CS5785 Fall 2018: Homework 4

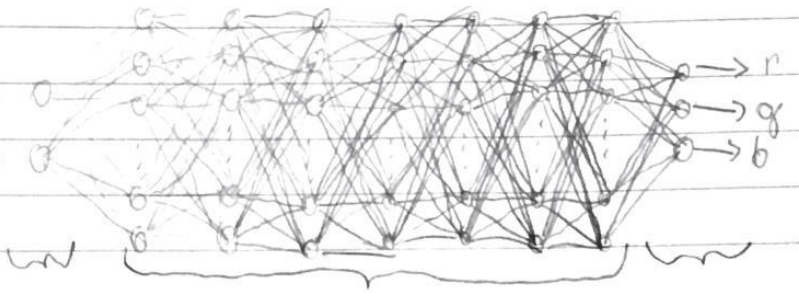
Matheus Clemente Bafutto (mc2722)

Sungseo Park (sp2528)

AML - Programming Exercise

1

a)



2 pixel naive input layer

7 fully connected layers with 20 units each. Activation = ReLU (processes y & x)

3 unit regression output layer (outputs numeric rgb values for the pixel at x & y.)

b) Loss for this convnetjs demo in specific is the L2 loss. The L2 is the squared difference between the network's output and the example's ground truth label.

$$L_i = ||f - y_i||_2^2 \ *$$

* reference: - cs231n.github.io/neural-networks-2/
- https://cs.stanford.edu/people/karpathy/convnetjs/docs.html

c) OK!

d) Yes, halving the learning rate every 1000 iterations has resulted in a final Loss value of 0.0033153. This value is nearly 42% smaller than its counterpart Loss without any learning weight decay (valued 0.00564133) also at 5000 iterations).

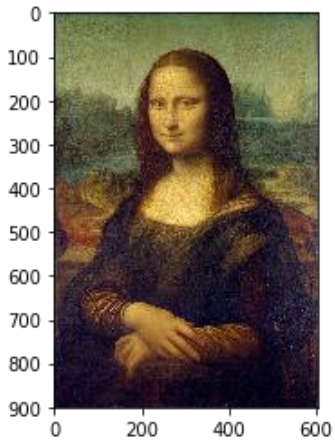
e) Lesion study has pointed lack of significant losses in cost minimization up to 3 out of the 7 20-unit hidden layers. Loss stayed stable at 0.005. After removing 4th hidden layer, significant decreases in paint quality started appearing. By the 6th layer removal, the drawing did not resulted in any cat-like form to the human observer.

f) An experiment was run with a network with double the hidden layer count of the original network. This network was able to outperform previous iterations, but not by a large amount. It also took a lot more time to run, the network ran for 15000 iterations and its loss was of 0.0027 compared to 0.003 of the original with the same learning rate.

Q2. Random forests for image approximation.

Q2(a). Start with an image of the Mona Lisa. If you don't like the Mona Lisa, pick another interesting image of your choice.

```
# Q2-a: Load Image
# Reference: https://www.geeksforgeeks.org/reading-images-in-python/
import matplotlib.image as mpimg
import numpy as np
from matplotlib import pylab as plt

image_rgb = mpimg.imread('MonaLisa.jpg')
plt.imshow(image_rgb)
plt.show()
```



Q2(b). Preprocessing the input. To build your "training set," uniformly sample 5,000 random (x, y) coordinate locations.

• 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?

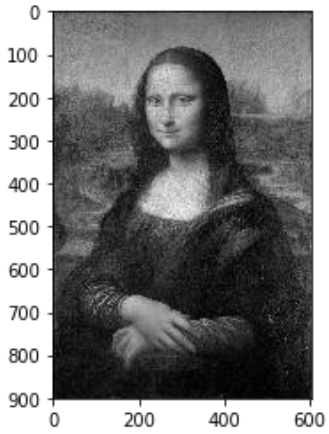
```
# Q2-
b: Preprocessing the input to build "training set," uniformly sample 5,000 random (x, y) coordinate l
ocations.
# Reference: https://www.geeksforgeeks.org/python-randint-function/
from random import randint

width = image_gray.shape[1]
height = image_gray.shape[0]

brightnesses = []
ran_coordinates = []
for i in range(5000):
    w = randint(0, width-1)
    h = randint(0, height-1)
    brightness = image_gray[h, w]
    brightnesses.append(brightness)
    ran_coordinates.append([h, w])
```

For the decision tree, the continuous input data does not need important preprocessing. Since the trees can select equivalent splits, the result of the tree would be consistent regardless of mean subtraction, standardization and unit-normalization.

Reference: https://datascience.stackexchange.com/questions/6721/how-to-preprocess-different-kinds-of-data-continuous-discrete-categorical-be

Q2(c). Preprocessing the output. Sample pixel values at each of the given coordinate locations. Each pixel contains red, green, and blue intensity values, so decide how you want to handle this. There are several options available to you:

```
# Q2-c: Preprocessing the output (Convert the image to grayscale)
# Reference - https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python\
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

image_gray = rgb2gray(image)
plt.imshow(image_gray, cmap = plt.get_cmap('gray'))
plt.show()
```

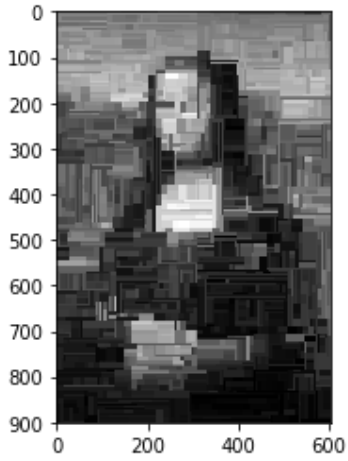


We chose to convert the image into grayscale. Transforming the image to grayscale will reduce the amount of the data processing time because color information is not necessary for performing the random forest.

Reference: https://stackoverflow.com/questions/12752168/why-we-should-use-gray-scale-for-image-processing

Q2(d). To build the final image, for each pixel of the output, feed the pixel coordinate through the random forest and color the resulting pixel with the output prediction.

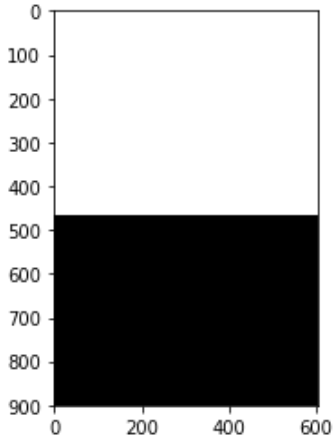
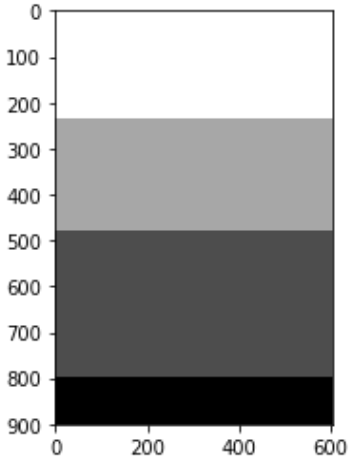
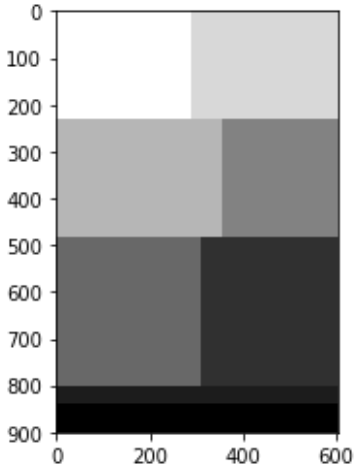
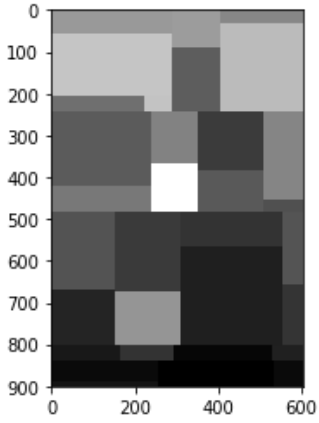
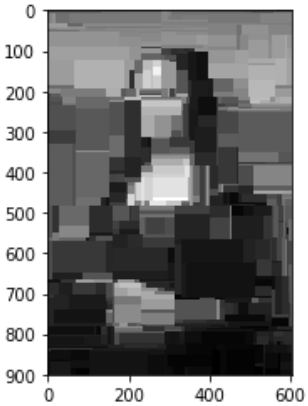
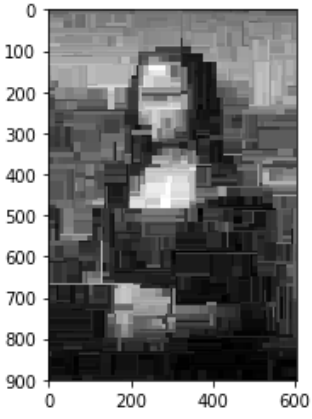
```
# Q2-d: Build the final image.
# Reference: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
from sklearn.ensemble import RandomForestRegressor

# Build an image according to the # of trees and depth.
def buildImage(num_trees=1, depth=None):
    rfr = RandomForestRegressor(n_estimators=num_trees, max_depth=depth)
    rfr.fit(ran_coordinates, brightnesses)
    prediction = rfr.predict(all_coordinates)
    plt.imshow(np.array(prediction).reshape(height, width), cmap="gray")
    plt.show()

buildImage()
```



Q2(e). Experimentation

i. Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15. How does depth impact the result? Describe in detail why.

```
# Q2-e(i): Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15.
depths = [1, 2, 3, 5, 10, 15]
for depth in depths:
    print('Depths: ' + str(depth))
    buildImage(1, depth)
```

Depth = 1 	Depth = 2 	Depth = 3 
Depth = 5 	Depth = 10 	Depth = 15 

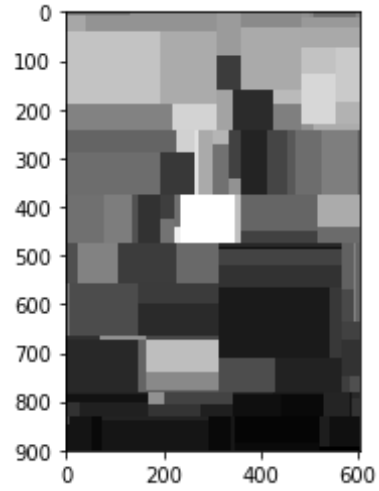
As depth increases, the number of splits would exponentially increase. For example, the number of splits (=nodes) would be 8 if depth is 3 ($2^3 = 8$). Thus, it works like binary tree since as depth increases, the number of nodes exponentially increases (deeper tree reduces the bias the model).

References:
http://www2.ims.nus.edu.sg/Programs/014swclass/files/mark.pdf
https://stats.stackexchange.com/questions/53240/practical-questions-on-tuning-random-forests

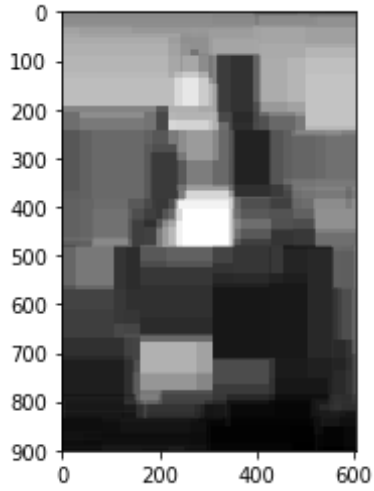
ii. Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.

```
# Q2-e(ii): Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100.
depth = 7
trees = [1, 3, 5, 10, 100]
for tree in trees:
    print('Depth: ' + str(depth) + ', Number of trees: ' + str(tree))
    buildImage(tree, depth)
```

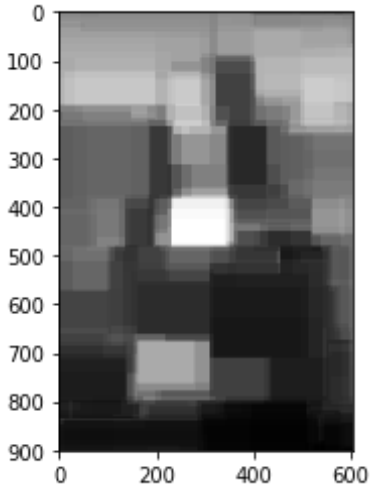
# of trees: 1



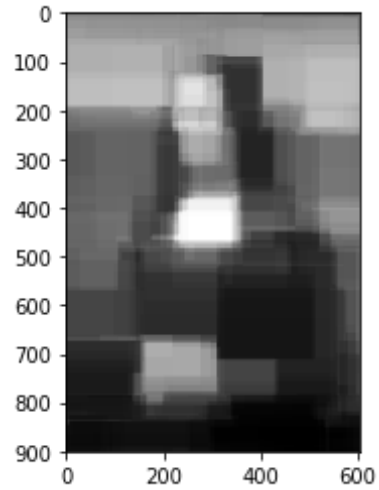
# of trees: 3



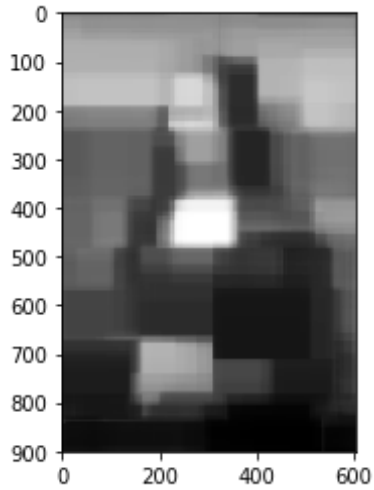
# of trees: 5



# of trees: 10



# of trees: 100

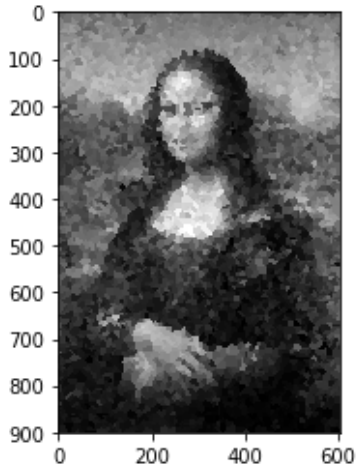


As the number of trees increases, the image becomes smoother because the increase in the number of trees (more prediction models) will reduce the variance and therefore makes better prediction.

References:
https://www.kaggle.com/general/4092
https://stats.stackexchange.com/questions/53240/practical-questions-on-tuning-random-forests

iii. As a simple baseline, repeat the experiment using a k-NN regressor, for k = 1. This means that every pixel in the output will equal the nearest pixel from the "training set." Compare and contrast the outlook: why does this look the way it does?

```
# Q2-e(iii): Repeat the experiment using a k-NN regressor, for k = 1
# Reference: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing

# Rounding a list of floats into integers.
new_brightness = [round(x) for x in brightnesses]

# Perform KNN with k=1.
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(ran_coordinates, new_brightness)
prediction = knn.predict(all_coordinates)
plt.imshow(np.array(prediction).reshape(height, width), cmap="gray")
plt.show()
```

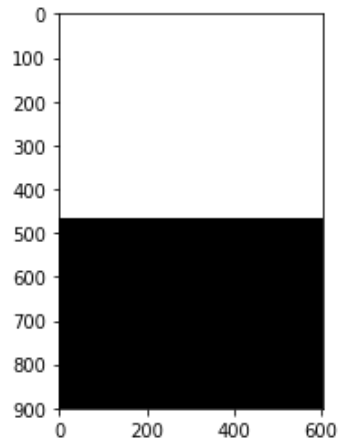


The image generated by kNN consists of rounded shapes because each pixel is classified by a majority vote of its neighbors. If k=1, then the pixel is assigned to the class of that single nearest neighbor. On the contrary, the decision trees for Random Forest divide the input space into axis-parallel rectangles. In this way, pixels are classified through the decision trees and placed in one of the rectangles. This is the reason that the image generated by Random Forest consists of rectangles.

References:
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/decisiontree-4.pdf
https://healthcare.ai/visual-tour-lasso-random-forest/

Q2(f). Analysis

i. 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 the trained decision trees inside the forest. Feel free to define any variables you need.



The image on the left shows a single decision tree with depth = 1. The decision tree classifies each pixel with its coordinates using the decision rule. For example, the left node is classified as white and the right node is classified as black. If the y-coordinate of a pixel is greater than ≅ 540, the pixel is classified as white, otherwise it's black. Therefore, the decision rule for the image on the left would be:

If y-coordinate of a pixel > 540:
    left_node (=white)
else:
    right_node (=black)

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

The decision trees for Random Forest divide the input space into axis-parallel rectangles. In this way, pixels are classified through the decision trees and placed in one of the rectangles. This is the reason that the image generated by Random Forest consists of rectangles. Since the image above shows a single decision tree with depth = 1, there are two nodes and colors in the image. As the depth increases, the number of nodes and colors increases.

Reference: http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/decisiontree-4.pdf

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

If the forest contains a single decision tree, the number of nodes/colors are determined by the depth of the tree. For example, if the depth is 3, the number of nodes/colors is 2^3 (=8). Thus, it works like a binary tree since as the depth increases, the number of nodes/colors exponentially increases.

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

Since a single decision tree has 2^d nodes/colors (d=depth), n decision tree has $n*2^d$. However, there may be overlapping nodes/colors in the trees. Thus, the number of colors in the resulting image can be reduced by the number of unique nodes (# of colors $\leq n*2^d$).

References:

http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf
https://math.tutorvista.com/geometry/distance-between-two-parallel-lines.html
https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
https://www.geeksforgeeks.org/python-randint-function/
https://stats.stackexchange.com/questions/53240/practical-questions-on-tuning-random-forests
http://www2.ims.nus.edu.sg/Programs/014swclass/files/mark.pdf
https://www.kaggle.com/general/4092
https://healthcare.ai/visual-tour-lasso-random-forest/
https://datascience.stackexchange.com/questions/6721/how-to-preprocess-different-kinds-of-data-continuous-discrete-categorical-be
https://stackoverflow.com/questions/12752168/why-we-should-use-gray-scale-for-image-processing
http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/decisiontree-4.pdf