AUTOMATED COUNTING OF WHEAT GRAINS IN IMAGES

Osagioduwa Edo-Osagie

A Dissertation submitted to
the School of Computing Sciences of The University of East Anglia
in partial fulfilment of the requirements for the degree of
MASTER OF SCIENCE.
AUGUST 2016

SUPERVISOR(S), MARKERS/CHECKER AND ORGANISER

The undersigned hereby certify that the markers have independently marked the dissertation entitled "**Automated Counting of Wheat Grains in Images**" by **Osagioduwa Edo-Osagie**, and the external examiner has checked the marking, in accordance with the marking criteria and the requirements for the degree of **Master of Science**.

Supervisor: _____
Dr. Wenjia Wang

Markers: _____
Marker 1: Dr. Wenjia Wang

_____
Marker 2: Dr. Ji Zhou

External Examiner: _____
Checker/Moderator

Moderator: _____
Dr. Wenjia Wang

DISSERTATION INFORMATION AND STATEMENT

Dissertation Submission Date: **August 2016**

Student:	**Osagioduwa Edo-Osagie**
Title:	**Automated Counting of Wheat Grains in Images**
School:	**Computing Sciences**
Course:	**Advanced Computing Science**
Degree:	**M.Sc.**
Year:	**2016**
Organiser:	**Dr. Wenjia Wang**

STATEMENT:

Unless otherwise noted or referenced in the text, the work described in this dissertation is, to the best of my knowledge and belief, my own work. It has not been submitted, either in whole or in part for any degree at this or any other academic or professional institution.

Permission is herewith granted to The University of East Anglia to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Student

Abstract

Counting the number of grains in a group of wheat plants is a good way to get an idea of the development progress of the plants. This dissertation aims to research and create a system to automatically count the number of grains in periodically captured images of wheat plants. The resulting system could be applied and extended to automatically monitor the growth/yield of wheat plants and raise alerts if anomalies occur, saving farms and research institutions a lot of time and resources which can then be channeled to other areas. In this project, image processing techniques will be applied to images to highlight and segment grain regions. The number of grains in the segmented regions will then be counted.

This dissertation presents and compares two methods for counting wheat grains in images. The first method takes a "counting by regression" approach and attempts to compute a grain density function F that provides a mapping between the densities of grains and the number of grains in the images. The second method takes a "counting by detection" approach and aims to detect all instances of grains in a given image, then compute the count as the number of instances detected.

The regression approach was found to have an R^2 score of 0.83. The detection approach had an accuracy of 80.68%. When compared to each other, the detection approach was found to perform better. However, this may be due to the fact that only 12 images were available for training the regression model in the regression approach so it is hard to tell whether or not the regression approach is truly inferior or not.

Acknowledgements

I thank God for enabling me to undertake this task. I would like to thank my mother and her siblings for their boundless support.
I would also like to thank my supervisor Dr Wenjia Wang for his advice and also Dr Ji Zhou for his guidance and support.

Finally, I am grateful to my friends for their faith in me.

Osagioduwa Edo-Osagie

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This project aims to research and create a system to automatically count the number of grains in periodically captured images of wheat plants. Agriculture is arguably one of the most important disciplines in the world. It is important that food is grown on a large scale. Large scale farmers often have multiple sites with multiple groups of plants growing. This makes it difficult for them to simultaneously monitor and coordinate all of these plants. Farmers need good tools to improve their agricultural processes, in turn improving yield. Computer science and technology has been used to grow many other disciplines but only applied to agriculture to a smaller extent. This project aims to apply computer vision techniques to automatically monitor growth rates of wheat plants. Farmers could potentially be warned when irregularities occur. In addition to this, the results of the project could also be applied in a research setting to help biological researchers better monitor their experiments.

The project involves the design and implementation of a system to automate the monitoring of wheat growth with periodically captured images. The system will

function as follows: Images are periodically captured automatically through an appropriate Raspberry Pi-powered setup (or an alternative setup). The images are then uploaded and stored in a database. The system can then analyze the images in the database, estimating the number of grains in each image and draw conclusions from its analyses.

1.1 Motivation

The problem is that it is not possible to automatically count the number of grains in an image of a (wheat) plant. At the moment, it is only either a fully or partially supervised process, requiring human intervention to a significant degree. There is no way to simply create a picture of the plants' progress without any human intervention.

While it may not seem like a big deal, the problem could yield great reward if solved. Also, not only does the field of agriculture have something to gain from the success of the project but also Biology, Botany, Food Sciences and basically anything to do with plant research. The resulting system *could* be applied and extended to automatically monitor the growth/yield of (wheat) plants and raise alerts if anomalies occur. The system would function with minimal supervision. If solved, the resulting system could save farms and research institutions a lot of time and resources which can then be channeled to other pressing areas.

1.2 Aims and Objectives

The aim of this project is to develop/improve machine learning based algorithms to analyze wheat images and give a count of the number of grains in the image. Its objectives are as follows:

- Apply image processing techniques to highlight and segment grain regions.
- Count the number of grains in each of the segmented regions.

Chapter 2

Literature Review and Related Work

2.1 The Counting Problem

A prevalent theme in the field of Computer Science is that of performing tasks traditional to humans, through the use of a computer. Counting is a trivial cognitive task for humans. In fact, it comes so naturally that studies have shown that children below the age of 3 demonstrate an understanding of numerosity without any knowledge of number or counting systems (Sella et al., 2015). For a computer, the task of estimating the number of objects (of some kind) in an image is slightly more complicated and comes up in many real world applications such as counting cells in microscopic images and monitoring crowds in surveillance systems.

The most established ways of counting objects in images involve the use of image processing or computer vision techniques in conjunction with machine learning methods. While there are many different approaches which employ different techniques to achieve these goals, they all share (to some extent) the same general outline. The

image is first transformed to greyscale. The greyscale image is thresholded to yield a binary image. The binary image is segmented and the features to be used for counting are extracted and passed into the model. Figure 2.1 shows a generalized flowchart for the activities involved in counting objects in images with the stages described as follows:

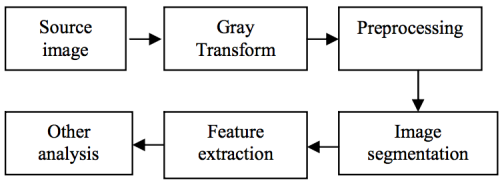


Figure 2.1: Generalized pipeline for approaching the counting problem

(i) Gray Transform: The image is first transformed to greyscale. This is because counting usually depends on morphological (ie. shape), spatial and textural features that have no use for colour. The exclusion of colour simplifies subsequent calculations and makes the whole process more computationally efficient. Also, a lot of the techniques used in processes further down the pipeline are designed to work on simple two-dimensional matrices and would be very difficult, if not impossible, to extend to colour images. There are various ways to achieve this conversion. One way is to simply average the RGB pixel values to give the greyscale pixel value (Cook, 2009). A more accurate method of conversion is the use of a luminance-preserving mapping. This method gives the greyscale pixel value, given as the *luminosity* of the pixel, to be calculated as a weighted

sum of the RGB pixel values (Anderson et al., 1996).

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{2.1.1}$$

Applying the formulae described above to each pixel in a colour image will result in a grayscale version of the image.

(iii) Preprocessing: The presence of noise in the image will affect the counting accuracy. Because of this, certain transformations must be carried out on the image before going further. Also, depending on the objects to be counted in an image, some processes can be performed on the image in order to highlight or de-emphasize appropriate elements of the image to make detection and counting easier and more reliable. Image smoothing and sharpening are usually used to achieve this. Image smoothing can be carried out by applying a *Gaussian filter* (Kopparapu and Satish, 2014) or a *median filter* (Church et al., 2008). However, in the process of removing noise from the image, image smoothing might also blur some of the edges in the image. Blurred edges are not conducive to segmentation, analysis and other potential follow-up processes. Hence, image sharpening is performed on the image in order to emphasize the edges in the image. Image sharpening can be carried out by applying a *high-pass filter* (Makandar and Halalli, 2015).

(iii) Binary Image Segmentation: Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such

that pixels with the same label share certain characteristics (Wikipedia, 2016). With binary image segmentation, which is what is usually used in object detection and counting, there is only one label - "edge". Pixels that form part of an edge are assigned a value of 1 and any other pixel is assigned a value of 0. Given this, image segmentation in this sense can be performed by applying an edge detection algorithm to the image. A popular edge detection algorithm used for image segmentation is the *Canny edge detection* algorithm (Canny, 1986). In simple terms, the Canny edge detection algorithm finds edges where the grayscale intensity of the image changes. These areas and changes can be approximated by calculating the gradient G in the image in the x and y directions.

$$
\begin{aligned}
|G| &= \sqrt{G_x^2 + G_y^2} \\
|G| &= |G_x| + |G_y| \\
\theta &= arctan(\frac{|G_x|}{|G_y|})
\end{aligned}
\tag{2}
$$

(iv) Feature Extraction: Feature extraction plays a very important role in the area of image processing with regards to machine learning. Feature extraction is a type of dimensionality reduction that efficiently represents interesting parts of an image as a compact feature vector (Mathworks, 2014). These extracted vectors can then be used as inputs and elements of machine learning algorithms and techniques. The approach of using the extracted feature vectors instead of the entire image proves advantageous when image sizes are large. Using a reduced representation of the image makes it possible to perform many common and important tasks quickly. Common feature extraction techniques include

Histogram of Oriented Gradients (HOG) (Dalal and Triggs, 2005), Speeded Up Robust Features (SURF) (Bay et al., 2006), color histograms (Swain and Ballard, 1992) and Scale-InvariantFeature Transform (SIFT) (Lowe, 1999).

(v) Analysis: In this stage, the extracted feature vector representation of the image has statistical models or machine learning algorithms applied to it in order to recognize, detect or count objects within the given image. The objective of the analysis stage is to learn some useful information from the image, and most times, all of the preceding stages are carried out solely in preparation for this stage.

2.2 Existing Approaches to Counting

A lot of research has been carried out on counting objects in images. However, the objects being considered for counting differ from project to project. Because of this, there are a lot of different ideas and approaches to solving this problem. While they all seem to employ machine learning, some of the ideas take an unsupervised learning approach while others take a supervised learning approach. Supervised learning involves training a model with labeled instances of the subjects being investigated. Examples of these methods are machine learning classifiers. In unsupervised learning, the system is not introduced to any labeled training instances but learns on the job. An example of this is *clustering*.

A few approaches to solving the counting problem employ unsupervised learning

by clustering or grouping similar aspects of the image. (Ahuja and Todorovic, 2007) propose a method for grouping based on self-similarities and (Rabaud and Belongie, 2006) propose a method for grouping based on motion similarities. However, the counting accuracy of such fully unsupervised methods is limited, and therefore others considered approaches based on supervised learning. These approaches fall into the following categories:

2.2.1 Counting by Detection

These approaches solve the problem by first detecting the objects in question and then counting them. Object detection is carried out by the use of a visual object detector that localizes individual instances of the object in the image. Given the localizations of the objects' instances, counting becomes trivial. However, visual object detection is far from being solved and is even more tricky when overlapping instances are involved. Several methods assume that objects are uniform and disconnected from each other by the distinct background color. Following this assumption, it is possible to localize individual instances via the Monte-Carlo process (Descombes et al., 2009) or morphological analysis (Anoraganingrum, 1999). These methods yield reliable results when the assumption is met but are not so applicable in challenging real life situations.

2.2.2 Counting by Regression

These approaches sidestep the hard detection problem altogether and instead, attempt to form a direct mapping from some features or groups of features of the image

to the number of objects contained within it. This is a standard regression problem that can be addressed by many different machine learning tools such as neural networks. This approach however has to discard any available information about the location of the objects, using only its 1-dimensional statistics (total number) for learning. As a result, a large number of training images with the supplied counts needs to be provided during training (Lempitsky and Zisserman, 2010).

2.3 Texture Analysis for Feature Extraction

Image texture analysis is a process which applies certain techniques to extract texture parameters from an image in order to quantify or qualify its perceived texture. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image (Singh et al., 2005). Generally speaking, textures are complex visual patterns that have characteristics such as brightness, colour and contrast. Because of this, texture is easily perceived by humans and is believed to be a rich source of visual information. Thus, texture is can be used as a metric for similarity between images.

The proposed approach makes use of ***Gray Level Co-occurrence Matrix (GLCM)*** texture analysis. A co-occurrence matrix can be defined as the distribution of co-occurring values at a given offset. It reflects the luminance distribution (as well as position distribution) between the same pixels or pixels with similar luminance values. The Gray Level Co-occurrence Matrix (GLCM) is formed from a gray-scale image and calculates how often a pixel with gray-scale intensity i occurs (horizontally, vertically

or diagonally) adjacent to pixels with gray-scale intensity j. Simply put, a GLCM characterizes an image by calculating how often pairs of pixels with specific values (and in a specified spatial relationship) occur in an image, where the spatial relationship could be horizontally adjacent, vertically adjacent or diagonally adjacent. The spatial relationship between pixels can be denoted using direction θ and distance d. For example, using this notation, the spatial relationship between two pixels i and j that are 3 pixels apart diagonally could be denoted as $(d, \theta) = (3, 45°)$.

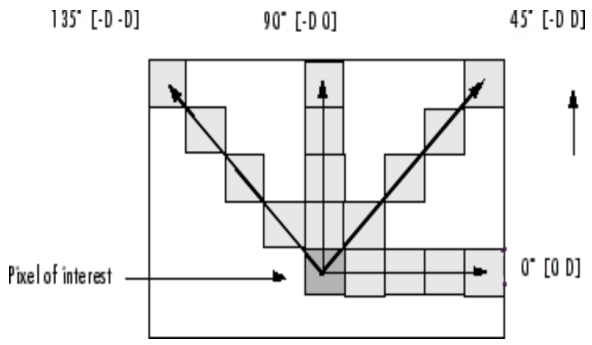


Figure 2.2: Illustration of GLCM spatial analysis (Hayes, 2007)

Below are the textural descriptors that can be derived from a GLCM p with size M by N:

- *Energy*

$$ASM = \sum_{i=1}^{M} \sum_{j=1}^{N} p(i, j|d, \theta)^2$$

is computed as the sum of squared elements in the GLCM. It is also known as *uniformity*, *uniformity of energy*, and *angular second moment* and it represents

texture coarseness.

- *Contrast*

$$CON = \sum_{i=1}^{M} \sum_{j=1}^{N} |i-j|^2 p(i,j|d,\theta)^2$$

returns a measure of the intensity contrast between a pixel and its neighbor over the whole image. It is also known as *variance* and *inertia*.

- *Homogeneity*

$$HOM = \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j|d,\theta)}{1+|i-j|}$$

represents the uniformity of the image and measures the change of local image texture.

- *Correlation*

$$CORR = \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{(i-\mu_i)(j-\mu_j)p(i,j|d,\theta)}{\sigma_i \sigma_j}$$

Returns a measure of how correlated a pixel is to its neighbour over the whole image.

2.4 Learning and Modeling

In order to solve the counting problem, some way of actually determining the number of objects present in a preprocessed representation of an image has to be established. This is usually done by building a model to be used to perform this determination using machine learning techniques. As explained above, supervised learning models yield better results and perform better in practice. A supervised learning model is built from previously encountered and labeled instances and is then applied to subsequently encountered instances.

One possible tool for building such models is the *support vector machine* (SVM). The Support vector machine is a machine learning tool that is usually used for classification tasks but can be extended to regression tasks as well. An SVM model is a representation of the training examples as points in space, mapped so that the examples of the separate classes are separated by a hyperplane (or set of hyperplanes in high dimension problems) while maximizing the distance from the hyperplanes to the nearest training point. Given a set D of n training points,

$$D = \{(x_1, x_2) \dots (x_n, y_n)\} \tag{2.4.1}$$

with a hyperplane described by the equation

$$\langle w, x \rangle + b = 0 \tag{2.4.2}$$

To minimize the margin of the hyperplane, the idea is to solve the optimization problem

$$\text{minimize } ||w|| \text{ such that } y_i(\langle w, x \rangle - b) \geq 1, i = 1 \dots n \tag{2.4.3}$$

For non-linearly separable scenarios, a loss function may be applied to the SVM. Usually the hinge-loss function is used for this. SVMs can be extended to regression problems by the introduction of an alternative loss function (Smola et al., 1996). The alternative loss function must be modified to include a distance measure. Some possible loss functions are the quadratic function, laplacian function and Huber function.

Another machine learning tool that can be used for modeling is the *artificial neural network*. Artificial neural networks are a family of machine learning models inspired by the way that biological nervous systems, such as the brain, process information. A neuron is a cell that has several inputs that can be activated by some outside process. Depending on the amount of activation, the neuron produces its own activity and sends this along its outputs (Stergiou and Siganos, 1996). In addition, specific input or output paths may be "strengthened" or weighted higher than other paths. The artificial neural network equivalent of a neuron is a ***node***. A node receives a set of (weighted) inputs, applies its activation function ϕ to their sum, and passes the result of the activation function to nodes further down the network. The activation function can thus be represented as such

$$\phi = \sum_{i=1}^{n} w_i.a_i \tag{2.4.4}$$

Many such nodes are chained together to form a network, passing the outputs of their activation functions along. The network is trained with labeled data by feeding inputs into the network and fine-tuning the weights until the network always yields the expected class label as its output. This is not very different from regression in that parameters are being tuned to create a function that yields certain values. Hence,

artificial neural networks can easily be adapted to solve regression problems.

2.5 Deep Learning

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations (Bengio and Courville, 2016). It is also known as deep structured learning, hierarchical learning or deep machine learning. The performance of traditional machine learning algorithms depends heavily on the representation of the data used. Features of the data in question are usually extracted and combined in some way before being used in the machine learning algorithm. However, for most tasks, it is difficult to know which features should be extracted. Deep learning offers a solution to this problem by replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction (Song and Lee, 2013). It allows the computer to learn the best features for itself. These learned representations often result in much better performance than can be obtained with hand-designed representations as those are often just a heuristic. They also allow Artificial Intelligence (AI) systems to rapidly adapt to new tasks, with little intervention from the programmer.

Some examples of deep learning architectures are *Convolutional Neural Networks (CNNs)*, *Recurrent Neural Networks (RNNs)* and *deep belief networks*. A CNN is a

type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex (Lab,). A CNN is comprised of one or more convolutional layers and then followed by one or more fully connected layers as in a standard multilayer neural network. CNNs are usually used in computer vision and image processing as their architecture is designed to take advantage of the 2D structure of their inputs. This is achieved with local connections and tied weights followed by a pooling layer which results in translation-invariant features.

The convolutional layer is the heart of a CNN. This layer is made up of a set of kernels (or convolution matrices). During the forward pass, each kernel is convolved across the width and height of the input - usually an image - computing the dot product between the entries of the kernel and the input and producing an activation map of that kernel. As a result, the network learns kernels that activate when they encounter some specific type of feature at some spatial position in the input. The pooling layer is another important part of the convolutional neural network architecture. The pooling layer applies a *pooling function* to the data in order to down-sample it and reduce the size of the problem. *Max pooling* is the most commonly used pooling function and it partitions the input image into a set of non-overlapping rectangles, then, for each such sub-region, outputs the maximum. The purpose of the pooling layer is to progressively reduce the spatial size of the input representation, thereby reducing the amount of parameters and computation in the network. It also controls overfitting. Pooling layers are usually placed periodically between successive convolutional layers. The pooling layer and its operations allow CNNs to be robust to

translations of the input image.

Deep learning architectures have been applied to fields like computer vision, speech recognition, natural language processing, audio recognition and bioinformatics where they have been found to produce state-of-the-art results in various tasks.

Chapter 3

Analysis

In this chapter, a critical look is taken at the reviewed literature. Literature concerning the construction of learning models using machine learning was reviewed in the previous chapter, as well as literature concerning different approaches to solving the counting problem. The reviewed methods, techniques and approaches are compared in an attempt to understand how they can be applied to solve our specific problem of counting wheat grains in an image. In addition to this, a closer look is taken at the problem at hand as the data used in the project is described.

3.1 Analysis of Literature Review

3.1.1 Learning and Modeling

In the literature reviewed, support vector machines (SVMs) and artificial neural networks were predominantly used for learning and modeling tasks when machine learning approaches were taken to solving the counting problem. In particular, SVMs were usually applied to regression tasks, using a slight modification to the traditional SVM algorithm known as *support vector regression*. Neural networks on the other hand,

were always applied to classification tasks. Occasionally, ensemble learning was used to improve the results of the classification tasks.

In some of the more recent literature, it was found that a deep learning approach to learning was taken. In particular, a variation of the artificial neural network known as the *Convolutional Neural Network* (CNN) was applied in most of these works. CNNs differ from traditional neural networks in that the layout of their neurons and layers is inspired by the way the visual cortex works in animals. Because of this, CNNs are suitable for image and video recognition tasks. However, for CNNs to be truly useful, they require several layers which translates to a high computational cost as well as higher technical difficulty in implementation. They also require a high number of images for training. While traditional neural networks (eg. Multi-Layer Perceptron neural networks) also require medium to large size datasets for training, they do not necessarily need to be structured to have many hidden layers. This is a fundamental requirement of deep learning which in some cases, would be more trouble than its worth.

3.1.2 Counting

From the literature reviewed, it can be seen that most approaches to solving the counting problem can be divided into two groups, namely - "*counting-by-detection*" and "*counting-by-regression*". Taking a detection approach is more challenging than taking a regression approach as regression approaches sidestep the hard detection problem altogether. However, in addition to yielding a count, detection approaches

also offer the bonus of identifying and localizing instances of the subject of the counting problem in the image. Also, the accuracy of regression approaches is limited. This is because these approaches simply aim to estimate some function which can provide a direct mapping from the image to a count, or more truthfully, an estimation of the actual count. It is simply impossible for one function to provide an exact mapping for all possible images. Detection methods on the other hand, could theoretically provide *exact* counts from all kinds of images. Of course, this relies completely on such a system being able to accurately detect instances of the query subject in images. Difficult to achieve as it might be, it is possible; even if not perfectly, at least to a very high degree.

Regression is a supervised learning problem. This means that regression methods will require labeled training data. Also, because regression approaches discard all information concerning the location, nature and features of the objects, using only its 1-dimensional statistics - object count - for learning, a large number of training images will need to be supplied. Keep in mind that counts will need to be supplied with each training image to serve as a label during training. While this may not usually be an issue, in certain scenarios, it could be problematic. This project demonstrates one of such scenarios. The images in question for this project contain wheat bushes with thousands of grains and a grain count will need to be supplied with each image. The labeling process cannot be automated because the same automated counting is the problem in question. This means that the grains in each of the numerous images will need to be counted by eye which is a very time-consuming and labour-intensive task.

Detection methods usually assume a uniformity in the problem space and depend on this assumption. In real-life applications, this may not always be the case. Object detection can be difficult depending on the scenario and detection problem in question. Overlapping instances occluding each other will make detecting and localizing individual instances tricky. Because detection approaches rely completely on the accurate detection of instances, the accuracy of these approaches in such scenarios and problem spaces is typically not very high. While counting-by-regression methods cannot really be completely accurate, they are simpler to implement and are less sensitive to non-uniformities in images. Counting-by-detection methods are more powerful and more accurate than counting-by-regression methods but are more difficult to get right. However, when gotten right, and in the right scenario, they are the better choice for an approach to solving counting problems. However, for this dissertation, two solutions to the problem were developed, one using a regression approach, and the other using a detection approach.

3.2 Data Description

We were provided with 12 high resolution images of a wheat growing plot. The images showed the wheat plants at different stages in their development. The orientation and colour of the wheat stalks differed in some of the images. The first issue here is that 12 images provides very little data for supervised learning. This makes things difficult for counting-by-regression approaches. The small size of the dataset made it possible to manually count the grains in each image, essentially providing labels for

Figure 3.1: Examples of images from dataset

any supervised or semi-supervised approaches that are used with said images as well as providing targets for evaluation. In the following chapter, we will discuss how the matter of the size of the dataset is dealt with using our proposed approaches.

Chapter 4

Proposed Approach 1: Using a Counting-by-Regression Method

In this dissertation, two methods for counting wheat grains in images are proposed. This chapter presents the first method which takes a counting-by-regression approach. The high-level idea of this method is extremely simple: given N training images with their grain counts provided, the goal was to recover a grain density function F as a real function of pixels in the images. The grain density function learns the relationship between the density of spikelets in areas of the image and provides a mapping between said density and the number of grains in the image. New images could then be provided as input to F and an estimate of the number of grains in the provided image returned as output. The grain density function (which serves as the predictive model) is built by constructing a regression model with the textural features computed from the GLCMs of the images in the dataset. The first step in this approach is the textural feature extraction from the images and the next step involves constructing the regression model. This chapter describes each step in the process.

4.1 Textural Feature Extraction

For the system proposed by this approach, all images involved are dealt with as a representation of their textural features as opposed to dealing with the actual images themselves. Textures are complex visual patterns that have characteristics such as brightness, colour and contrast. Because of this, texture is easily perceived by humans and is believed to be a rich source of visual information. Because it is a rich source of visual information, it can be used as a metric for similarity between images as well as for characterizing images. This approach makes use of *Gray Level*

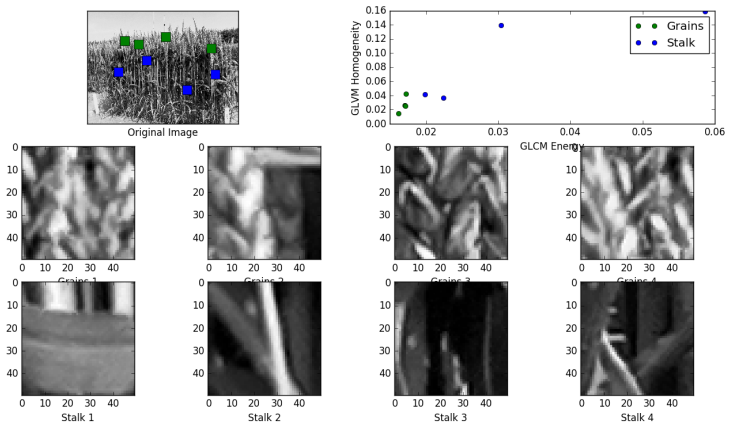


Figure 4.1: Plot of GLCM descriptors demonstrating their discriminative power

Co-occurrence Matrix (GLCM) texture analysis. GLCM texture analysis was chosen because descriptors computed from the GLCM are quite discriminative. Figure 4.1 shows a two-dimensional plot of the GLCM features extracted from wheat grain images and wheat stalk images. It can be seen that GLCM descriptors are able to adequately discriminate between grain regions and stalk regions.

In particular, the GLCM descriptors used to construct the feature vectors are *energy (**ASM**)*, *contrast (**CON**)*, *homogeneity (**HOM**)* and *correlation (**CORR**)*. For each image involved with the system in this approach, its GLCM is computed. Each of the previously mentioned descriptors is computed for an image for when the angle between neighbouring pixels $\theta = 0°$, $\theta = 45°$, $\theta = 90°$ and $\theta = 135°$ and the distance between neighbouring pixels, $d = 0$ to build a 16-dimension feature vector describing the image.

4.2 Building the Model: Linear Regression Analysis for Count Estimation

The prediction model is developed by performing linear regression analysis on the set of extracted texture-based features. The model learns the relationship between the textural features of the image and the number of grains in the image. Regression analysis is a classic statistical method, and its basic idea is that given some value y and another value x, which is a property of y, a function $y = f(x)$ is derived. This function f can then be used to determine the value of $y\prime$ given $x\prime$, where $x\prime$ is a newly encountered value of x and $y\prime$ is its corresponding value of y.

The proposed framework makes use of a linear regression model with the count as the dependent variable and from each of the 4 texture descriptors in the 16-dimension feature vector, it selects the most important values as independent variables. Recall that for each descriptor (*energy*, *contrast*, *homogeneity* and *correlation*), the value of the descriptor is computed in 4 different spatial arrangements (d, θ). This gives the

values ASM_{00}, ASM_{01}, ASM_{11}, ASM_{10}, CON_{00}, CON_{01}, CON_{11}, CON_{10}, HOM_{00}, HOM_{01}, HOM_{11}, HOM_{10}, $CORR_{00}$, $CORR_{01}$, $CORR_{11}$ and $CORR_{10}$. Before building the regression model, Principal Component Analysis (PCA) is applied to the four values denoting the arrangement of each descriptor to yield one representative value for each texture descriptor giving ASM', CON', HOM' and $CORR$'. The linear least squares method is used to compute the regression equation, ie. the model (Charnes et al., 1976). The linear regression model is of the form:

$$y = \alpha ASM' + \beta CON' + \gamma HOM' + \delta CORR' \tag{4.2.1}$$

where α, β, γ and δ are constants calculated by performing a linear regression analysis on the set of training feature vectors and y is the count of grains in the image. Linear regression is used because it does not seem like too much of a stretch to assume that the relationship between the density of grains in an image is directly proportional to the number of grains in the image. This assumption is validated in the Experiments and Discussions chapter.

4.3 Counting With the Regression Model

The only thing needed to count grains in images with this method is the regression equation, more specifically, the estimated coefficients and potentially, the intercepts. This means that the model can be represented in simple terms that are easily readable, understandable and accessible to humans (who aren't necessarily familiar with statistical modeling). It also means that the model can easily be stored and shared once it is computed with little or no cost.

Given a query image whose grain count is desired, the query image is first converted to a textural representation by extracting its GLCM texture features and forming the feature vector as described in the previous section. PCA is applied to each group of the four descriptors in the 16-element vector to yield one representative value for each texture descriptor giving ASM', CON', HOM' and $CORR$'. To obtain the grain count, the calculated coefficients (and potentially, intercepts) are substituted into equation 4.2.1 with the descriptor values. The solution of the equation is the estimate of the number of grains in the image.

Chapter 5

Proposed Approach 2: Using a Counting-by-Detection Method

In this dissertation, two methods for counting wheat grains in images are proposed. This chapter presents the second method which takes a counting-by-detection approach. The proposed method makes use of both image analysis and computer vision techniques and machine learning techniques. The detection of grains to be counted is carried out in a supervised learning manner and the image is manipulated using computer vision techniques. The system aims to first detect all instances of grains in a given image, then compute the count as the number of instances detected. Figure 5.1 illustrates the design of the system and its various components. The system first pre-processes images, extracting the regions of interest, then breaks the image into thousands of sub-images. The sub-images are passed to a classifier which has been trained initially and predicts whether a sub-image contains a grain or not. This chapter proposes a solution to the problem of counting the number of grains in an image. The proposed system is broken down into stages and each stage is described in this chapter.

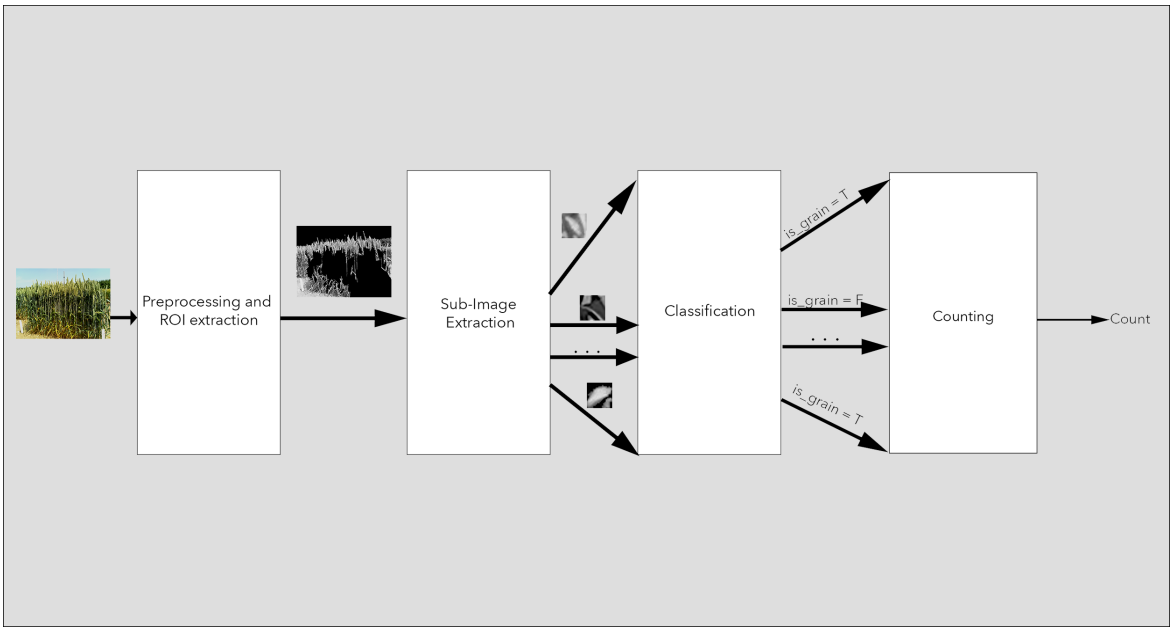


Figure 5.1: Pipeline of Counting-by-detection method

5.1 Grain Counting Pipeline

5.1.1 Image Pre-processing

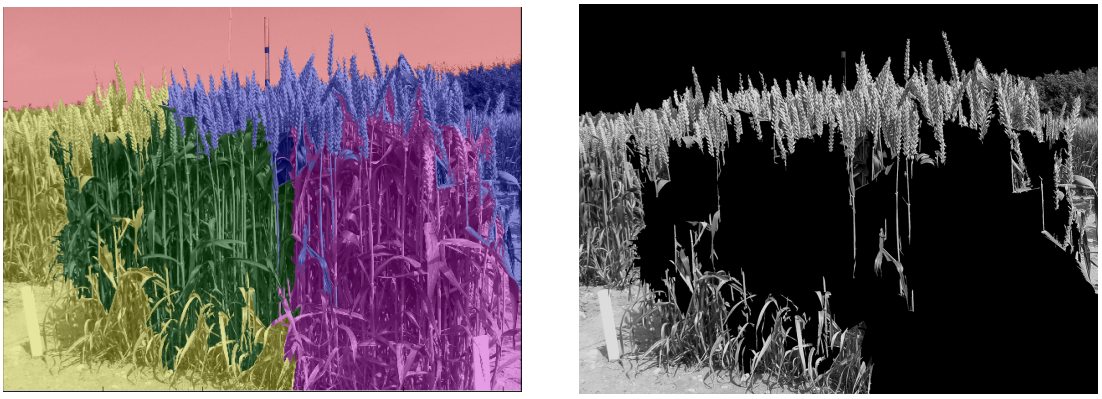
The first step in the grain counting process is to identify and remove all parts of the given images that do not contain grains. The aim of this stage is to reduce the problem space by extracting only the regions of the images that we are concerned with - that is, the spikelets and grains. Region of Interest (ROI) extraction also makes the grain detection and counting process accurate. It removes sky, ground, leaves, and background regions from the image. Otherwise, parts of these regions could be wrongly detected as grains and counted.

ROI extraction is achieved using *segmentation-based object categorization*. This process applies spectral clustering to an image in order to partition the image into distinct regions, known as clusters, based on their colour. The pixels in a cluster are similar to each other in colour but different from pixels in other clusters. In this project, the ***k-Means Clustering***(Hartigan, 1975) algorithm is used to cluster the pixels. Given an image of n pixels $(x_1, x_2, ..., x_n)$ in the HSV space (Joblove and Greenberg, 1978) where each element x_i is a vector made up of that pixels hue, saturation and value, we partition the n pixels into k clusters, $C = \{C_1, C_2, ..., C_k\}$ in order to minimize the distance between every pixel and the mean pixel in each cluster. Mathematically, spectral clustering via kmeans can be denoted as:

$$argmin \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{5.1.1}$$

where μ_i is the mean HSV pixel vector in cluster i

Figure 5.2 shows an illustration of the ROI process. The original image has spectral clustering applied to it as described above (with $k = 5$), and the result of this is shown in Figure 5.2_(a). Each cluster is shown with a different colour overlayed over it. From this image, we can select the clusters which contain relevant information (grains) and do away with other clusters. Figure 5.2(b) illustrates this as it shows the resulting image after the red, green and purple clusters are removed. The resulting ROI image still contains all of the useful information (ie. grains), but only little else.



(a) Image showing cluster memberships (b) ROI image with useful clusters extracted

Figure 5.2: ROI extraction using spectral clustering

5.1.2 Sub-image Extraction

The next step in the grain counting process is to break the extracted ROI image into tiny blocks. These blocks might contain grains or they might contain stalks and leaves that were overlooked in the ROI extraction process. A conceptual summary of the proposed system is that it counts which of these blocks contains a grain, thereby giving a grain count for the whole image. However, it is important to also note that one block might contain more than one grain or even only a portion of a grain. So if the conceptual description is applied, the result would not be very accurate. Strictly speaking, the system only yields an estimate of the grain count. To make this estimate as accurate as possible, the division of the image into blocks has to be done as precisely as possible, with blocks containing grains having either just one grain or a large portion of a grain in them.

The blocks are extracted using a kernel convolution approach. The convolution matrix (or kernel) is an M-by-N matrix containing all 1s. First the image is transformed to greyscale to simplify the convolution process. Each time the kernel is applied, the result of the convolution, which is the sub-image, is stored in a list. Edges are handled using a "cropping" approach. That is, if the kernel cannot completely be placed at an edge, that edge is simply ignored. In our implementation, we use a kernel of size

```
extract_sub_images(full_image, full_image_width, full_image_height, kernel_size){

    // Returns a list containing sub images

    sub_image_list = [];

    xStart = 1
    xStop = (full_image_width/kernel_size.x) * kernel_size.x

    yStart = 1;
    yStop = (full_image_height/kernel_size.y) * kernel_size.y

    for i = xStart to xStop, increment by kernel_size.x
        for j = yStart to yStop, , increment by kernel_size.y
            subImage = full_image[i:i+kernel_size.x, j:j+kernel_size.y]
            sub_image_list.add(subImage)

    return sub_image_list
}
```

Figure 5.3: Pseudocode of sub-image extraction using kernel method

20-by-20 giving square sub-images of length 20. This value is ideal because it is large enough to contain one grain. Due to visual perspective issues, grains closer to the background appear smaller. A 20-by-20 kernel is still small enough to contain only one or one and a bit more of a grain scaled down. Also, with a kernel of this size, the cropping edge handling approach is of no negative consequence as it highly unlikely that more than a minuscule number of grains will be in the remainder of the edge (which would be less than 20 pixels wide).

5.1.3 Classification

This is the main step in the grain counting process. Here, the sub-images extracted from the previous step are classified as either containing a grain or not. In particular, a ***Multi-Layer Perceptron (MLP)*** neural network is used for the classification. The neural network does not deal directly with the images as a whole, but rather with sub-images. Recall that our dataset contains only 12 images which would not be anywhere near sufficient to properly train the neural network. However, because the neural network deals with sub-images, 13 images is a lot more than enough to train it. In fact, it can be trained using just one of the images in the dataset. When this one image is selected, it can then be broken down into its sub-images of size 20-by-20. Each image yields 5000 sub-images on average. This is more than sufficient for training the neural network. Unfortunately, the generated sub-images will still need to be labeled in order to be used for training the neural network. This would be a very time consuming and labour-intensive task. Because of this, only 350 sub-images were hand selected and labeled. The neural network is then trained on these sub-images. When selecting sub-images to be used for training the neural network, care was taken to keep the training data balanced. That is, the numbers of grain sub-images and non-grain sub-images were fairly close. No one class dominated the selected sample.

It is worth noting that the whole sub-image is not passed into the neural network. Instead, discriminative features are extracted from the sub-images to form feature vectors. We make use of textural features based on the textures of the sub-images. Generally speaking, textures are complex visual patterns that have characteristics such as brightness, colour and contrast. Because of this, texture is easily perceived

by humans and is believed to be a rich source of visual information. For each sub-image, the *Gray Level Co-occurrence Matrix (GLCM)* is computed. From the GLCM, we then compute the following texture descriptors - *energy*, *homogeneity*, *correlation* and *dissimilarity*. Each of these descriptors is computed for when the angle between neighbouring pixels, $\theta = 0°, \theta = 45°, \theta = 90°$ and $\theta = 135°$ and the distance between neighbouring pixels, $d = 0$ to build a 16-dimension feature vector describing the sub-image. This means that 16-element vectors are passed through the neural network, as opposed to 20-by-20 matrices (400-element vectors). This saves memory and also allows the neural network to work faster as well. Figure 4.1 shows GLCM descriptors extracted from different parts of a wheat image and plotted against each other. From the figure, it can be seen that GLCM descriptors are able to adequately discriminate between grain regions and stalk regions. By using GLCM feature vectors instead of raw image data, we can boost the discriminative power of the neural network and improve the accuracy of the detection system.

5.1.4 Employing the System to Counting

Once the neural network is built, it can then be used for the detection of grains. Given a query image, it is passed through the pipeline. First, the query image goes through pre-processing to have the ROI extracted from it. The query ROI image is then broken into query sub-images. Next is the detection stage - the neural network is applied to the query sub-images to determine whether a given sub-image contains a grain or not. The number of sub-images classified as containing grains by the neural network is then returned as an estimate of the number of grains in the image.

Chapter 6

Experimentation and Discussion

6.1 Evaluating the Counting-by-Regression Approach

6.1.1 Evaluation Methodology

After designing and implementing the counting-by-regression system, experiments were performed on the system in order to evaluate its performance and validate its worth (or potentially, lack thereof). To do this, statistical techniques were applied to determine whether or not the results quantifying the hypothesized relationships between the density function F and the grain count, obtained from regression analysis, are acceptable descriptions of the data. In particular, the ***goodness of fit*** of the regression model and ***analysis of the regression residuals*** were used to accomplish this. The goodness of fit was estimated using the ***coefficient of determination*** (R^2). R^2 is a number that indicates the proportion of the variance in the dependent variable that is predictable from the independent variable (Trek,). The coefficient R^2 is defined as follows:

$$R^2 = 1 - \frac{u}{v} \tag{6.1.1}$$

where u is the regression sum of squares:

$$u = \sum_{i=1}^{n} (O_i - E_i)^2 \tag{6.1.2}$$

and v is the residual sum of squares:

$$v = \sum_{i=1}^{n} (E_\mu - E_i)^2 \tag{6.1.3}$$

R^2 is a value up to 1.0 that can be negative. However, R^2 can always be increased by adding more variables to the model. This could reduce its usefulness as an evaluation metric in some cases. Because of this, an analysis of the regression residuals is also performed to evaluate the regression model. The *residuals* are the differences between the observed value of the dependent variable (ie. count) and the expected value. Mathematically, the definition of the residual for the i^{th} observation in the data set is written

$$e_i = F(x_i) - y_i \tag{6.1.4}$$

where y_i denotes the i^{th} response in the data set and x_i the input of textural feature vectors to the density function F, each set at the corresponding values found in the i^{th} observation in the data set. If the model fits the data perfectly, the residuals will approximate the random errors that make the relationship between the observed and expected values a statistical relationship (NIST,). Therefore, if the residuals appear to be random, it would suggest that the model fits the data well.

6.1.2 System Performance

After fitting, the linear regression model yielded an R^2 value of 0.83. Considering that an R^2 value of 1 would be a perfect fit, our model seems to fit the data quite well. To eliminate any biases of the R^2 metric, an analysis of the regression residuals

was also performed. The residuals were computed and a scatter plot of the residuals against the predicted responses was drawn and can be seen in Figure 6.1. The residu-

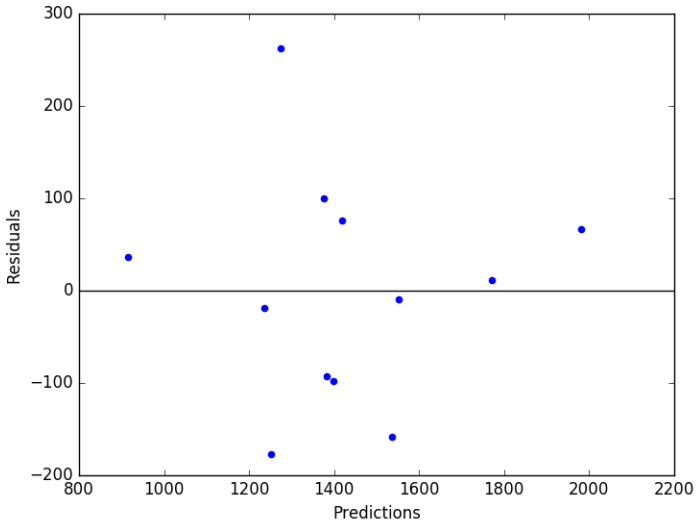


Figure 6.1: Visual analysis of regression residuals

als are scattered evenly and randomly showing that they are consistent with random error. This proves that the regression model is systematically correct. In order to clearly illustrate the how well or poorly the system performed, a plot of the expected responses and observed responses was drawn and is shown in Figure 6.2. While the previous experiments focused more specifically on assessing the quality of the model, this experiment focused more on the big picture of the system, ie. what it predicted compared to what it should have predicted. While the first two experiments asked more of a "will it work?" question, this asked more of a "does it work?" question. Figure 6.1 shows that nearly half the points were on or just off the line while majority of the remaining points fell fairly close to the line. This proves that the derived grain

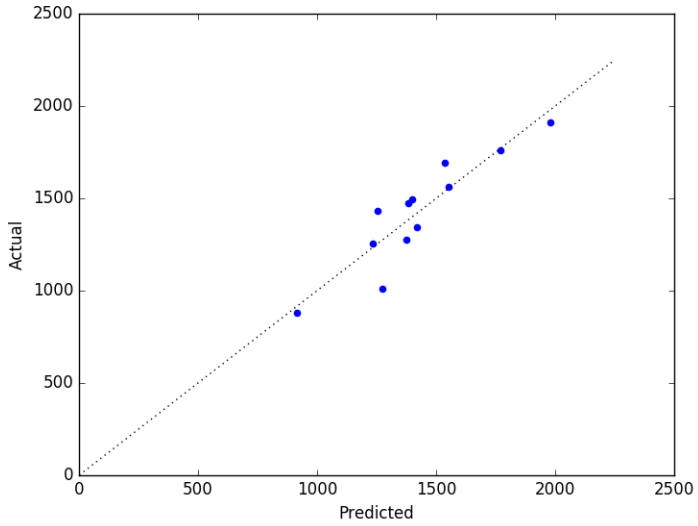


Figure 6.2: Plot of expected responses against predicted responses

density function F was reasonably correct.

However, the derived model could be better were it not for two main factors hindering its performance. The first factor is the fact that only 12 images were used in training and building the model. By normal standards, this is a very small dataset for a typical supervised learning problem. The second factor is the fact that the expected (or target) responses could not have been very accurate as the number of grain in each image had to manually be counted by eye. Only grains in the foreground and larger than a certain scale were counted. To mitigate the issues arising from this, care was taken to count grains in all the training images in this same manner. All in all, the proposed counting-by-regression system seems to perform well but it is hard to say for sure without more images.

6.2 Evaluating the Counting-by-Detection Approach

6.2.1 The Model Setup

The model used for the prediction (and in essence, detection) of grains was the artificial neural network using the Multi-Layer Perceptron (MLP) learning algorithm. The MLP made use of the ***sigmoid function*** σ (also known as the ***logistic function***) as its activation function (Nielsen, 2015). The sigmoid function is as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6.2.1}$$

The MLP used ***backpropagation by means of the stochastic gradient descent*** algorithm for tuning its weights (Bottou, 2010). Neural networks possess a number of *hyperparameters* which cannot be learned by fitting the model to the data but could influence their performance greatly. In particular, the MLP neural network used for this task had the following hyperparameters:

- *hidden layer arrangement* - This refers to the number of hidden layers in the network as well as the number of neurons in each layer.
- *alpha* - This is the *L2* regularization penalty for minimizing prediction error
- *learning rate* - This controls how quickly the gradient descent algorithm travels down the slope to find the minimum of the cost function when tuning the weights of the neurons.
- *batch size* - This refers to the size of the mini-batches chosen by the stochastic gradient descent algorithm to speed up the learning process.

In order to get the best performance from the MLP, an attempt was made to determine the optimum values for these hyperparameters. To do this, the ***Grid Search***

algorithm for hyperparameter optimization was employed. Grid search is simply an exhaustive search through a manually specified subset of the hyperparameter space (Hsu et al., 2003).

6.2.2 Evaluation Methodology

350 images were hand selected and labeled to form a data set. The images were selected in a balanced manner with roughly the same amount of instances of all classes. All of this data was used for training and building the neural network. The same data was also used in evaluating the performance of the neural network. Things were done this way because while there was no shortage of sub-images, they had to be labeled manually. Labeling more sub-images for testing than the 350 images already labeled would have been a time-consuming process. On the other hand, we thought it would have been unwise to split the data into separate training and testing sets as the dataset was not very large to begin with. To overcome this problem, *cross validation* was used for evaluation. Cross validation aims to define a dataset to "test" a model in the training phase, in order to limit problems like overfitting and give an idea of how the model will generalize to an independent dataset. In addition to this, the *sensitivity* and *specificity* of the classifier used for grain detection was also calculated and used as an evaluation metric. Sensitivity and specificity are statistical metrics used to evaluate binary classification problems. Sensitivity (also known as recall or true positive rate) measures the proportion of positive instances that are correctly identified as such. Similarly, specificity (also known as true negative rate) measures the proportion of negative instances that are correctly identified as such.

Mathematically, they can be expressed as follows:

$$sensitivity = \frac{\text{number of true positives}}{\text{number of true positives + number of false negatives}} \tag{6.2.2}$$

$$specificity = \frac{\text{number of true negatives}}{\text{number of true negatives + number of false positives}} \tag{6.2.3}$$

Sensitivity estimates how well a classifier avoids false negatives while specificity estimates how well it avoids false positives. A perfect classifier would have 100% sensitivity and 100% specificity, that is, a high sensitivity and an equally high specificity. However, in practice there is usually a trade-off between the two.

6.2.3 System Performance

For evaluating the accuracy of our MLP classifier, 5-fold cross validation was used. The data was split into five sets and five iterations of the following procedure were carried out. At each iteration, one set was selected to be the training set (used to build the classifier) and the remaining four were used to test the classifier. The set selected as the training set was changed in each of the five iterations. Once all the iterations were completed, the accuracies from each one was averaged to give the mean accuracy of the classifier. This was needed to get as good a classifier as possible. This is because MLP neural networks start with random weights assigned to the neurons. The values of these starting weights affect the weights that the MLP ends up with, thereby affecting its performance. To get the best model possible, the neural network was built (with random weights) n times. After each build, it was evaluated using the 5-fold cross validation strategy to paint a picture of its accuracy. The MLP with the best accuracy (found to be 80.68%) was selected and serialized to disk to be used as the model as needed.

Once constructed, the characteristics of the model are evaluated by calculating its sensitivity (true positive rate) and specificity (true negative rate). This gives a better idea of how the classifier performs in a more generalized sense. Table 6.1 shows a confusion matrix illustrating the results of applying the classifier to a test set of sub-images independent from the ones used to train and validate it. From this, we were able to calculate the sensitivity and specificity for the final model after crossvalidation and gridsearching the space of the hyperparameters to produce the optimum model. The final classifier had a sensitivity of 72% and a specificity of 90%. From the confusion matrix and sensitivity and specificity metrics, it can be seen that while the classifier performs fairly well in identifying sub-images that contain grains, it is better at identifying sub-images that do not contain grains. To further examine

		Expected Response		
		True	False	Total
Observed Response	True	TP (34)	FP (4)	38
	False	FN (13)	TN (37)	50
	Total	47	41	$N = 88$

Table 6.1: Confusion matrix for grain classification

the general performance of the classifier, a ***Receiver Operating Characteristic (ROC) curve*** was plotted. An ROC curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. Its discrimination threshold refers to the threshold with which it separates predictions into positive and negative responses. The curve was obtained by plotting the true

positive rate and the false positive rate (1 − specificity) at varying values of the discrimination threshold. The true positive rate represents how many sub-images are correctly classified as containing a grain while the false positive rate represents how many sub-images are wrongly classified as containing grains. Figure 6.3 shows the ROC curve obtained from varying the discrimination threshold. The ROC curve illustrates the trade-off between true positives and false positives. A perfect classifier would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). This means the closer to the top the curve lies, the better the classifier. Our classifier lies fairly close to the top left, further illustrating that it performs reasonably well. Having determined that the classifier could detect grains in images with

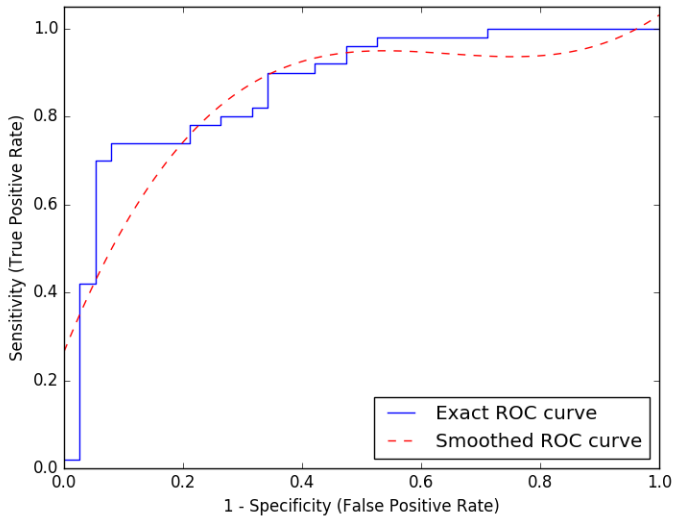


Figure 6.3: Receiver Operating Characteristic (ROC) curve

an accuracy of 80.68%, it would be interesting to find out what this meant for the counting system itself. Tables 6.2 and 6.3 show the actual and predicted counts for

the images provided. From these tables, it can be seen that the system seemed to perform better on certain kinds of images than others. This was due to the fact that in the ROI extraction stage, the clusters to be extracted from the original image were hardcoded. This caused problems because the hardcoded clusters were not always the optimum (or sometimes even particularly good) clusters to select as regions of interest. A way around this problem would be to manually select the clusters denoting regions of interest to be extracted before passing a query image to the system. A simple interactive program was written to this end. Figure 6.4 shows a screenshot of the ROI extraction interface. The program allows a user to upload an image and set the number of clusters that they wish to detect using the k-means clustering algorithm. The program then displays the image with the clusters overlayed over it and the user is able to select the clusters that they would like to extract. While this method might be slightly labour-intensive when a medium to large number of images need to have their grains counted, it yields much better performance. The trade-off here between labour-intensiveness and performance is something that would need to be considered in practice.

6.3 Thoughts and Comparisons

It was difficult to properly evaluate the counting-by-regression approach. While it showed poorer results by being off the actual count by more than the counting-by-detection method, it should be noted that it only had a dataset consisting of 12 images. Using only 12 images would definitely not have built a very good model. The

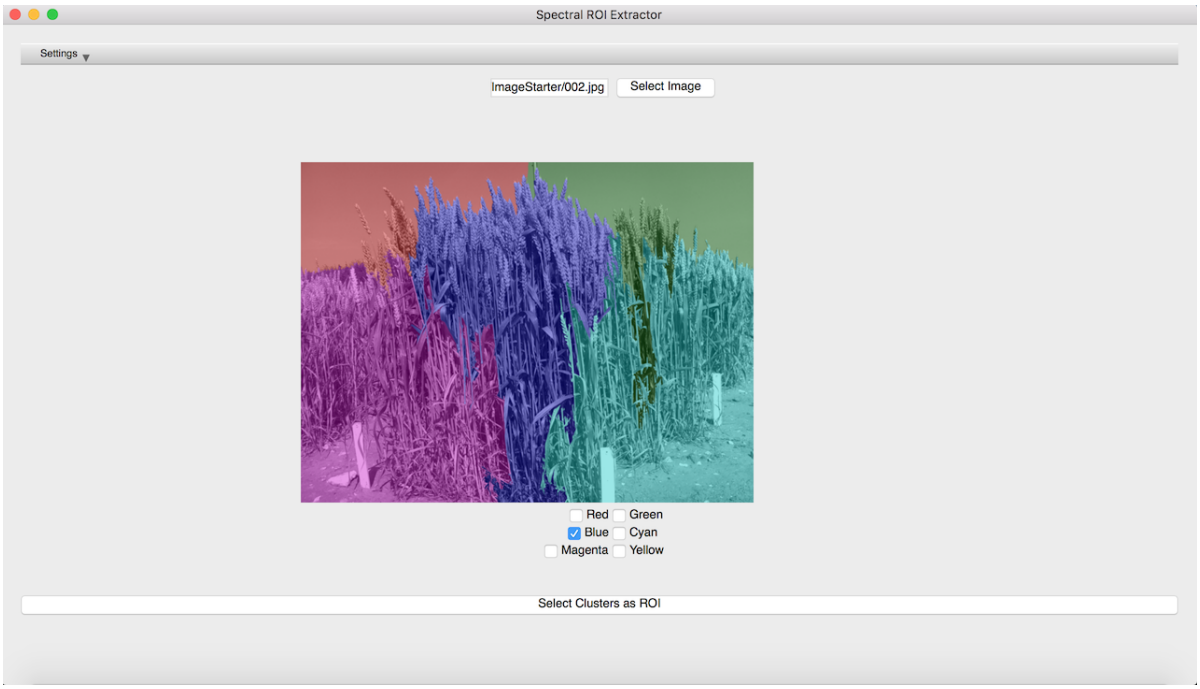


Figure 6.4: Interactive program for manual ROI extraction

counting-by-detection approach on the other hand had access to 5000 potential examples from each wheat image. Therefore, while the counting-by-detection approach out-performed the counting-by-regression approach, it is still too soon to write off the regression approach as being inferior. However, because we know the accuracy of regression approaches is limited as there is no universal function that can map images to grain counts for any and all images. On the other hand, if a system can correctly detect grains in images (of any kind) to a reasonable extent, such a system would outclass any built on regression approaches.

The counting-by-detection system proposed relies on a machine learning classifier for grain detection. Our solution makes use of a neural network. We tried other classifiers to see how they all measured up. For image classification, neural networks

and support vector machines perform best so the two were pit against each other in order to determine which was best for this problem. In particular, an MLP neural network and an SVM with the *radial basis function* as the kernel function were used. While the SVM was hundreds of times faster than the neural network, it only yielded a prediction accuracy of 65.9% while the neural network yielded an accuracy of 80.68%. Despite the fact that the SVM was able to count grains in a fraction of the time that the neural network did, the difference in prediction accuracy was too great to be disregarded in favour of speed. There are still more kinds of neural networks besides the MLP variation used in our solution and in the future, we intend to look into how they can be applied to the problem. Convolutional neural networks are one such example.

Also, for the proposed counting-by-detection approach, the size of the kernel used for its sub-image extraction is an important component that will be needed to be considered in practice. In our implementation, we have used a kernel of size 20-by-20. This size suits the data used in this project. For the scale of our images, this is an adequate size. Also, all of our images were taken from the same distance so had the same scale. This means that the same kernel size could be used for all the images. However, in practice, the scale of a query image and its contents could vary from that of the images observed in training. This means that consideration needs to be put into whether or not the kernel size needs to be changed between applications and by how much they need to be changed. There is a simple and naive way of determining how to change the kernel size but it depends on the assumption that the training images are all (around) the same scale. To naively determine the kernel size for a query

observation, it is possible to compute the ratio of the scale of the observed image to the training image. The ratio can then be multiplied by the width and height of the kernel in order to scale it up or down.

Image	Actual count	Predicted count
	1432	1252
	1760	1771
	1914	1980
	880	916
	1562	1552
	1276	1376

Table 6.2: Results of proposed system

Image	Actual count	Predicted count
	1694	1536
	1497	1398
	1474	1381
	1342	1417
	1012	1274

Table 6.3: Results of proposed system

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this paper, two frameworks for counting the number of wheat grains in images is proposed. The first framework proposed takes a counting-by-regression approach to solving the problem. Given N training images, this method computes a grain density function F as a real function of pixels in the images. The grain density function, F, then provides a mapping between the density of grains in the image and the number of grains in the image. New images can then be provided as input to F and an estimate of the number of grains in the provided image returned as output. The regression model used to compute F was evaluated using the R^2 metric and was found to have a score of 0.83.

The second framework proposed takes a counting-by-detection approach which calculates estimates of grain counts by attempting to detect instances of wheat grains in a given image. The image is broken down into thousands of tiny sub-images. A neural network is then employed against the sub-images to predict whether or not they contain a grain. The number of sub-images classified to contain a grain is counted

and returned as an estimate for the number of grains in the image. Experiments were performed with this framework and it was found to have an 80.68% prediction accuracy.

When compared to each other, the counting-by-detection framework was found to perform better. This might however be due to the fact that the dataset provided contained only 12 images and the regression-based solution did not have the advantage of being able to use sub-images obtained from dividing the original images. A general limitation of the counting-by-detection framework is the automation of the process of extracting regions of interest to be examined for grains. To help mitigate the effects of this, a standalone interactive program for manually extracting regions of interest was developed which can be used with this approach.

7.2 Future Work

The classifier used for grain detection only has a binary target. This means that it classifies sub-images as either containing a grain or not. In the future, the classifier can be set to have a categorical target. This would allow it to classify sub-images as containing a grain and not containing as well as containing part of a grain. Other nominal options such as “is sky”, “is stalk” and “is ground” could also be added, making the classifier more robust. This would make the system less dependent on the ROI extraction as it would not need the sky or ground regions to be removed because it can already tell them apart.

Furthermore, in the future, the MLP neural network being used for grain detection

could be replaced with a CNN. CNNs are more suited to image and video recognition tasks than other neural networks. They are good at learning the important features from basically any data structure, without having to manually derive features. This would eliminate the need to compute GLCM features from sub-images and save processing time.

Bibliography

Ahuja, N. and Todorovic, S. (2007). Extracting texels in 2.1 d natural textures. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

Anderson, M., Srinivasan, C., Motta, R., and Stokes, M. (1996). A standard default color space for the internet: srgb. Technical report, Technical report, International Color Consortium.

Anoraganingrum, D. (1999). Cell segmentation with median filter and mathematical morphology operation. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1043–1046. IEEE.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer.

Bengio, I. G. Y. and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Canny, J. (1986). A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698.

Charnes, A., Frome, E., and Yu, P.-L. (1976). The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *Journal of the American Statistical Association*, 71(353):169–171.

Church, J. C., Chen, Y., and Rice, S. V. (2008). A spatial median filter for noise removal in digital images. In *Southeastcon, 2008. IEEE*, pages 618–623. IEEE.

Cook, J. D. (2009). Three algorithms for converting color to grayscale. `http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/`.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

Descombes, X., Minlos, R., and Zhizhina, E. (2009). Object extraction using a stochastic birth-and-death dynamics in continuum. *Journal of Mathematical Imaging and Vision*, 33(3):347–359.

Hartigan, J. A. (1975). Clustering algorithms (probability & mathematical statistics).

Hayes, J. (2007). Image classification - gray level co-occurrence matrix (glcm). `http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Hayes_GreyScaleCoOccurrenceMatrix.pdf`.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

Joblove, G. H. and Greenberg, D. (1978). Color spaces for computer graphics. In *ACM siggraph computer graphics*, volume 12, pages 20–25. ACM.

Kopparapu, S. and Satish, M. (2014). Optimal gaussian filter for effective noise filtering. *arXiv preprint arXiv:1406.3172*.

Lab, L. Convolutional neural networks (lenet). http://deeplearning.net/tutorial/lenet.html.

Lempitsky, V. and Zisserman, A. (2010). Learning to count objects in images. In *Advances in Neural Information Processing Systems*, pages 1324–1332.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee.

Makandar, A. and Halalli, B. (2015). Image enhancement techniques using highpass and lowpass filters. *International Journal of Computer Applications*, 109(14).

Mathworks (2014). Feature extraction for compact representation of image data in computer vision. http://uk.mathworks.com/discovery/feature-extraction.html.

Nielsen, M. A. (2015). Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/.(visited: 01.11. 2014)*.

NIST. Engineering statistics handbook. http://www.itl.nist.gov/div898/handbook/pmd/section4/pmd44.htm.

Rabaud, V. and Belongie, S. (2006). Counting crowded moving objects. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 705–711. IEEE.

Sella, F., Berteletti, I., Lucangeli, D., and Zorzi, M. (2015). Spontaneous non-verbal counting in toddlers. *Developmental science*.

Singh, K., Ma, M., Park, D. W., and An, S. (2005). Image indexing based on mpeg-7 scalable color descriptor. In *Key Engineering Materials*, volume 277, pages 375–382. Trans Tech Publ.

Smola, A. J. et al. (1996). Regression estimation with support vector learning machines. *Master's thesis, Technische Universit at M unchen*.

Song, H. A. and Lee, S.-Y. (2013). Hierarchical representation using nmf. In *International Conference on Neural Information Processing*, pages 466–473. Springer.

Stergiou, C. and Siganos, D. (1996). Introduction to neural networks. `https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks`.

Swain, M. J. and Ballard, D. H. (1992). Indexing via color histograms. In *Active Perception and Robot Vision*, pages 261–273. Springer.

Trek, S. Coefficient of determination. `http://stattrek.com/statistics/dictionary.aspx?definition=coefficient_of_determination`.

Wikipedia (2016). Image segmentation. `https://en.wikipedia.org/wiki/Image_segmentation`.