HOMEWORK 4

Class: CS 5785

Authors: Anna Guidi, Samantha Yip

November 27th, 2017

A. Purpose of the Study

Problem 2 – Random forests for image approximation:

Preprocess images through gathering a sample set of pixels and normalization; Train the sample set with Random Forest algorithm and reproduce the image based on the trained dataset; Compare the resulting image with the one replicated using KNN algorithm

B. Procedure

Problem 2 – Random forests for image approximation:

First, we store the data for the Mona Lisa image into an array called "mona." The array contains the RGB intensity of each pixel on the image, which is limited between 0 to 255.

For preprocessing, we take 5000 uniform random sample (x,y) from all the pixel coordinates of the image, and store the RGB intensity of the sample points in an array. Then, we normalize the pixel intensities to lie between 0.0 and 1.0 by dividing each RGB intensity by 255. We build the final image by plotting the processed dataset using matplotlib.pyplot library.

For experimentation, we make two functions: *rfPredictions* and *plotRF.*

rfPredictions: This function is used to predict the RGB intensities of all pixels in an image based on the trained dataset using Random Forest. First, we need four inputs, which are the coordinates, labels, number of depths and number of trees needed. We create a Random Forest model using the *RandomForestRegressor* with the number of depths and trees required. Then, we fit the model onto the coordinates and labels given using the *fit* function, and, with the *predict* function, predict the labels for all of the coordinates on the image. Finally, the function returns the RGB predictions of the entire image.

plotRF: This function is used to plot predicted images using Random Forest and save the images in the desired directory. First, we need two inputs, which are the RGB dataset of the image and the title of the plot. We load the dataset onto a plot, remove ticks on the x and y axis, insert a title onto the plot, save the figure into the directory and show the completed plot.

Using rfPredictions and plotRF functions, we create images based on the random forest containing a single decision tree with depths 1, 2, 3, 5, 10, 15. Then, we create images based on random forest containing 1, 3, 5, 10, 100 with a depth of 7. In the end, we repeat the experiment using a KNN regressor with the *KNeighborsClassifier* function and compare the results with the images created using Random Forest regressor.

C. Results

Problem 2 – Random forests for image approximation:

What other preprocessing steps are necessary for random forests inputs? Describe them, implement them,and justify your decisions. In particular, do you need to perform mean subtraction, standardization, or unit-normalization?

With Random Forest, very little pre-processing needs to be done. Because Random Forest uses average/majority class, Random Forest is robust to outliers. We also do not usually have to worry about doing any type of data transformation. Non-important variables are set aside automatically, so we do not have to do any selection or elimination of dimensions either. Therefore, all we did was normalize each pixel value to a number between 0.0 and 1.0

The original and preprocessed images are as follows:


Original


Scaled Image

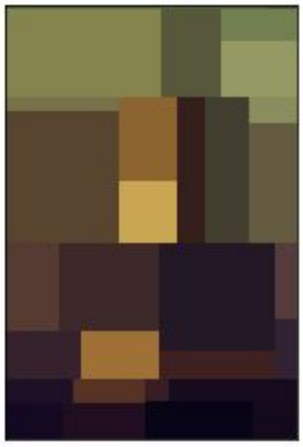
Experiment with depths in Random Forest:
Images created using Random Forest Regressor with a single tree and depth of 1, 2, 3, 5, 10, 15:

Random Forest with Depth 1

Random Forest with Depth 2

Random Forest with Depth 3

Random Forest with Depth 5



Random Forest with Depth 10

Random Forest with Depth 15

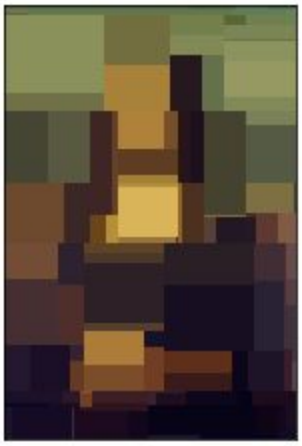
How does the depth impact the result? Describe in detail why.

As shown from the images above, when the depth increases, the image gets clearer and becomes more similar to the original image. This is because when there are more depths, each pixels will be categorized more times before it lands within a certain bucket. For example, if depth = 1, the pixels get categorized into two buckets in the one layer and that's the end of the random forest algorithm. Hence, in the random forest with depth 1 image, there are only two colors, which, in this case, are the categories. If depth = 2, the pixels get categorized into two buckets in the first depth, and within the bucket, each pixel gets categorized into another two buckets in the second layer. Hence, in the random forest with depth 2 image, there are four colors. As the depth increases, there are more layers within the random forest, and pixels are categorized into more specific and detailed buckets. Hence, our image will end up with more colors.

Experiment with number of trees:

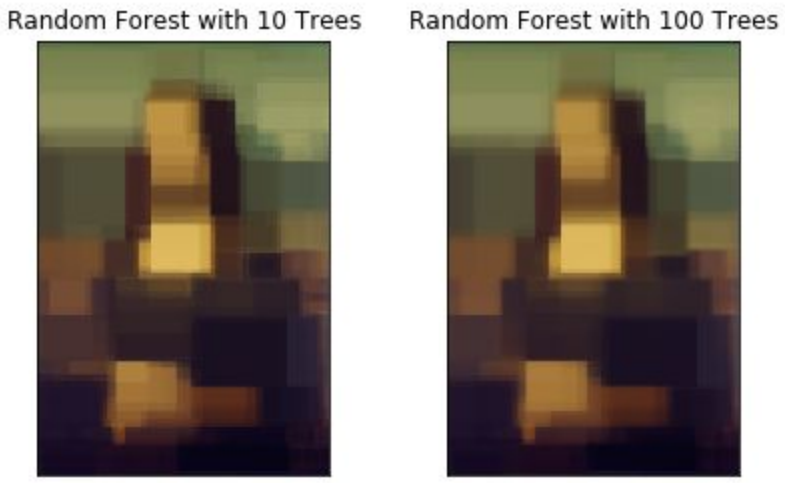
Images created using Random Forest with depth of 7 and number of trees = 1, 3, 5, 10, 100:

Random Forest with 1 Tree



Random Forest with 3 Trees

Random Forest with 5 Trees

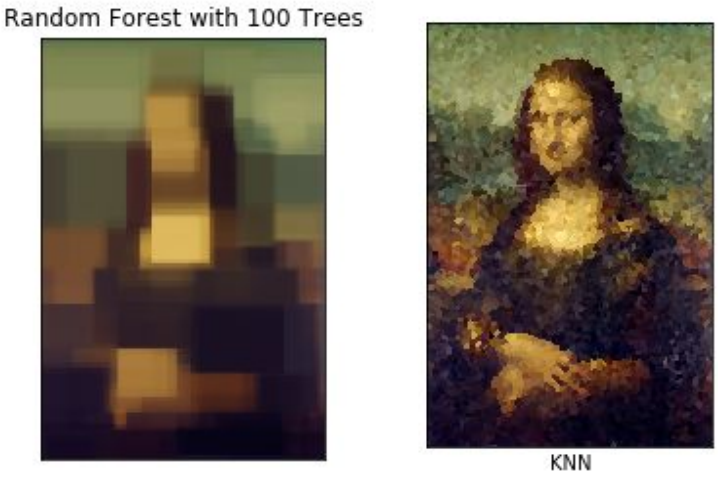


How does the number of trees impact the result? Describe in detail why.

Similar to increasing depth, the image becomes clearer and more similar to the original image when the number of trees increases. This is because a larger number of trees within a Random Forest algorithm gives a prediction model with better accuracy, and has more "point of references" (trees) to categorize the pixels into a particular RGB intensity. In our case, the pixel, based on the trained model, will be categorized into a more accurate RGB intensity if the model trained has a larger number of trees. For example, the image in the random forest with 1 tree has larger clusters (bigger groups of the same color) than the other images with higher number of trees. On the other hand, the image of random forest with 100 trees has more faded and vague edges and less color clusters.

Experiment with KNN vs Random Forest:

Image created by Random Forest with 100 Trees vs Image created by KNN:



Compare and contrast the outlook: why does this look the way it does?

With Random Forest, the image has more rectangular clusters with sharper edges. On the other hand, with KNN, the image has more circular clusters that is clearer and more similar to the original image. This happens because in Random Forest, each layer is divided into two

categories, which splits a group of pixels into two rectangles, giving the cluster a rectangular shape. When there are overlaps between clusters, the overlap would just be cut into smaller squares, since all the clusters are rectangular. In KNN, each pixel is classified into a RGB intensity based on its closest neighbor (k = 1). Since KNN uses euclidean distance, the pixel would pick its nearest neighbor within its shortest radius, which gives the clusters a circular shape. Based on the two images above, KNN with k = 1 seems to be a more accurate model.

Analysis

What is the decision rule at each split point? Write down the 1-line formula for the split point at the root node for one of the trained decision trees inside the forest. Feel free to define any variables you need.

The input for each split point is the image pixel (x,y), and the output would be the leaf node on the next level of the tree until the leaf node is the RGB intensity of the pixel.

Let the threshold be T.
If x >= T, the output would be the left leaf node on the next level. If x<T, the output would be the right leaf node on the next level.

Why does the resulting image look like the way it does? What shape are the patches of color, and how are they arranged?

Please refer to above question "Compare and contrast the outlook: why does this look the way it does?"

How many patches of color may be in the resulting image if the forest contains a single decision tree? Define any variables you need.

Let the maximum depth of a tree be d. Since each split point of the decision has two leaf nodes, there are two patches of color at each depth. For example, if the depth is 2, the first split point has two leaf nodes, and each leaf node is split into another two leaf nodes. In the end, we are left with 4 patches of color. Therefore, if there is only one tree, the number of patches of color would be 2^d.

How many patches of color might be in the resulting image if the forest contains n decision trees? Define any variables you need.

Based on the question above, if there is one tree, there would be 2^depth patches of color. For n trees, the number of patches of color is at most n*2^depth. There are (n*2^depths choose n) ways of combinations to choose n subtrees out of n*2^depth options.

D. Conclusion

Problem 2 – Random forests for image approximation

How well your solution works and any insights you found

Random forest is a simple but powerful regression and classification tool that is easy to use and requires little pre-processing. It works better the greater the depth and number of trees, however that does make it more computationally heavy and therefore slower. Therefore, we should always try to find the optimal depth and number of trees to balance out the accuracy and computational cost. While our experiment with KNN (k = 1) yielded a more accurate depiction of the actual picture, it is worth noting that overfitting will never be an issue when we use the random forest algorithm in any classification problem.

E. Problem 1- Approximating images with neural networks

a. Describe the structure of the network. How many layers does this network have? What is the purpose of each layer?

The network has 9 layers (1 input, 1 output, and 7 intermediary layers). The purpose of each layer is to identify specific components within a picture.

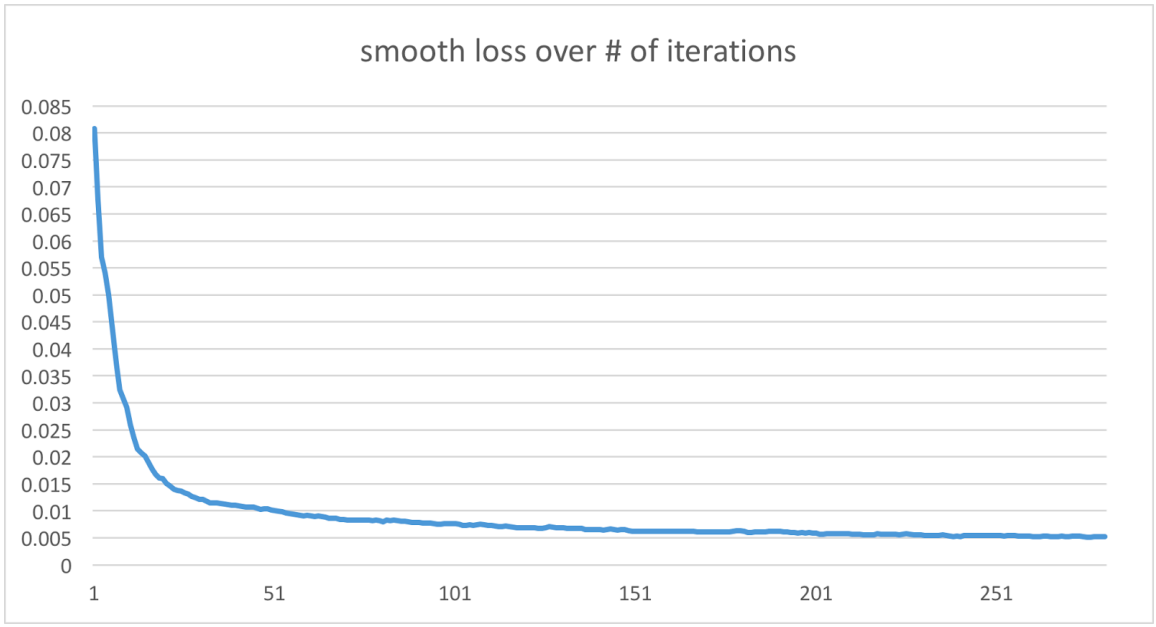
b. What does "Loss" mean here? What is the actual loss function? You may need to consult the source code, which is available on Github.

According to the source code on Github, other than the loss value of the first iteration, the formula for all subsequent loss values are:

smooth_loss = 0.99*smooth_loss + 0.01*loss;

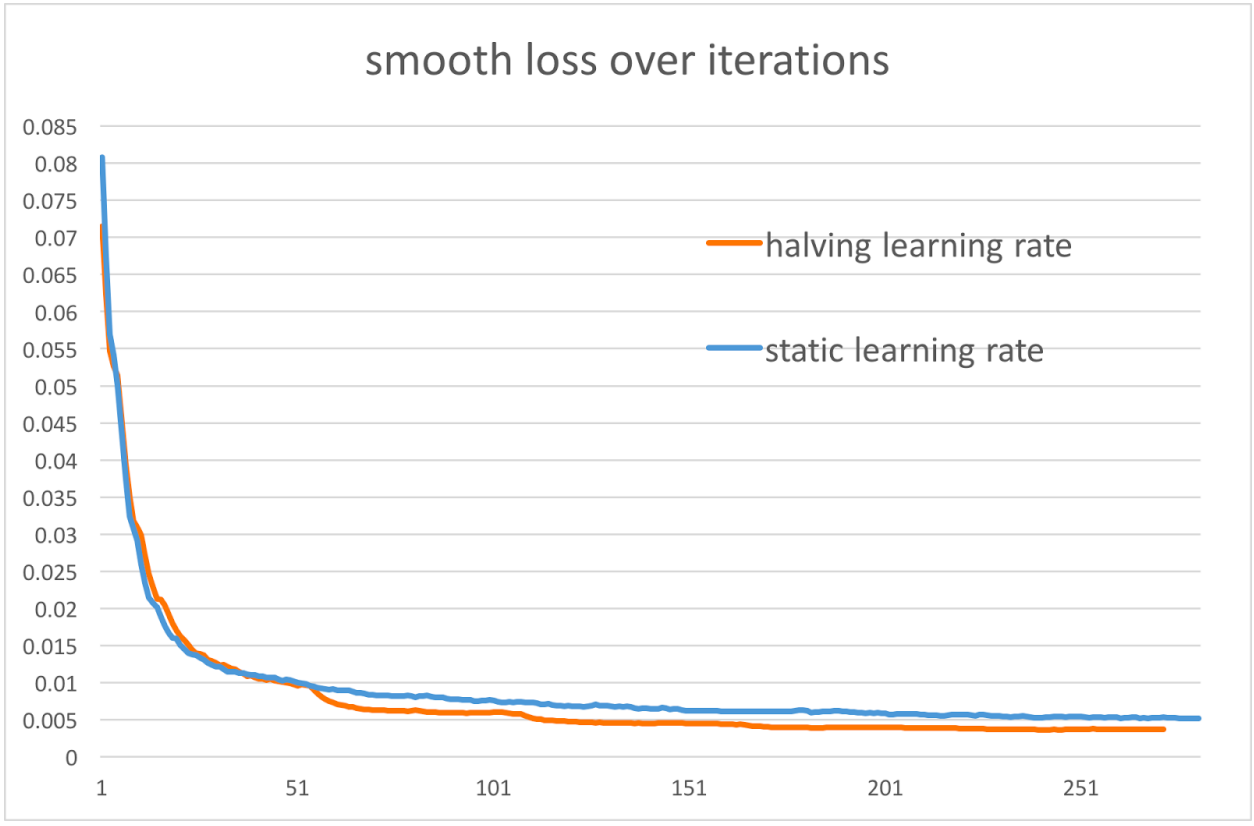
where "*smooth_loss*" is the current loss, and "*loss*" is the previous iteration's loss

c. Plot the loss over time, after letting it run for 5,000 iterations. How good does the network eventually get?



The smooth loss converges to ~0.005

d. Can you make the network converge to a lower loss function by lowering the learning rate every 1,000 iterations? (Some learning rate schedules, for example, halve the learning rate every n iterations.) Does this technique let the network converge to a lower training loss?



The network converges to a slightly lower training loss (~0.37) over the same number of iterations, so we may say that it is more efficient.

e. Lesion study. The text box contains a small snippet of Javascript code that initializes the network. You can change the network structure by clicking the "Reload network" button, which simply evaluates the code. Let's perform some brain surgery: Try commenting out each layer, one by one. Report some results: How many layers can you drop before the accuracy drops below a useful value?

At 3 layers the loss becomes noticeable, but at 5 layers removed the accuracy drops below a useful value

How few hidden units can you get away with before quality drops noticeably?

3* 20 hidden units = 60

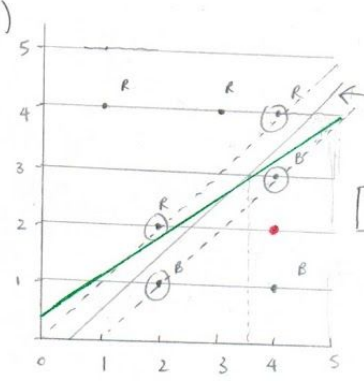
f. Try adding a few layers by copy+pasting lines in the network definition. Can you noticeably increase the accuracy of the network?

No, there does not seem to be much improvement.

F. Written Questions

Written Question Q1

1a)



1b) Classify to Red if $X_2 - X_1 + 0.5 \geq 0$

Classify to Blue if $X_2 - X_1 + 0.5 \leq 0$

$\beta_0 = 0.5$, $\beta_1 = -1$, $\beta_2 = 1$

maximum-margin separating hyperplane

1c) The margins for the maximal margin hyperplane are the dotted lines on the graph.

1d) The support vectors are the data points that are circled on the graph. The coordinates are (2,2), (2,1), (4,3), (4,4).

1e) From the graph, we see that (4,1), the seventh observation is not a support vector. Hence, it would not affect the margins or the maximal margin hyperplane itself.

1f) The green line on the graph is a hyperplane that separates the data but is not a maximal margin hyperplane. The equation for the hyperplane is $\frac{7}{10} X_1 + 0.5 - X_2 = 0$.

1g) The red dot on the graph is an additional observation so that the two classes are no longer separable by a hyperplane.

Written Question Q2

Functions in the graph:

$\hat{y}_0 = 0$

$\hat{y}_1 = 2x - 2$

$\hat{y}_2 = \frac{1}{3}x + \frac{4}{3}$

$\hat{y}_3 = 2x - 7$

$\hat{y}_4 = -\frac{5}{3}x + 15$

$\hat{y}_5 = 0$

Relu functions:

$y_1 = \delta(\hat{y}_1 - \hat{y}_0) = \delta(2x-2)$

$y_2 = \delta(\hat{y}_2 - \hat{y}_1) = \delta\left(\frac{1}{3}x + \frac{4}{3} - 2x + 2\right) = \delta\left(-\frac{5}{3}x + \frac{10}{3}\right)$

$y_3 = \delta(\hat{y}_3 - \hat{y}_1) = \delta\left(x - 7 - \frac{1}{3}x - \frac{4}{3}\right) = \delta\left(\frac{5}{3}x - \frac{25}{3}\right)$

$y_4 = \delta(\hat{y}_4 - \hat{y}_3) = \delta\left(-\frac{5}{3}x + 15 - 2x + 7\right) = \delta\left(-\frac{11}{3}x + 22\right)$

$y_5 = \delta(\hat{y}_5 - \hat{y}_4) = \delta\left[0 - \left(-\frac{5}{3}x + 15\right)\right] = \delta\left(\frac{5}{3}x - 15\right)$

$$Y = \delta(2x-2) + \delta\left(-\frac{5}{3}x + \frac{10}{3}\right) + \delta\left(\frac{5}{3}x - \frac{25}{3}\right) + \delta\left(\frac{-11}{3}x + 22\right) + \delta\left(\frac{5}{3}x - 15\right)$$

Try to make β_1 all -1 but keep the function the same:

$$Y = 2\delta\left(\frac{1}{2}(2x-2)\right) - \frac{10}{3}\delta\left(-\frac{3}{10}\left(-\frac{5}{3}x + \frac{10}{3}\right)\right) + \frac{25}{3}\delta\left(\frac{3}{25}\left(\frac{5}{3}x - \frac{25}{3}\right)\right) - 22\,\delta\left(\frac{-1}{22}\left(-\frac{11}{3}x + 22\right)\right) + 15\,\delta\left(\frac{1}{15}\left(\frac{5}{3}x - 15\right)\right)$$

$$\therefore Y = \delta(2y_1) + \delta\left(-\frac{10}{3}y_2\right) + \delta\left(\frac{25}{3}y_3\right) + \delta(-22y_4) + \delta(15y_5)$$

$$\Rightarrow W_2 = \begin{bmatrix} 2 \\ -\frac{10}{3} \\ \frac{25}{3} \\ -22 \\ 15 \end{bmatrix} \qquad \beta_2 = 0$$

$y_1 = \delta(x-1)$

$y_2 = \delta\left(\frac{1}{2}x - 1\right)$

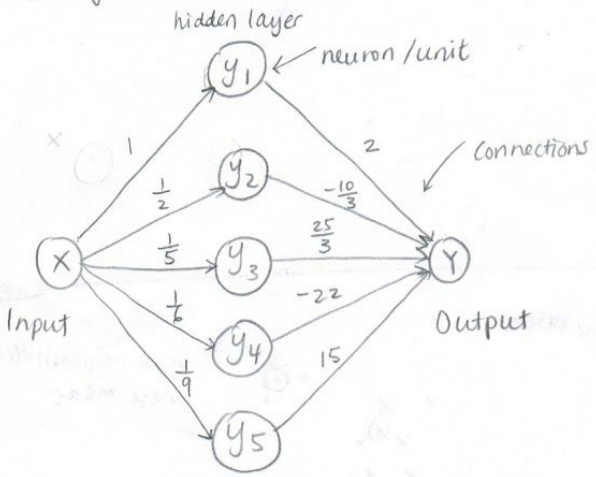
$y_3 = \delta\left(\frac{1}{5}x - 1\right)$

$y_4 = \delta\left(\frac{1}{6}x - 1\right)$

$y_5 = \delta\left(\frac{1}{9}x - 1\right)$

$$W_1 = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{5} \\ \frac{1}{6} \\ \frac{1}{9} \end{bmatrix} \qquad \beta_1 = -1$$

Diagram



∴ We need one hidden layer that contains 5 neurons