (cls). We use n = 3 in this paper, noting that the effective receptive field on the input image is large (171 and 228 pixels for ZF and VGG, respectively). This mininetwork is illustrated at a single position in Figure 4.7. Note that because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. This architecture is naturally implemented with an n $\times$ n conv layer followed by two sibling 1 $\times$ 1 conv layers (for reg and cls, respectively). ReLUs are applied to the output of the n $\times$ n conv layer.

4.3.1 Translation-Invariant Anchors

At each sliding-window location, we simultaneously predict k region proposals, so the reg layer has 4k outputs encoding the coordinates of k boxes. The cls layer outputs 2k scores that estimate probability of object / not-object for each proposal.2 The k proposals are parameterized relative to k reference boxes, called anchors. Each anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio. We use

3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position. For a conv feature map of a size W $\times$ H (typically 2,400), there are W Hk anchors in total. An important property of our approach is that it is translation invariant, both in terms of the anchors and the functions that compute proposals relative to the anchors.

As a comparison, the MultiBox method uses k-means to generate 800 anchors, which are not translation invariant. If one translates an object in an image, the proposal should translate and the same function should be able to predict the proposal in either location. Moreover, because the MultiBox anchors are not translation invariant, it requires a (4+1) $\times$800-dimensional output layer, whereas our method requires a (4+2) $\times$9-dimensional output layer. Our proposal layers have an order of magnitude fewer parameters (27 million for MultiBox using GoogLeNet vs. 2.4 million for RPN using VGG-16), and thus have less risk of overfitting on small datasets, like PASCAL VOC.

4.3.2 A Loss Function for Learning Region Proposals

For training RPNs, we assign a binary class label (of being an object or not) to each anchor. We assign a positive label to two kinds of anchors: (i) the anchor/anchors with the highest Intersectionover-Union (IoU) overlap with a ground-truth box, or (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. Note that a single ground-truth box may assign positive labels to multiple anchors. We assign a negative label to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the training objective.

With these definitions, we minimize an objective function following the multi-task loss in Fast RCNN. Our loss function for an image is defined as:

$$L(p_i,t_i) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*)$$

(Note : For simplicity we implement the cls layer as a two-class softmax layer. Alternatively, one may use logistic regression to produce k scores.)

Here, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes (object vs. not object). For the regression loss, we use $L_{reg}(t_i,t_i^*) = R(t_i - t_i^*)$ where R is the robust loss function (smooth L1) defined in paper. The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$). The outputs of the cls and reg layers consist of p_i and t_i respectively. The two terms are normalized with N_{cls} and N_{reg} , and a balancing weight λ.

For regression, we adopt the parameterizations of the 4 coordinates following :

$$t_x = (x - x_a)/w_a, t_y = (y - y_a)/h_a, t_w = log(w/w_a), t_h = log(h/h_a)$$

$$t_x^* = (x^* - x_a)/w_a, t_y^* = (y^* - y_a)/h_a, t_w^* = log(w^*/w_a), t_h^* = log(h^*/h_a)$$

where x, y, w, and h denote the two coordinates of the box center, width, and height. Variables x, x_a, and $x^*$are for the predicted box, anchor box, and ground-truth box respectively (likewise for y, w, h). This can be thought of as bounding-box regression from an anchor box to a nearby ground-truth box.

Nevertheless, our method achieves bounding-box regression by a different manner from previous feature-map-based methods. In bounding-box regression is performed on features pooled from arbitrarily sized regions, and the regression weights are shared by all region sizes. In our formulation, the features used for regression are of the same spatial size (n $\times$ n) on the feature maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor is responsible for one scale and one aspect ratio, and the k regressors do not share weights. As such, it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale.

4.3.3 Optimization

The RPN, which is naturally implemented as a fully-convolutional network, can be trained end-to-end by back-propagation and stochastic gradient descent (SGD). We follow the "imagecentric" sampling strategy from to train this network. Each mini-batch arises from a single image that contains many positive and negative anchors. It is possible to optimize for the loss functions of all anchors, but this will bias towards negative samples as they are dominate. Instead, we randomly sample 256 anchors in an image to compute the loss function of a mini-batch, where the sampled positive and negative anchors have a ratio of up to 1:1. If there are fewer than 128 positive samples in an image, we pad the mini-batch with negative ones.

We randomly initialize all new layers by drawing weights from a zero-mean Gaussian distribution with standard deviation 0.01. All other layers (i.e., the shared conv layers) are initialized by pretraining a model for ImageNet classification, as is standard practice. We tune all layers of the ZF net, and conv3 1 and up for the VGG net to conserve memory. We use a learning rate of 0.001 for 60k mini-batches, and 0.0001 for the next 20k mini-batches on the PASCAL dataset. We also use a momentum of 0.9 and a weight decay of 0.0005. Our implementation uses Caffe.

4.3.4 Sharing Convolutional Features for Region Proposal and Object Detection

Thus far we have described how to train a network for region proposal generation, without considering the region-based object detection CNN that will utilize these proposals. For the detection network, we adopt Fast R-CNN and now describe an algorithm that learns conv layers that are shared between the RPN and Fast R-CNN.

Both RPN and Fast R-CNN, trained independently, will modify their conv layers in different ways. We therefore need to develop a technique that allows for sharing conv layers between the two networks, rather than learning two separate networks. Note that this is not as easy as simply defining a single network that includes both RPN and Fast R-CNN, and then optimizing it jointly with backpropagation. The reason is that Fast R-CNN training depends on fixed object proposals and it is not clear a priori if learning Fast R-CNN while simultaneously changing the proposal mechanism will converge. While this joint optimizing is an interesting question for future work, we develop a pragmatic 4-step training algorithm to learn shared features via alternating optimization.

In the first step, we train the RPN as described above. This network is initialized with an ImageNet-pre-trained model and fine-tuned end-to-end for the region proposal task. In the second step, we train a separate detection network by Fast R-CNN using the proposals generated by the step-1 RPN. This detection network is also initialized by the ImageNet-pre-trained model. At this point the two networks do not share conv layers. In the third step, we use the detector network to initialize RPN training, but we fix the shared conv layers and only fine-tune the layers unique to RPN. Now the two networks share conv layers. Finally, keeping the shared conv layers fixed, we fine-tune the fc layers of the Fast R-CNN. As such, both networks share the same conv layers and form a unified network.

4.4 Edge Boxes Parameters

As explained in section 3.3.3, Edge Boxes finds the initial region candidates using a sliding window search. The step size of the search is controlled by parameter α, which defines the intersection over union (IoU) of neighbouring boxes. The same parameter defines the step size for translation, scale and aspect ratio. After the boxes have been found and refined, they are sorted and non-maximum suppression (NMS) is performed. During NMS, boxes are discarded if their IoU with a higher-ranking box is more than β.

α and β are the most important parameters of Edge Boxes. The authors of the method studied the effect of these parameters from the perspective of finding a suitable candidate accuracy for the object detector. Even though an IoU value higher than 0.5 is typically used as the criterion for evaluating the performance of an object detector (as the limit for determining true matches), the object detector often performs better, if the candidate objects are a closer match to true objects. Thus, the target IoU should be set higher than the minimum acceptable IoU.

The authors define the target IoU as δ and performed experiments with δ values of 0.5, 0.7 and 0.9. Since Fast R-CNN in any case calculates refinements of the bounding boxes, we decided that increasing δ above 0.9 is not needed for the present experiments. The resulting increase in the number of candidates would only slow down the algorithm. We chose a more practical value $\delta = 0.7$ as the target IoU.

α and β are set according to δ. If a higher value of δ is required, α and β are selected

to output denser sets of bounding boxes around apparent objects. The authors found that a value of $\alpha = 0.65$ provided the best results for $\delta < 0.9$. The optimal value of β was determined to be $\delta + 0.05$. Thus, we chose $\alpha = 0.65$ and $\beta = 0.75$ as the parameter values.

For the minimum objectness score limit h_b^{in} (used for discarding uninteresting windows before the refinement stage), we used the default value of 0.01. We also used the default value of 10,000 as the maximum number of object proposals.

4.5 Faster R-CNN Architecture Model

One of the state-of-the-art object detection models is Faster R-CNN. The architecture of Faster R-CNN is shown in Figure 4.8. Given an image, we first employ a pre-trained deep convolutional neural network, such as VGG, to extract feature maps from it. Then, they use a Region Proposal Network (RPN), which consists of two convolutional layers, to detect the regions that might contain object in the feature maps (image). Then the network employ a RoI pooling layer to crop and resize the the feature maps according to these region proposals. The new feature maps of each region are then used for classification and finer bounding box regression through three fully connected layers.

One issue of this Faster R-CNN architecture is that the region proposals output from RPN are not very accurate. This is due to RPN using some hard coded anchors with fixed scales and aspects to guess the potential regions. Although RPN also has a bounding box regression part, it only aims to give general boxes that may contain any kind of objects. Without class specific knowledge, these boxes won't be very accurate. We argue that the error of classification and final bounding box regression might be partly caused by error of proposed region. Suppose we can have finer region proposals, the accuracy of classification and final bounding box regression may be further improved.

4.5.1 Iterative Region Proposal Refinement Model

A nature better region proposal will be the regressed bounding box since it is designed to refine the rough region output by RPN using class specific knowledge encoded in the network. Since regressed bounding box is one of the outputs of the whole Faster R-CNN network, we cannot use it as our region proposal at the beginning. However, once we get the finer bounding box, we can start another round of classification and bounding box regression, using the regressed bounding box in previous round as the region proposals in the new round. This process can keep going for several iterations. In this iterative way, we can refine the region proposals again and again and might get a better result after each iteration.

The model of Faster R-CNN with iterative region proposal refinement is shown in Figure 4.9. In the first iteration, it's exactly the same as Faster R-CNN: extract feature

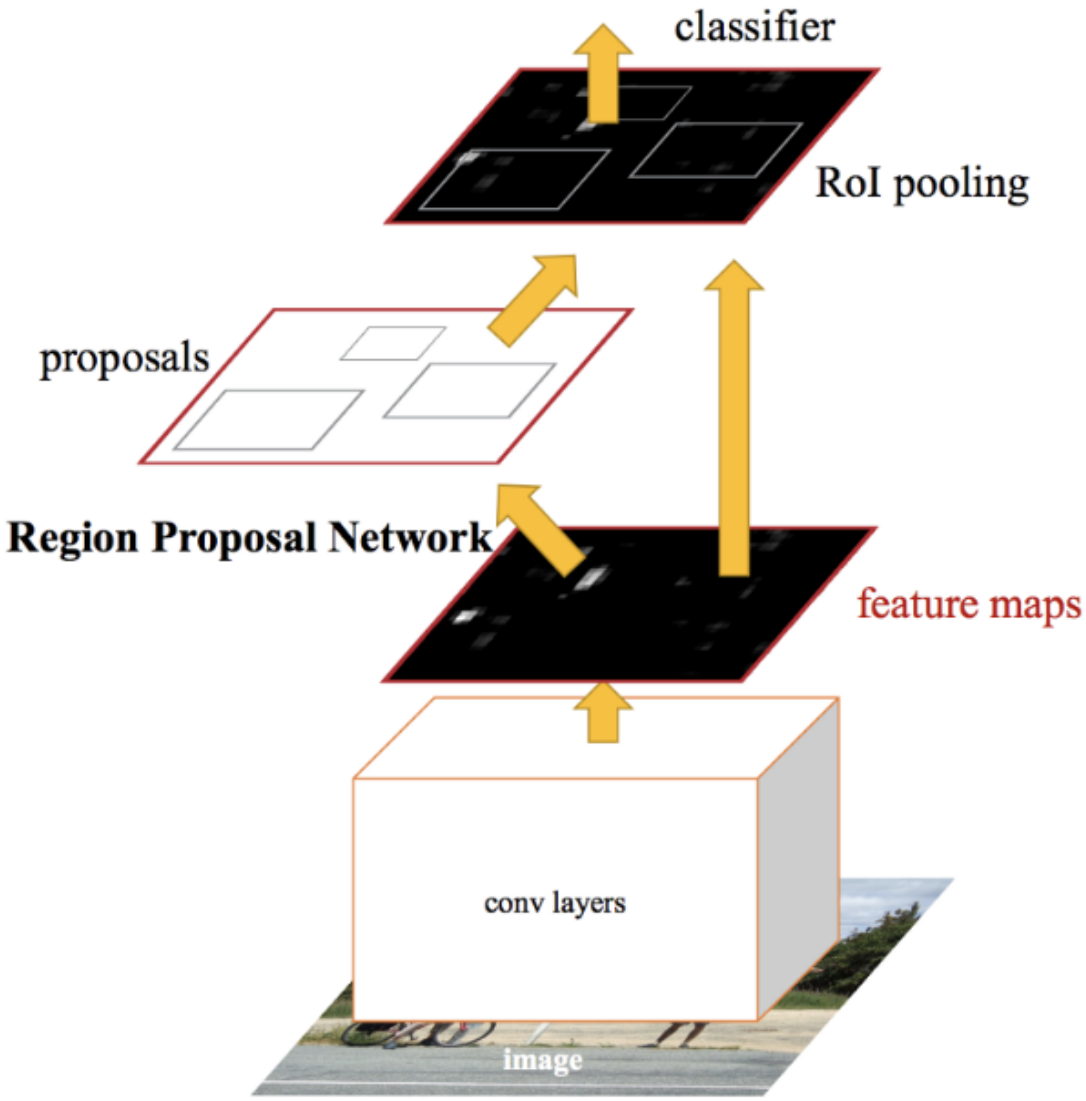

Figure 4.8: Architecture of Faster R-CNN
link

maps from image by VGG, pass them to RPN to get region proposals, employ RoI pooling layer to crop and resize new feature maps for each proposal, and use a three-layer fully connected network to get the final class scores and bounding box regression for each class.

After the first iteration, for each proposed region, we select the regressed bounding box with the maximum class score as the region proposal in the second iteration. And the rest of second iteration is the same as the first iteration: we use RoI pooling layer to crop and resize feature maps for each proposal, classify and regress the bounding box using new feature maps. Note that for the input of RoI pooling layer, we reuse the feature maps in first iteration and there is no need for recalculation. Also, we reuse parameters of the three-layer fully connected network in different iterations. After the second iteration, we can repeat the same process for the third iteration, and so on. Figure 4.9 is a demonstration of iteration number = 3.

Our iterative refinement model can be represented by the following equations. For the feature maps extraction and each initial RPN region proposal, we have

$$f = VGG(image)$$

$$RP_1 = RPN(f)$$

Then suppose we set iteration number = T, then for each iteration i, the model can be represented as :

for i =1, 2, ...T,

$$r_i = RoIPooling(f, RP_i)$$

$$scores_i, boxes_i = FC^3(r_i)$$

$$RP_{i+1} = boxes_{i, argmax_{j \in 0...C} scores_{ij}}$$

$$loss_i = loss_{cross-entropy}(scores_i, C_i) + \lambda [c \neq bg] loss_{smooth_{L1}}(boxes_i, b_i)$$

where FC^3 denotes the three-layer fully connected network, C denotes the number of classes, c_i denotes the class label for the proposed region RP_i , b_i denotes the ground truth bounding box of object in this region,$[c_i \neq b_g]$ = 1 if the region is not a background and contains object, otherwise $[c_i \neq b_g]$ = 0, and λ is the weight parameter to balance classification loss and bounding box regression loss.

During training, for the classification, we use softmaxcross entropy loss, and for the bounding box regression, we use smooth L1 norm loss. We will get a $loss_i$ in each iteration. The final loss will be the sum of all iterations (loss from RPN might also be added to the final loss). And in test time, we use the scores and bounding boxes from the last iteration as the final output:

$$loss_{final} = \sum_{i=1}^{T} loss_i$$

$$output = scores_T, boxes_T$$

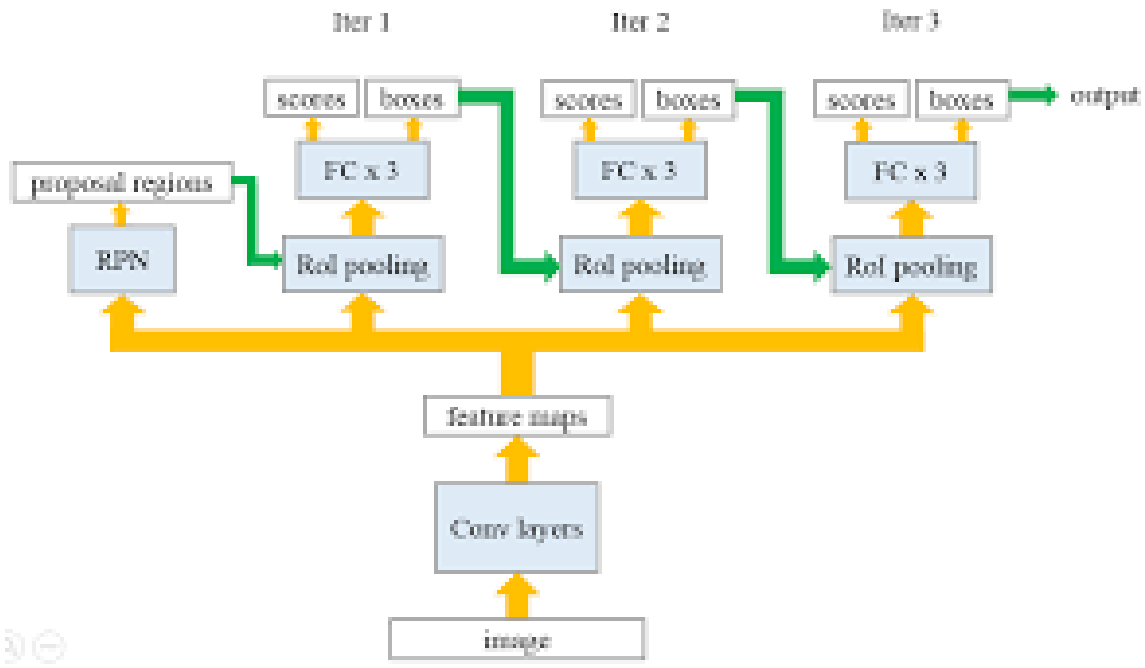

Figure 4.9: Architecture of Faster R-CNN with iterative region proposal refinement link

4.5.2 LSTM Region Proposal Refinement Model

One issue of the iterative refinement model is the gradient of loss can not be backpropagated from the later iteration to the earlier iterations. Since the error of classification and bounding box regression of later iteration might be caused by errors of earlier iterations, we hope those earlier errors can also be corrected by gradient backpropagation. However, this cannot be done by iterative refinement model since RoI layer cannot backpropagate the gradient to the region proposal coordinates. This is because RoI layer uses these coordinates to crop the feature maps, but the "crop" operation is not differentiable with respect to coordinates. As a result, in Figure 4.9 the gradient of each loss in iterative model can only be propagated downward (yellow arrows) but cannot be propagated to the left (green arrows).

Inspired by Reccurrent Neural Network (RNN), which can pass useful information to later time steps and also propagate the gradient backward to the previous time steps, we propose a LSTM region proposal refinement model, where LSTM is one of RNN models. The architecture of our model is shown in Figure 4.10. The different between iterative and LSTM refinement models is that we add a LSTM layer right before the final fully connected (output) layer, and pass the hidden states of LSTM to the next iteration (blue arrows). As a result, if the iteration number = T, then in each iteration i, the equations become

for i =1, 2, ...T

$$r_i = RoIPooling(f, RoI_i)$$

$$a_i = FC^2(r_i)$$

$$h_i = LSTM(h_{i-1}, a_i)$$

$$scores_i, boxes_i = FC(h_i)$$

$$RoI_{i+1} = boxes_{i,argmax_{j\in 0...C} scores_{ij}}$$

$$loss_i = loss_{cross-entropy}(scores_i, C_i) + \lambda[c \neq bg] loss_{smooth_{L1}}(boxes_i, b_i)$$

where h_i is the hidden state of LSTM at iteration i, and the input of the LSTM layer is h_i - 1, the hidden state of previous iteration, and a_i, the output of two-layer fully connected network in current iteration. The rest of this model is the same as iterative refinement model.

One benefit of adding a LSTM layer is that the hidden state of previous iteration can contain information that is useful to improve classification and bounding box regression results in current iteration. Another benefit is that we can now backpropagate the gradient of loss from the later iterations to the earlier ones (through the blue arrows in Figure 4.10), since LSTM is differentiable with respect to the previous hidden state. As a result, if the error of prediction in current iteration comes from the errors of previous iterations, LSTM gives it a chance to correct those errors by gradient backpropagation.

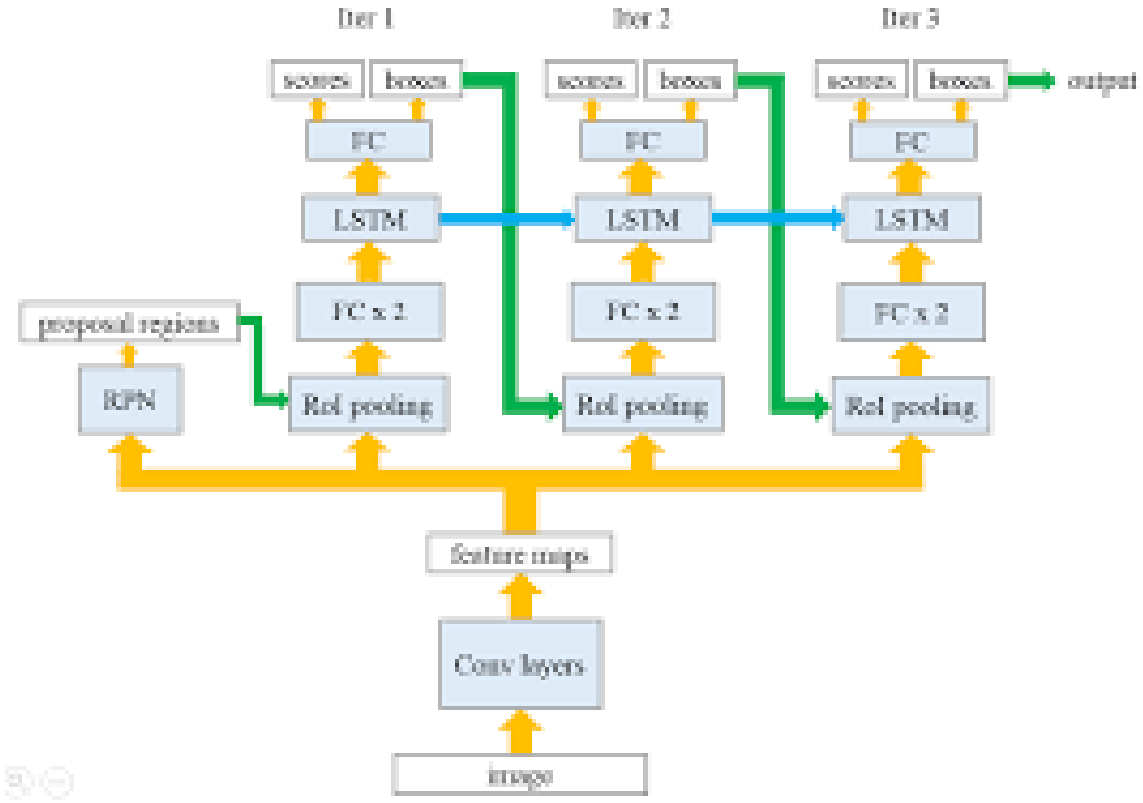

Figure 4.10: Architecture of Faster R-CNN with LSTM region proposal refinement. link

Actually, this is an unusual use of LSTM/RNN model. RNNs are more often used in sequential input data such as text, audio and video. The hidden state of RNN was proved to have the ability to capture the temporal information of data at previous time steps. Since the meaning of current frame in sequential data is often related to the frames previous to it, RNN makes a great success in encoding features of sequential data.

Although our data is image, which is not sequential data, but the way we refine the region proposals and make progress step by step also contains temporal information. We can see the iterations as multiple guesses. In each iteration, we look into the guessed region of the image and get some information that is useful to decide a better guess next time. This way of sequential processing of non-sequential data is often called attention or glimpse model, which has been used in both multiple object detection and question answering . In the paper, they also use a LSTM to refine the answer span for a question for multiple times.

Chapter 5

Result Analysis and Conclusion

Outline: This chapter presents the following:

1. Result Analysis
2. Conclusion
3. The Future
4. Hardware Requirement

5.1 Result Analysis

Regarding precision, the results were promising. We showed how a system trained on general image data can be used to detect rice crop insects and weed in a specific task (object detection), thus demonstrating the adaptability of the methods. In some cases, Faster R-CNN detected more insects or weed than the annotators of the original data had marked. These were labelled as false positives, despite clearly having the right species class by visual inspection.

On the other hand, the system did also make some actual normal mistakes. We deduced that this was most likely due to a lack of negative training examples. Further, we demonstrated the importance of postprocessing the results. Non-maximum suppression has a significant impact on the average precision. In general, overlapping objects and bounding boxes cause problems for object detection systems. We also showed that choice of the region generation method affects both speed and accuracy of the object detection system.

The results from the Google search images are shown in following figure 5.1 table.

5.2 Conclusion

An accurate and efficient object detection system has been developed which achieves comparable metrics with the existing state-of-the-art system. This project uses recent techniques in the field of computer vision and deep learning. Custom dataset was created using labelImg and the evaluation was consistent. This can be used in real-time applications which require object detection for pre-processing in their pipeline.

An important scope would be to train the system on a video sequence for usage in coltivation applications. Addition of a temporally consistent network would enable smooth detection and more optimal than per-frame detection.

5.2.1 Overview

We began the thesis with a review of the theoretical background. We explained how neural networks function and what object detection entails. We demonstrated why regular neural networks are insufficient to image-related tasks and how translation invariant convolutional networks provide an effective solution to many computer vision problems.

Next, we demonstrated how convolutional object detection has evolved from the relatively slow R-CNN to the current optimized methods. This development is mostly not related to the structure of the convolutional network itself. Rather, it is related to how the convolutional network is used and to computation that takes place before and after the convolutional network. In the previous methods, there were many more separate phases involving preprocessing, region generation, computation of the fully connected layers and

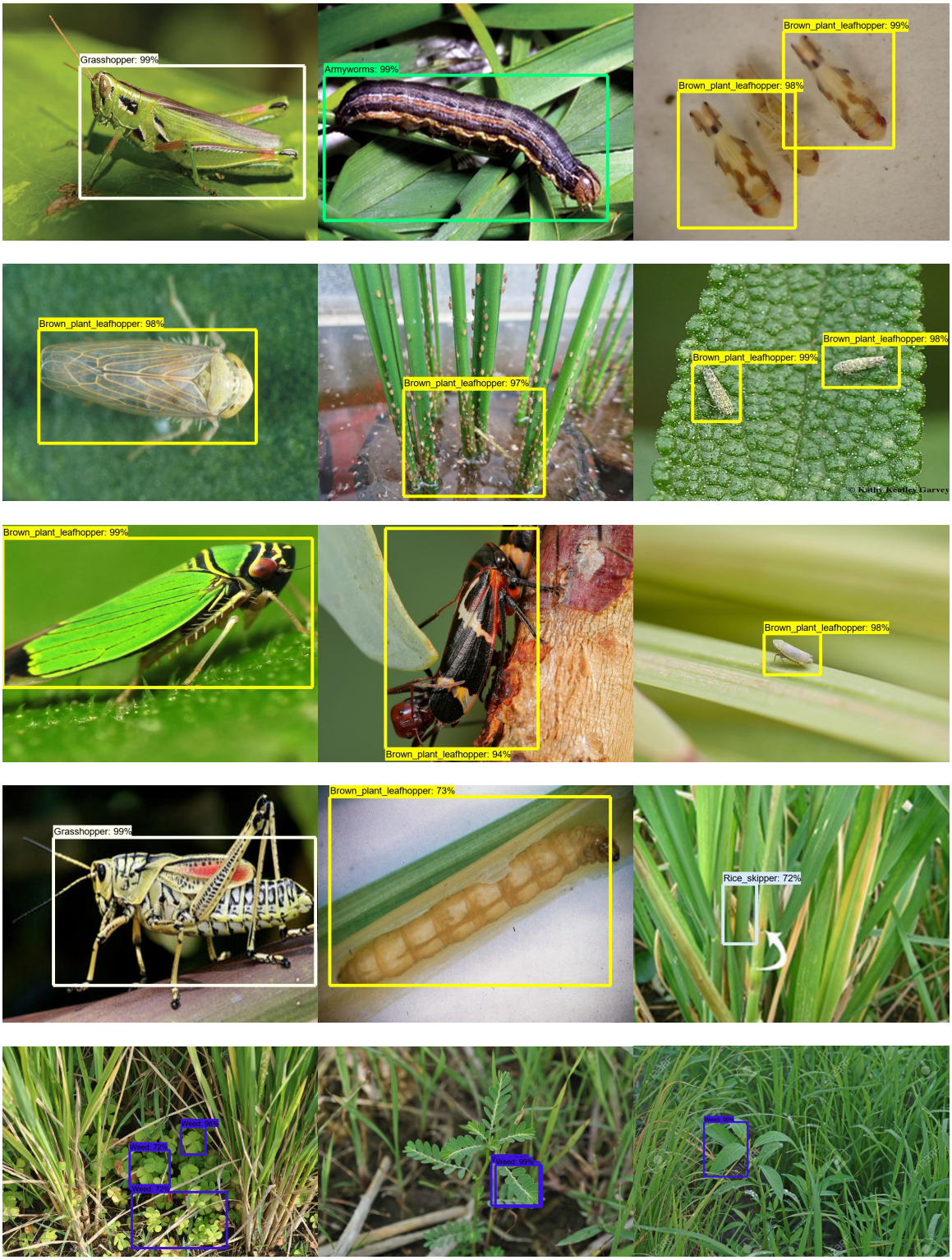

Figure 5.1: Classification and Detection Result-I

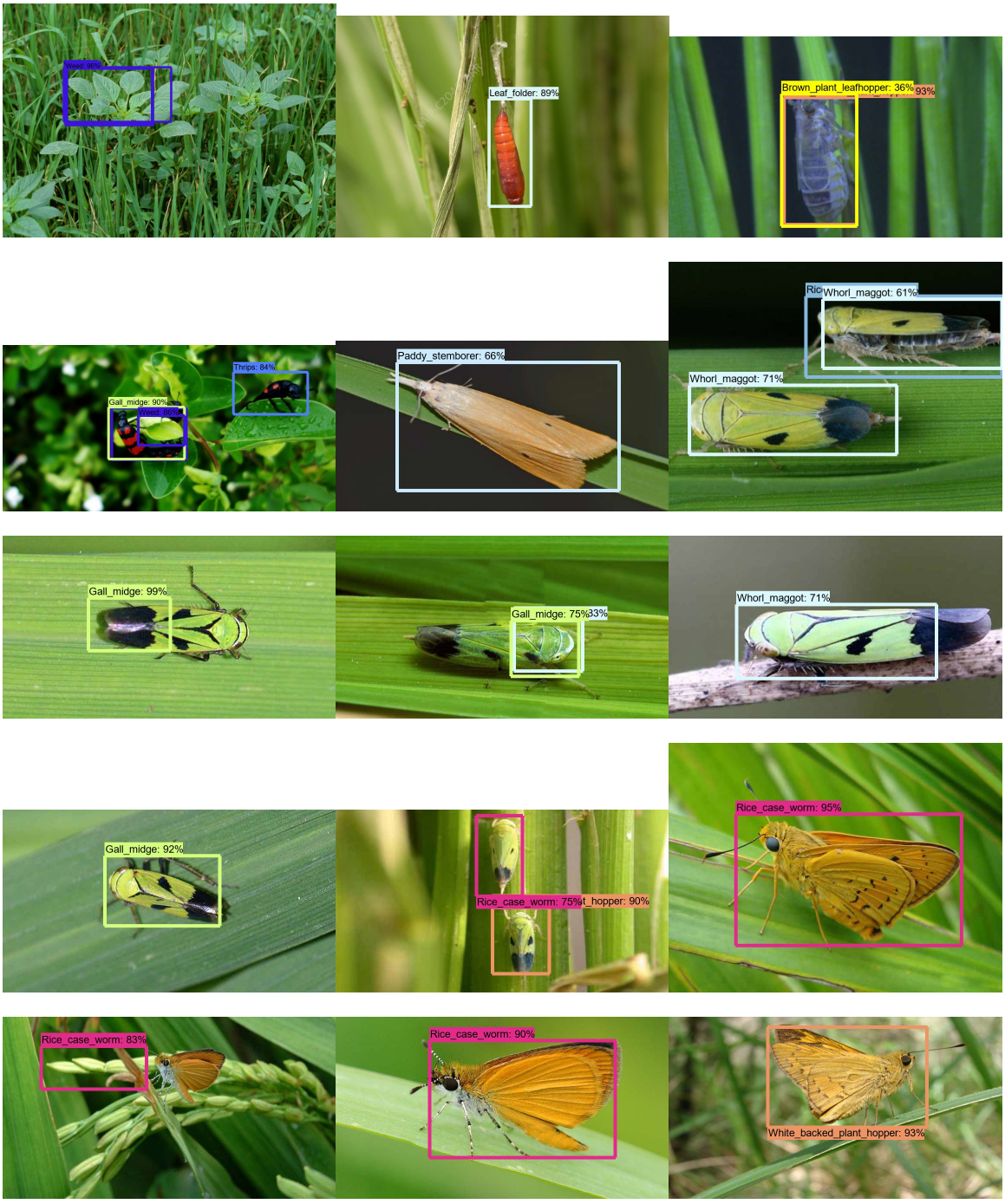

Figure 5.2: Classification and Detection Result-II

the final classification. In the latest methods, these phases have been increasingly integrated into the convolutional network itself, while keeping the basic CNN model intact. On the other hand, the 2016 winner of the ImageNet challenge is again a model composed of many separate components. Nonetheless, several computational bottlenecks have disappeared. Over the past few years, the speed of object detection has improved more dramatically than precision.

5.2.2 Practice

To experiment with a convolutional method in practice, we created a working PYTHON implementation of Faster R-CNN. We learned that the most challenging part of implementing a deep learning system is collecting the training data and performing the training itself. The publicly available datasets serve as a useful starting point for both research and practical implementations. Training time can be further shortened by using a pretrained network. Even if the final system does not feature the same objects classes as the benchmark data, visual problems are universal enough to benefit from detectors trained for a different problem. The optimal bottom-layers of a convolutional network are often similar regardless of the problem, just like human eye uses the same receptive fields for all visual tasks. Thus, it makes sense to initialize the layers using a pretrained network.

Related to the implementation, we also learned that there are no easy "out-of-the-box" solutions for effectively implementing convolutional networks. Current software tools, such as Caffe and MatConvNet, require specialist skills. If it is possible to use such tools, creating a working implementation is not too difficult. However, the tools are quite finicky regarding interoperability of software versions and hardware, especially if implementation is attempted on a GPU.

5.3 The Future

Exact study of the speed of the method variants was left outside the scope of the thesis. More detailed execution-time evaluation on consumer-level computers would be an interesting topic for further research. Many methods that claim "real time performance" can achieve this only on hardware costing thousands of euros. Even though hardware cost is not a major issue for certain applications such as self-driving cars and computer servers, which use expensive hardware in any case, more applications become practical after CNNs can be evaluated on home computers and mobile devices. Yet, our implementation showed that the methods have improved enough to detect objects in a few seconds on a consumer laptop, even though we did not measure this exactly.

One of the strengths of convolutional networks is their inherent translation invariance. Yet, taking the context of the whole image into consideration could potentially create an even more precise system. We experimented with a geometry-based inference system,

which alters the probabilities of object detections based on their geometric plausibility. However, the method did not improve the accuracy of Faster R-CNN. We provided an analysis of how the geometric detection method functioned and determined that, while the method eliminated some false detections, it also created many new ones. The geometric method was also impractically slow. Yet, lessons learned from the study can be used to explore further ways of implementing context-sensitive object detection.

As explained in the previous chapter, a trend can be perceived in the literature of making detection systems wholly neural or convolutional. Just like a deep neural network learns to automatically learn the inherent features of an object class, an even deeper and more smartly used neural network could learn the probabilities of finding an object from a certain part of the scene. This would make a separate geometric inference method irrelevant. However, time will show if this is the route future research takes. As a Danish proverb says, it is difficult to make predictions, especially about the future.

5.4 Hardware Requirement

We have implemented and tested this thesis project with the help of following hardware equipments :

1. Windows 10 Operating System
2. Intel i5-7200U 7th(Gen) CPU @ 2.50GHz- 2.71GHz
3. CPU 8GB RAM
4. NVIDIA Graphic 940MX with 4GB RAM
5. NVIDIA Graphic Driver 430.86
6. CUDA Toolkit 9.0
7. cuDNN SDK v7.6.0
8. Tensor-flow 1.13.1

References

Barrero, O., Rojas, D., Gonzalez, C., & Perdomo, S. (2016, Aug). Weed detection in rice fields using aerial images and neural networks. In *2016 xxi symposium on signal processing, images and artificial vision (stsiva)* (p. 1-4).

Chu, H., Zhang, D., Shao, Y., Chang, Z., Guo, Y., & Zhang, N. (2018, Nov). Using hog descriptors and uav for crop pest monitoring. In *2018 chinese automation congress (cac)* (p. 1516-1519).

de Oca, A. M., Arreola, L., Flores, A., Sanchez, J., & Flores, G. (2018, June). Low-cost multispectral imaging system for crop monitoring. In *2018 international conference on unmanned aircraft systems (icuas)* (p. 443-451).

Figshare [Computer software manual]. (accessed June 15, 2019). Retrieved from `https://figshare.com/articles/rice_seedlings_and_weeds/7488830`

Hameed, S., & Amin, I. (2018, Nov). Detection of weed and wheat using image processing. In *2018 ieee 5th international conference on engineering technologies and applied sciences (icetas)* (p. 1-5). doi: 10.1109/ICETAS.2018.8629137

Liu, Y. (2018, Dec). An improved faster r-cnn for object detection. In *2018 11th international symposium on computational intelligence and design (iscid)* (Vol. 02, p. 119-123). doi: 10.1109/ISCID.2018.10128

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 91–99). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf`

Ren, S., He, K., Girshick, R., & Sun, J. (2017, June). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137-1149. doi: 10.1109/TPAMI.2016.2577031

Rice knowledge bank [Computer software manual]. (accessed June 10, 2019). Retrieved from `http://www.knowledgebank.irri.org/step-by-step-production/growth/pests-and-diseases/insects`

Saha, A. K., Saha, J., Ray, R., Sircar, S., Dutta, S., Chattopadhyay, S. P., & Saha, H. N. (2018, Jan). Iot-based drone for improvement of crop quality in agricultural field.

In *2018 ieee 8th annual computing and communication workshop and conference (ccwc)* (p. 612-615). doi: 10.1109/CCWC.2018.8301662

Shah, J. P., Prajapati, H. B., & Dabhi, V. K. (2016, March). A survey on detection and classification of rice plant diseases. In *2016 ieee international conference on current trends in advanced computing (icctac)* (p. 1-8). doi: 10.1109/ICCTAC.2016.7567333

University, S. (accessed June 30, 2019). Convolutional neural networks [Computer software manual]. Retrieved from `http://cs231n.github.io/convolutional-networks/`

(Ren, He, Girshick, & Sun, 2017) (Shah, Prajapati, & Dabhi, 2016) (Barrero, Rojas, Gonzalez, & Perdomo, 2016) (Saha et al., 2018) (de Oca, Arreola, Flores, Sanchez, & Flores, 2018) (Chu et al., 2018) (Hameed & Amin, 2018) (Liu, 2018) (Ren et al., 2015) ("Rice Knowledge Bank", accessed June 10, 2019) ("Figshare", accessed June 15, 2019) (University, accessed June 30, 2019)

Appendix A

Image Detection and Classification Source Code

```
1
2 # Import packages
3 import os
4 import cv2
5 from matplotlib import pyplot as plt
6 import numpy as np
7 import tensorflow as tf
8 import sys
9
10 # This is needed since the notebook is stored in the object_detection
    folder.
11 sys.path.append("..")
12
13 # Import utilites
14 from utils import label_map_util
15 from utils import visualization_utils as vis_util
16
17
18 # Name of the directory containing the object detection module we're
    using
19 MODEL_NAME = 'inference_graph'
20 IMAGE_DIR = 'test_images'
21 IMAGE_NAME = '250.jpg'
22
23 # Grab path to current working directory
24 CWD_PATH = os.getcwd()
25
26 # Path to frozen detection graph .pb file, which contains the model
    that is used
27 # for object detection.
28 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph
```

```
    .pb')

PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')

PATH_TO_IMAGE = os.path.join(CWD_PATH,IMAGE_DIR,IMAGE_NAME)



# Number of classes the object detector can identify
NUM_CLASSES = 15



# network predicts 5, we know that this corresponds to "Leaf_folder"
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
    max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)


detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')

  sess = tf.Session(graph=detection_graph)


image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0
    ')

detection_scores = detection_graph.get_tensor_by_name('detection_scores
    :0')
detection_classes = detection_graph.get_tensor_by_name('
    detection_classes:0')

num_detections = detection_graph.get_tensor_by_name('num_detections:0')

image = cv2.imread(PATH_TO_IMAGE)
image_expanded = np.expand_dims(image, axis=0)


real_image = image.copy()
plt.imshow(real_image)
```

```
73
74
75 (boxes, scores, classes, num) = sess.run(
76 [detection_boxes, detection_scores, detection_classes, num_detections],
77 feed_dict={image_tensor: image_expanded})
78
79 vis_util.visualize_boxes_and_labels_on_image_array(
80 image,
81 np.squeeze(boxes),
82 np.squeeze(classes).astype(np.int32),
83 np.squeeze(scores),
84 category_index,
85 use_normalized_coordinates=True,
86 line_thickness=8,
87 min_score_thresh=0.50)
88
89 cv2.namedWindow('Object detector', cv2.WINDOW_NORMAL)
90 cv2.resizeWindow('Object detector', 800,400)
91 output = np.hstack((real_image,image))
92 cv2.imwrite('result/'+ IMAGE_NAME, output)
93 cv2.imshow('Object detector', output)
94
95 cv2.waitKey(0)
96 cv2.destroyAllWindows()
```

Listing A.1: Image Detection and Classification

Appendix B

Video Detection and Classification Source Code

```
1
2  # Import packages
3  import os
4  import cv2
5  import numpy as np
6  import tensorflow as tf
7  import sys
8  sys.path.append("..")
9
10 # Import utilites
11 from utils import label_map_util
12 from utils import visualization_utils as vis_util
13
14
15 # Name of the directory containing the object detection module we're
       using
16 MODEL_NAME = 'inference_graph'
17 VIDEO_NAME = 'test_images/weed.mp4'
18
19 # Grab path to current working directory
20 CWD_PATH = os.getcwd()
21
22 # Path to frozen detection graph .pb file, which contains the model
       that is used
23 # for object detection.
24
25 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph
       .pb')
26
27
28 PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')
```

```
29
30
31 PATH_TO_VIDEO = os.path.join(CWD_PATH, VIDEO_NAME)
32
33 NUM_CLASSES = 16
34
35
36 # network predicts '16', we know that this corresponds to 'Weed'.
37 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
38 categories = label_map_util.convert_label_map_to_categories(label_map,
       max_num_classes=NUM_CLASSES, use_display_name=True)
39 category_index = label_map_util.create_category_index(categories)
40
41
42 detection_graph = tf.Graph()
43 with detection_graph.as_default():
44   od_graph_def = tf.GraphDef()
45   with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
46     serialized_graph = fid.read()
47     od_graph_def.ParseFromString(serialized_graph)
48     tf.import_graph_def(od_graph_def, name='')
49
50   sess = tf.Session(graph=detection_graph)
51
52
53
54 image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
55
56 detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0
       ')
57
58 detection_scores = detection_graph.get_tensor_by_name('detection_scores
       :0')
59 detection_classes = detection_graph.get_tensor_by_name('
       detection_classes:0')
60
61 num_detections = detection_graph.get_tensor_by_name('num_detections:0')
62
63
64
65 video = cv2.VideoCapture(PATH_TO_VIDEO)
66
67 while(video.isOpened()):
68
69   ret, frame = video.read()
70   frame_expanded = np.expand_dims(frame, axis=0)
71
72
73   (boxes, scores, classes, num) = sess.run(
```

```
74        [detection_boxes, detection_scores, detection_classes,
    num_detections],
75        feed_dict={image_tensor: frame_expanded})
76
77
78    vis_util.visualize_boxes_and_labels_on_image_array(
79      frame,
80      np.squeeze(boxes),
81      np.squeeze(classes).astype(np.int32),
82      np.squeeze(scores),
83      category_index,
84      use_normalized_coordinates=True,
85      line_thickness=8,
86      min_score_thresh=0.60)
87
88
89    cv2.imshow('Object detector', frame)
90
91    if cv2.waitKey(1) == ord('q'):
92      break
93
94
95  video.release()
96  cv2.destroyAllWindows()
```

Listing B.1: Video Detection and Classification

Appendix C

Webcam Detection and Classification Source Code

```
1
2  # Import packages
3  import os
4  import cv2
5  import numpy as np
6  import tensorflow as tf
7  import sys
8
9  # This is needed since the notebook is stored in the object_detection
       folder.
10 sys.path.append("..")
11
12 # Import utilites
13 from utils import label_map_util
14 from utils import visualization_utils as vis_util
15
16
17 # Name of the directory containing the object detection module we're
       using
18 MODEL_NAME = 'inference_graph'
19
20 # Grab path to current working directory
21 CWD_PATH = os.getcwd()
22
23 # Path to frozen detection graph .pb file, which contains the model
       that is used
24 # for object detection.
25 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph
       .pb')
26
27 # Path to label map file
```

```
28 PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')
29
30 # Number of classes the object detector can identify
31 NUM_CLASSES = 16
32
33
34
35 ## Load the label map.
36 # Label maps map indices to category names, so that when our
       convolution
37 # network predicts `5`, we know that this corresponds to `king`.
38 # Here we use internal utility functions, but anything that returns a
39 # dictionary mapping integers to appropriate string labels would be
       fine
40 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
41 categories = label_map_util.convert_label_map_to_categories(label_map,
       max_num_classes=NUM_CLASSES, use_display_name=True)
42 category_index = label_map_util.create_category_index(categories)
43
44
45 # Load the Tensorflow model into memory.
46 detection_graph = tf.Graph()
47 with detection_graph.as_default():
48   od_graph_def = tf.GraphDef()
49   with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
50     serialized_graph = fid.read()
51     od_graph_def.ParseFromString(serialized_graph)
52     tf.import_graph_def(od_graph_def, name='')
53
54   sess = tf.Session(graph=detection_graph)
55
56
57 # Define input and output tensors (i.e. data) for the object detection
       classifier
58
59 # Input tensor is the image
60 image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
61
62 # Output tensors are the detection boxes, scores, and classes
63 # Each box represents a part of the image where a particular object was
       detected
64 detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0
       ')
65
66 # Each score represents level of confidence for each of the objects.
67 # The score is shown on the result image, together with the class label
       .
68 detection_scores = detection_graph.get_tensor_by_name('detection_scores
       :0')
```

```
69 detection_classes = detection_graph.get_tensor_by_name('
       detection_classes:0')
70
71 # Number of objects detected
72 num_detections = detection_graph.get_tensor_by_name('num_detections:0')
73
74
75 # Initialize webcam feed
76 video = cv2.VideoCapture(0)
77 ret = video.set(3,1280)
78 ret = video.set(4,720)
79
80
81 while(True):
82
83   # Acquire frame and expand frame dimensions to have shape: [1, None,
       None, 3]
84   # i.e. a single-column array, where each item in the column has the
       pixel RGB value
85   ret, frame = video.read()
86   frame_expanded = np.expand_dims(frame, axis=0)
87
88   # Perform the actual detection by running the model with the image as
       input
89   (boxes, scores, classes, num) = sess.run(
90     [detection_boxes, detection_scores, detection_classes,
       num_detections],
91       feed_dict={image_tensor: frame_expanded})
92
93   # Draw the results of the detection (aka 'visulaize the results')
94   vis_util.visualize_boxes_and_labels_on_image_array(
95     frame,
96     np.squeeze(boxes),
97     np.squeeze(classes).astype(np.int32),
98     np.squeeze(scores),
99     category_index,
100    use_normalized_coordinates=True,
101    line_thickness=8,
102    min_score_thresh=0.60)
103
104  # All the results have been drawn on the frame, so it's time to
       display it.
105  cv2.imshow('Object detector', frame)
106
107  # Press 'q' to quit
108  if cv2.waitKey(1) == ord('q'):
109    break
110
111 # Clean up
```

```
112 video.release()
113 cv2.destroyAllWindows()
```

Listing C.1: Webcam Detection and Classification