Vegetable Image Classification: Use of Transformer Architecture and Captum Interpretability in Image Classification

Cooper Atkins

I. Introduction

The goal of this project is to be able to identify and classify different types of vegetables for the real world application of helping someone visually impaired understand what vegetable they're looking at or holding. The project, at the broad level, focuses on developing a CNN as the benchmark model and developing a more complicated model as the primary model, which in this case was a ResNet transformer model. The work was divided between the groupmates according to skillset and comfort with the various pieces.

II. Description of Individual Work

My individual work was developing the CNN model and testing various iterations to achieve a relative success on the benchmark. We could've chosen an MLP or another benchmark that would objectively be worse, but we decided that using a well trained and developed CNN model as the benchmark would serve two potential purposes. The first purpose was that if the CNN model was far worse than the ResNet model, then it would be clear that the more complicated ResNet model may be necessary to achieve the desired accuracy. Alternatively, if the CNN showed similar performance to the ResNet model, it could be determined that for this setup, and likely similar setups, that a CNN model would suffice and that the compute power required for the ResNet was unnecessary to achieve success.

Beyond the development of the CNN model, I developed the EDA and post-hoc files and analysis. This included determining class balances within each dataset, showing sample images from each class, and building heatmap confusion matrices to demonstrate the performance of each model against each class.

Lastly, I was also responsible for the code that allowed the training loop to save the best model, which in our case happened to match directly with the epochs, but is capable of not saving when a subsequent epoch fails to be better. This allows us to separate the testing and training files. I also developed the code to show and print out the loss and accuracy curves for both the benchmark and primary models. All images were chosen to be saved as png files to avoid needing to retrain the models each time we wanted to

see the trend of the metrics, as well as for ease of export for presentations and reports. All saving features are togglable in all files by changing the inputs from “True” to “False”.

For our benchmark model we developed a convolution neural network. The choice to use a convolution network rather than a multi-layer perception is quite apparent as there are obvious pixel dependencies within the images selected for training (and indeed in any image that would be fed through the completed model. As a result, the CNN is the lowest complexity model that we’d be comfortable with using as a baseline or benchmark here before testing out more complex models as a primary.

For the benchmarked CNN, several iterations were tested including various numbers of convolution layers, pooling layer, dropout layers, linear layers, and hyperparameter changes. The final model contains 6 convolution layers, 5 pooling layers (where convolution layers 2 and 3 occur sequentially with no pooling layer in between) and no dropout layers followed by 3 successive linear layers using the ReLu function. Lastly, given this is a multiclass classification problem, the outputs are passed through a softmax layer to normalize the values such that an argmax() function can later be applied for classification. One thing to note here is that the first pooling layer uses maximum pooling, to help with noise suppression as well as dimensionality reduction. The remaining pooling layers all use average pooling which allows the images to be smoothed and can adapt well to different levels of contrast between the background and primary target within the images.

The data was normalized by my groupmate according to what was required for the primary ResNet model and was sufficient for use in the CNN.

Training, Validation, Test Metrics

Benchmark_training results:
Epoch 30
train_loss : 0.07811081487462561 val_loss : 0.06370944497377976
train_accuracy : 0.8514957264957265 val_accuracy : 0.9144021739130435

Benchmark_testing results:
Test Accuracy: 0.9156534954407295 Test Loss: 0.06209164560633771

Test Data: F1 Macro, Precision, Recall by class for Benchmark Model

Class	Precision	Recall	F1-Score

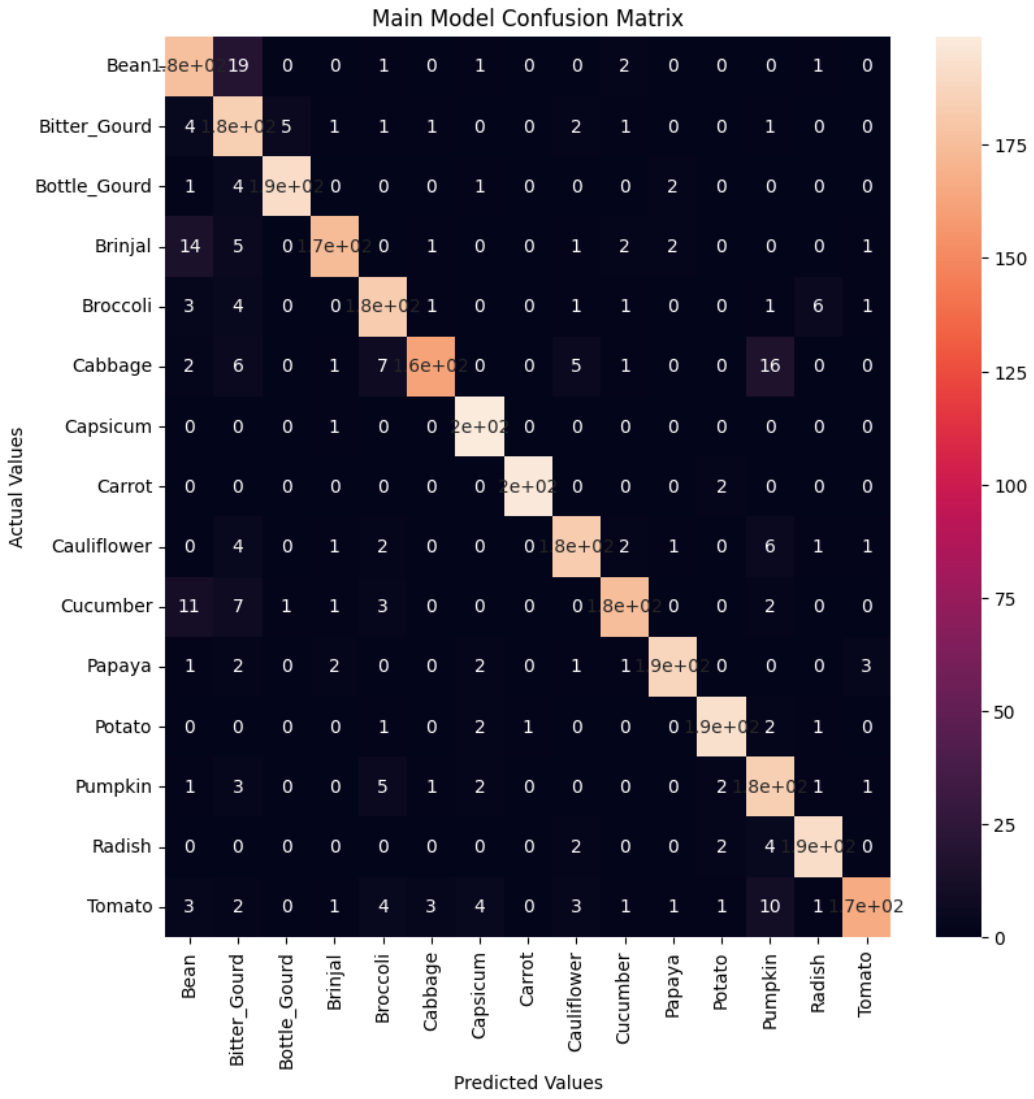
Bean	0.81	0.88	0.85
Bitter Gourd	0.77	0.92	0.84
Bottle Gourd	0.97	0.96	0.96
Brinjal	0.96	0.87	0.91
Broccoli	0.88	0.91	0.90
Cabbage	0.96	0.81	0.88
Capsicum	0.94	0.99	0.97
Carrot	0.99	0.99	0.99
Cauliflower	0.92	0.91	0.92
Cucumber	0.94	0.88	0.91
Papaya	0.97	0.94	0.95
Potato	0.96	0.96	0.96
Pumpkin	0.81	0.92	0.86
Radish	0.95	0.96	0.95
Tomato	0.96	0.83	0.89

Test Data: F1 Macro, Precision, Recall by class for Primary Model

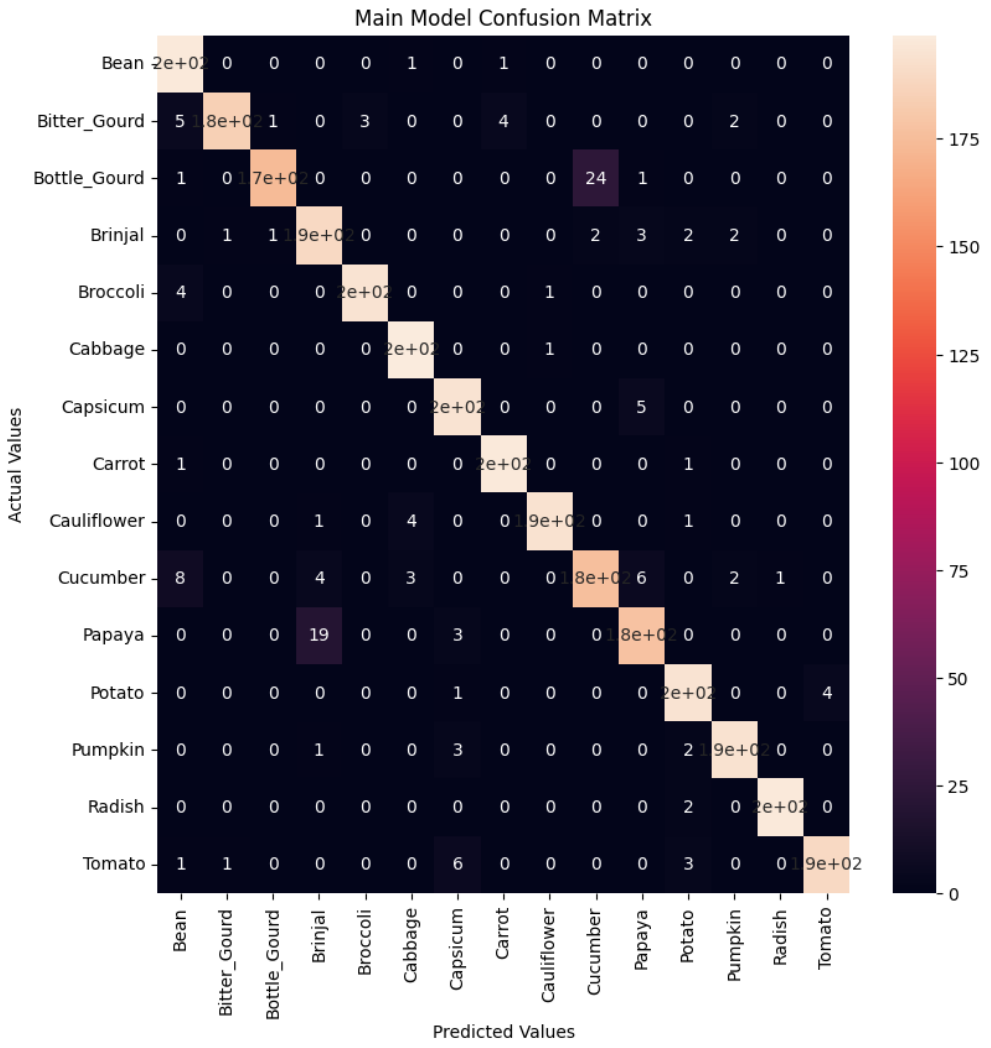
Class	Precision	Recall	F1-Score
Bean	0.91	0.99	0.95
Bitter Gourd	0.99	0.93	0.96
Bottle Gourd	0.99	0.87	0.93
Brinjal	0.88	0.94	0.91
Broccoli	0.98	0.97	0.98
Cabbage	0.96	0.99	0.98

Capsicum	0.94	0.97	0.96
Carrot	0.98	0.99	0.98
Cauliflower	0.99	0.97	0.98
Cucumber	0.87	0.88	0.88
Papaya	0.92	0.89	0.91
Potato	0.95	0.97	0.96
Pumpkin	0.97	0.97	0.97
Radish	0.99	0.99	0.99
Tomato	0.98	0.94	0.96

Confusion Matrix for Benchmark Model:



Confusion Matrix for Primary Model:



While both models clearly show some good results we see 5 common misclassifications in the benchmark model and 2 common misclassifications in the primary model. For the benchmark model, we see beans commonly misclassified as bitter gourds, birnjal and cucumbers commonly misclassified as beans, and cabbage and tomato commonly misclassified as pumpkin. For the primary model we see papaya commonly misclassified as brinjal and bottle gourd commonly misclassified as cucumber. The misclassifications in the CNN model show a larger degree of misunderstanding and likely indicate that the model may not necessarily be focusing on the right details within the images. However, the misclassifications within the primary model match with what our earlier intuition had suggested would be the difficult categories to classify. Further, having only these two misclassifications, and the unidirectional nature of them, provides opportunities for further analysis and modeling to solve in a practical manner. For example, within the app it may be possible to train this model to predict “brinjal or papaya”, which would then feed to a separate model to predict the difference between

them. The benchmark model, since it has far more misclassification problems, particularly in categories that we wouldn't expect to see problems in, would require additional work such as further model tuning, additional data, or more permutations of the provided input data. Given that we already have a better model, it's likely simpler to focus on the transformer model than to continue progress on the CNN.

III. Summary and Conclusions

The results we obtained support the idea that while CNN can be used for this type of classification and perform quite well, the Transformer ResNet34 model (as discussed and developed by my groupmate) is a more powerful model for these classifications. The precision and recall of this model is stronger and the limited scope of the misclassifications lends itself to the idea of further model development opportunities. A separate opportunity is to train ripe and unripe vegetables separately. As in the case with brinjal, a ripe brinjal has no resemblance to a papaya, whereas the unripe brinjal does have that similarity. Training the model with added categories for ripe vs unripe may aid in the classification of the ripe fruits, which are more likely to be found in the store or in a refrigerator. Additional future work on this could include a secondary model to predict the ripeness of the vegetable or multilabel classification to identify multiple vegetables within a refrigerator, both of which would require additional data or additional labeling of the existing data.

Additional improvements we would like to make if we had the time would be to bring in true out of sample data for testing. Although the testing data is distinct from the training and validation data, because they came from the same original project, it's likely that there are still some biases inherent in the broader dataset. Bringing in true out of sample testing, whether through photos we take or other available image sets online would allow us to understand if the model would perform well in a potential real world setting and not just on similarly constructed images.

As my groupmate will discuss in her paper, the limitation we faced for the use of Captum restricted us to analyzing a single image from the main training model. Given the possibility that the benchmark model is not focusing on the key pixels within the images for classification (as seen through the broad misclassification where we wouldn't expect it), it would be interesting to also analyze images from that model in a similar fashion for the comparison.

Through this project and course I've learned quite a lot. Firstly I've learned what neural networks are and the differences between them as well as the applicability of MLP vs CNNs in classification. Specifically that using a CNN is critical when we have pixel dependencies to maintain that ordering, while MLP will classify using each pixel as its

own distinct entity. I learned how to use sequential vs functional programming to put these models into practice, how to use terminal to run code and use argparsing. I'm comfortable now developing MLP and CNN models, understanding the package documentation for TensorFlow and PyTorch. I've also learned the importance of developing separate files for training and testing, which can allow for more iterations and easier to read and clean code. Working on this project with my groupmate, I have also learned what a transformer is, the ways in which it is more complicated than a CNN model, and benefits in using it. I feel confident that if I come across another neural network based model that I would be able to interpret the differences between it and the models we've learned, understand where to go for more documentation in order to implement the model, and that at the end of the day, all of these models are transformations of matrices and vectors that provide mappings from the input data to the output range.

IV. Coding

My Code (not internet): 207 Lines
Internet Code (not modified): 24 Lines
Internet Code (modified): 23 Lines
Percentage of Code from Internet: ~10.4%

V. References

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9#:~:text=Average%20pooling%20method%20smooths%20out,lighter%20pixels%20of%20the%20image.
https://www.analyticsvidhya.com/blog/2021/06/confusion-matrix-for-multi-class-classification/