University of Washington

Computer Vision: Classical and Deep Methods

EE P 596

The Veggie Classifier

Blake Downey

December 16th, 2021

INTRODUCTION

Classification is a fluid problem that has been tried again and again over the years, especially with the ILSVRC from 2010 to 2017. Now, living in the time that we do, being able to download entire datasets, use pretrained models, transfer learn, and the endless things that you can do with PyTorch and Keras for deep learning, we can tackle just about any classification problem if we have enough data to do so. I took my own approach and looked through some datasets that were easy to get my hands on. What I found was that many of the images in these datasets were a single object on a white background. In terms of pure classification of a test image following the same pattern of an object on a white background, this works with very high accuracy. Keeping this in mind, I populated my dataset with images that were more generalized, because at the end of the day whether a bell pepper is green, red, yellow, cut in half or sliced, it is still a bell pepper. The problem that I was trying to solve was can a DNN correctly classify the same objects if the images for training and testing are not just an object on a white background.

Part of the motivation for this problem was taking classification from a different perspective with more generalized data to produce good results. Another motivation for this project was very simple: Though I am able to differentiate between particular vegetables like cucumbers and zucchinis, can a trained NN do better than me? I knew that it was going to be challenging to start this whole thing from scratch, collecting all of my data, writing code to label the data correctly and creating augmentation functions that actually generate good data instead of just garbage data. That last part rings especially true since after all, data is the algorithm.

There are several simple projects out on the internet with image classification of vegetables [1][2] for example, but each was using Fruits-360 by Kaggle [3]. This kaggle dataset I looked at prior to creating my own. I decided that this dataset, along with others, were not going to work for my proposed generalization idea since these images were single objects on white backgrounds.

The key contributions of this project are a generalized dataset that contains categories that are found in ImageNet so that I can have very simple baseline results, a pretrained model (ResNet-34) which is used for baseline, and transfer learning with my dataset to attempt to improve generalizing within the categories I chose.

METHOD

Simply put, the categories: bell pepper, broccoli, cauliflower, cucumber, mushroom and zucchini, can all be found in ImageNet categories [4], so this made baseline testing incredibly straight forward (Appendix Figure 4). All I needed was my generated test sets for V1 and V2 and run those through the ResNet-34 architecture that has been pre-trained on ImageNet, and chart my results.

My method was two part: V1 which has 5 categories, 15 images per category and 3 cases of successively deeper transfer learning, and V2 which has 6 categories, 20 images per category and uses case 2 from V1 which was the best performing case.

Looking at the very high level layer by layer definition of ResNet-34 on the right, there are three chunks of layer freezing on the bottom. These are labeled 'resnet34_fc', 'resnet34_fc_4', and 'resnet34_fc_4_3'. These are the 3 cases, 0,1 and 2 respectively, and as the code suggests, in case 0 only the fully connected layer is unfrozen for training, and successively layer 4 and layer 3 are unfrozen in addition to the fc layer in cases 1 and 2. I used 3 different cases because I wasn't absolutely positive that the network would perform better or worse with deeper retraining.

```
Defining ResNet34 retrain models...

ResNet34 Layer Definition:
conv1
bn1
relu
maxpool
layer1
layer2
layer3
layer4
avgpool
fc

ResNet34 ("resnet34_fc") with fc unfrozen:
fc is unfrozen
-- all other layers remain frozen

ResNet34 ("resnet34_fc_4") with fc and layer4 unfrozen:
layer4 is unfrozen
fc is unfrozen
-- all other layers remain frozen

ResNet34 ("resnet34_fc_4_3") with fc, layer4 and layer3 unfrozen:
layer3 is unfrozen
layer4 is unfrozen
fc is unfrozen
-- all other layers remain frozen
```

As shown in figure 1 below, I have the two versions V1 and V2. In V1 I have the section labeled 'Dataset Generation' which is where I do all of my data augmentation. For visualization purposes the flow of the chart above doesn't exactly match the directory and code flow for data generation, but instead serves a more comprehensive breakdown. For both V1 and V2 the process was the same; gather images of the categories and store them in folders called train and test for each version. My code then reads those images using the glob module [5] and randomly augments those images with different functions that I wrote, 75 augmented images per original train and test image. The data augmentation contains functions that first crop and resize the image to 224 x 224 for the ResNet34 model, and then perform addition of gaussian noise, flipping, flopping, rotating, brightening/darkening and smoothing/sharpening. It took a while to get a good set of images since gaussian noise is a very overpowering augmentation and when performing certain functions in combination they made horrible image results.

After determining the best combinations of data augmentation, I had train and test folders for each V1 and V2 (with the size of each labeled in figure 2). In the training

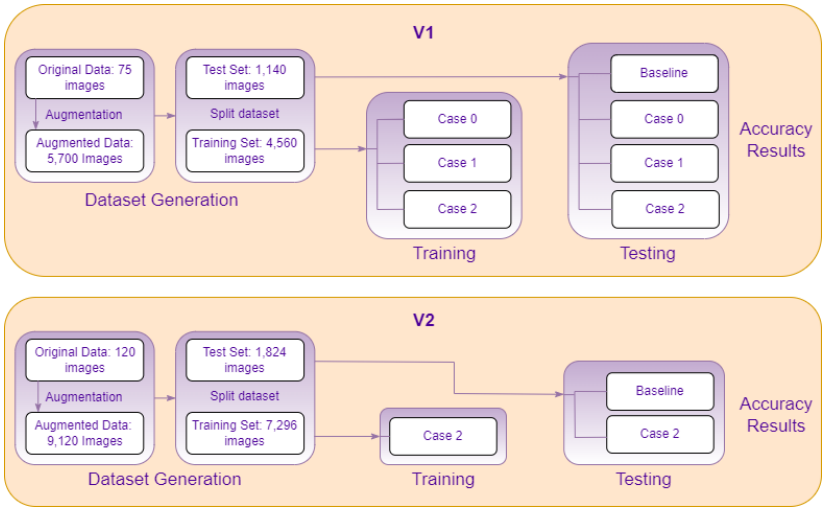


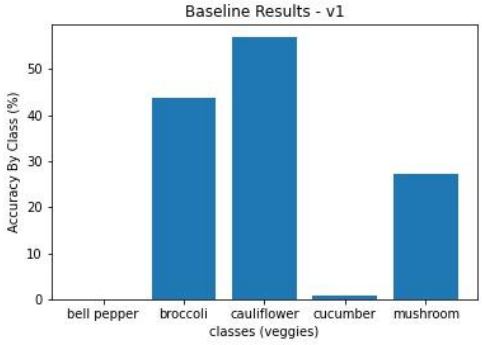
Figure 1. My Method

block of V1 in figure 1 I have the three cases with an arrow from the training set. I trained these models on the data, generated training loss curves, and saved each step of the model for test-error curves. In figure 1, V2 I only have the best performing case, case 2, from V1 and use the larger V2 training set to apply transfer learning again to the ResNet-34 model with the fc, layer4 and layer3 layers unfrozen. I generated training loss and models for each step for test-error curves as well. For all of the training in this project, I used a constant learning rate of 0.0005, trained for 5 epochs and saved models during training in periodic intervals for each version. With the models for V1 and V2 trained, I was able to simply run my test data on cases 0, 1, and 2 for V1 and case 2 for V2 and compare those results with the baseline results generated by the flow in figure 1.

RESULTS

Before getting to the accuracy results, a quick section on the training loss curves for each V1 and V2. From Appendix Figure 3, only looking at the blue lines (the best performing case, case 2), we can note that the training loss sees its biggest decrease in the first 3-4 iterations (steps of the saved models during training time). The first 3-4 iterations are between the first and second epochs of training. After that the training loss decreases rather slowly but is a downward slope. After 5 epochs of training, both V1 and V2 training loss is ~0.05. That is a very small training loss and is susceptible to overfitting.

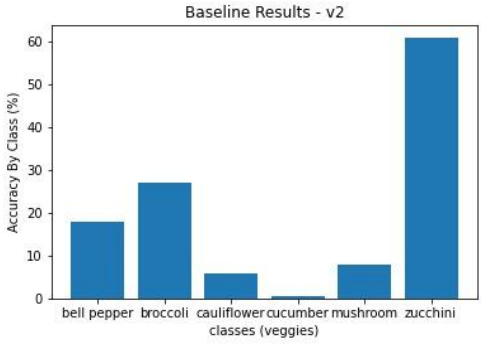
For quantifying this project, I used class accuracy and total accuracy. The baseline results are listed below in figure 2:



Accuracy for Baseline Method -- v1 by class

bell pepper : 0.00 %
broccoli : 43.86 %
cauliflower : 57.02 %
cucumber : 0.88 %
mushroom : 27.19 %

Total accuracy -- v1: 25.79 %



Accuracy for Baseline Method -- v2 by class

bell pepper : 17.82 %
broccoli : 26.97 %
cauliflower : 5.92 %
cucumber : 0.66 %
mushroom : 7.89 %
zucchini : 60.86 %

Total accuracy -- v2: 20.02 %

Figure 2. Baseline Results by Category and Total Accuracy

The biggest takeaway from the baseline testing is that these are all categories that should in theory test well on a pretrained model if the images in ImageNet were more generalized. An interesting note is that zucchini in V2 tested very well in comparison to the other categories. I believe that is because of the types of images in my V2 test folder for zucchini. Moving to the class and total accuracy of V1 case 2, and V2 case 2, we have results shown below in figure 5:

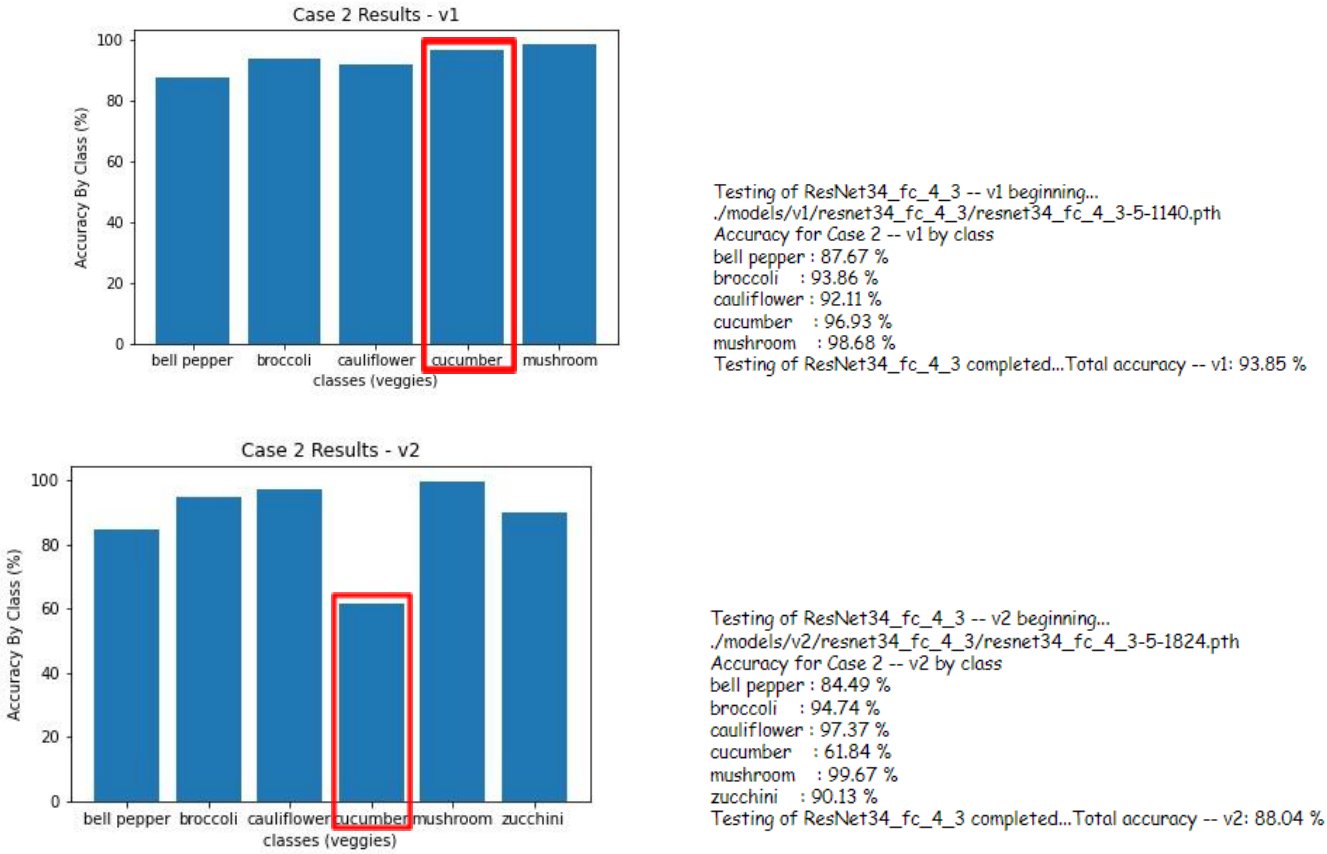


Figure 3. V1 and V2 Class and Total Accuracy Results

Across the results for V1 (cases 0 and 1 for V1 can be found in the Appendix Figures 5 and 6), the total accuracy increased, as hypothesized, as the cases progressed with deeper transfer learning. With the final bar graph for V1 titled 'Case 2 Results - v1' we can see that each category tested very well with a total accuracy of 93.85%. These results were really promising and showed that the DNN can indeed learn with a more generalized set of data and not simply an object on a white background. Looking at the V2 results, we can make a large discovery (boxed in red in the last two bar graphs above) that the addition of zucchini actually significantly lowered the accuracy of cucumbers from 96.93% in case 2 V1 to 61.84% in case 2 V2. This means that the lowered total accuracy of the network was in large part due to the significant lowering of the cucumber category. The test accuracy and test error curves can be found in the appendix labeled Appendix Figure 1 and 2.

CONCLUSION

By choosing this simple experiment of classifying vegetables with my self generated dataset, I was able to solidify my learned knowledge from the entire quarter as I used concepts from nearly every homework assignment. From my results, I learned that it doesn't really matter how generalized the data is, the DNN model can accurately predict the category if it's given enough data to train with. Another takeaway, is that even though the results for V2 showed lowered accuracy of cucumbers from the previous version without zucchinis, if the network was fed better images and more images, I believe that the zucchini / cucumber debacle would be resolved and that the DNN could indeed improve the accuracy of cucumbers back to the ~95% range.

This project could be improved with another case where the network isn't pre-trained and I use my dataset alone to train the entire network and compare the results across my cases. In terms of where this project could go, I do truly believe that an idea like this could have big implications for the future of AR glasses. Let's take the scenario of a small architecture with high accuracy, capable of running real time image classification, which then uses a speaker to speak the results to the user. So people hard of sight would be able to hold different vegetables up and the glasses would speak to the user what the vegetable is. I think this application is a long way out from being feasible but it is an interesting idea of where this project could go with modification.

References

[1] Blogs, Code AI. “Classifying Fruits and Vegetables with Machine Learning.” *Medium*, CodeAI, 23 Aug. 2021, https://medium.com/m2mtechconnect/classifying-fruits-and-vegetables-with-machine-learning-ebc3c1cca82e.

[2] Dominic, Jose. “Building a Deep Learning Model with Pytorch to Classify Fruits and Vegetables.” *Medium*, The Startup, 2 July 2020, https://medium.com/swlh/building-a-deep-learning-model-with-pytorch-to-classify-fruits-and-vegetables-30e1a8ffbe8c.

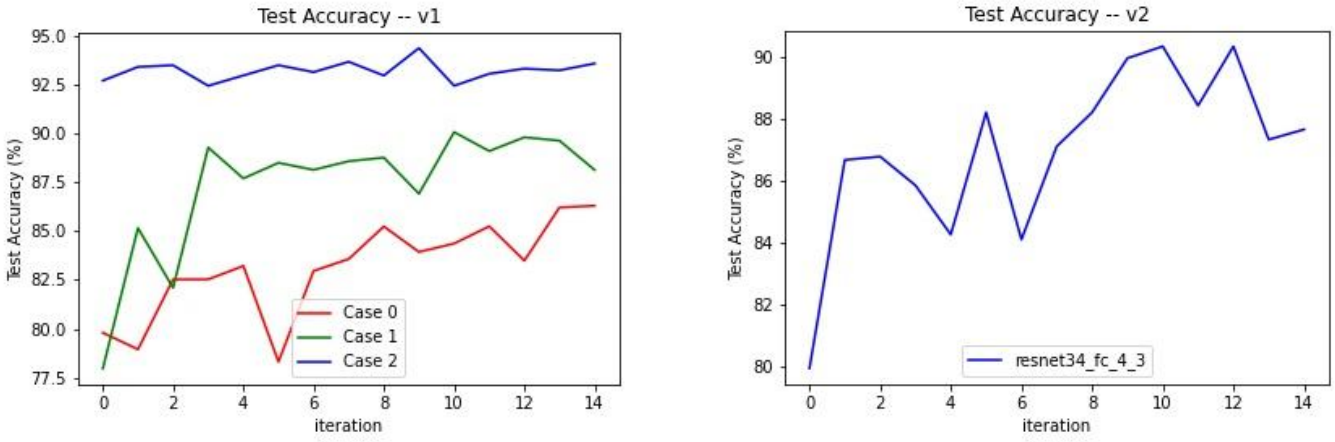
[3] Oltean, Mihai. “Fruits 360.” *Kaggle*, 12 Sept. 2021, https://www.kaggle.com/moltean/fruits/version/9.

[4] “IMAGENET 1000 Class List.” *IMAGENET 1000 Class List - WekaDeeplearning4j*, https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/.

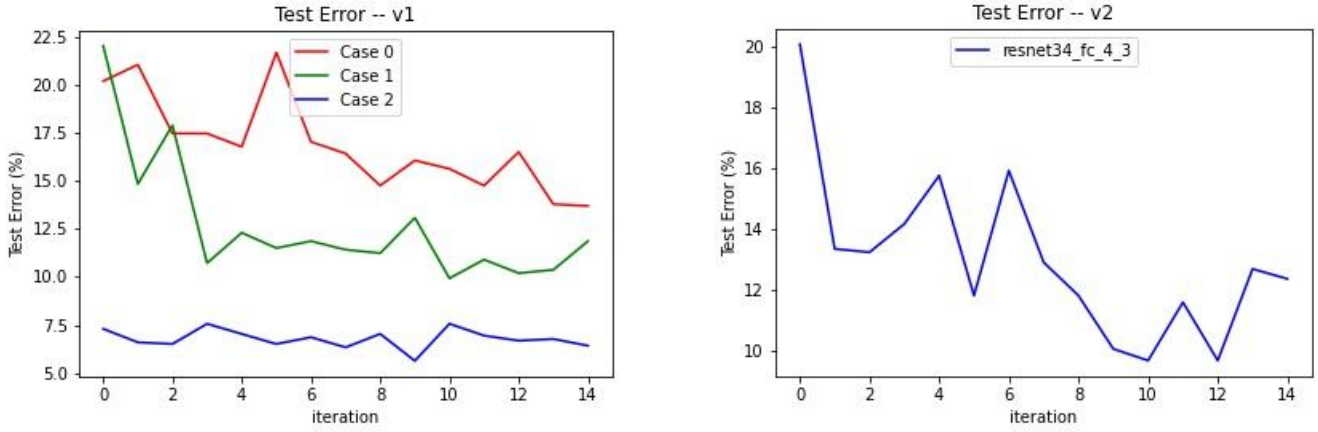
[5] “Glob - Unix Style Pathname Pattern Expansion¶.” *Glob - Unix Style Pathname Pattern Expansion - Python 3.10.1 Documentation*, https://docs.python.org/3/library/glob.html.

Appendix

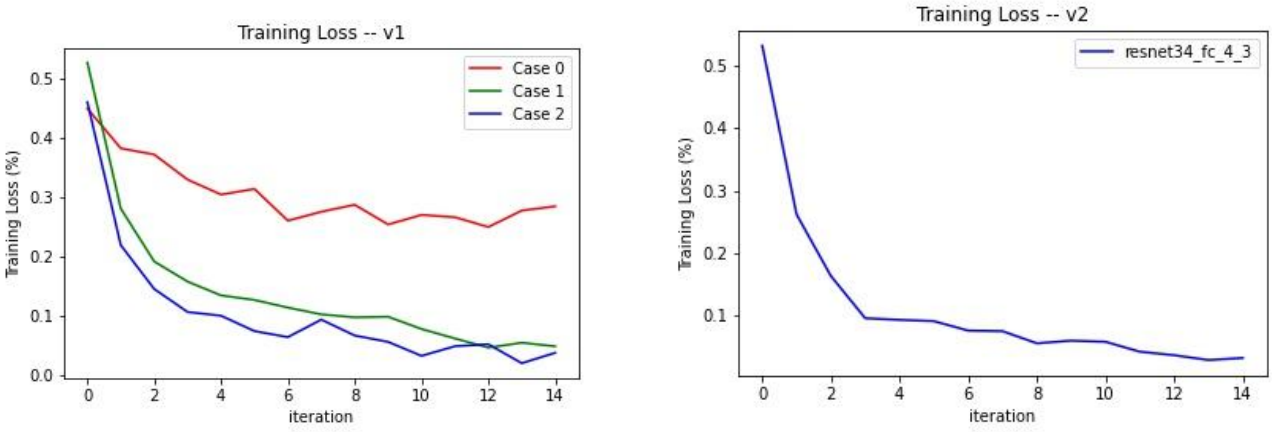
Please find my code on my github @ https://github.com/bdowney49/veggie-classifier.git



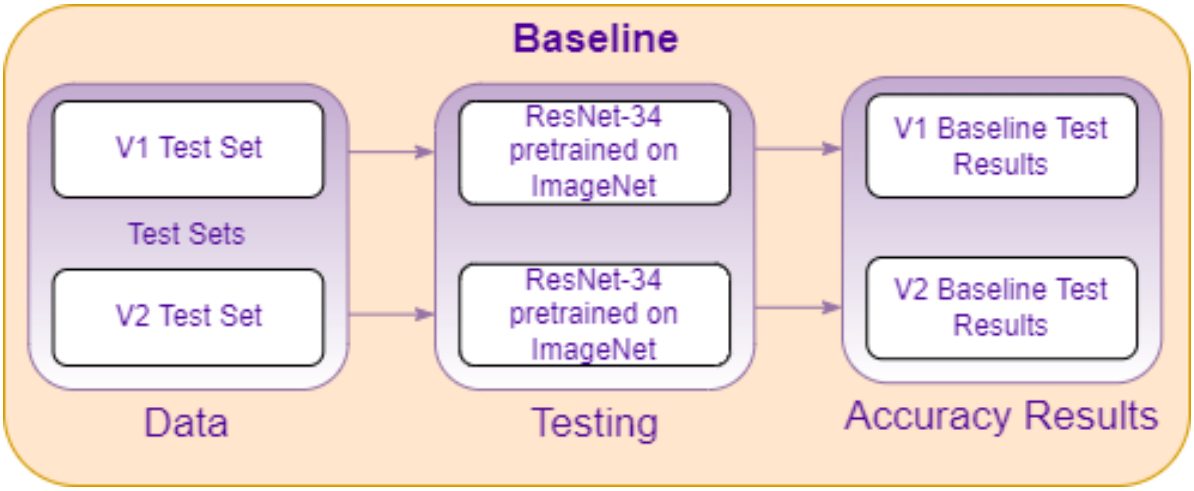
Appendix Figure 1. Test Accuracy Curves from V1 and V2



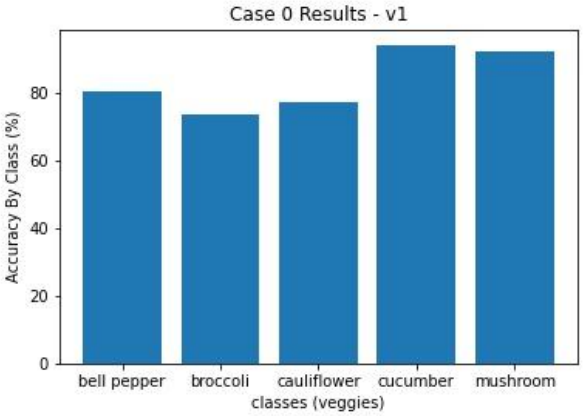
Appendix Figure 2. Test Error Curves from V1 and V2



Appendix Figure 3. Training Loss

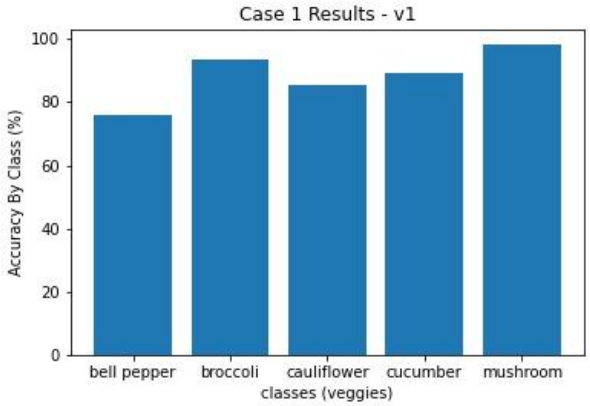


Appendix Figure 4. Baseline Model Flow




```
Testing of ResNet34_fc -- v1 beginning...
./models/v1/resnet34_fc/resnet34_fc-5-1140.pth
Accuracy for Case 0 -- v1 by class
bell pepper : 80.62 %
broccoli    : 73.68 %
cauliflower : 77.19 %
cucumber    : 94.30 %
mushroom    : 92.54 %
Testing of ResNet34_fc completed...Total accuracy -- v1: 83.67 %
```

Appendix Figure 5. Case 0 V1 Results



Testing of ResNet34_fc_4 -- v1 beginning...
./models/v1/resnet34_fc_4/resnet34_fc_4-5-1140.pth
Accuracy for Case 1 -- v1 by class
bell pepper : 75.77 %
broccoli : 93.42 %
cauliflower : 85.53 %
cucumber : 89.04 %
mushroom : 98.25 %
Testing of ResNet34_fc_4 completed...Total accuracy -- v1: 88.40 %

Appendix Figure 6. Case 1 V1 Results