An Embedded Machine Learning System for Diseased Leaf Detection and Classification

Nha Do and Lucas He
Department of Electrical and Computer Engineering, University of California, Los Angeles.

***Abstract*—In recent decades, many applications have been developed to solve real-world problems, specifically in agriculture [1]. The U.S. apple industry is worth $15 billion annually but the apple orchards are under threat not only by pathogens, insects but also by incorrect and delayed diagnoses [2]. In this project, we focus on developing a solution to accommodate a computer-vision-based problem deployed on a digital signal processor, which has limited memory and performance, without cloud computing, to detect diseased leaf even at the early stage. Eventually, we came up with a 4 layers 2D CNN model which has been successfully implemented on the embedded system with 94.8% accuracy for detecting healthy leaves and classifying them into different diseased categories. We propose an effective approach to enhance the embedded machine learning model's accuracy by cropping input images to focus only on the particular patterns while ignoring other redundant information.**

***Index Terms*—Convolutional Neural Networks, Embedded Machine Learning, Diseased Leaf Detection, Image Processing, Plant Pathology.**

I. Introduction

In this project, our main goal is to detect a diseased leaf from a given input image and classify it into 2 disease categories: rust or scab if it is not healthy. The entire project is performed on the STMicroelectronics H7 Nucleo board which has only 1MB RAM [6]. Thus, we come up with a pre-processing technique when we cut out the redundant information on the original image to make it suitable with the H7 board and by using a lightweight convolutional neural network in deep learning [3] that outputs probabilities of different cases. We create a model such that there will always be one probability that dominates the rest, so we can identify the output of interest.

A. History

Apples are one of the most important temperate fruit crops in the world and their products are very popular and being used in daily activity. However, apple orchards in the United States are under threat from a large number of pathogens, insects, and other uncertainty factors [2]. Diseased leaves raise a major threat to the overall productivity and quality of apple's products [4]. It may be too late to help when the issue is recognized, but appropriate and timely diagnosis may be extremely important in preventing the problem. The current process of disease diagnosis is mainly based on human scouting. This is a set of several steps which sometimes require having deep knowledge and specific equipment for a correct diagnosis and it may include [5]:

1) Proper plant identification.
2) Recognize healthy plant appearance.
3) Identify characteristic symptoms.
4) Identify symptom variability.
5) Look for signs of biotic causal agents.
6) Identify plant parts affected.
7) Check the distribution of symptoms
8) Check for host specificity.
9) Lab tests.

Obviously, they are quite complicated for the majority of people or gardeners and are time-consuming or even expensive. More seriously, the misdiagnosis has caused significant impacts on agriculture and can lead to misuse of chemical compounds, increased input costs, even economic loss, and environmental impacts [4].

To address this kind of problem, computer vision-based models have shown promise for plant disease detection and classification and have been proved for convenience and reducing costs [1].

Thus, there are many online competitions for designing a useful machine learning model, and "Plant Pathology" is one of them. It has been organized by Kaggle continuously for a couple of years and the organizer has significantly increased the number of foliar disease images and added additional disease categories for each year [4].

The training dataset that we collected and used in this project is from the 2020 competition, which contains 1729 RGB images of 3 categories: healthy, rust, and scab.

B. Global Constraints

We use the STMicroelectronics H7 Nucleo board which has only 1MB memory constraint in total [6]. As a result, we must be mindful of the memory allocation required for the CNN model and its computation. Our collected inputs are the RGB images which contain a lot more information than gray-scale images and are too large for the H7's memory. We have to come up with a solution to reduce the input's size while keeping all the important information for the model to predict the output correctly.

With these constraints in mind, we understood that pre-processing is extremely important and could significantly affect the final predictions. Clearly, in order to create an embeddable model, the trade-off between performance and accuracy is inevitable, but our optimal solution will minimize it to be acceptable.

II. Motivation

The U.S. apple industry is worth $15 billion annually, which leads to millions of dollar losses due to the misdiagnosis or the time-consuming cost of human scouting [2]. Over the growing seasons, apple orchards are becoming the target of insects, fungi, bacteria, and other environmental factors causing the poor quality of fruit and huge economic losses. Incorrect or delayed diagnosis and treatment can lead to the rapid spread of diseases. On the other hand, one of the most popular solutions now is based on the computer vision technique, which has been proved to be very useful and promising for image processing, classification, and object detection [7]. Because of the explosion of computer science and image processing techniques, Artificial Intelligence (AI) is now being applied in many aspects of agriculture, and AI-based equipment and machine learning have taken today's agriculture system to a higher level and enhanced the quality of products by improving real-time monitoring, harvesting, processing, and diagnosing [8]. Furthermore, from a high level of image processing, there is one method from deep learning that has the most attention for computer vision's application, which is the convolutional neural networks (CNNs), because it is specifically designed for image data [7].

Our project addresses not only a single task of detecting diseased leaves but also proposes an effective approach to perform multi-tasks of detecting and classifying on the same digital signal processor using the combination of image processing and computer vision-based techniques. And from here, we can further develop other applications on the H7 board to solve more real-world problems. Success at this stage can lead to the implementation of such models into accessible technology such as phones or drones which can be used to sweep large areas of land and give early detection of possible worrying diseases.

Regardless of the embedded system's memory constraints, we wanted to build a very useful application that can detect and classify a diseased leaf even at the early stage of the disease.

Fig. 1. Examples of healthy, rust, and scab leaves

III. Approach

Here, we will go into detail about each group member's goal, the theories, the methods, and the strategies we employed to create a successful model.

A. Team Organization

The team structure would be divided into 2 parts corresponding to Python and C. Nha Do was in charge of creating a machine learning model, modified, optimizing a 2D CNN, and extracting the weights and biases in 16-bit floating-point such that it could fit into the H7 board. Nha pruned the model to make it smaller until the number of parameters is good enough for implementation while keeping the accuracy and performance as high as possible until it meets the goal. During the entire project, Nha had created at least 20 different models by tuning the hyperparameters, considering the number of training and testing sets, manipulating the learning rate, particularly using cropped images and data-centric algorithms to generate additional augmented samples. Eventually, three versions were reached: 22,000-parameter model, 17,000-parameter model and 10,000-parameter model. By considering the tradeoff between accuracy, memory, and speed, we decided to use the 17,000-parameter model. Lucas He was in charge of H7 implementation focusing on developing each layer, optimizing the code for memory allocation, and deploying effectively on 4 convolutional layers. This involved creating strategies to allow the H7 to support the models developed by Nha such as image pre-processing and flash read-in optimization. Lucas improved the model's biases in the dense layer by multiplying with a factor of 5 and dividing the intermediate output by 10 before using this result to calculate the final probability. By doing so, the overall accuracy was improved and some of the model's overturning was fixed.

Even though the tasks were distributed equally, during the project, we also worked together and proposed the new idea to help each other out for the mutual final goal.

B. Plan

The main skills required in this project are a background in machine learning, deep learning, computer vision, embedded system, digital signal processing, image processing, Python, and C programming. We collected and analyzed the training dataset provided by joining an online competition organized by Kaggle in 2020 [4]. However, the dataset is still being provided for free and the submission link is still available.

The roadmap of the project is expressed through the milestones that the team set at the beginning. By the third week, we would have a machine learning model completely built on the cloud with a target accuracy. By the seventh week, we would have the model deployed successfully into the H7 board and achieve the main goal of detecting if the leaf is healthy or not. We also considered the stretch goal of classifying the diseased leaf into appropriate categories and used the rest of the time (about 2 weeks) for reviewing and optimizing the entire project one more time.

However, our original plan was not suitable, so for the first 2 weeks, we got lost and wasted the time switching around between H7 board and the OpenMV Camera. This desire stemmed from the initial goal of utilizing the model in a maneuverable drone with an onboard camera. However, thanks to the meeting in week 3 with Dr. Briggs and TA Dezhan Tu to clarify the project, we finally got back on track as the table below.

C. Standard

Data visualization is one of the most important steps in the project to transform complex information into an easy-to-digest chart, thus we decided to use matplotlib and plotly express from the Python library. Matplotlib library is more popular and will be easily reproduced by other people [9]. On the other hand, Plotly Express performs beautiful and interactive charts with just a clean line of code [10]. A good understanding of the data will help us to have a better strategy to tackle the project.

We also used Keras Image Augmentation API to generate some additional augmented samples such as brightness adjustment, rotation, flip… to improve the training set because it is simple and powerful. Keras also provides the ImageDataGenerator class that defines the configuration for image data preparation and augmentation [11].

In addition, we used CNNs model, a deep learning algorithm, to solve our detection and classification problem because the preprocessing required in a CNNs is much lower as compared to other classification algorithms and while in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these filters/characteristics [2].

For embeddable weights and biases, we exported them into 16-bit floating-point types in a model during training to make it run faster and use less memory [12].

TABLE I
PLAN AND RESULTS

Plan	**Results**
Collect, extract, visualize and analyze the dataset in week 1	Joined a ‘Plan Pathology’ competition on Kaggle, accepted their rule, and downloaded the dataset.
Create a CNN model with the target accuracy (above 90%) on the cloud by week 3	The actual accuracy we achieved at this time is only about 40%.
Have the model deployed successfully on H7 for at least the main goal of diseased leaf detection by week 7	We finally achieved the target accuracy by tuning some hyperparameters, adding more additional augmented samples, and specifically by cropping the training dataset. The model was also successfully deployed on H7 but there were still some inconsistencies and overflow values (NaN values in prediction)
Debug, improve, optimize the model and work toward the stretch goal of classifying diseases by week 9	We figured out the problem of inconsistency is because of the defective flash drive, so we ordered a new one and fixed that bug. Up to this point, we were deploying 2 separate models at the same time in the H7 chip. The model’s biases were also adjusted to avoid overflow and fortunately, by doing so, we eventually combined successfully 2 models into 1 to improve the performance and achieved both main and stretch goals.

D. Theory

We analyze the influence of pixels by using the 3x3 kernel, and the number of kernels is set up such that it could fit into the H7 board without overflow. This is the main reason why we chose to use only 16 kernels at the first 2 convolutional layers because the input size at these layers is pretty large. The filters will move from the top left corner in the same way, across the rows then move down the columns until it reaches the bottom right corner. For each point on the image, a value is calculated by multiplying and adding the

corresponding weights and biases [13]. The same padding technique is also used.

Following each convolutional layer is the max-pooling layer which is responsible for reducing the spatial size of the Convolved Feature [2]. This also decreases the computational power required to process the data and we can increase the number of filters to 32 in the third and fourth layers.

We tried both ReLU and sigmoids as the activation function, but after many tests, we realized that the ReLU is more efficient. The ReLU is easier to compute and combats the vanishing gradient problem occurring in sigmoids [13].

There are also many ways to quantize the model effectively. We explore that the 16-bit floating-point is simple and results in the expected halving of memory usage.

E. Software/ Hardware

All of the data pre-processing is performed on Google Colab using Python so that we can take advantage of Google's GPU and all team members can access the notebook. The training dataset is also uploaded on Google Drive and can be easily extracted from Google Colab.

We use the NumPy, Pandas, Matplotlib, Plotly, and Cv2 libraries for data cleaning, data analysis, and image visualization. These libraries are often used and can be efficiently stored and loaded for either training or testing. Tensorflow and Keras are used for creating data augmentation, creating and training the model. TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs used for easily building and training models, but Keras is more user-friendly because it's built-in Python [14]. The framework was developed by Google Brain. After training, the model weights and biases are saved in text files using NumPy.

The model is deployed on an STMicroelectronics H7 Nucleo board (STM32H743ZI2) microcontroller, which has 2 MB of flash memory, and 1 MB of SRAM [15]. The CubeIDE is used for implementing the 2D CNNs model in C. The model weights and biases are stored on a USB Flash Drive that can be accessed through a user USB connector on the board.

F. System build/ Operation

The first thing is to explore our dataset. Our raw dataset is not balanced, we have 4 categories with 1820 images but the multiple diseases case consists of only 91 images. Due to the memory constraint and the complexity of RGB image, the 2D CNNs itself is not good enough for classifying this case, we must have to use "imagenet" feature to train the model, but it will be impossible to deploy into H7 because the number of parameters will be very large (up to millions). We have decided to get rid of this case to obtain a balanced dataset and thus, we are able to achieve the target accuracy.

The most important step in our preprocessing is that we cropped the training images such that the new dataset only focuses on a very particular spot on each leaf such that the model can recognize those patterns and detect or classify them

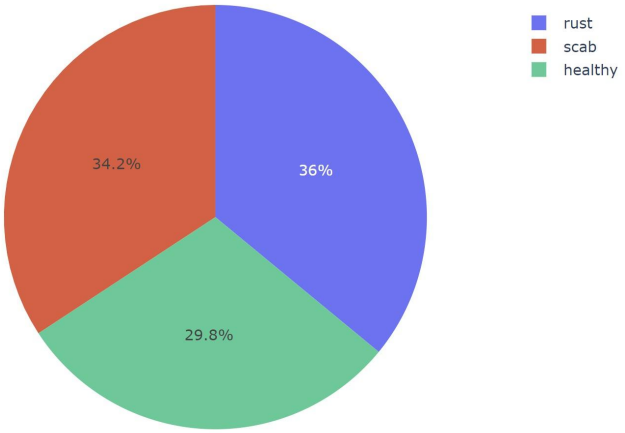

Fig. 2. The distribution of each category

in lieu of using the raw images which contain a lot of redundant information. The reason for that is because the H7 board can only read the input up to 48x48 in size, hence if we resized the raw images, they would be very blurry and the model cannot detect the pattern correctly. By doing this, the input image will be scaled small enough to train the model.

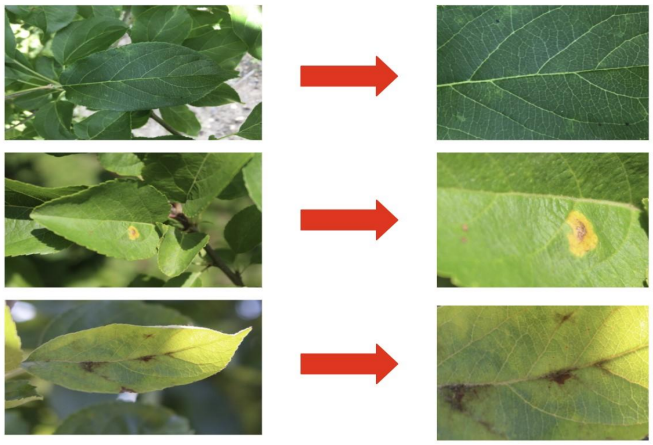

Fig. 3. Original images (left) vs cropped images (right)

When the new dataset is ready, we split it into 80% for training and 20% for validation.

The next step is to use data-centric augmentation to generate additional augmented samples by randomly adjusting brightness, rescaling, rotating, zooming, flipping vertically or horizontally, and shifting. The random transformation is necessary for augmenting the training dataset and making the model's learning more nuanced and more diverse. Then, we generated batches of datasets containing the images and labels, which will be used later to train. Besides setting up the training and validation generator, we also used bilinear interpolation in which it considers the closest 2x2

neighborhood of known pixel values surrounding the unknown pixel. It then takes a weighted average of these 4 pixels to arrive at its final interpolated value [16]. This results in much smoother images.

The model we used consists of four blocks of convolutional and max-pooling layers followed by two fully connected layers (including the output layer). The activation used is ReLU.

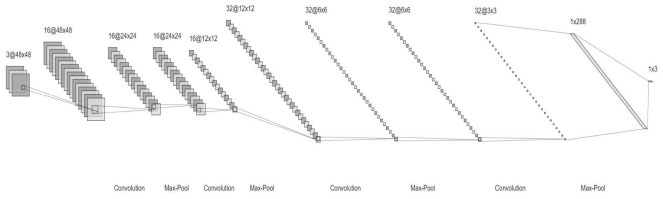

Fig. 4. 17k-parameter model architecture (created on http://alexlenail.me/NNSVG/LeNet.html).

The model was trained by using the 'Adam' optimizer and 'Categorical Cross Entropy' loss for up to 100 epochs.

For H7 deployment, we first had to implement our model in C, which contains 2D convolution (with ReLU activation), 2D max pooling, and fully connected layers with ReLU and softmax activation. We also loaded the weights and biases of the model as well as the test images from the USB flash drive during run time. Note that, since we trained the model with cropped images, the test images also need to be cropped. The input images are also needed to be converted to .bmp to save the memory and during the run time, we cannot load all of them at once due to memory constraints. Furthermore, as we are working with color images, the computations are more complicated and the output values could overflow which leads to NaN values. To fix this, we adjust the biases by multiplying with a factor of 5 and dividing the intermediate output by 10 before using this result to calculate the final probability. To read the .bmp images, we also wrote the function which can read RGB images (note that RGB images have 3 channels instead of 1 for Grayscale). During the run time, we load one image at a time from the file. Once the process is completed, the weights and biases are loaded and the input image will be passed through 4 blocks of the model, which will output its prediction of whether the input leaf is healthy or not, if not so what kind of disease it is (rust or scab). Our model almost reaches the upper limitation of H7's memory, so during the entire project, we always have to keep track of the memory usage. Even the array length changes by 1 could lead to a mess up. After each layer's computation, the free command needs to be used to remove the arrays that have completed the task.

IV. Results

A. Description of Results

In the end, we have decided to use the 17,523 parameters model with 94.8% accuracy.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 16)	448
max_pooling2d (MaxPooling2D)	(None, 24, 24, 16)	0
conv2d_1 (Conv2D)	(None, 24, 24, 16)	2320
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 16)	0
conv2d_2 (Conv2D)	(None, 12, 12, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 32)	0
conv2d_3 (Conv2D)	(None, 6, 6, 32)	9248
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten (Flatten)	(None, 288)	0
dense (Dense)	(None, 3)	867

Total params: 17,523
Trainable params: 17,523
Non-trainable params: 0

Fig. 5. Model Architecture

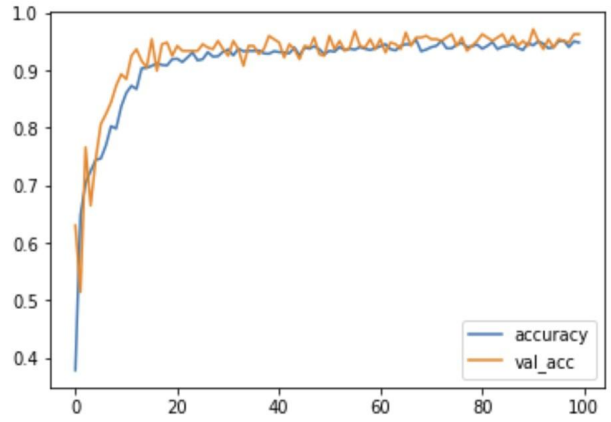

Fig. 6. Model Accuracy

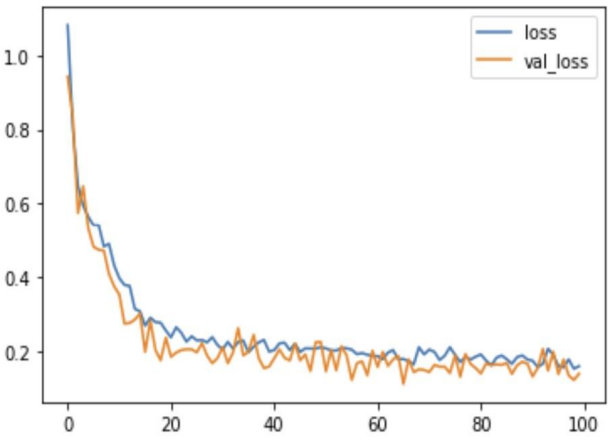

Fig. 7. Model Loss

For the testing on H7, we use the 3 images below (Healthy, Rust, and Scab in order). Note that the second image is not healthy but a rusty leaf. There are at least 2 small spots on the leaf. The model is only 80% confident in this case because it looks like a healthy leaf but it also points out that the model works pretty well in detecting the diseased leaf even at the early stage.

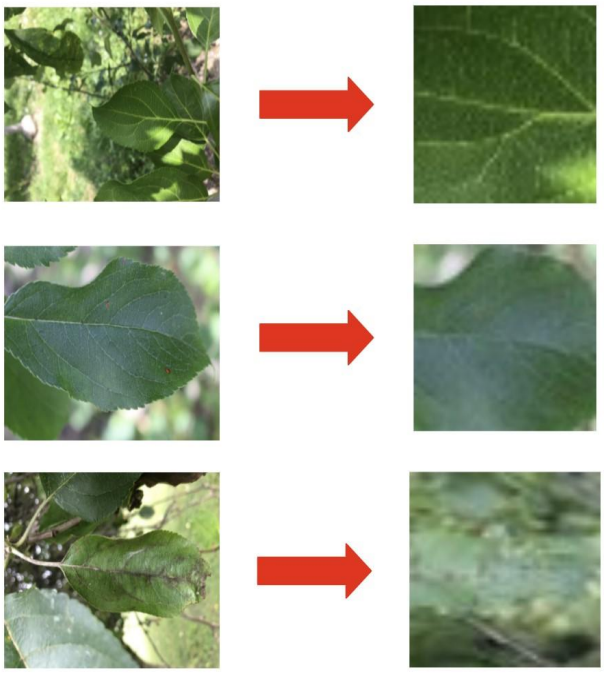

Fig. 8. Healthy, Rust, and Scab used for testing

```
prediction of case Healthy1.bmp being Healthy is 0.999999999487436
prediction of case Healthy1.bmp being Rust is 0.000000000000000
prediction of case Healthy1.bmp being Scab is 0.000000000512564

prediction of case Rust1.bmp being Healthy is 0.200961813306612
prediction of case Rust1.bmp being Rust is 0.799038181003867
prediction of case Rust1.bmp being Scab is 0.000000005689521

prediction of case Scab1.bmp being Healthy is 0.000000000000018
prediction of case Scab1.bmp being Rust is 0.000000000000000
prediction of case Scab1.bmp being Scab is 0.999999999999982
```

Fig. 9. Final predictions

The real runtime for each prediction is less than 10s which meets our original goal of less than 30s

B. Discussion of Results

Our training dataset is very imbalanced, the multiple diseases are only 5% on the entire set, thus, under the global constraint, we cannot create a good enough model to detect this category. Lowering the number of 3 other categories to make the dataset balanced is even worse overall. Therefore, we came up with a solution to get rid of this category to ensure the high accuracy of the remaining parts. If we used the original image (without cropping), we cannot achieve the target accuracy although we used the larger embeddable model (22k parameters).

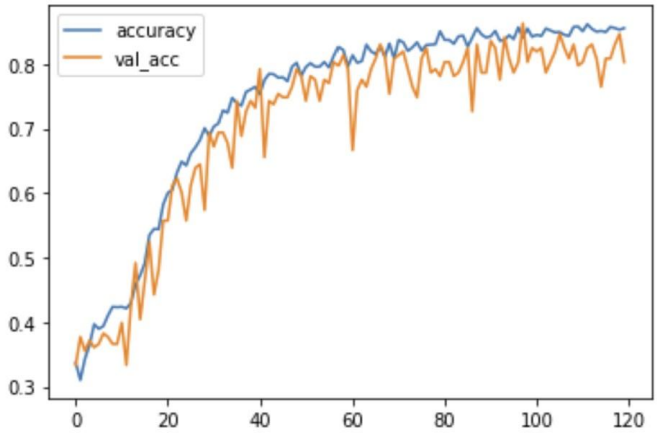

Fig. 10. Model accuracy using original images

The accuracy, in this case, is only 85.59% and the loss is pretty high as well (0.42).

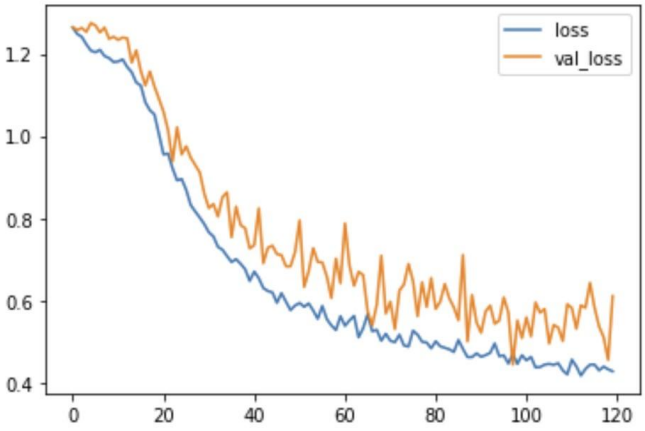

Fig. 11. Model loss using original images

This is because the original images contain redundant information and just a few blocks of a 2D convolutional layer are not enough to handle. On the other hand, the original dimension is 1365x2048, if we resized it to 48x48, the input would be very blurry and the model would be very confusing. Hence, the solution of cropping the images is suitable and in fact, it helped us to improve the performance. The downside of this solution is that if we crop a wrong image's edges, the prediction will be incorrect. However, to make it embeddable in the H7 board, this can be considered as an acceptable solution.

In essence, we can see that the project was a success. Although we got rid of one of the categories, we still ensure that it can handle the main task of detecting and classifying other diseases. The 17k parameter is a nice sweet model between accuracy and performance.

Since we still use a high number of parameters, if we are not careful in memory allocation, the overflow can occur at

any point. To address this problem, we can use the idea of the 3D CNN model and try to quantize the model to in8 which consumes less memory. Also, for a limited memory, if we can increase the training set to make it better and more diverse, it will be a sweet solution to enhance accuracy with fewer parameters. At this time, we have not successfully implemented the new idea, but we think that the result would hold and even better for the more complicated techniques.

References

[1] Jain, Pravar, "Artificial Intelligence in Agriculture : Using Modern Day AI to Solve Traditional Farming Problems", *Analytics Vidhya*, 4 November, 2020, https://www.analyticsvidhya.com/blog/2020/11/artificial-intelligence-in-agriculture-using-modern-day-ai-to-solve-traditional-farming-problems/

[2] Saha, Sumit, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", *Towards Data Science*, 15 December, 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] Thapa, Ranjita, et al. "The Plant Pathology Challenge 2020 data set to classify foliar disease of apples", 28 Sept, 2020, https://bsapubs.onlinelibrary.wiley.com/doi/10.1002/aps3.11390

[4] "Plant Pathology 2020 - FGVC7", *Kaggle,* https://www.kaggle.com/c/plant-pathology-2020-fgvc7

[5] Riley, Melissa, Margaret Williamson, Otis Maloy, "Plant Disease Diagnosis", *APS*, https://www.apsnet.org/edcenter/disimpactmngmnt/casestudies/Pages/PlantDiseaseDiagnosis.aspx

[6] STMicroelectronics, "STM32 Nucleo-144 development board with STM32H743ZI MCU, supports Arduino, ST Zio and morpho connectivity", *ST*, https://www.st.com/en/evaluation-tools/nucleo-h743zi.html

[7] Brownlee, Jason, "A Gentle Introduction to the Promise of Deep Learning for Computer Vision", *Machine Learning Mastery*, 26 March, 2019, https://machinelearningmastery.com/promise-of-deep-learning-for-computer-vision/

[8] Talaviya, Tanha, et al. "Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides", *Science Direct*, https://www.sciencedirect.com/science/article/pii/S258972172030012X#:~:text=AI%20is%20an%20emerging%20technology,et%20al.%2C%202007).

[9] Araujo, Ismael, "Matplotlib vs. Plotly Express: Which One is the Best Library for Data Visualization?", *Towards Data Science,* 3 July, 2021, https://towardsdatascience.com/matplotlib-vs-plotly-express-which-one-is-the-best-library-for-data-visualization-7a96dbe3ff09

[10] Meinert, Reilly, "Plotly Express: the Good, the Bad, and the Ugly", *Towards Data Science*, 14 June, 2019, https://towardsdatascience.com/plotly-express-the-good-the-bad-and-the-ugly-dc941649687c#:~:text=One%20of%20the%20coolest%20features,how%20something%20changes%20over%20time.

[11] Brownlee, Jason, "Image Augmentation for Deep Learning With Keras", *Machine Learning Mastery*, 29 June, 2016, https://machinelearningmastery.com/image-augmentation-deep-learning-keras/

[12] "Mixed precision", *Tensorflow*, https://www.tensorflow.org/guide/mixed_precision

[13] Stewart, Matthew, "Simple Introduction to Convolutional Neural Networks", *Towards Data Science*, 26 Feb, 2019, https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

[14] Terra, John, "Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework", *Simpli Learn*, 4 Mar, 2022. https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20is%20an%20open%2Dsourced,because%20it's%20built%2Din%20Python.

[15] "NUCLEO-H743ZI2", *Arm Mbed*, https://os.mbed.com/platforms/ST-Nucleo-H743ZI2/

[16] "DIGITAL IMAGE INTERPOLATION", *Cambridge In Colour*, https://www.cambridgeincolour.com/tutorials/image-interpolation.htm