Norwegian University
of Life Sciences

Master's Thesis 2017
30 ECTS

Faculty of Science and Technology

Deep Neural Networks for Object Detection in Agricultural Robotics

Eirik Solberg
Mechanical Engineering and Product Development

Preface

Choosing machine learning as the subject of my thesis has probably had a lot to do with recent hype about workers being displaced by artificial intelligence and incredible research results bordering on magic. Machine learning has very little to do with the machines I am familiar with from the mechanical engineering coursework I completed in the years leading up to this thesis. Working on this thesis has however provided the excitement of rapidly ascending a steep learning curve as well as the frustration encountered when nothing works. Most importantly it has served to satisfy my curiosity in a big way and I think I have learned things that will serve me well in the future.

I would like to thank my thesis advisors, Professor Pål From and Lars Grimstad for their help and feedback throughout this semester. In preparation for this thesis I was given the opportunity to visit the University of Minnesota together with the robotics and control group, and would like to thank Professor Volkan Isler and the robotic sensor networks group for their hospitality during our visit. The trip was made with funds granted by Tekna's master thesis stipend, for which I am immensely grateful. Thank you also to fellow students with which I have crossed paths in the past five years.

I would also like to acknowledge the contributions of my family in enabling me to dedicate the time to write this thesis, and a special thanks to Berit and Jon for your invaluable help with everything.

And finally, thank you dear Linn, for your hard work, support, encouragement and patience through five years of studies.

Ås, 19_{th} of May 2017

Eirik Solberg

Sammendrag

Dype nevrale nettverk for objektdeteksjon i landbruksrobotikk

av Eirik Solberg

Robotisering av arbeidsoppgaver i landbruket har potensial til å transformere matproduksjonen gjennom kontinuerlig overvåking av avlinger som muliggjør presis gjødsling, vanning og bekjempelse av ugress og sykdom m.m. . En slik forandring vil føre til et mer bærekraftig landbruk og økt matsikkerhet i fremtiden.

Denne masteroppgaven tar for seg bruk av dype nevrale nettverk til å detektere jordbær i videobilder med henblikk på å muliggjøre overvåking av plantehelse, estimering av avling og posisjon. Produksjon av disse dataene kan muliggjøre effektivisering av driften basert påinnsamlet data og ha stor verdi for jordbærbønder, og ogsåpåsikt kunne bidra til robotisert plukking av jordbær.

Basert på innhentet video fra en jordbrgård og bilder av jordbær lastet ned fra internett utarbeides det et datasett av jordbær merket med koordinater og tilstand. Et sett med klassifiseringsalgoritmer basert på dype nevralte nettverk trenes på enkeltbær fra datasettet og anvendes i en deteksjonsalgoritme. Til slutt anvendes dype nevrale nettverk for integrert objektdeteksjon på jordbærdatasettet.

Det utvikles dype nevrale nettverk som fungerer godt på jordbærdetekterings-oppgaven, og prosesserer video i sanntid på en datamaskin som kan integreres i mobile landbruksroboter.

Abstract

Deep Neural Networks for Object Detection in Agricultural Robotics

by Eirik Solberg

Robotization of tasks in the agricultural domain has the potential to transform food production through continuous surveillance of crops which can facilitate precise administration of nutrients, fertilizers and treatments for weeds and diseases. Such a transformation will increase the sustainability of agricultural practices and improve food security in the future.

This thesis applies deep neural network to the task of strawberry detection in video with a view to facilitate surveillance of plant health, crop estimation and logging positions of strawberries. The availability of such data can provide value for growers by enabling optimization of operations based on observed data, and facilitate progress towards robotic strawberry harvesting.

Based on videos sampled from a strawberry growing facility and strawberry images downloaded from the internet, a dataset of strawberries annotated with a state label and coordinates is developed. A set of classification models based on deep neural networks are trained on samples from the dataset and applied in a sliding window detection algorithm. Finally unified deep neural networks for strawberry detection are trained for the strawberry detection task.

Deep neural networks are shown to perform well on the strawberry detection task and real-time processing speeds are demonstrated on an embedded system.

Contents

List of Figures

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
AP	**A**verage **P**recision
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
FPS	**F**rames **P**er **S**econd
GPU	**G**raphical **P**rocessing **U**nit
ILSVRC	**I**magenet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
IOU	**I**ntersection **O**ver **U**nion
mAP	**m**ean **A**verage **P**recision
MSE	**M**ean **S**quared **E**rror
ReLU	**R**ectified **L**inear **U**nit
ROS	**R**obot **O** **S**ystem
SGD	**S**tochastic **G**radient **D**escent
SVM	**S**upport **V**ector **M**achine
PASCAL VOC	PASCAL Visual Object Classes (benchmark dataset)
YOLO	**Y**ou **O**nly **L**ook **O**nce

Symbols

	Physical constants	**Unit**
s	Time	Second
m	Distance	Meter

	Statistics	
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution	-
μ	Mean	-
σ	Standard deviation	-
σ^2	Variance	-
P	Probability	
p	Probability distribution	

	Computational graphs	
$\mathbb{G}$	Computational graph	-
$\mathbb{B}$	Computational graph	-
$\mathbb{A}^{(i)}$	Set of arguments to node i	-
$u^{(i)}$	Value of node i	-
$Pa(u^{(i)})$	Parents of $u^{(i)}$	-
$u^{(n)}$	Output of graph	-

	Machine learning notation	
h_θ	Model hypothesis	-
$\mathcal{H}$	Hypothesis space	-
$\hat{y}$	Estimate of function y	

θ	Parameter vector	-
$\theta_{j,i}^{k}$	parameter for node j in layer k from node i in layer $k-1$	-
ϵ	Learning rate	-
$f(x;\theta)$	Function of x given parameter θ	-
$J(\theta)$	Cost function	-
λ	Regularization parameter	-
$\mathbb{X}$	Design matrix	-
$x^{(i)}$	Feature vector for sample i in $\mathbb{X}$	-
$x_j^{(i)}$	feature j for sample i	-
b	Bias term	-
B	Mini-batch of data samples	
g	gradient	
z_k	Vector of summed inputs at layer k	
$a^(k)$	Acivation at layer k	-
$\sigma(z)$	Logistic sigmoid function of z	-
γ	Batch normalization shift parameter	
β	Batch normalization scale parameter	

Chapter 1

Introduction

1.1 Background

1.1.1 Agricultural Robotics

Agricultural robotics

Agricultural robotics is an industry which is expected to see significant growth over the coming years. Robots address several challenges in conventional farming which hold back efficiency and productivity such as workforce shortage, environmental harms caused by large machinery and lack of precision in applying pesticides. Developing agricultural robotic systems contributes to increased efficiency and is a key factor in sustainably increasing food production to meet increased demand in the future.

Thorvald agricultural robotics platform

The Thorvald agricultural robotics platform (figure 1.1) was developed by the Robotics and Control Group at the Norwegian University for Life Sciences. It is designed as a modular robotic system which is adaptable both in terms of it's shape and size as well as it's functionality. One current application of this system is the application of ultraviolet light to strawberry plants to prevent fungal growth.

Real-Time Robotic Sensing and Manipulation for Fruit Picking

This master thesis is done in parallel with the research project "Real-Time Robotic Sensing and Manipulation for Fruit Picking", which is a collaboration between the robotics and control group at The Norwegian University of Life Sciences and the Robotics Sensor Networks group at The University of Minnesota. The purpose of the project is to develop a robotic system capable of picking strawberries. This requires the development of computer vision algorithms to detect strawberries and compute estimates for their position in relation to the robot.

FIGURE 1.1: The Thorvald robotic platform in action at a table-top strawberry growing facility. Image provided by Erling Bjurbeck.

Applications of agricultural robotics in strawberry farming

One major benefit that robotics can bring to strawberry farming is continual surveillance and precision treatment of plant disease. In 2016, the Norwegian Farmer's association estimated that one third of the Norwegian strawberry harvest was lost to fungal infections from the strain *botrytis cinerea*[7]. Robotic precision farming could play a major role in tackling plant disease while reducing the use of pesticides. Robots could also be used by growers to deploy alternative remedies against fungus, such as ultraviolet light.

A robotic sensing system capable of detecting and logging information about individual fruits and berries will provide growers with a precise estimate of their crop size, health and state. Such data can be used to deploy optimal treatments of the crops based on their individual state, which could contribute to higher yields and eliminate waste. Growers could also aggregate the data and harvest their crops at an optimal time and produce valuable market and economical forecasts for their farming operations.

In the fruit and berry farming industry, the availability of seasonal workers for the harvesting season presents a major challenge. In the United States, it is estimated that the apple harvesting workforce is short about 20% [8]. Closing the gap with robotic systems could be achieved both by developing crop monitoring systems in order to deploy the workforce more efficiently, by developing systems capable of assisting the workforce by performing logistical tasks, or replacing the workforce altogether with complex systems capable of harvesting fruit and/or berries.

1.1.2 Computer Vision and Machine Learning

Visual detection of strawberries is a trivial task for human beings. When shown a digital image of a strawberry field, a human would quickly be able to identify strawberries by features such as color, texture or shape and determine the quantity of berries and position of individual berries.

Computer vision is the computer science field of engineering artificial visual systems to use images to comprehend and interpret the physical world [9]. To a computer a digital image is a 2-dimensional matrix of intensity values represented on a screen as pixels [10]. Features which are immediately identified by humans, such as shapes and textures, are not easily identified by computers, and it is necessary to use various image processing techniques to enhance them. These representations of the original images can then be translated into signals and be used in models capable of recognizing objects in images.

Traditional image classifiers require careful selection or hand-engineering of suitable feature descriptors for specific tasks. This type of object classifiers dominated the field of object recognition until 2012. In the 2012 edition of the "ImageNet Large Scale Visual Recognition Challenge" (ILSVRC [11]), an international computer vision contest, the authors of the paper "ImageNet Classification with Deep Convolutional Neural Networks" (AlexNet)[12] were able to achieve a vastly improved object recognition performance using deep neural networks, which have continued to dominate the competition ever since.

So-called "Deep Learning" is a special branch of machine learning which uses hierarchical layers of artificial neurons to mimic the activity in layers of neurons in the neocortex[13]. Artificual Neural Networks (ANNs) have been known for several decades, however their recent success stems from an increase of available datasets for research and improvements in computing power and algorithms which allow training of extremely large ANNs. Deep learning is not only successful in image recognition, it is also applied to tasks such as speech recognition and self-driving cars with incredible results.

1.2 Problem statement

1.2.1 Thesis main objective

The work in this thesis aims to research and develop methods for real-time strawberry detection using deep neural networks. The methods shall be capable of real-time processing and logging of detected strawberries as part of a mobile agricultural robotic system.

1.2.2 Thesis sub objectives

The following sub objectives have been identified as activities to be completed as part of fulfilling the main objective:

- Create a labelled dataset for strawberry detection.
- Develop strawberry detection algorithm using deep neural networks.
- Research published methods for deep learning based object detection algorithms and train and evaluate deep learning strawberry detection models.
- Deployment of the detection algorithm on an embedded system.

The structure of the thesis follows this list approximately, with the dataset development documented in chapter 3 and strawberry detection algorithms developed in chapter 4 and 5. Each chapter includes sections on the applied methods, results and a discussion of the results. An overview of the practical aspects of deploying the system is described in chapter 6.

Chapter 2

Theory

2.1 General Machine Learning Concepts

Machine learning is a subfield of computer science which focuses on algorithms that make predictions and estimates by learning from example data rather than be explicitly programmed. One definition of machine learning is stated below:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [14].

This thesis will focus on supervised learning algorithms where the task is to to learn some function based on a set of annotated example data. Although applying Deep Learning is a main objective, there are a number of important concepts which apply to machine learning algorithms in general which are presented in this section.

2.1.1 Machine learning tasks

Machine learning can be applied to a wide variety of different tasks. In general, machine learning is useful for performing tasks involving many input features, making it impractical or too difficult for to program a general solution. A rule of thumb for determining whether machine learning is suitable for solving a task, is that it should take a human being less than one second to evaluate the task [15].

In this thesis the task is to (a) identify strawberries and (b) to estimate their position in an image by predicting bounding boxes around them. Both of these tasks are trivial for a human and are demonstrably solvable tasks for a machine learning algorithm. This type of machine vision task is called object detection and can be framed as the combination of performing regression to predict the coordinates of a bounding box and classification to predict the object class.

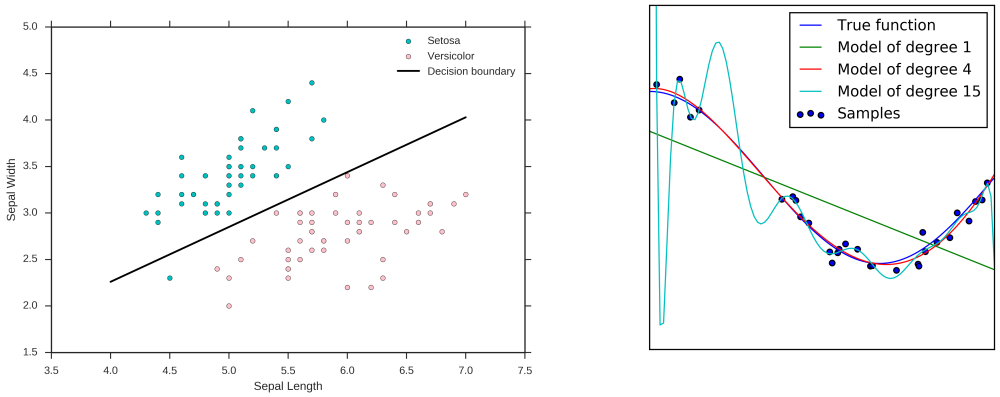


FIGURE 2.1: **Left:** An example of a logistic regression classifier trained on the iris dataset[1]. The model learns a decision boundary which separate the feature space into two regions. **Right:** A linear regression model approximating a function (blue) by fitting the sampled points (blue) to polynomial regression models of degree 1 (green), 4 (red) and 15 (cyan).

Classification is the task of learning to map an input vector to a category label. Another variant is to map an input to a probability distribution of labels. The learning algorithms learns to fit a line which separates the category regions.

Regression is the task of estimating a function which maps inputs to a numerical real valued output.

2.1.2 Learning algorithms

Machine learning algorithms are often formulated as a modular combination of a dataset, a cost function, an optimization procedure and a model.

Applying the machine learning algorithm yields a hypothesis $h_\theta \in \mathcal{H}$ where $\mathcal{H}$ is the set of functions we can draw hypotheses from, called the hypothesis space. The hypothesis space is determined by the type of model we select and the parameters of the model denoted θ. The objective of a learning algorithm is to learn the parameters θ which minimize the cost function $J(\theta)$. This is usually achieved by applying an optimization algorithm which updates the parameters θ, yielding a new hypothesis h_θ. This procedure is repeated until some performance criterion is satisfied and a final hypothesis is selected.

When faced with a machine learning problem, there are a number of different types of algorithms to choose from, and depending on variables such as the dimensionality of the data and dataset size, one method may be preferable to another. In the following a few different algorithms are briefly presented.

Linear Regression

Linear regression may be applied when the goal is to fit a numerical function to a set of data samples represented as features contained in a vector x and their corresponding target values y. Linear regression models take the form

$$\hat{y} = \theta^T x + b \tag{2.1}$$

Here θ is a vector of parameters θ_j, x is a vector of features x_i for one sample and b is the intercept term, also called the bias.

Linear regression is limited to modelling linear combinations of the input features x, however by choosing features $\{x_1 = x,\ x_2 = x^2,\ ...,\ x_n = x^n\}$ we may perform polynomial regression and fit higher order polynomials as well.

Optimizing a linear regression model can be done both analytically by applying the normal equations and iteratively by applying an optimization algorithm.

Logistic regression

Logistic regression is an algorithm for binary classification which outputs a number in $[0, 1]$ which can be interpreted as an estimate of the conditional probability $p(y = 1|x; \theta)$. The hypothesis formulation for binary logistic regression is

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+\exp\{-\theta^T x\}}$$

Logistic regression models improve by iteratively applying an optimization algorithm to update the parameter vector θ.

2.1.3 Gradient based optimization

Optimization is central to machine learning, and is most commonly achieved by an iterative procedure called gradient descent. In gradient based learning a loss function $J(\theta)$ which serves as a proxy measure for how well the learning algorithm is performing is specified. By minimizing the loss function, the learning algorithm indirectly improves. Cost, loss and error are terms used interchangeably to describe this objective function.

Minimizing the cost function is achieved by computing the cost function's gradient with respect to each parameter in the model and updating the parameters by the update rule

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\partial}{\partial \theta_j} J_\theta \tag{2.2}$$

This technique is called batch gradient descent. Although parameter vectors are often high-dimensional and hard to visualize, gradient descent is analogous to descending

a 3-dimensional terrain such as a mountain by taking ϵ length steps in the steepest downward direction.

2.1.4 Generalization

In order for a machine learning algorithm to work well, it must learn concepts and features from the training data that applies to samples that are not included in the training data as well. When an model is optimized on a training dataset, the objective of the model is to optimize the loss function as measured on the training set, however the true target is to obtain a model which generalizes to unseen data as well.

This generalization error is obtained by measuring it on a validation set consisting of previously unseen data drawn from the training set prior to training. This metric is called the validation error. For the relationship between our training error and validation error to hold, we make the so-called *i.i.d. assumptions* (Independent and Identically Distributed) about our training and validation datasets. Each example in the datasets is assumed to be independent from one another, and the training and validation sets are assumed to be drawn from identical probability distributions produced by the *data generating process.*

Capacity, underfit and overfit

The representational capacity of a machine learning model includes all the possible hypotheses (functions) contained in it's hypothesis space. For example, in the case of linear regression, we may choose any model on the form

$$\hat{y} = b + \sum_{i=1}^{n} \theta_i^T x^i \tag{2.3}$$

The model capacity is increased as the polynomial degree increases. Choosing a high enough polynomial degree allows the model to fit to the data perfectly. A model with an excessive hypothesis space reduces the likelihood of selecting a hypothesis that generalizes well, and is called overfitting. Choosing too small a hypothesis space reduces the likelihood that the hypothesis space contains any hypothesis that generalize well at all, and is called underfitting.

An algorithm which suffers from overfitting typically has a low training error and a high validation error. The corresponding case for underfitting is a high training error and a high validation error. Thus a hypothesis that performs well, will have both a low training error and a small gap between training error and validation error. This hypothesis is obtained by choosing a model with an appropriate capacity (figure 2.2).

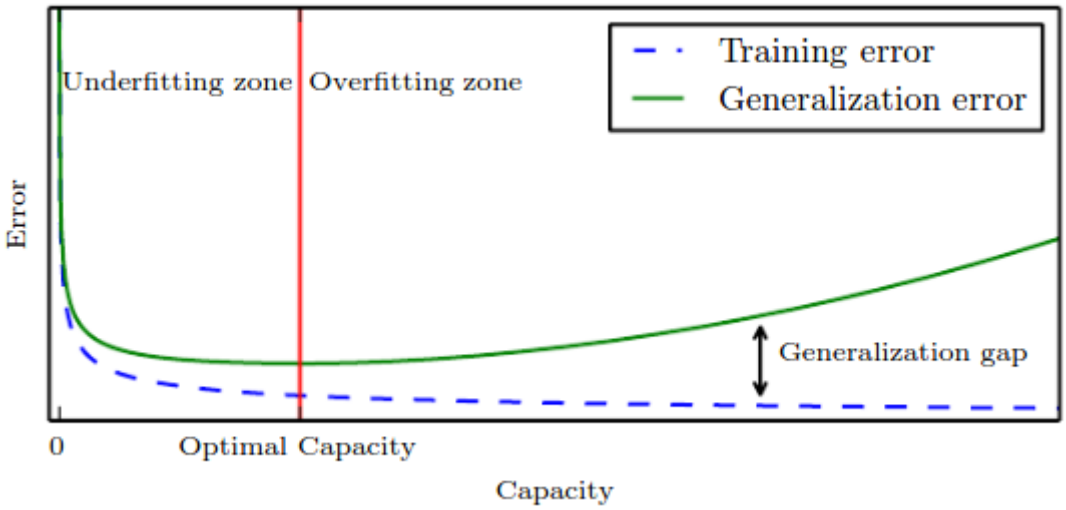


FIGURE 2.2: An illustration of the relationship between model capacity and training and generalization error. From Goodfellow et. al[2]

2.1.5 Regularization

Regularization is a term used for modifications made to a learning algorithm that is intended to reduce its generalization error but not its training error [2]. In simpler terms, regularization aims to reduce overfitting to the training data. In machine learning models, there are often enough parameters available to the model to effectively memorize the training data. For this reason it is important to employ regularizing techniques.

A common way to regularize parametric models is to add a penalizing term to the cost function $J(\theta)$ such that larger parameters are penalized. This can be achieved by adding the absolute value of the parameters to the cost function, or by adding the squared parameters to the cost function. These methods are referred to as L1 and L2 regularization respectively.

$$J(\theta) = J(\theta)_0 + \frac{\lambda}{2n} \sum_w \theta^2 \tag{2.4}$$

Regularized models are constrained to learn common patterns which occur often in the data, and are resistant to learning peculiarities of the noise in the training data [16]. Keeping weights small reduces the chance that a small number of parameters can have a large impact on our model's prediction, which in turn causes better generalizations.

Regularization techniques specific to deep neural networks will be discussed further in the section on artificial neural networks.

2.2 Deep Neural Networks

The earliest neural networks date back to the 1940s, when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on the workings of biological neurons and modeled a neural network using electric circuits. Various researchers made some progress and succeeded in constructing machine learning models throughout the 1950s and 1960s with the first neural network of multiple layers developed in the 1970s[17]. Ultimately other models became dominant and neural network research was mostly forgotten.

Deep learning is a term used to describe the machine learning techniques based on networks consisting of hierarchical layers of artificial neurons. Neural nets have received a lot of attention in recent years as it in 2012 suddenly emerged as the most powerful technique for classifying images[11]. Deep learning models have also demonstrated other incredible capabilities such as the ability to synthesise bodies of text[18] and the ability to drive a car in a real environment based on images from a front facing camera[19]. Modern neural networks may have more than a hundred layers and as inputs are processed through the layers, the networks produce increasingly abstract representations of the raw input data which enhance specific features of objects.

The recent progress in the field of deep neural nets is largely due to fairly recent research and development of the algorithms which govern the learning process, the availability of large datasets and an increase in parallel computing capabilities.

2.2.1 Feedforward Neural Networks

Feedforward neural networks are machine learning models in which artificial neurons are organized into hierarchical layers with the first layer being the input vector x. Artificial neurons, or units, are the elements of a neural network which perform the work of evaluating an input by performing mathematical operations on them and passing them on to the units in the succeeding layer. There are many different possible configurations of layers, such as convolutional layers, which will be discussed in further detail, but for now the layers can be viewed as one-dimensional vectors.

> "A feedforward network defines a mapping $y = f(x; \theta)$ and learns the parameters θ that result in the best function approximation." [2]

Artificial neurons

The input layer x contains elements x_i which are the raw values of the input data. As the input layer passes the values x_i to the j_{th} neuron in the k_{th} layer, each value x_i is multiplied by a parameter $\theta^{(}_{j,i}k)$ and a bias term b is added. Vectorized, this becomes

$$z_{k-1} = \theta^T x + b \tag{2.5}$$

The receiving unit then applies an activation function $a^{(k)} = \sigma(z)$ and obtains the activation a which is propagated forward to the units of the $(k+1)_{th}$ layer, which repeat the process until the values arrive at the output layer.

By convention the input layer, or the bottom layer, is given the index 0, and the intermediate layers between the bottom and top (output) layers are referred to as hidden layers containing hidden units.

2.2.2 Activation functions

The mathematical operations performed by the hidden units are called activation functions, based on the notion that they model the activation of an electrical signal in a neuron. Functions which have historically been common in neural networks, such as the logistic function, were chosen to resemble binary signals (on/off), however fairly recent research ([20], [21]) has demonstrated the efficacy of "Rectified Linear Units" (ReLU), which are considered a default recommendation for hidden units.

A primary role of the activation function is to introduce non-linearities into the neural network, without which the model would remain a linear combination of its inputs. The importance of the activation function comes from its impact on the ability to minimize the cost function and the ability to propagate signals through several layers.

Sigmoid (Logistic function)

The Sigmoid function is given by

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2.6}$$

Because the function returns a value on the open interval $(0, 1)$, it is commonly used as the final output in binary classification where the desired output is a Bernoulli distribution $P(y = 1|x)$. It is also continuous and differentiable, which is essential for updating the weight parameters.

When the value of $abs(z)$ grows, the slope of the function approaches zero and the sigmoid saturates. When used as an activation function for the hidden units, this property of the sigmoid may impair the learning algorithm's ability to make sufficient adjustments to it's weight parameters. When used as an output combined with the cost function $J(\theta) = -\log(P(y|x))$, it can be shown that the gradient only saturates when the model's estimate is correct (the model takes no action when no action is the correct action).

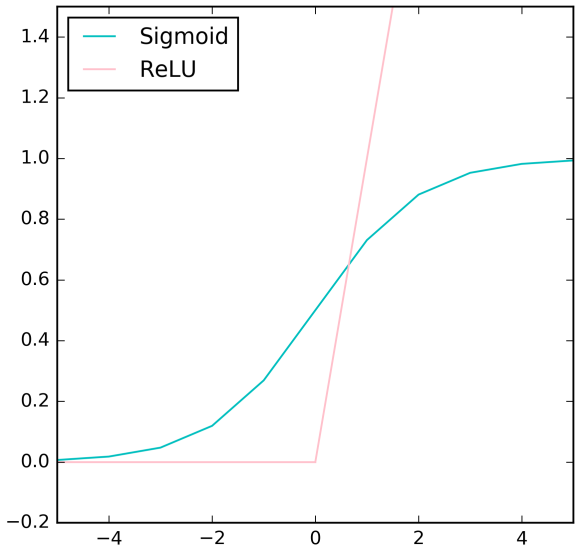


FIGURE 2.3: The sigmoid activation (blue) is approximately linear around zero, but saturates as the absolute value of x increases. The rectified linear unit activation ReLU (pink) is linear when $x > 0$, otherwise it is zero.

Softmax

The softmax function is given by

$$softmax(z)_i = \frac{exp(z_i)}{\sum_k^K \exp(z_k)} \tag{2.7}$$

The output of the softmax is a valid Multinoulli probability distribution $\hat{y} = P(y_i = 1|x)$ over the discrete classes y_i. Each probability returned is on the interval $(0, 1)$ and the vector sums to 1. Similarly to the sigmoid, the softmax may saturate. This happens when one value is much greater than the others, and to counter this a suitable cost function must be chosen. Softmax is a vectorized version of the sigmoid function.

Rectified Linear Unit (ReLU)

Rectified Linear Units compute the activation as

$$ReLU(z) = max(0, z) \tag{2.8}$$

A variant named leaky ReLU is also common:

$$leakyReLU(z) = max(0.1z, z) \tag{2.9}$$

ReLU units has demonstrated improved learning performance over saturating activation functions [12], and is the standard recommendation for hidden units. Although it is discontinuous and not differentiable at $z = 0$, this is easily handled by choosing the left- or right-sided derivative. Although this is analytically dubious, the input 0 from a digital computer is likely to contain numerical errors[2]. Derivation of the function is trivial, and gives a large gradient when the unit is active and 0 when it is inactive (or 0.1 if using leaky ReLU).

2.2.3 Regularization for Neural Networks

Deep modern neural network models have enough parameters that close attention should be paid to signs of overfitting. Unregularized neural nets generalize surprisingly well, and it has been conjectured that the dynamics of gradient based learning in multilayer neural nets has a self-regularizing effect[22]. That being said, applying regularizing techniques is standard and improves generalization. The L2-regularization described in the subsection 2.1.5 also applies to neural networks.

Dropout

Dropout is a regularization technique introduced by Hinton et. al.[23], in which the units in the network are randomly set to 0 with a probability P, usually 0.5, during training. In this way, the architecture of the network varies for each iteration of parameter updates, and the network can be viewed as consisting of several networks in one model. Dropout reduces complex co-adaptions of neurons by denying neurons the option of relying on the presence of other neurons [12]. At test-time with all neurons active, there are twice as many neurons active, and so the network weights are multiplied by 0.5 in order to obtain the mean signal of the neurons.

Batch Normalization

Batch normalization is a fairly recent innovation by Ioffe and Szegedy of Google Inc. [24] which seeks to reduce the change of distribution in internal network nodes (units) which they refer to as Internal Covariate Shift. The technique improves the flow of gradients through the network by reducing the gradients' dependence on the scale and initial values of parameters which permits the use of larger learning rates. Batch normalization drastically accelerates training of networks. Although it is not a regularizer, batch normalization has been shown to have a regularizing effect in neural networks.

The steps in the algorithm consist of normalizing the minibatch distribution so that it has a mean $\mu_B = 0$ and variance $\sigma_B^2 = 1$. Additionally it learns two parameters γ and β which scales and shifts the distribution. This last step in the algorithm leads the network to learn the most useful distribution. That is, if the network learns parameters $\gamma = \sqrt{\sigma_B^2}$ and $\beta = \mu_B$, then the network recovers the original distribution of the minibatch.

Algorithm 1 The batch normalization algorithm applied to an activation x_i over a mini-batch $B = x_{1...n}$

$\mu_B \leftarrow \sum_{i=1}^{m} x_i$
$\sigma_B^2 \leftarrow (x_i - \mu_B)^2$
$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}}$
$y_i \leftarrow \gamma\hat{x} + \beta \equiv BN_{\gamma,\beta}(x_i)$ **return** $\{y_i = BN_{\gamma,\beta}(x_i)\}$

Data Augmentation

One weakness of deep neural network algorithms is a need for relatively large amounts of annotated training data. In the case of images, the training dataset can be artificially expanded by performing various transformations on the image such as zooming, rotating, skewing or flipping an image. Augmenting the data alters the original, true probability distribution of the training set, so the augmentation operations should be limited to realistic alterations of the data (i.e. flipping an image of a number or a character will cause confusion in a character recognition algorithm, but flipping an image of a strawberry will contribute to invariance in classifying strawberries). This technique was successfully applied to the MNIST dataset for character recognition in [25]. They improved the current state of the art performance on the MNIST dataset using affine transformations and their novel method of elastic distortions, ultimately achieving an error of 0.4%.

2.2.4 Stochastic Gradient Descent

The large datasets used for training deep neural network algorithm are often impractical for batch gradient descent due to the memory requirements of both the millions of model parameters and the data itself. In order to circumvent this, stochastic gradient descent samples the gradient from a subset of the training set, processing the entire dataset of size m in mini-batches of size m'. The estimate of the gradient is

$$gradient = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} J(x^{(i)}, y^{(i)}, \theta) \tag{2.10}$$

And the parameters are adjusted so as to take a step of length ϵ in the direction of the estimated gradient:

$$\theta \leftarrow \theta - \epsilon gradient \tag{2.11}$$

RMS-prop

Finding a good minimum value for the cost function using stochastic gradient descent requires some trial and error. It is also necessary to adjust the learning rate during

training to achieve good results. The RMS-prop[26] is a variant of gradient descent which incorporates an adaptive learning rate. The learning rate adapts based on a running average of the previous gradients.

$$E[g]_t = \gamma E[g]_{t-1} + (1-\gamma) g_t^2 \tag{2.12}$$

$$\sigma_{t+1} \leftarrow \sigma_t - \frac{\epsilon}{\sqrt{E[g]_t}} \tag{2.13}$$

The effect of this running average is to dampen oscillations in directions orthogonal to the true gradient.

2.2.5 Cost functions and Maximum Likelihood

Maximum likelihood, also called the maximum likelihood method, is the procedure of finding the value of one or more parameters for a given statistic which makes the known likelihood distribution a maximum [27]. For a Bernoulli probability distribution, it can be shown that obtaining the maximum likelihood estimator $p_{model}(x|\theta)_{max}$ is achieved by minimizing the cross entropy between the model distribution p_{model} and the empirical distribution p_{data} [2].

$$-\mathbb{E}_{x\ \hat{p}_{data}}[\log p_{model}(x)] \tag{2.14}$$

Similarly, it can be shown for a Gaussian probability distribution that obtaining the maximum likelihood estimator $p_{model}(x|\theta)_{max}$ is achieved by minimizing the mean squared error (MSE).

$$\frac{1}{m}\sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}|| \tag{2.15}$$

For the purpose of learning algorithms, the maximum likelihood method provides a way to derive a cost function $J(\theta)$ given the task. For regression models, MSE (2.15) will be used, and for classification the cross entropy (2.14) will be used.

2.2.6 Computational graphs and the back-propagation algorithm

The parameter update rule in 2.11 requires the computation of the derivative of the cost function with respect to each of the numerous parameters in the model. These derivatives are computed by means of the back-propagation algorithm [28], without which the training of modern neural nets would likely be computationally intractable.

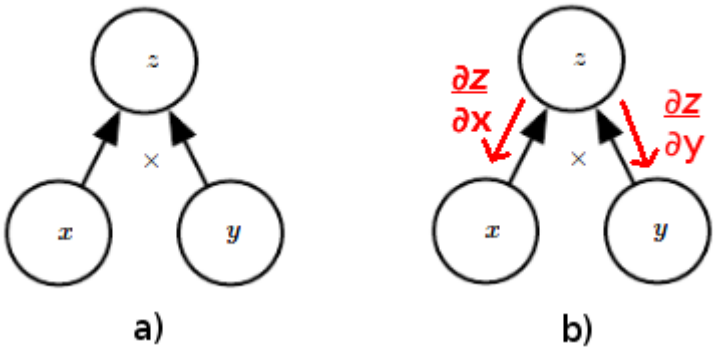


FIGURE 2.4: a) Example of the operation of multiplying two variables x and y represented as a computational graph $\mathbb{G}$. b) The red arrows show the edges of a subgraph $\mathbb{B}$, corresponding exactly to the edges in $\mathbb{G}$. The partial derivatives of the output z with respect to each node x and y is computed along the edges of $\mathbb{B}$.

Computational graphs

The process of computing the output of a neural net given an input can be represented with a computational graph. In a computational graph, each variable is represented by a node to which we apply simple functions called operations. An operation is represented by a directed edge from the input variable to a single output variable and is annotated with the type of operation performed.

Algorithm 2 Forward pass of an input through a neural network represented as a computational graph. The input vector x consists of n_i elements which are fed into the nodes $u^{(1)}, ..., u^{(n_i)}$. Each node computes a node $u^{(i)}$ by applying a function $f^{(i)}$ to the set of arguments $\mathbb{A}^{(i)}$ which comprises the previous nodes $u^{(j)}, j < i, j \in Pa(u^{(i)})$. The algorithm returns the output node $u^{(n)}$.[2]

for $i = 1, ..., n_i$ **do**
  $u^{(i)} \leftarrow x_i$
for $i = n_i + 1, ..., n$ **do**
  $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} | j \in Pa(u^{(i)})\}$
  $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$
return $u^{(n)}$

The back-propagation algorithm

The back-propagation algorithm follows the edges of the computational graph $\mathbb{G}$ described in Algorithm 2 backwards exactly, computing partial derivatives along the way by applying the chain rule recursively.

The algorithms presented here describe simpler implementations of the back-propagation algorithm than those in use in common software packages. The dimensions of the output *grad_table* from Algorithm 3 corresponds exactly to the parameters θ of the models in it's dimensions and contains the gradients g used in the update rule (2.11).

Algorithm 3 A simplified formulation of the back-propagation algorithm, computing a backward pass through a computational graph defined by the procedure in in Algorithm 2. [2]

Compute the forward pass by Algorithm 3.
Initialize $grad\_table$, a data structure that will store the derivatives that have been computed.
$grad\_table[u^{(i)}] = \frac{\partial u^{(n)}}{\partial u^{(i)}}$
$grad\_table[u^{(n)}] = 1$
for $j = n - 1$ down to 1 **do**
The next line computes $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$
$grad\_table[u^{(n)}] \leftarrow \sum_{i:j \in Pa(u^{(i)})} grad\_table[u^{(n)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$
return $\{grad\_table[u^{(i)}] | i = 1, ..., n_i\}$

2.2.7 Convolutional networks

When working with data types which have a grid-like structure, such as images, processing them in fully connected neural networks requires flattening the grid into a one-dimensional vector which causes spatial information to be lost. When working with vision tasks and image data, there are a few standard operations which are applied in order to retain and process this information efficiently.

Convolutional neural networks

The convolution operation is performed using an input (i.e. an image) and a kernel, which is usually a multi-dimensional array. For the purpose of machine learning and neural networks, the convolution operation is defined as

$$S(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n) \tag{2.16}$$

The kernel K, sometimes also called a filter is an array of learned parameters. The parameter values enhance certain features in an image such as edges or corners. Since the kernel is applied to the entire image, the entire image is processed using very few parameters compared to a fully connected network. The output of the convolution operation is called a representation or a feature map, and it is common to configure convolution layers with many filters in order to learn many useful representations.

The feature maps preserve information about the location of a feature in the image, and provides translational invariance. As input images are propagated through several layers of convolutions, the feature maps become more and more abstract representation of the original image.

Pooling layers

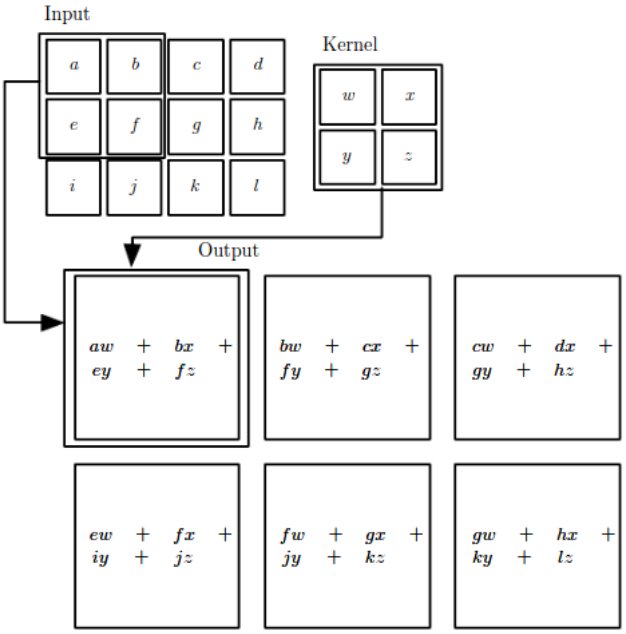


FIGURE 2.5: Convolution applied to a 3×4 input array using a 2×2 kernel. Figure from Goodfellow et al [2]

$$\textbf{max pool}\begin{pmatrix} -1 & 3 \\ 2 & -4 \end{pmatrix} \rightarrow 3$$

FIGURE 2.6: An example of a max pooling operation applied to a 2×2 array.

When using convolutional layers in a neural network, the convolution is typically followed by an activation as described in 2.2.2 and a pooling operation. The pooling operation replaces the activations of a rectangular neighbourhood with a statistic of that neighbourhood. One example of a pooling operation is the max pooling which returns the maximum value in the rectangular neighbourhood. The pooling layer creates invariance to small translations of activations, i.e. the value of the max pool output stays the same even if the activations shift slightly.

Pooling can also be applied using a larger than 1 pixel step size between pooling regions. The pooling operation then reduces the size of it's input dimensions which reduces the number of computations in succeeding layers.

2.2.8 Deep learning for vision tasks

High-dimensional data

Digital images are usually stored on a computer as $3 \times width \times height$-sized arrays, each of the three channels representing the pixel intensities for red, green and blue colors respectively. With each pixel comprising a dimension of data, the number of possible

a)

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

b)

c)

FIGURE 2.7: The figure shows **a)** an input image, which is processed with a convolution operation using the **b)** Sobel edge filter and the **c)** resulting image from applying the edge detection filter.

configurations of data, or the image space, for an image is enormous, even for relatively small images. Considering all the possible images one could generate by randomly choosing pixel values, it becomes apparent that images containing objects are very rare, and that the distances in image space between images of the same object type may be very large. Deep neural networks perform very well on such high-dimensional data.

Manifold learning

A manifold is a topological space which is locally euclidean. One example is the earth's surface, which is locally 2-dimensional from the viewpoint of a human being, but actually resides in 3-dimensional space when you zoom out a bit. In machine learning the manifold hypothesis is the concept that most valid and interpretable configurations of input data lies near a collection of manifolds containing a small subset of points. For example, one can imagine a "cat manifold", which contain all the points that represent images of all cats. It is then possible to alter any cat image in small steps along the manifold to obtain any image of a cat which also lies on the "cat manifold".

Image filtering

Image filters are the primary tool for extracting useful information from raw pixel intensity values in an image, and are used as the kernels in the convolution operations (equation 2.16) discussed previously. One common image filter is the Sobel edge filter shown in figure 2.7. The conventional approach to object recognition tasks, is to select or engineer a set of such filters for the application which extract the most informative features from the image.

In convolutional neural networks, the filters are learned rather than selected, which means the filters a network converges on are generally useful for extracting information.

Representations in deep networks

In a CNN, each layer is configured with some number of filters for each layer. Adding layers to a network then allows the network to learn increasingly complex combinations of filters as the network grows deeper. These deep representations are provided to the output layer, for example a fully connected layer, which is in principle a linear classifier.

One useful property of neural networks is that representations learned by neural nets can be reused between different classes which share certain features. For example, a feature map with strong activations for eyes can be used to detect both humans and animals, and an additional feature map with activations for fur can help decide which class is correct. This property is called distributed representations and make for powerful classification models.

Transfer learning

Deep representations can also be transferred to entirely separate tasks than the one they were trained on. Classification models are often framed as a feature extraction module combined with a classifier. By training classification models on large datasets, good feature extraction models can be obtained and transferred to entirely different tasks or sets of objects. This is referred to as transfer learning and has been shown to improve generalization for a model initialized with pre-trained parameters versus one initialized with random parameters[29].

2.3 Performance metrics

In evaluating the models the following performance metrics will be used

Classification accuracy

The classification accuracy A is the fraction of correctly classified examples

$$A = \frac{p}{N} \tag{2.17}$$

Where p is the number of correctly classified examples in a set of N examples.

Intersection over union

When performing detection tasks, the estimated bounding box area $\hat{B}$ is evaluated by comparison to the ground truth bounding box area B by intersection over union (IOU):

$$IOU = \frac{\hat{B} \cap B}{\hat{B} \cup B} \tag{2.18}$$

Precision

The precision metric is computed by

$$Precision = \frac{Number\ of\ correct\ classifications}{Number\ of\ possible\ classifications} \tag{2.19}$$

Recall

The recall metric is computed by

$$Recall = \frac{Number\ of\ correct\ classifications}{Number\ of\ possible\ correct\ classifications} \tag{2.20}$$

Chapter 3

Dataset development

3.1 Data collection

3.1.1 Robotic sensing setup

The envisioned strawberry detection system should have the capability to detect and track instances of strawberry by processing 2-dimensional, RGB video frames in real-time. Although strawberries are often grown in rows planted in the ground, the methods developed in this thesis is meant to be applied in a table-top strawberry growing facility such as that shown in figure 1.1. Tables are organized in rows, with the plants situated about $1.5m$ above ground level. The robotic system performs various tasks along these rows, such as robotic berry picking, which allow the crop monitoring to be performed simultaneously as a secondary task. The monitoring of strawberry plants is done with a camera mounted at an angle of about 45 deg below the horisontal, so that the strawberries are minimally occluded by the canopy.

3.1.2 Data sources

The primary source of strawberry images is video filmed at a table-top strawberry farming facility in Tasmania. The video mimics the envisioned robotic sensing setup described in subsection 3.1.1 with one handheld cellphone camera viewing the strawberries from the side and angled slightly upwards. The variation in the cameras vertical height contributes to

Since the videos are filmed in early spring, the majority of berries are green, and there are fewer clusters of berries present than what is to be expected in the growing season.

Strawberries are also included as one of the object classes for the ImageNet Large Scale Visual Recognition Challenge [11]. These images consist of ripe berries for the most part,

and most of the images include clusters of berries (i.e. in baskets or bowls). The images are representative of the images returned in an internet image search for "Strawberry".

3.1.3 Data labels

The data labels and bounding boxes were registered using the open source software Sloth[30]. The data available for the experiments has been annotated with one of four category labels:

- Ripe strawberries
- Non-ripe strawberries
- Cluster of strawberries
- Not a strawberry (i.e. Background)

A classifier capable of distinguish between these four categories can be used to estimate the total number of strawberries in the field in addition to identifying ripe berries ready for picking. The cluster category can be used to identify image regions that require further processing (i.e. for segmentation). The "Not a strawberry" category is needed to train the classifier on negative examples.

3.1.4 Bounding boxes

In order to register the location of different occurrences of strawberry in images, each class instance is labelled with the pixel coordinates for the top left corner of a bounding box and the box's width and height. The boxes are represented as a numerical array on the format

$$[x \quad y \quad width \quad height]$$

3.2 Dataset

For the dataset, frames were extracted from the strawberry videos at a rate of 10 frames per second (FPS). Each frame was visually inspected and strawberry instances were annotated with class labels and bounding box coordinates. Berries are labelled if they are more than approximately 50% visible. Berries which are uniformly red are labelled as ripe, otherwise they are labelled as non-ripe. Clusters of berries are labelled as such if 3 or more berries are inseparable or occlude each other. This part of the dataset consists of 1285 images.

The part of the dataset sourced from Imagenet was labelled in the same manner, however images containing large amounts of berries (for example in a bowl, basket etc.) were discarded. The Imagenet dataset consists of 693 images.

3.3 Discussion

Obtaining enough data which is varied enough that it is possible to learn a model which generalizes well to unseen data can be a challenge in applying deep learning. In this respect, it is worth noting a few challenges which should be addressed when developing the machine learning models.

The part of the dataset obtained by extracting frames from video will over a sequence of frames contain several instances of the same individual strawberry as it moves across the frame. Although the strawberry is the same individual strawberry as in a previous frame, the perspective, lighting and pixel values measured by the camera sensor changes slightly, and so it should be considered a separate sample. These samples likely aren't sufficient to provide a representative variation which generalizes to all strawberries, however the supplemental Imagenet strawberries helps to rectify this.

Since the Tasmania video dataset contains mainly non-ripe strawsberries, and the Imagenet strawberries contain mainly ripe strawberries, it is possible that the models learned from transfer poorly to the test setting (growing facility). One strategy to circumvent this lack of data is to concatenate the two classes and develop a binary strawberry classifier. The more fine-grained classification of ripeness can then be postponed until more data becomes available. The two sets of data then provide complementary samples of strawberries from a wider distribution of strawberries as a whole.

Chapter 4

Object detection with a sliding window

4.1 Motivation

In this chapter, a sliding window object detection algorithm which uses a neural network both as a classifier and a regression model to fine-tune the detector's position estimate is developed. Sliding window object detection requires the algorithm to extract patches in a grid over the entire image and evaluating the content of each patch. The evaluation of the contents of each section is commonly performed using computationally economical feature descriptors. Several such feature descriptors exist, and it is also possible to engineer filters which accomplish a specific task well.

Applying convolutional neural networks (CNN) to object detection tasks circumvents the process of selecting or engineering feature descriptors entirely. Neural networks learn feature descriptors by means of gradient descent and back-propagation, and have in recent years surpassed human performance on the Imagenet benchmark dataset for classification. This experiment explores the use of relatively small neural networks as the classifier module of a sliding window object detection algorithm.

This approach to object detection is quite naive and computationally expensive at test-time, but has several advantages in the training and development phase. Restricting the image classifier to small patches of 60×60 pixels, means the computational cost of training the neural network to convergence is manageable even on a CPU in a matter of hours rather than days [1]. The use of the single class image patches as training examples also has some significant advantages for the dataset, as we may both include a large amount of supplementary examples taken from the other sources (i.e. any image of a

[1] Graphical Processing Units (GPUs) allow a greater extent of parallel computation and are orders of magnitude faster at this type of task than Central Processing Units (CPUs).

strawberry) and employ several modes of data augmentation. Lastly, this approach will yield insights on the efficacy of deep learning in this particular domain.

4.2 Methods

4.2.1 Object Detection Pipeline

The object detection algorithm is to be applied to single frames taken from a video stream. Since strawberries vary in size and distance from the camera, making detections at different scales is necessary. This is achieved by generating an image pyramid for each image frame. Image pyramids are data structures containing copies of the image frame at several scales. The algorithm shall process each image frame from the image pyramid by extracting all patches in a grid over the input image and process each patch with a deep neural network classifier and location regression. To discriminate between detections with overlapping bounding boxes, a non-maxima suppression is applied. The output of the algorithm is a set of bounding box coordinates and confidences for the detected strawberries.

The algorithm ensures that the final set of bounding boxes contain the regions of the image with the highest confidence scores for strawberries.

4.2.2 Model architecture

The neural network has a feature extraction module, a classifier module and a regression module. The input to the neural network is a $60 \times 60 \times 3$ array.

Feature extraction was performed with a convolutional neural network. Each convolutional layer is followed by rectified linear unit activation, a batch normalization layer and a max pooling operation. The filter sizes are kept at 3×3 pixels for all layers. For each layer added to the model, the number of filters is doubled. The max pooling operation is used as a compression mechanism reduce the size of the feature vector passed to the fully connected layer.

The architecture of the convolutional neural networks for image classification follows the template of a module of convolutional feature extraction, followed by fully connected layers which connect to the output layer consisting of a sigmoid unit. Four different architectures were trained and evaluated for use in the detection algorithm.

In order to improve the location estimate of the sliding window classifier, a regression model based on a similar architecture was trained. The regression model outputs two pixel values for the offsets (x, y) relative to the location of the frame it is evaluating.

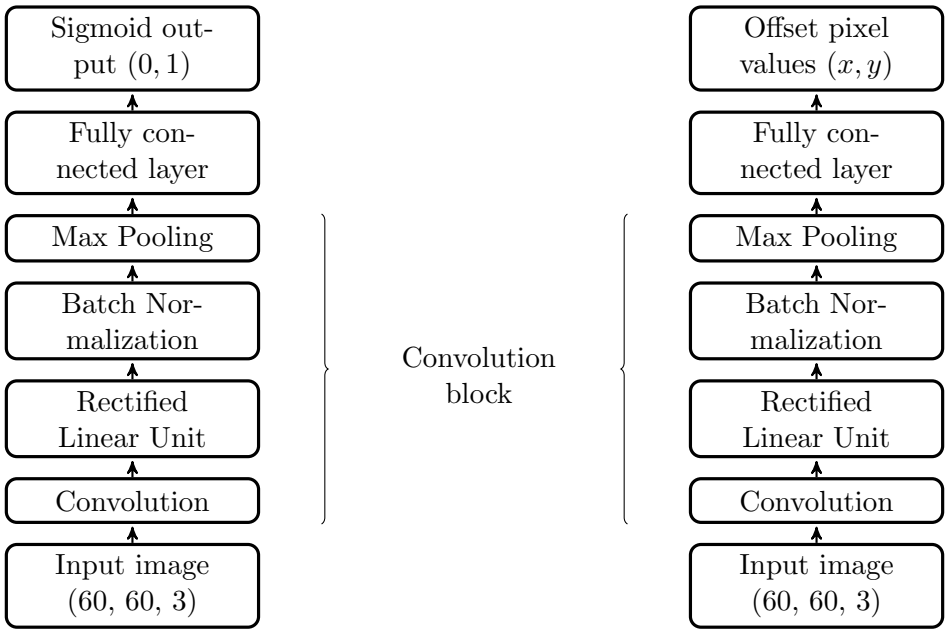


FIGURE 4.1: The architecture of the Convolutional Neural Network. The four layers starting with the convolution layer and ending with the max pooling layer makes up a convolution module. This module is repeated three times for the deepest model, and one time for the shallowest models.

A successful regression model allows the sliding window to take longer strides between evaluations, since it can regain the localization accuracy by computing the offset.

4.2.3 Training data

Preprocessing

Images are three-channel RGB, resized to $60 \times 60 \times 3$ pixels. The pixel intensity values were rescaled from the interval $[0, 255]$ to the interval $[0, 1]$.

Classification dataset

The classification dataset consists of positive (i.e. image patch contains a strawberry) and negative sample patches extracted from the labelled dataset of strawberry plants and positive strawberry samples from the Imagenet dataset. The strawberry samples used are both the ripe and non-ripe labelled berries from the dataset developed in chapter . The choice of developing a binary classifier rather than a multi-class classifier was made due to the limitations identified in the dataset in chapter 3.

The dataset was split into three sets. Approximately 70% of the data for the training set, 15% for each of the validation and test sets. In training, the samples drawn from the training set were augmented with random transformations to increase the number of samples. Allowable transformations are rotations of up to 10 deg, and horizontal and

FIGURE 4.2: The leftmost image shows an image patch containing the original strawberry image. The remaining three images show the same image patch transformed with random horizontal and vertical flips, and a modest amount of rotation. This augmentation regime artificially expands the dataset with observations which represent reasonable variations which could occur naturally.

FIGURE 4.3: Examples of the generated training data for the regression task. The leftmost image shows the original image patch with a strawberry positioned approximately in the center. The three remaining examples show the same image patch randomly offset in the x and y directions.

vertical flips, all of which are randomly combined when drawing a sample. Examples of the transformations are shown in figure 4.2.

Bounding box regression dataset

Regression samples were generated from the positive sample patches in the classification dataset. For a given sample, it is assumed that the strawberry is approximately centered in the patch. Random fractions drawn from a uniform distribution over the range $[-0.40, 0.40]$ of the image were cropped from the top or bottom and from the left or right side of the image. The resulting strawberry image patch is off-center, and the cropped fractions are converted into pixel unit offset values $[x, y]$ for the sample patch to be used in training. 10 samples were generated for each strawberry image in the classification dataset. Examples of the generated data is shown in figure 4.3

4.2.4 Training and optimization

The CNNs were trained using the RMSprop variant of stochastic gradient descent on batches of 64 images.

The loss function was selected by the principle of maximum likelihood. Mean squared error was used for the regression model and cross-entropy for the classification model.

TABLE 4.1: Test results for the different architectures.

Model	**Description**	**Batch processing time** [s]	**Test accuracy** [%]
Model 1	One convolution block, no fully connected layer	0.589	54.9
Model 2	One convolution block, one fully connected layer	0.624	98.5
Model 3	Two convolution blocks, one fully connected layer	0.692	98.7
Model 4	Three convolution blocks, one fully connected layer	0.697	98.4

4.2.5 Regularization

The models were regularized with dropout applied to the fully connected layer with probability $p = 0.5$. Additionally the applied batch normalization has a regularizing effect.

4.2.6 Model selection

In training the model, training was suspended when the validation loss showed no improvement over 3 training epochs (three full cycles of the entire dataset). This helps to avoid overfitting to the data.

The different architectures which were tested are evaluated with regards to their accuracy on the test set as well as their computation time for test samples.

The selected model is implemented in a sliding window algorithm as the classification module.

4.2.7 Evaluation

The developed algorithm was evaluated using intersection over union and recall measured on a test set. A detection threshold of $IOU <= 0.20$ was used to qualify successful detections.

4.3 Model development

4.3.1 Classification model

Four models of varying feature extraction depth were trained on the task of classifying strawberry images. The training plots are shown in figure 4.4.

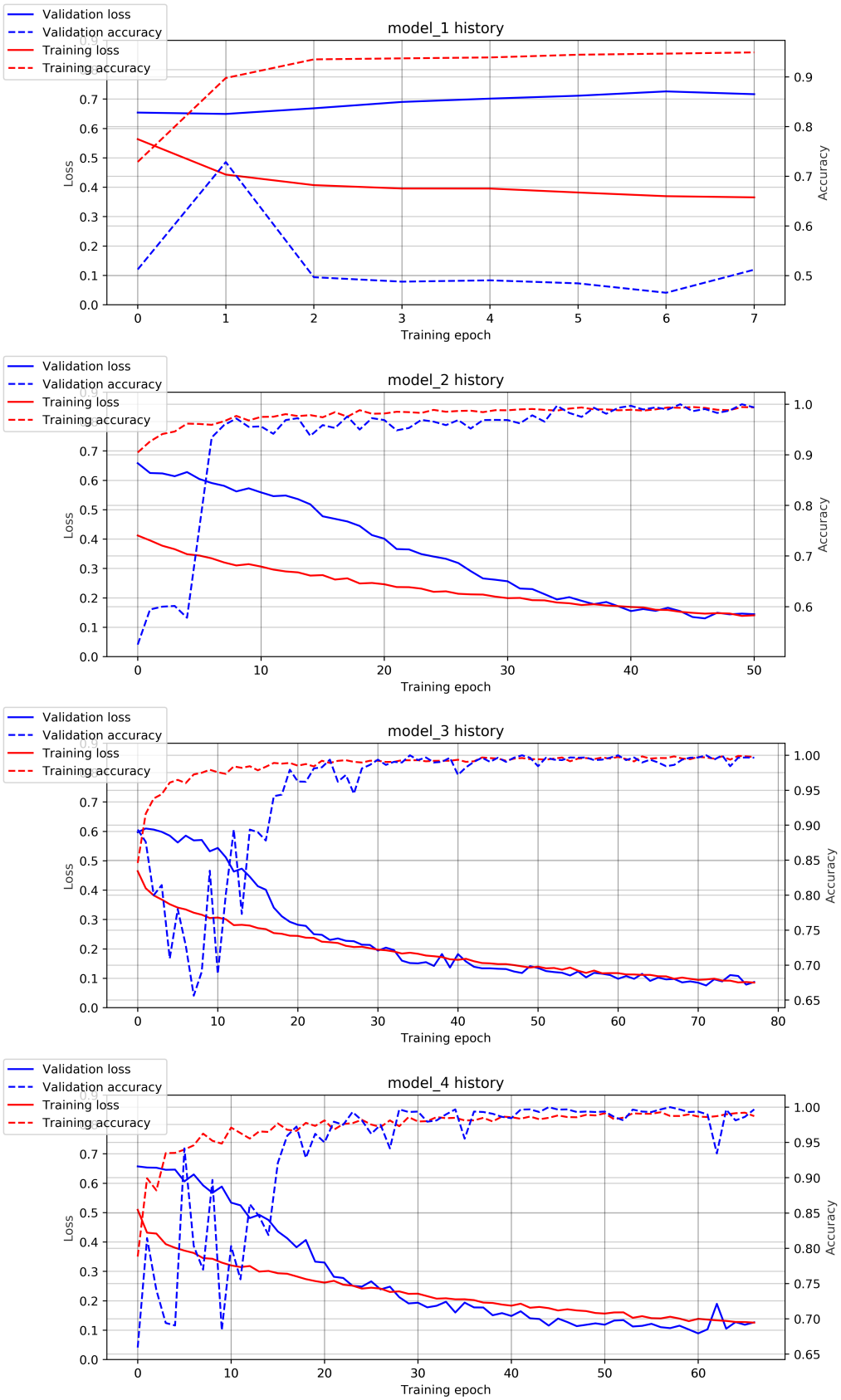


FIGURE 4.4: The training plots for the different architectures. Every model except model 1 converges. The validation metrics correspond well with the training metrics, and training progressively decreases the loss.

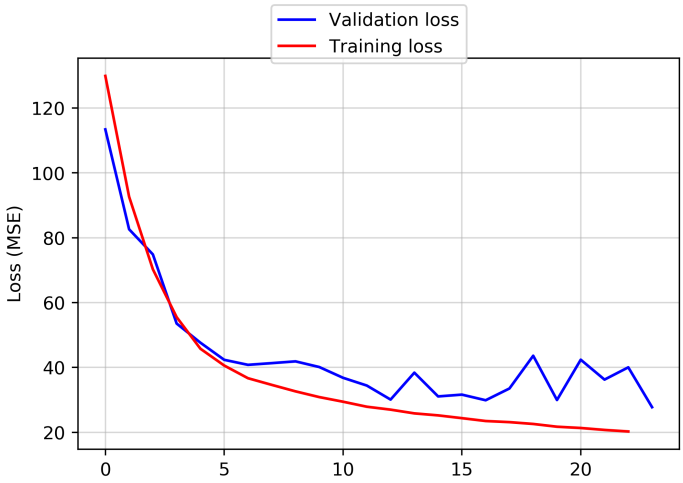


FIGURE 4.5: Training plot for the regression model showing the mean squared error for the on the training data (red) and the validation data (blue). The training of the model is stopped as the validation error plateaus around $MSE = 30$. The regression model's error on the test set is $MSE = 28.67$.

The comparison of the validation results in table 4.1 show negligible difference in the classification accuracy on the test set.

4.3.2 Regression model

The task of the regression model is to provide an estimate of the location of a detected strawberry relative to the location of the current sliding window location. The training progresses with decreasing training and validation loss as seen in figure 4.5.

4.4 Algorithm development

4.4.1 Algorithm parameters

The image pyramid is a data structure which contain copies of an image scaled by some fixed ratio. The purpose of generating image pyramids is to facilitate detections of strawberries with a fixed input classifier regardless of their apparent size in the image. To maximize the likelihood of detecting strawberries, the initial image size and scaling factor for the pyramid is determined based on the statistics of the training data. The distribution of relative width of the labelled image patches is shown in figure 4.6. Based on the data, the parameters for the image pyramid are set to include the range $(0.111, 0, 375)$ with a scaling factor of 1.5.

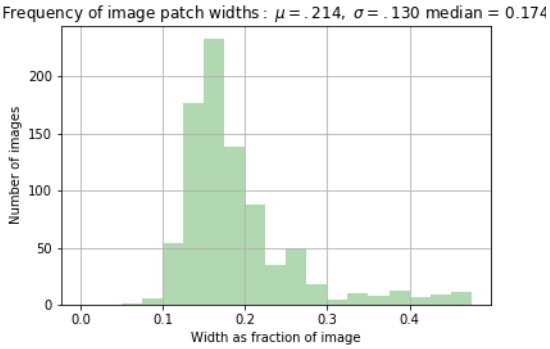


FIGURE 4.6: Distribution of strawberry image widths as a fraction of the video frame widths in the dataset.

The sliding window is applied with a stride length of 15 pixels, or a 75% overlap with a classifier input size of 60×60 pixels. This reduces the number of computations compared with evaluating patches from every possible location, but reduces the localization accuracy.

4.4.2 Implementation

The object detection algorithm as implemented is presented in 4.

Algorithm 4 Pseudocode for the siding window object detection algorithm

Input: One image frame
Output: $\mathcal{B}$, set of bounding box coordinates $b_{i,j} = \{x_{i,j}, x_{i,j}, w_{i,j}, h_{i,j}, c_{i,j}\}$ for detections with confidence $c_{i,j} > threshold$.
s : Stride length
$Pyramid \leftarrow$ rescaled copies of I
for Each image in $Pyramid$ **do** $i, j = 0$
  for Every s_{th} row in image **do**
    for Every s_{th} column in image **do**
      Extract square patch of dimensions 60×60 pixels.
      $c_{i,j} \leftarrow$ Classification confidence computed by neural network
      if $c_{i,j} > threshold$ **then**
        Compute bounding box location offset values (x_o, y_o)
        $\mathcal{B} \leftarrow b_{i,j} = \{x_i + x_o, y_i + y_o, c_{i,j}\}$
      $j = j + 1$
    $i = i + 1$
for Each box and confidence $b_{i,j} \in \mathcal{B}$ **do**
  for Each box $b_{n,m} \in \mathcal{B}$, $(n, m) \neq (i, j)$ **do**
    Compute Intersection over Union, $IOU(b_{i,j}, b_{n,m})$
    if $IOU > threshold_{IOU}$ **then**
      Discard box with lower confidence c
return $\mathcal{B}$

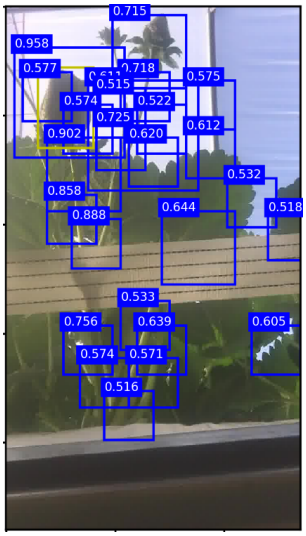


FIGURE 4.7: Before hard negative mining

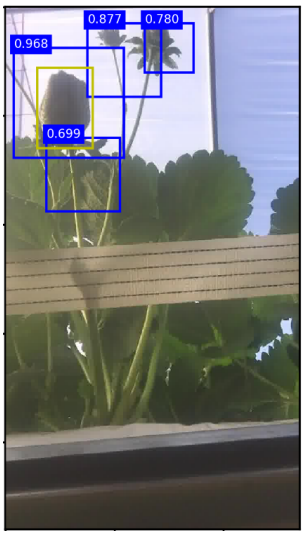


FIGURE 4.8: After hard negative mining

FIGURE 4.9: The effect of expanding the training dataset with hard negative mined examples was to drastically reduce false positives.

4.4.3 Fine tuning

After training the model to convergence according to the early stopping policy, the classification model suffered from a large number of false positive classifications. To counter this, hard negative mining was performed as follows: The sliding window detection algorithm was run on a set of negative frames (i.e. frames containing no strawberries). False positive detections from this set was then added to the training set as negative (background) samples. The hard-negative mining samples significantly reduced the number of false positives, as shown in figure 4.8.

Performing non-maxima suppression on the sets of bounding boxes output by the algorithm proved difficult. The model tended to saturate and output values equal to 1.0 for most detections due to the use of a sigmoid input. To counter this, the model was retrained with a batch normalization prior to the sigmoid activation. This modification successfully narrowed the distribution of outputs to a more centered distribution.

4.5 Results

4.5.1 Detection performance

The three model architectures were tested as the classifier in the detection pipeline. To measure the performance of the different models, their performance with regards to

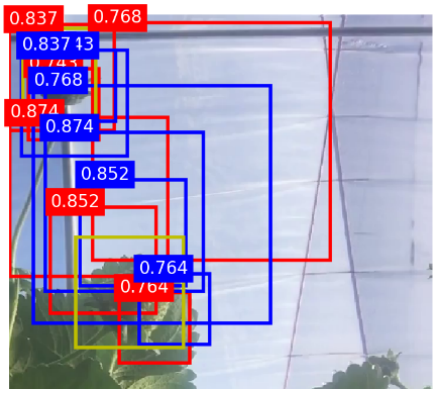


FIGURE 4.10: Detections with no suppression of duplicate bounding boxes.

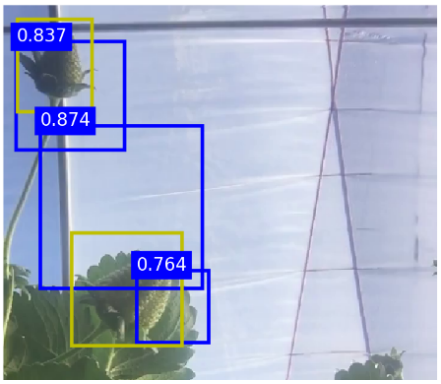


FIGURE 4.11: Detections after suppressing duplicate detections (no regression results are shown).

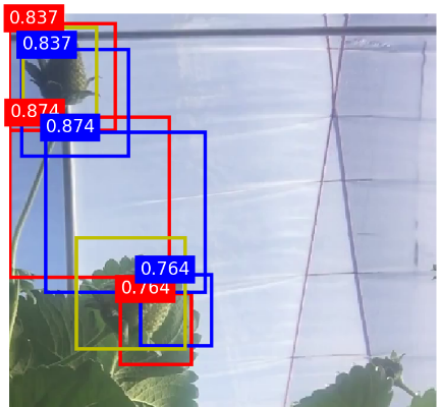


FIGURE 4.12: Final detections, both with and without regression offsets.

FIGURE 4.13: The figures show sample strawberry detection results before and after the suppression of overlapping boxes. Yellow bounding boxes represent ground truth boxes, blue bounding boxes represent the detections from the sliding window grid, and red bounding boxes represent detections offset with estimates from the regression model.

TABLE 4.2: Performance metrics for the models. Increasing model depth seems to impact IOU and recall positively.

Model name	**Average IOU**	**Recall**
Model 2	0.120	0.871
Model 3	0.142	0.781
Model 4	0.173	0.910
Model 4 with regression model	0.145	0.850

average intersection over union and recall were measured on a labelled test set. The results are shown in table 4.2.

The deeper model produces the best detection results when applied to the test images as a sliding window.

4.5.2 What the neural networks have learned

Although the different model architectures managed to discriminate between strawberry and background images with near perfect accuracy in training, this level of performance transfers poorly to sliding window detection. In figure 4.18, the feature maps taken from the activation layers in the deepest model, model 4, are plotted. Inspecting the model's response to different types of data allows us to visually inspect which features the model has learnt to recognise and spot potential failures or areas of improvement.

In figure 4.19 model generated input images are shown. The images are produced by back-propagating gradients from individual filters and obtaining the gradient of the input with regards to the input image. Starting from random noise and adding these gradients produce the images shown.

4.6 Conclusion

4.6.1 Conclusion

The task of classification is found to be performed well using even shallow neural networks of only one convolution layer. When applied as a sliding window, the deeper models perform better with regards to IOU overlap with ground truth boxes and recall. The regression model which was intended to provide estimates of the location of positive detections and allow a sparser evaluation of the input image actually worsened the detection performance of the best model.

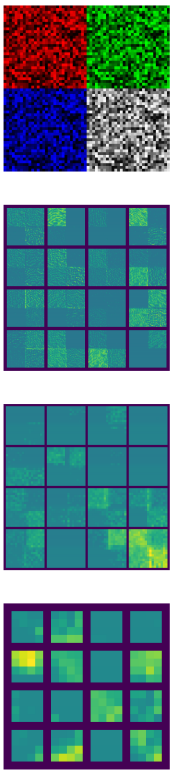


FIGURE 4.14: Feature maps over random noise, as individual color channels (R/G/B) and 3-channel RGB noise (bottom right).

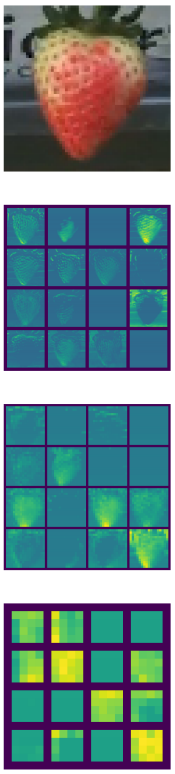


FIGURE 4.15: Feature maps over a red strawberry.

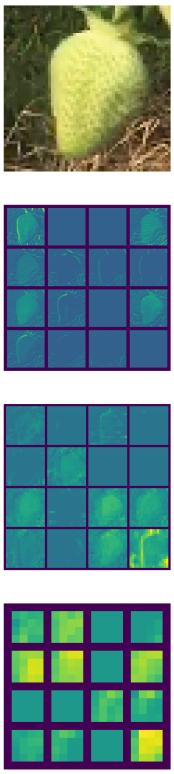


FIGURE 4.16: Feature maps over a green strawberry

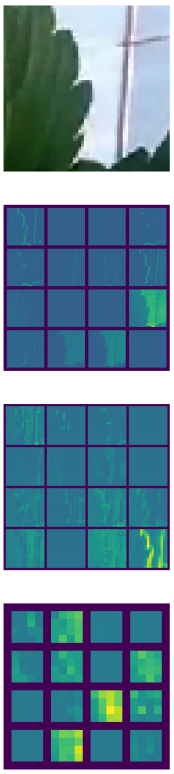


FIGURE 4.17: Feature maps over a false positive detection.

FIGURE 4.18: The figures show the activations in a random set of filters drawn from each of the three convolution blocks from Model 4. The filters are comparable between the images, but there is no (visible) relation between filters in different layers. Brighter pixels correspond to stronger activation levels. Studying these activations makes it apparent that some of the feature maps in the first convolution highlight the strawberry well. The feature maps over individual color channel show that the model contains several filters which respond to colors. Hint of an edge filter can also be seen in the top left corner of the first set of filters.

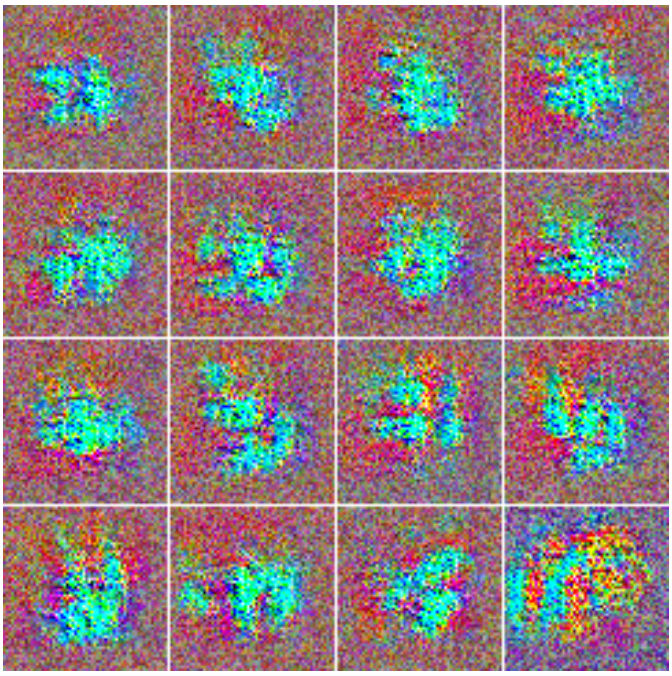


FIGURE 4.19: This figure shows images generated from the activations of 16 filters in the fourth convolution layer of model 4. The images are produced by forward propagating a 60×60 pixel RGB image consisting of random noise to the relevant filters. Setting the cost function as the mean of the filter's activations and performing back-propagation through the computational graph gives the gradients of the input image with respect to the activations of a filter. The input image is updated by adding the gradients to the input image's pixels.

.

4.6.2 Discussion

A key finding in this chapter is that the number of convolution blocks used in training the classifier had little impact on the training, validation and test performance. The feature maps in figure 4.18 show that some of the features extracted after the first convolution block highlight both red and green strawberries fairly well. This result suggests that deep neural networks have a larger capacity than necessary for the task of detecting strawberries, and that carefully selecting and engineering filter kernels could possibly achieve the same level of performance in a significantly more compact and computationally economical classification model.

The images in figure 4.19 show some hint of strawberry texture and shape, and also have strong activations for red. The strong blue/green activations show that the model has learned to look for strawberries in the center of images, and also explain some typical false positives observed containing bright blue spots. Adding more negative samples of this sort would likely improve these failures.

The slightly better IOU achieved by the deeper model when inserted in the detection algorithm may suggest that a deep model might produce more precise location estimates

by leveraging the abstract features of the deeper layers (some of the activations of the third convolution layer shown in 4.18 show hints of this).

Adding the regression model did not improve the model performance, in fact it decreased the performance. The model converged in training and produced similar loss in terms of MSE on both the validation and test sets, however the training data transferred poorly to the data encountered with the sliding window algorithm. A likely explanation for the failure of the regression model is that the approach taken in generating the data is fundamentally flawed in several ways. The approach taken in generating data assumed that all berries were centered in their patch on average, and simulated translation of berries by cropping the image. This way of generating data produced a lot of noise because it failed to account for the various aspect ratios and orientations of strawberries encountered in the dataset. Because of the zoom effect, it also produces an error in the labelled outputs which is proportional to the cropped fraction.

Applying a convolutional network classifier as a sliding window over image pyramids produces poor detection results in this implementation. Evaluating the image at every possible location would likely improve the detection results significantly, but isn't a feasible approach because of the computational cost.

4.6.3 Further work and improvements

Using a classifier as a sliding window over an image combined with a regression model for localization is a valid way of performing the task of object detection, however the algorithm developed and presented in this chapter has room for improvement in several areas.

Primarily, the regression model and classifier should train on the same data. A model could be trained to output both the class and bounding box coordinates for a sampled sliding window, with a joint cost function for both types of output. This would reduce the computational cost by using only one feature extraction module as the base for both the classification and regression output. To obtain training data more similar to the data encountered in the test setting (sliding window), the data for this approach could be sampled from labelled images similar to the way the patches were sampled, but with the patches randomly translated to obtain patches containing translated strawberries. Sampling the data in this manner would preclude data augmentation, however it would allow for many more samples to be drawn per instance of strawberry.

The classifier was shown to distinguish well between strawberries and background, and can be improved further by including more data. Expanding the dataset, in particular by adding data collected during the growth season is important for expanding the model to more fine-grained classification between different types of berries (i.e. ripe, non-ripe, fungal infections etc.).

Chapter 5

Real-time object detection with deep neural networks

5.1 Introduction

In contrast to the naive algorithm developed in 4, modern deep learning algorithms leverage the flexibility of neural networks to a much greater extent. In this chapter, a selection of modern neural network object detection algorithms are reviewed. These algorithms have demonstrated competitive performance on benchmark datasets, and show the progress of the field towards high-performance deep neural nets which can operate in real-time. Finally the object detection algorithm You Only Look Once (YOLO) [5], [31] is applied to the problem of strawberry detection using the annotated strawberry dataset.

5.2 Literature

Applying classifiers as sliding windows over images using quick to compute features such as Histogram of Gradients (HOG) for a long time proved a successful and effective way of performing object detection. Since the computational load of sliding windows is proportional to the number of windows to test, the need to search at several scales and/or aspect ratios of objects increases the search space of sliding windows by several orders of magnitude [32]. Increasingly complex classifiers such as deep neural network perform better at evaluating the content of images, but make a sliding window approach infeasible for real-time applications. This section outlines some of the object detection algorithms which have achieved both high precision and real-time evaluation speeds using deep neural nets.

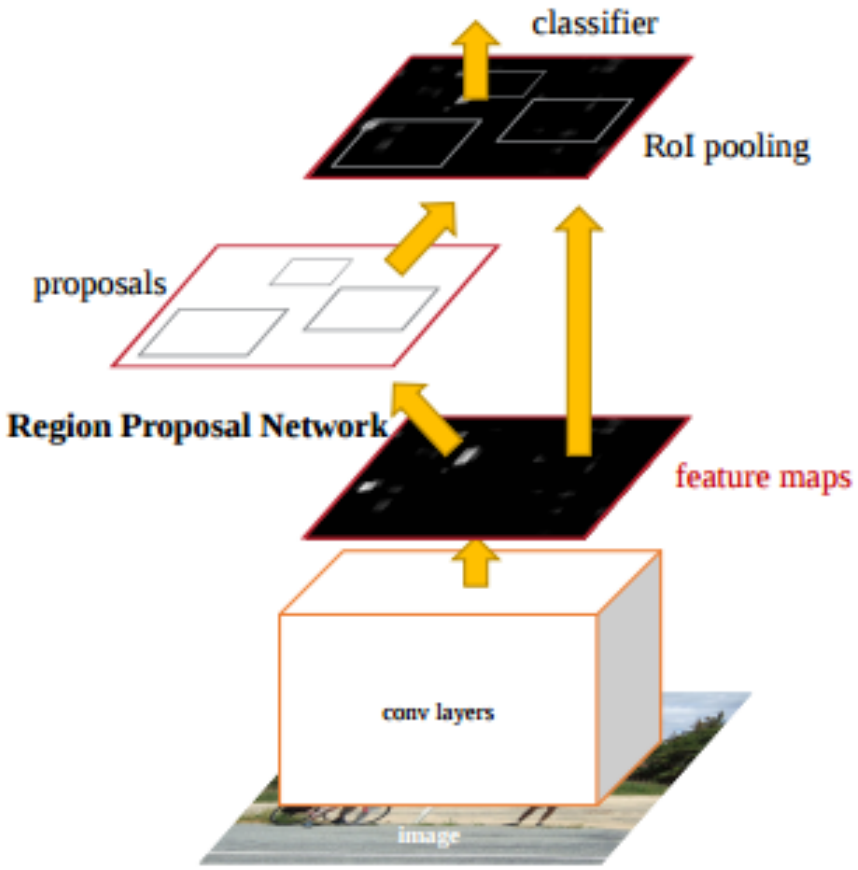


FIGURE 5.1: Faster R-CNN relies on the deep representations provided by an arbitrary pre-trained convolutional neural network architecture, from which it produces intermediate proposals for regions of interest which are passed to the classification layer. Figure from [3].

5.2.1 Region proposal object detection

One way to overcome the need for sliding windows is by assuming that all objects that are of interest share certain features which can be learned as a general model of objects. Under this assumption, the *objectness* of a region can be evaluated, and subsequently processed further if it has enough resemblance to any object.

The first method to apply deep neural networks in this manner was Girshick et. al. [33], which proposed the method R-CNN. The method is comprised of three sequential modules:

1. Region proposal generation
2. Feature extraction
3. Classification

The region proposals utilise an object detection method called selective search [34] which merges the most similar neighbouring regions in an image into connected regions or blobs. R-CNN produces around 2000 of these proposals, for which it extracts a feature vector using a deep neural network of five convolutional layers and by two fully connected layers. In the classification stage, a set of single-class Support Vector Machines (SVM, a type of classifier) compute class scores for each class. Finally they perform non-maxima suppression to obtain the final object detections.

Although R-CNN increased mean average precision (mAP) on the PASCAL VOC dataset by more than 20%, the processing time for each image was 47s. The authors subsequently published papers on streamlined versions of R-CNN, "Fast R-CNN" [35] and "Faster R-CNN" [3] which improved performance both in terms of processing time and mean average precision (mAP). Fast R-CNN computes a shared feature map for the original image, onto which the region proposals are projected. In this manner, the CNN is run once for each image instead of individually for each of the 2000 regions.

In the "Faster R-CNN" paper, the authors discard the selective search region proposals and introduce a Region Proposal Network (RPN), which are deep neural nets trained specifically to propose object regions. By sharing features from a deep convolutional network to perform both region proposals and classification, they drastically reduce the computational cost of computing region proposals and achieve a frame rate of $5FPS$.

5.2.2 Single Shot Multibox Detection(SSD)

The Single Shot Multibox Detector (SSD) [4] performs object detection in a single deep neural network. SSD uses a pre-trained model for extracting image features ([36], winning submission in ILSVRC'14). On top of this base, they add a series of convolutional layers which produce feature maps at several scales. Each cell in each feature map produce class scores and spatial offsets for 5 default shaped anchor boxes. At the final layer, all of the outputs from the feature maps are evaluated and non-maxima suppression is performed on the bounding boxes. An illustration of the model architecture is provided in figure 5.2.

SSD achieved state of the art mAP on the PASCAL VOC dataset and reports a processing speed of 59 FPS.

5.2.3 You Only Look Once (YOLO)

The object detection algorithm called "You Only Look Once" (YOLO)[5] was the first detection algorithm based on deep neural nets to achieve real-time performance with a processing speed of 45 FPS. Contrary to the other methods reviewed, the authors frame object detection as a regression problem. Predicting detections consists of a forward pass

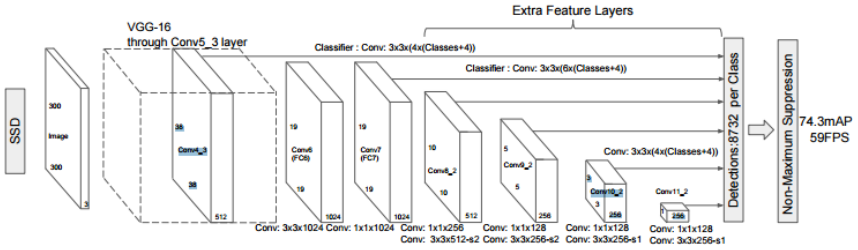


FIGURE 5.2: After extracting features using an arbitrary pre-trained deep convolutional neural network, SSD generates outputs from a series of convolutional layers of decreasing resolution. These convolutions produce predictions for different sized objects, with the coarser resolutions detecting large objects and vice versa. Figure from [4].

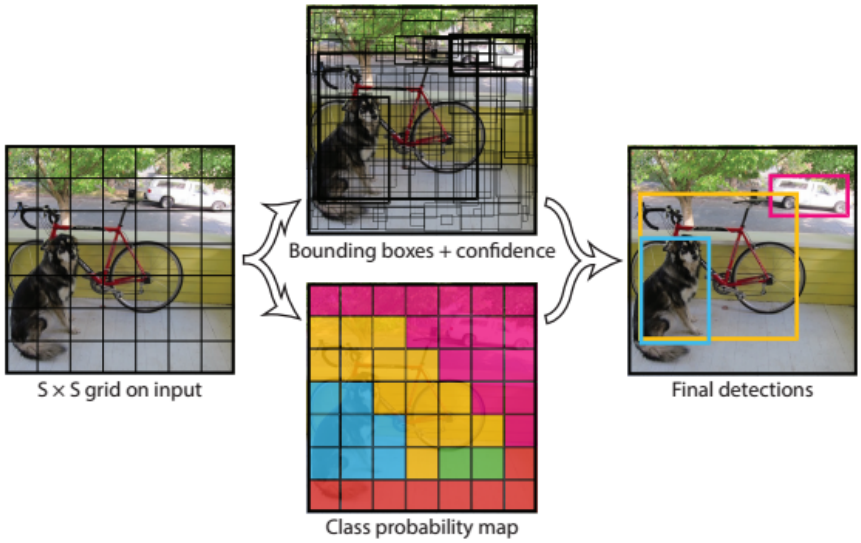


FIGURE 5.3: The figure illustrates how YOLO generates it's outputs on top of the deep convolutional feature extraction architecture. For each of the grid cells over the input image, 5 vectors containing bounding box coordinates and confidence score for the objectness, and a probability distribution over classes. Figure from [5]

of the image through a convolutional neural network. The output of the network (see figure 5.3) is a grid over the image, with each cell output containing predictions of class confidence scores and bounding box coordinates.

In its latest incarnation, the algorithm was modified to be able to predict detections for all 9000 object categories included in the Imagenet dataset while maintaining real-time computation performance. The authors achieved this by developing methods for semi-supervised learning, allowing them to train object detection algorithms on more readily available classification datasets.

YOLOv2 adresses several shortcomings of YOLO and also draws on ideas from both R-CNN and SSD. Batch-normalization is applied at every layer, and the model is modified to a fully convolutional architecture. The output is modified to a 13×13 grid over the image which also receives inputs from from a more fine-grained feature map, which

Algorithm name	mAP (PASCAL VOC dataset)	FPS
SSD300	74.3	46
YOLOv2 416	76.8	67
Faster R-CNN (VGG-16)	73.2	7

TABLE 5.1: Summary of achieved performance on the PASCAL VOC benchmark dataset and frame rates for three different object detection algorithms based on neural networks. The different approaches achieve comparable results, however YOLOv2 reports the highest framerate. [31][3][4].

improves detection of smaller objects. The number of predicted bounding boxes per grid cell in the output layer is increased from two boxes to five boxes with prior settings for size and aspect ratios of objects (similar to the anchor boxes used in SSD). YOLOv2 has a mAP on the PASCAL VOC dataset and processing speed which is comparable to SSD depending on their configurations.

5.2.4 Summary

The three approaches to object detection reviewed here have achieved a high performance on benchmark datasets using slightly different approaches. In the evolution of these methods, there is a tendency structure training data, network architectures and training methods in increasingly clever ways which allow object detection to be achieved with a single unified neural network. To summarize, the methods all rely on deep convolutional networks to extract features from images, but structure the processing of the extracted features in different ways. A summary of the performance metrics of the three methods is included in table 5.1

5.3 Methods

5.3.1 Model architectures

Three model architectures based on the YOLO detection framework were trained on the strawberry detection task. The authors of the YOLO papers provide their pre-trained model parameters and configurations on their project website [37]. Due to the goal of achieving real-time processing of images on an embedded system, the "Tiny-YOLO" model architecture was chosen as a starting point. Tiny-YOLO is a version of YOLOv2 designed to balance detection performance with processing speed.

The Tiny-YOLO model was trained using both pre-trained parameters and random initialization of parameters. This model retains the anchor box settings from the original model, which are based on the most common object shapes and sizes in the original training set. These two models serve to reveal any benefit gained from transfer learning.

TABLE 5.2: Overview of the different YOLO models.

Model	**Number of layers**	**Input dimensions**	**Output dimensions dimensions**
Tiny-YOLO, pretrained	15	$416 \times 416 \times 3$	$13 \times 13 \times 30$
Tiny-YOLO, randomly initialized	15	$416 \times 416 \times 3$	$13 \times 13 \times 30$
Tiny-YOLO, better boxes	15	$416 \times 416 \times 3$	$13 \times 13 \times 30$
Smaller Tiny-YOLO	13	$414 \times 414 \times 3$	$13 \times 13 \times 30$
Tiny Tiny-YOLO	11	$360 \times 360 \times 3$	$20 \times 20 \times 30$

In the remaining models, the five anchor boxes were modified to reflect an expectation of round objects which fit best in square bounding boxes. Since the findings in chapter four suggested that strawberry detection is achievable using relatively shallow models, two model architectures which omit layers are also trained. To maintain sufficient detail in the input images, the input dimension is kept at around 400 pixels for all the models. The models have the same output format, although the smallest model has a higher resolution at the output.

Each convolutional layer is applied with 3×3 pixel filter kernels followed by an activation layer, batch-normalization layer and a max pooling layer. The model uses leaky Rectified Linear Units for it's activations. Inputs to convolution layers are padded with a one pixel border of zeros, which maintains the input dimensions through the convolutional layer. The model has 16 filters at the first convolution, increasing by a factor of 2 for each successive convolutional layer. A template for the different model architectures is shown in figure 5.4, and a table of model summaries is provided in 5.2.

5.3.2 Training data

The annotated training dataset was adapted for training with the YOLO-algorithm in the Darknet framework[38]. Instead of training on positive and negative sampled parts of images, YOLO is trained on large images of any aspect ratio, containing object(s) with a wide variety of sizes and positions using both class labels and ground truth location coordinates. Training images are reshaped to the appropriate size before processing by the models.

The labelled bounding box coordinates was converted to the YOLO annotation format and augmented with $90, 180$ and 270 degree rotations.

After the augmentation, the training set contains 3144 frames from which 600 samples were drawn for the validation set.

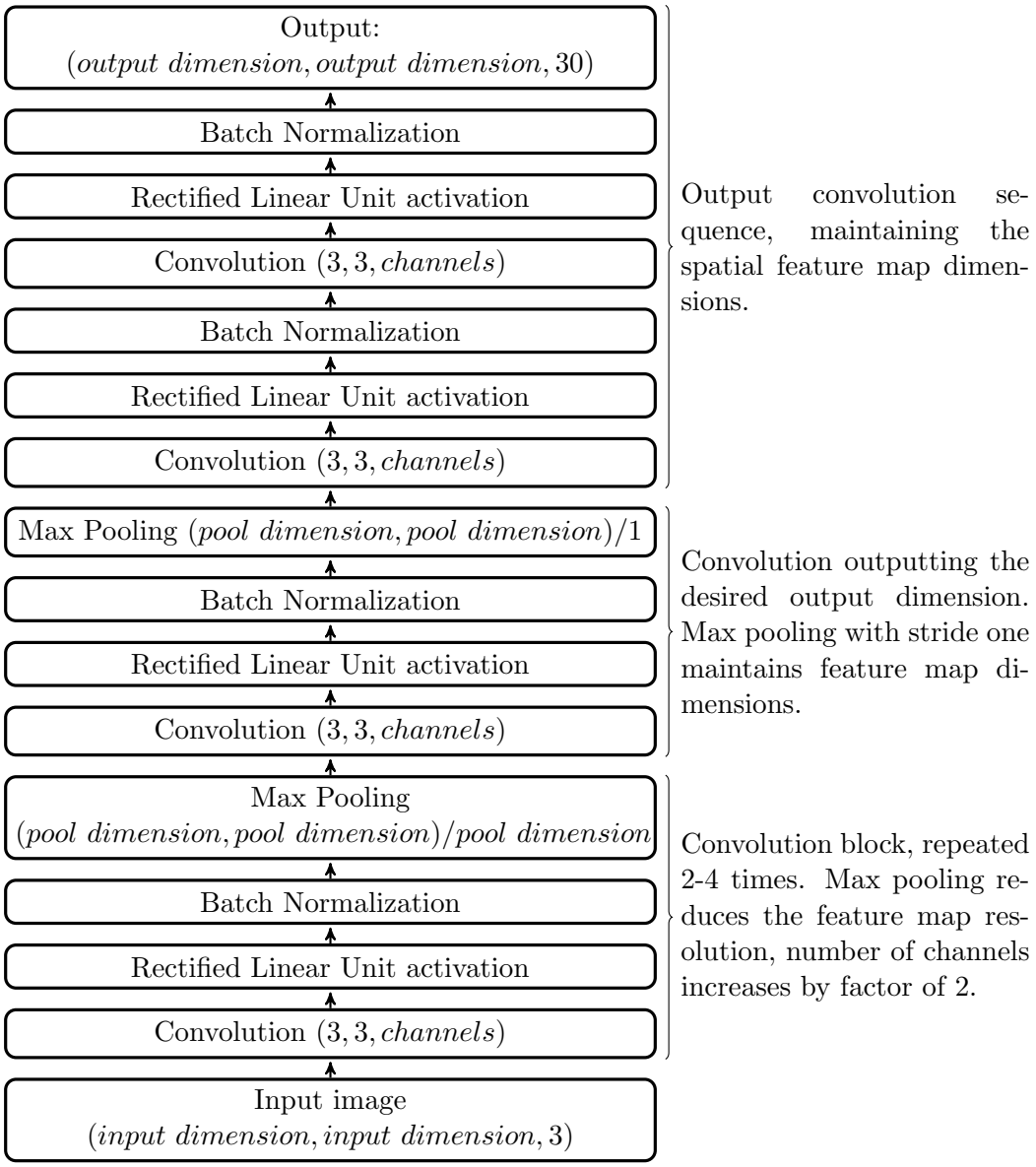


FIGURE 5.4: Template for the YOLO models.. The first block of convolution reduces the resolution of the output feature maps by a factor determined by the pool dimension and also increases the number of filter channels by a factor of two. This block is repeated 2 times for the smallest model and 4 times for the original tiny-YOLO model. When the desired output resolution is achieved, the remaining max pool and convolution layers maintains feature map dimensions.

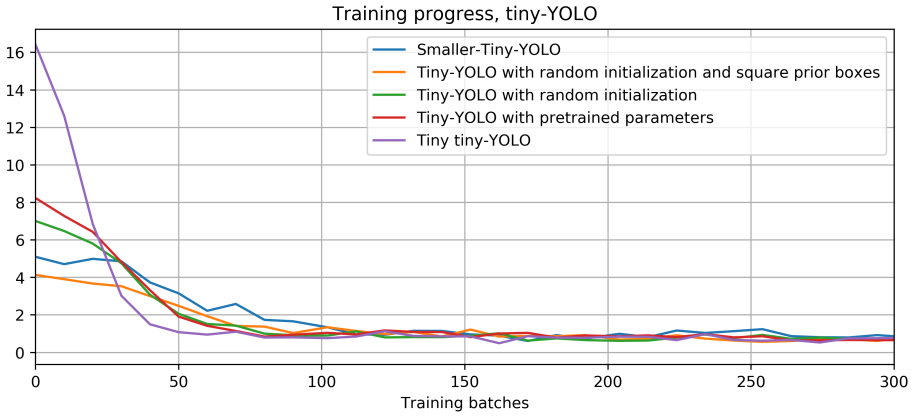


FIGURE 5.5: The training loss curves from the various models plateau after around 150 batch iterations, however generalization to unseen data isn't achieved until a few thousand iterations of training have been completed.

5.3.3 Training settings

The models were trained with a batch size of 64 images, and a learning rate 0.001 with a decay parameter of 0.005 and momentum 0.9.

The training loss is a weighted combination of mean squared errors for classification error and bounding box coordinate errors.

5.3.4 Model selection

For validation and model selection, the model parameters were saved for each 1000_{th} training batch. Model evaluation was performed with regards to average IOU and recall on a test set using the saved parameters for each model.

5.3.5 Software

The models are trained and developed using the Darknet deep learning framework [38] with minor modifications for hardware compatibility and memory management.

5.4 Results

Training of the models was suspended when detection performance plateaued. The validation set performance is plotted for each of the models in figure 5.6. A summary of the highest achieved detection performance for each model is included in 5.3. The Tiny Tiny-YOLO model was found to be the best model among those tested with regard to

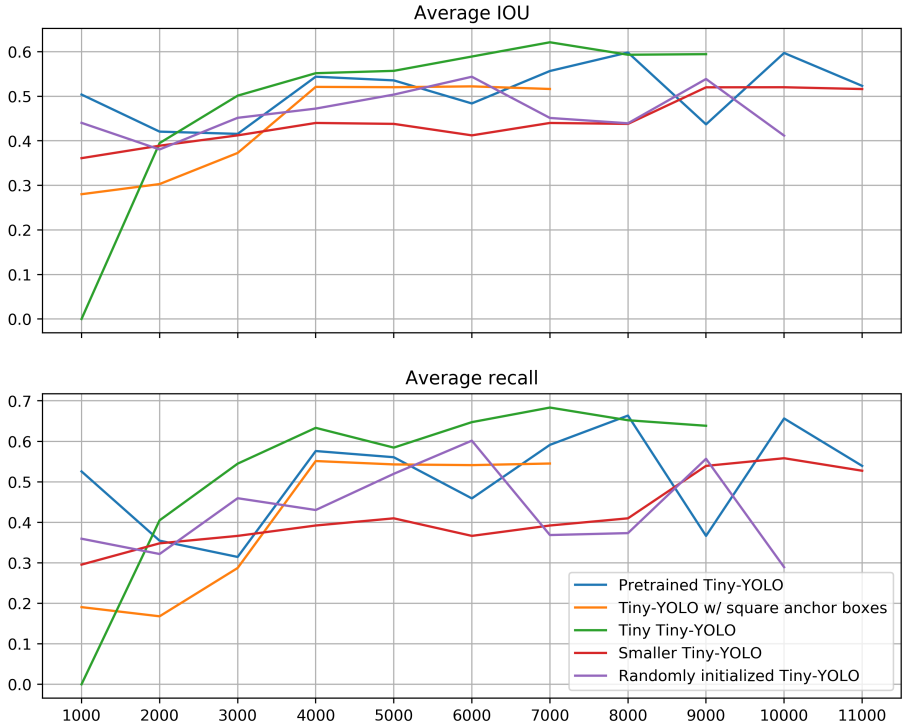


FIGURE 5.6: Validation results for the five models. Models were selected by highest measured IOU.

TABLE 5.3: The validation set detection performance for each model's best set of parameters, selected by highest IOU. Tiny Tiny-YOLO achieves the best result in all of the evaluated categories.

Model	**Average IOU**	**Average recall**	**Test set processing speed**
Tiny-YOLO, pretrained	0.598	0.664	13.6FPS
Tiny-YOLO, randomly initialized	0.544	0.602	13.6FPS
Tiny-YOLO, better boxes	0.522	0.541	13.6FPS
Smaller Tiny-YOLO	0.520	0.558	26.1FPS
Tiny Tiny-YOLO	**0.621**	**0.683**	**30FPS**

IOU. A sample of detections obtained with this model is shown in figure 5.7 and in video format in the appendix.

The results show that the pretrained model performs better than the randomly initialized one after 1000 training iterations, however the performance fluctuates, and no conclusion can be drawn from this experiment.

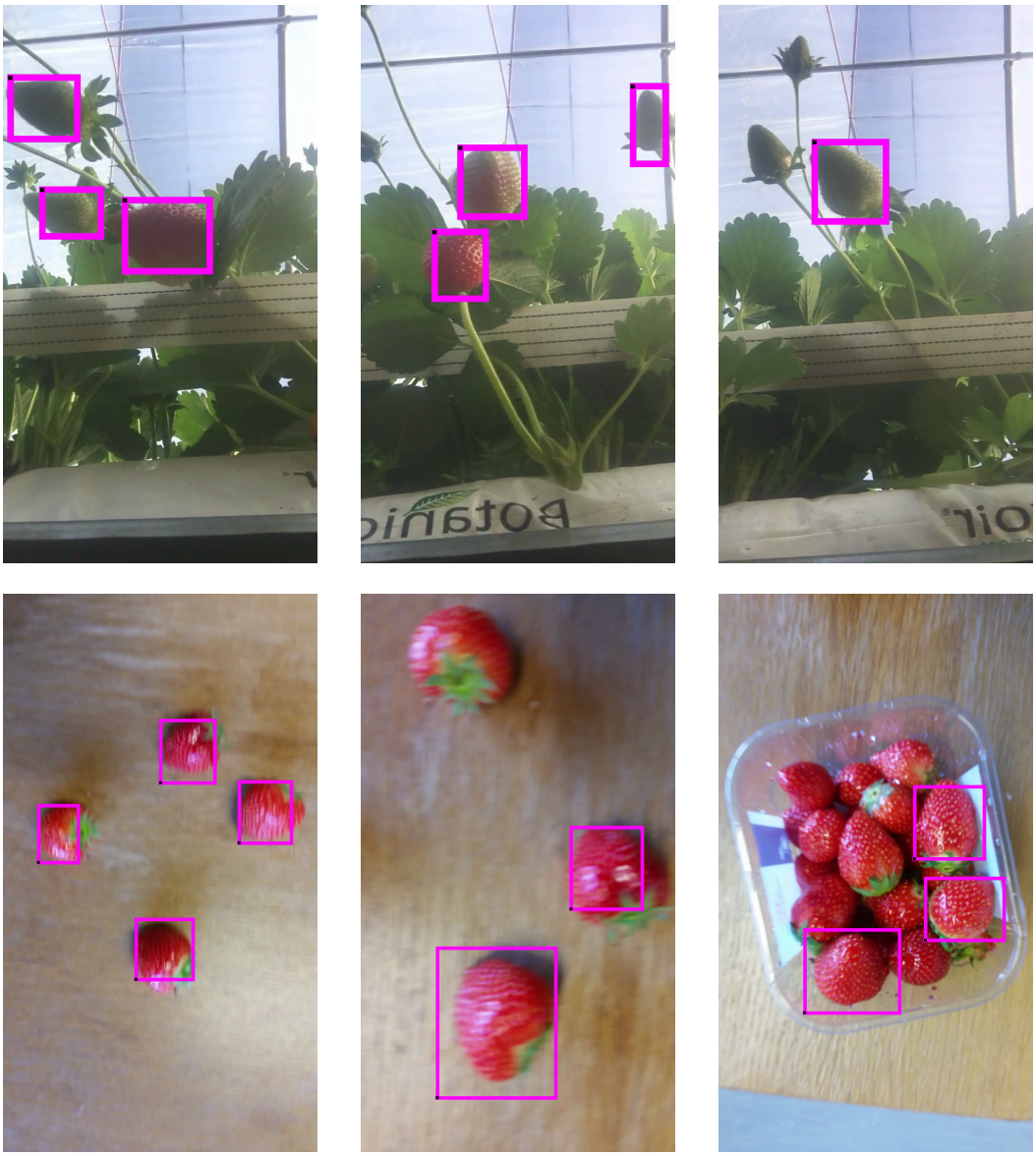


FIGURE 5.7: Sample detection results obtained with the Tiny-tiny YOLO model. The model is both fast and precise. Strawberries dominated by uniformly dark blobs is a typical detection failure along with berries viewed from above, which are partly covered by the green strawberry stem. The model also struggles with clusters of berries.

5.5 Conclusion

Five different models with different initial parameters and architectures were trained for the task of strawberry detection. Testing the models on the validation data demonstrated the capability of deep neural network object detectors to detect strawberries on plants in a growing facility.

No conclusive advantage was found in applying transfer learning to this problem. Shallower model architectures was found to perform on par with deeper models for strawberry

detection. The model with the highest output resolution performed the best measured by IOU.

The fastest model, which also had the best detection performance achieved a processing speed of 25 FPS on the NVidia TX1 embedded platform.

5.5.1 Discussion

Although the shallowest model performed the best on a test set, it is a fair assumption that this is attributable to the higher output resolution of the model, rather than the decreased depth. The model with a 20×20 output grid predicts 400×5 boxes vs. 169×5 for the models with a 13×13 grid. It can however be concluded that the depth of the smallest model is sufficient for the task, and could possibly be decreased further.

Transfer learning provided no discernible benefit. One possible explanation for this is that strawberry detection relies on a narrow set of filter combinations which occupy a relatively tiny fraction of the parameters in the model. The method applied in retraining the parameters is also somewhat crude as it back-propagates large gradients through the network, causing large parameter updates. A better approach would be to iteratively fine tune the model layer by layer, starting with the output and working backwards.

The processing speed of the fastest model demonstrated suitability of these types of detection algorithms for embedded applications in mobile robotics. Further work should explore models which further reduce the number of layers while maintaining a high detection performance.

Chapter 6

Deployment on an embedded system

This chapter discusses some of the practical aspects of implementing a strawberry detection and logging system with a robot such as Thorvald (figure 1.1. Although the thesis does not tackle the problem of position estimation in the world frame, a possible path to achieving this based on the developed detection models and implementing it with a robotic system is presented.

6.1 Object tracking and position estimation

The goal of this thesis is to provide a robot with object detection algorithms which enable location estimation for detected strawberries in the world frame. Measuring the position or pose of an object from 2-dimensional image data is generally accomplished by photogrammetry, requiring images of the object from multiple angles and mapping correspondences between features which are visible in two or more frames. From this information it is possible to compute an estimate of the camera pose using epipolar geometry (figure 6.1). In the strawberry detection setting, both the pose and motion of the robot is available in real-time, and computing an estimate for the position of the strawberry can be done by mapping the corresponding strawberry detections between several succeding frames.

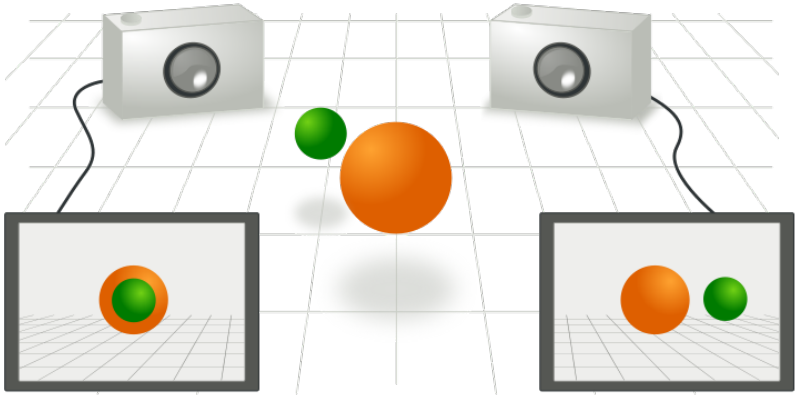


FIGURE 6.1: Epipolar geometry describes the relation between cameras viewing a scene from different viewpoints. Figure courtesy of Arne Nordmann [6].

6.2 System deployment with Robot Operating System

6.2.1 Robot Operating System

The Robot Operating System (ROS) is used to deploy the strawberry detection and localization system. ROS is an ecosystem of open source applications and software libraries which facilitate the operation of complex robotic systems. The main functionality provided by ROS is a communication system for all of the processes that makes the robot work. Each process operates as a node in the system, and can publish to or subscribe to named topic buses which are relevant to the node's operation. The messages exchanged over the topic buses contain information relevant to the operation of the robot, such as sensor readings or location information. Another important feature of ROS is the robot geometry library, which is a way of specifying a robot's physical attributes and relationships between different coordinate systems.

6.2.2 ROS Implementation

The Strawberry detector is implemented in ROS as a node which is run as part of an arbitrary robotic system. The detector node subscribes to images from the video topic which and runs the detection algorithm. Object detections are encoded in a ROS message as a 2-dimensional array containing four coordinates for each detected object and published to a detection topic.

A separate ROS node computes the 3D-position estimates for detected strawberries. This node subscribes to global positions and velocities for the robot as well as the coordinates of detections, and has all the information required to assign position and rotation to a strawberry.

6.3 Hardware

6.3.1 NVidia Jetson TX1

NVidia Jetson TX1 is selected as the hardware platform to run the detection algorithm. The system features 1 teraFLOPs (10^{12} floating points operations per second) computing performance, and is designed for computationally expensive applications such as embedded deep learning. The final detection model, "Tiny Tiny-YOLO", has a processing speed of approximately 25 FPS on this system.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis, an annotated dataset for strawberry detection has been developed. The sparsely populated strawberry plants in the collected video were supplemented with images of strawberry from the ILSVRC competition to provide a wider distribution of strawberry types and states of ripeness. In the final dataset, image frames extracted from the videos and images from ILSVRC were annotated with pixel coordinates and class labels for the state of strawberries.

Two different approaches to object detection with deep neural networks have been explored; a classifier applied as a sliding window object detector and various model architectures performing both classification and bounding box regression using a single convolutional neural network.

The sliding window algorithm was found to be ineffective both in terms of detection performance and computational efficiency. The algorithm achieved an average intersection over union detection performance of 17.3%. Analysing the classifiers' responses to different input images at various points in their computational graph yielded some insights about the complexity of the strawberry classification and detection task.

The "You Only Look Once"-framework for object detection unified in a single neural network was applied to the strawberry detection task. Five different models with varying architectures and initial parameters were trained on the task. The shallower models performed on par with the deepest models, showing that the task of strawberry detection can be achieved using relatively shallow neural networks capable of running in real-time on an embedded computation platform. The best performing model generalizes well to unseen data and produces fast and accurate detection of strawberries.

Finally a description of some of the practical aspects of deploying the detection models as part of a mobile robotic system is given. A ROS implementation and a framework for

estimation of 3D-position of detected strawberries using the developed detection models is discussed.

7.1.1 Discussion and further work

Primarily, further work should focus on gathering more data, preferably also 3-dimensional data. Such data can support the development of more precise and robust algorithms for position estimation than the one outlined in this thesis, and can also serve to expand the dataset with a larger variety of strawberry types. Such data is not easy to obtain or develop, and devising ways of automated labelling of such data is an interesting research direction in itself.

The labelling scheme in this thesis annotates strawberry locations with coordinates for the box which contains it. An alternative to this is to parametrize the strawberries as two or more vectors which describe a strawberry's physical attributes such as diameter and axial length. Successful estimation of these parameters for a strawberry over a sequence of images could enable estimation of it's rotation in addition to the location.

To improve the position estimates of detected strawberries, the algorithm should be modified to track objects between frames rather than just detect coordinates for individual frames. In taking this approach, the algorithm is given memory of the previous position(s) of detected objects, and can utilize this prior knowledge to form better predictions. This can be achieved using traditional approaches such as applying a Kalman filter to the series of bounding box coordinates, alternatively Recurrent Neural Networks could be applied in an end-to-end neural network approach.

One shortcoming of this thesis is that the approaches taken fail to fully utilise the potential of deep learning algorithms. The task of drawing boxes around objects is quite far removed from the way humans interpret spatial structure, and narrowing this gap significantly is achievable using neural networks. Several results in the deep learning domain have demonstrated an impressive capability of convolutional neural networks to interpret 3-dimensional spatial structure from images, and future work should explore methods for end-to-end position estimation from 2-dimensional image data similar to that used in this thesis.

Appendix A

Software

One SD-card containing

1. Chapter 3: Labelled strawberry dataset
2. Chapter 4: Sliding window detection software
3. Chapter 5: Deep neural networks for strawberry detection, detection models

Bibliography

[1] R A Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7:179–188, 1936. also in.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL `http://arxiv.org/abs/1506.01497`.

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. URL `http://arxiv.org/abs/1512.02325`.

[5] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL `http://arxiv.org/abs/1506.02640`.

[6] Arne Nordmann.

[7] Norges Bondelag. `http://www.bondelaget.no/nyhetsarkiv/en-av-tre-norske-jordbar-odelagt-i-akeren-article84494-3805.html`, 2016. Accessed: 12.01.2017.

[8] Pl From, Volkan Isler, and James Luby. Project description, real-time robotic sensing and manipulation for fruit picking. `http://cbs.umn.edu/norwegian-centennial-chair/currently-supported-activities`, 2016. Accessed: 25.01.2017.

[9] Wilhelm Burger and Mark James Burge. *Digital Image Processing*. Springer, 2008.

[10] Knut Kvaal. Lecture: Digital images. , 2016.

[11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[13] MIT technology review. `https://www.technologyreview.com/s/513696/deep-learning/`, 2017. Accessed: 12.01.2017.

[14] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[15] Andrew Ng. `https://hbr.org/2016/11/what-artificial-intelligence-can-and-cant-do-right-no` 2016. Accessed: 15.02.2017.

[16] *Neural Networks and Deep Learning.* Determination Press, 2015. `http://neuralnetworksanddeeplearning.com/`.

[17] Eric Roberts. `https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html`, 2000. Accessed: 13.03.2017.

[18] Andrej Karpathy. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015. Accessed: 13.03.2017.

[19] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL `http://arxiv.org/abs/1604.07316`.

[20] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[21] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[22] Yann LeCun, L eon Bottou, Yoshua Bengio, et al. Gradient-based learning applied to document recognition. *PROC. OF THE IEEE*, page 1, 1998.

[23] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[25] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962. Citeseer, 2003.

[26] Geoff Hinton. Overview of mini-batch gradient descent. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[27] Eric W. Weisstein. "maximum likelihood." from mathworld–a wolfram web resource. `http://mathworld.wolfram.com/MaximumLikelihood.html`, 2017. Accessed: 24.03.2017.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[29] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. URL `http://arxiv.org/abs/1411.1792`.

[30] Karlsruhe Institute of Technology. Sloth. `https://github.com/cvhciKIT/sloth`, 2016.

[31] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. URL `http://arxiv.org/abs/1612.08242`.

[32] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals?

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

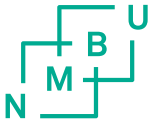
[34] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[35] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL `http://arxiv.org/abs/1504.08083`.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

[37] Joseph Redmon. Darknet deep learning framework. `https://pjreddie.com/darknet/`, 2017. Accessed: 12.05.2017.

[38] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.



Norges miljø- og biovitenskapelig universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway