UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Disease Detection and Classification in Crops Using Mobile-Oriented Deep Convolutional Neural Networks

Diego Xavier Matheu Vasco

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 15 de mayo de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Disease Detection and Classification in Crops Using Mobile-Oriented Deep Convolutional Neural Networks

Diego Xavier Matheu Vasco

Nombre del profesor, Título académico **Noel Perez, Ph.D.**

Quito, 15 de mayo de 2022

© DERECHOS DE AUTOR

Nombres y apellidos: Diego Xavier Matheu Vasco

Código: 201564

Cédula de identidad: 1804270732

Lugar y fecha: Quito, 15 de mayo de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practices for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Cada año casi un tercio de los cultivos se pierde frende a pestes y enfermedades. Las tecnicas modernas de detection para estos problemas se basan en laboratorios y requieren de espacios, equipos y especialistas dedicados. Estos métodos son relativamente caros en tiempo y dinero. Estas limitaciones de las tecnicqas actuales hacer que el diagnostic temprano y el pronto tratamiento para mitigar perdidas menos probable. En años investigacion reciente, modelos redes neuronales convolucionales han sido desarrollados para deteción de objetos y clasificación de imágenes con resultados casi perfectos. En este trabajo proponemos modificar y entrenar la arquitectura por excelencia de detecion de objetos en su ultima versión (YOLOv4) y la arquitectura por excelencia de clasificación de imágenes para mobiles en su ultima versión (MobileNetV3) en un set de datos dedicado a imágenes de plantas enfermas y sanas. Se compilo un dataset combinando los datasets de PlantVillage y PlantDoc para tener todas las imagenes posibles en un dataset balanceado que combina imágenes tomadas en campo con imágenes en ambientes controlados. Entrenamos, ajustamos y comparamos los resultados de ambos modelos usando distintas técnicas. Para la arquitectura de YOLO, se intentó aprovechar las características del aprendizaje transferido para entrenar la inteligencia artificial desde otra ya entrenada mientras que se entreno y ajusto la arquitectura de MobileNet desde cero. Sorprendentemente, la arquitectura más simple de MobileNet fue capaz de clasificar imágenes de mejor forma que YOLO pudo identificarlas.

Palabras clave: Inteligencia rtifial, aprendizaje prefundo, redes neuronales, redes convolucionales profundas, epidemiología digital.

ABSTRACT

This paper aims to test and develop deep learning models dedicated to mobile deployments in order to build a tool that helps growers and farmers diagnose diseases and pests in crop plantations. Every year almost a third of planted crops is lost to pests and diseases. The modern techniques to detect diseases and pests are laboratory-based and require dedicated spaces and professionals and are considerably expensive. These limitations of the currently used techniques make early diagnosis and early treatment to mitigate losses less likely. In recent research convolutional neural network models have been developed for object detection and image classification with near-perfect results. We propose the training of the go-to object detection architecture YOLOv4 and the go-to image classification MobileNetV3 in a dedicated dataset. We compiled a new dataset combining the augmented PlantVillage dataset with the PlantDoc dataset to have as many images as possible in a well-balanced dataset that combines images taken on-field and in a controlled environment. We trained and compared both models and their respective mobile dedicated variations and compared the results. We tried to leverage YOLO's transfer learning-focused architecture to train a model that already has weights while building and fine-tuning the MobileNet model from the ground up. Surprisingly, MobileNet's simpler architecture was able to perform better-classifying images than YOLO's bounding box object detection.

Keywords: Machine Learning, Plant Disease Detection, Digital Epidemiology, DCNN, Computer Vision, PlantVillage, Deep Learning

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCTION

The Population Reference Bureau in its latest World Population Sheet estimates that the current 7.8 billion people will turn into more than 9.6 billion by 2050 (PRB, 2021). For this to happen responsibly, humankind has to increase its food production by about 60 percent according to (FAO, 2018). Researchers have shown in (Savary et al 2019, pp 430-439) that, of the world's crop production, almost one-third is lost to pathogens and pests.

Nowadays farmers and growers rely on their knowledge and experience. They use books and the internet as tools to evaluate symptoms and try to diagnose but the losses are still significant. Besides this "Do It Yourself" method, there is also a variety of laboratory methods (Fang & Ramasamy, 2015 pp 537-561)but they require special professionals, a dedicated space, and implements. These alternatives are expensive in time, effort, and money. These limitations make on-field diagnosis impossible at times and early treatment less likely and show the need for a mobile tool that can be used by farmers and growers by themselves without relying on (Fang & Ramasamy, 2015 pp 537-561).

The catalyst for developing a mobile solution using deep learning models to detect diseases in plants was made in 2015 with the development of the platform and subsequent dataset PlantVillage (Hughes & Marcel, 2015). This dataset was used for the first time to train deep Machine Learning architecture, household name GoogLeNet, reaching an accuracy of 99.34 percent with the optimal hyperparameters in (Mohanty et al, 2016, p 7) showing that a deep learning solution is more than possible. The next step in solving this problem is making this solution mobile and accessible.

Referring to mobile solutions, it is worth mentioning that developed countries are already reaching over 80 percent of smartphone ownership according to (Pew Research Center, 2021).

The increasingly widespread use of smartphones brings the processing power needed to deploy previously trained large neural networks to a consumer-level application.

In recent research, the main concern for a possible mobile application was first tackled, image variability. The two-branch variant of the Inception-V3 architecture preserved accuracy over several types of noise trying to replicate real-world image variability(Schuler et al, 2021, pp 375-381). Moreover, some researchers have been done on other architectures but few have been done to develop a dedicated model for mobile application as shown in (Saleem et al, 2019, p 8).

In this work, we propose a mobile tool for the early diagnosis of crop pests and diseases. It is based on the well-known YOLO architecture and MobileNet architecture, specifically YOLOv4 and MobileNetV3. Both are the latest version with published and peer-reviewed results of each architecture. The deep learning models automatically identify damage or deformation on the leaves and classify the corresponding pest or disease that caused the anomaly. With this tool, it will be possible to help farmers and growers control their crops, make a quick on-field diagnosis if needed, and facilitate early treatment to reduce the percentage of crops lost and end the dependency on expensive diagnosing techniques. Ultimately, making food production more efficient and sustainable for future generations

.

MATERIALS AND METHODS

Database

The starting database (Arun & Geetharamani, 2019) we used for this research is known as PlantVillage. It is the largest public dataset for plant disease classification. The latest version can be found on (Arun & Geetharamani, 2019). The dataset has been studied and improved upon in (Arun & Geetharamani, 2019) and in (Singh et al, 2019). In the latest version from (Arun & Geetharamani, 2019), the dataset now has 61,486 images that include healthy leaves, different categories of unhealthy leaves, and background images totaling 39 categories. This dataset is larger than the original due to data augmentation techniques to replicate real-world variability and 3 new categories. In \(Singh et al, 2019), also recognizing the possible bias in the PlantVillage dataset for the controlled environment pictures, an alternative/complementary dataset was compiled with a combination of lab, field, and free online images. We combined the (Singh et al, 2019) dataset with the (Arun & Geetharamani, 2019) dataset to get the best dataset for the real-world useful training. The difference between both datasets can be seen in figure \ref{databaseComparison}.

Figure 1. Comparison between PlantVillage dataset and PlantDoc dataset

In Figure 1 we can see sample images to illustrate the difference between real-world images from (Singh et al, 2019) (first row) versus controlled environment images from the original PlantVillage dataset (Arun & Geetharamani, 2019) (second row). Plant and diseased by column from left to right: apple scab, corn rust, potato early blight, and tomato bacterial spot.

Deep Learning Models

Artificial intelligence (AI) can be defined as a form of intelligence displayed by a computer. Symbolic AIs have inputs that they can process through fixed rules, algorithms developed by programmers, to produce a determinable output (Russell & Norvig, 2021). For complex problems, with no fixed rules to solve, Alan Turing introduced the concept of machines that can learn on their own (Turing, 1950 pp 433-460). This concept can be further explained in contrast to symbolic AI. In Machine Learning (ML), programmers introduce data but, instead of the rules, they introduce the expected answers for said data, and the AI's work is to deduct the rules needed to process the data into the expected answers, hence learning(Chollet, 2017).

In Machine Learning there are different techniques and algorithms, known as models, best suited to different applications. Neural networks are models that adjust the parameters of a

sequence of functions to take an input X the closest to the desired output Y(Russell & Norvig, 2021). They are called neural because they are loosely based on neuroscience and networks since the functions are layered and each layer of functions is connected and feeds the next. Deep Learning refers to the number of layers or depth of the ML model. A neural network is considered deep when it has at least three layers: An input layer, hidden layers whose use will be determined by the AI, and finally the output layer. (Goodfellow et al, 2016)

Convolutional neural networks (CNNs) implement at least one layer that applies the convolutional operation, an operation on two functions of an argument. The second function that distinguishes this operation is called the kernel. It can counteract noise in the input data, but it is used to transform the shape of the input tensor (multidimensional array), extracting relevant features (Goodfellow et al, 2016). These neural networks have been tremendously successful in processing grid-like data such as images and have been proven continuously in the ImageNet challenge(Chollet, 2017). They are now considered the go-to algorithm for image processing (computer vision).

YOLOv4

YOLO is a category of deep convolutional neural networks (DCNN(that classify parts of images into categories and generates bounding boxes around them. The latest stable version with published results is YOLOv4\cite{Bochkovskiy_2020}. This architecture starts with the input layer. Then a backbone section focused on feature extraction with convolutional layers using the CSPDarknet-53 architecture. Then, the neck section uses PAN and SPP architecture and mainly has max-pooling layers to collect feature maps and is densely connected (Huang et al, 2016) with the next layers. Finally, the head is the YOLOv3 algorithm (Redmon & Farhadi, 2018) that works as the object detection part of YOLOv4 and gives the final output. The authors

of YOLOv4 also released a reduced version that only has 24 convolutional layers into a faster but not so accurate version called YOLOv4-tiny made for embedded and mobile devices.

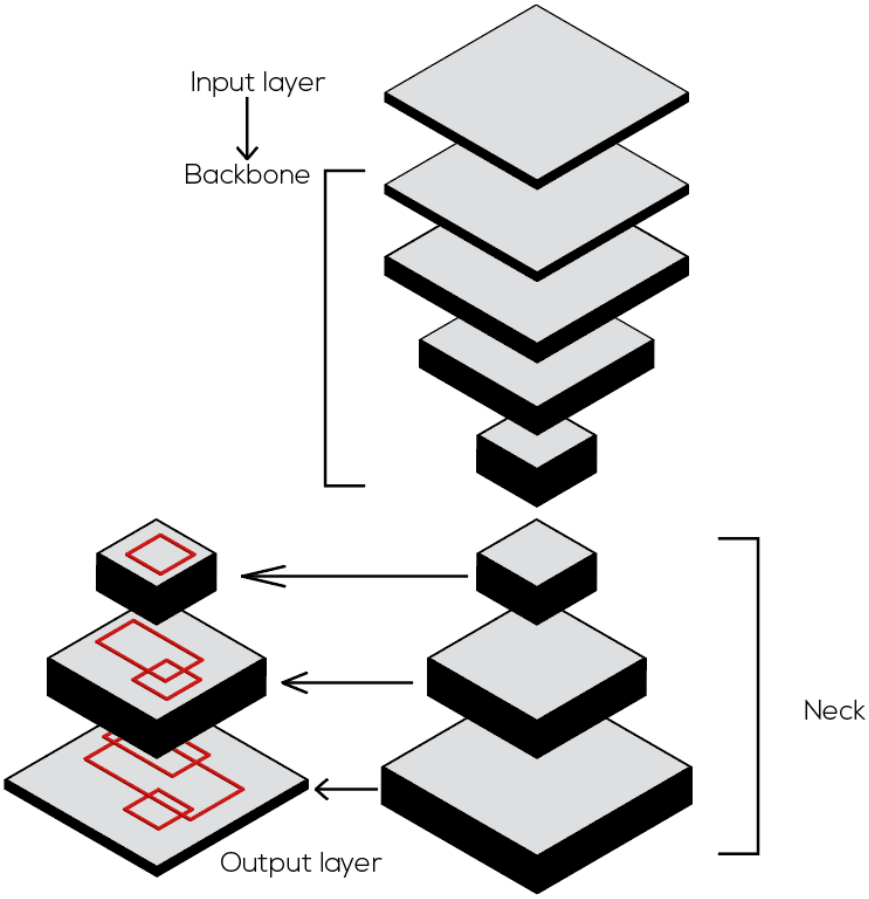

Figure 2. graphic representation of the YOLOv4 architecture

MobileNetV3

This architecture in comparison is simple. It is also a DCNN architecture designed for fast performance in embedded and mobile devices. Its architecture is solely based on convolutional layers with different sizes and activation functions for feature extraction. It only has a backbone network, namely MobileNetV3, for this. (Howard et al, 2019)

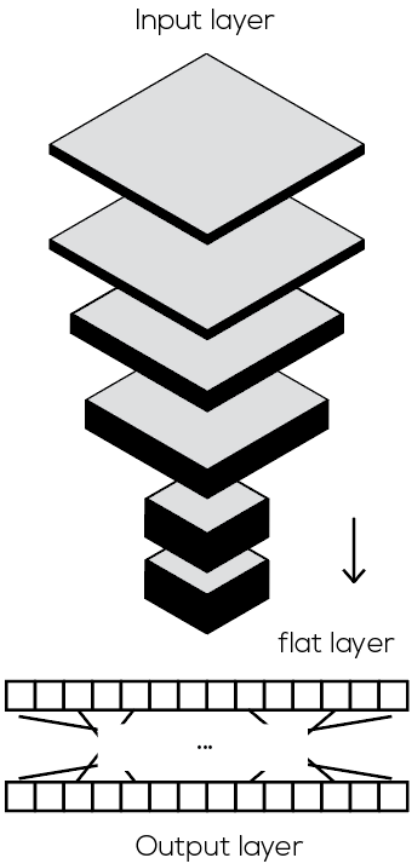

Figure 3. graphic representation of the MobileNet architecture

Transfer Learning

Transfer learning is the hypothesis that deep learning models can benefit of previous training. When a model is trained, the weights of the activation functions in the various layers get updated as the model learns and these weights can be saved. When a model is trained, it is possible to upload saved weights of previous training so the learning starts from a pre-trained model. Most modern models such as the ones we are using here support some form of transfer learning. The YOLOv4 only has 1 weights file that is periodically updated by the creators of the architecture. The MobileNet architecture, on the other hand, has multiple published weights of the model trained in different datasets.

Since the various weights available for transfer learning in MobileNet are from general-purpose datasets like ImageNet and COCO unlike ours and previous research is done in (Singh et al,

2019) shows transfer learning may negatively affect results with this architecture, we decided not to use transfer learning for the MobileNet models and train it from the ground up. Since we do not have similar research for the YOLO architecture and the DCNN community's consensus seems to be that using the most up to date weights published is the rule of thumb, we trained the YOLO architecture using transfer learning but keeping in mind that some research suggests this will affect negatively the results as stated above.

Experimental setup

To prepare the datasets, configure, train, and test the models, a .ipynb file with all the algorithms needed to do so was created for each model. This format enabled us to run the models on different platforms with relative ease while giving us the ability to use the Jupyte Notebook IDE.

Data pre-processing

First, both datasets had to be combined by joining (Singh et al, 2019) images into (Arun & Geetharamani, 2019) for those categories present in both. The remaining images in (Singh et al, 2019) were negligible and were discarded since those classes would have been severely underrepresented. For the MobileNet model, all images had to be normalized to the 224 by 224 pixels images this particular model expects. This model also expects images to be organized into folders by category.

For the YOLOv4 model, no resize had to be done since the BBox mechanism it uses allows any image size as input but we still resized to 416 by 416 as recommended in (Bochkovskiy et al, 2020). The preprocessing for this model consisted in programmatically generating a BBoxes the same size as the image for (Arun & Geetharamani, 2019) since the controlled environment of the pictures in this dataset made it so only one leaf of each category was in the frame every

time. the (Singh et al, 2019) dataset was published with BBoxes so it does not need any more preprocessing.

Training and test sets

For the partitioning of the train and validation sets, we implemented a custom K-fold cross-validation partitioning function for the YOLO architecture, and for the MobileNet we used the Kfold TensorFlow-compatible sklearn library. The K-fold technique consists in separating a test set(10\% of the data in our case) and splitting the remaining train/validation set in K partitions (there is no literature for a specific k value but the consensus is that K should be between 5 to 10 folds, we chose 10 following the rule of thumb). Then a model is trained in k-1 folds and validated in the remaining fold. Then the model is discarded and the results saved. A different model is trained for each fold and discarded as they serve their purpose. This technique ensures a less biased and less optimistic model thanks to the variation it introduces in testing the model on unseen data. We can separate just 10\% of data for testing since the smallest of categories have at least 1000 images, we can count with a reasonable 100 images for testing in the worst-case scenario and 530 images in the best scenario without neglecting training and validation sets.

Model configuration

The YOLOv4 model does not support much hyperparameter tuning since most parameters like epochs, and optimizers are handled by its complex architecture and programmatically defined. The only tunable parameters in this model are batch size, learning rate, and subdivisions(a parameter for the CUDA architecture and the parallelization of training). We started training with a 0.001 learning rate that proved effective so it was not tested further. The rest of the tuneable hyperparameters can accelerate learning speed but this would also

increase the demand for more processing resources. We tested with the recommended batch size of 8 and also with 16, 32, and 64.

Next, for the YOLOv4 we wrote a custom configuration file that adjusted the output of some layers to our number of classes. Appending this configuration file to the training and testing commands lets the model run to our custom number of classes. This was done as indicated in (Bochkovskiy et al, 2020)'s accompanying official release of it's code in GitHub and its documentation.

For MobileNetV3's simpler architecture we tested a series of epochs from 500 to 1000 in increments of 100 and a learning rate from 0.1 to $0.1x10^{-5}$ with steps by a factor of 10^{-1}. We then used the GridSearchCV function in the sklearn library (compatible with the TensorFlow official implementation of the MobileNetV3 model) to obtain the best hyperparameters. With this function, we were able to extract and save the best parameters: a learning rate of 0.001 and 1000 epochs.

Similar to the YOLO custom configuration file, for the MobileNet we cut the last 5 layers and appended a custom flatten file that densely connects to a custom output layer with our custom number of output classes. It is worth mentioning that we can cut more or append more layers on this model as further fine-tuning of the model but we start with 5 as the rule of thumb for this model.

Assessment Metrics

The mAP is based on a group of sub metrics: confusion matrix, intersection over union (IoU), recall, and precision. The base metric is the Intersection over Union (IoU). The IoU is calculated using the ground truth BBox and the predicted BBox. To get the IoU we divide the

area of overlap between both boxes over the area of the union of both boxes. The higher and closer to 1 this metric is, the better the prediction was.

This is the first relevant metric since we use it to set an IoU threshold, that is a minimum IoU value of a prediction to be considered true or false. In the YOLO architecture, if the IoU of a prediction is greater than 0.5 it is considered good enough and counts as a true prediction. In MobileNet the IoU threshold is another hyperparameter that can be tunned but for now, we only tested the model with the same 0.5 thresholds based on previous research results (Singh et al, 2019), also the Mean Average Precision with a 0.5 IoU threshold can be considered a standard metric.

The confusion matrix counts and graphs the predicted categories for the test data versus the actual categories. In our case, with multiple categories, the cells where the predicted value and the actual value for each category intersect can be called the True section (the diagonal cells from top left corner to bottom right corner) and this value will be the times the model predicted the correct category. Any value outside this true section will be a false prediction.

The Precision and Recall metrics must be calculated for each category and then averaged to obtain the metric for the whole model. For the precision of each category, we must divide the times the model correctly predicted over the total amount of times the model predicted that category. Recall, similarly, is calculated by the times the model correctly predicted each category over the total amount of elements of each category.

RESULTS AND DISCUSSION

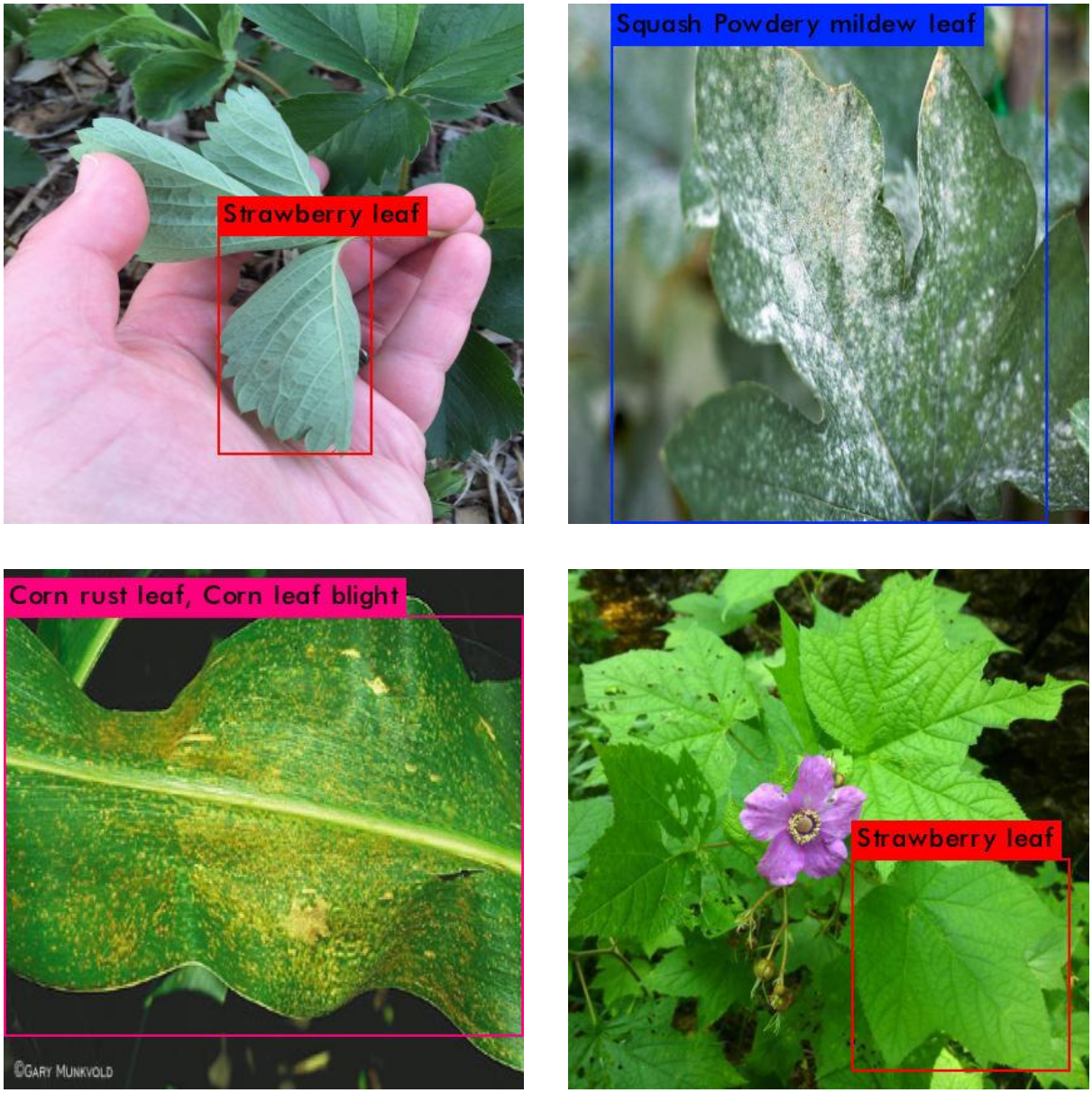

Figure 4. Examples of the predictions made by the YOLO architecture

To evaluate the results we can use not only the evaluation metrics but for a more visual evaluation we plotted some of test images with the predictions made by the YOLO architecture. The visual advantage we have with YOLO is the prediction with BBox illustrations. In figure \ref{predictionsPlot} we can see the prediction of some of the test

images. To compare we can further explain the results of the YOLO predictions compared with the MobileNet predictions.

	Predicion on:			
Model	**Image 1**	**Image 2**	**Image 3**	**Image 4**
YOLOv4	• Corn rust: 64% • Corn blight: 40%	• Strawberry: 32%	• Powdery mildew: 61%	• Strawberry: 64%
MobileNetV3	• Corn rust: 74% • Corn blight: 48% • Gray spot: 14%	• Raspberry: 48% • Strawberry: 42%	• Powdery mildew: 51% • Squash leaf: 27%	• Strawberry: 56%
Ground truth	Corn rust	Raspberry	Powdery mildew	Strawberry

Table 1. Summary comparing some test predictions of YOLO vs MobileNet

Both made the predictions correctly in most of these tests but from these sampled results we begin to see a trend favoring MobileNet, this can be further explored in the next subsection.

Performance evaluation

Backbone architecture	$\boldsymbol{mAP_{0.5}}$
Tiny YOLOv4	20.4%
YOLOv4	27.6%
MobileNetV3 large	34.6%
MobileNetV3 small	25.7%

Table 2. Summary of results on test subset

We tested both architectures in their full form as well as the compact, more mobile-oriented version of each one. YOLOv4 mainly served as a benchmark since it has weights that are constantly updated and a more complex design. YOLO architecture is more focused on taking advantage of transfer learning and we hypothesized these advantages will give make it

better. MobileNet, on the other hand, is not as focused on transfer learning but has a repository of weights for the model trained in different datasets. Because of the reasons stated in the models section, we cannot take advantage of transfer learning with this model so we built, trained and fine-tuned our MobileNet models from the ground up using as a foundation its implementation in TensorFlow.

In table \ref{Results_table} we can compare the lighter versions of each architecture with the full version. As expected, smaller architectures are not as precise as the full architectures. MobileNet, comparing both of its versions, reflects the smaller set back dropping less percentage in $mAP_{0.5}$. YOLO presents a more significant loss of its precision. We hypothesize that this happens because tiny YOLO can not take full advantage of the CSPDarknet-53 backbone architecture designed to extract features from the images. There are core differences in the backbone between YOLOv4 and Tiny YOLOv4 as noted in (Szegedy, 2015). These core differences, that make the model more adequate for mobiles do sacrifice too much in its functionality as shown in table \ref{Results_table}.

The starting database (Arun & Geetharamani, 2019) we used for this research is known as PlantVillage. It is the largest public dataset for plant disease classification. The latest version

ACKNOWLEDGEMENT

The authors thank the Applied Signal Processing and Machine Learning Research Group of USFQ for providing the computing infrastructure (NVidia DGX workstation) to implement and execute the developed source code. Publication of this article was funded by the Academic Articles Publication Fund of Universidad San Francisco de Quito USFQ.

REFERENCIAS BIBLIOGRÁFICAS

PRB. (2021). 2021 WORLD POPULATION DATASHEET.

FAO. (2018). The future of food and agriculture – Alternative pathways to 2050.

Savary, S., Willocquet, L., Pethybridg, S., Esker, P., McRoberts, N., & Nelson, A. (2019). The global burden of pathogens and pests on major food crops. Nature Ecology & Evolution, 3,3, 430 - 439.

David P. Hughes, & Marcel, S. (2015). 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.

Schuler, J., Romaní, S., Abdel-nasser, M., Rashwan, H., & Puig, D. (2021). Reliable Deep Learning Plant Leaf Disease Classification Based on Light-Chroma Separated Branches. Journal is required!, 375-381.

Mohanty, S., Hughes, D., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. Frontiers in Plant Science, 7.

Saleem, M., Potgieter, J., & Arif, K. (2019). Plant Disease Detection and Classification by Deep Learning. Plants, 8(11).

Pew Research Center. (2021). Demographics of mobile device ownership and adoption in the United States.

J, A., & GOPAL GEETHARAMANI. (2019). Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network.

Fang, Y., & Ramasamy, R. (2015). Current and Prospective Methods for Plant Disease Detection. Biosensors, 5(3), 537–561.

Stuart J. Russell, & Peter Norvig (2021). Artificial intelligence: a modern approach. PEARSON.

Ian Goodfellow, Yoshua Bengio, & Aaron Courville (2016). Deep Learning. MIT Press.

Turing, A. (1950). Computing Machinery and Intelligence. Mind, 59(236), 433–460.

Chollet, F. (2017). Deep Learning with Python. Manning.

Gao Huang, Zhuang Liu, Laurens van der Maaten, & Kilian Q. Weinberger. (2016). Densely Connected Convolutional Networks.

Bochkovskiy, A., Wang, C.Y., & Liao, H.Y. (2020). YOLOv4: Optimal speed and accuracy of object detection. CoRR.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. CoRR.

Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q., & Adam, H. (2019). Searching for MobileNetV3. CoRR.

Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat, & Nipun Batra (2019). PlantDoc: A Dataset for Visual Plant Disease Detection. CoRR, abs/1911.10317.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, & Zbigniew Wojna (2015). Rethinking the Inception Architecture for Computer Vision. CoRR, abs/1512.00567.