

# A resource-efficient deep learning framework for pest and disease detection in plants.

Tejasvi S Tomar

tejasvi18@iitg.ac.in

## Summary

*The rapid growth of crop production has made early pest and disease detection particularly important in terms of efficiency and reduction of the use of pesticides. Conventional methods involving human experts are nonefficient for large crops. The recent advances in computer vision using deep learning can play an important role in increasing the accessibility and reliability. We apply a modified CNN based on MobileNetv2 to the task of pest and disease identification. We investigate various augmentation techniques to make the model more suitable for field conditions. However, the challenge here is that the performant deep learning models often require large computation and storage for mobile solutions. To overcome, we experiment with various optimization strategies to achieve a sufficiently lightweight implementation for Android OS. We use the dataset PlantVillage for disease classification which has 54K images and we achieve the accuracy of 96.3%. For pest classification we use the IP102 dataset having 75K images and achieve the state of the art accuracy of 64.8%.*

## 1. Introduction

Pests and diseases are among the two most important factors affecting agricultural yields. Types of pests and diseases are numerous in nature. Some of them are similar in appearance, making it difficult for farmers and even experts to precisely recognize them in time. Detecting the early presence of pest and disease plays a critical role in minimizing yield loss if contained swiftly.

The conventional practice relies on naked-eye observations of the experts and gathering statistics, which, however, has many shortcomings, such as needing a large amount of labor, low efficiency, feedback delay, and prohibitive costs. Further, in developing regions, farmers may have to go long distances to contact experts; this makes consulting experts too expensive and time-consuming. Automatic detection would avoid those problems and provide the benefits of continuous observation and improved accessibility.

In recent years, using machine learning for image recog-

nition has rapidly evolved into an area of significant activity surpassing human capabilities. Deep learning has emerged as the most successful approach based on neural networks that act as function approximators. However, most of the high performing models require computational power and storage beyond the capacity of mobile devices. Therefore the final model is usually optimized using various techniques to make it less resource hungry. In theory, the complexity of the CNN model depends on the scale of dataset. Large datasets generally require large number of parameters to capture the maximum amount of features required for classification. The space required for model storage is so large that they cannot be deployed on mobile devices with little memory with low transfers speed. To solve this problem few lightweight architectures like MobileNets, SqueezeNet, EfficientNet, ShuffleNet, etc have been proposed. They have comparatively few parameters and have special structures like inverted residual connections to minimize computations.

We use a modified architecture based on MobileNet v2 which is a simple but effective network architecture which is an improvement over original MobileNet. A custom head composed of linear, batch norm, and drop out layer is used. For the trunk we use pretrained weights for ImageNet to speedup the training using transfer learning.

For disease classification we use PlantVillage which includes 15 common type of diseases. It has 20K samples making it a relatively complete image dataset for the diagnosis of plant diseases. However, most of the samples are shot in ideal conditions with plain background. To make the model more applicable in field conditions and to reinforce model generalization ability, we experiment with image augmentations which significantly reduce the similarity between generated images.

The pest classification uses the IP102 dataset which has 102 species of pest in their natural conditions. Within each category there is a high intra-class variation due to distinct forms of insects in their life cycle. This brings the dataset size to 75K samples and benefits little from augmentation. However, the number of samples for each category is highly uneven requiring extended training with rare categories.

We also employ various techniques to make training more interpretable. Since IP102 samples are heterogeneous, one of the primary concern is model may learn the surroundings instead of the pest characteristics for prediction. To monitor the features learned by the model we use Grad-CAM heatmaps proposed by Selvaraju K, et al.[9]. We also employ confusion matrix to finetune the model for less performant classes.

For model optimization, we consider quantization, pruning, and distillation individually and the combinations thereof. For quantization we test quantization aware training, dynamic, and static quantization with the goal to understand accuracy and model size tradeoff. We also experiment with pruning the individual layers of the model and iterative pruning.

This suggests a question: how much the size of model can be reduced while maintaining sufficient accuracy? Generally the rate of accuracy drop increases with increased optimization. However, there are few cases which demonstrate counterintuitive increase in model performance like shown by Polino. et al.[7]

## 2. Related Work

Recently, detecting diseases and pests in crop-fields has become a hot topic. More and more farmers, corporations, and researchers are working on pest and disease detection, which is regarded as an important tool in precision agriculture. Currently, there are two types of approaches for image-based detection. One approach uses traditional image processing techniques and requires considerable domain expertise; while the other is based on deep learning.

Although there is no lack of number of datasets, the size of most of them fall short for application using deep learning. For pests recently a large dataset, IP102 was collected and analysed by Wu et al., 2019[11] and it contains around 75K pest images corresponding to 102 types of pests. Since then it has been used as a benchmark dataset for researchers. Another such dataset, PlantVillage (Hughes et al., 2015)[4] contains 54K healthy and unhealthy leaf images divided into 38 categories by species and disease. However, unlike IP102 which contains pests in natural environment, the images in PlantVillage are well-shot in plain background.

Training support vector machines on hand-crafted features was the common approach, especially before the rise of deep learning. It was still competitive on small datasets also after the advent of neural networks until improvements in hardware allowed to train deeper neural networks. Xie et al. (2016)[12] used a spatial pyramid pooling with sparse coding to identify farmland pest images and had recognition accuracy of pest images with background better than previous methods. However almost all such method require complex pretreatment which has considerable effect on final performance.

Deep neural networks have proved to be best performing solutions, and many developments have lead to achieve SOTA invariably. For the task of image classification, the most advanced models heavily rely on convolution neural networks. (Dawei et al., 2019) attempt to classify pests using CNNs using a pretrained version of AlexNet (Krizhevsky, Sutskever and Hinton, 2012) using transfer learning. They also evaluated the accuracy rate of six human experts on the same dataset for baseline comparison, and reach a higher accuracy than four of the six human experts.

Due to high resource requirements of most DNN architectures, several high performing architectures have been proposed. Among the recent, ShuffleNet (Zhang et al., 2018) uses channel shuffle and  $1 \times 1$  convolutions convolutions to have a very low complexity and is a very light network. The result is that the network is 13 times faster than AlexNet, maintaining similar accuracy.

MobileNetV2 (Sandler et al., 2018) is another light convolutional network specifically designed for mobile applications. It contains several depthwise separable convolutions, which can be thought as convolutional layers where the 3D weights tensor is factorized in one 2D tensor and a 1D tensor, requiring much less memory. It also contains unusual inverted residual connections which reducing the number of required operations.

It has been observed that irrespective of architecture complexity, substantial optimization can be done with negligible loss in accuracy. Quantization aware training along with pruning (Han et al, 2015)[2] has demonstrated 49x size reduction of VGG-16 with upto 4x layerwise speedup.

Generalized by Hinton et al., 2015, distillation provides a straightforward way to generate compact models. A large and complex network or an ensemble model is used to train a small network by minimizing a loss function in which the target is the distribution of class probabilities of larger model. with the help of the cumbersome model. This small network will be able to produce comparable results, and in some cases, it can improve the model generalization as shown by Polino et al, 2018 [7].

## 3. Data

For pest classification we use IP102 released by Wu et al., 2019[11] which contains images of pests in their natural conditions. Whereas for disease classification we use PlantVillage released by Hughes et al., 2015[4] containing leaf images against plain background.

### 3.1. Pest classification

Most of the datasets released in past typically contain fewer than 1,000 samples. Moreover they have plain backgrounds and similar orientations which is hard for CNN to generalize, making it difficult to cope with the complexities

of real-life scenes. However the recently proposed dataset by Wu et al., 2019[11] IP102 overcomes all previous limitations. IP102 covers 102 common insect pest species for practical applications and is built in the wild.

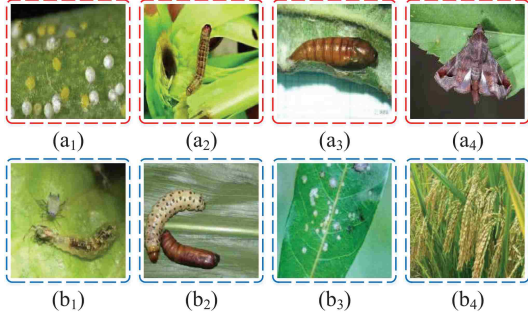


Figure 1. Different forms of insect pest images. The red dashed boxes denote different forms of pests, containing (a1) egg, (a2) larva, (a3) pupa, and (a4) adult, which belong to the same subclass. The images surrounded by blue dashed box are dropped because there are no or more than one insect pest category.

The least represented class in the dataset only has 71 samples, while the most represented class contains more than 5740 samples, hence the dataset is highly unbalanced as shown in Figure 2. For more on how we attempted to address this class imbalance, see the Methods section. It also worth mentioning that, in this dataset, insects of very different life stage are included in every class, which is a challenging problem since the shape of an insect, like a butterfly, might change completely during its life. Some examples are shown in Figure 1.

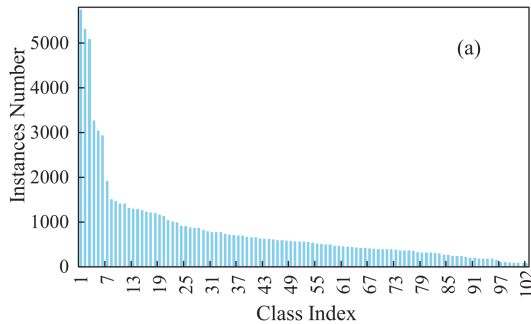


Figure 2. Sample number distribution of the IP102 dataset in different levels.

We use around 45K images for training, 8K images for validation, and 22K images for testing.

### 3.2. Disease classification

For disease classification we use PlantVillage released by Hughes et al., 2015[4] which contains healthy and diseased images of individual leaves. The dataset contains 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, and we make an attempt to predict both the crop and the disease given just the image of the plant leaf. Figure 1 shows few examples each from different crop-disease pair from the PlantVillage dataset.

There exist three different versions of the whole PlantVillage dataset; color, gray-scaled, and background-less. However we only use the PlantVillage dataset as it is, in color since we expect it to provide better results in field conditions. While training, however, there is the potential to introduce some inherent bias in the dataset due to the regularized process of data collection in case of PlantVillage dataset. For more on how we attempted to address this, see the Methods section.

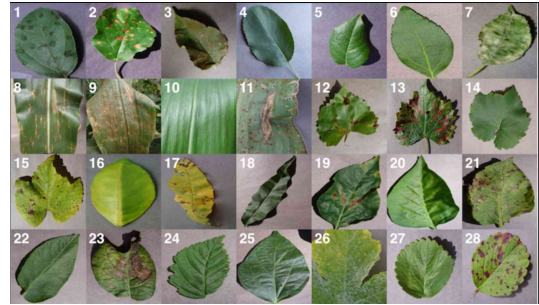


Figure 3. Example of leaf images from the PlantVillage dataset, representing the different crop-disease pair used.

We use around 40K images for training, 5K images for validation, and remaining 9K images as the test split.

## 4. Methods

Our strategy for solving this classification problem was to use transfer learning from ImageNet, rather than training a CNN from scratch. We chose this strategy because this allows faster prototyping and experimentation. We were bolstered in this decision by the multitude of successful case studies of transfer learning from ImageNet (see the Related Works section).

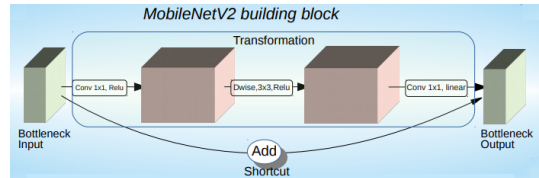


Figure 4. A schema for the MobileNet v2 architecture.

The particular ImageNet classifier which we decided to use was MobileNet V2, a network released by Google in 2018[8]. The architecture of MobileNet consists of two important pieces: *Inverted residual structure*, where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use ex-

panded representations in the input. The lightweight depth-wise convolutions serve to filter features in the intermediate expansion layer. Another feature is the absence of nonlinearities in the narrow layers in order to maintain representational power. This improves performance and the original publication provides an intuition that led to this design, which creates “shortcuts” in the model that seem to facilitate backpropagation. We chose MobileNet v2 also because of its world-class performance on the ImageNet among compact architectures challenge, scoring a top-five accuracy of 90.5%.

```
(1): Head(
  (0): AdaptiveConcatPool2d(
    (ap):
      ↪ AdaptiveAvgPool2d(output_size=1)
    (mp):
      ↪ AdaptiveMaxPool2d(output_size=1)
  )
  (1): Flatten()
  (2): BatchNorm1d(2560, eps=1e-05,
    ↪ momentum=0.1, affine=True,
    ↪ track_running_stats=True)
  (3): Dropout(p=0.25, inplace=False)
  (4): Linear(in_features=2560,
    ↪ out_features=512, bias=True)
  (5): ReLU(inplace=True)
  (6): BatchNorm1d(512, eps=1e-05,
    ↪ momentum=0.1, affine=True,
    ↪ track_running_stats=True)
  (7): Dropout(p=0.5, inplace=False)
  (8): Linear(in_features=512,
    ↪ out_features=15, bias=True)
)
```

A schema for the head added to the MobileNet v2 architecture.

The MobileNet v2 architecture without its head provides 2560 features. In the final stage of the classifier, a custom head is attached to map the features to the number of prediction classes. We experiment with batch norm, linear, and dropout layers for the head. The sequence and the number of parameters is validated using the validation split. Finally, we would use a softmax loss function to convert the output scores into normalized “probabilities.”

Since the pretrained model accepted input of size 224x224 we do resizing of the training samples during the training. An alternative approach we considered was multi-cropped inputs. However after initial testing the payoff of high training times was not enough to pursue it further. In the case of the disease and to some extent pest classification problem, our dataset suffered from a class imbalance. As mentioned in the previous section, our largest category,

had roughly 80x as many images as our smallest category. Class imbalances sometimes impede the training of machine learning algorithms, since the algorithm might decide to disregard the smaller classes if they don’t contribute significantly to the cumulative loss. Our class imbalance is not nearly as extreme as that encountered in some datasets. In medical imaging, for example, a dataset might contain hundreds or thousands of examples of “healthy” images for each example of a “diseased” image. Still, we wanted to explore techniques for mitigating the impact of this class imbalance in case it impeded training.

As a first step, we decided to create a “confusion matrix” for the classification problems, to determine if our smaller classes were failing to train properly. If fear was frequently being confused for some larger class, it would point to problems in our training pipeline. As we will see in the Experiments section, the confusion matrix also serves another purpose, which is to suggest which categories are “similar” to each other according to the algorithm. We were curious to discover if the categories which we expected to be similar based on human intuition were actually similar according to the algorithm, or whether the pipeline was being confounded by errors which we couldn’t interpret.

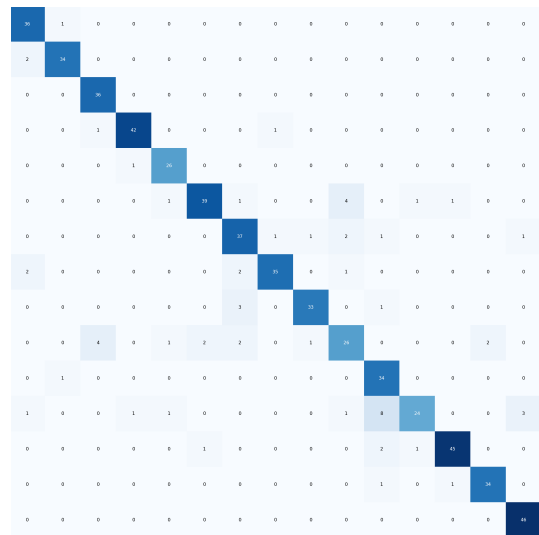


Figure 5. The confusion matrix for subset of categories used in disease classification. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. The shade intensity represents the prediction confidence.

Two primary approaches have been proposed to deal with class imbalances: oversampling and undersampling[1]. In the case of oversampling, the smaller classes are “oversampled” in the training phase, to increase their effect on the gradient updates. This might mean that a particular image is included twice in the same batch update. Oversampling carries the risk

that the algorithm will overfit the smaller classes, since the same image from a rare class might contribute to the gradient update a large number of times. In undersampling, by contrast, the excess images from a common class are randomly discarded. The problem with undersampling is that lots of information about common classes is lost.

Undersampling did not seem like a viable approach for us, since our smallest class had just a few hundred images, so we decided to try oversampling. To mitigate the dangers of overfitting, instead of reusing the same images from our smallest categories, we decided to try *image augmentation*. Image augmentation consists of a set of standard computer vision techniques for creating a similar image from a base image. This might mean taking random crops of the image, or rotating it, or flipping it on its axis.

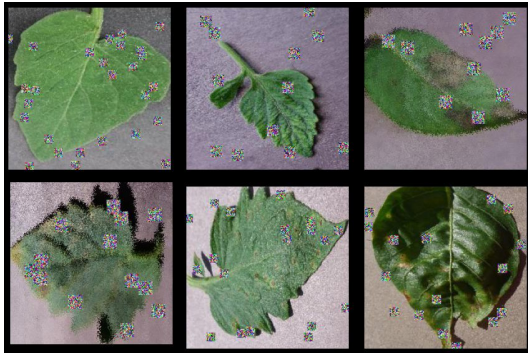


Figure 6. Augmented sample of images taken from PlantVillage. The augmentations depicted are rotation, jitter, and the gaussian noise.

We felt that we should be cautious in which augmentation techniques we applied, since certain techniques seemed likely to change the category depicted by an image. We did multiple test runs for viable augmentations. We found improved results with *image blurs*, *image rotation*, *image flips* and random generated patches of *gaussian noise*. While deep learning has facilitated unprecedented accuracy in image classification, one of their biggest problems is model interpretability, a core component in model understanding and model debugging. In practice, deep learning models are treated as “black box” methods, we ask how can trust can be established better? Some aspect of it is answered by Gradient-weighted Class Activation Mapping, or more simply, Grad-CAM proposed by Selvaraju et al.[9]

Grad-CAM uses the gradients of any target concept, flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept. Using Grad-CAM, we visually validate where our network is looking, verifying that it is indeed looking at the correct patterns in the image and activating around those patterns.

For optimization we first consider quantization of the model to 8-bit precision. We first apply dynamic quantiza-

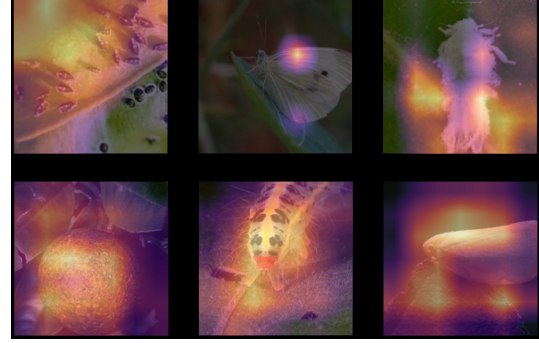


Figure 7. Coarse heatmap generated for missclassified samples from IP102 using Grad-CAM. The brighter regions represent where the network is “looking” for the object.

tion to the previously trained model. After getting baseline accuracy loss, we apply static quantization and as expected observe improved accuracy and greater size reduction. Finally we do quantization aware training of the model and achieve accuracy almost equal to the non-quantized model.

In case of pruning we first do sensitivity analysis for each layer of the model. Then we iteratively identify least sensitive layers and prune the 10% of weights. After each pruning we finetune the model to recover the critical features lost to pruning. We repeat this procedure till we achieve 50% reduction of the model size. We further use quantization aware training along with pruning and achieve 75% size reduction with minimum loss in accuracy.

Finally we investigate distillation of the previously pruned and quantized model. The student model having one-fourth the number of parameters in the parent model is trained against distillation as well as prediction loss. The student model consists of linear and convolution layers for simplicity. We see modest compression results with 30% loss in accuracy. However, considerable improvement is expected if ensembles are used as the parent model.

## 5. Experiment

Our first step was to evaluate the performance of a simple linear subnetwork tacked on the end of MobileNet v2. We trained this model with an Adam optimizer and use 0.9 as the momentum determined using validation. The weights are initialized using Kaiming initialization as proposed by Kaiming He et al[3] for faster convergence. The weight decay is handled using the method proposed in *Fixing Weight Decay Regularization in Adam* by [5] proposed by Loshchilov et al, 2018. (We also tried training with a simple SGD optimizer, and this produced equivalent results, although it required more epochs to train.)

For training, we use *One cycle policy* proposed by Leslie Smith, 2018[10]. First the learning rate is progressively increased and at the same time the momentum parameter

is progressively decreased. After reaching the limits, the learning rate is progressively decreased and at the same time the momentum is increased back to its original value. Then the learning rate is further decreased while keeping momentum constant.

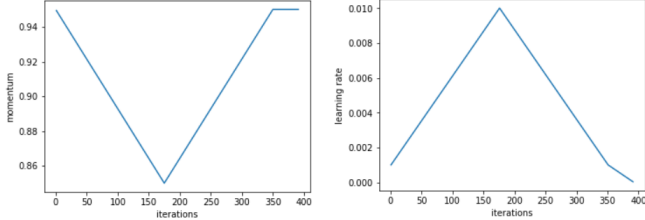


Figure 8. In the 1 cycle policy, learning rate is increased to its peak and is subsequently decreased. The reverse happens with the momentum parameter.

The optimum learning rate to use with the 1cycle policy is found using a learning rate finder. It does a mock training by going over a large range of learning rates, then plot them against the losses using the validation data. We then pick a value slightly lower than the rate with minimum loss, where the loss still improves. Our graph would look something like in the figure.

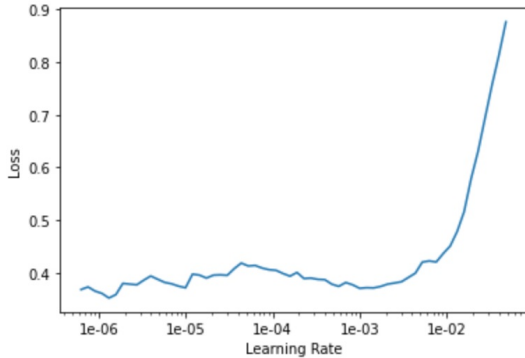


Figure 9. Finding optimal learning rate for training. Start with a very low rate and change it at each mini-batch until it gives a very high loss. The losses are plotted against the learning rate to find the optimal value before it diverges.

After fitting the head of MobileNet v2, we unfreeze the inner layers to finetune the weights. Since the initial layers are highly sensitive compared to end layers we use differential learning rates. The layers of architecture is subdivided into 4 groups, each of which learns at the rate relative to the location in the model. During the training the learning rate is reduced by a factor of 0.2 if validation loss doesn't improve for the two successive epochs. We achieve the accuracy of 96.3% for disease classification on PlantVillage dataset. We achieve 64.8% accuracy for pest classification using IP102 and improve the previous state of the art by 4.6% which used ensembles of DenseNet.[6]

| Classifier                       | Accuracy on test set |
|----------------------------------|----------------------|
| Linear head only                 | 86.62%               |
| Full architecture                | 96.3%                |
| 8-bit Quantization               | 96.1 %               |
| 50% Pruning                      | 95.9 %               |
| 50% Pruning + 8-bit Quantization | 95.7%                |
| Distillation                     | 70.2 %               |

Figure 10. Classification accuracies on the test set of PlantVillage for different strategies.

To optimize the model, we experiment initially with dynamic and static quantization. Further we do quantization aware training in 8-bit precision along with the iterative pruning. The ideal pruning ratio is determined to be 50% with negligible loss in accuracy. In process we reduce the model size by 75% to approximately 4 MB. However for the task of pest classification we observe sizable loss in accuracy for certain categories after the optimization. The loss can be explained by huge imbalance of the IP102 dataset and larger number of categories (102) compared to the only 38 categories for the disease classification.

| Classifier                       | Accuracy on test set |
|----------------------------------|----------------------|
| Linear head only                 | 50.1%                |
| Full architecture                | 64.8%                |
| 8-bit Quantization               | 64.3%                |
| 50% Pruning                      | 62.3%                |
| 50% Pruning + 8-bit Quantization | 61.78%               |

Figure 11. Classification accuracies on the test set of IP102 for different strategies.

As an attempt to compress the model even further we use distillation for the disease classification model. The student model uses 94% lesser space than the non-optimized original model. As expected we observe significant loss in accuracy upto 26.7%. However using ensembles as for the parent model is expected to give better results.

## 6. Conclusion

We tried to determine the applicability of various optimization methods on the problem of pest and disease classification. We also demonstrate that transfer learning from ImageNet could produce a classifier with high accuracy with minimal training. We use the same architecture for two tasks: first, determine plant category and the disease from the leaf image; and second, determine the pest species using the images from various stages of insect life cycle. The classifier proved to be quite effective at the disease classification problem, but was less effective at the pest classification problem. Previous works suggest that classification

of pests in natural environment is hitting the limits even for complex networks like DenseNet. We achieve higher accuracy using much more resource efficient method which makes the solution deployable on mobile devices. We also explore various augmentation techniques for application in practical use cases. To improve the model interpretability as opposed to the black box nature of neural networks, we use Grad-CAM as a method to visualize the decision process. Finally we experiment with a few combinations of optimization strategies using quantization, pruning, and distillation. We reduce the model size by 75% without noticeable loss in accuracy.

For future work, we suggest that more data needs to be gathered, particularly for the classes with negative valence. The means by which data is gathered also need to be adjusted. Instead of taking images of leaves with plain background and similar orientation, it is more useful to have samples in natural conditions. Currently quantization is being heavily researched and it could be possible to use lower precision upto 2-bits in near future without significant loss in accuracy. In case of pruning we can experiment further using heuristic algorithms such as *Automated Gradual Pruner* for more specialized pruning.[13] Distillation is a promising optimization technique and use of ensembles during training can provide substantial compression with low accuracy loss. Our preliminary results are sufficiently promising that we believe that, if model is trained properly on suitable data, mobile deployment through various optimization methods is an achievable goal.

## References

- [1] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29, June 2004.
- [2] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [3] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [4] D. P. Hughes and M. Salath'e. An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. *CoRR*, abs/1511.08060, 2015.
- [5] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. 2018.
- [6] L. Nanni, G. Maguolo, and F. Pancino. Insect pest image detection and recognition based on bio-inspired methods. *Ecological Informatics*, page 101089, 2020.
- [7] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.
- [8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [9] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.
- [10] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1—learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.
- [11] X. Wu, C. Zhan, Y.-K. Lai, M.-M. Cheng, and J. Yang. Ip102: A large-scale benchmark dataset for insect pest recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8787–8796, 2019.
- [12] C. Xie, R. Li, W. Dong, L. Song, J. Zhang, H. Chen, and T. Chen. Recognition for insects via spatial pyramid model using sparse coding. *Transactions of the Chinese Society of Agricultural Engineering*, 32(17):144–151, 2016.
- [13] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.