AML Homework 3

Yue (Billy) Liu (yl992)

Teammate: Zheng Zhou (zz273)

Programming Exercise

Question 1: ConvolutionalNeuralNetworks

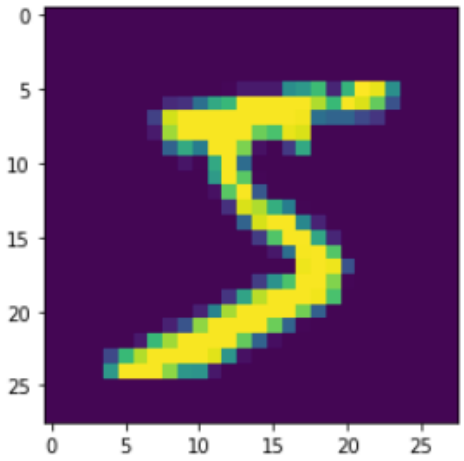
(a) Loading Dataset

```
(train_X, train_Y), (test_X, test_Y) = mnist.load_data()
#printing out the shape of train an dtest dataset matrices
print("The shape for train_X is: " + str(train_X.shape))
print("The shape for train_Y is: " + str(train_Y.shape))
print("The shape for test_X is: " + str(test_X.shape))
print("The shape for test_Y is: " + str(test_Y.shape))
#Vsisualize individual images
plt.imshow(train_X[0])
```

Printing out the shape of train and test metrics

```
The shape for train_X is: (60000, 28, 28)
The shape for train_Y is: (60000,)
The shape for test_X is: (10000, 28, 28)
The shape for test_Y is: (10000,)
```

Visualize a image using imshow()



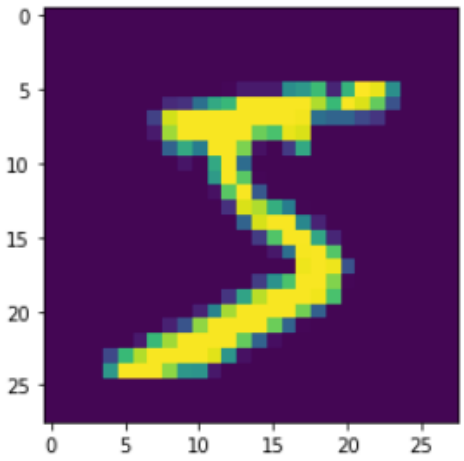
(b) Preprocessing

Reshape, Scaling, Output One-Hot Encoding in one python function

```
def preprocess(x, y):
    #Scale the pixel values so they lie between 0.0 and 1.0
    x = x.astype("float32") / 255
    #Reshape the matrix to 28*28*1
    x = np.expand_dims(x, -1)
    y = np.expand_dims(y, -1)
    #One-hot encode output variable
```

```
    y = to_categorical(y, 10)

    return x, y

train_X, train_Y = preprocess(train_X, train_Y)
test_X, test_Y = preprocess(test_X, test_Y)
#Visualize image
plt.imshow(train_X[0])
```

Visualize Image



(c) Implementation

Used Implementation Provided

Print model.layers

```
[<keras.layers.convolutional.Conv2D object at 0x00000185AA2FA430>, <keras.layers.pooling.MaxPooling2D object at 0x000
00185A4E5B280>, <keras.layers.core.flatten.Flatten object at 0x00000185A4E5B250>, <keras.layers.core.dense.Dense obje
ct at 0x00000185A4E5B4C0>, <keras.layers.core.dense.Dense object at 0x00000185A4E908B0>]
```

(d) Training and Evaluating CNN

```
model.fit(train_X,train_Y,batch_size=32,epochs=10,validation_split=0.1)
score = model.evaluate(test_X,test_Y,verbose =0)
print("The loss and accuracy on test data is: " + str(score))
```

Report accuracy on test data

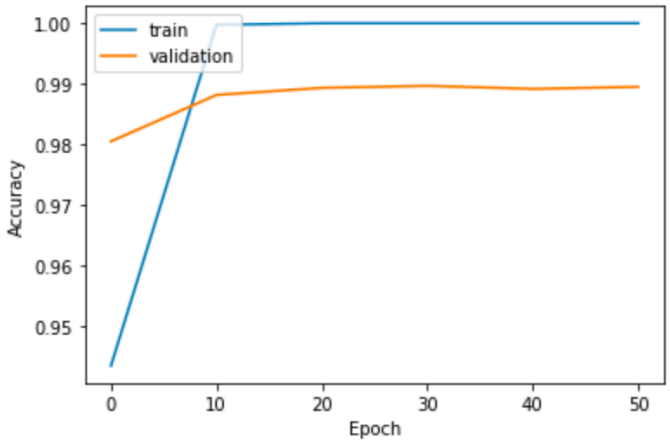
```
The loss and accuracy on test data is: [0.03966928645968437, 0.9879000186920166]
```

(e) Experimentation

(i)

```
model2 = create_cnn()
epoch_history = model2.fit(train_X,train_Y,
                           batch_size=32,
                           epochs=50,
                           validation_split=0.1)
#summarize history for accuracy
plt.plot(epoch_history.history['accuracy'])
plt.plot(epoch_history.history['val_accuracy'])
plt.title('Validation Accuracy and Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Graph the validaton and training accruacy after every 10 epochs



Is there a steady improvement for both training and validatoin accuracy?

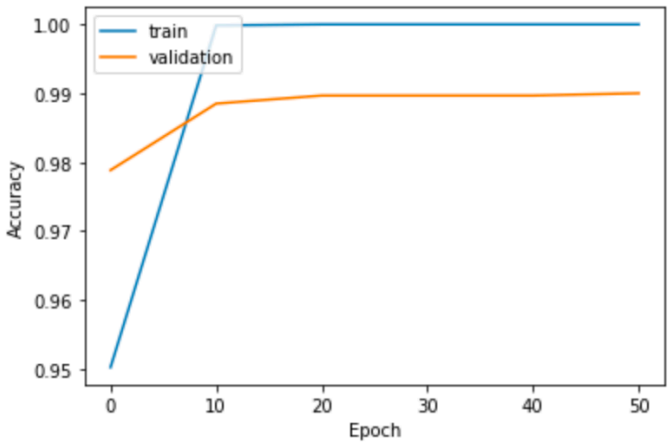
The validation accuracy and training accuracy rised quickly in the first 10 epoch. Validation accuracy peaked and fluctuated around that peak.

(ii)

```
#Function to build model with dropout
def create_cnn_with_dropout():
    # define using Sequential
    model = Sequential()
    # Convolution layer
    model.add(Conv2D(32,(3,3),
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer
    model.add (MaxPooling2D ((2, 2)))
    # Flatten output
    model.add(Flatten())
    # Dropout
    model.add(Dropout(0.5))
    # Dense layer of 100 neurons
    model.add(Dense(100, activation = 'relu', kernel_initializer = 'he_uniform'))
    model.add (Dense (10 , activation= 'softmax' ) )
    # initialize optimizer
```

```
    opt = SGD(lr=0.01, momentum=0.9)
    # compile model
    model.compile (optimizer=opt,
                   loss = 'categorical_crossentropy',
                   metrics =['accuracy'])
    return model

model3 = create_cnn_with_dropout()

epoch_history = model3.fit(train_X,train_Y,
                           batch_size=32,
                           epochs=50,
                           validation_split=0.1)

plt.plot(epoch_history.history['accuracy'])
plt.plot(epoch_history.history['val_accuracy'])
plt.title('Validation Accuracy and Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Graph the validation and train accuracy after every 10 epochs



(iii)

```
def create_cnn_with_dropout_double():
    # define using Sequential
    model = Sequential()
    # Convolution layer
    model.add(Conv2D(32,(3,3),
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer
    model.add (MaxPooling2D ((2, 2)))
    # Convolution layer 2
    model.add(Conv2D(64,(3,3),#64 Output filters
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer 2
```

```
    model.add (MaxPooling2D ((2, 2)))
    # Flatten output
    model.add(Flatten())
    # Dropout
    model.add(Dropout(0.5))
    # Dense layer of 100 neurons
    model.add(Dense(100, activation = 'relu', kernel_initializer = 'he_uniform'))
    model.add (Dense (10 , activation= 'softmax' ) )
    # initialize optimizer
    opt = SGD(lr=0.01, momentum=0.9)
    # compile model
    model.compile (optimizer=opt,
                   loss = 'categorical_crossentropy',
                   metrics =['accuracy'])
    return model

model4 = create_cnn_with_dropout_double()
epoch_history = model4.fit(train_X,train_Y,
                           batch_size=32,
                           epochs=10,#Train for 10 epoches
                           validation_split=0.1)
score = model4.evaluate(test_X,test_Y,verbose =0)
print("The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer on test
data is: " + str(score))
```

Train this for 10 epochs and report the test accuracy

```
The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer on test data is: [0.02
45823934674263, 0.9911999702453613]
```

(iv)

```
def create_cnn_with_dropout_double_1E1():
    # define using Sequential
    model = Sequential()
    # Convolution layer
    model.add(Conv2D(32,(3,3),
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer
    model.add (MaxPooling2D ((2, 2)))
    # Convolution layer 2
    model.add(Conv2D(64,(3,3),#64 Output filters
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer 2
    model.add (MaxPooling2D ((2, 2)))
    # Flatten output
    model.add(Flatten())
    # Dropout
    model.add(Dropout(0.5))
    # Dense layer of 100 neurons
    model.add(Dense(100, activation = 'relu', kernel_initializer = 'he_uniform'))
    model.add (Dense (10 , activation= 'softmax' ) )
    # initialize optimizer
```

```
    opt = SGD(lr=0.1, momentum=0.9)
    # compile model
    model.compile (optimizer=opt,
                   loss = 'categorical_crossentropy',
                   metrics =['accuracy'])
    return model

def create_cnn_with_dropout_double_1E3():
    # define using Sequential
    model = Sequential()
    # Convolution layer
    model.add(Conv2D(32,(3,3),
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer
    model.add (MaxPooling2D ((2, 2)))
    # Convolution layer 2
    model.add(Conv2D(64,(3,3),#64 Output filters
              activation = 'relu',
              kernel_initializer = 'he_uniform',
              input_shape = (28, 28, 1)))
    # Maxpooling layer 2
    model.add (MaxPooling2D ((2, 2)))
    # Flatten output
    model.add(Flatten())
    # Dropout
    model.add(Dropout(0.5))
    # Dense layer of 100 neurons
    model.add(Dense(100, activation = 'relu', kernel_initializer = 'he_uniform'))
    model.add (Dense (10 , activation= 'softmax' ) )
    # initialize optimizer
    opt = SGD(lr=0.001, momentum=0.9)
    # compile model
    model.compile (optimizer=opt,
                   loss = 'categorical_crossentropy',
                   metrics =['accuracy'])
    return model

model5 = create_cnn_with_dropout_double_1E1()
model6 = create_cnn_with_dropout_double_1E3()

epoch_history = model5.fit(train_X,train_Y,
                           batch_size=32,
                           epochs=10,#Train for 10 epoches
                           validation_split=0.1)
score = model5.evaluate(test_X,test_Y,verbose =0)
print("The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer with
lr=0.1 on test data is: " + str(score))

epoch_history = model6.fit(train_X,train_Y,
                           batch_size=32,
                           epochs=10,#Train for 10 epoches
                           validation_split=0.1)
score = model6.evaluate(test_X,test_Y,verbose =0)
```

```
print("The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer with lr=0.001 on test data is: " + str(score))
```

Train the model and report accuracy on test dataset

```
The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer with lr=0.1 on test data is: [2.3078458309173584, 0.10480000078678131]

The loss and accuracy for model with Dropout and double convolution layer and maxpooling layer with lr=0.001 on test data is: [0.036580294370651245, 0.988099992275238]
```

(f) Analysis

(i) Explain how the trends in validation and train accuracy change after using the dropout layer in the experiments

The validation accruacy increased after using the dropout layer. Additionally, the training accuracy improved more steadily over time.

(ii) How does the performance of CNN with two convolution layers differ as compared to CNN with a single convolution layer in your experiments?

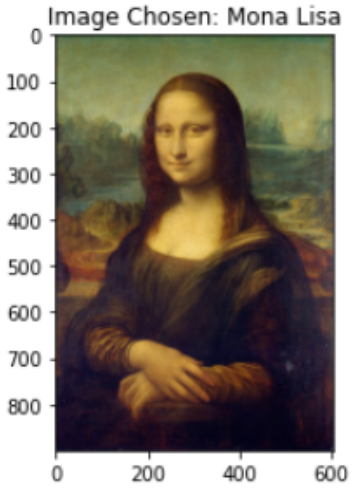
After training for 10 epochs, performance of these two models are very close to each other (0.9910 vs 0.9912), there is really not worth it to add complexity to the model.

(iii) How did changing learning rates change your experimental results inpart(iv)?

A learning rate of 0.1 makes the model performed terribly, the model is barely learning anything while a learning of 0.001 makes the model performed worse than learning rate of 0.01 probably because it makes the model to stuck on a suboptimal solution.

Question 2: Random Forests for Image Approximation

a. Start with an image of the MonaLisa. If you don't like the MonaLisa, pick another interesting image of your choice.



b. Preprocessing the input.

```
#Additional preprocessing step: nomalization of rgb values
def preprocess(data):
    chosen = set()
    pixel_num = data.shape[0] * data.shape[1]
    x = []
    y = []

    for i in range(5000):#Generate coordiantes
        index = int(random.random()*pixel_num)
```

```
        while (index in chosen):
            index = int(random.random()*pixel_num)
        chosen.add(index)

    for i in chosen:
        w = i//data.shape[1]
        h = i%data.shape[1]

        x.append([w,h])
        y.append(data[w][h]/255.0)#Normalizes RGB to 0 and 1

    coordinate = np.array(x)
    rgb = np.array(y)
    return coordinate, rgb

train_X, train_Y = preprocess(image)
```

What other preprocessing steps did you do?

The only additional preprocessing step I did was normalize the rgb value between 0 and 1 to help the model converage faster.

c. Preprocessing the output.

I choose method 3, one function for each channel

```
def preprocess_label(data):
    r = []
    g = []
    b = []

    for i in range(len(data)):
        r.append(data[i][0])
        g.append(data[i][1])
        b.append(data[i][2])

    return np.array(r), np.array(g), np.array(b)

r,g,b = preprocess_label(train_Y)
```

What other preprocessing steps did you do?

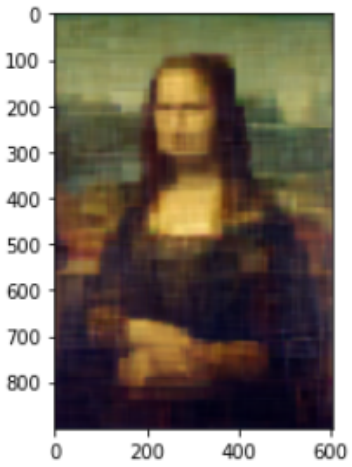
The only additional preprocessing step I did was normalize the rgb value for labels to be between 0 and 1 to help the model converage faster and remain consistent with training data.

d. Build the final image.

```
#I used Random Forrest from sklearn library
def createRF(tree_num=100, depth=None):
    model = RandomForestRegressor(n_estimators = tree_num, max_depth= depth)
    return model

f_r = createRF()
f_g = createRF()
f_b = createRF()

f_r.fit(train_X,r)
f_g.fit(train_X,g)
```

```
f_b.fit(train_X,b)

def show(models,data,train_X,train_Y):
    coordinates = []
    p = train_X.tolist()
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            tmp = [i,j]
            if tmp in p:
                continue
            coordinates.append([i,j])

    r = models[0].predict(coordinates)
    g = models[1].predict(coordinates)
    b = models[2].predict(coordinates)

    pic = []

    for i in range(data.shape[0]):
        pic.append([0]*data.shape[1])

    for i in range(len(coordinates)):
        w = coordinates[i][0]
        h = coordinates[i][1]
        pic[w][h]=[int(r[i]*255),int(g[i]*255),int(b[i]*255)]
#         print(pic[w][h])

    for i in range(len(train_X)):
        w = train_X[i][0]
        h = train_X[i][1]
        pic[w][h] = [int(train_Y[i][0]*255),int(train_Y[i][1]*255),int(train_Y[i][2]*255)]
#         print(pic[w][h])

    pic = np.array(pic)

    fig = plt.figure()
    plt.imshow(pic)
    plt.show()

show([f_r,f_g,f_b],image,train_X,train_Y)
```

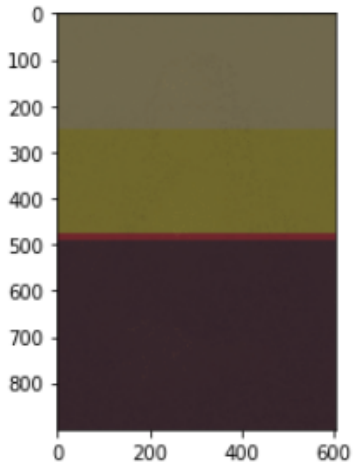
View the result



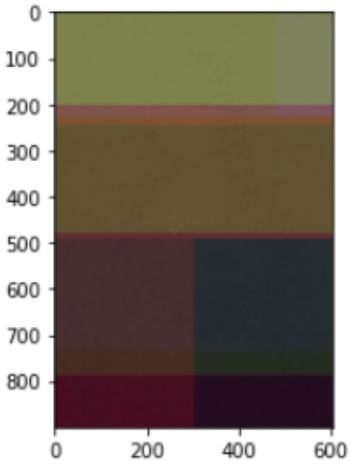
e. Experimentation.

(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

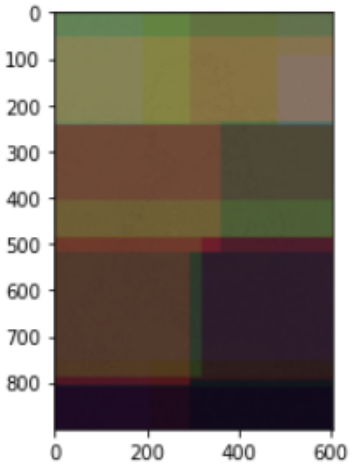
1 Tree, Depth 1



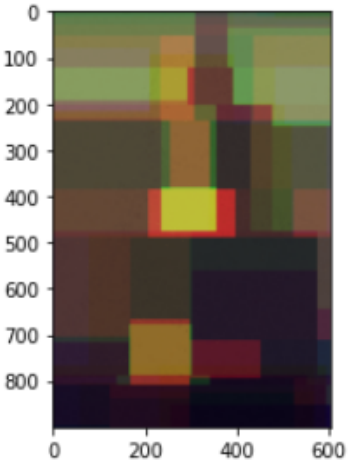
1 Tree, Depth 2



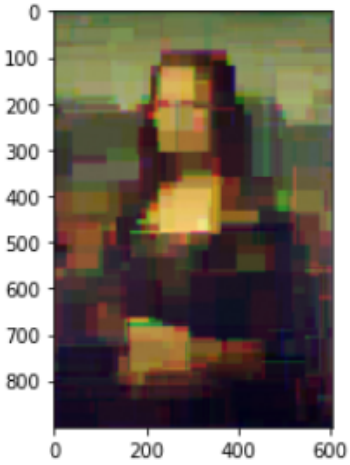
1 Tree, Depth 3



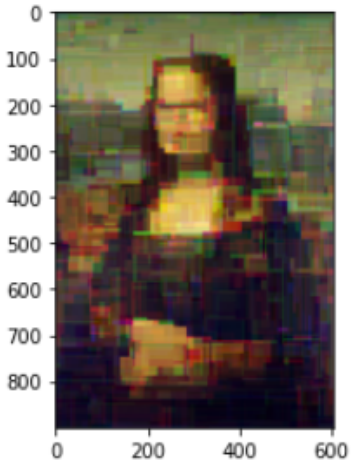
1 Tree, Depth 5



1 Tree, Depth 10



1 Tree, Depth 15

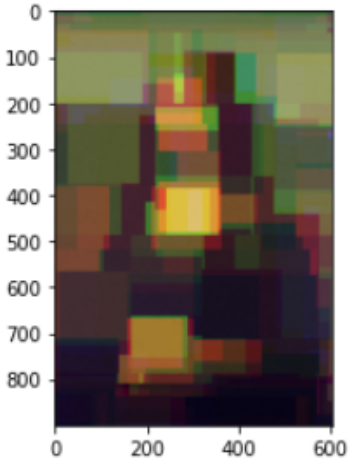


How does depth impact the result? Descibe in detail why

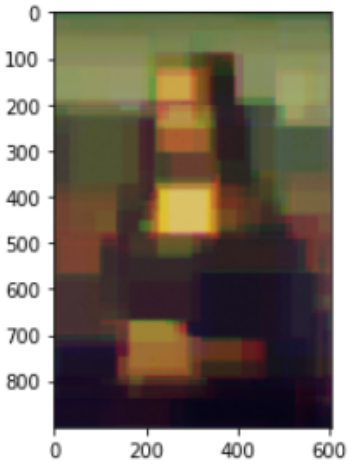
The results show that the image becomes clearer as the depth increases. The reason is that the larger the depth the more divisions can be generate by the random forrest, which contributes to the number of patches of color (similar to that we prefer 4k monitors over FHD because it has more patches of colors).

(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

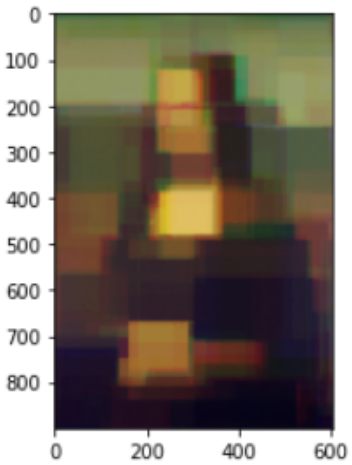
1 Tree, Depth 7



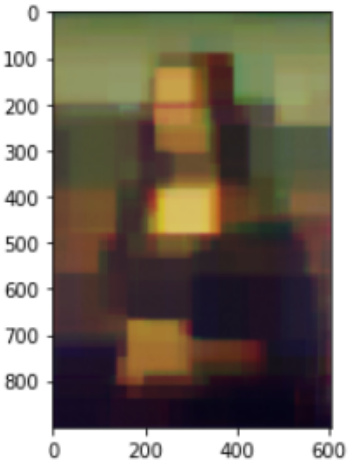
3 Tree, Depth 7



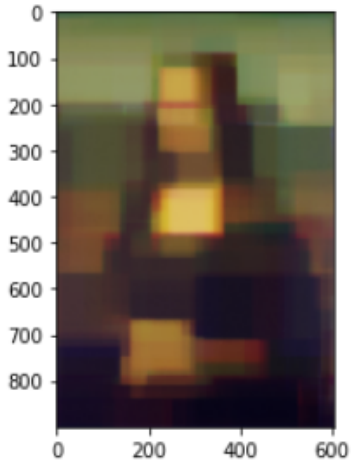
5 Tree, Depth 7



10 Tree, Depth 7



100 Tree, Depth 7

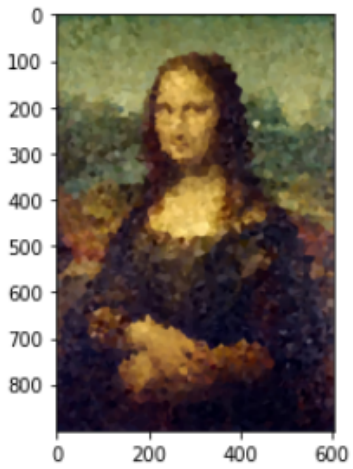


How does the numbre of trees impact the result? Describe in detail why

The results show that the image becomes clearer as the number of trees increases, however, not by a lot after having 5 trees. This is because depth is really what matter when it comes to the number of patches of colors, while having more trees makes the color transition smoother it does not improve the resolution.

(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?

```
f_r = KNeighborsRegressor(n_neighbors = 1)
f_g = KNeighborsRegressor(n_neighbors = 1)
f_b = KNeighborsRegressor(n_neighbors = 1)
f_r.fit(train_X, r)
f_g.fit(train_X, g)
f_b.fit(train_X, b)
show([f_r,f_g,f_b],image,train_X,train_Y)
```



Why does this look the way it does?

The patches of color generated by KNN has different shape compared to patches of color generated by random forest. While random forest generates patches in rectangular shape, KNN generates patches in irregular shapes. This is because KNN uses K-nearest neighbor method when generating divisions.

(iv) Experiment with different pruning strategies of your choice

The pruning strategy I used was to find out the optimum parameters for the random forest model: 1. Number of decision trees 2. Depth of random forest.

In order to do this, I uses 2 for loops to find out the parameters that achieve the best score. The score is the average for red, green, and blue channel.

```
def pruning(num_Tree, num_Depth):
    max_score = 0
    for depth in range(1,num_Depth):
        for tree in range(1,num_Tree):
            f_r = createRF(tree,depth)
            f_g = createRF(tree,depth)
            f_b = createRF(tree,depth)
            f_r.fit(train_X,r)
            f_g.fit(train_X,g)
            f_b.fit(train_X,b)

            score = (f_r.score(train_X,r) + f_g.score(train_X,g) + f_b.score(train_X,b))/3
            if score > max_score*1.01:
                max_score = score
                param = [tree,depth]

    return param
```

```
param = pruning(30,30)
num_Tree = param[0]
num_Depth = param[1]
print("The optimum parameters for the random forest model are " + str(num_Tree) + " trees with a depth of " +
str(num_Depth) + ".")
```

The optimum parameters for the random forest model are 22 trees with a depth of 17.

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 decision tree regressor: $r(x) = \begin{cases} \text{true if} & x_i \geq C \\ \text{false if} & x_i < C \end{cases}$ for each coordinate, where threshold C is learnt from decision rules.

We have the following optimization problem:

$$min_{r \in u}(L((x,y) \in D | r(x) = T) + L((x,y) \in D | r(x) = F))$$

where L is a loss function over a subset of the data flagged by the rule and u is the set of possible rules.

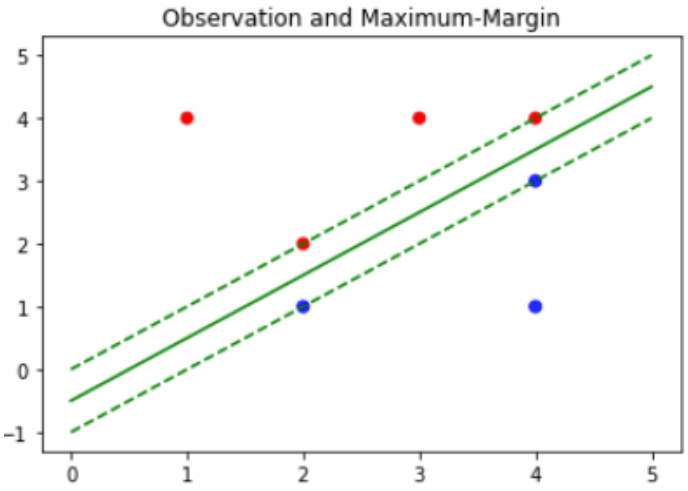
(ii) Why does the resulting image look like the way it does? What shape are the patches of color, and how are they arranged?

KNN generates patches of color of irregular shape while random forest generates patches of color in rectangular shape. This is because divisions of KNN are generated by K-nearest neighbor that yields in divisions of different shapes. On the other hand, random forest regressor uses hyperplanes that are parallel or vertical to each other which results in rectangular shape, and the number of divisions is related to the depth of random forest model.

Written Exercise

Question 1: Maximum Margin Classifiers

a. Sketch the observations and the maximum- margin separating hyperplane.



The solid green line as the maximum margin separating hyperplane

b. Describe the classification rule for the maxi- mal margin classifier.

Classification Rule:

$$\begin{cases} \text{red if} & 0.5 - X_1 + X_2 > 0 \\ \text{blue if} & 0.5 - X_1 + X_2 \leq 0 \end{cases}$$

$$\begin{aligned} \beta_0 &= 0.5 \\ \beta_1 &= -1 \\ \beta_2 &= 1 \end{aligned}$$

c. On your sketch, indicate the margin for the maximal margin hyperplane.

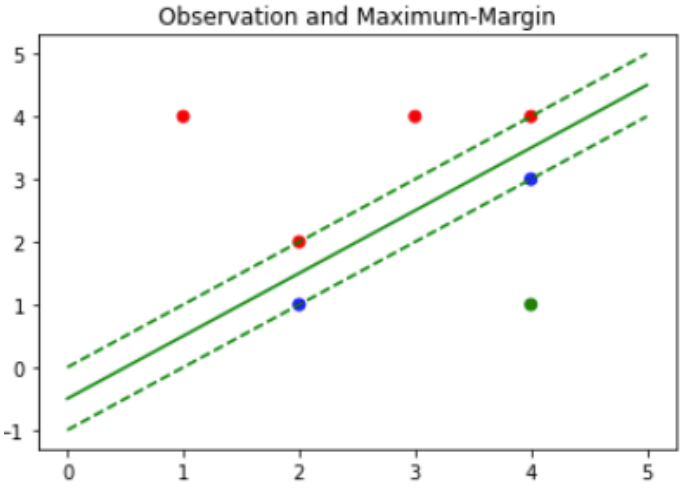
The margin is between (2,2), (4,4) and (2,1), (4,3) as shown in question (a).

d. Indicate the support vectors for the maximal margin classifier.

The support vectors for the maximal margin classifier are the points being passed through by the margins: (2,2), (4,4), (2,1), (4,3).

e. Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

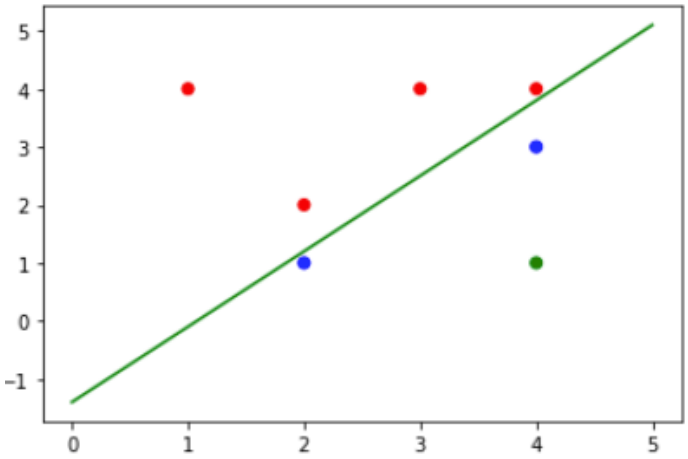
The seventh observation (marked in green) is not one of the support vecntor of the maximal margin, as long as it does not move past the lower margin line, it will not affect the maximal margin hyperplane.



f. Sketch a hyperplane that separates the data, but is not the maximum-margin separating hyperplane. Provide the equation for this hyperplane.

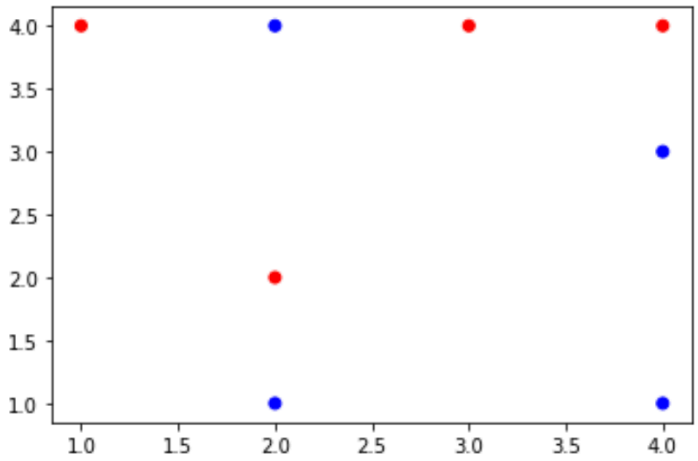
This hyperplane also separates the data but is not the the maximum-margin separating hyperplane.

$$X_2 = 1.3X_1 - 1.4$$



g. Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

By adding (2,4) to the data, the data is no longer seprable by a hyperplane.



Question 2: Neural Networks as Function Approximators

We can approximiate using the equation below:

$$f(x) \simeq b + \sum_{k=0}^{n} a_k * \sigma(x-k)$$

$$\text{From } a_1(x-1) = 2, \text{ we have } a_1 = 2$$

$$\text{From } 2(x-1) + a_2(x-2) = 3, \text{ we have } a_2 = -\frac{5}{3}$$

$$\text{From } 2(x-1) - \frac{5}{3}(x-2) + a_3(x-5) = 5, \text{ we have } a_3 = \frac{5}{3}$$

$$\text{From } 2(x-1) - \frac{5}{3}(x-2) + \frac{5}{3}(x-5) + a_4(x-6) = 0, \text{ we have } a_4 = -\frac{11}{3}$$

$$\text{From } 2(x-1) - \frac{5}{3}(x-2) + \frac{5}{3}(x-5) - \frac{11}{3}(x-6) + a_5(x-9) = 0, \text{ we have } a_5 = \frac{5}{3}$$

How many hidden layers do you need? How many units are there within each layer? Show the hidden layers, units, connections, weights, and biases

Weights and Biases:

$$w_{11} = 1, w_{12} = 1, w_{13} = 1, w_{14} = 1, w_{15} = 1$$
$$w_{21} = 2, w_{22} = -\frac{5}{3}, w_{23} = \frac{5}{3}, w_{24} = -\frac{11}{3}, w_{25} = \frac{5}{3}$$
$$b_{11} = -1, b_{12} = -2, b_{13} = -5, b_{14} = -6, b_{15} = -9$$
$$b_{21} = b_{22} = b_{23} = b_{24} = b_{25} = 0$$

Suppose input is y0, the value in hiddenl ayer is y1, and output is y2. The weights and biases in the graph should match the form:

$$y1 = w_1 * y_0^T + b_1$$
$$y2 = w_2 * y_1^T + b_2$$

Visualization of the neural network, it has 1 hidden layer with 5 units in each hidden layer: