CROP AND WEED STEM CLASSIFICATION WITH CONVOLUTIONAL LONG SHORT-TERM MEMORY NETWORKS

by

Dieudonné N. D'Arnall

Thesis submitted in partial fulfillment of the

requirements for the Degree of

Bachelor of Science with

Honours in Mathematics and Statistics

Acadia University

September, 2020

This thesis by Dieudonné N. D'Arnall

is accepted in its present form by the

Department of Mathematics and Statistics

as satisfying the thesis requirements for the degree of

Bachelor of Science with Honours

Approved by the Thesis Supervisors

_____ _____

Dr. Hugh Chipman Date

_____ _____

Dr. Andrew McIntyre Date

Approved by the Head of the Department

_____ _____

Dr. Jeff Hooper Date

Approved by the Honours Committee

_____ _____

Dr. Joseph Hayes Date

Acknowledgements

I am deeply grateful for my supervisors, Dr. Hugh Chipman and Dr. Andrew McIntyre, for their abundant guidance and encouragement throughout this whole process. I would like to thank the Acadia Institute for Data Analytics for this research opportunity, specifically Dr. Daniel Silver for providing machine learning insight, and Keilani Tupper for her support and friendship. I would like to thank Nexus Robotics for an incredible summer work term that started me on this research journey. Thank you to Michelle Larson, Beth Mackenzie, and Acadia's Co-op office for the opportunities you have facilitated for me throughout my degree and for your consistent dedication and enthusiasm. I would also like to thank the ML lab group for their insights and advice as well as providing a supportive community. Finally, I would like to thank my friends and family for their constant support and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

This study applies deep learning neural networks to the problem of identifying crop and weed stem origin points in images. An autonomous weeding robot drives through a field, taking overhead images as it drives over the crop row. A sample of images, with weed and crop labels added by an operator, is used to train a neural network. The trained neural network determines the locations of weed stem emergence points, then a mechanical picking arm travels to the location of the weed and pulls it from the ground. The sequential nature of the image data over time raises the question of whether learning sequential features in addition to local features of the data could improve weed classification. This thesis offers an investigation into the utility of convolutional long short-term memory layers for classifying crop and weed stem origins from a sequence of two images and introduces a novel weed/crop confusion matrix for evaluating model performance.

1 Introduction

1.1 Motivation

Weeds are a common problem in farming. They reduce crop productivity and thus the profitability of farms. Weeding requires costly human labor or environmentally harmful herbicides. An alternative is to have a robot autonomously drive through a field of crops identifying weeds and removing them, requiring no human labor or chemicals. Nexus Robotics developed such a robot, pictured in Figure 1.1.

Figure 1.1: Nexus Robotics autonomous weeding robot "R2-WEED2". [1]

The robot's computer vision system identifies stem emergence points from images captured by two high-resolution cameras mounted on the robot. One camera is mounted on the base frame of the robot and takes overhead pictures of the crop row. The other is attached to a picking arm with a mechanical gripper on the end, used to pull the weeds once they are identified. Each of the cameras takes images at a predetermined rate as the robot drives, feeding the overhead images to a neural network which finds the

locations of the weed stem emergence points. Once the emergence point of a weed is successfully identified, the robot can deploy the picking arm and pull the weed, then move on to find the next candidate point for removal. Accurate discrimination between weeds and crops, in addition to the identification of the precise locations where weed stems emerge from the soil, is critical to the weeding task.

From a machine learning perspective, this application is a *supervised* machine learning problem. Prior to this research, a convolutional neural network (CNN) was the supervised learning algorithm. The CNN is trained to identify weed and crop stems in a sequence of images from the robot, where each pixel is labeled weed stem, crop stem, or background. The training uses one image as input and its corresponding label image is the target output.

This vision system identifies weeds at each time step independently. That is, each image is fed through the network to produce a separate output. The sequential nature of the image data over time raises the question of whether learning sequential features in addition to local features of the data could improve weed classification.

The utility of image sequences over time in attempting to identify weeds and their emergence points is of particular interest to the present study. Three models are trained against a dataset of hand-labeled carrot crop imagery: a fully convolutional encoder-decoder network (CNN), a hybrid CNN-convolutional long short-term memory (ConvLSTM) network and a full ConvLSTM network. The hybrid CNN-ConvLSTM and full ConvLSTM networks each accept a sequence of two images and output the pixelwise segmentation of the second image in the sequence while the CNN accepts only one image at a time.

The study presents a novel weed/crop confusion matrix algorithm to evaluate model performance. The algorithm counts the number of weeds and crops found in each of the predicted images and target images to determine weed and crop detection rates as well as rates of incorrect predictions.

The remainder of the thesis is organized as follows: Chapter 2 provides preliminary information on supervised learning, shallow neural networks, and deep neural networks, including CNN, RNN, LSTM,

and ConvLSTM architectures. Chapter 3 summarizes other academic work relevant to the present study. Chapter 4 overviews the methods used including descriptions of the data, model architectures, and evaluation metrics. Chapter 5 discusses the performance of each model, including the novel performance measure based on the weed/crop confusion matrix. Other comparisons are made, including time to train the models and generate predictions. Chapter 6 summarizes the thesis and discusses directions for future research.

2 Background

This chapter provides a brief description of relevant information to the study. Topics include supervised learning, artificial neural networks – both shallow and deep, including CNN, RNN, LSTM, and ConvLSTM, and encoder-decoder neural network architectures.

2.1 Supervised Learning

Supervised learning is a subset of machine learning in which an algorithm is used to learn a mapping f, from each element in an input set X to an element in an output set Y such that $Y = f(X)$. Training data must contain both an X and a Y set for this mapping to be learned. If the desired output of the function consists of one or more continuous variables, this is called a regression problem. Otherwise, if the aim is to map an input vector to one of a finite number of discrete categories, this is a classification problem. [2].

2.2 Artificial Neural Networks

The human brain is composed of billions of interconnected neurons. Using a large number of these simple processing units, the brain is able to perform extremely complex tasks. Artificial neural networks (ANNs), inspired by biological neural networks, are based on a collection of artificial neurons. Figure 2.1 shows a mathematical model of a biological neuron, called an artificial neuron. Artificial neurons are indexed by k, with a single neuron displayed here.

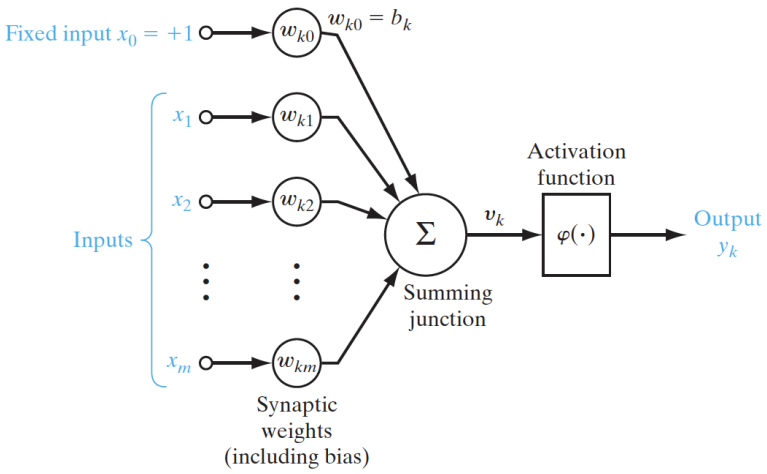


Figure 2.1: McCulloch-Pitts' artificial neuron *k*. [3]

According to [3], the three basic elements of the neural model are:

(1) A set of synapses, or connecting links, each characterized by a weight. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}.

(2) An adder for summing input signal weighted by respective synaptic strengths of the neuron; the operation described here constitute a linear combiner.

(3) An activation function for limiting the amplitude of the output of a neuron. Also traditionally called a squashing function, it limits the permissible amplitude range of the output signal to some finite value.

The output of the neuron in Figure 1 may be represented by the equation

$$y_k = \varphi(\sum_{j=1}^{m}(w_{kj}x_j) + b_k), \qquad (2.1)$$

where b_k is a bias term and $\varphi()$ is an activation function. The lines connecting each input $x_j \in X$ and its assigned weight w_{kj} represent the product of the two terms x_j and w_{kj}. The products of each $x_j \in X$ and its assigned weight w_{kj} are summed, shown as the summing junction in Figure 2.1. The sum of the

products of inputs and weights are then passed through an activation function, such as sigmoid or Rectified Linear Unit (ReLU), described later in Section 2.2.2.

The sigmoid activation function is given by

$$\varphi(a) = \frac{1}{1 + \exp(-a)} = s(a).$$

Letting $x_0 = 1$ and $b_{1k} = w_{k0}$, (2.1) may be written as $y = f(\sum_{j=1}^{m} w_{kj} x_j)$.

This simplified form is illustrated in Figure 2.2.

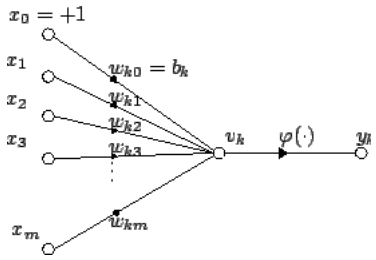


Figure 2.2: Simplified artificial neuron. [4]

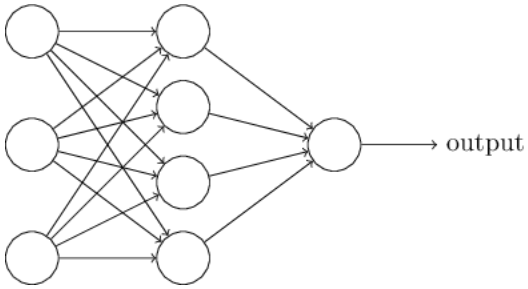


Figure 2.3: ANN with 3 inputs, 1 hidden layer containing 4 hidden nodes and 1 output. [5]

Figure 2.3 shows a configuration of neurons in a fully connected artificial neural network with a single hidden layer. The network accepts three inputs, x_1, x_2, and x_3, represented by the first column containing three nodes (circles). The weights for the network are represented by the edges between each node. The second column of circles represents a hidden layer of neurons, where in each neuron, the products of the inputs and the weights are summed then passed through an activation function. Each of

the outputs from the hidden layer are then multiplied by their respective weights, summed, then passed through an activation function to get the output of the entire network.

The output of the network shown in Figure 2.3 may be represented as

$$y_k = \varphi(\sum_{k=0}^{h} w_k \varphi(\sum_{j=0}^{m} w_{kj} x_j)).$$

Here, the same activation function is used in the hidden layer as well as the output layer. In general, a network may use different activation functions for different layers, depending on the input to the layer and the desired output.

2.2.1 Deep Learning

The human brain is composed of billions of interconnected neurons. Using a large number of these simple processing units the brain is able to perform extremely complex tasks. With multiple levels of processing, it is believed that each level learns features or representations at increasing levels of abstraction. [6].

The human brain uses many levels of processing, it is believed that each level learns features or representations at increasing levels of abstraction.

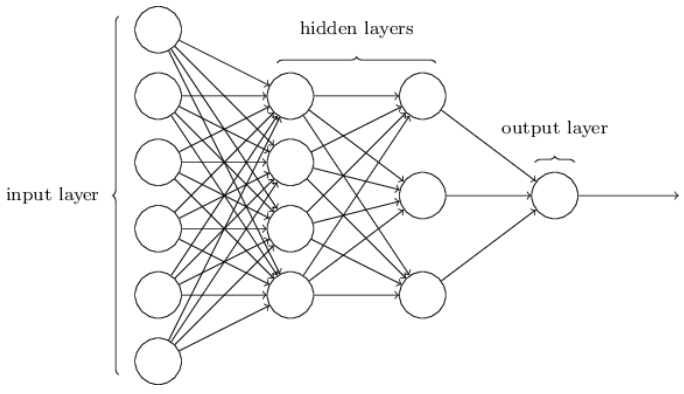


Figure 2.4: Fully connected ANN with 6 inputs, 1 output, and 2 hidden layers. The first has 4 nodes and the second has 3 nodes. [5]

Figure 2.4 shows a fully connected deep artificial neural network with 2 hidden layers. Similar to the network in Figure 2.2, the first column of circles represent the inputs to the network, the edges represent weights, the circles in the second, third, fourth, and fifth columns represent the operations of taking linear combinations of the inputs to the nodes then passing these values to activation functions.

2.2.2 Activation Functions

The descriptions of sigmoid, softmax and ReLU activation functions below are based on [7].

Sigmoid Function

A sigmoid function is a mathematical function which possesses a characteristic "S"-shaped curved, known as a sigmoid curve. A common example of a sigmoid function is the logistic function, shown in Figure 2.5.

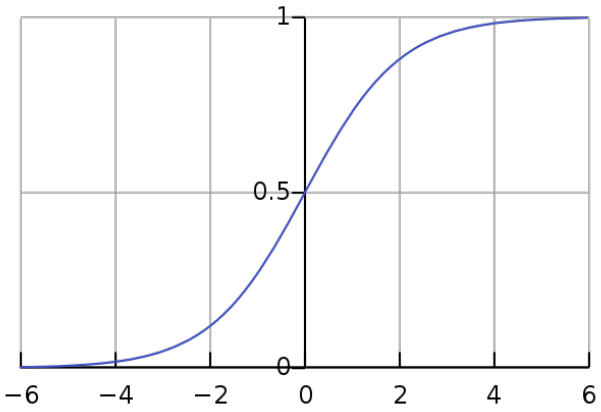


Figure 2.5: Graph of logistic curve. [8]

The logistic function $S: R^1 \rightarrow R^1$ is defined as

$$S(x) = \frac{1}{1 + e^{-x}}.$$

It may be used as an activation function in a binary problem where the output of the function is the probability the input belongs to a specific class, *p(x)=S(x)*. The probability the input belongs to the other class may be found by subtracting *p(x)* from 1.

Softmax Function

The softmax function is a generalization of the logistic function for problems where the number of classes is greater than 2. The softmax function $s: R^K \rightarrow R^K$ is defined as

$$s(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},$$

for $i = 1, \ldots, K$ and $x = (x_1, \ldots, x_K) \in R^K$. The function gives K outputs for the K classes. Each output is bounded between 0 and 1 with the K outputs summing to 1. When the number of classes $K = 2$, the softmax function is equivalent to the logistic function.

Rectified Linear Unit Function

The fact that the gradients of some activation functions are sharp in specific directions and change slowly or are even zero in some other directions creates a problem for algorithms for learning parameters of the model. How the gradient flows within the network is a common problem for most learning-based systems.

The Rectified Linear Unit (ReLU) activation function performs a threshold operation where input values less than zero are set to zero. Thus ReLU is defined by the function

$$f(x) = \begin{cases} x, & if\ x \geq 0 \\ 0, & if\ x < 0 \end{cases}.$$

The function gets its name from its ability to rectify the values of negative inputs by forcing the values to be zero. ReLU functions are popular because they speed up network training. [9]

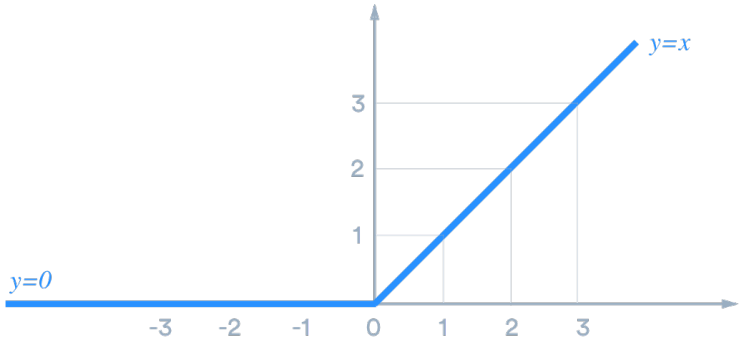


Figure 2.6: Graph of ReLU function. [9]

2.2.3 Network Training

The standard method for learning weights in a neural network is backpropagation [3]. A loss function evaluates a network's error using the output predicted by the model and the target output. The backpropagation algorithm seeks to minimize this error or "loss" as a function of the weights. To do this, the derivative of the loss function is taken with respect to each of the weights. Starting from the output layer of the network, the error is backpropagated through each layer to the input layer.

Some popular loss functions include Mean Squared Error (for regression problems) or Cross-Entropy (for classification problems). Section 4.2.2 provides more detail and examples of loss functions.

2.2.4 Regularization/Dropout

The description of regularization and dropout is based on [10].

Overfitting refers to the memorization of training data when the learned network weights produce output unrealistically close to the target training data. When the model is given new previously unseen data, the predicted output will be much farther from the target output. This is known as a large generalization error. Training a model for too long or using a small dataset increases the risk of overfitting the model to the training data, creating errors in generalization when the model is given previously unseen data. Regularization techniques constrain the network weights, helping to prevent overfitting. Dropout is one regularization technique.

The term "dropout" refers to dropping out network units, that is temporarily removing the units along with its incoming and outgoing connections. The units that are dropped are selected at random and may be hidden or visible. The probability of each unit in a layer being dropped at any iteration is controlled by a hyperparameter set by the user. During training iterations when a hidden unit is dropped, the learning algorithm does not update the corresponding weights. Figure 2.7 illustrates an example of a standard neural network before and after dropout is applied. In Figure 2.7 the probability each unit is dropped might have been 0.5. We see a random realization of 15 Bernoulli trials with $p=0.5$, giving in this case 7 out of 15 units dropped.

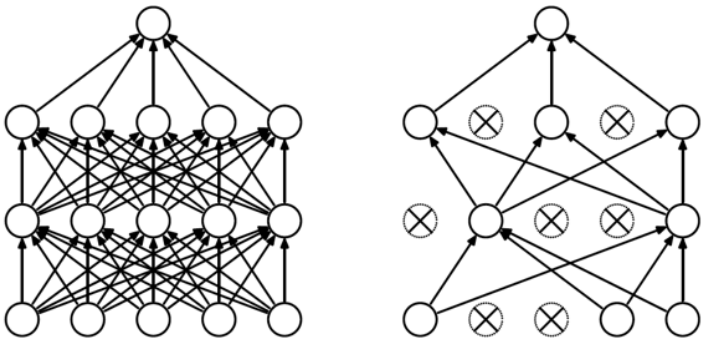


Figure 2.7: Example of standard neural network with 2 hidden layers (left) and example of a thinned network produced by applying dropout to the network on the left where the units with "X" have been dropped (right). [10]

Later in Chapter 4, the application of dropout will be represented by dropout layers, specifying where in the network dropout is applied.

2.3 Image Classification

Pixels in an image may be represented as a matrix of size (height, width) with entries ranging between 0 and 255. A greyscale image has one channel, where a pixel value of 0 represents black and a value of 255 represents white. A colored image has three channels, red, green and blue, stacked on top of each other, giving the matrix a third dimension.

The image classification problem is the task of assigning an input image one label from a fixed set of categories. A pixel-wise image classification problem assigns a label to each pixel as opposed to having only one label for the entire image. In pixel-wise image classification, the number of outputs of the network would equal the number of pixels in the image.

The pixel-wise image classification is also referred to as semantic segmentation. Figure 2.8 shows an example of an image and a semantic segmentation of the image with 8 class labels, namely, road, pole, sidewalk, vegetation, building, vehicle, fence, and unlabeled. Each of the pixels in the original image is

mapped to the color corresponding to the appropriate label in the segmentation image. The key of which color represents which label is found below the segmentation image.

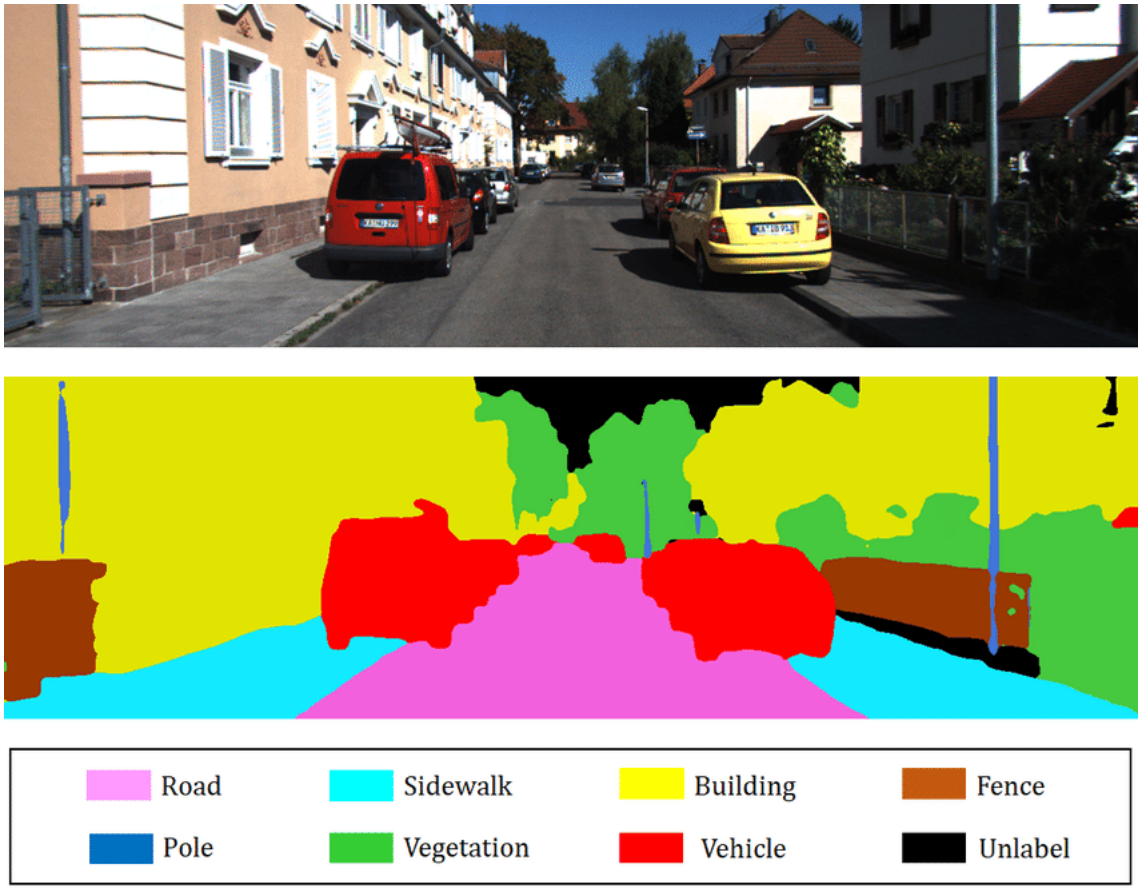


Figure 2.8: Semantic segmentation example. [11]

2.4 Convolutional Neural Networks

The description of Convolutional Neural Networks, convolution, and pooling is based on [12, 14].

A Convolutional Neural Network (CNN) is a type of neural network found to be very effective in areas such as image recognition and classification. A basic CNN architecture is composed of convolutional layers, activation functions, pooling or subsampling, and a fully connected layer at the end of the network for classification.

2.4.1 Convolution

For applications in image recognition or classification, the primary purpose of convolution is to extract features from the input image. Image features are learned using small squares of the input data to preserve the spatial relationship between pixels.

In a convolutional layer, a small matrix, called a 'filter' or 'kernel', slides over the input matrix shifting its position a specified number of pixels each time. At each position, element-wise multiplication between the two matrices is performed, and the products are summed to produce an integer that is a single element in the output matrix. Figure 2.9 gives an example of a convolution with a 3x3 kernel.

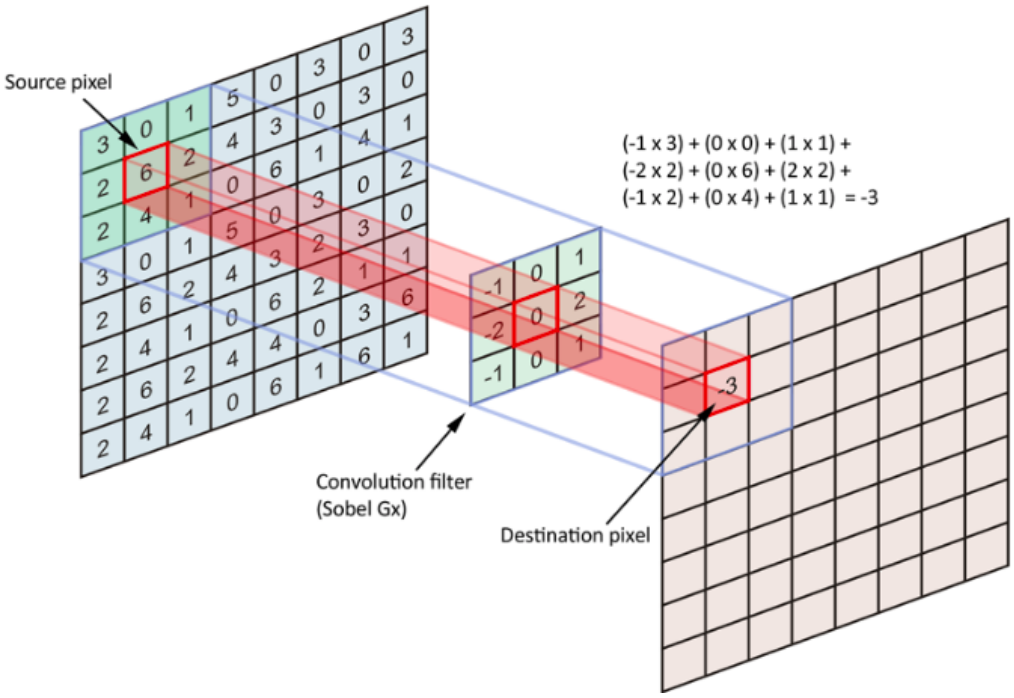


Figure 2.9: Example of convolution with 3x3 kernel. [13]

The output matrix of a convolution is called the 'convolved feature' or 'feature map'. The filters act as feature detectors from the original input image. Different values in the filter matrix produce different feature maps of the same image. The values in the filter matrix are weights of the network, and so are learned during training. With more filters, the network will extract more image features, enabling

the network to better recognize patterns in unseen images. Table 2.1 shows how different filter weights can extract different features from an image.

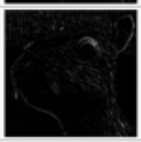
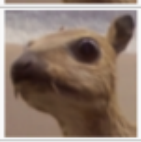
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Table 2.1: Different filters and corresponding output images. [12]

The size of the feature map depends on three parameters; depth, stride, and zero padding. Depth is the number of kernels used. Stride is the number of pixels the kernel is slid over the input image between computations. Zero padding is the addition of zero cells around the border of the input image so that the filter may be applied to the pixels along the border.

The convolution example in Figure 2.9 uses one 3x3 kernel so the depth of the convolution is 1. The example uses zero padding of one pixel along both the vertical and horizontal dimensions of the feature map. The example may use a stride value of 1 although it is not explicitly stated. A stride value of 1 would correspond to moving the kernel one cell to the right, covering the 3x3 square with 0, 1, 5 along the top and 4, 1, 0 along the bottom.

2.4.2 Pooling

In addition to convolution, pooling operations are another essential part of CNNs. Pooling operations are down-sampling functions which summarize subregions to reduce the size of feature maps. These subregions are usually non-overlapping. Some examples of potential pooling functions are taking the average (average pooling) or taking the maximum (max pooling) of the values in each subregion.

Similar to convolution, pooling is done by sliding a window across the input. However, in a pooling layer, the values in the window are fed to a predetermined pooling function instead of taking a linear combination described by the kernel (as in a convolutional layer). A pooling layer, like a convolution layer, depends on the parameters kernel size, stride, and zero padding.

Suppose a 2x2 max pooling kernel was applied to the image in Figure 2.9 instead of the convolution kernel. Using a stride of 2 in each direction and no zero padding, the 8x8 matrix will be reduced to a 4x4 matrix. The upper left element of the pooled matrix will be 6, the maximum of the 2x2 region with entries 3, 0, 2, 6 located in the upper left corner of the input matrix.

2.5 Recurrent Neural Networks

The description of RNN and LSTM networks in this section is based on [15].

Humans do not start their thinking from scratch every second. When reading a sentence, each word is understood based on an understanding of previous words. Traditional feedforward neural networks do not have this ability to factor in previous time steps of data when making decisions.

Recurrent Neural Networks (RNNs) contain temporal loops allowing information to persist from one time step to the next. An RNN may be thought of as multiple copies of the same network in succession with each network passing a message to the next.

A visual representation of RNN structure is shown in Figure 2.10. Each block represents a copy of neural network, *A*, that learns sequential features in the form of weights. x_t is the input data at time step *t*, and h_t is the hidden state prediction for time step *t*. Each of the rectangles labeled *A* are copies of the same network, including the same weights. The horizontal arrow connecting the network at time step i with the network at time step $i + 1$ represents a copy of the hidden state output at time step i, h_i, being passed to the network for time step $i + 1$. This allows information to persist over time so that sequentially time-dependent features of the data may be learned by the network.

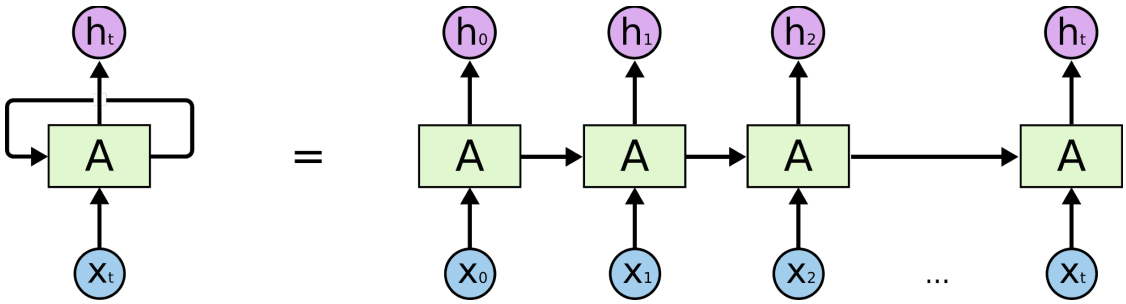


Figure 2.10: Recurrent layer representation as a chain-like structure of layers. [15]

In the application of the present study, x_t is the image at time step *t*. Network *A* learns local features for each image as well as sequential features h_t.

In cases where the gap between relevant information and the place it is needed is small, RNNs are able to learn past information. But as this gap grows larger, RNNs are no longer able to learn to connect the information. This limitation motivates long short-term memory networks.

2.5.1 Long Short-Term Memory Networks

Long short-term memory networks (LSTMs) are a type of RNN designed to learn long-term dependencies. All RNNs have the form of a chain of repeating modules; in a standard RNN this repeating module will have a very simple structure, such as a single tanh layer, illustrated in Figure 2.11 below. This is the same structure as Figure 2.10.

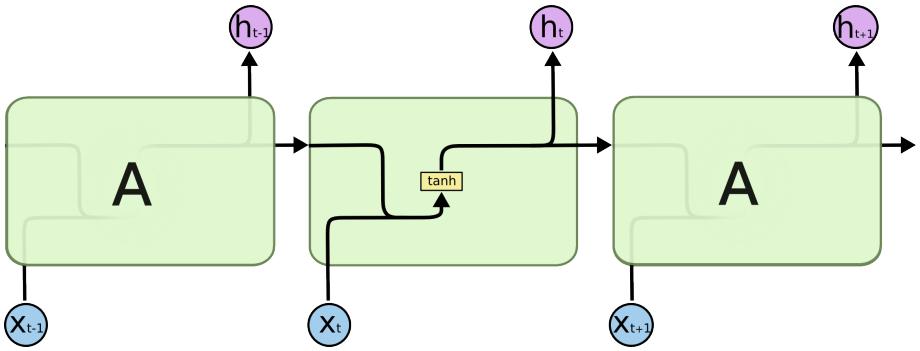


Figure 2.11: RNN representation as a chain of repeating tanh layers. [15]

LSTMs also have this chain-like structure, but the repeating module is composed of four interacting layers; three sigmoid layers and one tanh layer, illustrated in Figure 2.12.

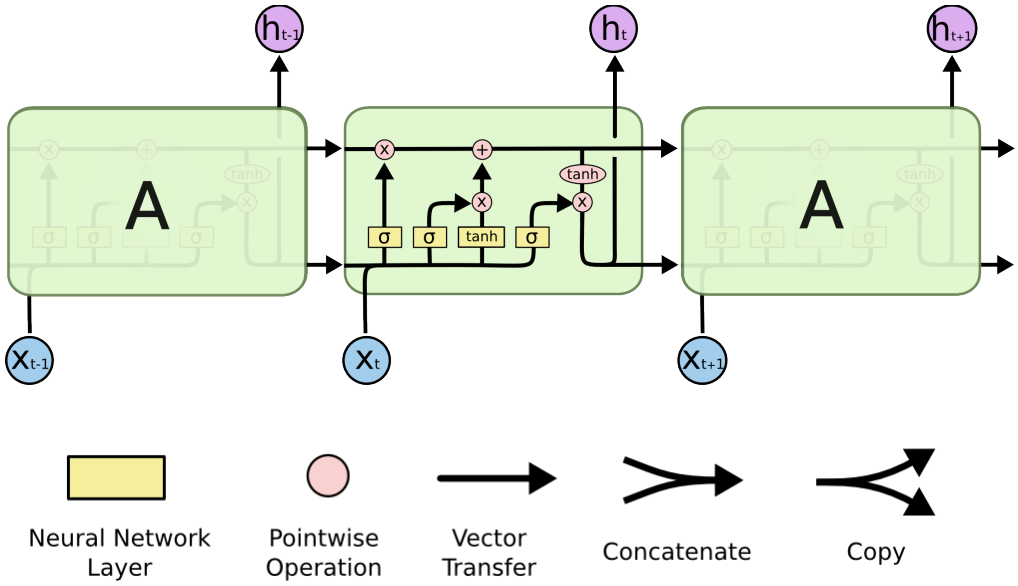


Figure 2.12: Chain representation of LSTM layer. [15]

Unlike a traditional RNN, the LSTM has two kinds of output that get passed to the next layer. In addition to the hidden state prediction, represented by the lower arrow in Figure 2.12, an LSTM also outputs the cell state. The cell state is key to the LSTM. Represented by the horizontal line running through the top of the diagram, it acts like a conveyer belt moving information through the entire chain with only some minor linear interactions.

Gates are structures to optionally add or remove information to the cell state. They are composed of a sigmoid neural network layer and a pointwise multiplication operation; an LSTM has three of these gates. The sigmoid layer outputs a number between 0 and 1, describing how much of each component should be let through.

The first step of an LSTM cell is a sigmoid layer called the forget gate layer, represented in Figure 2.12 by the leftmost yellow rectangle near the bottom of the central network. It takes inputs h_{t-1} and x_t to decide which information to throw away from the cell state of the previous time step, C_{t-1} . Cell state C_{t-1} is the top horizontal arrow feeding into the network.

The next step is deciding what information to store in the cell state. This step is composed of two parts. The first part is the input gate layer, a sigmoid layer that decides which values to update,

represented by the second yellow rectangle. A tanh layer represented by the third rectangle, creates a vector of new candidate values, that could be added to the state.

The old cell state, C_{t-1}, is updated to the new cell state, C_t, forgetting the information that was decided by the first sigmoid and then adding the new candidate values scaled by the output of the second sigmoid layer.

Last is the output gate, a sigmoid layer, represented by the rightmost yellow rectangle. This layer decides which information to output to the next cell. The cell state is put through a tanh layer to push values between -1 and 1. The output of the sigmoid layer and tanh layer are multiplied to produce the hidden layer output to feed into the next cell.

2.6 Convolutional LSTM

A convolutional LSTM (ConvLSTM) layer, proposed by [16], possesses a similar recurrent structure as the LSTM layer. Instead of a full connection, a convolution is performed in both the feedforward and recurrent connections. This allows the model to learn spatial relationships among pixels in an image while also learning patterns among sequential images.

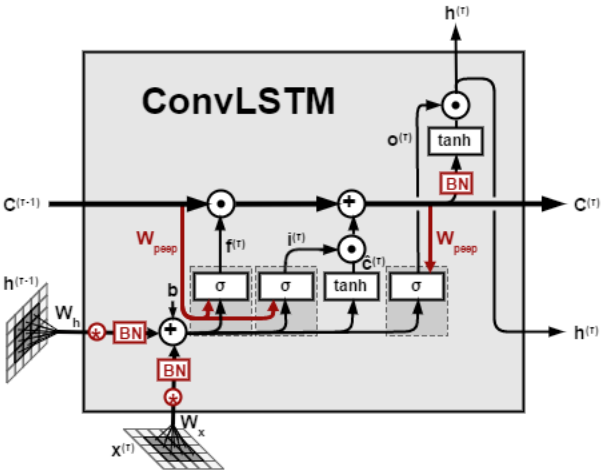


Figure 2.13: Representation of ConvLSTM cell. [17]

A ConvLSTM cell is illustrated in Figure 2.13, with inputs $C^{(t-1)}$, $h^{(t-1)}$ and $x^{(t)}$, where $C^{(t-1)}$ is the cell state from the previous time step, $h^{(t-1)}$ is the hidden state prediction from the previous time step and $x^{(t)}$ is the image at time step *t*. The ConvLSTM cell works similarly to an LSTM cell, except that the input image and hidden state prediction from the previous time step are convolved rather than using a full connection.

2.7 Encoder-Decoder Architecture

An encoder-decoder neural network architecture, also called a "bottleneck network" is used to learn a compressed feature representation of the data before reconstructing the features for better generalization. The encoder is composed of layers to downsample the input space to obtain a smaller, higher-level representation of the data. Pooling layers are one example of a layer used in downsampling. The compressed feature vector is then passed to the decoder, a collection of layers which upsample the input space to a larger size and lower-level representation. Learning a compressed feature representation of data is essential to deep learning. Compressing the features further in a bottleneck architecture requires the network to learn to reconstruct the features of the data, resulting in better generalization. An example of an encoder-decoder architecture is shown in Figure 2.14.

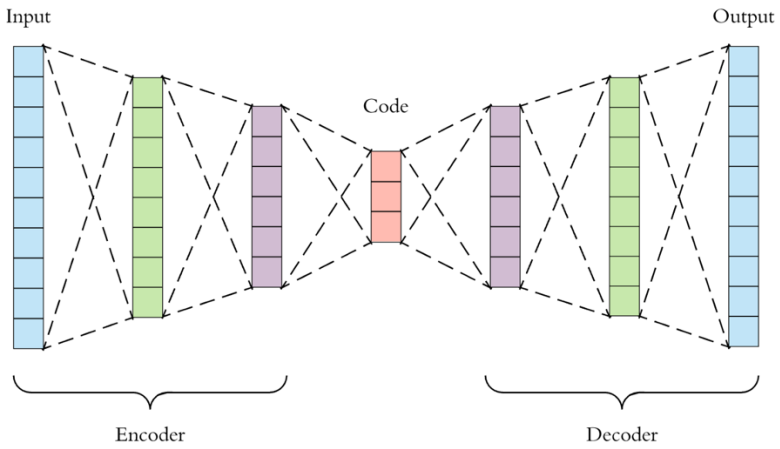


Figure 2.14: Example encoder-decoder network architecture diagram. [18]

Upsampling (in 2 dimensions) is done by repeating a specified number of rows and columns of the data. A simple example would be to double the rows and columns of a 2x2 matrix to obtain a 4x4 matrix, shown in Figure 2.15.

Input =

1	2
3	4

Output =

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Figure 2.15: Example of upsampling.

3 Related Work

For the crop weeding application, some previous studies have been published. Vision-based crop and weed classification systems typically use handcrafted features or end-to-end methods based on convolutional neural networks (CNN). In [19] an architecture is proposed to simultaneously provide a semantic segmentation of the input image as well as the stem locations of weeds and crops. The model uses an encoder to extract a compressed and highly informative representation of the image. This encoded representation is used as input to two task-specific decoders, one to produce a semantic segmentation of the crops and weeds, the other to produce a segmentation of the crop and weed stem regions. This approach works well for image segmentation. Each image is processed independently, so when images are taken in sequence over time, temporal information is not used.

Neural networks capable of using time sequence data are central to this thesis. The remainder of this chapter briefly mentions early research in these areas.

Long short term memory (LSTM) networks were first introduced in [20] to store information over extended time intervals and solve the issues of exploding or vanishing gradients when using back propagation through time, or real time recurrent learning with a traditional RNN. This is done by truncating the gradient where it does no harm to enforce a constant error flow through internal states of special units. Multiplicative gate units learn to open and close access to the constant error flow and allow information to persist between time steps.

Shi et al. proposed the convolutional LSTM (ConvLSTM), using it to build an end-to-end trainable model for solving a spatiotemporal forecasting problem in [16]. ConvLSTMs extend the fully connected LSTM, containing convolutional structures in both the input-to-state and state-to-state transitions. In [16], ConvLSTM layers are stacked in an encoding-forecasting structure to solve the precipitation nowcasting problem.

A comparison of CNN architectures, extended by ConvLSTM layers at different positions for semantic segmentation of video sequences is found in [21]. It is found that placing a ConvLSTM layer

between the encoder and decoder gives the best results. Placing one ConvLSTM layer between the encoder and decoder and a second ConvLSTM layer at the end of the network before the softmax output layer gives the worst results.

4 Methods

4.1 Data

The study uses a dataset of 300 carrot crop images taken by a camera mounted on the base frame of the robot at a rate of 2 Hz as it traverses crop rows. Each image is hand-labeled using a segmentation tool allowing a user to place red and green circles of adjustable size at locations of weed and crop stem emergence points, respectively, while coloring the rest of the image black. Each overhead image and segmentation label image are loaded into each model with a width of 512 pixels, height of 384 pixels, and 3 channels for red, green, and blue. Figure 4.1 shows one overhead image and corresponding segmentation label image.

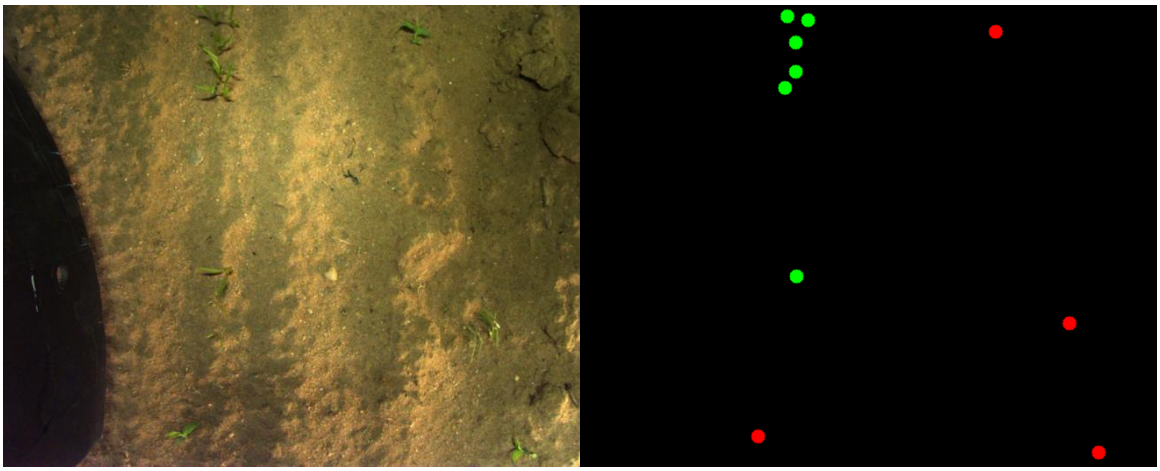


Figure 4.1: Overhead image of carrot crop (left) and corresponding semantic segmentation (right).

Each of the 196 608 pixels in an input image is assigned a class label of either crop stem, weed stem, or background, denoted in the semantic segmentation image with color labels green, red, or black, respectively. Each circle must cover the entirety of the stem emergence point; thus the size of a circle directly coincides with the size of the region in the image where the crop or weed emerges from the ground.

Adjacent color labels are arranged in non-overlapping circular configurations of solid color. The example in Figure 4.1 is representative of the entire dataset, i.e., these features are common to all of the images.

4.1.1 Data Preparation

A channel-wise contrast stretch, illustrated in Figure 4.2, is performed on each image to linearly map the pixel values to a specified range and remove extreme outliers from the image. For each channel in the image, the 1st percentile of pixel values are all set to 0 while the 99th percentile of pixel values is set to 255.

Figure 4.2: Overhead image of carrot crop (left) and image after contrast stretch applied (right).

After the contrast stretch is applied to the overhead image, pixel values are divided by 255, giving values in the range 0 to 1. The label matrix is also divided by 255 then it is rounded so that each pixel value is either 0 or 1. This gives each depth-wise pixel vector in the label image a one-hot encoding where a value of 1 in the first entry and 0 in the other two entries indicate a weed stem, shown in the label image as a red pixel. Similarly, a 1 in the second entry and 0 in the first and third positions indicates a crop stem, shown in the label image as a green pixel. A pixel vector containing only zeros indicates “background”

(non-crop and non-weed), shown by a black pixel in the label image. The only possible values for pixels in the label image are [1, 0, 0], [0, 1, 0], or [0, 0, 0].

4.1.2 Train/Test Sets

The dataset consists of 300 hand-labeled carrot images taken from an overhead camera mounted on the robot where the captured ordering of the images over time is preserved. The first 240 images in the dataset create a train set for the vanilla CNN model; the remaining 60 images are used as a test set. There was some experimentation with including a validation set to help prevent over-fitting. However, this proved to be unproductive for the performance of the model as the images were taken from the training set to create a validation set. This gave the model less images to train on, and so the mapping learned was less robust and the results were worse than when the data was divided into only two sets.

Train/Test sets are slightly different for the other two models, due to their architecture. The CNN-ConvLSTM and ConvLSTM models take as input two sequential images stacked. The set of input images has dimension (299, 2, 384, 512, 3) where 299 is the number of elements, and each element is 2 images of size 384 by 512 pixels in each of the 3 channels. This input set is divided similarly to the set for the vanilla CNN where 80% of the data is used for the train set and the remaining 20% is used for testing. The train set for these models is of size 240, and the test set is of size 59 where the second image of sequence i is the first image in sequence $i+1$ for both the train set and test set.

4.2 Model Architectures and Experiments

All models used in this research are implemented in python using the Keras deep learning library with TensorFlow 2.0 backend [22]. Training parameters are consistent in all three models with the exception of the sizes of the test sets (Section 4.1.2). The models are each created using the Keras Sequential class. The popular Adam optimizer [23] is selected for all experiments. The Keras implementation of binary cross-entropy (BCE) is applied as a pixel-wise loss function for modifying weights during training of each model. The learning rate and number of epochs are fixed (Table 5.1). A 3x3 kernel is used for every

convolution in every network and a 2x2 kernel is used for all pooling and upsampling. All hyperparameters not explicitly mentioned here or in Chapter 2 are set to default values.

To train the models in a timely fashion, the Google Cloud Platform (GCP) Compute Engine (CE) is used. The virtual machine chosen for this work is the preconfigured Deep Learning VM [24]. The compute environment is also configured to use a dedicated NVIDIA Tesla V100 GPU for achieving fast tensor math operations under Tensorflow.

4.2.1 Model Architectures

Baseline "Vanilla" CNN Architecture

A plain "vanilla" convolutional neural network is the most basic model of the study. Used as a foundation to build other more complex models, the vanilla CNN serves as a simpler "baseline" model for comparison to the other models. The model has an encoder-decoder architecture, accepting one image as input and outputting a semantic segmentation of the crop and weed stem emergence points in the image.

A diagram representing the vanilla CNN architecture is shown in Figure 4.3. The encoder is composed of a convolutional layer at the input layer with dropout and ReLU activation, then a maxpooling layer, followed by another convolutional layer with ReLU activation and maxpooling layer and finally a third convolutional layer with ReLU activation and another maxpooling layer.

The decoder is composed of a convolutional layer with ReLU activation and upsampling layer, then a convolutional layer with dropout and ReLU activation and an upsampling layer, followed by a convolutional layer with ReLU activation and an upsampling layer, then another convolutional layer to obtain the correct dimensions for the output, and finally a softmax output layer.

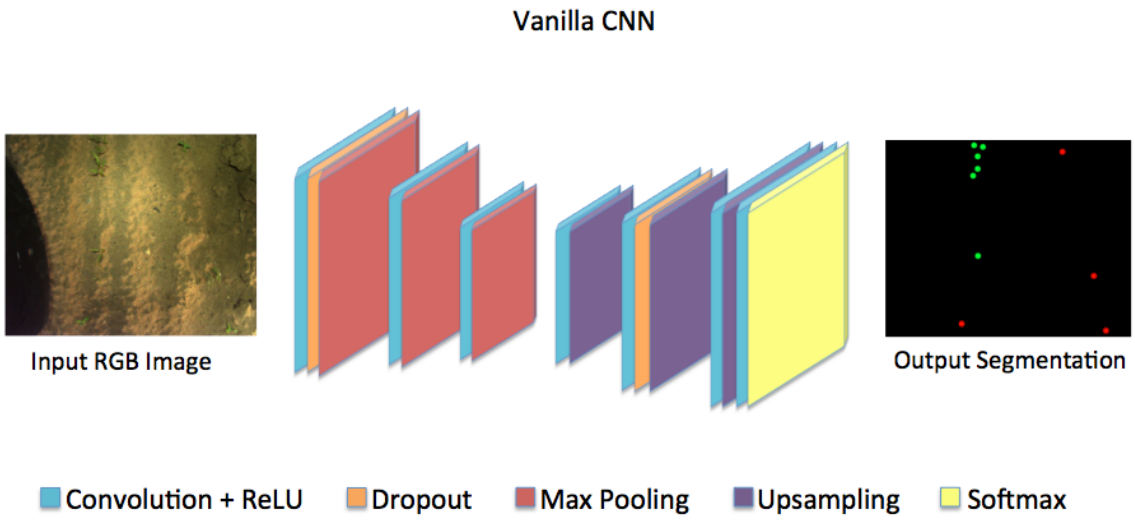


Figure 4.3: Diagram of vanilla CNN architecture.

Hybrid CNN-ConvLSTM model

The Hybrid ConvLSTM model, created to learn sequential features in the images, takes a sequence of two images stacked along the time axis as input, and outputs the label image corresponding to the second input image in the sequence. The architecture is similar to the vanilla CNN but with the addition of two ConvLSTM layers. The first ConvLSTM layer is placed after the encoder, before the decoder, at the bottle neck, or "*chokepoint*" of the network. The second ConvLSTM layer is placed before the softmax output layer at the end of the network. A visual representation of the model is shown in Figure 4.4.

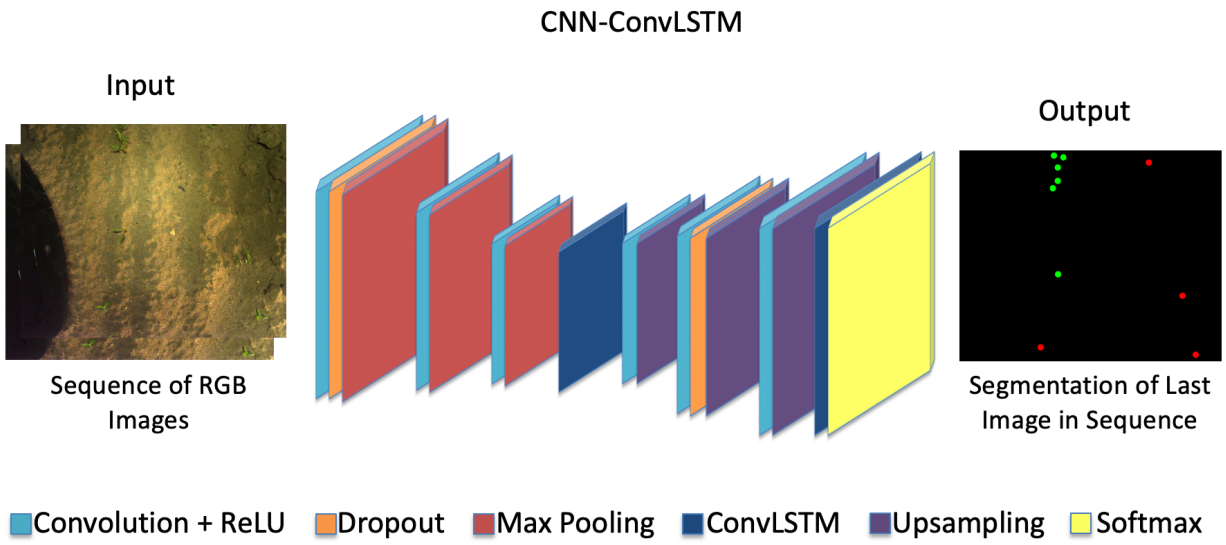


Figure 4.4: Diagram of the hybrid CNN-ConvLSTM architecture.

ConvLSTM model

Expanding on the use of ConvLSTM layers for learning sequential features, the ConvLSTM model architecture replaces each of the convolutional layers in the vanilla CNN with a ConvLSTM with the same number of filters. Like the Hybrid CNN-ConvLSTM model, the ConvLSTM model takes a sequence of two consecutive images stacked along the time axis as input, and outputs the label image corresponding to the second image in the sequence.

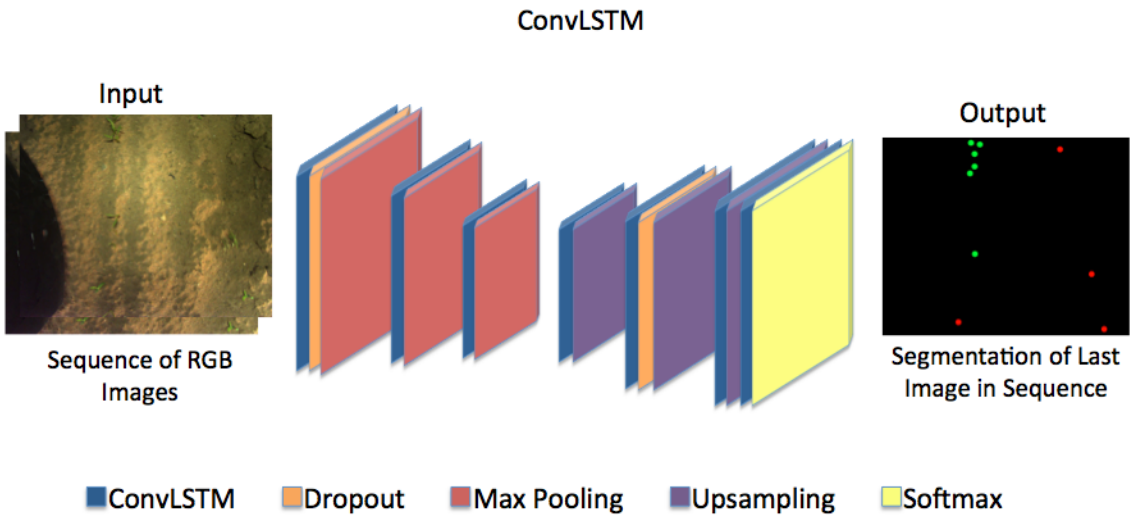


Figure 4.5: Diagram of ConvLSTM model architecture.

4.2.2 Evaluating Error in the Models

The performance of a classification model is measured using a function of the target output, the set of preprocessed label images, denoted y, and the set of predicted image output from the model, denoted $\hat{y}$.

During the training phase, an error function is often called a 'loss' or 'cost' function and is used to update the model weights through backpropagation. Error is used in the testing or 'feedforward' phase to evaluate the performance of the model once the optimal weights have been fixed. The functions used at both of these stages evaluate model error at the level of individual pixels. This section reviews some common loss functions. Near the end of the section a weed crop confusion matrix metric is proposed to evaluate error at the level of clusters of pixels of the same class.

Classification Accuracy

One straightforward metric to evaluate model performance at either stage is classification accuracy, the ratio of the number of correct predictions to the total number of predictions made.

$$\text{classification accuracy} = \frac{\text{correct predictions}}{\text{total predictions}}.$$

Often classification accuracy is presented as a percentage by multiplying the above value by 100.

Classification accuracy alone can be misleading if there is an unequal number of observations in each class or if there are more than two classes in the dataset.

Cross Entropy

Cross entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross entropy loss increases as the predicted probability differs increasingly from the actual label. Cross entropy is represented by the expression

$$-\sum_{classes} y \log(\hat{y}), \tag{4.1}$$

where y is a binary indicator (0 or 1) for class c, and $\hat{y}$ is the predicted probability for class c.

The most used loss function for image segmentation is a pixel-wise cross entropy loss, where each pixel is individually examined by comparing the depth-wise pixel vector of class predictions to the corresponding one-hot encoded target vector. Pixel-wise loss is calculated as the log loss, summed over all possible classes (4.1). The scoring is repeated over all pixels and averaged.

Since this loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, equal learning is essentially asserted to each pixel in the image. This can be a problem if the various classes have unbalanced representation in the image, as training may be dominated by the most prevalent class. The class imbalance in the dataset may be counteracted by weighting this loss for each output channel. Although the class representations in the images of this study are significantly imbalanced (more background than weed or crop), weighting the classes did not seem to improve performance in this application.

Binary Cross Entropy (BCE)

In a binary problem, the log loss formula may be simplified to include only two classes, this is known as binary cross entropy, given by

$$-(ylog(\hat{y}) + (1-y)log\ (1-\hat{y})\ ),$$

where y is a binary indicator (0 or 1) for class c, and $\hat{y}$ is the predicted probability for class c.

Pixel values of the true label image are either 0 or 1, where a 1 in the red channel [1, 0, 0] corresponds to a weed stem origin, a 1 in the green channel [0, 1, 0] corresponds to a crop stem and zeros in every channel [0, 0, 0] corresponds to background.

A binary representation of the problem emerges when viewing each channel of the images separately. Each pixel in the red channel of the target image is either 1 or 0, either the pixel is part of a weed stem, or it is not. Similarly, the green channel of the target image has only values of 1 or 0, either the pixel is part of a crop stem or it is not. The blue channel of each target image is always 0, so the channel is ignored by the learning algorithm. For image data, the blue channel is necessary for displaying colored image output. The softmax function in the final layer of each model outputs pixel-wise probabilities for each class.

Each model in the study is trained using binary cross entropy as the loss function. The loss is calculated separately for each channel in the images and summed:

$$\begin{aligned} Loss = & -(y_R log(\hat{y}_R) + (1-y_R)log\ (1-\hat{y}_R)\ ) \\ & -(y_G log(\hat{y}_G) + (1-y_G)log\ (1-\hat{y}_G)\ ) \\ & -(y_B log(\hat{y}_B) + (1-y_B)log\ (1-\hat{y}_B)\ ), \end{aligned}$$

where $y_R$and $\hat{y}_R$ are the target and predicted outputs for the red channel, y_G and $\hat{y}_G$ are the target and predicted outputs for the green channel, and y_B and $\hat{y}_B$ are the target and predicted outputs for the blue channel. The blue channel does not contribute to the loss, since values are always 0.

Confusion Matrix

The confusion matrix is a technique for summarizing the performance of a classification model in terms of predicted and actual class labels. The confusion matrix shows ways a classifier is 'confused' when

making predictions. The number of correct and incorrect predictions are represented with count values for each class to give insight into which kinds of errors are being made.

For illustration, consider a confusion matrix for a two-class problem. Assume the classes are labelled “positive” and “negative”. A confusion matrix gives counts of the number of observations in all 4 possible combinations of actual class and predicted class. Table 2 shows such a confusion matrix. The four counts are often named: true positive (TP) for correctly predicted positive cases, false positive (FP), for negative cases predicted as positives, false negative (FN), for positive cases predicted as negatives, and true negative (TN) for correctly predicted negative cases.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Table 4.1: Binary confusion matrix. [23]

A two-class confusion matrix is useful for calculating many performance measures, including recall, precision, accuracy, F-score, and AUC. Although the present study contains a three-class problem, only two of the classes are of interest: weed stems and crop stems.

Weed Crop Confusion Matrix

The pixelwise error calculated using BCE is suitable for training and testing the network, however such a loss is not easily interpreted for the application. Evaluating error at the stem level using groupings of pixels of the same class is informative for the practical application of this study in agricultural robotics. The Weed Crop Confusion Matrix is an algorithm designed to evaluate the number of crops and weeds correctly and incorrectly identified.

The algorithm accepts a segmentation image predicted by the model and the corresponding target image. It compares the locations of the crops and weeds in the images and returns a confusion matrix of the form of true positive, true negative, false positive and false negative values corresponding to the number of correctly predicted (true) weeds, correctly predicted (true) crops, incorrectly predicted (false) weeds, and incorrectly predicted (false) crops, respectively.

A false crop represents a weed that was incorrectly classified as a crop. In the weeding application, the robot will not deploy its picking arm to remove such a weed. The surviving weed continues to grow, stealing nutrients from crops. The most dangerous type of error is a false weed, when a crop is misclassified as a weed, resulting in a crop being removed by the robot. Metrics based on errors for individual plants will be most informative.

To illustrate how the weed crop confusion matrix algorithm obtains counts such as false crops, false weeds, true crops and true weeds, consider a small example. Figures 4.6 and 4.7 show an 8x8 grid of pixels with labels W (weed), C (crop) and "empty cell" (background or soil).

W	W		C	C		W	W
						W	W
W	W		C	C			
W	W		C	C			
						W	W
						W	W
			C	C			

Figure 4.6: Target image example.

			W	W		C	C
						C	C
W	W						
W	W		C	C			
			C	C	C		
W		C				W	W
W		C					

Figure 4.7: Predicted image example.

Given Figures 4.6 and 4.7, the algorithm would identify 2 true weeds (left middle and bottom right), 1 true crop (middle), 1 false weed (top middle) and 1 false crop (top right). Additionally, the weeds in the top left of Figure 4.6 are labelled as background. This 1 "false background" case could be counted as a false crop, since the error (not picking a weed) is the same.

The robot's picking arm would be deployed for each of the 4 predicted weed instances in the predicted image. This would result in the removal of the two weeds correctly classified, soil displacement in the bottom left corner of the frame and damage to the crop located in the top center of the frame.

With this intuitive understanding of how crops and weeds would be identified and counted, consider a more detailed description of the algorithm.

First the algorithm breaks the input images into channels to get four matrices: the red channel of the target image, the red channel of the predicted image, the green channel of the target image, and the green channel of the predicted image. These matrices are then rounded so that each of their entries is either equal to 0 or 1. This gives four masks, one for weed stems and one for crop stems in each of the images. The weed and crop masks for the target and predicted image examples (Figures 4.6 - 4.7) are shown below in Figures 4.8 - 4.11.

1	1					1	1
						1	1
1	1						
1	1						
						1	1
						1	1

Figure 4.8: Red channel target image example (target weed mask).

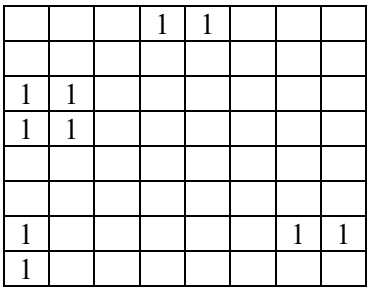


Figure 4.9: Red channel predicted image example (predicted weed mask).

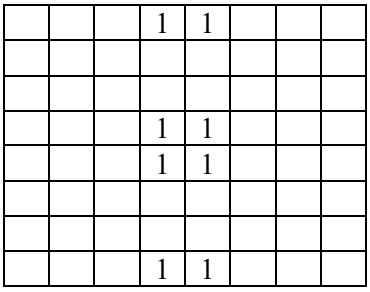


Figure 4.10: Green channel target image example (target crop mask).

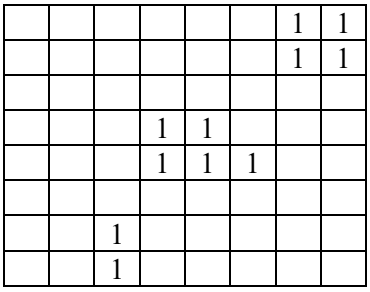


Figure 4.11: Green channel predicted image example (predicted crop mask).

Products of target and predictions masks give masks for pixels present in both matrices. Figure 4.12 shows the "true weed" mask, a product of the target weed mask and predicted weed mask. The true crop mask of the image examples, a product of the target crop mask and predicted crop mask is given in Figure 4.13. The false weed mask, a product of the target crop and predicted weed masks is shown in Figure 4.14 and Figure 4.15 shows the false crop mask, a product of the target weed and predicted crop mask.

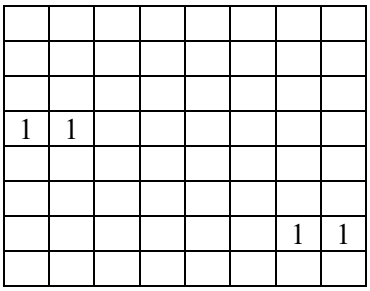


Figure 4.12: True Weed Mask = Target Weed Mask * Predicted Weed Mask.

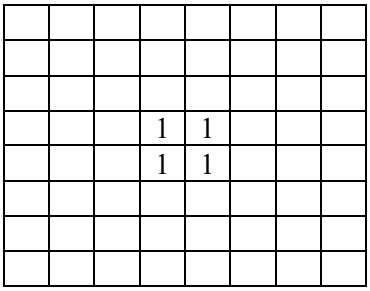


Figure 4.13: True Crop Mask = Target Crop Mask * Predicted Crop Mask.

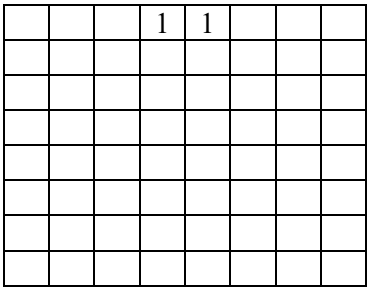


Figure 4.14: False Weed Mask = Target Crop Mask * Predicted Weed Mask.

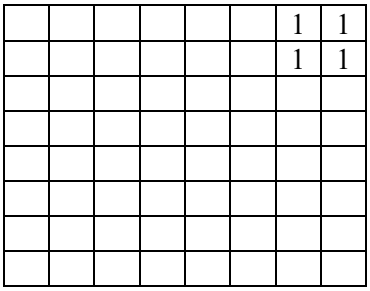


Figure 4.15: False Crop Mask = Target Crop Mask * Predicted Weed Mask.

These masks are passed to a function which counts the number of instances by turning the semantic segmentation mask into an instance segmentation mask. The algorithm traverses the pixels from left to right and top down. When a value of 1 is found, there is a check for other "1" cells in the 8-neighbour points to ensure that instances are not counted more than once. The function returns an instance segmentation of the given mask, with labels starting from 1. Figure 4.16 illustrates the instance segmentation for the true weed mask. The confusion matrix values are the maximum valued entries in the instance segmentation masks.

The algorithm would find confusion matrix values True Weed=2, True Crop=1, False Weed=1, False Crop=1. Total Weed Instances=4 and Total Crop Instances=3 may be found by giving the target images to the semantic to instance segmentation function (instead of the TW/TC/FW/FC masks). The count can be used to calculate $Weeding\ Accuracy = \frac{True\ Weed}{Total\ Weed} = \frac{2}{4} = 0.5$.

1	1						
						2	2

Figure 4.16: True weed instance mask.

Incorrectly predicted weeds and crops in locations labeled background in the target image are not of interest as the robot may deploy the picking arm but likely would not pull a crop. Additionally, one may apply threshold values for the number of pixels in a predicted weed cluster or for some radius for around the center of a predicted crop stem to assist in the prevention of incorrect predictions resulting in crop damage.

Conversely, the robot would not deploy its picking arm for instances of predicted pixels incorrectly labeled background at locations of weeds and crops in the target image. The case of a crop

stem origin being incorrectly labeled background in the prediction image bears little significance to the study since the picking arm would not have been deployed if the crop stem had been correctly identified. However, the case of a weed stem origin incorrectly predicted as background (labeled "False Background" in Table 4.3) is of interest since the weed would not be removed. These false background instances are accounted for in the count of total weed instances in the target images (Total Weed Instances = True Weed + False Crop + False Background).

Below, Table 4.2 illustrates the two-class weed-crop confusion matrix used in the study, where background is ignored. Table 4.3 shows a three-class weed-crop-background confusion matrix showing all three classes.

	Actual Values		
Predicted Values		Weed	Crop
	Weed	True Weed	False Weed
	Crop	False Crop	True Crop

Table 4.2: Weed-crop confusion matrix used in study.

	Actual Values			
Predicted Values		Weed	Crop	Background
	Weed	True Weed	False Weed	-
	Crop	False Crop	True Crop	-
	Background	False Background	-	True Background

Table 4.3: Weed-crop-background confusion matrix containing all three classes.

5 Results

5.1 Training

The training curves of the three models are shown below in Figures 5.1 (CNN), 5.2 (CNN-ConvLSTM), and 5.3 (ConvLSTM). These curves show training loss (BCE) as a function of epoch, or iterations of the training algorithm. The rate of convergence between the models is of interest. All other factors being equal, a model architecture that enables faster convergence is preferred. Convergence is assessed informally by visual inspection of the training curves.

In Figure 5.1, the vanilla CNN trains less evenly for the first 100 epochs then the loss decreases more steadily for the next 300 epochs before converging relatively late in the run.

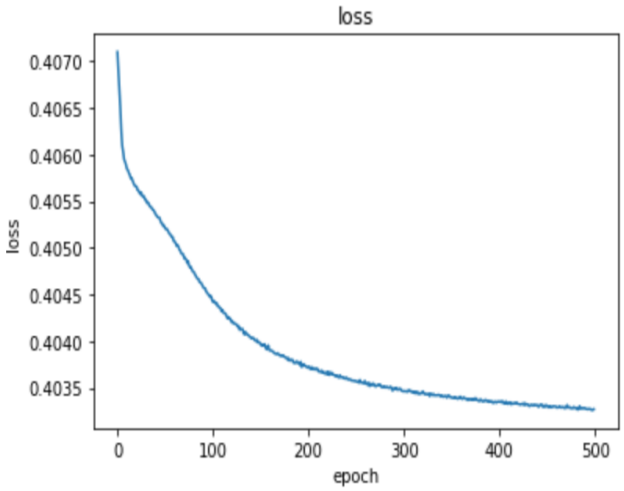


Figure 5.1: Training curve – BCE loss, CNN.

The hybrid CNN-ConvLSTM model trains more slowly than the pure CNN, however the model converges about 200 epochs sooner (Figure 5.2). Notice that in the training curves for both models there is a similar shallow region creating a 'bump' around the 50 epoch mark. After the 50 epoch mark, the hybrid model very quickly returns to steep learning whereas the CNN remains more shallow.

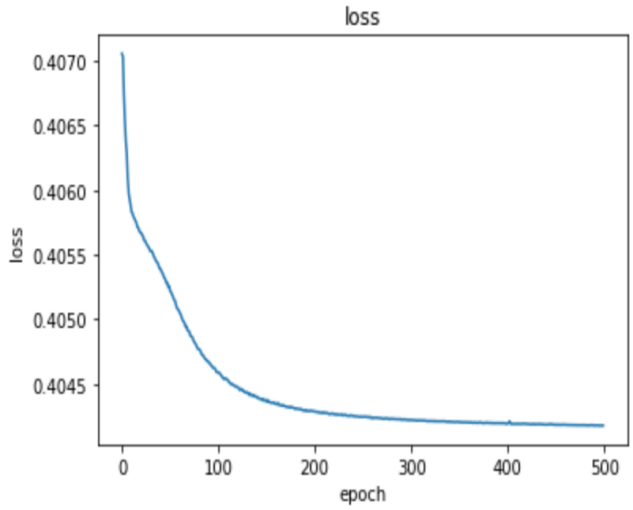


Figure 5.2: Training curve - BCE loss, CNN-ConvLSTM.

Lastly, the training curve for the pure ConvLSTM model (Figure 5.3) reaches convergence quickly and has no interruptions in learning over the first 100 epochs. The characteristic 'bump' shown in the training curves of the previous two models is not present for the ConvLSTM training, resulting in a convergence being reached approximately 300 epochs sooner than the pure CNN and about 100 epochs earlier than the hybrid model.

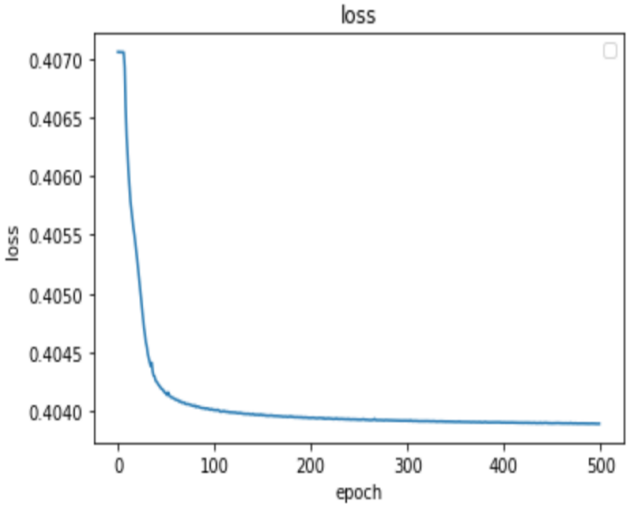


Figure 5.3: Training curve – BCE loss, pure ConvLSTM.

5.2 Model Performance Comparison

Table 5.1 summarizes the performance of the three models described in Section 4.2.1. The discussion refers to the table results by row (row 1 = "Model name").

	Model 1	Model 2	Model 3
Model name	CNN	CNN-convLSTM	convLSTM
Loss function	BCE	BCE	BCE
Learning rate	0.0001	0.0001	0.0001
Train set size	240	240	240
Test set size	60	59	59
Number of epochs	500	500	500
Test loss	0.4087	0.4062	0.4063
Train time (s)	1094	3594	8516
Test time (s)	0.44915	1.3373	2.4055
True Weeds	176	176	184
True Crops	318	287	256
False Weeds	7	21	18
False Crops	36	15	22
Total Weed Instances	282	280	280
Total Crop Instances	455	446	446

Table 5.1: Train/ test performance results across the three experimental architectures with fixed parameterization.

Crop/ Weed Stem Locations

Row 7 of Table 5.1 indicate pixel-wise BCE loss for each model. The vanilla CNN model provides the most accurate results for crop and weed stem locations. For the recurrent architectures, the hybrid model outperformed the pure ConvLSTM model in crop location accuracy, however, the pure ConvLSTM was superior to the hybrid model in weed location accuracy.

Train and Test Times

The model offline (train) and feed-forward (test) times of each model in seconds are shown above in Table 5.1 in rows 8 and 9. The CNN has shortest train and test times, and the pure ConvLSTM is has the

longest. Given the complexity of each model, it seems sensible that the CNN takes the least amount of time, both to train and test, and the pure ConvLSTM takes the longest.

The training time of each model is not of concern to the weeding task as training is performed offline before the robot is deployed. So for training, more compute resources may be used if necessary. The test time of a model must support the actual weeding application. Because the robot takes about two images per second as it drives, the rate required of a model to identify weed stem locations is approximately 2 Hz or two frames per second so that the robot may complete the weeding task before moving on to the next region in the crop row. In row 9 of the table each model meets this speed requirement. The pure ConvLSTM is the slowest of the three models with a test time of around 2.4 seconds for 60 test frames, or about 25 frame predictions per second.

Weed/ Crop Detection

A human-understandable evaluation metric is of considerable interest for the practical application of this study. Specifically, the weed/crop confusion matrix metric, detailed in Section 4.2.2. was devised to assess the number of weeds removed compared to the number of weeds encountered and the number of crops damaged in the process. The statistics true weeds, true crops, false weeds, and false crops, are provided in rows 10-13 of Table 5.1. Using these counts, and the total numbers of weed and crop instances (rows 14-15), more understandable metrics can be computed:

$$\text{Weed Detection Rate} = \frac{\text{True Weeds}}{\text{Total Weed Instances}} * 100,$$

$$\text{Crop Detection Rate} = \frac{\text{True Crops}}{\text{Total Crop Instances} * 100},$$

$$\text{False Weed Rate} = \frac{\text{False Weeds}}{\text{Total Crop Instances}} * 100,$$

$$\text{False Crop Rate} = \frac{\text{False Crops}}{\text{Total Weed Instances}} * 100.$$

Table 5.2 shows the weed and crop detection rates of the three models along with the false weed rates and false crop rates.

	CNN	CNN-ConvLSTM	ConvLSTM
Weed Detection Rate (%)	62.3	62.9	65.7
Crop Detection Rate (%)	69.9	64.3	57.4
False Weed Rate (%)	1.5	4.7	4.0
False Crop Rate (%)	12.8	5.4	7.9

Table 5.2: Model evaluation using true weeds, true crops, false weeds, false crops statistics.

The pure ConvLSTM model returned the most weeds at a rate of 65.7%. The next best weed detection rate was the hybrid model with 62.9%. The pure CNN had the lowest weed detection rate, 62.4%.

The trade-off for increased weed detection is an increase in the number of crops incorrectly classified as weeds or 'false weeds', resulting in crop damage. The pure CNN model returns 7 (1.5% of all crops) false weeds, the pure ConvLSTM model returns 18 (4.0% of all crops) false weeds, and the hybrid model returns the most false weeds, 21 (4.7% of all crops).

The two proposed architectures have lower crop detection rates than the pure CNN model, which correctly identifies the most crops (69.9%). The two temporal models ignore more crops with the hybrid model returning a crop detection rate of 64.3%, and the pure ConvLSTM with a crop detection rate of 57.4%.

Lastly, 'false crops' represents the number of weeds incorrectly classified as crops, which results in the weed failing to be removed. The CNN model incorrectly predicts the most with a value 36 (12.8% of all weeds) false crops while the two temporal models return considerably lower values for false crops. The pure ConvLSTM predicts 22 (7.9% of all weeds) false crops and the hybrid CNN-ConvLSTM model only misses 15 weeds (5.4%).

The two most important statistics from Table 5.2 are the weed detection rate and the false weed rate. The ideal model would detect 100% of the weeds, while finding 0% false weeds. The ConvLSTM model has the highest weed detection rate and second lowest percentage of false weeds. The CNN model has the lowest percentage of false weeds yet has the worst weed detection rate of the three models. The difference in weed detection rates (3.4 percentage points) is slightly larger than the difference between the false weeds rates (2.5 percentage points), so one might argue that the ConvLSTM is the best performing model.

Sample Results

For each run, a sequence of visual results (Figure 5.4) is output to assist in evaluating a model's ability to correctly identify locations of crop and weed stem origins in the input images. The goal of the study is for the predicted images in the second row to be identical to the true label images in the third row. Figure 5.4 shows the pure Conv-LSTM model can predict fairly reliably the locations of crop and weed stem emergence points, with a slight preference for weeds. Notice in the second row and second column there is an instance of a false crop, where a weed is incorrectly classified as a crop. Moreover, in columns 3, 4, and 5, the crop predictions in close clusters appear as larger green blobs where the stem points are merged or even ignored rather than making more precise crop distinctions. In the first column, there is an instance of a false weed, where a crop is incorrectly predicted as a weed. This type of error in is probably the most serious in a weeding application, as the robot would damage, if not remove, the crop.

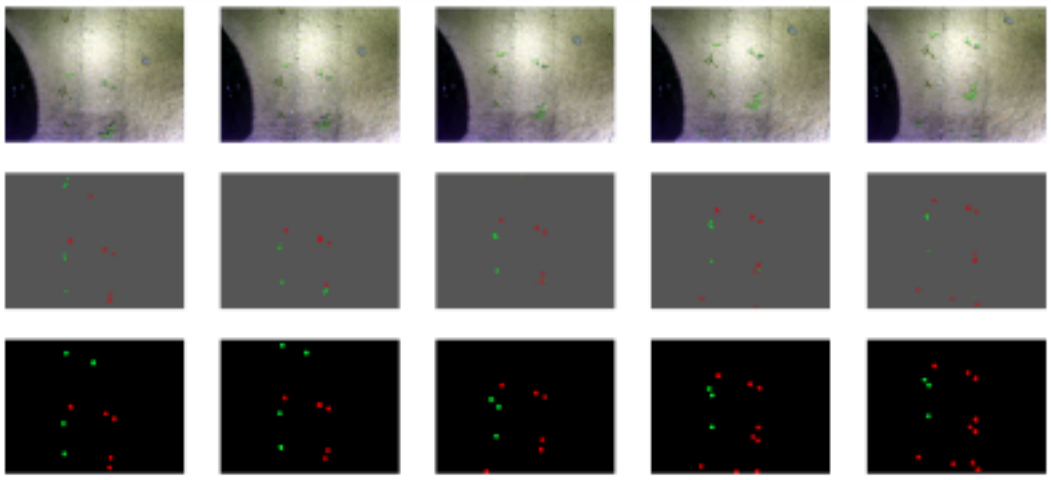


Figure 5.4: Sample image results. First row shows input images taken by the overhead camera on the robot, the second row shows images predicted by the ConvLSTM model and the third row shows target images for each input.

6 Conclusion

Three deep neural network architectures were designed then experimentally compared using a dataset of 300 overhead carrot crop/weed images to investigate the precision of weed-removal under fixed training and testing constraints. The utility of sequential images was studied with two of the three model architectures. One model, the pure CNN, assumes no temporal information, while the hybrid CNN-ConvLSTM and the pure LSTM models accept sequences of two images at a time. The hypothesis was that using temporal information could be advantageous in the robot-weeding task. The comparison of these models assumes identical train/test sets, similar resource allocations, and no special architectural tuning considerations including number of layers, convolutional filters, loss function, optimizer, and other associated hyper-parameter values. Both the hybrid and pure ConvLSTM models prove to be useful alternatives to the standard CNN model, with the pure ConvLSTM indicating potential as the best candidate for further experimentation. The test time of both of the temporal models is within the desired range of > 25 frames per second while naturally the pure CNN had the shortest times for training and testing. The pure Conv-LSTM, although it has the longest train time, converges quickly and provides the best overall weed detection and false crop rates. Interestingly, both the hybrid and pure ConvLSTM models have a tendency to focus on weed stem emergence points and ignore crop stems. Both temporal models appear to provide additional advantages in training behavior over the pure CNN and provide more visually appealing prediction images confirming the evaluation metric results. The pure CNN model returns the best false weed detection results by about 2-3% yet also provides the worst results on false crop identification with a rate of 13% compared to 5% for the hybrid model and 8% for the pure ConvLSTM model.

There are several possible directions for future research. Recurrent neural network architectures often assume that the “distance” between time-steps is uniform. In the robot weeding application, the assumption would imply that the robot moves the same physical distance between one image and the next. Future work could include incorporating GPS data of the robot’s location at each frame. Adding this

information may help the temporal models learn since the current imagery contains inconsistencies due to traversing uneven terrain or the robot stopping and recording multiple images of a single location.

Additional extensions to this study might include using a larger or more robust dataset. For example, one may use a dataset of images of the same crop in the same field on a longer run, images of the same crop on different fields, images of different crops, or images of crops at different stages of development. Training models on such extended data might improve generalization accuracy.

References

[1] “Nexus Webpage”. [Online]. Available: http://nexusrobotics.ca/. [Accessed: October 18, 2019].

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer Science+Business Media, LLC, 2006, pp. 3.

[3] S. Haykin, *Neural Networks and Learning Machines*. 3rd ed. Upper Saddle River, New Jersey: Pearson Education Inc., 2009.

[4] Y. Sewsynker-Sukai, F. Faloye, and E. B. G. Kana, “Artificial neural networks: an efficient tool for modelling and optimization of biofuel production (a mini review),” *Biotechnology & Biotechnological Equipment*, vol. 31, no. 2, p. 221-235, Dec. 2016. [Online] Available: https://doi.org/10.1080/13102818.2016.1269616.

[5] M. A. Nielson, “Using neural nets to recognize handwritten digits,” in *Neural Networks and Deep Learning*, Determination Press, 2015. [Online] Available: http://neuralnetworksanddeeplearning.com/chap1.html.

[6] “Deep Learning,” *Deep AI*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/deep-learning

[7] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning,” *arXiv:1811.03378 [cs.LG]*, Nov. 2018, arXiv:1811.03378. [Online]. Available: https://arxiv.org/abs/1811.03378.

[8] Qef, “The logistic sigmoid function,” *Wikimedia Commons*, July 2008. [Online]. Available: https://commons.wikimedia.org/wiki/File:Logistic-curve.svg.

[9] D. Liu, “A Practical Guide to ReLU,” *Medium*, Nov. 2017. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning*, vol. 15, no. 1,

p. 1929-1958, June 2014. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/2627435.2670313.

[11] J. Jeong, T. S. Yoon, and J. B. Park, "Towards a Meaningful 3D Map Using a 3D Lidar and a Camera," *Sensors*, vol. 18, no. 8, p. 2571, Aug. 2018. [Online]. Available: **https://doi.org/10.3390/s18082571**.

[12] ujjwalkarn, "An Intuitive Explanation of Convolutional Neural Networks," *The Data Science Blog*, Aug. 2016. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[13] D. Cornelisse, "An intuitive guide to Convolutional Neural Networks," *freeCodeCamp*, April 2018. [Online]. Available: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

[14] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285[stat.ML]*, Jan. 2018, arXiv:1603.07285. [Online]. Available: https://arxiv.org/abs/1603.07285.

[15] colah, "Understanding LSTM Networks," *colah's blog*, Aug. 27, 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[16] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *NIPS'15:Proceedings of the 28th International Conference on Neural Information Processing Systems, 7-12 Dec. 2015, Montreal, Quebec, Canada* [Online]. Available: https://dl.acm.org/doi/abs/10.5555/2969239.2969329.

[17] A. Xavier, "An introduction to ConvLSTM," *Medium*, Mar. 25, 2019. [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7.

[18] A. Dertat, "Applied Deep Learning – Part 3: Autoencoders," *towards data science*, Oct. 3, 2017.

[19] P. Lottes, J. Behley, N. Chebrolu, A. Milioto and C. Stachniss, "Joint Stem Detection and Crop-Weed Classification for Plant-Specific Treatment in Precision Farming," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1-5 Oct., 2018, Madrid, Spain* [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8593678

[20] S. Hochreiter, and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, Nov. 1997.

[21] A. Pfeuffer, K. Schulz, and K. Dietmayer, “Semantic Segmentation of Video Sequences with Convolutional LSTMs,” in *2019 IEEE Intelligent Vehicles Symposium (IV), 9-12 June 2019, Paris, France* [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8813852.

[22] F. Chollet, et al., *Keras*, 2015. [Online]. Available: https://keras.io/.

[23] D. P. Kingma and L. J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980[cs.LG]*, Dec. 2019, arXiv:1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[24] “Deep Learning VM Image,” *Google Cloud*. [Online]. Available: https://cloud.google.com/deep-learning-vm.

[25] S. Narkhede, “Understanding Confusion Matrix,” *towards data science*, May 9, 2018. [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.