FACULTY OF SCIENCE & TECHNOLOGY

BSc (Hons) [Degree Title]

May 2021

What's Wrong With My Crop? Using Convolutional Neural Networks to Detect Crop Defects

by

Ryan Syme

Faculty of Science & Technology

Department of Computing and Informatics

Final Year Project

Abstract

[The text within the square brackets must be deleted along with the square brackets when finalising your own abstract.

The abstract for an undergraduate dissertation should be between 200 - 350 words.

Arial, Normal, 11pt with 1.2 or 1.5 line spacing should be used. The text in this part has 1.5 line spacing.

An abstract is a brief, accurate and comprehensive summary of the entire dissertation. It is the first thing to be read by your examiners to help them know the brief content of the dissertation. It also serves as a “sales pitch” to form the first impression of your work.

A good abstract should be accurate, self-contained, concise, specific and clear. A quick way to assess the quality of your abstract is to check whether it answers the questions why, how, what and so what.

Researching the efficacy of using CNN’s (Convolutional neural networks to identify crop defects) and creating a suitable platform for users to interact with the network.

It is easier to write the Abstract the last.]

Dissertation Declaration

[The text within the square brackets must be deleted along with the square brackets when finalising your declaration.

Note if your project is CONFIDENTIAL because of your client, you will need to adapt this declaration based on the agreement between you and your client accordingly. Do not forget to state the name of your client clearly. You must contact and inform Project Coordinator if your project is CONFIDENTIAL.]

I agree that, should the University wish to retain it for reference purposes, a copy of my dissertation may be held by Bournemouth University normally for a period of 3 academic years. I understand that once the retention period has expired my dissertation will be destroyed.

Confidentiality

I confirm that this dissertation does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or sex life has been anonymised unless permission has been granted for its publication from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this dissertation may be made available as the result of a request for information under the Freedom of Information Act.

Signed: ____

Name: [Your name]

Date: [Date of signing this declaration]

Programme: [Your degree title]

Original Work Declaration

This dissertation and the project that it is based on are my own work, except where stated, in accordance with University regulations.

Signed: ____

Name: [Your name]

Date: [Date of signing this declaration]

Acknowledgements

[The text within the square brackets must be deleted along with the square brackets when finalising your own acknowledgements.

Arial, Normal, 11pt with 1.2 or 1.5 line spacing should be used. The text in this part has 1.5 line spacing.

This is your opportunity to mention individuals who have been particularly helpful. Reading the acknowledgements in the past dissertations in the project library will give you an idea of the ways in which different kinds of help have been appreciated and mentioned.]

Contents

List of Figures

List of Tables

Chapter 1 - Background and Lit Review

1.1 Context

the application area / industry / domain

1.2 Technological Aspects

lorem ipsum

Chapter 2 - Introduction

2.1 Context

With the increased availability of smartphones Sta (2021), digital cameras Ima (2021) and Internet access Wik (2021) Glo (2021). Coupled with the increased interest in home food cultivation Goo (2021) and the large number of people reliant on food grown in smallholdings Walpole and Hutton (2013). The ability to identify defects with crops using technology has potential to be impactful to many people.

2.2 Literature Review

Firstly, existing research regarding image classification of plants and plant diseases will be explored. Secondly, an analysis of CNN architecture development.

2.2.1 General Software Development Practice

Software, often complex by nature, benefits from being created with a set of best practices in mind. A key principle one has gleaned from the Pragmatic Programmer [CITE PRAG PROG] (a key reference guide that has helped one's understanding of best practice) is that of developing for orthogonality. This means creating software from sub modules that operate as independently as possible from the whole. Meaning that changes to part of the system do not cascasde to unexpected effects elesewhere in the sytem. In practice this involves reducing, or if possible, entirely eliminating the usage of global variables. As this can make it difficult to trace the source of bugs when one does not know which part of the system is modifying the global. It also forces any changes to the modules that interact with the global to be considering the knock on effects it may have to other modules. Additionally creating orthogonal code involves making objects store as little information about their clients as possible, as this allows for the client to be changed more readily without then needing to alter the internal logic of anything it is interfacing with. How to solve it [Site G polya] Also reinforces the modular aprach to development by encouriging the proplem solver to break apart large problems in to smaller more manageable problems that one knows how to solve, eventually, reaching a point where the known unknown becomes a smaller and smaller peice of the problem until the remaining steps of the solution are trivial or at the very least clearly defined.

Another useful development practice outlined in The Pragmatic Programmer, is that of 'tracer-code'. This is much the same idea as opting to acheive a Minimum Viable Product (MVP) to provide a framework to build additional features from. For instance creating simple client and server applications that are successffully interfacing with one another to send and respond with simple data, can then be expanded to include the capacity to send more complex data and for the server to perform ever more complex operations on said data. Once a basic outline of the software is constructed it can also be extended to include, better response to erronus data and provide a greater number of features.

Development of software is iterative and cyclical. Involving the creation of features to a set of requirments, and then subsequetly refactoring their implementation to better align with best practices. Some reasons code may need to be refactored as covered in [Cite Pragmatic Programmer] are; Quote "Duplication", "Nonorthogonal design", "Outdated Knowledge", "Performance". As it is the case with software that it can be configured in infinitely many ways to acheive the same outcome. It is inevitable that sometimes features will be implemented in a way that breaches best practices. Or in a way that is not conducive to the overall 'health' of the codebase. A useful analogy shown in the book is to think of the codebase as a garden, that will grow and change over time, with refactoring being analagous to weeding and pruning.

2.2.2 Disease Detection Usage Of CNNs

In 2009 a study was conducted using deep learning to identify three different disease classes on rice plants. The results showed over 70% classification accuracy on 50 sample images. The method employed used segmentation, followed by image feature extraction using three different algorithms to extract color, shape and texture information from the image. The feature data was input to a classification algorithm whereby the output would be one of the three disease types or no disease present. Anthonys and Wickramarachchi (2009)

In 2015 another experiment (using similar aproach toAnthonys and Wickramarachchi (2009). Involved using image segmentation using using K means clustering and other image processing techniques, to find features in the image and create a one dimensional binary feature vector, to be processed by an ANN. Khirade and Patil (2015) The accuracy of detecting powdery mildew, yellow rust and aphids on wheat were 86.5%, 85.2%, 91.6% and 93.5% respectively. This non CNN teqnique has subsequently been rendered redundant as this method requires a greater number of computational processes and acheives results that have been surpassed by CNN's. However, the image segmentation teqnique (with the purpose of isolating the leaf from the background) Is also sometimes used in CNN aproaches.

A year later and there have been great successes in identifying crop disease with CNN's. In 2016 a paper was published running experiments on a 38 class crop disease dataset over 14 crop species and 26 diseases (or absense thereof). Resulting in 99.35% accuracy on a held-out test set (using GoogLeNet). Mohanty *et al.* (2016). This study utilized two established CNN architectures, namely AlexNet Krizhevsky *et al.* (2012) & GoogLeNet. Szegedy *et al.* (2015a) With GoogLeNet achieving a higher F1 score in almost all cases. This study also highlighted the effectiveness of using colour images when training the models. In all experiemnts, the color or segmented image models performed better as appose to grey scale images. A suprising aspect of the results is the fact that the segmented image models almost always permed worse than the colour image models, with the best performing model being trained on colour, non-segmented images. This may be due to some bias being present in backgrounds of the dataset images. Or it may be more effective to not perform segmentation on the images prior to training.

Then in 2018 InceptionNetV3 (a later iteration on the GoogLeNet i.e. InceptionNetV1 architecture) is used on a very similar if not the same dataset of 38 class crop diseases (this paper cites the number of crop species to be 13 appose to 14) and 26 diseases. Resulting in a slight increase in accuracy of 0.39%, to 99.74% classification accuracy. Kulkarni (2018) Prior to training the models, the training images were segmented to give the crop leaves a black background. Notably this study began with pre-trained InceptionV3 models and fine tuned them by training a seperate model for each type of crop. This allowed a system whereby the network is fed an image, it determines the crop, then it passes the image to the specific network tailored to that crop species. Unfortunately there are no results available of experiments with non-segmented data to compare with the Mohanty paper. Interestingly the author (Omkar Kulkarni) states 'The pre-processing of image is essential for removing noise and segmentation of the image which helps in improving the accuracy of CNN model'. However, the results table [APPENDIX LINK] produced by Mohanty *et al.* (2016) show non-segmented images acheiving higher accuracy. The increase in accuracy for this paper when comapared to Mohanty *et al.* (2016) can be explained by the better performing InceptionNet architecture

A study performed in 2015 by Sungbin Choi Choi which involved plant species identification from a multi-image observation query. Found that an ensemble of CNN's performed with better classification performance. The study utilized an ensemble of fine-tuned[2] GoogLeNet architectures.

This paper Zhu *et al.* (2018) performed experiments for plant species identification and justified that 'using CNN's can provide better feature representation compared to hand-crafted features.'

[2]meaning pre-trained on generalized data and then improved with domain specific data

From the reviewed sources it is aparent that the best performing architectures have employed the Inception Szegedy *et al.* (2015b) module. which is consistent with the findings of Wu *et al.* (2019). This paper found that when pitted against Resnet He *et al.* and InceptionNet varieties the Inception-ResNet-v2 was the best performing well known architecture. However the researchers crafted a bespoke architecture using inception modules that slightly outperformed the Inception-Resnet-v2.

2.2.3 Existing CNN Architectures

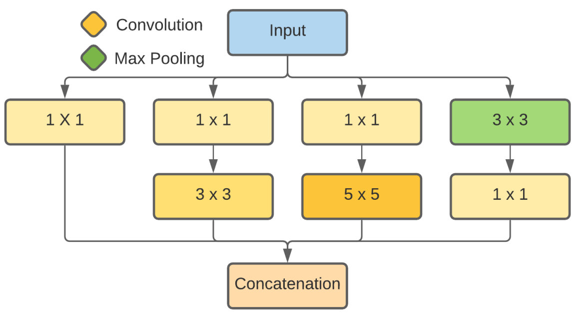

Figure 1: Inception Module Example.

On of the first major pioneering works in the field of machine vision has been Cun *et al.* (1989) which designed a CNN architecture to be used for character recognition and printed on a chip. With all of it's 49 templates (today known as filters) being designed by hand. And later goes on to to introduce 'Digit Recognition Using Constrained Automatic Learning' which allows some parameters of the network, including filter values to be updated automatically via backpropogation, removing the problem of having to design all of the filters manually. Manually designing filters can be seen as introducing some priori information to the network. Which also aleviates the problem of having a scarce amount of data. A noteworthy step when training their network is the final stages of training whereby they waited for their model to converge at a minima (minimizing loss, i.e. error) which took 23 learning passes, then trained it for a further 5 passes on a dataset that quote "had undergone slightly different preprocessing", resulting finally in a 5% error rate at classifying handwritten digits. Later in the work of creating LeNet5 LeCun *et al.* (1998) we see the invention of fully automated parameter tuning via backpropogation (for more on backpropogation see Cun Yann le (1988)) that led to, acheiving 0.8% error rate at identifying hand written digits. This was in part made possible by the increased amount of data available to train the network.

Following from LeCun we see the next popular architecture (winner of the 2011 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)) that is often used as a benchmark in later papers AlexNetKrizhevsky *et al.* (2012). A noteworthy aspect of it's design is the choice to change from using the tanh activation function that we find in **?** to the relu activation function, which has been adopted in all state of the art aproaches today. The relu funciton is linear so we see a decrease in training time by virtue of using a function that is less computationally expensive. They found that due to the large number of paramters in the network, it had a tendency to overfit the training set. To mitigate this, data augmentation is used including reflections and zooming in on only parts of the image such as corners/middle. Other augmentation included normalizing the RGB intensity across the image as it stated that 'object identity is invariant to changes in the intensity and color of the illumination'. This network also employs dropout as discussed later. When training the model stochastic gradient descent is used, whereby the parameters of the network are updated based on the loss of each training example. We also see mention of the concept of 'fine tuning' a network, whereby one takes a network that is pre-trained on a generalized dataset and then further trains it on domain or dataset specific input. Furthermore, we see evidence for the virtue of using ensembles of CNN's to reduce classification error. Finally the paper concludes stressing the importance of depth for classification accuracy citing a 2% reduction in classification accuracy for top-1 performance if any convolutional layers are removed. However a later paper Zagoruyko and Komodakis has established that 'widening' networks can be just as or more effective than deepening. The concept of widening can also be used to explain the effectiveness of InceptionNets.

Another major improvement in CNN architecture comes in the form of VGGNet (Visual Geometry Group) Simonyan and Zisserman (2015). In this paper they studied a range of different network depths. They take the aproach of further deepening the network to improve accuracy, citing their use of small receptive feild filters (3x3) as the main key to their success. This logically follows as the smaller the receptive field, less weights need to be tuned. This fact coupled with using a moderate amount of filter channels (512 at most) allows for more conv layers. Earlier designs were using larger filters especially in the first layer namely Krizhevsky *et al.* (2012)&Sermanet *et al.* (2013). To aid in training their deeper network variations, pre trained layers of shallower networks are used as the initial and final layers, with the middle layers remaining randomly initilaized. When training the network mini-batch training is employed whereby the weights of the network are not updated until a batch of training examples has been seen, as apose to stochastic whereby the weights are updated after a single example. Data augmentation is also important in the aproach; each training image was cropped a [what goes here] . Their best performing ensemble of 2 VGG nets acheived 6.8% top-5 test error on the ILSSVRC test classification dataset. VGG Much like it's predecessor AlexNet, employs max pooling layers and Relu activation function.

Continuing the theme of deepening networks we are introduced to the ResNet He *et al.* Winner of the ILSVRC 2015 classificaiton task. ResNets introduce the concept of the skip/shortcut/residual connection. Meaning ouptut from an earlier layer is added to the output of a layer some convolutions ahead of it. The emergent result is performance no longer degrading with greater network depths. A reason for this, is less feature information is lost through the convolution operations.

Due to the fact that every conv operation reduces the volume of input to the next layer, information is lost after each operation. The skip connection mitigates this by adding earlier information to later layers. With an ensemble of ResNets it's top-5 err on the ILSVRC test set was 3.57%

Taking the residual connection further is the DenseNet. In this architecture we see a stack of dense blocks, in a dense block each layers output is concatenated with each proceeding layer, notably a ResNet adds the output of the previous layer whereas DenseNets concatenate. This has the benefit of preventing the network learning redundant feature maps and preventing the vanishing gradient problem. The network also employs 1x1 convoution layers to reduce input size to the more expensive 3x3 layers, much like the InceptionNet. After each dense block is a transition layer which consists of a 1x1 convolution operation and a 2x2 average pool with stride 2, the transition layer has the purpose of compressing the output volume. it is unusual to see average pooling used as it is typical to see max pooling, no justification is given for this choice. The authors also claim on the basis that models with over 25m parameters still see increase in accuracy that the DenseNet architecture prevents overfitting. For comparison the original ResNet contains 1.7m parameters.

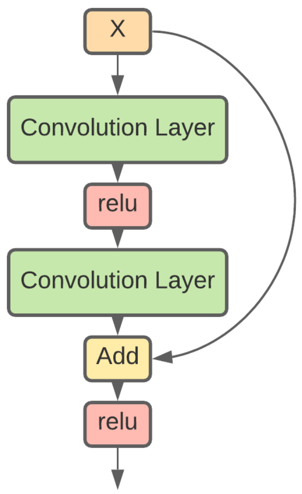

Figure 2: Residual Identity Block

An earlier but more distinctive when compared to earlier iterations of design, comes in the form of InceptionNet/GoogLeNet Szegedy *et al.* (2015b). Part of their philosophy when creating the network was to make something that could perform well on hardware that was more widely available quote "the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that they do not end up to be a purely academic curiosity."Szegedy *et al.* (2015b).The feature that sets InceptionNet architectures apart is the different varieties of Inception Module. An inception module is multiple convoulutional operations occuring in parallel and finally being concatenated together. Aditionally 1x1 convolutions are employed to reduce the input volume to later convolutions and therefore improve training time and prevent bottlenecks. This aproach of adding more channels in a single layer is commonly refered to as 'widening' the network.

As it stands, there are few, (if any) papers exploring the efficacy of the novel fractalNet architecture Larsson *et al.* (2016) for crop disease detection or on any other dataset aside from the commonly used CIFAR10 & 100.

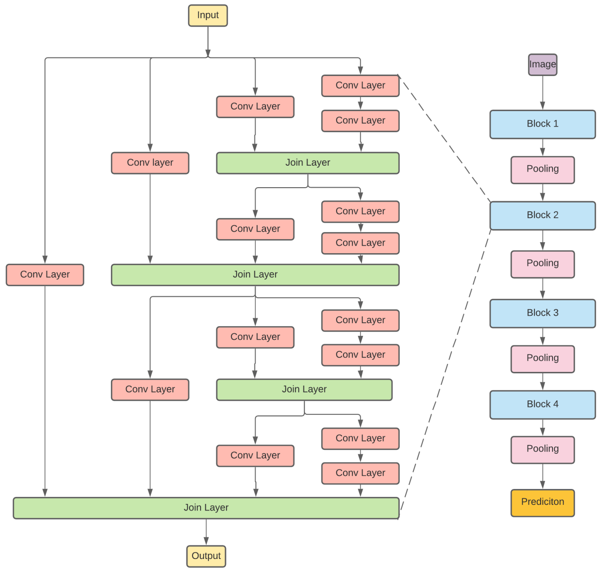

Figure 3: FractalNet Architecture.

The Inventors of the fractalNet performed experiments that justified their use over ResNets. Demonstrating results that showed improved classification accuracy. Another feature of the fractalNet architecture is the ability for the user to choose between speed of prediction and accuracy of prediciton as it is possible to take longer or shorter paths through the network. Longer paths being more accurate and more time consuming. Drop-out is also applied during training but at a coarser level than individual filters, rather, entire paths are dropped at join layers, leading to the creation of pathways of equally strong predictors. It is said in the paper they see their architecture as a more generic design that is not in it's final form as the best configuration of convulutional module and join layer has not been determined. The architecture can also be seen as a harness for creating an ensemble of networks of different depths.

Research has been conducted in 2016 Zagoruyko and Komodakis that determined 'wider' networks perform better than their deep narrow counterparts. They found that their 16-layer deep network had the same accuracy as a 1000 layer thin deep network with a comparable number of parameters. With the wider network being faster to train.

A key feature of CNN design is size and number of filters. In all examples mentioned (Resnet, InceptionNet, AlexNet etc), filter size is always odd and square. For instance 1x1, 3x3, 5x5 and so on. Size of filter will determine the size of the feature the filter will encode for. Number of filters will determine the depth of the convolution output. It is noteworthy that the Resnet50 architecture He *et al.* uses almost entirely 3x3 size filters. Whereas, the Inception module employs a mixture of 1x1 through to 5x5 convoutions.

A technique known as dropout is a feature that has been employed in increasingly deep networks to prevent overfitting and was first introduced by Srivastava *et al.* (2014). The principle behind its operation is randomly dropping paths between neurons during training. This ensures that predictions do not become overly reliant on a single (or gorup of) neuronal activation(s), that can correlate to some bias in the training data. And prevents neurons from becoming co-adaptive. This teqnique was employed in AlexNet Krizhevsky *et al.* (2012) whereby they used dropout in the first two fully connected layers of their model. They found that using dropout prevented overfitting but made training take twice as long. This exact method whereby dropout is used in the first two fully-connected layers is seen again in VGGNetSimonyan and Zisserman (2015); both methods choosing to employ a dropout chance of 50%.

Another aspect of CNN's is the process by which they progressively narrow down the possibilities until they arrive at their conclusion. This is done by encoding loose abstract forms that relate to groupings of objects in the early layers and gradually arriving at very generalized forms such as horizontal lines or sine waves in the later layers. So to give an example drawn from the AI microscope [CITE AI MICROSCOPE] . In an early layer we may find a neuron that encodes for two semantically unrelated objects, yet objects that have a form in common such as dogs and turnstiles, a middle layer may encode for things under the sea or star shaped objects. Final layers encode for more generic features such as diagonal lines or squares. In the case of the dog and the turnstile, one can observe that a 'branch' of a turnstile is equateable to the leg of a dog and the 'console?' of the turnstile be equateable to the dogs body.

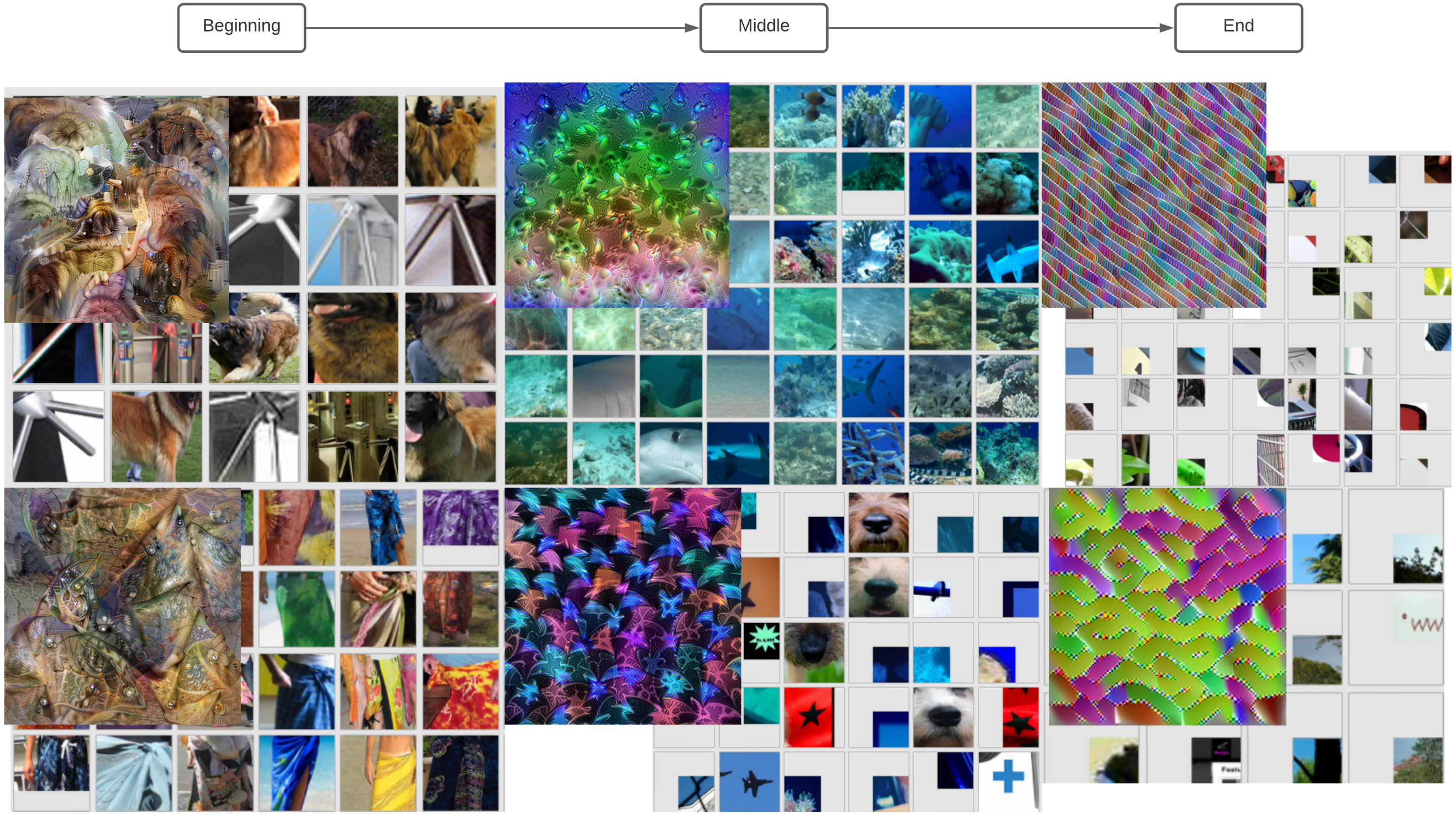

Figure 4: InceptionNet Filter Activation Maps (Examples taken from OpenAI Microscope)

2.3 Problem Definition

As it stands there are currently (09/02/2021) no easily found [1](i.e. not present in the first 3 pages of a google search for 'crop defect identification' and 'What's wrong with my crop') web interfaces for interacting with a crop defect identification service.

Although there exists very capable publicly available image classification networks. Yan (2021) There is no bespoke application catering soley to crop defect identification, that has the benefit of providing recourse information to the user.

2.4 Proposed Solution

To provide a web service that interacts with a convolutional neural network (CNN) backend to diagnose crop defects such as, nurturing problems e.g. lack of water/nitrogen/C02, too hot/cold. And external threats such as crop disease/pest infestation. The interface will be simple and intuitive as possible. The UI should minimise points of interaction and streamline the process of uploading a crop image to be analysed. The web service will return information regarding the percentage likelihood of each kind of crop defect, including images that are of similar nature to the one analysed.

2.5 Aims & Objectives

These should be SMART with clear success criteria defined specific, measurable, achievable, realistic, Timebound

2.5.1 Aims

- To aid gardneners and smallholders in identifying crop defects.
- To aid gardeners and smallholders in taking relevant recourse.

2.5.2 Objectives

- Provide a way for a user to upload an image to be analysed.
- Display infomration regarding the likelihood of each kind of defect.
- Display recourse information alongside defect information.
- Have gallery of images filtered by crop and disease type.

2.6 Risk Table

Table 2.1: Risks Table

ID	Name	Likelihood	Impact	Control Mechanisms
1	Improper Time Management	med/low	high	Follow the Gannt chart
2	HDD/storage failure	low	high	All work will be backed up to github
3	Illness/Injury	med	med	Should the need arise I will apply for an extention to the due date.
4	RSI (repetetive strain injury)	med	low	Work with proper postureand set up workstation properly.And take frequent breaks
5	Eye strain	med	low	Ensure room is well lit when working on a screen.
6	Incorrect Task Prioritisation	med	med	Iteratively re-asses the work being done and compare it to the mark scheme.
7	Postural problems	med	low	Work with proper postureand set up workstation properly.And take frequent breaks

(ID, name, likelihood, impact, control mechanisms / accept)

2.7 Overview

Introducing rest of dissertation (with cross references to sections)

Chapter 3 - Methodology

3.1 Project management methodology

I will use a Feature Driven Agile method. Meaning the workflow will by cyclical and focus on iterating over designs and prototypes. This will involve:

- Requirements elicitation.
  - This involves determening the needs of the user and defining requirements to meet those ends.
- Feature design (UI).
  - Features will be designed at first using wireframe models. Then on later iterations, colour and shading will be added alongside further usability considerations such as highlight on hover etc.
- Feature implementation research.
  - This step involves determining the apropriate technologies and libraries to achieve the design. This is nececarry to realize the constraints that are imposed by the implementation method and know to what extent the design is feasible.
- Feature implementation.
  - Writing the code to create the feature.
- Feature testing.
  - Initially testing will be done manually with valid values until later iterations whereby extraneous values will be introduced. Once the feature is in it's final iterations a unit test will be introduced.
- Evaluation.
  - Does the feature meet the requirements and fulfill the needs of the user?

This workflow will consist of a single cyclical workflow, with two nested ”sub workflows” whereby upon completion of a step, it is sometimes nececarry to loop back on oneself to perform futher refinement. As illustrated by the diagram below. Throughout the project the focus of the workflow

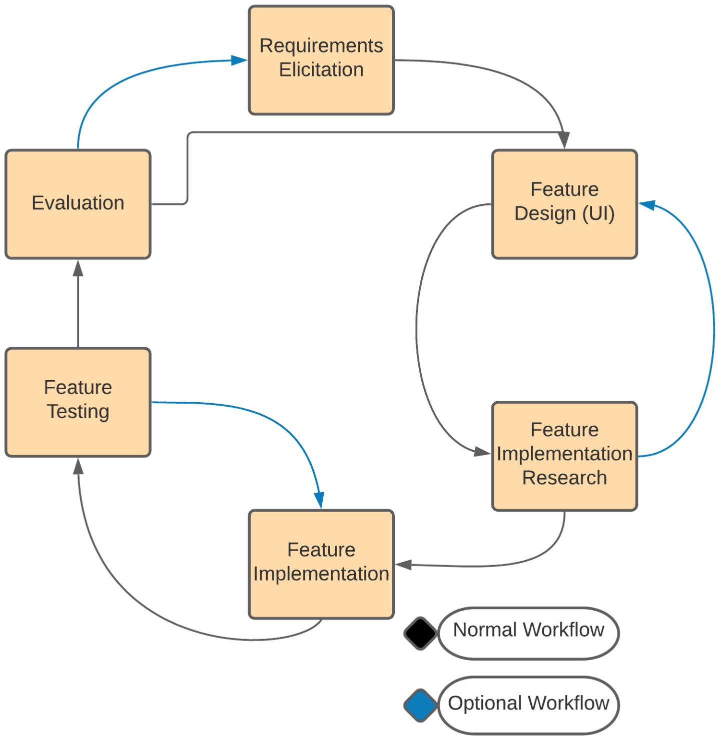

Figure 5: Development Lifecycle

will shift as illustrated by the diagram below.

The paper [CITE BIS OF INNOV] makes a compelling argurment for the usage of agile methodologies over more linear project management methodology (PMM) styles such as waterfall. Agile focuses on design being done 'on an ongoing bases in smaller chunks' [CITE BIS OF INNOV]. It is also highlighted that agile allows development to respond to feedback and change. In fact agile has customer, designer and developer feedback baked in to it's formula in the form of regular team meetings, especially if using the SCRUM 'flavour'.

For a lone developer Agile with Kanban is the PMM of choice as it allows the flexebility to iterate on designs as one learns more about the technologies being used and gives the freedom to modify one's requirments in light of newly found research and technical limitations. Aditionally the nature of agile is to haves short lifecycles which has the virtue of frequent milestones. This allows one to better track progress of the project by being able to see how many tasks have been completed on the Kanban board during the development cycle[4].

Using Kanban to track tasks makes sense for a single developer as appose to using feature driven development. As this allows for non-programming tasks to be tracked in the same way as programming tasks.

[4]Granted tasks alone are not the be-all, end-all metric but it does give some idea of progress

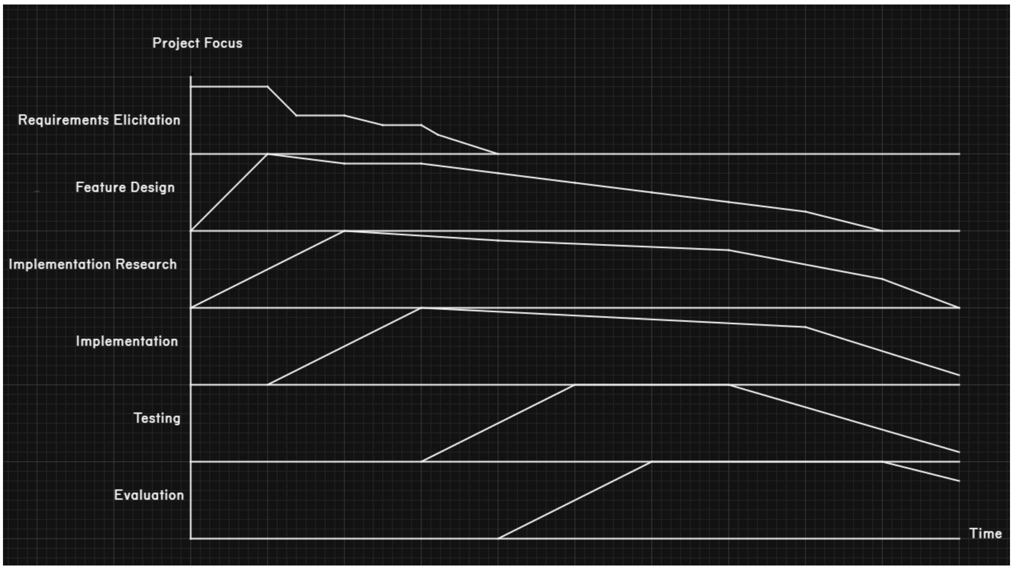

Figure 6: Project Focus Over Time

Furthermore, agile is a worthwhile methodology to use and familiarise oneself with as it has become the new norm in industry. As shown by a study conducted by Hewlett Packard.

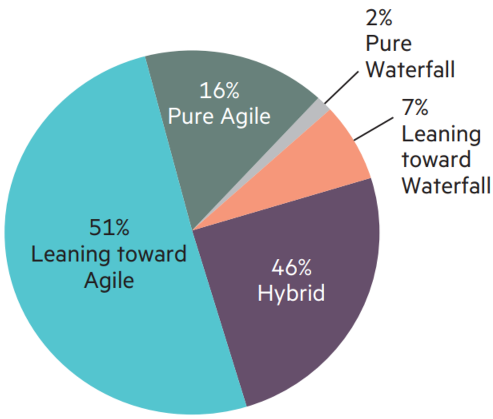

Figure 7: Primary development method used in organization across projects (601 respondents) diagram created by Hewlett Packard

While creating the product one's goal will be to implement a Minimum Viable Product (MVP). Defined by a product that fulfils the 'must have' requirements before expanding it's features to include 'could have' features etc. Aditionally, once the MVP is acheived, design considerations such as ease of extensability and addition of new features will be more heavily focused on. Sometimes with major code refactor occuring at this point, as to acheive the MVP quickly it is sometimes nececarry to 'hard code' parts or implement features in a way that may not be conducive to easy maintinence or serving content dynamically.

3.2 Evaluation Design

This will be conducted by linking users to the website with a link to an online survey alongside a link to a google drive folder containing relevant test images. With the questionaire mostly consisting of System Usability Survey (SUS) questions. And some aditional bespoke questions regarding accuracy of prediction.

3.3 Requirements Elicitation

As the scope of the problem domain is narrow in terms of interaction points for the user. Requirment elicitation will be driven by determining features that will build towards the solving the problem at hand. If for instance the problem was providing some kind of E-commerce website, project management application or social media website, the number of different features that could be employed in any one of these domains is vast and therefore one would need to consult the target audience and elicit the kind of requirements they would like to have. On the contrary, this project is far narrower in terms of points of interaction for the user. Therefore requirements will be determined on the basis of whether or not they contribute towards providing information about crop defects to the user.

This is not to say user feedback is not useful for this project it still a good way to be alterted to any usability issues. Such as bugs, difficulty understanding how to use the application (what icons mean, order to carry out steps etc) and difficulty seeing UI elements due to poor colour choice. And for other projects with a specific customer in mind can use customer feedback to determine for instance how the program should handle erronus data. Or is it imperative that this program exececutes quickly? Or will the client be handling sensitive data and need to have the security of the application in mind?

3.4 Feature management

To track the creation and completion of features, a Kanban board will be used. This will include columns for 'To do', 'Doing' and 'Done'. To determine which features will be prioritised one will employ the MOSCOW method see Feature Management.

3.5 Design Methods

- Requirements Elicitation

  - To better conceptualize the needs of the user. Use case diagrams and activity diagrams will be utilized.

- User Interface
  - Wireframes will be uitilized to establish interface element placement i.e. layout.
  - More detailed mockups will be created when the earlier wireframes are constructed as prototypes and the concept is proved acheivable.
  - A colour picker will be utilized to define the colour scheme.
  - In later iterations of the design, once there is a functioning UI, usability will continue to be refined with the help of existing usability research, to guide the usage of font/colour/highlight on hover/font size etc.
  - Additionally once a desktop friendly layout has been established, work will begin on optimizing a version for mobile.

- Back-End
  - UML will be used to show the overall design of the system through structural diagrams. These will show the interfaces of the classes and how they will interact with one another.

3.6 Testing methods

Testing will be conducted iterativley as the functionality expands, with unit tests being introduced for some components depending on time constraints. Testing will consist of firstly confirming that when interfaces are interogated with sound data, the responses are consistent with requirments. And secondly stress testing the interfaces with extraneous data to ensure that apropriate error responses are given and the application does not simply crash. And if in the case of crashing, is able to automatically re-start.

3.7 Version control

I will be using Git and Github. This will allow the creation of branches to explore experimental parts of the soloution space without disrupting the progress of the main branch. If the experimental implementation is successfull it will be merged with the main branch. It also allows the development of features in paralel, with any conflicts in their implementation being resolved at the merge stage. The inclusion of a remote repository allows for work to continue on a seperate machine if nececarry and later be synced with the local main branch.

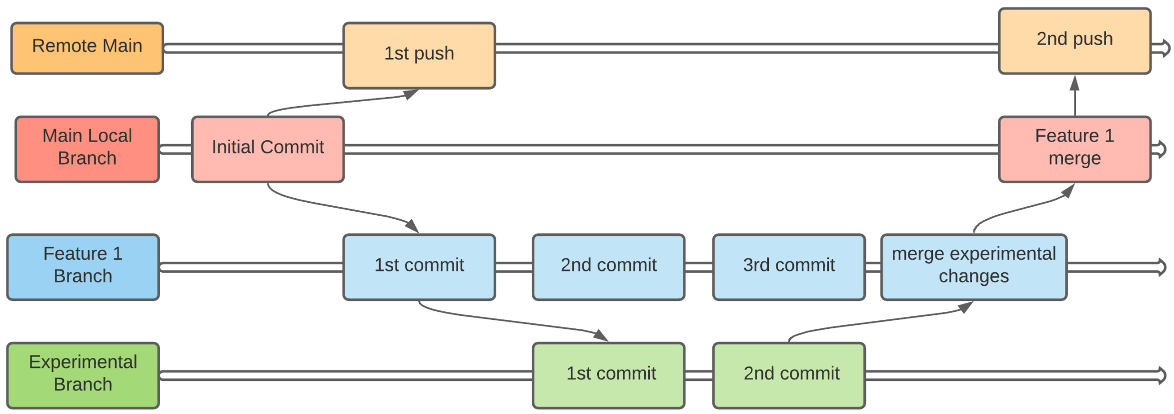

Figure 8: Example Workflow To Highlight Branch Usage

3.8 Evaluation methods

3.8.1 User Interface

The main method that will be utilized to determine the quality of the user interface will be the System Usability Scale (SUS) which can be seen here. (CITATION) The evaluator will be given remote access to the webservice. They will also be provided with some sample images to test the perfomance of the CNN incase they do not have suitable images of their own. The opinion data will be collected via online questionare.

3.8.2 Convolutional Neural Network (CNN)

Metrics for the evaluation of the CNN will be:

- Time to train the network on available hardware
  - The constraint here being if the network cannot be trained on the available hardware in under sixteen hours. Purely for practical considerations.
- Accuracy of CNN predictions. (which will be most effective when there are equal numbers of samples belonging to each class) $Accuracy = \frac{CorrectPredicitons}{TotalPredictions}$ Else if the samples are sqewed, the network could be a faliure at detecting a specific under-represented class, yet still score high accuracy.
- Precision. This is the number of correctly predicted images out of all predictions of that class. $Precision = \frac{CorrectlyPredictedforClass}{TotalPredictedforClass}$ The network is precice for a class when the predictions it does make are correct. Precicion cannot be used in isolation due to the fact that the network can have a high precicion for a class but still fail to identify the majority of images

for that class. Succeeding soley on the fact that the images it has classified are correct.

- Recall. Is the correct number of predictions for a class out of the number present of that class. $Recall = \frac{CorrectPredictedforClass}{No.PresentForClass}$ This metric can also not be used in isolation due to the fact it does not take in to account the number of false positives. i.e. The number of images incorectly classified as the class in question. For example, if an image dataset contained three classes A, B, C, and the classfiier labeled all images A. The recall for A would be 100 percent.
- F1 score. This metric tries to find the balance between precision and recall and can be expressed as $F1 = 2 \times \frac{1}{\frac{1}{precicion}+\frac{1}{recall}}$

3.9 Initial Designs

Firstly I have created a wireframe UI

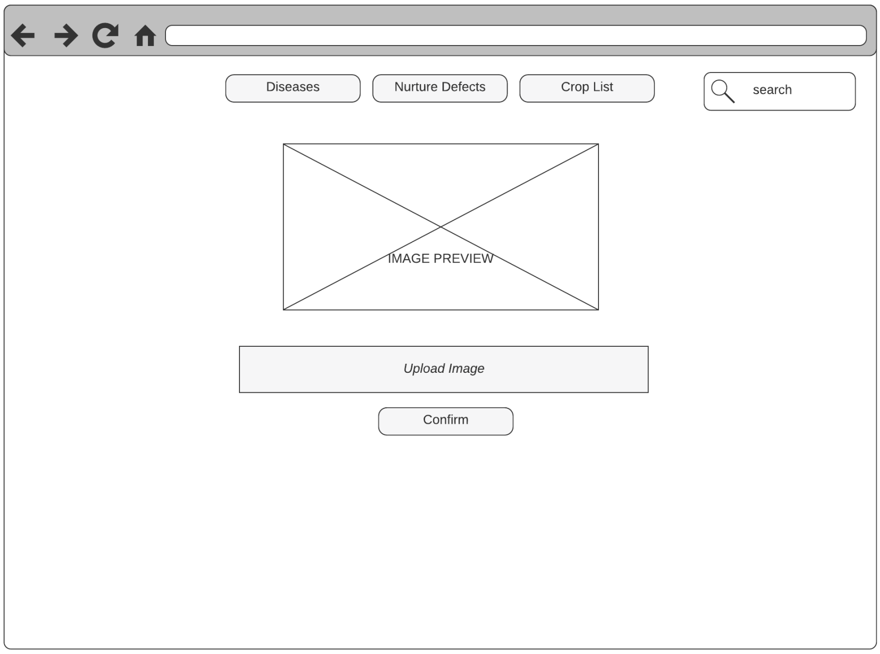

Figure 9: Homepage Wireframe

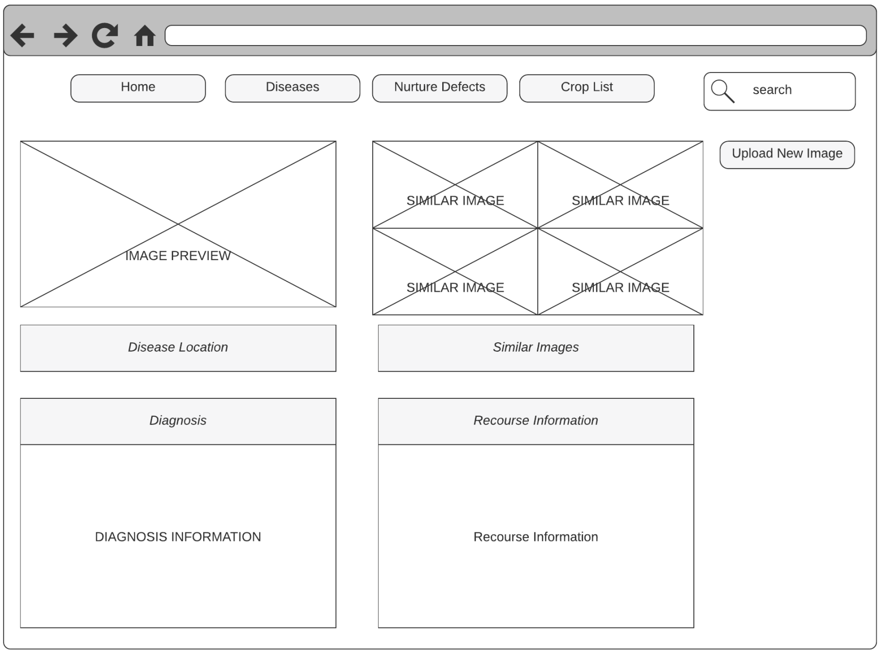

Figure 10: Defect Information Wireframe

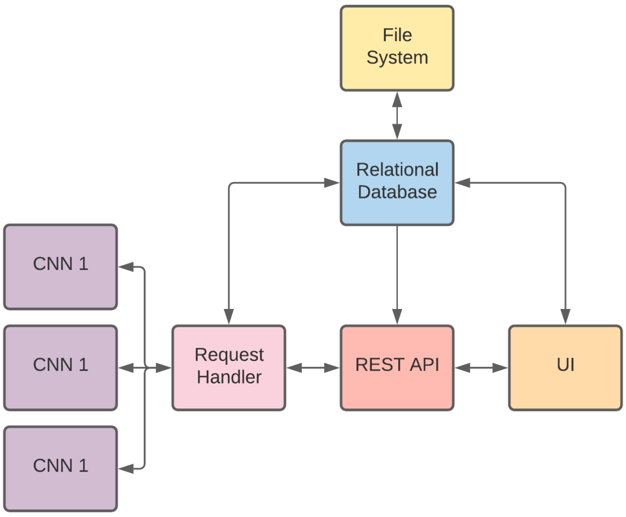

Figure 11: System Overview

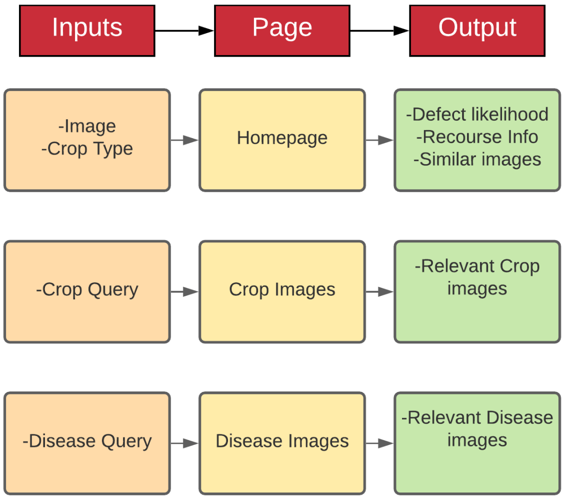

Figure 12: Input/Output overview

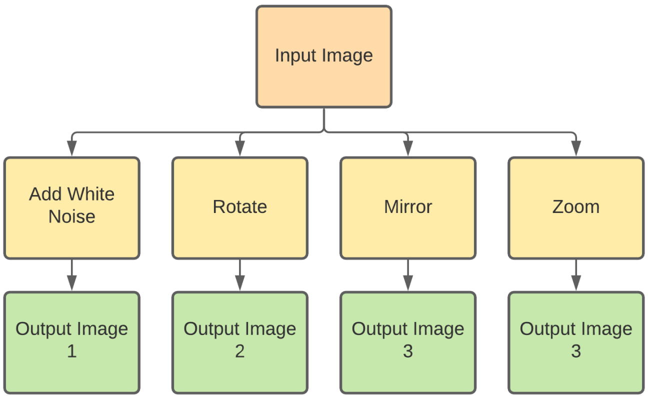

Figure 13: Input/Data Augmentation Methods

3.10 Employed Technologies & Justifications

3.10.1 Software Structure

3.10.2 Back-End

The design aims to seprate the functionality of the CNN and the User Interface (UI) through use of a Representational State Transfer (RESTFul) Aplication Interface (API). The central idea of this aproach is to allow a seperation of distinct components which allows for flexebility in managing updates and changes to each component. The 'backend' which in this context refers to the CNN and it's related functionality, such as processing it's input's and outputs; will both recieve input and serve it's output through the interface of the RESTFul API. each 'endpoint' or in other words internal class of the API will all inherit from the Resource interface which means it will get the main four HTTP protocol functions. POST, GET, PUT, DELETE. All interogations of the CNN will be conducted through these requests.

A key feature of A RESTFul interface is the fact that no client information is stored between requests, this aids in making the interface more scalable and iteroperable. Scalable in the sense that increasing clients does not increase the amount of information the backend will have to store and interoperable in the sense that any client, be it mobile appliation, CLI program or webpage can use the interface.

To increase orthogonality of the system, defect images are stored standalone on an image server. With the API serving the name of the defect which gives the information nececarry for the webpage to retrieve the correct images. This is because the directory structure of the image server is set up such that directory names are defect classes and all image names are numbers. Having the images stored on a seperate server rather than bundled with the front-end makes the service more useful. As it means any other interface can also choose to obtain the defect images (although image server endpoint information is not served by the API).

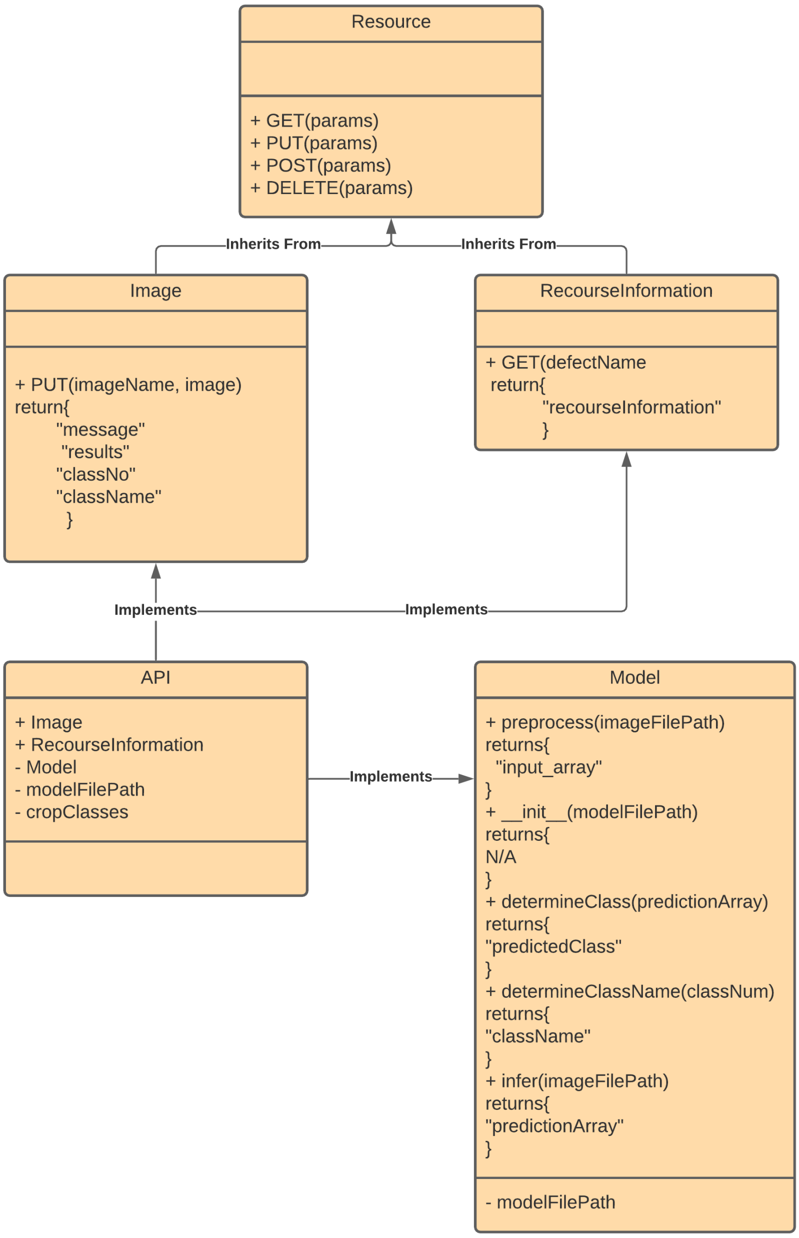

Figure 14: Backend Class Diagram

3.10.3 Front-End

Using the Vue.js webpack framework means the front-end will be split in to components. A component acting much like an endpoint in an API. Each component will have it's own set of functions and data, with the purpose of seprerating functionality of the UI in to manageable subsets of functionality and styling with the objective being to decouple dependancies between compoonents as much as possible. This helps simplify the development process as it seperates the problem in to distinct parts and it helps with maintinence, allowing them to be tested and developed on in isolation.

UI components will be defined using Hypertext Markup language (HTML) and Cascading Style Sheets (CSS). Interativity will be scripted in Javascript. All three extremely common in web development.

The main functionality of the web-app will be uploading an image to be analysed by the CNN and recieving a prediciton of which defect is/isn't present. This will be conducted by a single component. Once a prediction has been made, the user can choose to see more information (consisting of images and recourse information) about the defect, which will be handled by a second component.

Responses from the API will come in JSON format. With the intention being that as much data processing, such as determining which class is predicted, image preprocessing, and determining of defect image URL's, defect names etc. is handled on the back-end, meaning that any client wishing to interact with the API does not have to implement these features themselves. Therefore all functionality implemented in the web app will concern the consumption of user input and display of API response.

3.10.4 Hosting

To make the application accessible on the internet one will utilize virtual machines (VM's) running linux, hosted in the cloud. There will be three VM's, one to host the front-end, one to host the back-end and the other to host the defect images. In all cases nginx will be used to create HTTP servers that will handle all functionality regarding the HTTP protocol and the socket layer.

Aside from creating backups of work, version control allows production environments (in this case the VM's) to be easily updated with new features, simply by pulling the latest changes from the repo, installing any new dependencies and re-building the application.

3.10.5 CNN Model Creation & Integration

Seperation of the sofware components allows for the easy interchange of CNN models. This means one can train a model in GoogleColab, utilizing their GPU run-time and save it in the .h5 format. Meaning that, changing the model is as simple as changing a single filepath on the API server. Additionally if one wishes to extend the number of defects (classes) the model is able to predict; a number of steps must take place. Firstly, the filepath to the JSON file defining all classes must be updated on the API to reflect the new set of classes. and the images for the new defect must be added to the image server, or in this case due to hosting limitations, added to the webservice filesystem. Making the process one that is as straightforward as updating two file paths and uploading images using secure file transfer protocol (SFTP) to a server.

When training the CNN it isn't clear if 'fine-tuning'(see lit review) an existing CNN is going to be more effective than training a network from scratch. Therefore some experimentation will be nececarry. However, training large networks with millions of parameters can take days or weeks, even when utilizing high end GPU's. For this reason the CNN accuracy is unlikely to acheive state of the art performance in the timeframe available for the project. Fortunately, the application is designed in a way that allows easy swapping of the model when better performance is realised.

3.10.6 where to put this paragraph?

The initial architecture one experimented with was the InceptionNetV3, the reason being that one already had a familiarity with the architecture and therefore could rationalise about what may need to change in order to get better performance on the target dataset if initial experiments were not satisfactory. One opted to use a model pre-trained on the ILSSVRC dataset and fine-tune it to suit the crop dataset. This resulted in the model taking around six hours to train over an epoch (using Google Colab GPU runtime) and also drastically overtiffing the training data. One reasoned that the vast number of weights could be to blame for both problems, firstly the more weights the greater the training time, secondly, more filters leads to the network making less generalisations and being able to store many bespoke feature maps to target subsets or perhaps even individual images. Hence when it comes to an unseen example it's feature maps are not generic enough to aid in identifying the unseen image. Therefore one decided to use a a smaller network, 'EfficentNetB0', a network available as a pre-trained[1][1] CNN through the Keras library. For comparison the InceptionV3 architecture is 92mb and contains 23,851,784 parameters (weights) whereas the EfficientNetB1 is 29mb and contains 5,330,571 parameters (approx 1/3 the parameters). Fine-tuning this network proved very successfull, acheiving a 98.6% accuracy. For this dataset, due to the fact all classes the network is classifiying for are relatively similar, less feature maps are needed to generalise about the forms present. To contrast, a network that must classify 100 different classes ranging from a baseball to a shark. with all objects present in different contexts, for instance a closeup of the baseball sat on a table, and a long range shot of it mid flight across the stadium, set against the background of spectators. And it becomes easy to see why so many filters would be nececarry. If we take the crop-example we see all images are closeups of leaves with a neutral background. Therefore it is possible to generalise about this smaller subset of possible images with fewer filters. In fact few parameters becomes a desirable quality of the network as it forces the network to generalise as it does not have the ability to 'memorise' all the individual training images.

[1] on the ILSSVRC dataset

3.10.7 where to put this/call this

Throughout iterative process of creating the application, the design of the software is affected by factors which have a cause and effect relationship with one another. Following are some hypothetical scenarios that illustrate this point, such as; designing the system to include a relational database such MySql, as one beleives there may be a need to handle a large amount of relational data. Later in the project, one realizes they only need the application to be able to retrieve dispirate and simple data, so one drops the database, in favour of using the hierachical storage of the filesystem and structured data languages such as JSON or XML. Another example being, one decides to use a novel, bleeding edge, front-end javascript library to aid in creating user-interactive graphing of data. After creating a substantial poriton of the UI functionality, and finding it working smoothly on a high-end desktop PC, one decides to test the application on a smartphone, only to be left with the problem of extremely slow responsiveness. One is then left with the realization, that the novel UI-library is far from optimized and one is now forced in to a bind. If requirments don't specifically request mobile functionality then it would be possible to carry on using the library, however, if one is to take this route, one should have stated in the requirments 'Won't have mobile device support'. However if requirments dictate that mobile support is a 'Must Have' or even 'Should-Have' two main possibilites arise, either the library is being used inefectually and time is needed for the developer to become proficcient at using it. Or the library is simply slow. If for whatever reason the library must be dropped from the project, aspects of the UI design will need to change, to better reflect the reduction in expected features; this being caused by the extra time taken to implement what was previously offered by the library. Either this, or extending the project deadline. A final example is a situation whereby an application stores and handles sensitive information about a user, for instance an online medical checkup. This can lead to large poritons of a codebase needing to be re-factored and documentation updated, if later in the development process the team discovers a security vunerability in a technology or library they are using. Modifications may involve updating class diagrams and system overview diagrams etc.

3.10.8 where to put this/call this

an example to illustrate the process of refactoring. The first peice code was arrived at because the base case was to retrieve the most likely defect. The extended requirment was to find the top three most likely defects. This has resulted in two methods being created for each case as the base case was coded first and extended requirment added in later. Although this is still functional, it's unnececarry and the code can be simplified, by using the result produced by the newly created method, the base-case method and any calls to it can be deleted. (note the method is declared in Model class which has been ommitted for clarity) The final refactor of example one is entirely debateable as to whether or not it is an improvement. It is certainly more terse, which makes it arguably more difficult to read. The counterpoints to this being; one is using less memory to store the code, less keystrokes, and one is not allocating a computationally redundant variable at runtime. The final point being moreso noteable in the language of Python. Given it's interpreted nature, the interpreter will cause the machine to execute this operation (granted it's a fairly inexpensive one). If it were a compiled however, it is likely the compiler would ignore this redundancy and it would have no bearing on the run-time of the program.

```
def determineClass(self, predictionArray):
    return np.argmax(predictionArray,axis=1)

def determineTopClasses(self, predictionArray):
    predictDict = {}
    topClasses = []
    for idx, prediction in enumerate(predictionArray):
        predictDict[idx] = prediction
    predictDict = sorted(predictDict.items(),key=operator.itemgetter(1),reverse=True)

    for entries in list(predictDict)[0:3]:
        topClasses.append(entries[0])

    return topClasses

predictedClass = determineClass(results)
topClasses = model.determineTopClasses(results)

return{
"classNo" : predictedClas,
"topClasses": topClasses,
}
```

Figure 15: Example 1 Code Initial

```python
def determineTopClasses(self, predictionArray):
    predictDict = {}
    topClasses = []
    for idx, prediction in enumerate(predictionArray):
        predictDict[idx] = prediction
    predictDict = sorted(predictDict.items(),key=operator.itemgetter(1),reverse=True)

    for entries in list(predictDict)[0:3]:
        topClasses.append(entries[0])

    return topClasses

topClasses = model.determineTopClasses(results)

return{
"classNo" : topClasses[0],
"topClasses": topClasses,
}
```

Figure 16: Example 1 Code Refactor 1

```python
def determineTopClasses(self, predictionArray):
    predictDict = {}
    topClasses = []
    for idx, prediction in enumerate(predictionArray):
        predictDict[idx] = prediction
    for entries in list(sorted(predictDict.items(),
    key=operator.itemgetter(1),
    reverse=True))[0:3]:
        topClasses.append(entries[0])
    return topClasses

topClasses = model.determineTopClasses(results)

return{
"classNo" : topClasses[0],
"topClasses": topClasses,
  }
```

Figure 17: Example 1 Code Refactor 3

```python
#Used to find X most likely predictions from a prediction array created by keras
#CNN model
def topClasses(self, predictionArray, xClasses=3):
    predictDict = {} #key = original array position & therefore class number
                     #value = prediction likelihood
    topClasses = []#values to return from the function
    #we zip the array into a dictionary to keep track of the values original position
    #as this correlates to the predicted class.
    for idx, prediction in enumerate(predictionArray):
        predictDict[idx] = predictionArray
    #we sort the dictionary in ascending order by value.
    #then we take the first X entries of the sorted dictionary.
    #and take the virst value i.e. class-number from the entries, as each entry
    #is a tuple, key,value pair.
    for entries in list(sorted(predictDict.items(),
    key=operator.itemgetter(1),
    reverse=True))[0:xClasses]:
        topClasses.append(entries[0])
    return topClasses

    topClasses = model.topClasses(results)

    return{
    "classNo": topClasses[0],
    "topClasses": topClasses,
    }
```

Figure 18: Example 1 Refactor 4

Further examples of refactoring. In this example of refactoring we see how

```python
results = results[0]
results = results.tolist()
jsonResults = json.dumps(results)

return{
"results" : jsonResults,
}
```

Figure 19: Example 2 Code Initial

```python
return{
"results" : json.dumps(results[0].tolist()),
}
```

Figure 20: Example 2 Refactor

CNN Data and Training

One of the most important parts when creating a CNN is the data it is trained on [CITE Halevy]. The more properly labled data the network sees, the better. The most comprehensive dataset available (and the one which will mostly be used) is the Plant Village dataset. it contains 54303 images of healthy and unhealthy leaf images, in 38 catagories divided by species and disease. This dataset also contains pre-segmented[2][1] versions of all the images. This dataset was used in two seperate studies cited in the literature review.

Other datasets that may be used to pre-train the network before are the CIFAR10, CIFAR100 & ILSVRC datasets. These datasets are not specifically plant based, but may benefit the network to build initial feature maps. In Choi the authors used networks pre-trained on generic data such as the datasets mentioned. Experiments will be conducted to determine if this benefits classification accuracy

One unfortunate aspect was the fact that one was unable to obtain a dataset that reflected plant defects other than diseases, such as nitrogen/water/heat deficiencies.

3.10.9 Technologies

- Vue.js
  - Vue.js allows for rapid creation of usable websites thanks to its webpack framework, which handles most of the boilerplate needed to handle URL routing and overall structure. It is also considered easy to learn when compared to React and Angular im Studiengang Bachelor *et al.* which for this project is optimal. Although Vue is currently less popular than React and Angular its popularity is rising, which means its not entirely redundant to learn from an employability standpoint. Vue is a component based framework, meaning it creates a structure to allow the programmer to utilize component technologies to build the website. The component technologies are; custom elements, which allow the programmer to embed javascript code in to either existing HTML elemnts by inheriting from them, or creating entirely new ones. HTML templates, which allows for sections of HTML to be re-used throuought the application, for instance navigation bars. And Shadow DOM's, which allows for DOM elements to contain sub-trees of elements, this is to aid in encapsulating the scope of the contained scripting and styling, meaning individual elements can have their own layers of encapsulation within them.

[2] meaning the background has been blacked out

- Bootstrap
  - Bootstrap is a CSS framework that has predefined classes for styling HTML elements, this helps with the asthetics of the webpage. For a solo developer this can allow for more time to spent on making the functionality more robust instead of spending time on looks.
- Jupyter Notebook / Google Colab
  - This pthon environment allows for easy prototyping of code. In the case of Colab, it allows for the usage of google servers and GPUS which can be used to train neural networks. With the benefit of being able to then save the trained weights for usage elsewhere.
- Tensorflow & Keras
  - These pthon libraries allow for the creation of complex CNN architectures by abstracting away the underlying matrix/tensor multiplication of deep learning. Allowing for the programmer to declare relationships between CNN features such as convolutional layers, batch normalisation, dropout etc. Without having to construct their inner workings. It also allows for the creation of user defined layers, so adding in ones own bespoke functionality for a layer of the CNN is possible.
- NginX
  - NginX is a web server. This software, installed on the virtual machines (VM's) hosting, front/back end and files. Handles incoming HTTP protocol requests on ports it is configured to listen to, and serves content accordingly. Additionally, if the need arises, NginX can be used as a load balancer. Taking incoming requests and distributing them amongst the back-end servers via a reverse-proxy. With the reverse-proxy aproach it is also possible to cache response data. Reducing load on the back-end.
- Gunicorn
  - Gunicorn is another technology used to host the application. This time it is used to interface with the Flask API. It is a Web Server Gateway Interface (WSGI) HTTP server. This gives an interface to the API to allow it to send data back and forth between the web server, NginX.
- Flask/FLask-Restful
  - This library provides a framework for creating RESTFul API's in Python and gives the

Resource class to inherit from for each endpoint. Flask also provides a built in development server for testing the appliation on your local machine.

•

3.11 Requirements

For assessing requirement priority one will employ the MOSCOW method [CITE Clegg, Dai; Barker, Richard]. This creates subcategory of requirments with the headings Must/Should/Could/Won't - have. The contents of the catagories can change during development in light of development progress. Items placed in the Won't-have category prevent the process of 'scope-creep'.

- Must Have
  - Ability for user to upload an image
    - The user will be able to choose an image file from their local storage using a file explorer popup.
  - CNN that is capable of classifying at least 2 different defects across 2 different plant species.
  - An API that allows communication from the UI to the CNN
  - API must be able to receive images.
    - Accepted formats being .jpg & .png
  - API must return defect information, which will be an array of probability values for each defect class
  - API must return recourse information.
  - Application must display images that show the predicted defect.
    - These images may be stored either on a seperate server to the front-end. Perhaps in the API servers 'static folder'. Alternatively they will be bundled with the front end.
  - The API will be robust enough to handle the receipt of erroneous requests.
  - A python backend that will handle image classification using a CNN.
  - A UI that will allow the user to upload an image to be analysed.
  - The UI will display information regarding the likelihood of each kind of possible defect.

  - To display the relevant images that fit the description of the most likely defects.
  - To display recourse information to rectify the defect.
  - Collecting, cleaning and pre-processing the image data.
  - Artificially grow the dataset by performing translations/rotations/adding noise to the images to make the training data more comprehensive.
- Should Have
  - A page to allow users to see a gallery of images sorted by defect type.
  - A page to allow users to see a gallery of images sorted by crop type.
  - The CNN should be able to classify at least 7 different defects across at least two different plant species.
  - The CNN should acheive at least 80% accuracy at classifying all different classes of defect in a held out test set that contains an equal number of each class.
  - Regularisation techniques to prvent the NN overfitting.
- Could Have
  - Ability for users to add additional information about the crop to determine the defect.
- Won't Have

3.12 Testing and Implementation details

3.13 Justification of Implementation Choices

Chapter 4 - Results and Discussion

4.1 Main Results

lorem ipsum

4.2 Evaluation Results

lorem ipsum

Chapter 5 - Conclusion

5.0.1 CNN

The Performance of the CNN given the available hardware (Google Colab GPU Runtime) is good. With a 98.6% classification accuracy on a held out validation set. With current cutting edge being 99.74% on the same dataset (as covered in the literature review). The CNN is capable of identifying 38 unique classes of crop images. Unfortunately due to the lack of data, one was unable to include additional defects such as lack of water/nitrogen/sunlight etc. However the design of the system allows for the CNN model to be changed very easily. As simply as changing two files on the API server. This means that as model performance improves and classification range broadens, the system is able accomodate for this.

5.0.2 Architecture

The seperation of responsibility between the API and Front-End has created two orthogonal systems that can be upgraded and interchanged easily. The front end need not know any details about the CNN(model) it is interacting with. The classes the model is able to identify are dynamically updated on the front-end and prediction data also remains consistent between model implementations. Recourse and prevention information are similarly easy to update and can be changed dynamically, simply by updating a JSON file on the back-end. Images the front-end display are also easily changed and are hosted on a seperate image server, further contributing to the systems orthogonality.

5.0.3 Development

A major change made to the design of the application was the choice to drop the relational database in favor of using heirachical storage as it was noted that the application did not need to handle complex relational data. Initially the purpose was to have the database handle filepaths to images and recourse/defect information but this was easily handled with consistent directory naming on the image server and JSON files.

Some of the main issues that arose during development were during the move to production. This involved learning new technologies such as NginX and Gunicorn to deploy the project as the configuration can be obtuse upon first interaction. Some API code also did not work the same way as on the development server and so had to be modified to work in the production environment.

Through the use of version control it has been straightforward to update the code on the production servers as it is simple as pulling the latest changes from the git repository. version control has been helpful throught the lifecycle thanks to the ability to branch the codebase (see version control).

REFERENCES

2021. URL https://yandex.com/images/.

ArabSat 5C - Internet by Satellite in Africa, 2021. URL https://www.globaltt.com/en/coverages-Arabsat5C_C.html.

best vegetables to grow - Explore - Google Trends, 2021. URL https://trends.google.com/trends/explore?q=bestvegetablestogrow&date=all&geo=US.

Digital Camera Market Share, Size, Trends and Forecast 2021-2026, 2021. URL https://www.imarcgroup.com/digital-camera-market.

File:Internet users per 100 inhabitants ITU.svg - Wikipedia, 2021. URL https://en.wikipedia.org/wiki/File:Internet_users_per_100_inhabitants_ITU.svg.

• Smartphone users 2020 — Statista, 2021. URL https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

G. Anthonys and N. Wickramarachchi. An image recognition system for crop disease identification of paddy fields in Sri Lanka. In *ICIIS 2009 - 4th International Conference on Industrial and Information Systems 2009, Conference Proceedings*, pages 403–407, 2009. ISBN 9781424448371.

Sungbin Choi. Plant identification with deep convolutional neural network: SNUMedinfo at LifeCLEF plant identification task 2015. Technical report.

Y. Le Cun, I. Guyon, L. D. Jackel, D. Henderson, B. Boser, R. E. Howard, J. S. Denker, W. Hubbard and H. P. Graf. 1989. Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning. *IEEE Communications Magazine*, **27**, 41–46. ISSN 01636804. (doi:10.1109/35.41400)

Cun Yann le. A Theoretical Framework for Back-Propagation, 1988. URL http://new.math.uiuc.edu/MathMLseminar/seminarPapers/LeCunBackprop1988.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. Technical report. URL http://image-net.org/challenges/LSVRC/2015/.

im Studiengang Bachelor, Betreuende Prüferin, Ulrike Steeens Zweitgutachter, Martin Behrmann geb Knoblauch and Eric Wohlgethan. Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung. Technical report.

Sachin D. Khirade and A. B. Patil. Plant disease detection using image processing. In *Proceedings - 1st International Conference on Computing, Communication, Control and Automation, ICCUBEA 2015*, pages 768–771. Institute of Electrical and Electronics Engineers Inc., jul 2015. ISBN 9781479968923.

Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. Technical report, 2012. URL `http://code.google.com/p/cuda-convnet/`.

Omkar Kulkarni. Crop Disease Detection Using Deep Learning. In *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*. Institute of Electrical and Electronics Engineers Inc., jul 2018. ISBN 9781538652572.

Gustav Larsson, Michael Maire and Gregory Shakhnarovich. may 2016. FractalNet: Ultra-Deep Neural Networks without Residuals. URL `http://arxiv.org/abs/1605.07648`.

Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**, 2278–2323. ISSN 00189219. (doi:10.1109/5.726791)

Sharada P. Mohanty, David P. Hughes and Marcel Salathé. sep 2016. Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, **7**, 1419. ISSN 1664-462X. URL `http://journal.frontiersin.org/article/10.3389/fpls.2016.01419/full`. (doi:10.3389/fpls.2016.01419)

Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus and Yann LeCun. dec 2013. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. URL `http://arxiv.org/abs/1312.6229`.

Karen Simonyan and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. Technical report, 2015. URL `http://www.robots.ox.ac.uk/`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Technical report, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015a.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. Going Deeper with Convolutions. Technical report, 2015b.

Smith J. Rosser A. Brown C. Schulte-Herbruggen B. Booth H. Sassen M. Mapendembe A. Fancourt M. Bieri M. Glaser S. Corrigan C. Narloch U. Runsten L. Jenkins M. Gomera M. Walpole, M. and J. Hutton. Enabling poor rural people to overcome poverty Smallholders, food security, and the environment. Technical report, 2013. URL `https://www.ifad.org/documents/38714170/39135645/smallholders_report.pdf/133e8903-0204-4e7d-a780-bca847933f2e`.

Zifeng Wu, Chunhua Shen and Anton van den Hengel. jun 2019. Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. *Pattern Recognition*, **90**, 119–133. ISSN 00313203. (doi:10.1016/j.patcog.2019.01.006)

Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. Technical report.

Heyan Zhu, Qinglin Liu, Yuankai Qi, Xinyuan Huang, Feng Jiang and Shengping Zhang. nov 2018. Plant identification based on very deep convolutional neural networks. *Multimedia Tools and Applications*, **77**, 29779–29797. ISSN 15737721. URL `https://doi.org/10.1007/s11042-017-5578-9`. (doi:10.1007/s11042-017-5578-9)

Appendix A - Project Proposal

Appendix B - Ethics Checklist