Teaching an Old Dog New Grapes: Options for Object Detection with Fruits

Wookyung Lee (wlee@uos.de), Piper Powell (ppowell@uos.de), Milad Rouygari (mrouygari@uos.de)

Abstract

Object detection tasks involve localizing and identifying objects in images versus simply classifying images as a whole, and are often tackled through the use of neural network models. Among the most famous neural network models for this task are those in the Single Shot Detector (SSD) family, which are made up of a backbone classifier network and additional convolutional layers to implement the localization functionality. Many pre-trained versions of networks in this family are available. Thus, two key questions arise when approaching the application of an SSD-style object detector to a new object detection task - whether to use a pre-trained network or to train the network from scratch on the data for the present task, and what kind of classifier to use as the backbone. We explore options for both in an example application of training an SSD object detector to detect grapes in input images.

1 Introduction

Object detection is essentially standard image classification on steroids. Instead of simply telling you *what* object is in an image, these networks can also tell you *where* it is, and can do this for multiple objects and even multiple kinds of objects in a single image. These networks are widely used today in such varied tasks as detecting dead fish in water (Yu et al., 2020), monitoring traffic density (Biswas et al., 2019), and tracking shoes and feet to assist in fall risk assessment (Fernandez & Wada, 2020).

Object detectors came in a variety of forms, with one of the most famous being the family of Single-Shot Object Detectors (SSDs), which unlike other detectors can identify multiple objects in an image with one view. In general, such a detector consists of two parts - a "backbone," which is simply a classifier (called a "feature extractor") network, and a "head," which is simply additional convolutional layers added on top of that backbone. The backbone is responsible for identifying objects in an image while the head localizes them, usually by drawing so-called "bounding boxes" around them (Liu et al., 2016). This architecture is shown in Figure 1 below, which is reproduced from the original 2016 paper by Wei Liu et. al.

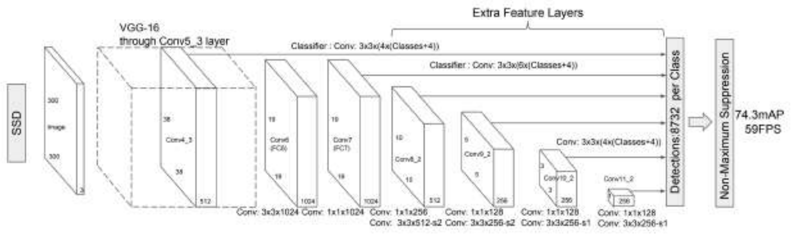


Figure 1: Architecture of an SSD object detector. Taken from Liu et al., 2016.

A trained detector begins by drawing many bounding boxes on many scales of an image, achieved by using a variety of different filter sizes. It then removes those bounding boxes where an object is not present, i.e. where the backbone classifier's confidence is low. During training, some of the generated boxes are matched to the provided correct boxes for the training images, with the rest of the boxes being taken as negative samples. Training data for these networks usually consist of a set of images and associated annotations which provide information on the correct locations of the objects in those images (in either bounding box or mask form) and the class of the objects in the boxes. Since such data therefore contains both classification and localization information, the entire detector (both head and backbone) can be trained as a whole. The classification information trains the feature detector backbone while the localization information trains the head (Liu et al., 2016).

Two key questions that must be addressed when approaching a new object detection task and constructing a new object detection model to tackle it are whether to train the new model from scratch versus building off of an existing model, and what kind of backbone classifier to use. In an example application of detecting grapes in images with an SSD-style object detector, we explore both of these questions by training models with and without prior training in image processing tasks (i.e. with or without transfer learning) and comparing the performance of two popular backend classifiers for SSD networks - MobileNet (Sandler et al., 2018) and EfficientNet (Tan & Le, 2019).

Object classification or object detection models can pose significant challenges when it comes to training them from scratch. This is because there are various complexities that need to be taken into account. Obtaining labeled data in large quantities is essential for training models, and this can be challenging since it is not always easy to obtain. Object detection datasets in particular can pose a challenge, as the information for each image must contain location and class information for numerous instances of objects in the image, versus a single general class label as is used with basic object classification. The architectures of models can also have a large number of parameters that require adjustment for optimal performance on a particular task. The training process can thus be computationally expensive and requires powerful hardware, as well as considerable time.

Transfer learning, a technique where a model developed on one task can be used for a similar but different task, presents one remedy to the aforementioned concerns. Essentially, it involves "transferring" the knowledge learned from one domain to another, usually in the sense of taking knowledge of broader basic principles from its prior training and then specializing for the current task with that background maintained. Figure 2, reproduced from Azunre et al. 2021, provides a handy graphic overview of the benefits of transfer learning and how it helps address the challenges discussed earlier.

In convolutional neural networks (CNNs), the early layers are usually responsible for performing general image processing functions, such as detecting edges in the image, while the later layers are responsible for more task-specific functions, like associating output with specific labels. Because of this structure, the main approach for transfer learning with such a network is to freeze the earlier layers and only allow the parameters in the later layers to be trained for the new task, thereby maintaining basic domain knowledge and simply "fine-tuning" functionality for the new task (Azunre et al., 2021, pp. 18-21). This process is shown graphically in Figure 3.

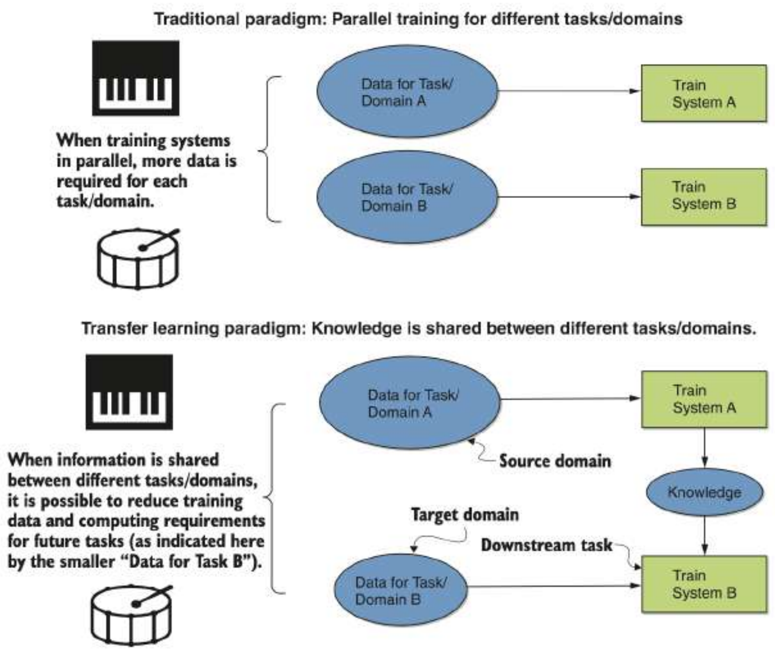

Traditional paradigm: Parallel training for different tasks/domains
When training systems in parallel, more data is required for each task/domain.
Data for Task/ Domain A
Train System A
Data for Task/ Domain B
Train System B
Transfer learning paradigm: Knowledge is shared between different tasks/domains.
When information is shared between different tasks/domains, it is possible to reduce training data and computing requirements for future tasks (as indicated here by the smaller "Data for Task B").
Data for Task/ Domain A
Source domain
Train System A
Knowledge
Target domain
Downstream task
Data for Task/ Domain B
Train System B

Figure 2: The basics of transfer learning. Taken from (Azunre et al., 2021, p. 6).

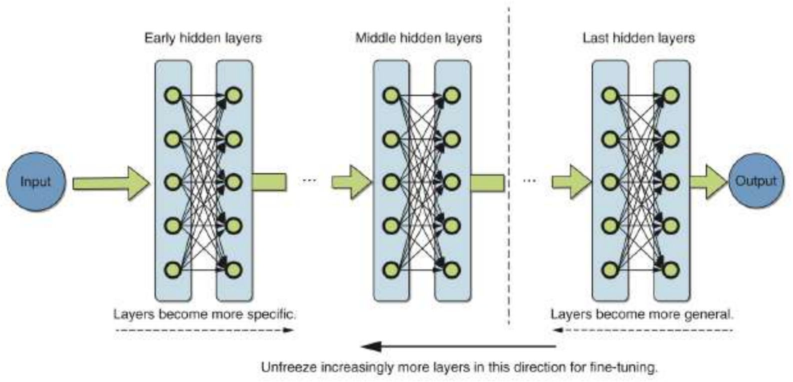

Early hidden layers
Middle hidden layers
Last hidden layers
Input
Output
Layers become more specific.
Layers become more general.
Unfreeze increasingly more layers in this direction for fine-tuning.

Figure 3: Transfer learning with CNN layers. Taken from (Azunre et al., 2021, p. 20).

While SSD can be used with any backbone, two of the most famous are the MobileNet and EfficientNet classifiers, the combinations we will discuss further in this paper and utilize in the associated application (Sandler et al., 2018; Tan & Le, 2019). In our comparison and work in this paper, we specifically work with MobileNetV2 (Sandler et al., 2018). A general comparison of these networks is provided in Table 1. For information on specific vocabularly related to EfficientNet that is presented in this table, see section 2.4. Further information on the MobileNetV2 architecture can be found in the background paper for this network (Sandler et al., 2018).

Broadly summarized, the advantage of using MobileNet as the backbone for an SSD object detector is its resource-efficient size, but EfficientNet may be able to provide better accuracy because of its more complex structure (**kaggle**). EfficientNet comes in multiple forms, named B0 through B7. As we progress from the B0 to B7 variants of the EfficientNet model, the scaling coefficient changes and the network becomes deeper, wider, and higher in resolution, which makes it larger and more powerful. However, this also implies that the computational demands of the network increase (Tan & Le, 2019). For this reason, we test EfficientNet-B0 in this project, as even this lower-power variant has been shown to outperform MobileNet v2, exceeding the latter network's accuracy on the ImageNet classification challenge as of 2019 (Sandler et al., 2018; Tan & Le, 2019).

Attribute	**MobileNetV2**	**EfficientNet**
Use	Primarily used for image classification tasks.	
Feature Extraction	Both networks use multi-scale feature extraction to capture features at multiple scales in an image, which is important for detecting objects of different sizes and viewing distances.	
Available Pretrained	Pretrained versions of both MobileNetV2 and EfficientNet are available.	
Convolution	Using depthwise separable convolution, both networks decompose the standard convolution operation into two separate operations: depthwise convolution and pointwise convolution, which reduces the number of parameters and computations in their convolutional layers.	
Model Size	Smaller: a compact model suitable for resource-constrained environments.	Bigger: a more powerful but more complicated model capable of achieving higher accuracy.
Model Architecture	Focuses primarily on depth-wise separable convolutions for efficiency, no scaling.	Uniform scaling of the dimensions of depth/width/resolution with a compound coefficient.

Table 1: Comparision between MobileNetV2 and EfficientNet (Sandler et al., 2018; Tan & Le, 2019).

The models tested and how they were tested is discussed in the following section.

2 Method

2.1 Models Tested

We ultimately tested 3 models in this project: SSD-MobileNet in a wiped state, SSD-MobileNet in a pre-trained state, and SSD-EfficientNet in a wiped state. Originally, our self-made EfficientNet (see section 2.4) was to be coupled to the SSD head and used as the third model for training, but we were unable to resolve errors which occurred in this process and chose to utilize a pre-made EfficientNet backbone instead. As a checkpoint file was not available for SSD-EfficientNet, and therefore it could not be loaded in a pre-trained state, it was instead worked with in a wiped state.

Three different losses were calculated for each network in each training configuration:

- Classification loss: This loss was used to measure how well the network is able to classify images into different categories.
- Localization loss: This loss was used to measure how well the network is able to predict the location of objects within an image.
- Regularization loss: This loss was used to prevent overfitting and encourage the network to learn simpler, more generalizable representations.

2.2 Custom Datasets

Two custom datasets were created for this project - one for grapes and one for bananas. The generation process for both is described below.

2.2.1 Grapes Dataset

The grapes dataset is a hand-crafted subgroup of the grapes dataset available in this repository. 100 images were manually selected by the team, with 50 displaying mostly light-colored green grapes and 50 displaying mostly dark-colored purple grapes, to give the networks trained on this dataset a good chance of detecting grapes of either coloring. The annotation data for the selected images was collected from the COCO-style annotation files provided in the dataset. The class data was also changed from the original 5-class system to a single class notation (simply '1' for grapes, as subclasses of grapes were not important to us). Later, the COCO-style bounding box data was translated into Pascal VOC format so it could be used by the code which drew the bounding boxes in the final Colab training notebooks.

2.2.2 Bananas Dataset

The bananas dataset is a subset of the the COCO 2017 dataset in which a selection of 100 banana images and their associated data was selectively downloaded using the "create_bananas_ds" Python script provided in our repository. This dataset was ultimately only used when training our custom EfficientNet classifier, when it was combined with the grapes data into a proof-of-function dataset for demonstrating our self-made EfficientNet model.

2.3 Training with the TensorFlow Object Detection API

We ultimately trained all of our models (except for our self-made EfficientNet model, the procedures for which are detailed in section 2.4), within the TensorFlow Object Detection API. This API handles model generation and training in a different format than the basic method of running a main file referencing model, training, and other scripts. Instead, Python scripts for a collection of backbone models, including various versions of MobileNet and EfficientNet, are provided with the API, along with meta architecture files (the heads) for each kind of object detector, including SSD. A model builder file is called which combines the meta architecture file for the selected object detector with the file for the selected feature selector backbone, thus creating the entire object detector. The master config file then contains a model name, which allows the chosen model to be looked up in the model builder file and the appropriate Python files for the head and backbone sections to be located. The master config file also contains all information relevant to training, including the usual parameters such as batch size, the number of training steps, and information regarding the optimizer and loss function to be used. Within this pipeline, models train off of TFRecord files, which our raw image and csv files are converted into ahead of model training in all of our linked Colab notebooks (see External Links).

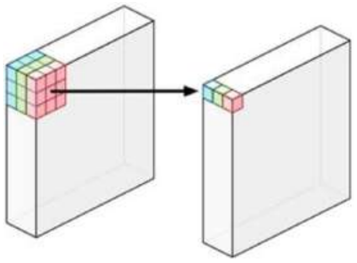
2.4 Building EfficientNet From Scratch

A convolutional network with increased depth has the ability to capture more intricate and diverse features, and can perform well on new tasks by generalizing its learned knowledge. Nevertheless, training deeper networks can be challenging due to the issue of vanishing gradients (Zagoruyko & Komodakis, 2016) Wider networks have the advantage of being able to capture more detailed and nuanced features, and are generally easier to train. However, very wide but shallow networks may struggle with capturing higher-level features (Zagoruyko & Komodakis, 2016). Convolutional networks also have the potential to capture more intricate patterns when provided with higher resolution input images, although the magnitude of accuracy gain tends to decrease for extremely high resolutions (Tan & Le, 2019)

EfficientNet, in contrast to traditional methods that randomly scale network depth, width, and resolution, applies a fixed set of scaling coefficients to uniformly adjust these dimensions (also known as uniform scaling for simplification), resulting in a fixed ratio (phi) increase in computational cost. Essentially, EfficientNet is a type of CNN architecture and scaling technique that uses a compound coefficient to uniformly scale depth, width, and resolution.

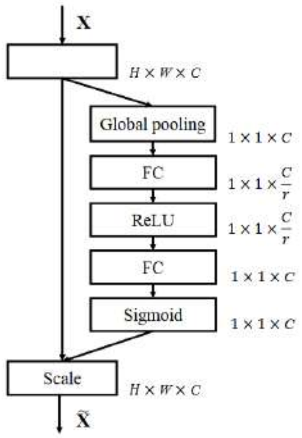
The so-called Inverted Residual Block (MBConv) is the major component of our network. Its component elements are detailed below, with the graphics taken from (Duong-Trung et al., 2019). Our code for implementing this architecture can be found in our GitHub repository (see External Links).

1. Depthwise Convolution



   Depthwise filters are independently applied to each individual input channel. The operation is specifically carried out on the depth dimension or channels of the input tensor (Duong-Trung et al., 2019).

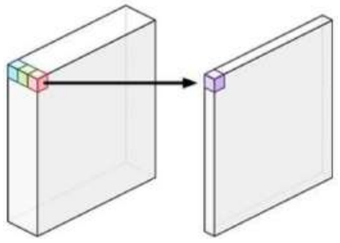
2. Squeeze Excitation



   The *squeeze* involves using global average pooling to incorporate global spatial information into channel descriptors. The output of the squeeze operation is fed into two fully-connected (FC) layers, with a ReLU activation function applied in between, and to effectively capture channel-wise dependencies, we utilize a gating mechanism with a sigmoid activation function. Finally a dimensionality-increasing layer that restores the original channel dimension is used.

   This enables the network to perform feature recalibration, enabling it to learn how to utilize global information to selectively emphasize informative features and suppress less useful ones (Hu et al., 2017).

3. Pointwise Convolution

   Also known as 1x1 convolution, this utilizes a kernel with a size of 1x1 that iterates over every point in the input image. The depth of this kernel is determined by the number of channels present in the input image (Duong-Trung et al., 2019).

4. Layer dropout during training using stochastic depth (SD)
   During training, SD randomly drops out entire residual blocks in a network with a certain probability. This technique can enhance network regularization by preventing overfitting and facilitating better gradient flow. By training the network to operate effectively even in the presence of missing blocks, SD can improve the network's robustness to noisy or incomplete input data, thereby enhancing its performance in challenging conditions (Huang et al., 2016).

We attempted to integrate our custom EfficientNet with SSD for object detection, but were unsuccessful. As a result, we ultimately decided to train our EfficientNet as a classification model with the specific objective of distinguishing between images of bananas and grapes. The SSD-MobileNet models and the official SSD-EfficientNet model from the TensorFlow Object Detection zoo were therefore trained and tested on our custom grapes dataset in the linked Colab notebooks below, while our custom EfficientNet was trained on a combined dataset of bananas and grapes to which appropriate labels were added to convert it to a classification dataset. The final classification dataset consisted of 200 images of bananas and grapes, with the training set consisting of 160 images and the validation set containing 40. This dataset can be found in our repository.

2.5 External Links

Our project ultimately consists of 3 Google Colab notebooks and a separate GitHub repository containing our custom datasets, our self-made EfficientNet network, and various scripts needed throughout the project. Links to all four resources are provided below:

- Colab Notebook for SSD-MobileNet in a Wiped State: Link
- Colab Notebook for SSD-MobileNet in a Pre-Trained State: Link
- Colab Notebook for SSD-EfficientNet in a Wiped State: Link

- GitHub Repository for Project: Link

3 Results

Due to resource constraints in Colab, SSD-EfficientNet could only be trained with a batch size of 4 and 1000 training steps. We therefore provide results for both SSD-MobileNet Wiped and SSD-MobileNet Pre-Trained trained with the same parameters. The latter two were also trained with 2000 steps in separate training runs, and the learning rate was adjusted to test for the best value across multiple runs. The following are the results from the various runs.

3.1 SSD-EfficientNet

SSD-EfficientNet Round 1

Training Steps: 1000
Batch Size: 4
Base Learning Rate: .09
Warmup Learning Rate: .002

This network trained under these conditions took 2.5 hours to train on a TPU hosted by Google, with a per-step training time of roughly 8 seconds. It achieved a classification loss of 0.55, a localization loss of 0.58, and a regularization loss of 0.21. The total loss was 1.34. The practical outcome of this was that the network did not achieve a confidence level higher than 36 across all drawn bounding boxes, and there were multiple false positives and multiple missed grape clusters. Example output from this network appears in Appendix Figures 1 and 2.

SSD-EfficientNet Round 2

Training Steps: 1000
Batch Size: 4
Base Learning Rate: .09
Warmup Learning Rate: .02

This network trained under these conditions achieved a classification loss of 0.48, a localization loss of 0.52, and a regularization loss of 0.21. The total loss was 1.2209. The practical outcome of this was there were multiple false positives in the dark-grape test image. However, the model seemed to locate light-colored grapes more accurately than darker-colored ones. Example output from this network appears in Appendix Figures 3 and 4.

3.2 SSD-MobileNet Wiped

SSD-MobileNet Wiped Round 1

Training Steps: 1000
Batch Size: 4
Base Learning Rate: .10
Warmup Learning Rate: .03

This network trained under these conditions took 30 minutes to train on a Google TPU, with a per-step training time of just under 2 seconds. It achieved a classification loss of 1.11, a localization loss of 0.79, and a regularization loss of 0.22. The total loss was 2.12. The practical outcome of this was that the maximum confidence achieved did not exceed 40 percent and the network drew multiple incorrect bounding boxes and missed most of the grape clusters. Example output from this network appears in Appendix Figures 5 and 6.

SSD-MobileNet Wiped Round 2

Training Steps: 2000
Batch Size: 4
Base Learning Rate: .10
Warmup Learning Rate: .03

This network trained under these conditions took just over an hour to train, with a per step training time of just under 2 seconds on a Google TPU. It achieved a classification loss of 0.78, a localization loss of 0.73, and a regularization loss of 0.32. The total loss was 1.84. The practical outcome of this was that the maximum confidence achieved was 47 percent, with multiple false positives and missed grapes. Example output from this network appears in Appendix Figures 7 and 8.

SSD-MobileNet Wiped Round 3

Training Steps: 1000
Batch Size: 4
Base Learning Rate: 0.5
Warmup Learning Rate: 0.08

This network trained under these conditions. It achieved a classification loss of 0.56, a localization loss of 0.71, and a regularization loss of 1.11. The total loss was 2.4. The practical outcome of this was that the maximum confidence achieved was 41 percent, with multiple false positives and missed grapes. Example output from this network appears in Appendix Figures 9 and 10.

SSD-MobileNet Wiped Round 4

Training Steps: 2000
Batch Size: 4
Base Learning Rate: 0.5
Warmup Learning Rate: 0.08

This network trained under these conditions achieved a classification loss of 0.60, a localization loss of 0.71, and a regularization loss of 0.7013 . The total loss was 2.02. The practical outcome of this was that the maximum confidence achieved was 50 percent, with multiple false positives and missed grapes. Example output from this network appears in Appendix Figures 11 and 12.

3.3 SSD-MobileNet Pre-Trained

SSD-MobileNet Pre-Trained Round 1

Training Steps: 1000
Batch Size: 4
Base Learning Rate: .10
Warmup Learning Rate: .03

This network trained under these conditions took under 10 minutes to train on a Google GPU, with a per step training time of around 0.18 seconds. It achieved a classification loss of 0.34, a localization loss of 0.31, and a regularization loss of 0.10. The total loss was 0.75. The practical outcome of this was that the network achieved confidence values of up to 79 percent and detected most of the grape clusters with fewer false positives and fewer missed grapes. Example output from this network appears in Appendix Figures 13 and 14.

SSD-MobileNet Pre-Trained Round 2

Training Steps: 2000
Batch Size: 4
Base Learning Rate: .10
Warmup Learning Rate: .03

This network trained under these conditions took under 10 minutes to train on a Google GPU, with a per step training time of around 0.18 seconds. It achieved a classification loss of 0.32, a localization loss of 0.46, and a regularization loss of 0.11. The total loss was 0.89. The practical outcome of this was that the network achieved confidence values exceeding 60 percent and detected most of the grape clusters with fewer false positives and fewer missed grapes. Example output from this network appears in Appendix Figures 15 and 16.

SSD-MobileNet Pre-Trained Round 3

Training Steps: 1000
Batch Size: 4
Base Learning Rate: .5
Warmup Learning Rate: .08

This network trained under these conditions achieved a classification loss of 0.71, a localization loss of 0.63, and a regularization loss of 0.33. The total loss was 1.69. The practical outcome of this was that the network achieved confidence values exceeding 40 percent and detected most of the grape clusters with some false positives and missed grapes. Example output from this network appears in Appendix Figures 17 and 18.

SSD-MobileNet Pre-Trained Round 4

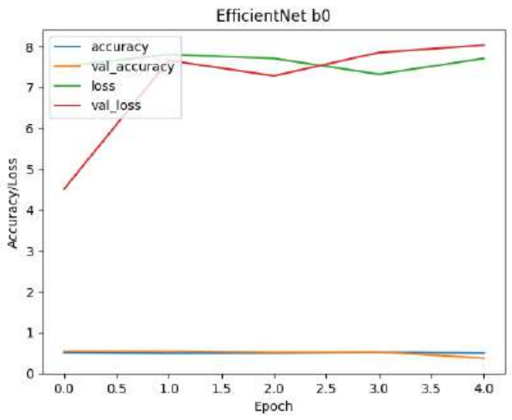
Training Steps: 2000
Batch Size: 4
Base Learning Rate: .5
Warmup Learning Rate: .08

This network trained under these conditions achieved a classification loss of 0.64, a localization loss of 0.62, and a regularization loss of 0.33. The total loss was 1.598. The practical outcome of this was that the network achieved confidence values exceeding 40 percent and detected most of the grape clusters with some false positives and missed grapes. Example output from this network appears in Appendix Figures 19 and 20.

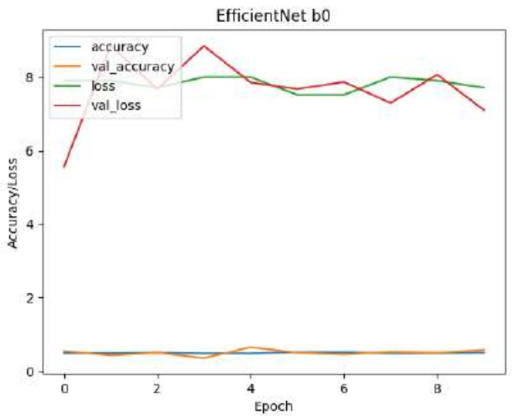
3.4 Custom EfficientNet Classification Performance

As the programming and training of a from-scratch EfficientNet classifier was not the primary focus of our project, we report our results with it only briefly here. The performance of our custom EfficientNet model with various hyper-parameter settings is demonstrated below. The low performance of our custom network could indicate that the training dataset was not sufficiently large to effectively train the classifier, the learning rates we chose to test were not appropriate for this task, or that measures against overfitting needed to be implemented.

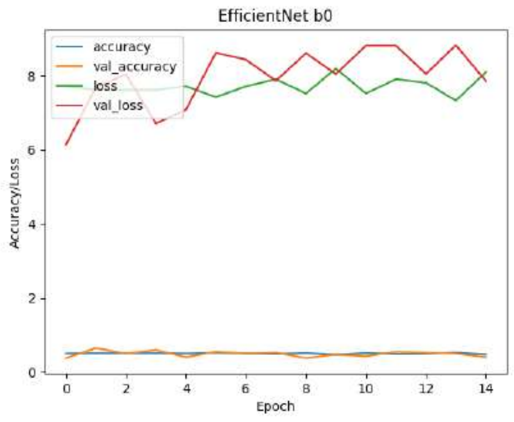
- 5 epochs, 1e-3 learning rate



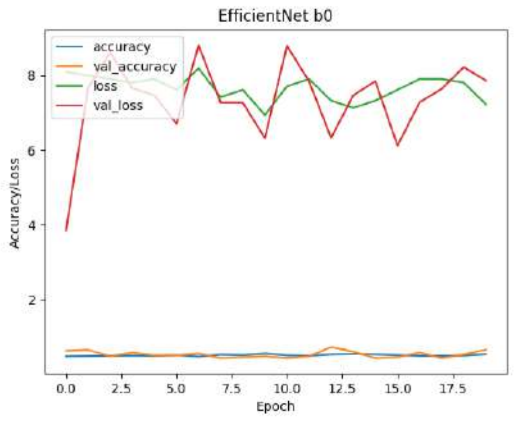
- 10 epochs, 1e-3 learning rate



- 15 epochs, 1e-5 learning rate



- 20 epochs, 1e-5 learning rate



4 Discussion

As could be expected based on prior literature, the SSD-MobileNet network performed best when left in a pre-trained state before being trained with the custom grapes dataset. This is likely due to the network already having learned general features of images which are useful for tasks such as object detection. Even though it had not seen grapes before, it was able to leverage this general knowledge to achieve better performance than it could achieve in a wiped state. Curiously, we found that this network sometimes performed less well with more steps or a higher learning rate. This may indicate that grape detection is a relatively simple task for this network in a pre-trained state and additional training time leads to overfitting. Future investigations could investigate this by running additional training sessions with a varying number of steps and employing measures such as early stopping to act against overfitting.

SSD-EfficientNet performed better than SSD-MobileNet when both were trained from a wiped state. It is especially notable that this was the case even when comparing SSD-EfficientNet, which here was trained only for 1000 steps due to resource constraints, with the SSD-MobileNet that was allowed to train for 2000 steps. Even with twice as much training time, SSD-MobileNet still had the lower performance. This may indicate that EfficientNet is a better backbone network for SSD-style object detectors, at least as concerns basic tasks like fruit detection.

References

Azunre, P., Osei, S., Addo, S., Adu-Gyamfi, L. A., Moore, S., Adabankah, B., Opoku, B., Asare-Nyarko, C., Nyarko, S., Amoaba, C., et al. (2021). Contextual text embeddings for twi. *arXiv preprint*. https://doi.org/https://doi.org/10.48550/arXiv.2103.15963

Biswas, D., Su, H., Wang, C., Stevanovic, A., & Wang, W. (2019). An automatic traffic density estimation using single shot detection (ssd) and mobilenet-ssd [Sensing and Sensor Systems for Urban Environmental Studies]. *Physics and Chemistry of the Earth, Parts A/B/C*, *110*, 176–184. https://doi.org/https://doi.org/10.1016/j.pce.2018.12.001

Duong-Trung, N., Tran Ngoc, T., & Huynh, H. X. (2019). Automated pneumonia detection in x-ray images via depthwise separable convolution based learning. https://doi.org/https://doi.org/10.15625/vap.2019.0005

Fernandez, I. G., & Wada, C. (2020). Shoe detection using ssd-mobilenet architecture. *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*, 171–172. https://doi.org/10.1109/LifeTech48969.2020.1570618965

Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2017). Squeeze-and-excitation networks. https://doi.org/https://doi.org/10.48550/arXiv.1709.01507

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. https://doi.org/https://doi.org/10.48550/arXiv.1603.09382

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot MultiBox detector. In *Computer vision – ECCV 2016* (pp. 21–37). Springer International Publishing. https://doi.org/10.1007/978-3-319-46448-0_2

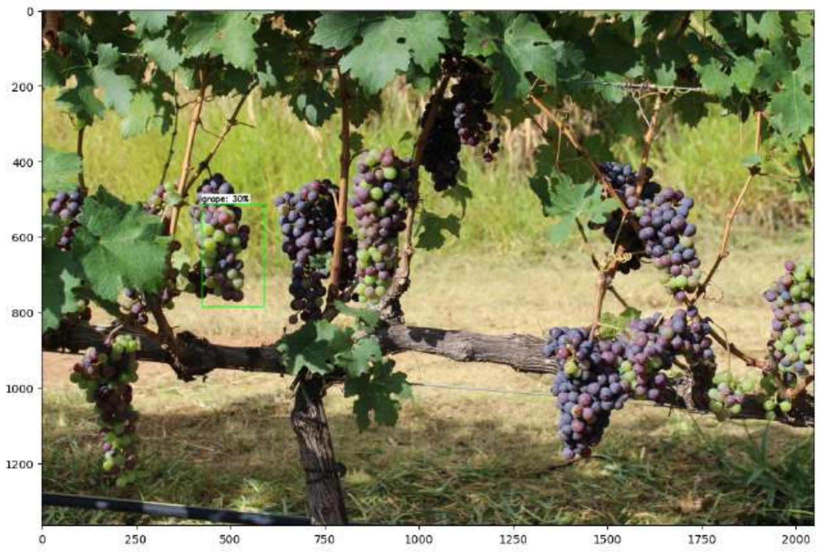
Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/https://doi.org/10.48550/arXiv.1801.04381

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*. https://doi.org/https://doi.org/10.48550/arXiv.1905.11946

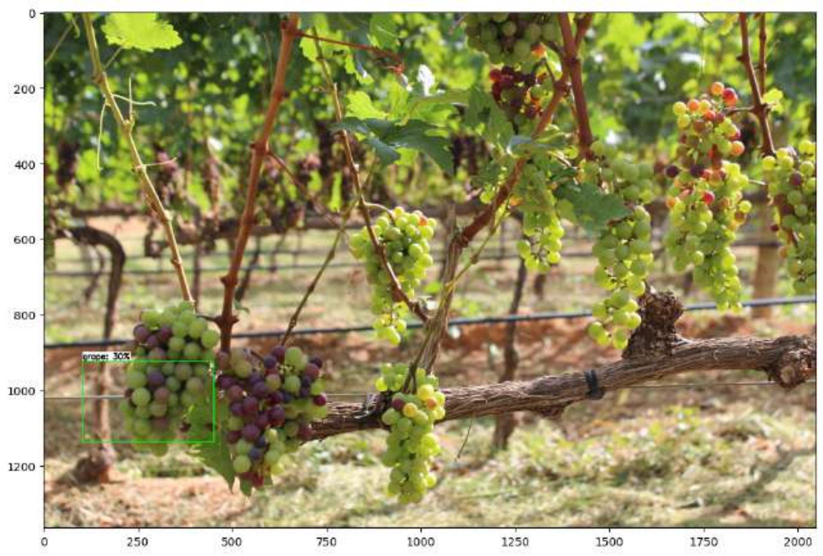
Yu, G., Wang, L., Hou, M., Liang, Y., & He, T. (2020). An adaptive dead fish detection approach using ssd-mobilenet. *2020 Chinese Automation Congress (CAC)*, 1973–1979. https://doi.org/10.1109/CAC51589.2020.9326648

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. https://doi.org/https://doi.org/10.48550/arXiv.1605.07146

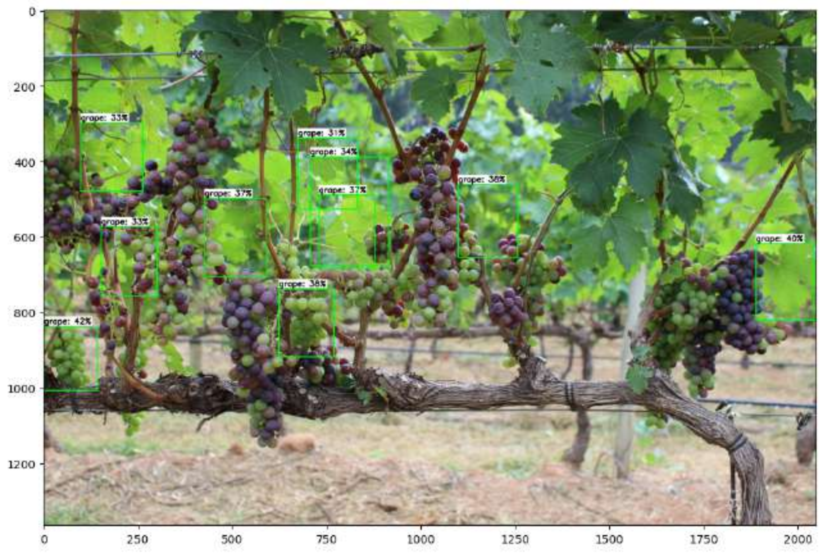
Appendix



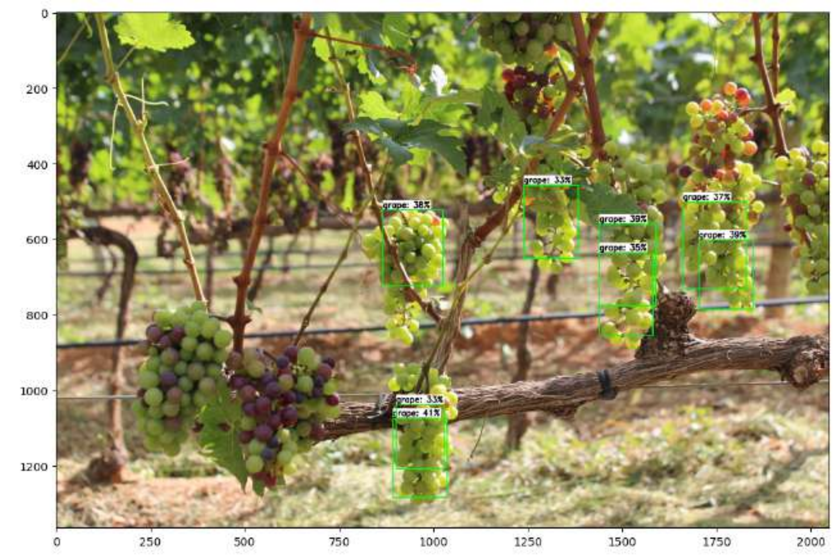
Appendix Figure 1: Example output from our trained SSD-EfficientNet in Round 1 Configuration. This image is balanced towards testing detection of dark-colored grapes.



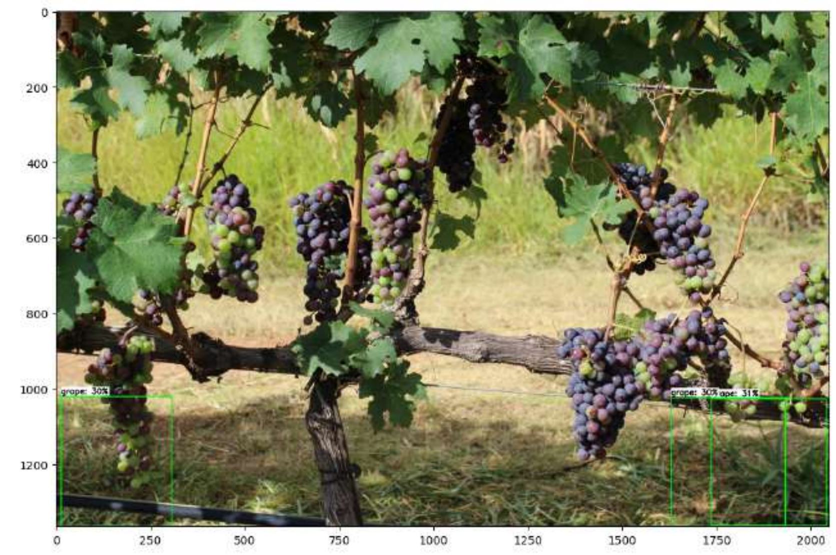
Appendix Figure 2: Example output from our trained SSD-EfficientNet in Round 1 Configuration. This image is balanced towards testing detection of light-colored grapes.



Appendix Figure 3: Example output from our trained SSD-EfficientNet in Round 2 Configuration. This image is balanced towards testing detection of dark-colored grapes.

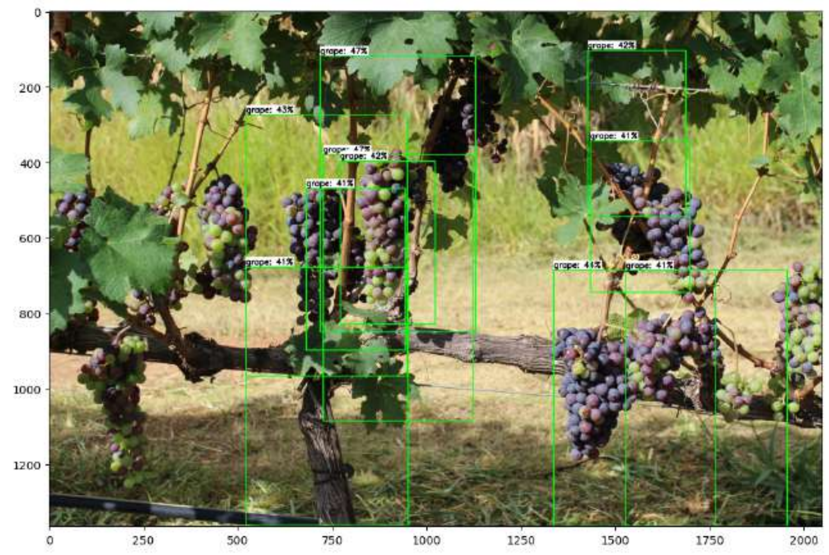


Appendix Figure 4: Example output from our trained SSD-EfficientNet in Round 2 Configuration. This image is balanced towards testing detection of light-colored grapes.

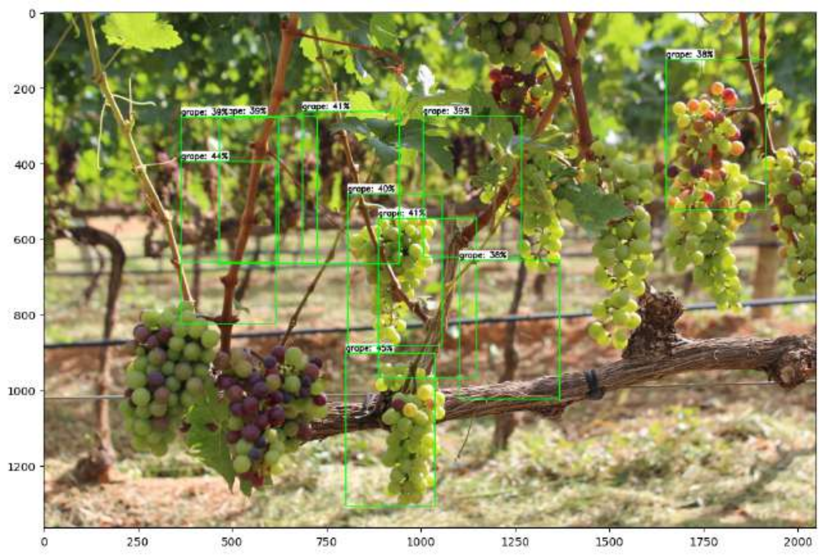


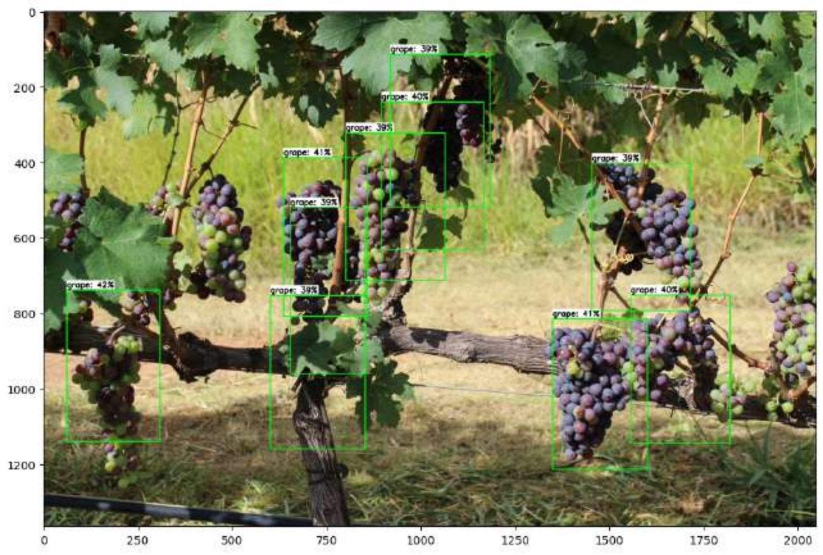
Appendix Figure 5: Example output from our trained SSD-MobileNet Wiped in Round 1 Configuration. This image is balanced towards testing detection of dark-colored grapes.

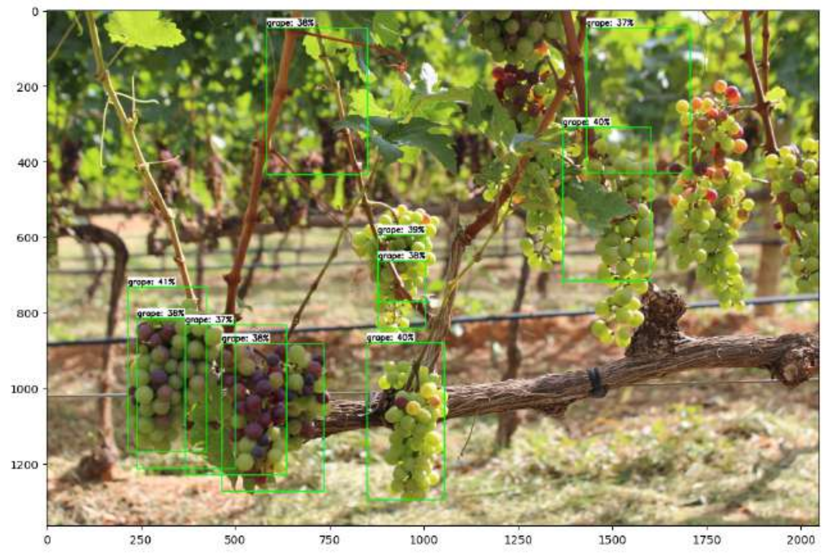
Appendix Figure 6: Example output from our trained SSD-MobileNet Wiped in Round 1 Configuration. This image is balanced towards testing detection of light-colored grapes.



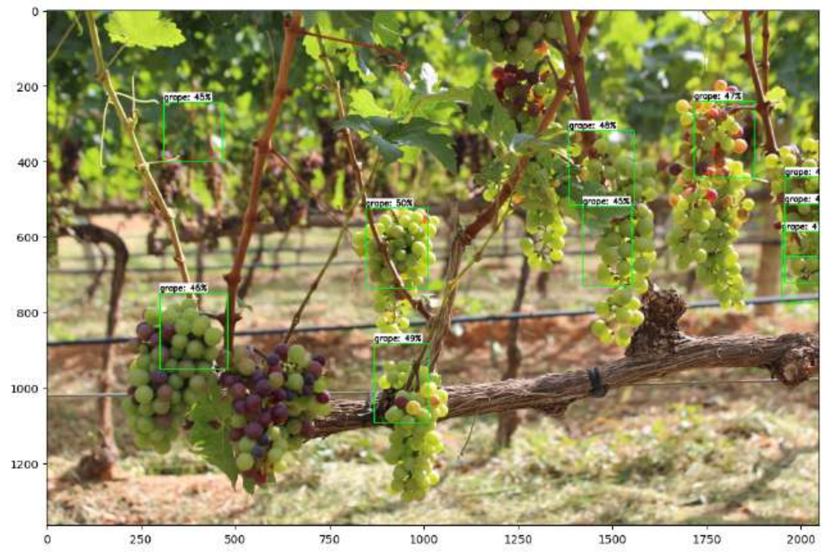
Appendix Figure 7: Example output from our trained SSD-MobileNet Wiped in Round 2 Configuration. This image is balanced towards testing detection of dark-colored grapes.


Appendix Figure 8: Example output from our trained SSD-MobileNet Wiped in Round 2 Configuration. This image is balanced towards testing detection of light-colored grapes.

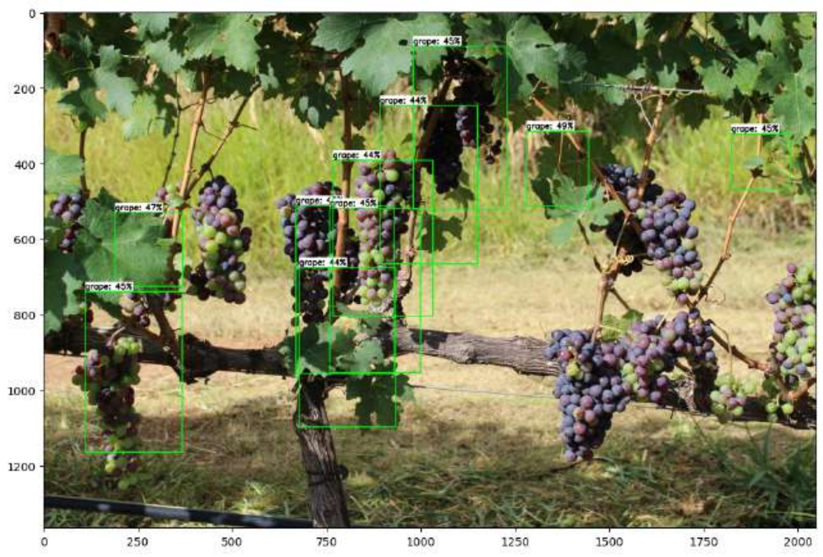

Appendix Figure 9: Example output from our trained SSD-MobileNet Wiped in Round 3 Configuration. This image is balanced towards testing detection of light-colored grapes.



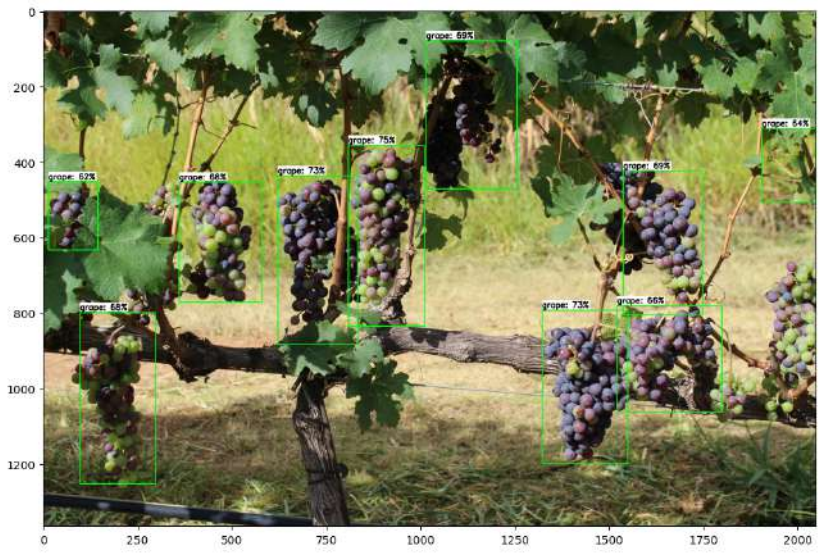
Appendix Figure 10: Example output from our trained SSD-MobileNet Wiped in Round 3 Configuration. This image is balanced towards testing detection of light-colored grapes.



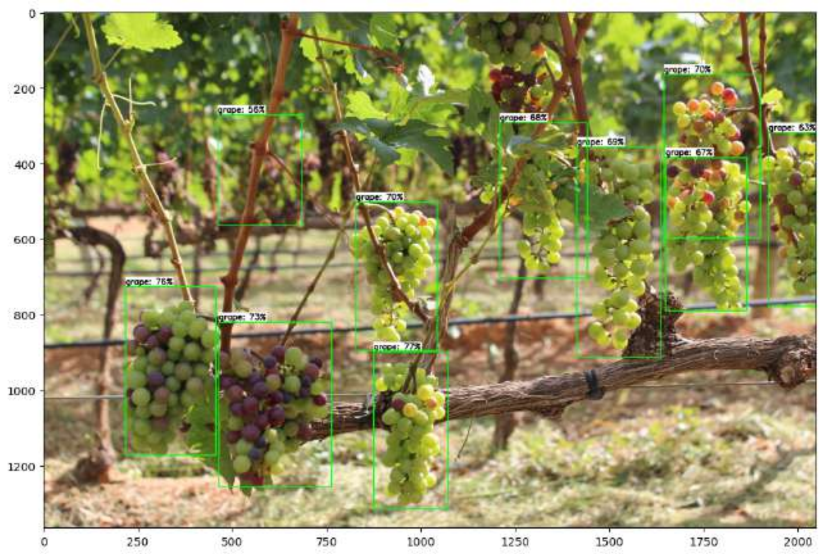
Appendix Figure 11: Example output from our trained SSD-MobileNet Wiped in Round 2 Configuration. This image is balanced towards testing detection of light-colored grapes.



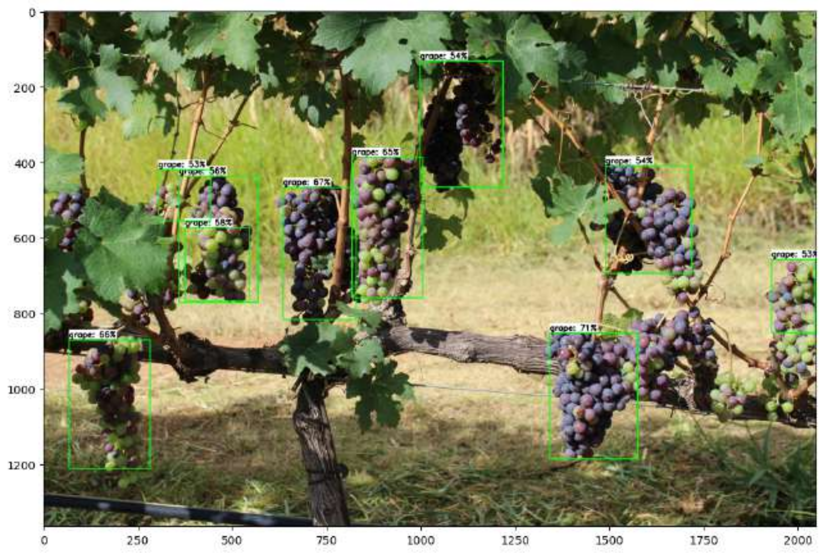
Appendix Figure 12: Example output from our trained SSD-MobileNet Wiped in Round 2 Configuration. This image is balanced towards testing detection of dark-colored grapes.



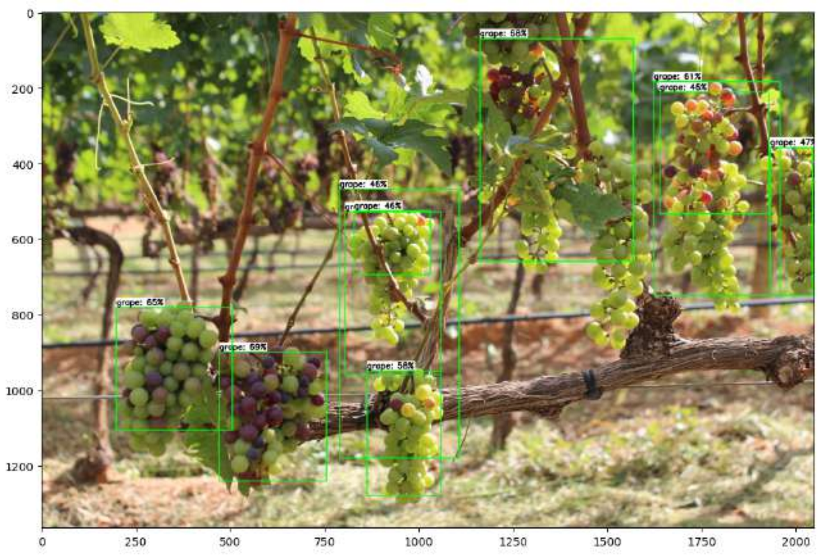
Appendix Figure 13: Example output from our trained SSD-MobileNet Pre-Trained in Round 1 Configuration. This image is balanced towards testing detection of dark-colored grapes.



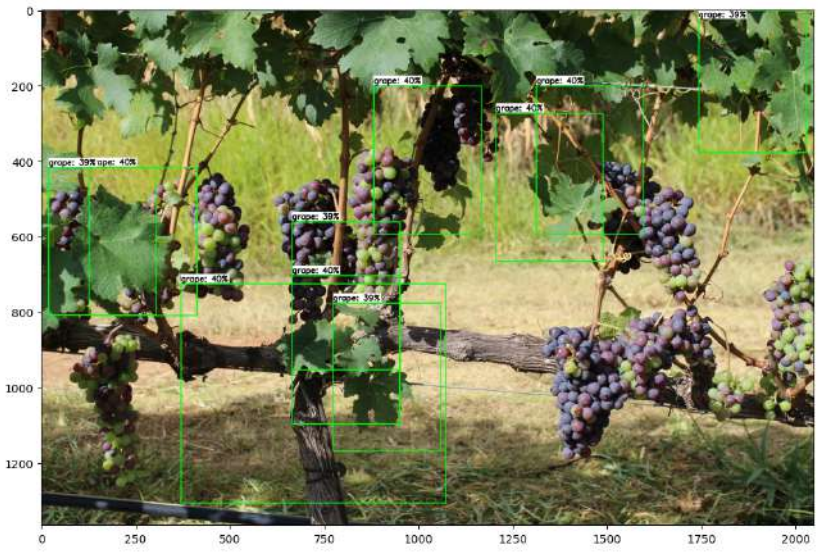
Appendix Figure 14: Example output from our trained SSD-MobileNet Pre-Trained in Round 1 Configuration. This image is balanced towards testing detection of light-colored grapes.



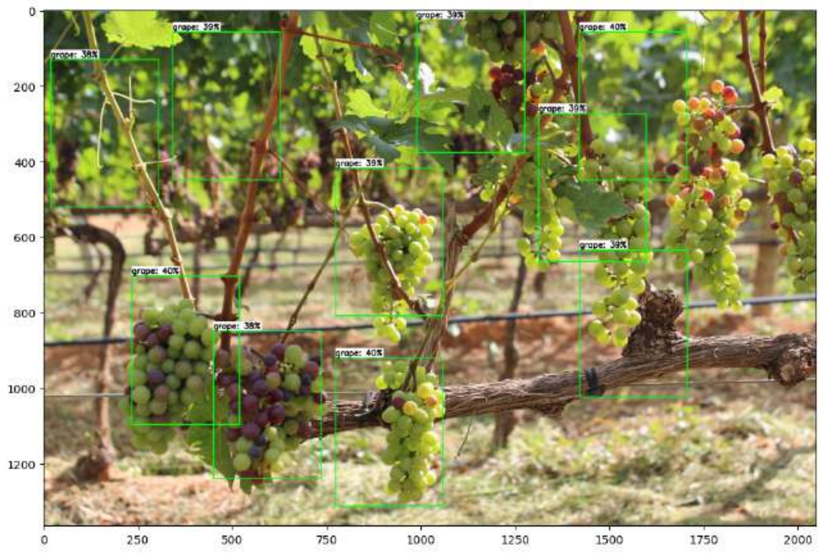
Appendix Figure 15: Example output from our trained SSD-MobileNet Pre-Trained in Round 1 Configuration. This image is balanced towards testing detection of dark-colored grapes.



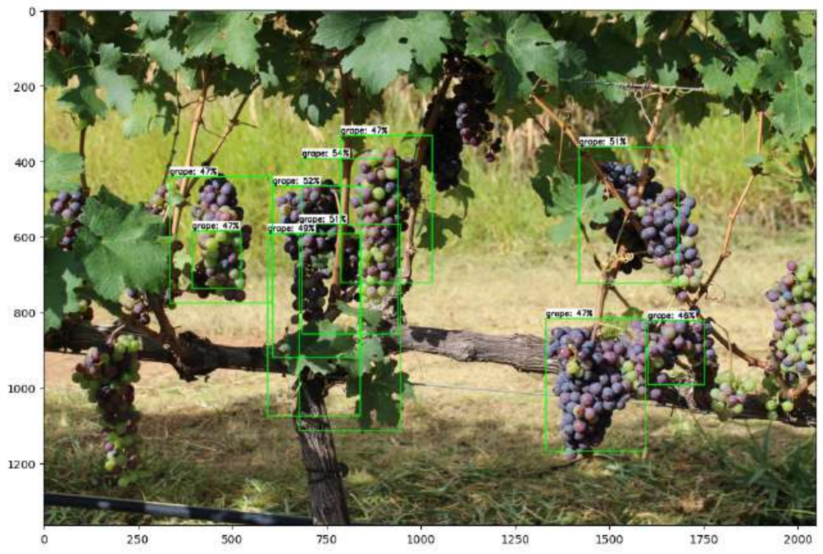
Appendix Figure 16: Example output from our trained SSD-MobileNet Pre-Trained in Round 1 Configuration. This image is balanced towards testing detection of light-colored grapes.



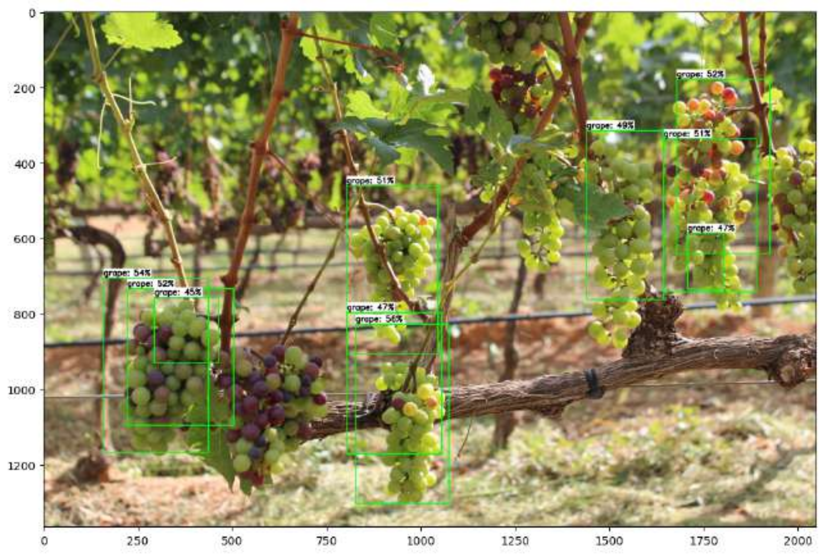
Appendix Figure 17: Example output from our trained SSD-MobileNet Pre-Trained in Round 3 Configuration. This image is balanced towards testing detection of dark-colored grapes.



Appendix Figure 18: Example output from our trained SSD-MobileNet Pre-Trained in Round 3 Configuration. This image is balanced towards testing detection of light-colored grapes.



Appendix Figure 19: Example output from our trained SSD-MobileNet Pre-Trained in Round 4 Configuration. This image is balanced towards testing detection of dark-colored grapes.



Appendix Figure 20: Example output from our trained SSD-MobileNet Pre-Trained in Round 4 Configuration. This image is balanced towards testing detection of light-colored grapes.