MACHINE LEARNING WITH APPLICATIONS IN AGRICULTURE

ABIODUN UTHMAN ALLISON
MATRIC NO: 204603

A M.Sc. RESEARCH PROJECT SUBMITTED TO THE DEPARTMENT OF MATHEMATICS, FACULTY OF SCIENCE, UNIVERSITY OF IBADAN, IBADAN, NIGERIA.

SUPERVISOR: PROF. G.O.S. EKHAGUERE

FEBRUARY, 2021

Overview

Introduction I

Agriculture could be referred to as the production, processing, promotion and distribution of agricultural products. It plays a critical role in the economy of developing countries. Besides supplying food and raw materials, agriculture also provides the populace with job opportunities. The Food and Agriculture Organization of the United Nations (FAO) estimates that pests and diseases lead to the loss of $20 - 40\%$ of global food production, constituting a threat to food security (Food and Agriculture Organization of the United Nation, International Plant Protection Convention, 2017).
Machine learning is the scientific study of algorithms and mathematical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is a branch of Artificial Intelligence (AI) focused on the premise that, with minimal human interaction, systems can learn from data, recognize trends and make decisions. In supervised machine learning, a training set of D

Introduction II

input vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_D\}$ is used to tune the parameters of an adaptive model, along with their corresponding target vectors $\boldsymbol{y}$, in order to accurately predict vector $\hat{\boldsymbol{y}}$. We focus on a subset of supervised learning, called classification, where the aim is to assign each input vector to one of a finite number of discrete categories.
Computer vision, and object recognition in particular, has made tremendous advances in the past few years. In 2012, a large, deep convolutional neural network achieved a top-5 test error (the fraction of test images for which the correct label is not among the five labels considered most probable by the model) rate of 15.3 %, at the Large Scale Visual Recognition Challenge (ILSVRC) 2012 image classification competition for the classification of 15 million images into 22 thousand possible categories Krizhevsky et al. (2012).

Literature Review I

The idea of connecting units to local receptive fields on the input goes back to the perceptron in 1958 Rosenblatt (1958). Although the perceptron initially seemed promising, the constraint that similar input patterns lead to similar outputs lead to an inability of the system to learn certain mappings from input to output, which proved that perceptrons could not be trained to recognise many classes of patterns. Minsky & Papert (1969) pointed out that if we have the right connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output through these hidden units. LeNet-5, a pioneering 7-level convolutional network LeCun et al. (1998) that classifies digits, was applied by several banks to recognize hand-written numbers on cheques digitized in 32×32 pixel images. This technique was constrained by the availability of computing resources since the ability to process higher resolution images requires larger and more layers of convolutional neural networks. With

Literature Review II

advancements in computing power driven by the advent of graphical processing units, convolutional neural networks were adopted more with great precision in the 2000s.
Different convolutional neural network (CNN) architectures has been applied in agriculture-analysis and phenotyping realm to solve complex problems. Measuring stalk count and stalk width of Sorghum plants Baweja et al. (2018); segmentation and counting of aphid nymphs Chen et al. (2018); crop/weed segmentation Sa et al. (2018); plant disease detection and diagnosis Ferentinos (2018); corn silage kernel processing Rasmussen & Moeslund (2019); pod-counting and flower detection in soybean crop Z. Zhang (2019); classifying and counting insect pests in soybeans Tetila et al. (2019); detecting weeds in Bermuda grass Yu et al. (2019); extracting features of tea plant diseases from images Chen et al. (2019).

Logistic Regression I

Logistic regression is an algorithm for for binary classification - in which the goal is to learn a classifier that can input data represented by feature vector $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and present a label $y \in \{0, 1\}$, where n_x is the dimension of the vector $\boldsymbol{x}$. To transform the feature vector $\boldsymbol{x}$ into a scalar we input $\boldsymbol{x}$ to the linear predictive model with parameters, weight $\boldsymbol{w} \in \mathbb{R}^{n_x}$, and bias $b \in \mathbb{R}$

$$z = \boldsymbol{w}^T \boldsymbol{x} + b \tag{1}$$

is used. The probabilistic sigmoid function $a = \sigma(z) = \dfrac{1}{1 + e^{-z}}$ is then used to estimate the class of a new input. For a given vector $\boldsymbol{x}$, the prediction is estimated thus

$$\text{Prediction} = \begin{cases} 0, & \text{if } a < 0.5 \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

Logistic Regression II

Given feature vector $\boldsymbol{x}$, the aim is to output prediction $\hat{y}$:

$$\hat{y} = P(y = 1|\boldsymbol{x}) = a = \frac{1}{1 + e^{-z}} \tag{3}$$

To find optimal values for parameters $\boldsymbol{w}$ and b, we minimise the error function which describes the error created by any arbitrary values of $\boldsymbol{w}$ and b. Since the predicted value belongs to either class $y = 0$ or $y = 1$, the output belongs to the Bernoulli distribution:

$$P(Y = y) = a^y (1 - a)^{1-y} = \begin{cases} a & \text{if} \quad y = 1 \\ 1 - a & \text{if} \quad y = 0 \end{cases} \tag{4}$$

Hence, we would to minimize the function L over $\boldsymbol{w}$ and b

$$\mathrm{L}(a, y) = -[y \log(a) + (1 - y) \log(1 - a)] \tag{5}$$

Logistic Regression III

L is the loss function which measures how good the output a is when given the true label y on a single training example $(\boldsymbol{x}, y)$.
Similarly, the overall loss function for the entire training set is given by

$$\begin{aligned} \mathrm{J}(\boldsymbol{w}, b) &= -\frac{1}{m}\sum_{i=1}^{m}\left[\left(y^{(i)}\log\left(a^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - a^{(i)}\right)\right]\right. \\ &= \frac{1}{m}\sum_{i=1}^{m}\mathrm{L}\left(a^{(i)}, y^{(i)}\right) \end{aligned} \tag{6}$$

Equation (6) is called the cost function which measures how well parameters $\boldsymbol{w}$ and b are doing on the entire training set $(\boldsymbol{X}, \boldsymbol{y})$. A global minimum is guaranteed since $\mathrm{J}(\boldsymbol{w}, b)$ is convex.

Gradient Descent I

To find the parameters $\boldsymbol{w},\ b$ that minimizes the cost function $\mathrm{J}(\boldsymbol{w},\ b)$, we use the gradient descent algorithm; which is an optimisation algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. Choosing a suitable learning rate $\alpha > 0$, at time step k gradient descent performs the iterative method:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} - \alpha \nabla f(\boldsymbol{x}^{k}) \tag{7}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function to be minimised, and $\boldsymbol{x} \in \mathbb{R}^n$

Gradient Descent II

Theorem (Bottou et al. (2018))

Suppose the function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and twice continuously differentiable, and it is L-smooth (i.e. its gradient is Lipschitz continuous with constant $L > 0$):

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{8}$$

for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$. Gradient descent for k iterations with fixed learning rate $\alpha \leq 1/L$ satisfies

$$f\left(\boldsymbol{x}^{(k)}\right) - f\left(\boldsymbol{x}^*\right) \leq \frac{\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\right\|_2^2}{2\alpha k} \tag{9}$$

where $f\left(\boldsymbol{x}^*\right)$ is the optimal value of $f(\boldsymbol{x})$.

Gradient Descent III

Intuitively, this means that gradient descent converges sublinearly with rate $\mathcal{O}(1/k)$.

Using, gradient descent, the computing cost for each independent variable iteration grows linearly with the dataset size. Therefore, when the model training data instance is large, the cost of gradient descent for each iteration will be very high.

Stochastic gradient descent (SGD) reduces computational cost at each iteration by uniform sampling an index $i_k \in \{1, \ldots, m\}$ for data instances at random, and compute the gradient $\nabla f_{i_k}(\boldsymbol{x})$ to update $\boldsymbol{x}$:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha_k \cdot \nabla f_{i_k}\left(\boldsymbol{x}^k\right), \quad k = 1, 2, 3, \ldots \tag{10}$$

Gradient Descent IV

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k and α_k is the learning rate at iteration k. Stochastic gradient descent uses an unbiased estimate of the gradient $\nabla f(\boldsymbol{x})$ at each step, i.e,

$$\mathbb{E}[\nabla f_{i_k}(\boldsymbol{x})] = \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \tag{11}$$

A popular method used in machine learning is to compute the gradient against more than one training example, called a minibatch, at each step. This method improves computational efficiency compared to stochastic gradient descent, because the gradient can be calculated using vectorisation libraries rather than computing each step separately. In minibatch stochastic gradient descent, we choose a random subset $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, repeat:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i\left(\boldsymbol{x}^k\right), \quad k = 1, 2, 3, \ldots \tag{12}$$

Regularisation

Regularisation is used to prevent a model from overfitting - where the model would memorise the data and perform poorly on the hold out test data. It involves modulating the amount of information the model can store by reducing the number of learnable parameters in the model. Regularization is done by adding to the cost function of the model a cost associated with having large weights. With L^2 regularisation, the cost added is the square L^2 norm of the weights:

$$\mathrm{J}(\boldsymbol{w}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[\left(y^{(i)}\log\left(a^{(i)}\right)+\left(1-y^{(i)}\right)\log\left(1-a^{(i)}\right)\right)\right]+\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \tag{13}$$

where $\lambda \geq 0$ is an hyperparameter called the regularisation constant, and it governs the amount of regularisation.

Neural Networks I

Logistic regression and similar vanilla machine learning models discussed earlier only work effectively in cases where a hyperplane can distinguish the line between class $y = 0$ and class $y = 1$. But there are situations in which the data are not well separated by a linear classifier, neural networks are ideal to solve these problems.
A basic neural network model can be described a series of functional transformations. First we construct linear combinations of the input variables $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_D]$ in the form

$$\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]} \tag{14}$$

the superscript [1] indicates that the corresponding parameters $\boldsymbol{W}$ (weights) and $\boldsymbol{b}$ (biases) are in the first 'layer' of the network. Each of the linear combinations is then transformed using a differentiable, non-linear activation function $g(.)$ to give

$$\boldsymbol{a}^{[1]} = g^{[1]}(\boldsymbol{z}^{[1]}) \tag{15}$$

Neural Networks II

where $\boldsymbol{a}^{[l]}$ represents the activations at layer l. These quantities in the context of neural networks, are called hidden units which forms the hidden layer. These values are again linearly combined to give output unit activations

$$\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]} \tag{16}$$

This transformation corresponds to the second layer of the network. Finally, the output unit activations are transformed using an appropriate activation function to give a set of network output $\hat{y}$. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables.

Neural Networks III

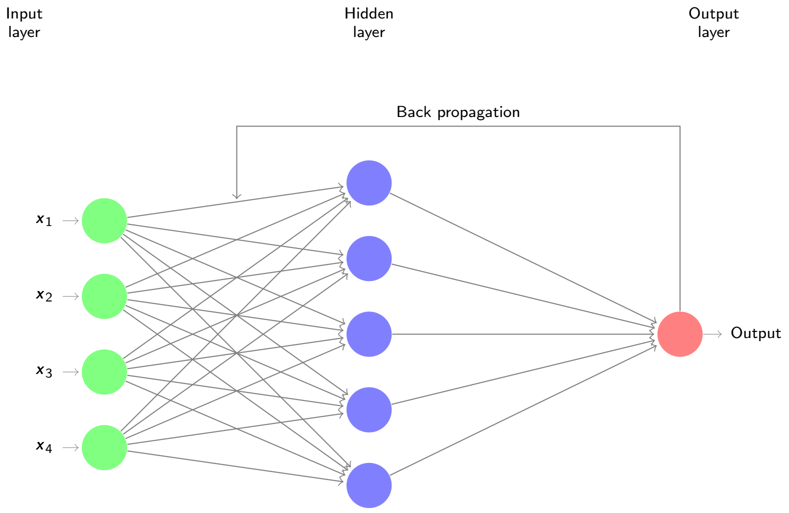


Figure 1: Network diagram for one hidden layer neural network

Neural Networks IV

Let $n^{[i]}$ be the number of nodes at layer i. The dimension of the parameters $\boldsymbol{W}^{[1]}$, $\boldsymbol{b}^{[1]}$, $\boldsymbol{W}^{[2]}$ and $\boldsymbol{b}^{[2]}$ depends on $n^{[i]}$ at different layers. The dimensions are

- $\boldsymbol{W}^{[1]}$ - $(n^{[1]}, n^{[0]})$
- $\boldsymbol{W}^{[2]}$ - $(n^{[2]}, n^{[1]})$
- $\boldsymbol{b}^{[1]}$ - $(n^{[1]}, 1)$
- $\boldsymbol{b}^{[2]}$ - $(n^{[2]}, 1)$

The mostly widely used activation function in hidden layers is the Rectified Linear Activation Function (ReLU), it is a simple calculation that returns the value provided as input directly, or the value 0 if the input is 0 or negative. The function is given by:

$$h(z) = \max(0, z) \tag{17}$$

Neural Networks V

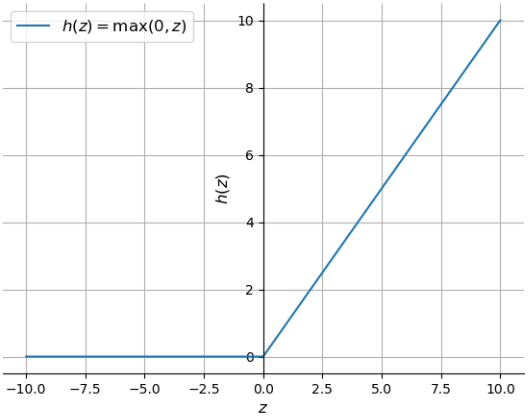


Figure 2: ReLU function

Neural Networks VI

Softmax function is widely used as the activation function of the output layer for classification classes with more than two classes. it would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs. This essentially gives the probability of the input being in a particular class. The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula:

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \boldsymbol{z} = (z_1, \dots, z_K) \in \mathbb{R}^K \tag{18}$$

After applying softmax, each component would be in the interval $(0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

Backpropagation

Neural networks are trained using backpropagation. Backpropagation refers to the method of calculating the gradient of neural network parameters using chain rule. The core insight is that the objective function derivative with respect to a module's input can be computed by working backwards from the gradient with respect to that module's output (or the corresponding module's input). The backpropagation equation can be repeatedly applied to propagate gradients through all modules, starting from the output at the top (where the network produces its prediction) all the way down (where the external input is fed).

Convolutional Layers I

Convolutional neural networks (CNNs) are similar to neural networks in that they consist of neurons which optimize themselves through learning, each neuron will still receive an input and perform an operation such as a scalar product followed by a non-linear function. They are designed to process data that are in form of multiple arrays, such as a color image consisting of three 2D arrays containing pixel intensities in three color channels; red, green and blue (RGB). CNNs are mostly made up of stacks convolutional layers followed by non-linearity layers, then pooling layers and fully connected layers with softmax at the output layer.
In a convolutional layer, an input array and a convolution kernel array are combined to produce an array through a cross-correlation operation, a scalar bias is added to the array to produce an output, followed by a non-linear activation function (mostly ReLU) being applied to the output array.

Convolutional Layers II

Cross-correlation is the measure of the overlap between two functions $f, g : \mathbb{R}^n \to \mathbb{R}$ when one of the functions is shifted by $\boldsymbol{x}$, it is given by

$$(\boldsymbol{I} \star \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{I}(i+m, j+n)\boldsymbol{K}(m,n) \qquad (19)$$

In a two-dimensional cross-correlation operation, a convolutional kernel (or filter) is positioned at the at the top-left corner of the input array and we slide it across the input array, both from left to right and top to bottom. When the kernel slides to a certain position, the input subarray contained in that position and the kernel array are multiplied elementwise and the resulting array is summed up yielding a single scalar value. This result if precisely the value of the output array at the corresponding location. Since the kernel has a width greater than one, and we can only compute the cross-correlation for locations where the kernel fits wholly within the image, the output size is given by the input size $H \times W$ minus the size of

Convolutional Layers III

the convolutional kernel $h \times w$ via $(H - h + 1) \times (W - w + 1)$, since we need enough space to shift the convolutional kernel across the image. Convolution layers are also useful for local feature detection, since a feature that is useful in one part of the image is probably useful in another part of the image.

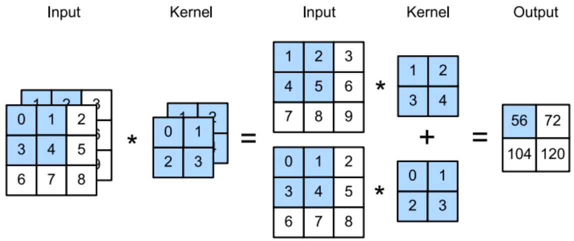


Figure 3: Cross-correlation computation with 2 input channels, A. Zhang et al. (2020)

Padding and Stride I

Since kernels generally have width and height greater than 1, after applying many successive convolutions, we tend to wind up with outputs that are considerably smaller than our input. To keep the shape of the input, we add additional filler pixels around the boundary of our input image, thereby increasing the image's effective size. Typically, we set the values of the extra pixels to zero.

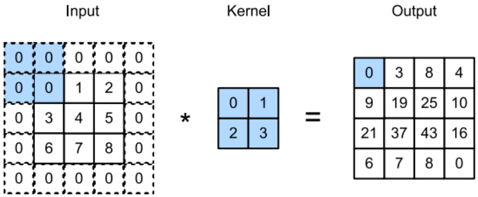


Figure 4: Two-dimensional cross-correlation with padding, A. Zhang et al. (2020)

Padding and Stride II

When computing the cross-correlation, we begin with the convolution window at the top-left corner of the input tensor, and then slide it both down and to the right over all locations, defaulting to sliding one element at a time in previous examples. However, often we shift our window more than one element at a time, skipping the intermediate positions, either for computational effectiveness or because we want to downsample. We refer to the number of rows and columns traversed per slide as the stride. In practice, we apply the same amount of stride vertically and horizontally. Hence, given an input image with size $H \times H$, a convolution kernel with size $h \times h$, applied at stride s with pad p, the output would be of height and width

$$\left\lfloor \frac{H + 2p - h}{s} + 1 \right\rfloor \tag{20}$$

Pooling Layers I

Like convolutional layers, pooling operators consist of a fixed-shaped window that slides across all regions of the input according to its stride, computing a single output for each position crossed by the window. Pooling layers serves the dual purpose of merging semantically similar features into one and of spatially downsampling representations. However, unlike the cross-correlation operation of inputs and kernels in the convolutional layer, the pooling layer does not include any parameters, i.e., there is no kernel. Instead, pooling operators are deterministic, usually measuring either the maximum (max pooling) or the average value (average pooling) of the elements in the pooling window.

Pooling Layers II

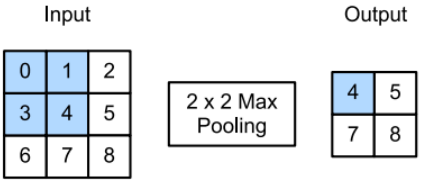


Figure 5: Maximum pooling with a pooling window shape of 2×2, A. Zhang et al. (2020)

Adam Optimiser I

We introduced minibatch gradient descent earlier, which improved on the convergence rate of gradient descent, where we made parameter updates using a minibatch with size b, instead of the whole dataset at each iteration. In practice, common loss functions for neural networks are highly non-convex, minimising the loss becomes a challenge using SGD algorithms due to saddle points in the function Dauphin et al. (2014), saddle points are usually surrounded by a plateau of the same error, which makes it notoriously difficult for gradient descent to escape, as the gradient is close to zero in all dimensions. Momentum Qian (1999) is a method that helps speed gradient descent in the relevant direction and dampens oscillations, by adding a fraction ρ of the update vector of the previous time step to the current update vector.
Adaptive Moment Estimation (Adam) Kingma & Ba (2014) computes individual adaptive learning rates for different parameters from estimate of the first and second moments of the gradients. Adam stores an

Adam Optimiser II

exponentially decaying average of past squared gradients $\boldsymbol{v}^k$, and an exponentially decaying average of past gradients $\boldsymbol{m}^k$, similar to momentum. The decaying averages of past and past squared gradients $\boldsymbol{m}^k$ and $\boldsymbol{v}^k$ respectively are calculated as follows:

$$\begin{aligned} \boldsymbol{m}^k &= \beta_1 \boldsymbol{m}^{k-1} + (1-\beta_1)\boldsymbol{g}^k \\ \boldsymbol{v}^k &= \beta_2 \boldsymbol{v}^{k-1} + (1-\beta_2)(\boldsymbol{g}^k)^2 \end{aligned} \tag{21}$$

$\boldsymbol{m}^k$ and $\boldsymbol{v}^k$ are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. Since $\boldsymbol{m}^k$ and $\boldsymbol{v}^k$ are initialized as vectors of 0's, they become biased towards zero, especially during the initial time steps, and especially when the decay rates

Adam Optimiser III

are small, i.e., β_1 and $\beta_2$are close to 1). Bias-corrected first and second moment estimates are calculated to tackle this problem:

$$\hat{\boldsymbol{m}}^k = \frac{\boldsymbol{m}^k}{1-\beta_1^k} \qquad \hat{\boldsymbol{v}}^k = \frac{\boldsymbol{v}^k}{1-\beta_2^k} \tag{22}$$

The Adam update rule is given as:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \frac{\alpha}{\sqrt{\hat{\boldsymbol{v}}^k} + \epsilon}\,\hat{\boldsymbol{m}}^k \tag{23}$$

Adam works well in practice and compares favourably to other adaptive learning-method algorithms.

Transfer Learning I

To improve training accuracy and speed up training time we adopt transfer learning. Transfer learning can be described as the improvement of learning of a new task by the transition of information from a previously learned similar task, model weights from pre-trained models created for large-scale image-classification datasets, such as ImageNet are reused to solve similar image-classification tasks. VGG16 and ResNext-50 were used as the pre-trained models in this project.

VGG16 I

VGG16 made improvement over AlexNet Krizhevsky et al. (2012) by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer respectively) with stacks of 3×3 kernel-sized filters. Each stack of 3×3 convolutional layer with stride 1, pad 1 and different depths is followed by 2×2 max pooling layers applied at stride 2, three fully-connected (FC) layers follow the final max pool layer; the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class), the final layer is the softmax layer and ReLU is applied to all hidden layers (Figure 6). It was shown that the added depth and smaller filters is beneficial for classification accuracy.

VGG16 II

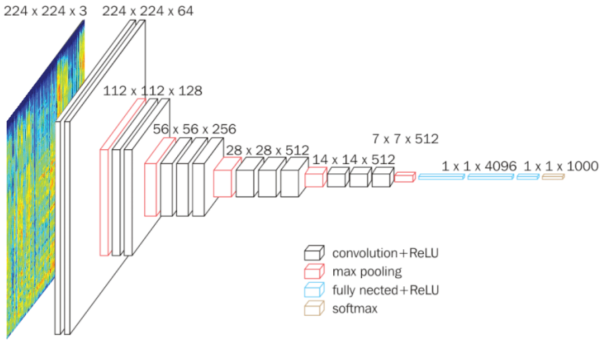
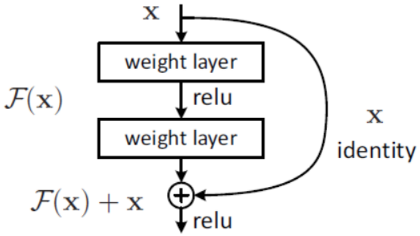


Figure 6: VGG16 model architecture, Das et al. (2019)

ResNext I

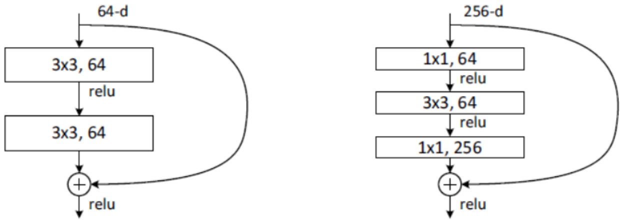
Deep networks are susceptible to vanishing/exploding gradients, making it harder to optimise compared to shallow networks. During backpropagation, deriving partial derivative of the error function with respect to the current weight in each iteration of training, has the effect of multiplying n of these small/large numbers to compute gradients of intermediate layers in an n-layer network.
Residual learning introduced in ResNet architecture He et al. (2015) tackles the problem stated above by using network layers to fit a residual mapping using skip connections instead of directly trying to fit a desired underlying mapping, i.e., for the output $H(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}$, use weight layers to learn residual mapping: $F(\boldsymbol{x}) = H(\boldsymbol{x}) - \boldsymbol{x}$ (Figure 7a) as we move to the next layer, instead of desired function $H(\boldsymbol{x})$ directly.

ResNext II


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(a) A residual block

ResNext III



(b) A bottleneck building block for ResNet-50/101/152

Figure 7: Residual learning blocks, He et al. (2015)

ResNext IV

ResNet architecture consists of a 7×7 convolutional layer, 2×2 pooling layer, stacks of residual blocks with two 3×3 convolutional layers, number of filters are double after each block, and the layers are down-sampled spatially using stride 2 and a global average pooling layer after the last convolutional layer.
ResNext Xie et al. (2017) tried to improve wide ResNet Zagoruyko & Komodakis (2017) by increasing width of the residual blocks through multiple parallel pathways. Multiple residual blocks with bottlenecks are stacked in parallel with d internal dimension for each path and c total number of pathways, called cardinality (Figure 8). The architecture for ResNext-50 is the same as ResNet-50 but with the extra parallel pathways in the residual blocks, it achieved a 6.6% top-5 error rate when trained on the ImageNet dataset with $1,000$ classes compared to a 8.2% error rate for ResNet-50 on the same dataset.

ResNext V

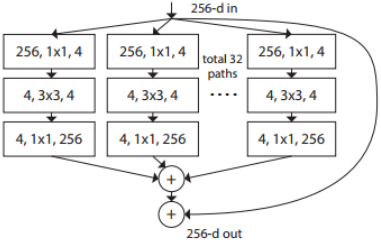


Figure 8: A block of ResNeXt with cardinality $=$ 32, Xie et al. (2017)

Application I

We adopted three different trained CNN models - a base model, VGG16 model and ResNext-50 to detect plant disease using the PlantVillage dataset (https://github.com/spMohanty/PlantVillage-Dataset) (Figure 9) and compared results of each model. This dataset comprises healthy or diseased leaf images classified into 38 labels, 42, 754 images, 14 different crop species and 20 different diseases including healthy ones. raining the models was performed using Fastai deep learning library Howard & Gugger (2020) and Pytorch deep leaning library Paszke et al. (2019) on Python 3.7, running on Google colab (https://colab.research.google.com) with the provided free GPU from colab, it took about 3 to 6 minutes to train the models per epoch. The network was initialized with random weights, using a categorical cross-entropy loss metric, network weights were optimized using the Adam optimization algorithm with a learning rate of 0.01, a set of 64 images with a size of 224×224 were fed to the network as a batch per iteration.

Application II

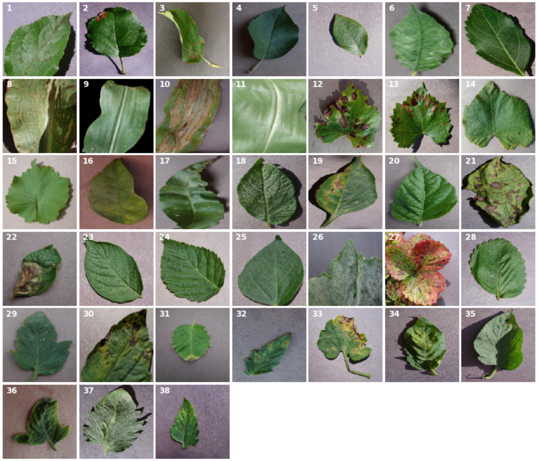


Figure 9: The PlantVillage image dataset

Application III

The base model consists of the input image, six convolution layers, two fully connected layers and the output layer (Figure 10). Each convolution layer was of kernel size 3×3, applied at stride 1 and padding 1, the number of kernels were 16, 32, 64 , 128 , 256 , 512 respectively for each

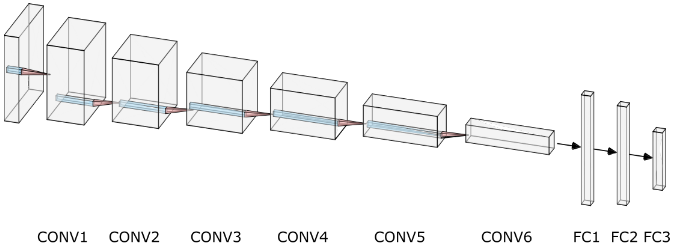


Figure 10: Simplified base model architecture

Application IV

preceding convolution layer. To reduce dimensions of output features after each convolution layer, max pooling layers of kernel size 2×2, applied at stride 2 and zero padding was added after each convolution layer. Output image from the final max pool layer was flattened to a 1D array and fed into fully connected layers with size 1024, , 512, and , 38 respectively, batch norm and dropout with rate 0.5 was added to the first two fully connected layers. ReLU activation was used for all layers except the output layer which uses softmax to compute probabilities of an image belonging to a particular class. The model contained $6,841,766$ trainable parameters, batch size 64 and learning rate 0.01 was used during training.

Results I

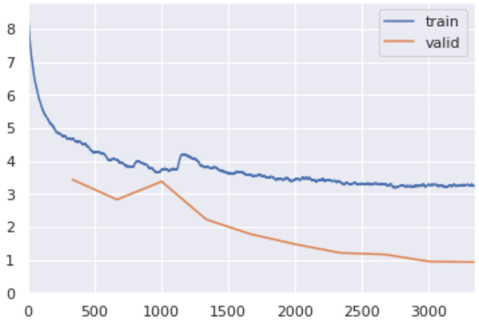
Results gotten on validation data are compared for the three different models studied in this research; a base CNN model, a VGG16 model and a ResNext-50 $32 \times 4d$ model. Performance measures such as accuracy - which is the ratio of correctly predicted observation to the total observations, precision - which is the ratio of correctly predicted positive observations to the total predicted positive observations, recall - which gives the number of labelled examples from all the images belonging to a class, and f1 score - which is the weighted average of precision and recall. Visualisation of the model was carried out using the cross-entropy (or logistic) loss defined in Equation 6, loss of the model decreases as training is done (Figure 11). ResNext-50 has the best performance since training loss and validation loss converges, with minimal space in between both curves as the model trains, the base model performs worst.

Results II

	Accuracy	Top-3 Accuracy	Precision	Recall	F1 score
Base Model	0.9277	0.9923	0.9329	0.9277	0.9277
VGG16	0.9821	0.9995	0.9853	0.9821	0.9819
ResNext-50	0.9823	1.00	0.9850	0.9823	0.9822

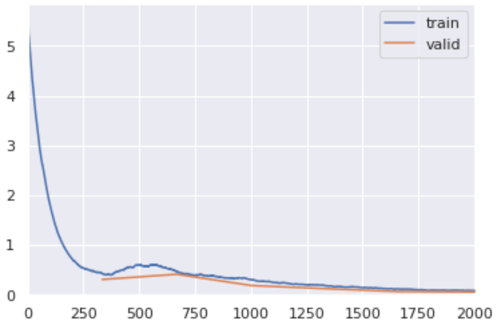
Table 1: Comparing different evaluation metrics for the models studied on validation data

Results III



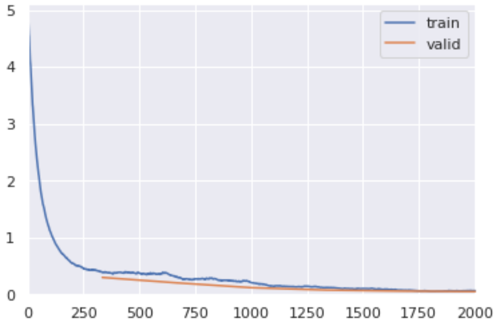
(a) Base model training loss

Results IV



(b) VGG16 training loss

Results V



(c) ResNext-50 training loss

Figure 11: Training loss for different models, the horizontal axes denotes number of epochs, while the vertical axes denotes loss

References I

Baweja, H. S., Parhar, T., Mirbod, O., & Nuske, S. (2018). Stalknet: A deep learning pipeline for high-throughput measurement of plant stalk count and stalk width. In M. Hutter & R. Siegwart (Eds.), *Field and service robotics* (pp. 271–284). Cham: Springer International Publishing.

Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *Siam Review*, *60*(2), 223–311.

Chen, J., Fan, Y., Wang, T., Zhang, C., Qiu, Z., & He, Y. (2018). Automatic segmentation and counting of aphid nymphs on leaves using convolutional neural networks. *Agronomy*, *8*(8), 129.

Chen, J., Liu, Q., & Gao, L. (2019). Visual tea leaf disease recognition using a convolutional neural network model. *Symmetry*, *11*(3), 343.

References II

Das, P., Acharjee, A., & Marium-E-Jannat. (2019). Double coated vgg16 architecture: An enhanced approach for genre classification of spectrographic representation of musical pieces. *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, 1-5.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 27, pp. 2933–2941). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2014/file/17e23e50bedc63b4095e3d8204ce063b-Paper.pdf

Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, *145*, 311–318.

References III

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

Howard, J., & Gugger, S. (2020, Feb). Fastai: A layered api for deep learning. *Information*, *11*(2), 108. Retrieved from `http://dx.doi.org/10.3390/info11020108` doi: 10.3390/info11020108

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012, 01). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*. doi: 10.1145/3065386

References IV

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the ieee* (Vol. 86, pp. 2278–2324). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *The Official Journal of the International Neural Network Society*, *12*(1), 145–151.

References V

Rasmussen, C. B., & Moeslund, T. B. (2019). Maize silage kernel fragment estimation using deep learning-based object recognition in non-separated kernel/stover rgb images. *Sensors*, *19*(16), 3506.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65–386.

Sa, I., Popović, M., Khanna, R., Chen, Z., Lottes, P., Liebisch, F., . . . Siegwart, R. (2018). Weedmap: a large-scale semantic weed mapping framework using aerial multispectral imaging and deep neural network for precision farming. *Remote Sensing*, *10*(9), 1423.

Tetila, E. C., Machado, B. B., Menezes, G. V., de Souza Belete, N. A., Astolfi, G., & Pistori, H. (2019). A deep-learning approach for automatic counting of soybean insect pests. *IEEE Geoscience and Remote Sensing Letters*.

References VI

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1492–1500).

Yu, J., Sharpe, S. M., Schumann, A. W., & Boyd, N. S. (2019). Deep learning for image-based weed detection in turfgrass. *European journal of agronomy*, *104*, 78–84.

Zagoruyko, S., & Komodakis, N. (2017). *Wide residual networks.*

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). *Dive into deep learning*. (`https://d2l.ai`)

Zhang, Z. (2019). Deep learning for field-based automated high-throughput plant phenotyping. *Graduate Theses and Dissertations*, *17628*.

THANK YOU FOR LISTENING