Machine Learning on the Edge: Diseased Crop Detection using Neural Networks and Model Reduction Techniques

James N. Pillot IV

Department of Electrical Engineering

Stanford University

jpillot@stanford.edu

Abstract

Successfully identifying crop diseases is an important task for our food production systems. Current computer vision techniques and machine learning algorithms present promising solutions to this problem but would benefit greatly from being able to run on resource limited embedded platforms. We create two different CNN models to identify plant disease before reducing their size in memory via conversion to a TensorFlow Lite model and Integer Quantization and comparing their performance to the SqueezeNet model trained using transfer learning. Both models we created and trained from scratch outperformed SqueezeNet in accuracy, size (memory used), and average time required to classify an image.

1. Introduction

Robots in agriculture are deployed to perform tasks ranging from automated dispensing of weed killing products to crop harvesting and sorting. Many of these tasks employ a blend of computer vision and machine learning to successfully identify crops and other plants of interest. Given that the computers in robots and automated tractors run on limited resources relative to the high-performance computing clusters that some machine learning models are trained on, this presents the challenge of bringing machine learning to a resource constrained environment.

We will investigate building a low memory, computationally inexpensive convolutional neural network (CNN) to successfully identify diseased crops with computer vision. The outcome will be a model small enough to run quickly in a resource constrained environment, such as a microcontroller, that takes in images of various plant leaves and determines if they are diseased and if so, identifies the type of disease.

2. Related Work

Previous work has been done with the Plant Village data set including a study performed by the authors themselves! In [1], AlexNet and GoogLeNet architectures are trained from scratch as well as applied using transfer learning (fine tuning the existing models) to identify diseased crops. In addition to different architectures, different pre-processing methods were tested that included converting images to grayscale and removing the background. All pre-processing methods involved cropping the images to be 256x256. The results were that GoogLeNet with transfer learning training, leaving the images in color with no background removal, and an 80/20 train/test split of the data performed the best with an overall accuracy of 99.34%. Additional previous work has been detailed in our proposal.

3. Dataset and Features

As mentioned in [1], The Plant Village Data set [2] is a collection of "54,306 images of plant leaves, which have a spread of 38 class labels assigned to them. Each class label is a crop-disease pair...". Some examples of the 38 class labels are, Apple-Apple Scab, Apple-Healthy, Grape-Black Rot, Grape-Leaf Blight, Grape-Healthy, and Cherry-Powdery Mildew.

Figure 1: Example of Leaf Images from Plant Village Data set. Each photo represents a class label. Taken from [1]

The dataset was split into training/validation/test with 34,525 images for training, 14,796 images for validation, and 4,985 images for test. This represents a 64/27/9 split. The unconventional numbers came from the way the test set was separated. We wanted to make sure the distribution of test data was representative of all the samples present without one category dominating, so we pulled away ~130 images per class (plus or minus one due to human error in dividing them) and performed a 70/30 train/validation split on the remaining data using TensorFlow [8].

After splitting the data, the RGB images were reshaped to be 256x256x3 and 227x227x3 for our two models and SqueezeNet respectively. SqueezeNet is 227 rather than 224 in accordance with the Keras implementation as seen in [9]. For our two models, the pixel values were rescaled from 0-255 to 0-1.

4. Methods

The first of the two CNN models, referred to as “Model 1”, we developed has the architecture shown in Figure 2. Model 2 has the architecture also shown in Figure 2. Both models were trained with the Adam optimization algorithm and used TensorFlow’s “SparseCategoricalCrossEntropy” loss function, an implementation of cross entropy for multiple class prediction [10]. Both models also follow a conv2d -> max pool back-to-back approach for multiple layers. The conv2d layers learn the parameters of convolution filters for feature extraction and the max pool layers help reduce the total number of parameters in the network (which is desired given our original goal of minimizing model size) and potentially reduce overfitting. After flattening the output of the convolutional layers, we go through a fully connected layer with a ReLu activation function (chosen due to its proven ability to perform well in fully connected layers), and finally an output layer with a SoftMax activation function with 38 neurons that return the probability the input image belongs to each class. The neuron with the highest output helps us label the input image. Inspiration for the architecture of both models came from the examples provided by the TensorFlow team [8].

Model 2 is 5 layers deeper than Model 1 to assess the performance/size tradeoffs of different models, but the architecture structure is similar for both.

After train/dev/test evaluation, both models were quantized via conversion to a TensorFlow Lite model as well as with Integer Quantization for evaluation on the test set. Conversion to a TensorFlow Lite model switches the original model to a flat buffer file format, greatly reducing its size in the process [11]. Integer Quantization involves converting weights and activations expressed as 32-bit floating point values to 8-bit integer values. Given the number of weights and activations in Model 1 and Model 2, this leads to significant reductions in model size on top of the already reduced size from being made into a TensorFlow Lite model [11].

Finally, SqueezeNet and its original weights were imported with an additional 128 neuron fully connected layer and a softmax output layer to evaluate its capabilities when used for transfer learning with this task. SqueezeNet is a CNN architecture developed by Iandola et al. at DeepScale, UC Berkeley, and Stanford [12]. The model, post compression, takes up less than 0.5MB and achieves 80.3% Top-5 ImageNet accuracy. Design decisions that help accomplish solid performance despite relatively few parameters were replacing many of the 3x3 filters with 1x1 filters, decreasing the number of input channels to the 3x3 filters that remain, and delaying down sampling to happen later in the network by concentrating layers with a stride size greater than 1 to be towards the end of the network [12].

As seen in Stanford's CS 230 Deep Learning class, the Adam optimization algorithm minimizes a loss function and leverages four hyperparameters: learning_rate, beta_1, beta_2, and epsilon. The default parameters that Keras provides of learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, and epsilon = 1e-07 were used since we trusted Google's TensorFlow team's recommended selections. The cross-entropy loss function used as seen in CS 230 is provided in Figure 3.

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	8388736
dense_1 (Dense)	(None, 38)	4902

Total params: 8,417,222
Trainable params: 8,417,222
Non-trainable params: 0

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 256)	295168
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 256)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 512)	8389120
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 38)	9766

Total params: 8,922,822
Trainable params: 8,922,822
Non-trainable params: 0

Figure 2: Model 1 Architecture (Left) and Model 2 Architecture (Right)

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$

Figure 3: Cross Entropy Loss Function

5. Experiments/Results/Discussion

First a discussion of more hyperparameters. A batch size of 64 was chosen because it was a power of 2 and fell in line with the recommendations provided by Kandel and Castelli in [13].

The default learning rate from Keras of 0.001 was also used as recommended by [13], a small batch size should correspond to a small learning rate. Each model was trained for 5 epochs to keep training time manageable in relation to performance.

The primary metrics for each model were test set accuracy, model size, and the average time to classify an image from the test set. Results are seen in the table below:

Model	Train Accuracy	Dev Accuracy	Test Accuracy	Model Size	Average Time to Classify
Model 1	96.2%	88.4%	81.3%	96.5 MB	0.008 seconds
Model 1 TF Lite	X	X	81.3%	32.9 MB	0.009 seconds
Model 1 Integer Quantization*	X	X	80.6%	8.2 MB	X
Model 2	95.9%	91.3%	86.5%	102 MB	0.009 seconds
Model 2 TF Lite	X	X	86.5%	34.9 MB	0.014 seconds
Model 2 Integer Quantization*	X	X	89.2%	8.7 MB	X
SqueezeNet	67.8%	68.3%	48.4%	30.8 MB	0.010 seconds

*The average time to classify for Integer Quantization was left out because these models were meant to run on an embedded device and not a PC. For the inference time to be reasonable (since these models would "hang" when ran on the PC), we had to test on just ten batches (~13% of the test data).

A confusion matrix for Model 1 is in Figure 4. Model 2 and SqueezeNet followed a similar trend.

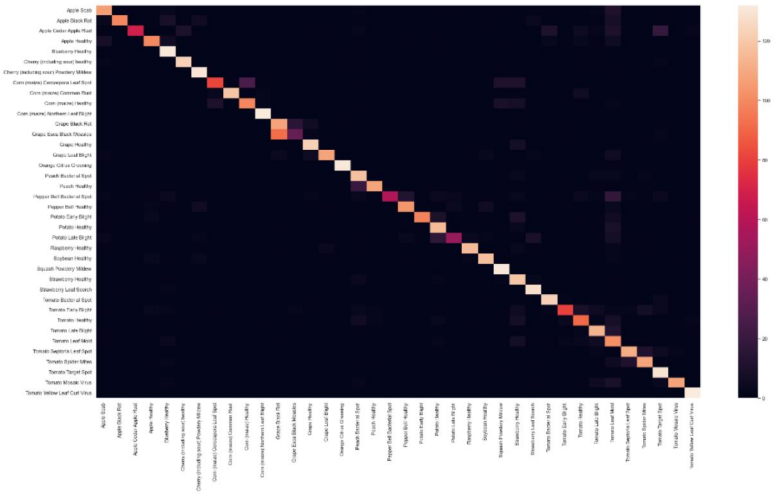

Figure 4: Confusion matrix for Model 1

Pre quantization, Model 1 is smaller with a lower test accuracy and Model 2 is larger with a higher test accuracy. Their classification times are similar. Post quantization, both models are reduced to similar sizes with Model 2 continuing to have a higher test accuracy. SqueezeNet performs significantly worse on all relevant metrics.

For all three models, the classification task they struggled with the most was differentiating between the grape leaves with black rot and the grape leaves with black measles. A side by side of the two leaves is shown in Figure 5.

If we were deploying these models to the field to be used real time on an embedded device, we would go with Model 2. Model 2 maintained the highest accuracy before and after quantization while still being a competitive size with the poorer performing Model 1 and having quick inference. We believe this result makes sense given the original depth of Model 2 compared to Model 1 and the effectiveness of the quantization methods used to reduce model size. More depth (to an unknown point) yields better accuracy while not necessarily hurting the ability of the quantization methods to work effectively.

The poor performance of the SqueezeNet model with imported weights shows the necessity of retraining the model for this task since the crop data probably differs severely from the ImageNet data.

Finally, the confusion between the grape leaves with black rot and the grape leaves with black measles falls in line with how similar the leaves look as seen in Figure 5. When inspecting by hand, we also struggled to differ between the two. More data would probably need to be gathered for these categories to rectify the shortcoming of the models.

Figure 5: Black rot on the left, black measles on the right

Based on the loss function plots seen in [17] we may be on the cusp of overfitting our data for Model 1 and Model 2 (if we had run for 6 epochs instead of 5) since the plots show the validation data beginning to plateau and increase. In future runs, using 4 epochs instead of 5 may be considered and evaluated.

6. Conclusion/Future Work

In conclusion, Model 2, the deepest convolutional neural network performed the best while being compressed to a model size that fits easily on many embedded devices and running an acceptably quick inference time.

With more time we will put this model in a resource constrained environment, such as a Raspberry PI or Arduino, and properly evaluate the inference time and accuracy of the Integer Quantized Model 2. We would also like to train an even deeper CNN with higher accuracy that can still be quantized down to < 10 MB. We believe we may be able to get the test accuracy into the 90s while keeping the quantized model small and fast with a similar architecture and just a few more layers.

7. Contributions

As a team of 1, James Pillot completed this work in consultation with Vincent Li, the course TA. Thank you for your help, Vincent!

References

[1] Sharada P. Mohanty, David P. Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 2016.

[2] Sharada P Mohanty. Spmohanty/plantvillage-dataset: Dataset of diseased plant leaf images and corresponding labels, Aug 2016.

[3] Konstantinos P. Ferentinos. Deep learning models for plant disease detection and diagnosis. Computers and Electronics in Agriculture, 145:311–318, 2018.

[4] Pranesh Kulkarni, Atharva Karwande, Tejas Kolhe, Soham Kamble, Akshay Joshi, and Medha Wyawahare. Plant disease detection using image processing and machine learning, 2021.

[5] Puranjay Mohan, Aditya Jyoti Paul, and Abhay Chirania. A tiny CNN architecture for medical face mask detection for resource-constrained endpoints. In Lecture Notes in Electrical Engineering, pages 657–670. Springer Singapore, 2021.

[6] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, page 1935–1944. JMLR.org, 2017.

[7] https://github.com/JamesPillot/CS230_FinalProject

[8] Team, Keras. "Keras Documentation: Image Classification From Scratch." *Keras*, 27 Apr. 2020, https://keras.io/examples/vision/image_classification_from_scratch/

[9] Malli Refik, keras-squeezenet, (2017), GitHub repository, https://github.com/rcmalli/keras-squeezenet

[10] Tensor Flow Team. "Tf.keras.losses.SparseCategoricalCrossentropy: Tensorflow Core v2.9.0." *TensorFlow*, 21 May 2022, https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

[11] "Post-Training Integer Quantization: Tensorflow Lite." *TensorFlow*, Google, 26 May 2022, https://www.tensorflow.org/lite/performance/post_training_integer_quant#convert_to_a_tensorflow_lite_model.

[12] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

[13] Kandel, Ibrahem, and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset." *ICT express* 6.4 (2020): 312-315.

[14] Brownlee, Jason. *Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models and Work Projects End-to-End*. Jason Brownlee, 2016.

[15] SqueezeNet Tensor Flow Library https://github.com/Tandon-A/SqueezeNet

[16] Confusion Matrix Plotting https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why/

[17] Code Repository: https://github.com/JamesPillot/CS230_FinalProject