Training and Testing Data:

To create an effective model for filling in the masks, we wanted to use training data that contained green and gray colors, similar to those found in the masked images. We decided not to use data from the masked images themselves to avoid overfitting. After reviewing a sizable portion of our camera roll, we chose two images to train and test our models on: a close-up of leaves and another of stone gargoyles in the grass. We then cropped the photos to 900 x 900, the same size as the masked images, so any data processing or image displaying functions we used for the training and testing data would also work for the masked images. The images were loaded into a Jupyter notebook using PIL and Numpy and separated into their red, green, and blue channels. Each color channel for each image was divided into overlapping 50 x 50 blocks, and the bottom right pixel of each block was removed and saved as the label. The blocks were then flattened, creating a dataset of size 724,201, each data point having 2499 features and one label. For both images, 80% of the data was used for training, and 20% was used for testing. Finally, we shuffled the training data so that the model received a random block from the image each time we trained it instead of receiving the blocks in the order they appear in the image. We did this to avoid biasing the model toward features from a specific region in the image.

Structure of Data:

The data given as input to our model is a 2499-dimensional vector of scalars, each corresponding to a 1-dimensional label vector with one scalar. The input vector is derived from a 50x50 block of pixels from the image, with the bottom right corner removed. The label corresponds to the removed bottom right corner. This structure has several advantages. First, it provides a large quantity of training and testing data points from a single image: the number of 50 by 50 pixel blocks in a 900 by 900 image is 724,201. Second, it allows us to use the same set of features to predict every pixel in the masked image, as we only use pixels above and to the left of a given pixel to predict its value, and we predict pixels starting from the top right downwards to the left. Finally, we can use pixels whose values we have predicted to predict the value of other masked pixels. The downside of this approach is that our model does not get any information about what is down and to the left of it, which may be useful, even critical, for predicting the pixel value. However, for the specific masked images in this problem, it likely will not be an issue because there aren't many pixels that are very different from those up and to the right of them.

Data Processing:

We converted our training and testing images into 900 by 900 by 3 arrays to capture the RGB values of each pixel. This approach is more descriptive than assigning a category like "Green" to each pixel and allows us to use an intuitive loss function: the

difference between the predicted RGB value and the true RGB value of a given pixel. Using image and data processing libraries PIL and Numpy, converting a jpg image to this format is straightforward. Additionally, because there are three distinct color channel values to predict for each pixel, we can use an ensemble of models to predict each pixel value, which we discuss in more detail in the “Models” section.

Model, Models, Modeling:

We employ three models for both the tree and non-tree versions of this problem. Our strategy was motivated by the discovery that the loss function for a model could be greatly simplified if the model predicted each color channel individually. The simplified loss function is the square of the difference between the predicted and real value of one of a pixel's color channels. For the non-tree model, we train three neural networks to predict the three different color channels of an image: red, green, and blue. The predictions of all three networks are then combined to form one prediction for all three color channels of a single pixel. We follow the same approach for the tree-based model, fitting a decision tree to each of the color channels and combining their outputs to produce a prediction.

Each neural network model consists of an input layer with 2499 neurons, one hidden layer with 16 neurons, and an output layer with one neuron. After initial testing, this arrangement was found to be the smallest possible one that produced reliable results.

Our tree models are based on termination-by-depth, with a maximum depth of 16. This was based on testing from previous homeworks which indicated that 14 was the optimal depth for other problems, and additional testing here which showed that 16 was a suitable depth for this problem.

This technique has several benefits in comparison to using a single model to predict all three color channel values for a given pixel. Because we are using an ensemble, each individual model can be smaller and thus will train and predict faster. Additionally, we can train the models concurrently, significantly reducing total training time. Lastly, we can use a simple and intuitive loss function, providing confidence that the model will generate high quality outputs.

However, using this approach may lead to a few trade-offs. By separating the color channels, we risk losing any cross-channel information that could be beneficial for predicting masked values. Furthermore, as three models are used to generate three outputs, the results must be combined through extra post-processing before we get our final result. Finally, because each individual model is relatively small, they may not be

able to effectively capture the relationship between pixels which may lead to poor predictions. On the other hand, given the relative color simplicity of the images we are filling in, neither of these downsides is of large concern.

Bootstrapping:

As mentioned in the Data Processing section, we use bootstrapping to fill in the masked image. For the first masked pixel, we only use data from pixels that where never masked. Subsequent to this, we gradually incorporate a larger portion of previously predicted pixels into the input data of the model. As we move down and to the left of the image, the proportion of these bootstrapped pixels increases. There are two main risks with this approach: a compounding in the probability that our model makes an error and an increase in magnitude of an error when one is made. To minimize these risks, the initial predictions of the model are primarily based on non-bootstrapped pixel values, so their error is minimized. Thus, the increased error chance and magnitude for later predictions will remain small, despite growing larger.

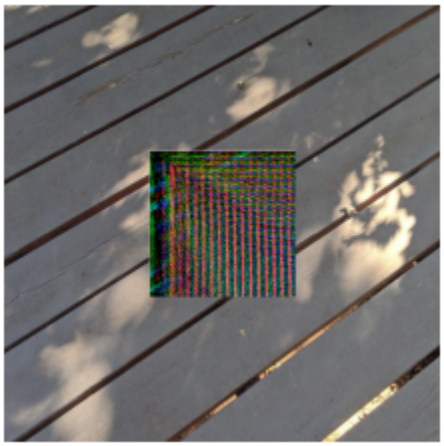
Measuring Loss:

For each model, the loss function we use is the square of the difference between the predicted value for a given color channel and the true value for that color channel. We expect this loss to work for creating models that fill in the mask in a realistic way, because the smaller the difference between the predicted color channel value and the true color channel value the closer the pixel that the model generates is to the real pixel. Thus we have strong reason to believe that a lost function with a small value corresponds to an image that looks realistic.

Training:

The first step to training our model as we mentioned in the Measuring Loss section, was to choose a loss function that we had confidence would lead our model to creating realistic looking images. To address the concerns of overfitting, we gathered enough training data such that we didn't have to repeat any of the training data when training or model. Thus, we did not become concerned that our model would overfit to the training data and not generalize well when we wanted it to fill in the masked images. The biggest problem we ran into when trying to create good models was that it took a very long time to train the models. We had to learn to balance the size of the model with the amount of data we allowed them to train on. When training our neural networks, we utilize the entire dataset and make a single pass over it. However, training our tree models using the same amount of data would take a significantly longer amount of time. Therefore, we randomly select 12,000 data points from our dataset to train the trees on. This reduces the training time and provides similar performance to training on the entire dataset.

Results:
Here are the results for the neural network method:

Here are the results for the tree method:


Image Analysis:

The neural network based model's predictions appear to be biased towards predicting red, which may be due to the lack of red as the dominant color in either of the training images. This could have caused the model to minimize its loss by predicting a value close to 127, the mean color value. Additionally, the predictions show distinct lines of color which make them look unnatural. While the tree image may be acceptable to some extent, the wood image does not appear realistic even from a distance.

The tree model performed significantly better than the neural network model. Like the neural network model, the tree model's predictions are similar to one another. The Leaves predictions have a green hue while the Wood predictions have a blue and yellow hue, which reflects the colors of the original images. Additionally, the filled in portions have a fractal characteristic to them: they look like a series of cascading

corners. The predictions, while clearly not part of the original images, fit within the images in a way that is visually pleasing, with a texture that is distinct but not overtly noticeable.

Assessing the Models:

When it comes to assessing the models, there is the scalable, mathematical way to do it, and there is the right way to do it.

The mathematical way to assess the models is to use our loss function on our testing data to get the average loss of each model on the testing Data. Then, compare the average losses and select the model with the lowest average loss.

The success of a model depends on its ability to generate realistic outputs. As such, the selection of the best model involves a subjective judgment of how realistic the outputs are. Even if a model has high loss on multiple loss functions, it could still be the best model if its outputs appear realistic. When assessing the quality of models, the outputs should be carefully evaluated, asking questions such as: does it look believable? Does it look good to me and to other people? While some loss functions are helpful in training the model to produce realistic outputs, it is ultimately the realism of the generated content that matters most.

We can assess whether our models are achieving their desired outcomes by examining their outputs and gaining feedback. If the majority of people feel that one model produces more realistic outputs than another, then that model is considered to be better.

Final Thoughts and Verdict:

Tree-based models demonstrated superior performance compared to neural networks in this context. Compared to neural networks, they required less coding effort, trained faster with reduced data requirements, produced more realistic outputs, and generated outputs in a shorter amount of time once training was complete. These factors clearly demonstrate that tree-based models outperformed neural networks in this case.