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

Ange Olson Individual Report

Introduction:

In this project, my group chose to focus on interpreting an image classification model using applications of Integrated Gradient. We chose a dataset of vegetable images, with 15 equally sampled classes. As a benchmark, we implement a Convolutional Neural Network (CNN) and for our main model we wanted to take advantage of transfer learning. We chose ResNet34 as our transformer base, repalcing the last fully-connected layer for one outputting the shape of our class dimension. In fine-tuning, we freeze all layers except for the last block and train with a low learning rate. 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.

In interpretation, given our test example of a correctly-identified carrot image, we find that our model does appear to be identifying and placing weight on pixels that should lead to a classification of a carrot (i.e. the outline of a carrot) but that given the other pixels it picks up on, namely those of the surrounding bowl, the model may be subject to some overfitting. Images of vegetables outside of the sample of images provided in this dataset may not be classified as well.

My primary work on the project was as follows:

- Developing the pre-processing pipeline
- Constructing the model classes and training/testing loops
- Using Captum *IntegratedGradient* and *GradientShap* to look at attribution scores for a prediction to get an understanding of whether or not our model was assigning classes to an image in a way that visually made sense.

In addition to the code above, the sections of the paper and presentation I primarily worked on were related to our preprocessing, model architecture (ResNet), and interpretation. My group member focused on modifying the model hyperparameters and the base CNN code I provided and EDA/results compiling/post-hoc work. My group member worked on the parts of the paper and presentation that discussed these tasks

as well as the summary and conclusion sections. My group member coded the part of our training loop that saves the model and plots the accuracy and loss for our training and validation sets.

Individual Work:

Pre-Processing:

I developed the code to download the data and construct the datasets/dataloaders for the model. ResNet documentation states that images need to first be read in with pixel values between values of zero and one and then normalized based on specific mean and standard deviation values for the channels. All samples of data (training, validation, and test) are pre-processed in this way. Transforms are applied at random on the training data to increase the diversity of our sample and avoid overfitting to specific image types. Based on the look of the images after applying normalization, I opted to not randomly transform the brightness or contrast of the images, as the colors and saturation were already augmented.

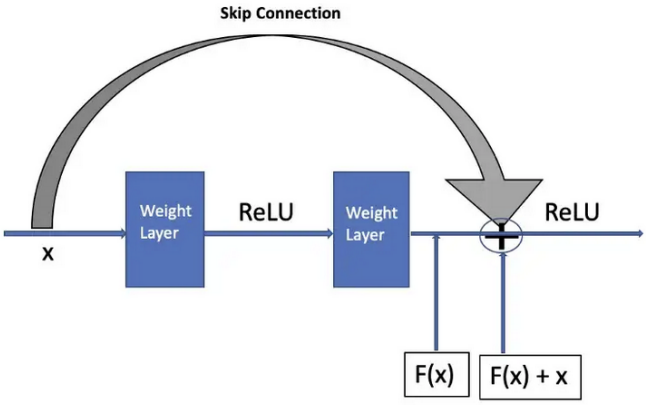
Model Classes and Training/Testing Loops:

I developed the base code for our CNN model (which my partner added layers to for a more complex and better-performing model) and developed the model definition and training and testing loops. To apply our interpretation methods, the softmax function is applied to the last output layer and those values saved as is for the testing data to provide confidence estimates on our final classification results.

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.

Interpretation:

Lastly, I followed Captum tutorials on image classification model interpretation so we could speak to the usefulness of our model more broadly. I 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 I 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.

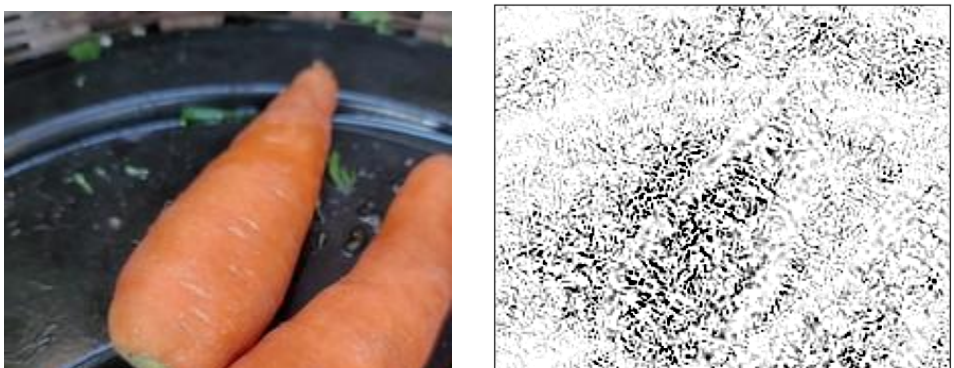
Results:

Our main model performs well. After 30 training epochs, we achieve the following results:

- Train Loss : 0.060631087065761924
- Validation Loss : 0.04103961908865882
- Test Loss: 0.04378727395483788
- Train Accuracy : 0.9109241452991453
- Validation Accuracy : 0.9551630434782609
- Test Accuracy: 0.9523176291793313

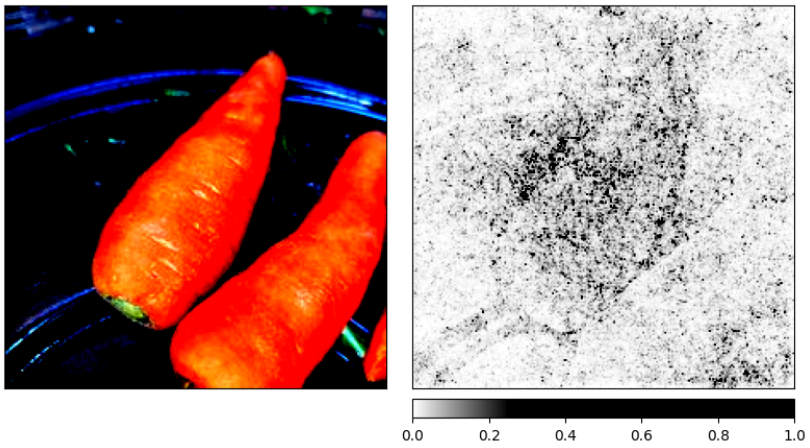
We try three total variations of gradient approximation: one is simple Integrated Gradients, one is Integrated Gradients with a smoothing effect from a noise tunnel, and one approximating SHAP values. We use one image from our test to analyze our model, a correctly labeled carrot (probability over 80%, 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 A: Original Image of a Carrot (Test Image 1039) and Heat Map



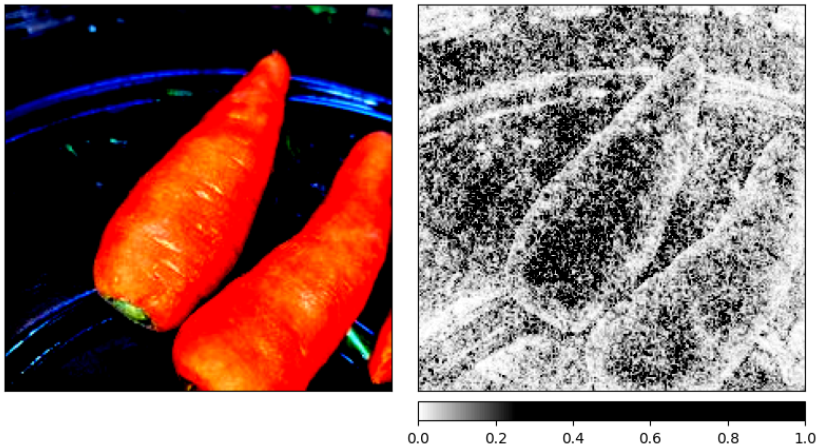
Left: the original image of the carrot, with no transformations applied.
Right: the corresponding Integrated Gradient heat map. A darker value corresponds to greater pixel activation. The pixels defining the shape of the carrots are clearly defined in this image.

Figure B: 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 C: 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.

Summary and Conclusions:

Overall, we found evidence that our model performs well, though with limited gains over our benchmark. In addition to looking at the metrics associated with model performance, however, interpretation also shows us why our model is performing well. By looking at the gradient with respect to individual inputs, in this case pixels, we can see through our example that our model does identify the shape of vegetables, suggesting it is learning in a way that makes it broadly applicable. Rather than learning minute details about our particular sample images, by recognizing the outline of a carrot, for example, our model could likely recognize carrots in other contexts. We do see some evidence, however, of the model assigning weight to pixels that may not be widely useful–for example, the outline of the bowl.

I learned a great deal in this project, particularly around integrated gradients and how to use the mathematics behind the model optimizer to find out feature importance in the context of natural language processing. Beyond learning the theory, my programming

skills and ability to understand the functions in the packages I am using has increased tremendously–I felt well-prepared by this class to read through Captum source files to find out what each function was doing, what I needed to input (given that I was not exactly following the guide) and why.

Given more time, I think that improvements to our project could be two-fold: one, with more computing resources, and two, with out-of-sample test data. One of the resource issues we ran into while using Captum was limited GPU space, which is why we focused on analysis of one test image. Ideally, we would have looked at several samples from different classifications, or even one that was misclassified. It would have been interesting to see if, like for the carrot image we chose, pixels of the surrounding container being associated with higher attribution scores was common. The other thing that would have been useful to try is bringing in out of sample images. Our test set came from the same project as the training and validation, and while the model had not yet seen this data before, there are likely underlying similarities. Images from an entirely different source would be an excellent test set to see how well our model has actually learned the shapes of the different vegetables when image resolution is different, or backgrounds are very different.

There are many methods out there that we did not have the time to learn about or implement that would have been interesting to try, like class activation mapping, which also uses the gradient calculation.

Code Calculation:
Novel Code: 311 lines (NLP_CodeNovel.py) Note: includes in-class code
Edited: 36 lines (ML_CodeEdited.py)
Code from Internet: 129 (ML_InternetCode.py)

129-36/(129+311) * 100 = 93/440 * 100 = ~21.14

Resources:
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