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

Cooper Atkins, Ange Olson

I. Introduction

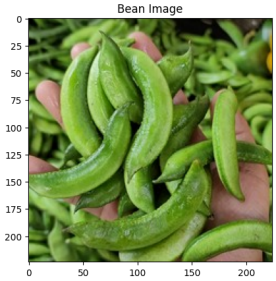
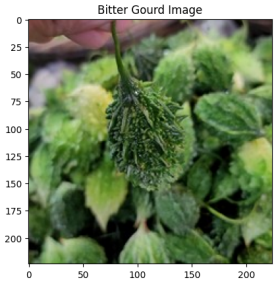
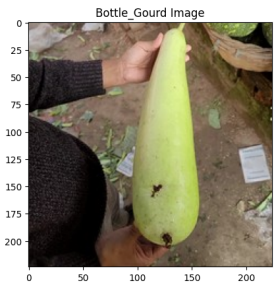
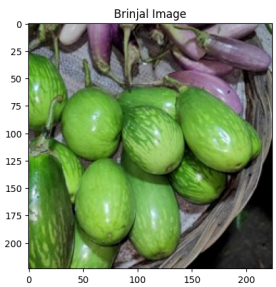
There are many scenarios in life that require eyesight that we often overlook or take for granted. One such scenario is being able to identify different objects. For those who are able to see, attempting to identify objects is relatively easy. We'll first look at the object and see if we can automatically identify it. If not, we may look around for context clues such as an identifying label or other reference. If that also fails, we can ultimately Google the item using visual identifiers. However, each and every one of those steps requires being able to visually assess the object.

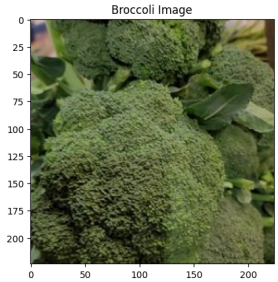
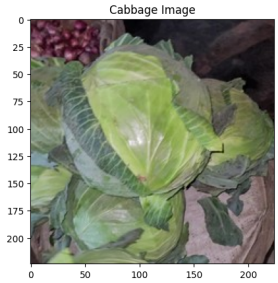
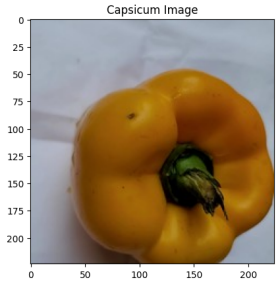
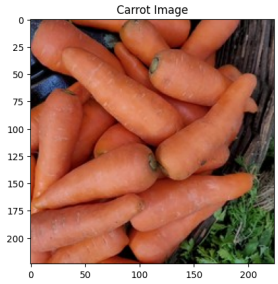
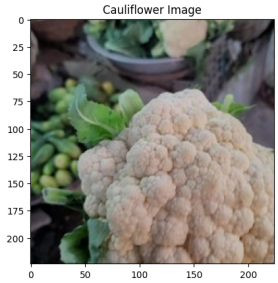
One place where this may have an impact is when identifying foods at a grocery store, particularly when identifying vegetables (or fruits) as they often don't have barcodes attached that would allow for scanning apps to solve this problem. Similarly, when selecting a vegetable in a refrigerator, there is also nothing to help identify the object besides physical feel. As a result, we are choosing to build a model that will use images to identify vegetables with theoretical applications to an App that could allow users to hold a vegetable, have the app take the photograph and subsequently identify the vegetable for the user. One additional application for this type of model would be within grocery stores at checkout, where the register could have the model built into the scanner so that instead of the cashier needing to manually punch in a produce code while weighing the product, the machine would automatically identify the produce on the scale.

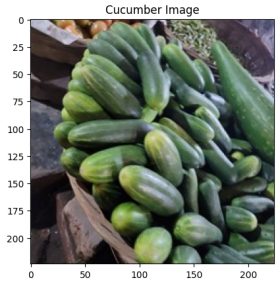
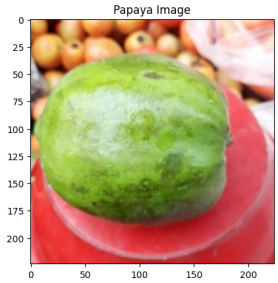
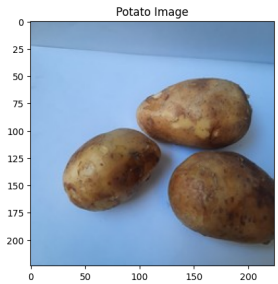
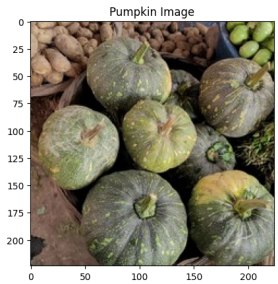
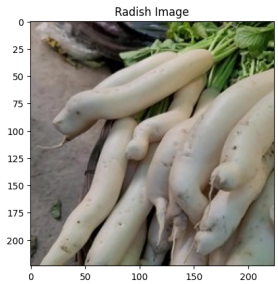
II. Description of Dataset

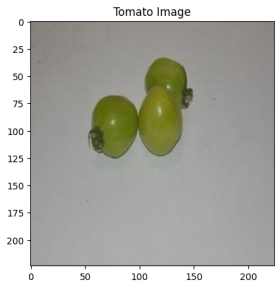
This dataset consists of 21,000 data points split into three development sets: train, validation, and test. The training dataset contains 15,000 images, while the validation and test sets each contain 3,000 images. There are 15 different vegetables or classes within the dataset as defined in Figure A with fully balanced class representation in all 3 datasets.

Figure A: Dataset Description

Type	Count	Example
Bean	Train: 1,000 Validation: 200 Test: 200	
Bitter Gourd	Train: 1,000 Validation: 200 Test: 200	
Bottle Gourd	Train: 1,000 Validation: 200 Test: 200	
Brinjal	Train: 1,000 Validation: 200 Test: 200	

Broccoli	Train: 1,000 Validation: 200 Test: 200	
Cabbage	Train: 1,000 Validation: 200 Test: 200	
Capsicum	Train: 1,000 Validation: 200 Test: 200	
Carrot	Train: 1,000 Validation: 200 Test: 200	
Cauliflower	Train: 1,000 Validation: 200 Test: 200	

Cucumber	Train: 1,000 Validation: 200 Test: 200	
Papaya	Train: 1,000 Validation: 200 Test: 200	
Potato	Train: 1,000 Validation: 200 Test: 200	
Pumpkin	Train: 1,000 Validation: 200 Test: 200	
Radish	Train: 1,000 Validation: 200 Test: 200	

Tomato	Train: 1,000 Validation: 200 Test: 200	

While looking through the data, it becomes clear that several classes are likely to provide the biggest challenge for classification. Specifically, it appears Brinjal and Papaya, as well as Cucumber and Bottle Gourd are relatively similar to the naked eye. The dataset contains images that are both ripe and underripe (as in the case of brinjal as we see above), which is where some of this difficulty may appear, but will also mean that a successful model is capable of classifying multiple states of each vegetable. We'll be looking to see at the end how well the models are able to distinguish these pairs even within the broader classification structure.

III. Description of Model

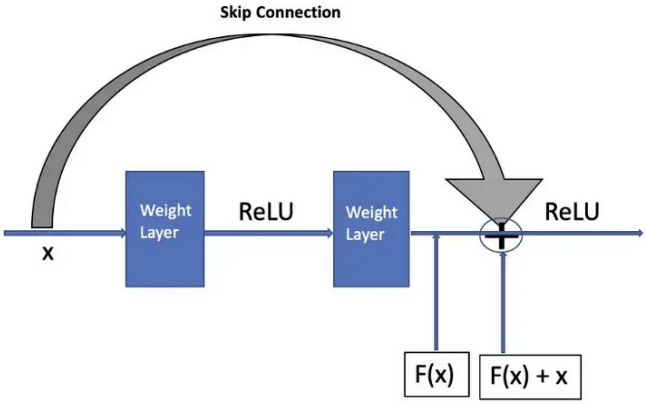
For our benchmark model we developed a convolution neural network. The choice to use a convolution network rather than a multi-layer perception is quite apparent as there are obvious pixel dependencies within the images selected for training (and indeed in any image that would be fed through the completed 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.

For our primary model, we seek to take advantage of transfer learning, using a transformer as our base model and fine-tuning based off of the pre-trained model weights. We selected a convolution-style network called ResNet34, a variation of one of many ResNet models. Deep networks can face the problem of a vanishing, or increasingly small, gradient as small values are multiplied by small values over many layers. A small gradient results in small weight updates–in other words, the model does not learn.

One of the benefits of ResNet, named because it is a residual network, is that inputs skip certain layers. This is introduced in residual blocks with skip connections, shown in Figure B below.

Figure B: ResNet Residual Block



Above, the gray arrow represents a skipped connection; the input jumps over the mapping in this layer and goes directly to the transfer function.

The model contains four of these blocks, with convolution layers followed by pooling layers. The authors of the paper where this model is first introduced provide evidence that this architecture improves optimization, and that the depth of the network leads to accuracy gains.

In fine-tuning, we freeze all layers except for the last block and train with a low learning rate (less than 1e-3) so as to not unlearn information given by the pre-trained weights. We have opted to include a single fully-connected layer on top of this model to change the output to the size of the number of classes we wish to classify into (15). We use a softmax layer following this output in order to interpret our model output as probabilities of belonging to any one class; this is used in our model interpretation method, Integrated Gradients, which depends on an estimate of probability or confidence for evaluation.

IV. Methodology

Data Preprocessing:

All data is normalized using mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225] for use in the ResNet model. All image sizes are 224x224 to accommodate this model as well; we use these same specifications for the CNN benchmark model.

Our training sample is transformed by randomly applied crops and horizontal flips to add diversity to our training set and ideally reduce overfitting. We also one-hot encode our labels, given that we are dealing with a multi-class problem in which we would like to assign some kind of probability of an image belonging to any one class.

Fine-Tuning:

In fine-tuning the parameters, all kernels are set to be 3x3 and a relatively low learning rate is chosen (less than 1e-3) so as to not unlearn information given by the pre-trained weights. Additionally, all pooling is done at the 2x2 level and the stride is set to 1 to maintain maximum image size between layers while still allowing for reductions in dimensionality. Lastly, padding is left at 1 to still allow for dimensionality reduction throughout the model. Dropout was also tested at the 0.20 and 0.25 level, but did not show meaningful impacts to the results, so was ultimately discarded for the final model.

Performance:

Our model loss function is BCELoss, or Binary Cross-Entropy loss, used for classification problems. As specified, each of the individual targets falls between zero and one. We use accuracy during training in conjunction with loss to track model performance over each epoch. For post-hoc analysis on our test sample, we also look at loss and accuracy but also look to measure recall, precision, and the F1 score for each class to see where our model specifically is performing well or not.

Interpretation:

One way we seek to judge the performance of our model is through interpretability. For image classification, we are looking for certain pixels in images to provide more weight than others. For example, if we were trying to identify an image as containing a fireboat,

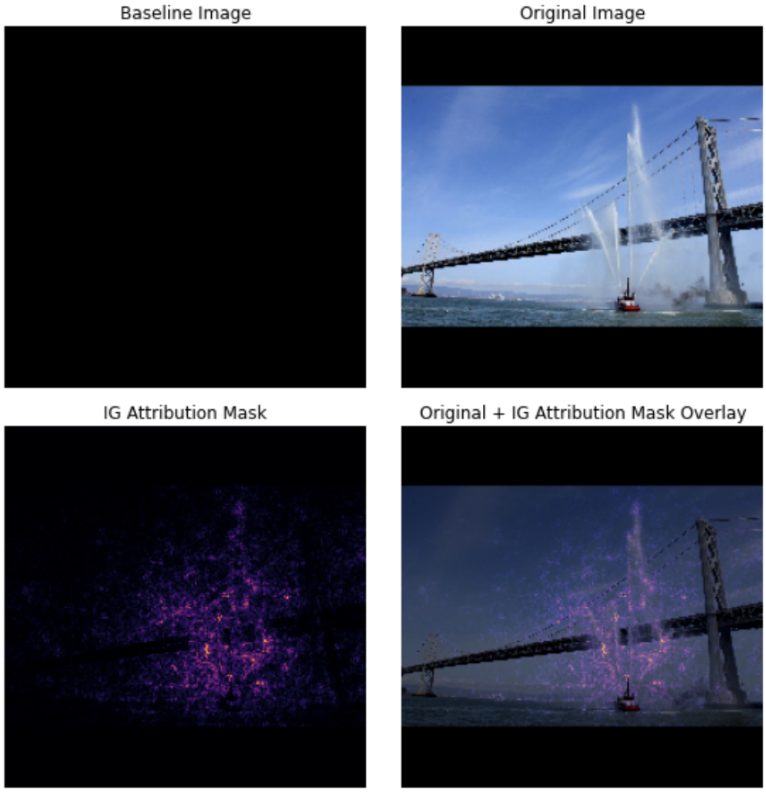
we would want the pixels defining the shape of the boat to provide more insight to the final model decision than pixels corresponding to a nearby tree or bridge.

Integrated Gradients are one way of interpreting the model. Where x is an input, and x' is a baseline, the formula for calculating integrated gradients is as follows:

$$IntegratedGrads_i(x) ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i}$$

Integrated Gradients does exactly what we are looking for. An example from TensorFlow below shows that for the fireboat image, the heatmap of 'activated' pixels corresponds to the shape of the boat and the water spouts rather than the water or the black padding on the top and bottom of the image. In this case, the baseline used is a black image, where the pixels are expected to provide no information.

Figure B: Example of Integrated Gradients



The Integrated Gradient (IG) Attribution Mask (bottom left) is applied over the original image (top right) to create a contextual heatmap of activated pixels (bottom right). This is what shows us that the model used to classify this image is looking at the "correct: pixels–the pixels surrounding the water and boat are most activated.

In the context of this project specifically, an input is the normalized tensor of pixel values, with the baseline as in the above example being a black image of the same size. The idea is that to understand the effect of individual pixels on an output, there must be a comparison made between pixels that should have absolutely no effect, i.e. the black pixels. In software and practical applications, the integrated gradient is approximated, using Reimann summation or the Gauss-Legendre quadrature rule.

We use the Captum package *IntegratedGradient* class to compute attribution scores for each pixel, which are then displayed on a heatmap. The algorithm assigns an attribution score based on the gradient with respect to each input, based on the approximation of the gradient calculation of the outputs with respect to the inputs from the baselines. For classification problems, a target is given corresponding to the class. High attribution score 'activate' the pixel, while low activation scores do not, indicating respective effect.

Another way to visualize which pixels are through approximating the SHAP (SHapley Additive exPlanations) values through the gradients. Captum has a class for this, called *GradientSha*p, which we implement in this project. Assuming inputs are independent (i.e. that images are not correlated with each other, which for this project is safe to assume), a linear approximation can be made between a background image and the image of interest. The expected gradients of this linear model function approximate SHAP values. Again, a reference image is used as comparison.

We use both Integrated Gradients and expected gradient SHAP approximation to analyze our model and see how it is making decisions based off of the images we feed it.

V. Results

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

Main_training results:
Epoch 30
train_loss : 0.060631087065761924 val_loss : 0.04103961908865882
train_accuracy : 0.9109241452991453 val_accuracy : 0.9551630434782609

Benchmark_testing results:
Test Accuracy: 0.9156534954407295 Test Loss: 0.06209164560633771

Main_testing results:
Test Accuracy: 0.9523176291793313 Test Loss: 0.04378727395483788

Clearly we see here that the testing results for the primary model, the transformer are superior to those in the CNN. The testing results show ~2% less loss in the ResNet model and an additional ~4% improved accuracy in the ResNet model as well. This validates the idea that the added complexity of .

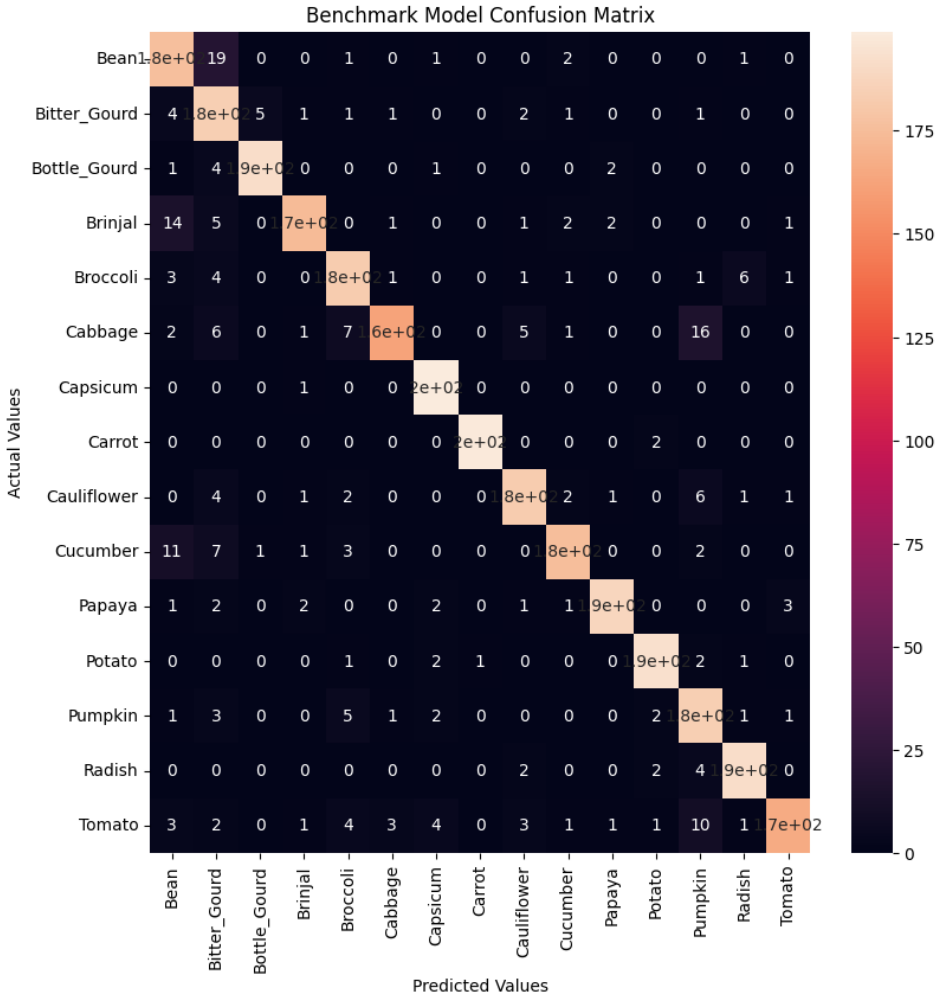
Test Data: F1 Macro, Precision, Recall by class for Benchmark Model

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

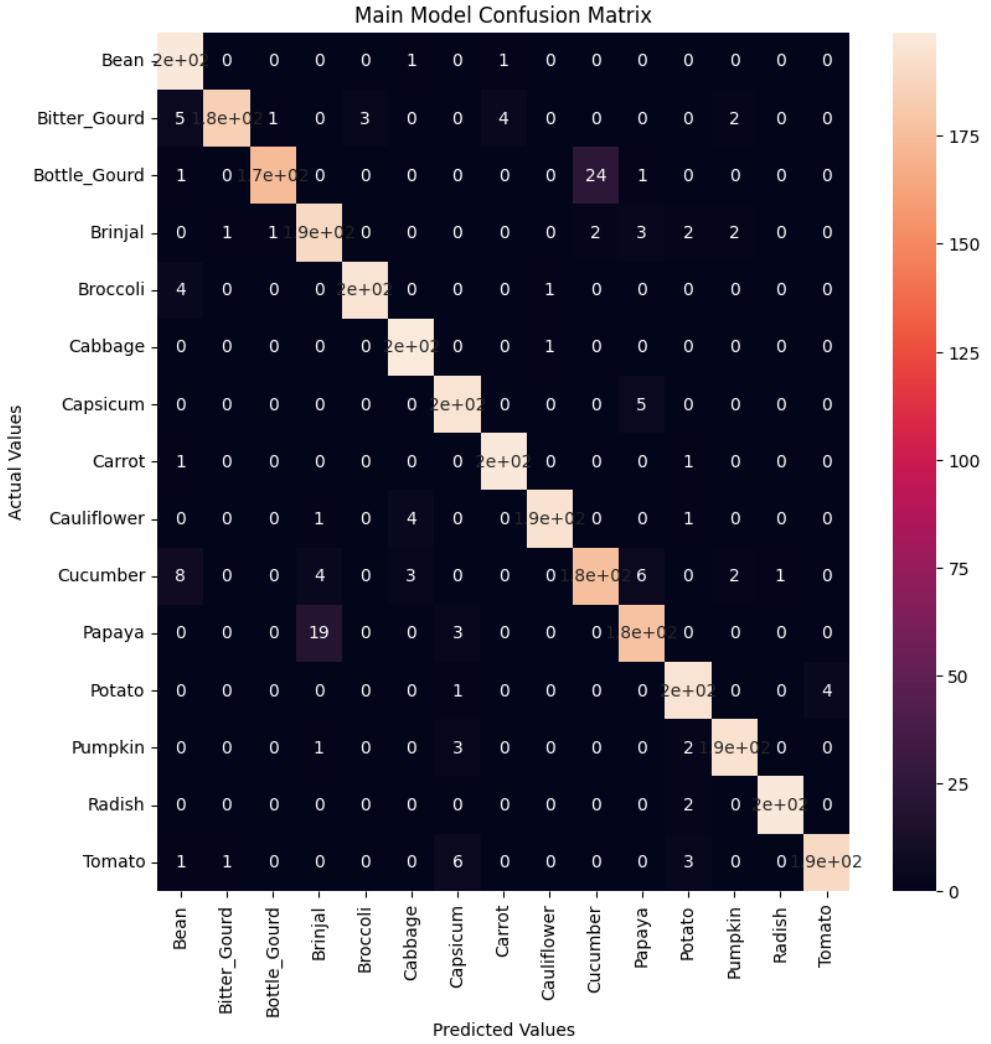
Test Data: F1 Macro, Precision, Recall by class for Primary Model

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:



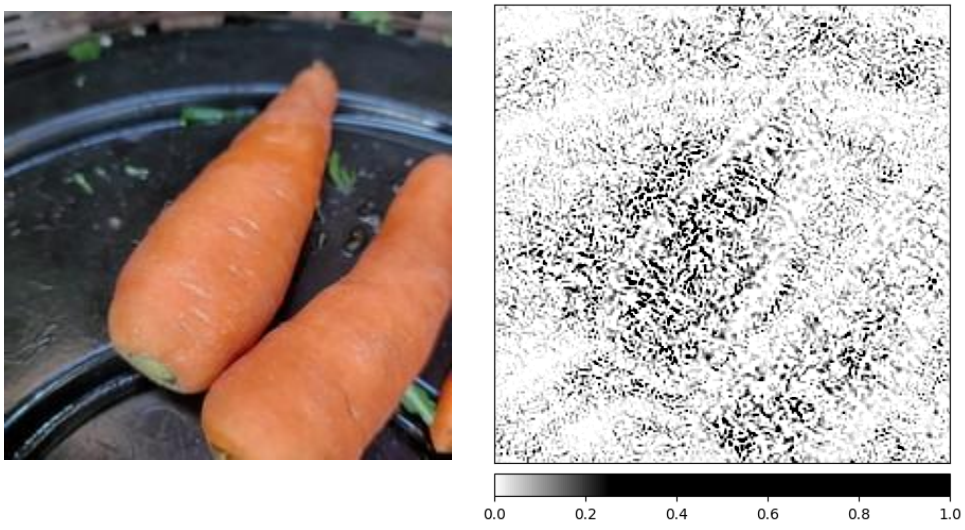
While both models clearly show some good results we see 5 common misclassifications in the benchmark model and 2 common misclassifications in the primary model. For the benchmark model, we see beans commonly misclassified as bitter gourds, birnjal and cucumbers commonly misclassified as beans, and cabbage 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.

Interpretation Results

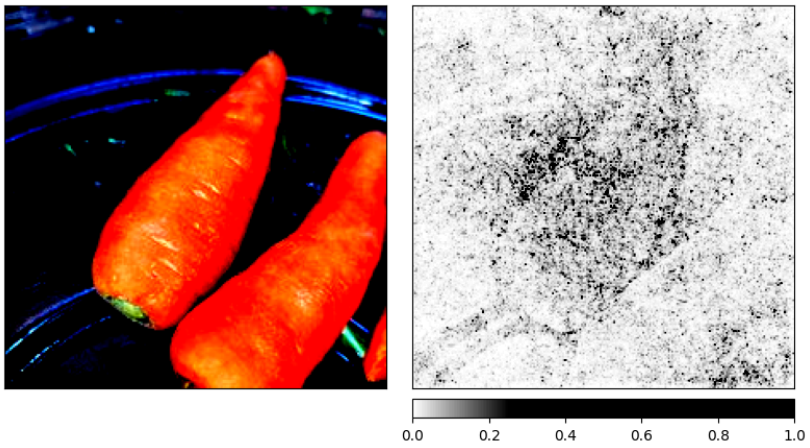
We try three total variations of gradient approximation: one is simple Integrated Gradients, one is Integrated Gradients with a smoothing effect from a noise tunnel, and one approximating SHAP values. We use one image from our test to analyze our model, a correctly labeled carrot (probability over 80%, so strongly recognized and confidently predicted). The original image with no transformations is presented first, then the Integrated Gradients heat map of the image. Next, the transformed image is plotted against the heat maps for Integrated Gradients with smoothing. This smoothing occurs via a noise tunnel, adding Gaussian noise and using *smoothgrad_sq* as the mean of the squared attributions across the number of samples generated by the noise tunnel. Lastly, we plot gradient SHAP estimates.

Figure C: Original Image of a Carrot (Test Image 1039) and Heat Map



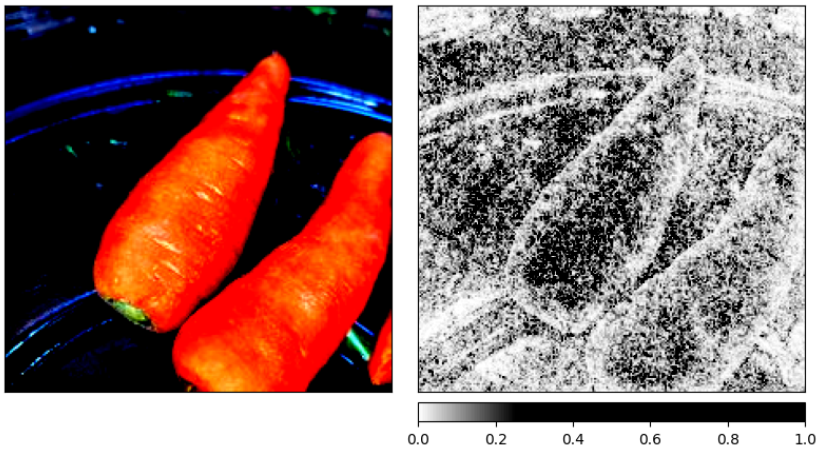
Left: the original image of the carrot, with no transformations applied.
Right: the corresponding Integrated Gradient heat map. A darker value corresponds to greater pixel activation. The pixels defining the shape of the carrots are clearly defined in this image.

Figure D: Integrated Gradients with Smoothing (Four Samples)



Left: transformed image of the carrot (normalized to ResNet standards–mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]) Right: the corresponding Integrated Gradient heat map with a noise tunnel smoothing with four samples. Again, a darker value corresponds to greater pixel activation. The pixels defining the shape of the carrots are still defined in this image, but the lower right carrot is less visible.

Figure E: SHAP Approximation



Left: transformed image of the carrot (normalized to ResNet standards–mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]) Right: the corresponding SHAP approximate by expected gradient. Again, a darker value corresponds to greater pixel activation. The pixels defining the shape of the carrots are very defined in this image, however the surrounding bowl is also clearly identified.

All images show at least one of the carrots outlined clearly–this tells us that the model identified the shape of the carrot clearly and used it to make the correct prediction. However, we can also see that pixels showing the outline of the bowl are activated, notably when looking at the last image with the SHAP approximation. This could signify that the model is overfitting in some regards–if put into production with enough images

of carrots not in bowls, the model may perform worse. To look into this issue further, we could run this same analysis on a sample of the carrot images not in a bowl and see how well the model did, or on other images with different vegetables in a similar bowl to see if a carrot was a probable prediction.

VI. Summary and Conclusion

Looking cumulatively at the results, we see that the transformer ResNet model clearly outperformed the baseline CNN model. The CNN model has misclassifications in several categories, including those that would be surprising to misclassifications in (such as pumpkin/cabbage and cucumber/bean), while the ResNet only has misclassifications in two categories and in the same direction each time. Similarly, when we conduct interpretability analysis, we can see that the ResNet model is clearly picking up on the right pixels for the identification of the carrot.

There are two main areas in which we would want to focus additional analysis and improvements. The first is that we were constrained by GPU and were therefore unable to analyze additional images for interpretability. In particular, we'd like to see the interpretability of the CNN and across the other 14 classes. A key area of focus would be on the two classes where we saw the most misclassification to understand if the misclassification is occurring in a consistent manner across the pixels. This would help us understand ways to tune the model or next steps including separate models for the classification of these categories as an overlay after the original classification. The second improvement would be the introduction of true out of sample data for testing. Although the testing data is distinct from the training and validation data used, all three datasets originated from the same sourcing, which may have introduced biases into the broader dataset. Understanding how the model would perform in the real world, such as in a grocery store or kitchen, would be imperative to understanding the true performance of the model.

VII. References

https://captum.ai/tutorials/Resnet_TorchVision_Interpret
https://towardsdatascience.com/how-to-load-a-custom-image-dataset-on-pytorch-bf10b2c529e0
https://www.tensorflow.org/tutorials/interpretability/integrated_gradients
https://papers.nips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf
https://github.com/slundberg/shap#deep-learning-example-with-gradientexplainer-tensorflowkeraspytorch-models
https://medium.com/analytics-vidhya/understanding-resnet-architecture-869915cc2a98
https://arxiv.org/pdf/1512.03385.pdf
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/
https://www.askpython.com/python/examples/display-images-using-python

VIII. Appendix

See Appendix Folder for files