Real-Time Image Classification of Fruits and Vegetables with MobileNet

Majd Jamal
KTH
majdj@kth.se

Abstract

Deep networks are growing deeper and require more energy resources. Previous research has shown that the deep learning field will stop progressing if contributors do not produce lighter and efficient network architectures. This study utilizes MobileNet, the state-of-the-art network for mobile devices, to classify images of fruits and vegetables, and compares its performances to networks with more extensive architectures. Networks were trained from randomly initialized weights and Transfer Learning using ImageNet weights. MobileNet trained with Transfer Learning produced a top-1 accuracy of 96.8% and performed like the more extensive network architectures. This study uses the top-1 MobileNet predictor to conclude a real-time image classification app.

1. Introduction

Deep Learning Networks are becoming deeper, powerful, and applicable for numerous tasks, such as image classification and voice recognition. Networks tend to perform better as they become bigger. (Soltanolkotabi et al, 2018; Zhou, Z. H., 2021) Increasing network size is not sustainable because larger networks make more computations and require more electricity. Research has shown that the Deep Learning field will face difficulties in progressing if networks keep growing and contributors do not produce efficient and lighter network architectures. (Thompson et al., 2020). Howards et al. (2017) constructed a light network architecture that competes with the larger state-of-the-art networks. They call it the MobileNet, and it performs well on classification and detection tasks. MobileNet has such light architecture that devices with inferior computing units can use it for inference, such as smartphones and tablets. The scope of the study is to re-implement the original MobileNet, use it for image classification of fruit and vegetables, and compare the result to those produced by bigger networks, such as EfficientNet and ResNet101. The aim is to utilize a light network architecture in MobileNet to solve real-life problems. The research question is whether MobileNet will perform similarly to more extensive networks when classifying fruit images.

2. Theoretical Background

Hinton et al. (2012) showed that convolutional neural networks achieve great accuracy on image classification, which has led to an increased popularity in creating deep networks. This research will experiment with three network architectures, namely MobileNet, EfficientNet, and ResNet. These networks are chosen because of their unique characteristics. MobileNet is the state-of-the-art network for mobile devices. EfficientNet performs exceptionally on classification tasks and currently has the highest accuracy on ImageNet. ResNet has unique properties which enable it to have many layers without becoming difficult to train.

2.1. MobileNet

Howard et al. (2017) introduced a light network architecture that makes efficient computations, and they call it the MobileNet. It has been re-iterated three times. The original network is based on depth-wise and point-wise convolutions, making independent convolutions on input channels and then sums up the result. For example, assume that we have an RGB image of an apple; it has three channels - red, green, and blue. Instead of making computations on all colors simultaneously, depth-wise separable convolutions make independent computations on one color at a time and then sum up the result with point-wise convolutions. See Appendix Figure 1. Depth-Wise Separable convolutions make nine times fewer computations than conventional convolutions, thus making it possible to construct light and efficient networks. MobileNet achieves excellent results on ImageNet and is appropriate for multiple tasks, such as classification and detection.
The second iteration of MobileNet focuses on achieving a memory-efficient structure using residual and manifold learning. MobileNetV2 is built on inverted residual blocks. These blocks input a low-dimensional data representation, decode the data to a higher dimension when it is time to make convolutions, and encode the result back to a low dimension. The low-dimensional representations are called

bottleneck layers, and they are added together for residual learning. See Appendix Figure 2. The intuition is that values in the channels usually lie in a manifold and can be encoded to lower dimensions without losing information. (Howard et al., 2018)
The latest MobileNet iteration includes depth-wise convolutions and inverted residual blocks, but it consists of different network architecture. There are two versions of MobileNetV3, one small and one big. These network architectures are constructed using Neural Architecture Search, and the start and end layers are made more efficient than previous models. (Howards et al., 2019)

2.2. EfficientNet

Convolutional Neural Networks consist of three variables: depth - the number of layers, width - the number of channels, and resolution - the dimension of input data. Researchers can construct multiple network architectures by adjusting these parameters. Networks are usually initialized small and then scaled up, and this process is referred to as model scaling. The method of model scaling is tedious, and researchers often focus on one of the parameters, e.g., by stacking multiple layers on each other. Tan & Le (2019) proposed a new scaling method that makes networks more accurate as they grow. The intuition is to use a grid search to quantify constant coefficients for each of the three variables, depth by α, width by β, and resolution by γ, and then uniformly scale these variables with a compound coefficient. Tan & Le concluded a set of eight network architectures called EfficientNet, and they have state-of-the-art performance on ImageNet.

2.3. ResNet

As the neural networks become deeper, they tend to produce better accuracies. One could argue, *why do we not stack as many layers as possible?* And this is a common misapprehension. Training deep networks has its limitations. Studies have shown that as networks become extremely big, they start performing badly, a phenomenon referred to as the degradation problem. He et al. (2015) addressed the degradation problem and proposed a set of networks built on Residual Learning, named ResNets. The intuition is to pass forward information from previous layers to deeper ones. Assume that a layer computes $\mathcal{F}(x)$, where x is the input data. With residual learning, the computation function becomes $\mathcal{H}(x) = \mathcal{F}(x) + x$, where x is the output from previous layer. ResNets allows for stacking multiple layers without facing the degradation problem. As a result, He et al. trained a thousand-layer network without observing any optimization difficulties.

2.4. Statistic Metrics

Statistical metrics are often used to evaluate network performances. These metrics include F1Score, Recall, and Precision. Recall indicates the model recognition ability, i.e., the fraction of true positives that were retrieved. Precision is a fraction that shows how many true positives are relevant. F1Score is the weighted average of recall and precision.

2.5. Related Work

Xiang et al. (2019) collected fruit images from search engines and used MobileNetV1 for training and classification. They used Adaptive Momentum Estimation ADAM as an optimizer and achieved top-1 accuracy of 77%. Duong et al. (2020) utilized EfficientNet on an open image dataset named Fruit360, containing fruit imaged in a homogenous background. Their model was trained using Stochastic Gradient Descent, and it achieved a top-1 accuracy of 99% with EfficientNetB5. There are also numerous contributions with other networks, such as InceptionV3, VGG-16, and MobileNetV2. (Pande et al., 2019; Hossain et al., 2019; Rojas-Aranda, 2020)

3. Method

Experiments were conducted in the Python programming environment. Python was the right choice for this project because it provides the necessary packages for mathematical computations. This project used TensorFlow as the main Deep Learning package. Utility packages were Matplotlib, NumPy, Ski-kit Learn, tqsm, and tensorflow_addons. The hardware was 25GB RAM and NVIDIA V100 16 GB GPU, provided by Google Colab Pro with limited access.

3.1. Data

The dataset was created from scratch. The intention is to learn how to gather data to solve a specific task without being dependent on open datasets. High-Quality Images with both homogenous and noisy backgrounds were downloaded from search engines and image databases, such as Google Images and iStockPhoto. A web scraper named Image Downloader was used to automatize the downloading process. The final dataset consists of 14000 images and 15 categories: red apple, green apple, banana, carrots, chili, corn, kiwi, lemon, orange, peach, pear, raspberry, strawberry, tomato, and watermelon.
Images were converted from files to NumPy arrays using ImageFolder [1], and normalized to values ranging from 0 to 1 using cv2.Normalize. Then, image augmentation was applied to the training set. Adding pixel distortions to images

[1] tfds.folder_dataset.ImageFolder — TensorFlow Datasets. URL: $https://www.tensorflow.org/datasets/api\_docs/python/tfds/folder\_dataset/ImageFolder$

during training is an expensive computation process. Since Google Colab gives limited access to their computing units, the decision was to add pixel distortions before training and store the data in .npy files.

3.2. Training

This study implemented MobileNetV1 from scratch and configured its architecture to achieve greater performance. MobileNetV2 and V3 are optimized for object detection and segmentation. However, all versions use depth-wise separable convolutions and perform similarly on classification. EfficientNet and ResNet were implemented using Keras pre-defined models. [2]
All models were trained for a maximum of twelve hours, and this is because of limited access to hardware. The gradient descent algorithm used in this project is a modified version of stochastic gradient descent proposed by (Loshchilov & Hutter, 2017), which is found in the Tensorflow Addons package, a module named SGDW. [3] SGD makes the loss function fluctuate and potentially jump to a better local minimum than other gradient descent algorithms. (Ruder, 2016) The modified algorithm includes regularization techniques such as weight decay and momentum. Moreover, an image generator was used to make unique image augmentations for each epoch. Augmentations include rotations, horizontal and vertical flips, and shifts along the horizontal axis. The generator algorithm was accessible through Keras preprocessing package, a module named ImageDataGenerator. [4]

3.3. MobileNet Congfiguration

Fruits and vegetables might look similar cross-categories, such as red apples and peaches. The intuition is to make the network wider, thus making it better at distinguishing between similar-looking categories. Wider networks tend to learn more fine-grained features. (Zagoruyko Komodakis, 2016) The network was also made shallower to decrease the parameter count. Depth of a network controls generalization. (Tan Le, 2019) Fruits and vegetables do not vary substantially in their appearances, e.g., bananas have a standard curved shape, and apples are usually round and red. Therefore, making the network slightly shallower should not penalize the performance.
For the configurations, the middle layers of MobileNetV1 were removed to decrease depth. See Appendix Figure 3 for detailed information. The hyperparameter alpha, which controls the network width, was increased from 1.0 to 2.6.

[2]Keras apps. Tensorflow. URL: https://www.tensorflow.org/api$_docs/python/tf/keras/apps$

[3]TensorFlow Addons. tfa.optimizers.SGDW URL: $https://www.tensorflow.org/addons/api\_docs/python/tfa/optimizers/SGDW$

[4]Tensorflow. tf.keras.preprocessing.image.ImageDataGenerator. URL: $https://www.tensorflow.org/api\_docs/python/tf/keras/preprocessing/image/ImageD$

3.4. Evaluation

The model with the highest validation accuracy is selected, and its classification accuracy on the test set is measured and documented. The evaluation process includes analysis of statistical measures such as Precision, Recall, and F1score. These measures are important because they can show how the model is performing on specific labels. Moreover, an error matrix was used to measure how the model distinguishes between labels.

3.5. Real-Time Image Classification

This project develops a real-time image classification app. Users can show fruit to their computer's webcam, and the app will classify the fruit in real-time. The app is built on the most accurate MobileNet model achieved from the training sessions. Photos are captured live from the user's webcam and preprocessed through algorithms available in the OpenCV-Python package, a module named VideoCapture.[5] After preprocessing, the photo is sent to the classifier, which computes predictions and creates a graphical user interface with Matplotlib, demonstrating (1) the captured photo and (2) the predicted label. The app was evaluated using 197 photos of fruits captured by Logitech C505 720p webcam.

4. Experiments

This section includes documentation of multiple experimentation results and observations. A summary of network classification accuracies is found in Table 1.

Model	Random Init. Accuracy	TL Accuracy	Params
MobileNetV1	92.7%	96.8%	3.2M
MobileNet Config.	93.9%	–	3.5M
EfficientNetB0	81.0%	94.1%	4.1M
EfficientNetB5	86.9%	97.4%	28.5M
ResNet101	89%	97.2%	42.7M

Table 1. Top-1 accuracy of various Deep Networks. All networks on this table were trained for a maximum of twelve hours because of hardware limitations. Remark: MobileNetV1 that was configured shallower and wider was the most reliable predictor amongst networks trained with randomly initialized weights. Note: Transfer Learning accuracy for MobileNet Config is missing because pre-trained models were not available for the particular architecture.

[5]OpenCV: Getting Started with Videos. URL: https://docs.opencv.org/4.5.2/dd/d43/tutorial_py_video$_display.html$

4.1. Networks

MobileNetV1 trained with randomly initialized weights had a top-1 accuracy of 92.7%. The loss and accuracy functions for the model are found in Appendix Figures 4 and 5. The confusion matrix is found in Appendix Figure 6.

Statistics show that MobileNetV1 trained with randomly initialized weights had difficulties distinguishing between red apples and peaches (27 errors), and green apples and pears (18 errors). F1Score was largest for Watermelon (0.97), Kiwi (0.96), and Banana (0.96), and lowest for Red Apple (0.88), Peach (0.88), Pear (0.88). The full table is found in Appendix Table 2.

MobileNetV1 trained with Transfer Learning had a top-1 accuracy of 96.8%. The loss and accuracy functions are found in Appendix Figure 7 and 8. All statistical scores exceed 0.9, and the full tables are found in Appendix Table 3. The confusion matrix is found in table 9.

MobileNetV1 that was configured shallower and wider resulted in the most accurate predictor amongst networks trained from randomly initialized weights. The top-1 accuracy was 93.9%, which is a +1.2% increase in comparison to the original architecture. The confusion matrix shows that error due to distinguishing between red apples and peaches was decreased from 27 to 12 errors. See confusion matrix in figure 10.

4.2. Real-Time Image Classification

Accuracy on webcam photos was 95.5%, and the app makes 4 inferences per second without a graphical processing unit. The graphical user interface is found in Appendix Figure 11.

5. Discussion

MobileNetV1 was the most accurate predictor among networks trained from randomly initialized weights. This observation shows that MobileNet requires less rigorous training, making it ideal for projects with limited access to computing units. The configuration of making MobileNetV1 wider and shallower led to performance gains, which indicates that the wider network was better at learning fine-grained features, and as such, the error between similarly-looking categories was reduced.
Training with Transfer Learning produced greater accuracies than with randomly initialized weights in all experiments. This observation shows that models trained from random weights can achieve better local minimum with hyperparameter adjustments, more image augmentation, longer training sessions, and possibly a different gradient descent algorithm.
With Transfer Learning, MobileNet performed similarly to larger networks when classifying images of fruits and vegetables.
Statistical measures show that the models had difficulties distinguishing between red apples and peaches, and green apples and pears. This source of error is understandable because these fruits look alike in certain angles and lighting conditions.
MobileNet performed better than EfficientNetB0, and this is a surprising result because EfficientNets are known for their strong classification power. The architecture of EfficientNetB0 could explain this observation. Tan & Le constructed EfficientNetB0 using Network Architecture Search, and it resulted in a baseline model that suits a particular dataset. That baseline might not suit the dataset used in this research. I.e., if Network Architecture Search was applied using fruit and vegetable images, then the baseline might become different.
It is worth noting that ResNet101 was the most straightforward network to train among the bigger networks. This observation shows that Residual Learning is indeed powerful and makes training large networks more manageable.
In general, all networks produced accuracies around 90 percent, and this is a massive achievement. The most accurate MobileNet predictor had a top-1 accuracy of 96.8%, and it was used to make a real-time image classification app.
The real-time image classification app worked as intended, and the predictor generalizes strongly on webcam photography. Further research can modify the graphical user interface and use the app in grocery stores. For example, cashiers could use the app to register fruits automatically without needing to type product codes, or customers can automatically identify fruits and print price tags at the scale.

6. Conclusion

Deep networks must become sustainable, light, and efficient for the deep learning field to progress. MobileNet was used to showcase a light architecture and compare its performance to bigger architectures, such as EfficientNet and ResNet101. The experimentations show that MobileNet trained with Transfer Learning performed similarly to the larger networks when classifying images of fruits and vegetables. Moreover, MobileNet required less rigorous training than the bigger networks, making it ideal for projects with limited access to hardware. This study uses the top-1 MobileNet predictor to conclude a production-grade real-time image classification app.

References

Duong, L. T., Nguyen, P. T., Di Sipio, C., & Di Ruscio, D. (2020). Automated fruit recognition using EfficientNet and MixNet. Computers and Electronics in Agriculture, 171, 105326.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Hossain, M. S., Al-Hammadi, M., Muhammad, G. (2018). Automatic fruit classification using deep learning for industrial apps. IEEE Transactions on Industrial Informatics, 15(2), 1027-1034.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision apps. arXiv preprint arXiv:1704.04861.

Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1314-1324).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

Pande, A., Munot, M., Sreeemathy, R., Bakare, R. V. (2019, March). An Efficient Approach to Fruit Classification and Grading using Deep Convolutional Neural Network. In 2019 IEEE 5th International Conference for Convergence in Technology (I2CT) (pp. 1-7). IEEE.

Rojas-Aranda, J. L., Nunez-Varela, J. I., Cuevas-Tello, J. C., & Rangel-Ramirez, G. (2020, June). Fruit Classification for Retail Stores Using Deep Learning. In Mexican Conference on Pattern Recognition (pp. 3-13). Springer, Cham.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition(pp.4510-4520).

Soltanolkotabi, M., Javanmard, A., & Lee, J. D. (2018). Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. IEEE Transactions on Information Theory, 65(2), 742-769.

Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.

Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The computational limits of deep learning. arXiv preprint arXiv:2007.05558.

Xiang, Q., Wang, X., Li, R., Zhang, G., Lai, J., & Hu, Q. (2019, October). Fruit image classification based on Mobilenetv2 with transfer learning technique. In Proceedings of the 3rd International Conference on Computer Science and app Engineering (pp. 1-7).

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv preprint arXiv:1605.07146.

Zhou, Z. H. (2021). Why over-parameterization of deep neural networks does not overfit?. Science China Information Sciences, 64(1), 1-3.

Appendix

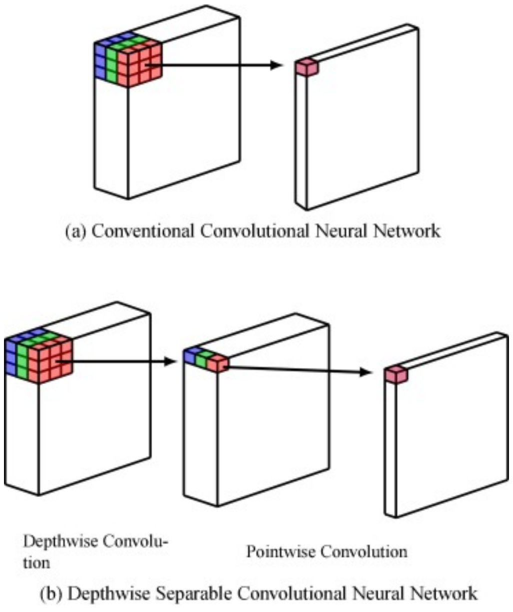
(a) Conventional Convolutional Neural Network
Depthwise Convolution
Pointwise Convolution
(b) Depthwise Separable Convolutional Neural Network

Figure 1. Illustration of Depth-Wise Separable Convolutional Neural Network. (a) Conventional convolutions neural networks make convolutions on all channels at the same time. (b) Depth-Wise Separable Convolutional Neural Network makes convolutions on one channel at a time, known as depthwise convolutions, and then sums up the result with a point-wise convolution. DWS convolutions make nine times fewer computations than Conventional Convolutions. (Howards et al., 2017)

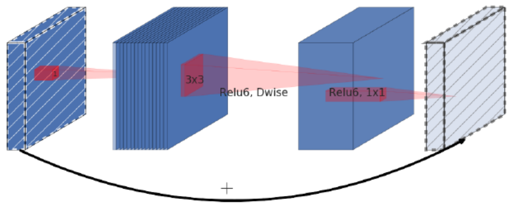

Figure 2. This is a demonstration of the inverted residual block that is used in MobileNetV2. The first layer takes as input a low-dimensional data representation. When it is time to make convolutions, the data get transformed to a higher dimension. The result is then transformed back to low dimension. The first and last layers are named bottleneck layers, and they are added together, as seen in the connecting arrow.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3. MobileNetV1 architecture and configuration details. Yellow-colored layers were modified, the repeated layers were decreased from 5x to 1x. Red-colored layers were deleted. The hyperparameter that controls network width, alpha, was increased from 1.0 to 2.6. Comment: This configuration increased performance by 1.2% and resulted in a top-1 accuracy of 93.9%.

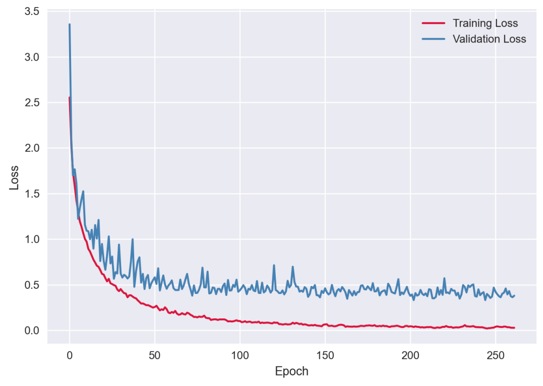

Figure 4. Training and validation loss for MobileNet trained from randomly initialized weights. Settings: learning_rate = 1e-3, weight_decay = 5e-5, momentum = 99e-2, dropout = 1e-5. Comment: We can see that the loss function follows a downward trend, which indicate that the gradient descent was properly implemented.

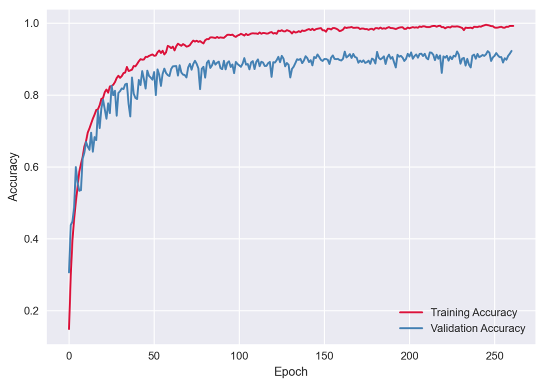

Figure 5. Training and validation accuracy for MobileNet trained from randomly initialized weights. Settings: learning_rate = 1e-3, weight_decay = 1e-5, momentum = 99e-2, dropout = 1e-5. Comment: The accuracy function moved in an upward trend, indicating successfull learning of the feature space.

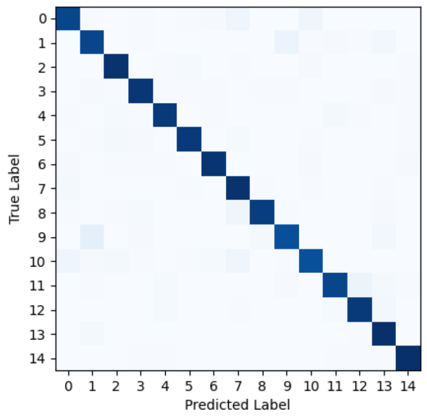

Figure 6. Confusion matrix from evaluating MobileNet trained with randomly initialized weights. Comment: We can see that the model has difficulties distinguishing between labels 1 and 9, red apples and peaches. This source of error is understandable since both fruits look like each other in some viewing angles and lighting conditions.

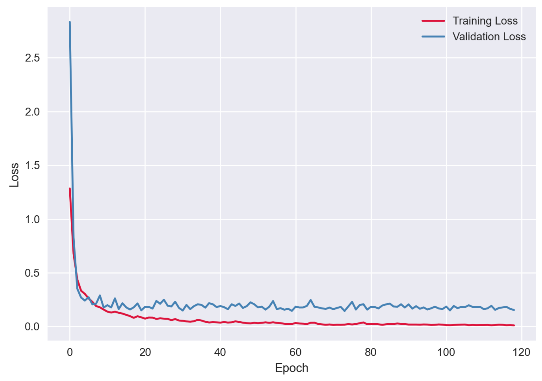

Figure 7. Training and validation loss for MobileNet trained with Transfer Learning. Settings: learning_rate = 1e-3, weight_decay = 1e-5, momentum = 99e-2, dropout = 1e-5. Comment: We can see that the loss function follows a downward trend, which indicate that the gradient descent was properly implemented.

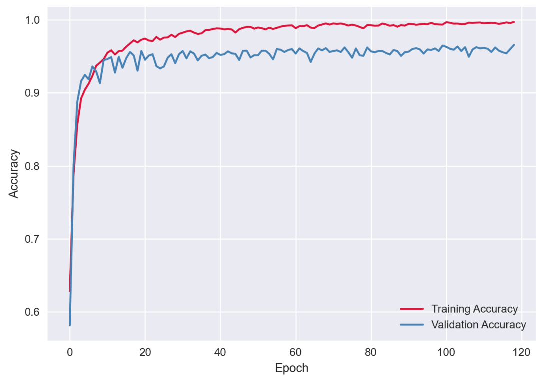

Figure 8. Training and validation accuracy for MobileNet trained with Transfer Learning. Settings: learning_rate = 1e-3, weight_decay = 1e-5, momentum = 99e-2, dropout = 1e-5. Comment: The accuracy function moved in an upward trend and reached greater heights than the model trained with random weights.

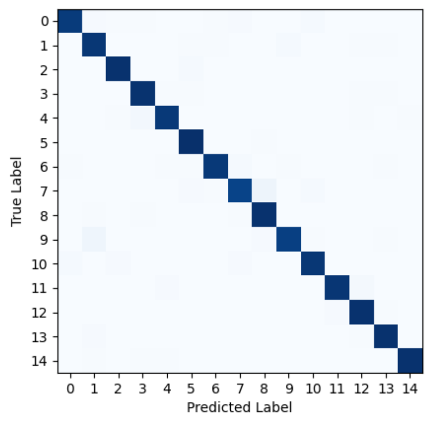

Figure 9. Confusion Matrix from evaluating MobileNet trained with Transfer Learning. Comment: We can see that there are few miss-classifications compared to the model trained with random weights. Moreover, this model did not have significant difficulties distinguishing between fruits, which is an outstanding accomplishment.

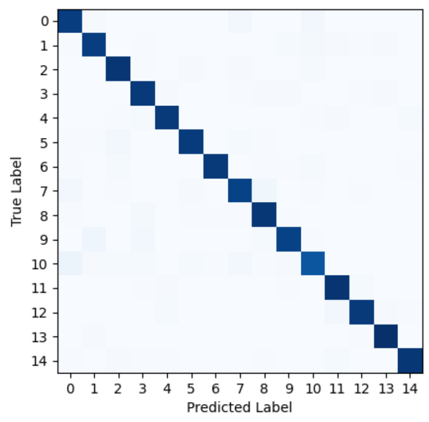

Figure 10. Confusion Matrix from evaluating the configured MobileNet architecture. Comment: If we compare this confusion matrix with Figure 6, we can see that the classification errors between similarly-looking categories was reduced. This indicates that making the network wider led to better learning of fine-grained features.

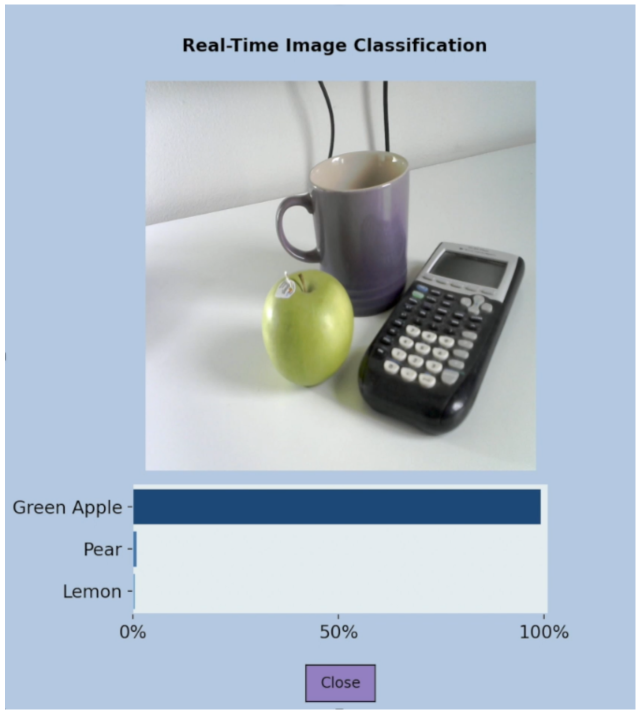

Figure 11. Graphical User Interface. The first window shows captured photos live from the user's webcam. The second window includes prediction bars, indicating which class the fruit belongs to. The demonstration shows 99% Green Apple. On the bottom, there is a purple button to close the app.

						Labels									
	G Apple	R Apple	Banana	Carrots	Chili	Corn	Kiwi	Lemon	Orange	Peach	Pear	Rspbry	Strwbry	Tomato	Watermelon
F1Score	0.901	0.880	0.946	0.940	0.945	0.952	0.960	0.912	0.944	0.890	0.884	0.920	0.932	0.924	0.970
Recall	0.890	0.895	0.960	0.945	0.940	0.940	0.955	0.960	0.920	0.860	0.855	0.895	0.935	0.975	0.975
Precision	0.913	0.865	0.932	0.936	0.950	0.964	0.965	0.869	0.968	0.920	0.914	0.947	0.930	0.878	0.965

Table 2. Statistical measure showing performance of MobileNet trained with randomly initialized weight. Comment: The network produces most accurate results on watermelon, kiwi and banana, and least accurate on red apples, peach and pear.

						Labels									
	G Apple	R Apple	Banana	Carrots	Chili	Corn	Kiwi	Lemon	Orange	Peach	Pear	Rspbry	Strwbry	Tomato	Watermelon
F1Score	0.964	0.946	0.982	0.970	0.967	0.978	0.972	0.939	0.961	0.956	0.965	0.980	0.973	0.980	0.990
Recall	0.9500	0.965	0.985	0.980	0.960	0.995	0.960	0.920	0.985	0.935	0.965	0.970	0.985	0.985	0.985
Precision	0.979	0.928	0.980	0.961	0.974	0.961	0.985	0.958	0.938	0.979	0.965	0.990	0.961	0.975	0.995

Table 3. Statistical measure showing performance of MobileNet trained with Transfer Learning. Comment: Traning with Transfer Learning resulted in a predictor that performs well on all classes, and much better than the one trained from randomly initialized weights.