Homework 4

CS 5785 Applied Machine Learning

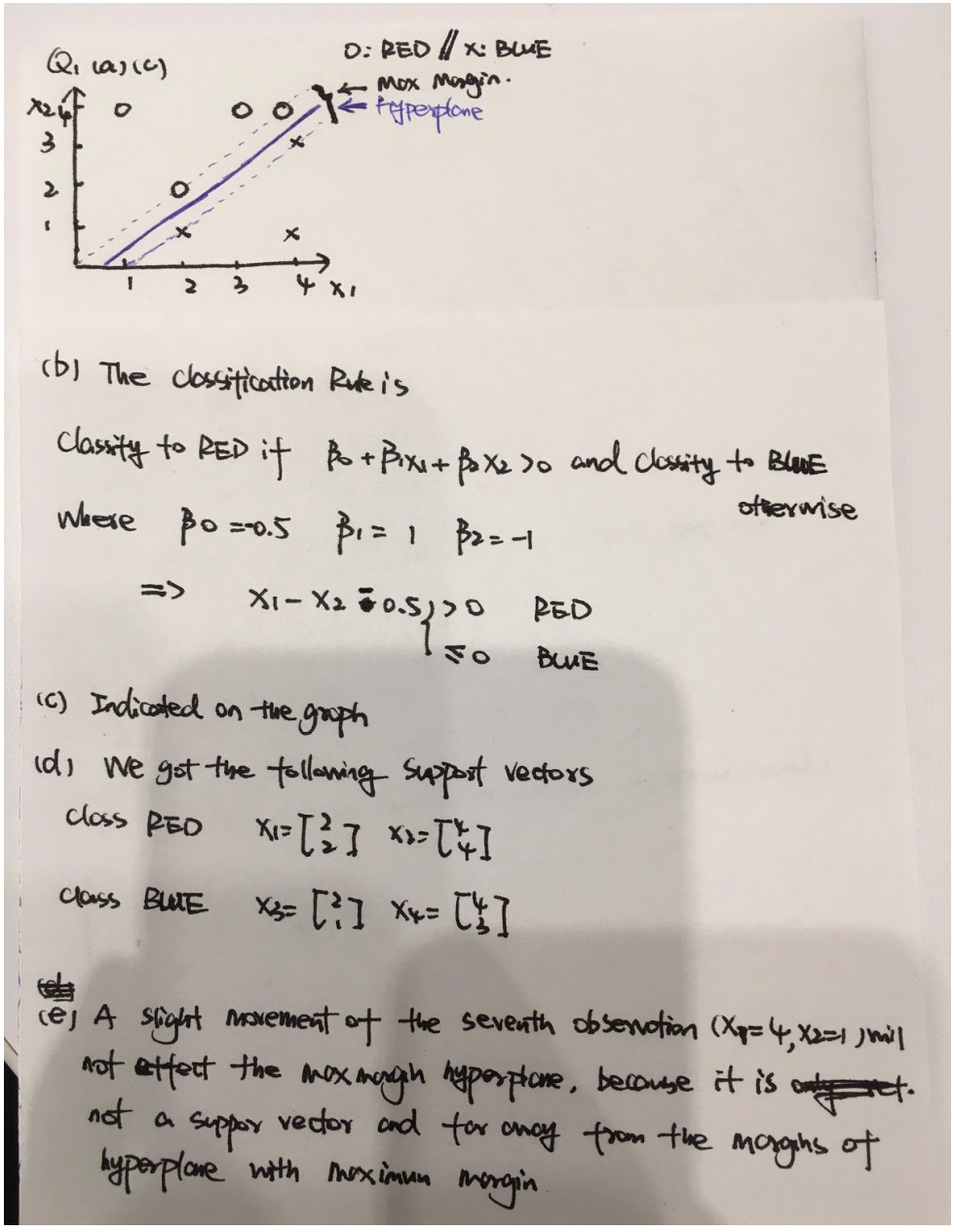
Xialin Shen <xs293@cornell.edu>

An Le <aql6@cornell.edu>

24th November, 2017

PART A - WRITTEN EXERCISE

Question 1:



(b) The classification Rule is

Classify to RED if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$ and classify to BLUE otherwise

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

$\Rightarrow$ $X_1 - X_2 - 0.5 \begin{cases} > 0 & \text{RED} \\ \leq 0 & \text{BLUE} \end{cases}$

(c) Indicated on the graph

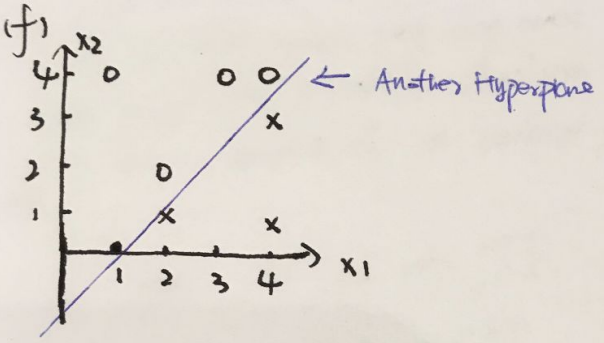
(d) We got the following support vectors

class RED $X_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ $X_3 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$

class BLUE $X_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $X_4 = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$

(e) A slight movement of the seventh observation ($X_7 = 4, X_2 = 1$) will not effect the max margin hyperplane, because it is not a support vector and far away from the margins of hyperplane with maximum margin.

(f)

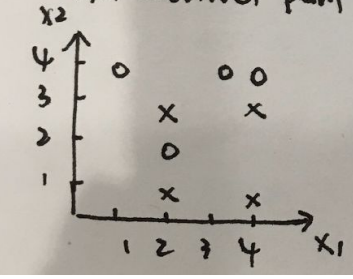


We can draw the following hyperplane who is not the max-margin seperating hyperplane.

$$1.2X_1 - X_2 - 1.4 \begin{cases} > 0 & \text{RED} \\ \leq 0 & \text{BLUE} \end{cases}$$

$\beta_0 = -1.4 \quad \beta_1 = 1.2 \quad \beta_3 = -1$

(g) An additional point can be $(X_1 = 2 \quad X_2 = 3)$ ~~BLUE~~ class.
BLUE



Question 2:

Neural networks as function approximators. Design a feed-forward neural network to approximate the 1-dimensional function given in Fig. 1. The output should match exactly. How many hidden layers do you need? How many units are there within each layer? Show the hidden layers, units, connections, weights, and biases.

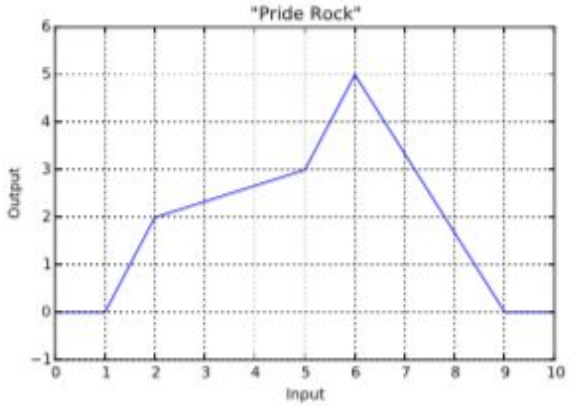


Figure 1: Example function to approximate using a neural network.

Use the ReLU nonlinearity for every unit. Every possible path from input to output must pass through the same number of layers. This means each layer should have the form

$$Y_i = \sigma(W_i Y_{i-1}^T + \beta_i), \tag{1}$$

where $Y_i \in \mathbb{R}^{d_i \times 1}$ is the output of the ith layer, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ is the weight matrix for that layer, $Y_0 = x \in \mathbb{R}^{1 \times 1}$, and the ReLU nonlinearity is defined as

$$\sigma(x) \triangleq \begin{cases} x & x \geq 0, \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

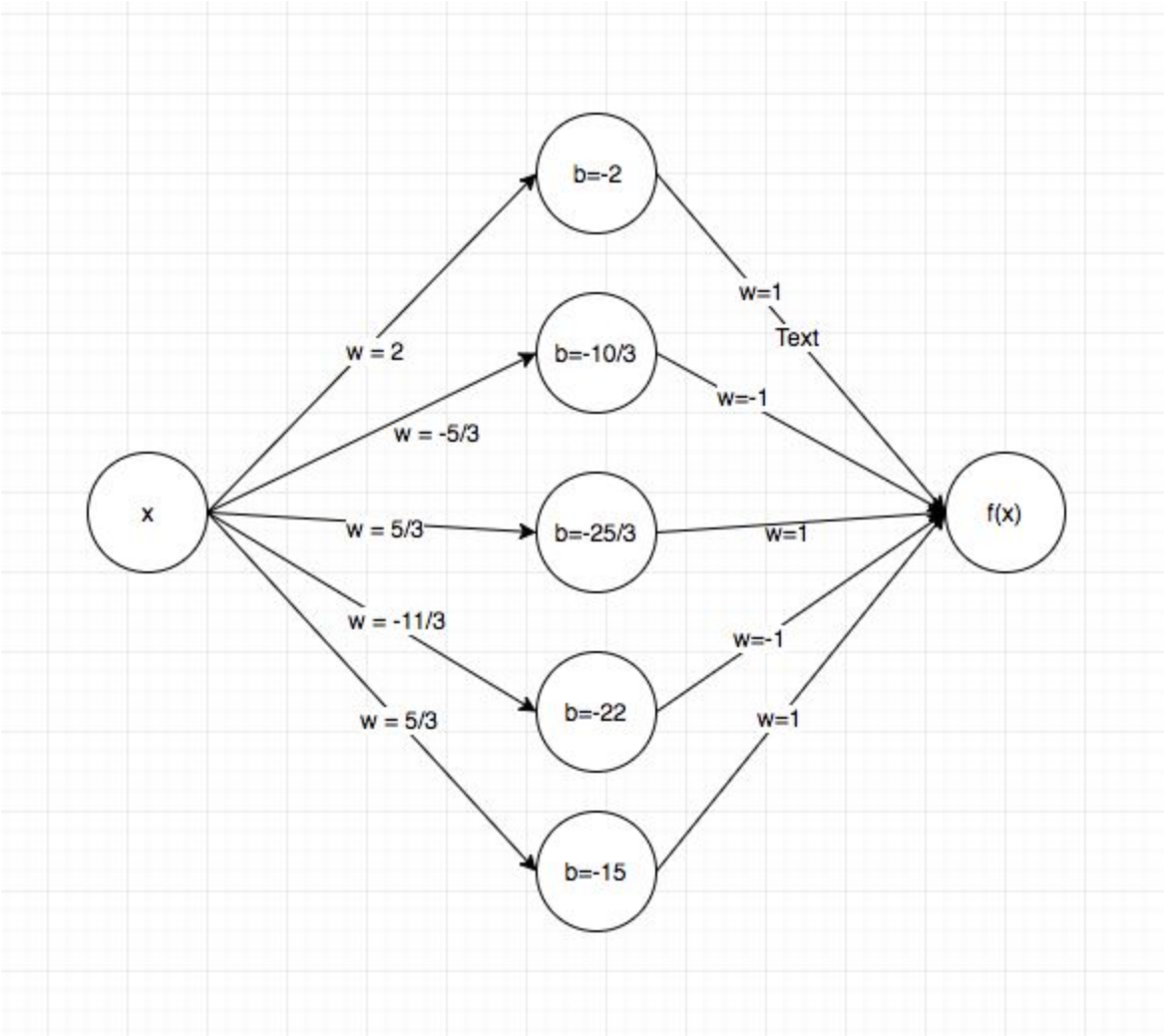
Answer:

We will express the input function as a sum of weighted and scaled ReLU units which have the following form: $\sigma(wx+b) = \sum_{i=1}^{n} w_i x + \sum_{i=1}^{n} b_i = 0$ **(1)**

There are 6 units from the given functions represented by 6 segmented lines:

1. $x \in [0,1]$. Substitute x to (1)
   $\sum_{i=1}^{1} w_i = 0$, $\sum_{i=1}^{1} b_i = 0 \Rightarrow w_1 = 0,\ b_1 = 0$
2. $x \in [1,2]$. Substitute x to (1)
   $\sum_{i=1}^{2} w_i = 2$, $\sum_{i=1}^{2} b_i = -2 \Rightarrow w_2 = 2,\ b_2 = -2$
3. $x \in [2,5]$. Substitute x to (1)
   $\sum_{i=1}^{3} w_i = \frac{1}{3}$, $\sum_{i=1}^{3} b_i = \frac{4}{3} \Rightarrow w_3 = \frac{1}{3} - 2 = \frac{-5}{3},\ b_3 = \frac{4}{3} + 2 = \frac{10}{3}$
4. $x \in [5,6]$. Substitute x to (1)
   $\sum_{i=1}^{4} w_i = 2$, $\sum_{i=1}^{4} b_i = -7 \Rightarrow w_4 = 2 - \frac{1}{3} = \frac{5}{3},\ b_4 = -7 - \frac{4}{3} = -\frac{25}{3}$
5. $x \in [6,9]$. Substitute x to (1)
   $\sum_{i=1}^{5} w_i = \frac{-5}{3}$, $\sum_{i=1}^{5} b_i = 15 \Rightarrow w_5 = -\frac{5}{3} - 2 = \frac{-11}{3},\ b_5 = 15 + 7 = 22$
6. $x \in [9,10]$. Substitute x to (1)
   $\sum_{i=1}^{6} w_i = 0$, $\sum_{i=1}^{6} b_i = 0 \Rightarrow w_6 = 0 + \frac{5}{3} = \frac{5}{3},\ b_6 = 0 - 15 = -15$

Since the first unit has weight and bias equal to 0, we can say that there is 1 hidden layer containing 5 neural units that are computed from 2-6 as shown in this Neural Network.


b=-2
w=1
Text
w = 2
b=-10/3
w=-1
w = -5/3
x
f(x)
w = 5/3
b=-25/3
w=1
w = -11/3
w=-1
w = 5/3
b=-22
w=1
b=-15

PART B - PROGRAMMING EXERCISE

Question 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?

Answer:

There are 9 layers in total:

(i) 1 input layer: receives input (x,y) which is position on a grid.

(ii) 7 fully connected hidden layers: have 20 neurons per layer with relu activation which includes non-linearity and accelerates gradient descent.

(iii) 1 output layer: indicates the network uses a regression layer which predicts the color at the given input point by outputting (r,g,b).

(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.

Answer:

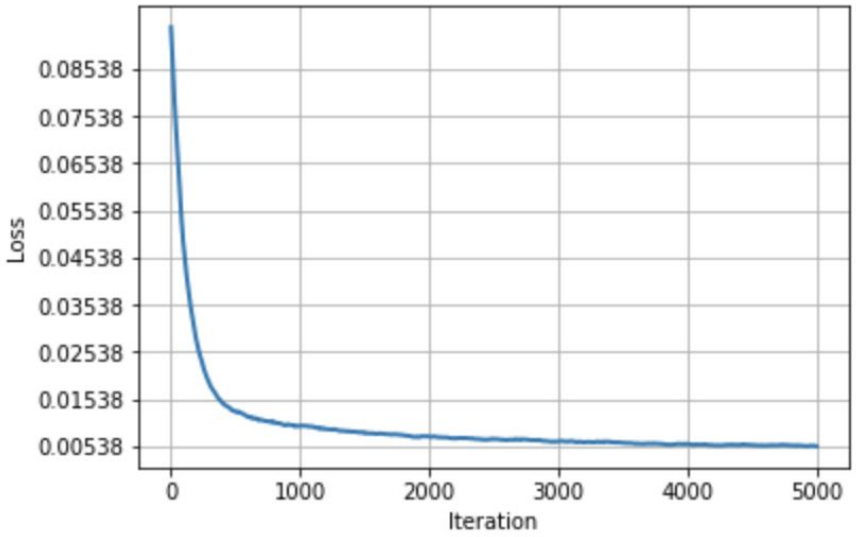
(i) Loss is a cost function. Actual cost function that is used is Regression L2 loss function.
In convnet_layers_loss.js, Loss is computed by smooth_loss = 0.9 * smooth_loss + 0.1 * loss, where loss refers to square errors:

```
x.dw[i] = dy;
loss += 0.5*dy*dy;
```

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

Answer:

The network eventually converged to 0.005 loss.
The collected data is attached to our submission.

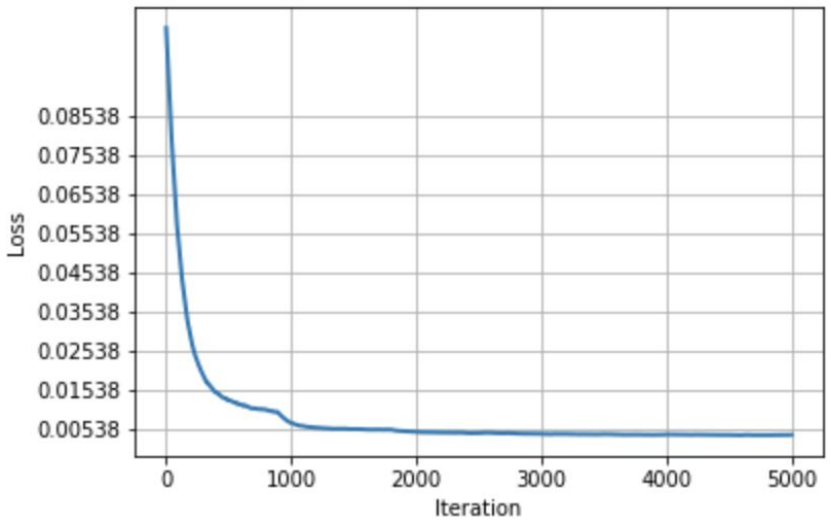


We use the following js function to store data from the web app:

```
function getLoss(){

        var start = counter;

        var arr = [];


        var i = setInterval(function(){

                console.log(smooth_loss);

                arr.push(smooth_loss);

                if(arr.length == 5000){

                        clearInterval(i);

                }

        }, 0.5);

        return arr;

}
```

(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?)

Answer:



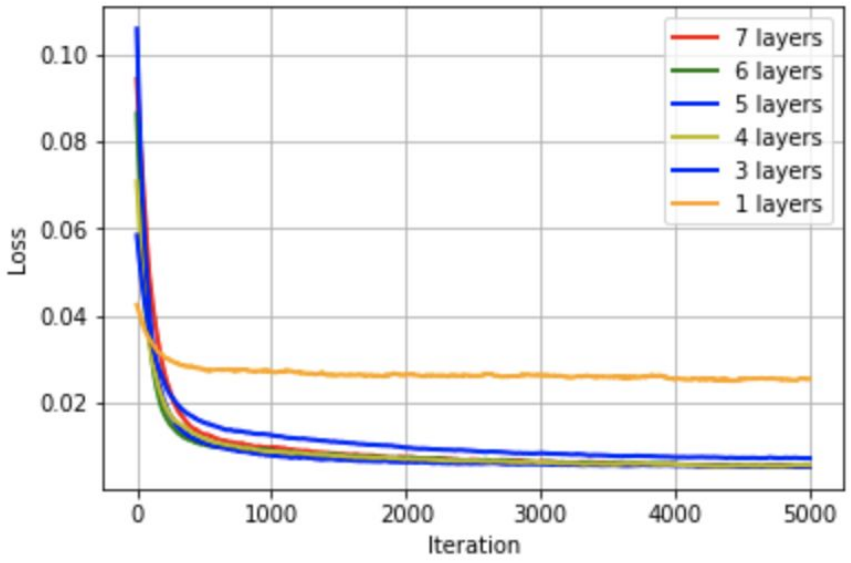
We divided the current learning rate which is defaulted to 0.01 by 5 and so reduced by 20% per 1000 iteration.
The square error loss output converged faster as we can see the slope at some points headed downward more than the original's and eventually converged to the same loss around 0.005

(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?

Answer:

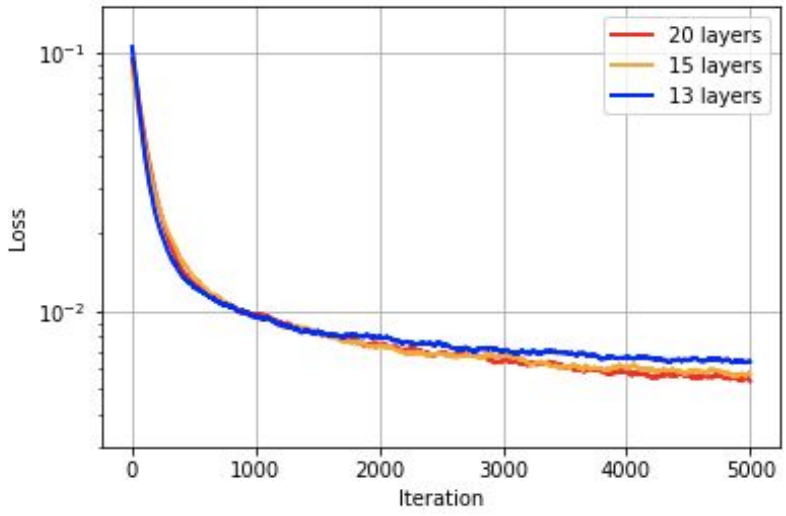
3 layers can be dropped before the loss becomes significant different and greater than 0.006



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

Answer:

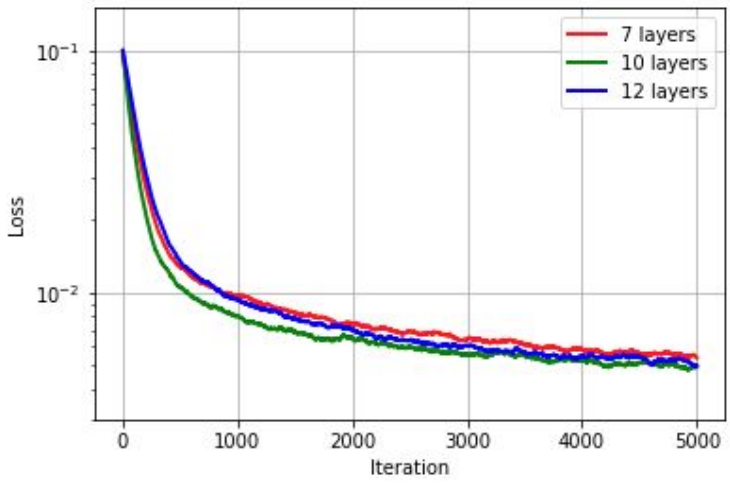
At 15 units in each hidden layer, we can see the loss appears to be becoming significantly different comparing to 20 layers which is greater than 0.006. Based on this limited test, we can say 5 hidden units can be dropped.



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

Answer:

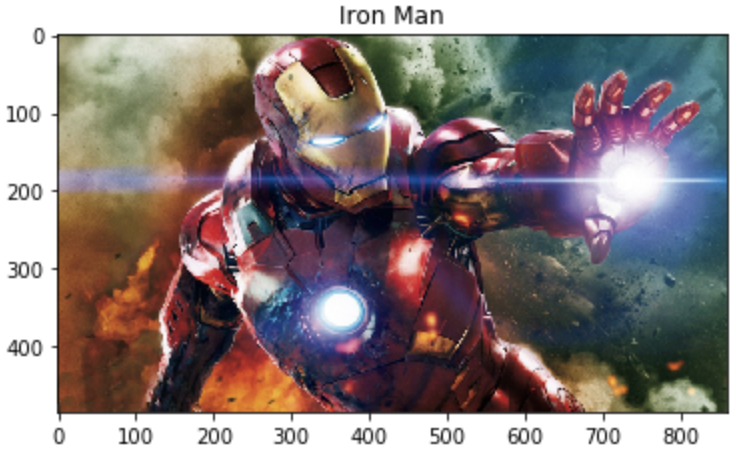
As we can see from the graph below, the accuracy is noticeably improved when we added 3 extra layers represented by the Green line with 10 layers in total.



Question 2 - Random Forest For Image Approximation

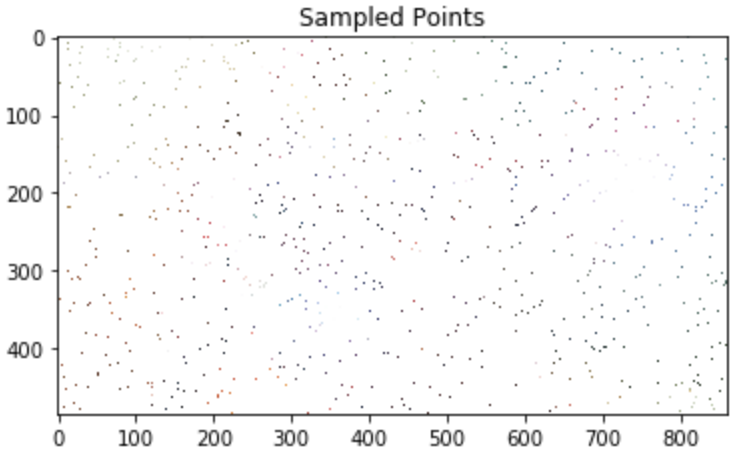
(a) Start with an image of the Mona Lisa. If you don't like the Mona Lisa, pick another interesting image of your choice.

We choose an image of 'Iron man' for experiment



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

There is no need to perform mean subtraction, standardization, or unit-normalization, since the pixel values are already in same unit and standardized. Besides, all these kinds of transformation will not affect the structure of the decision tree.



(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.

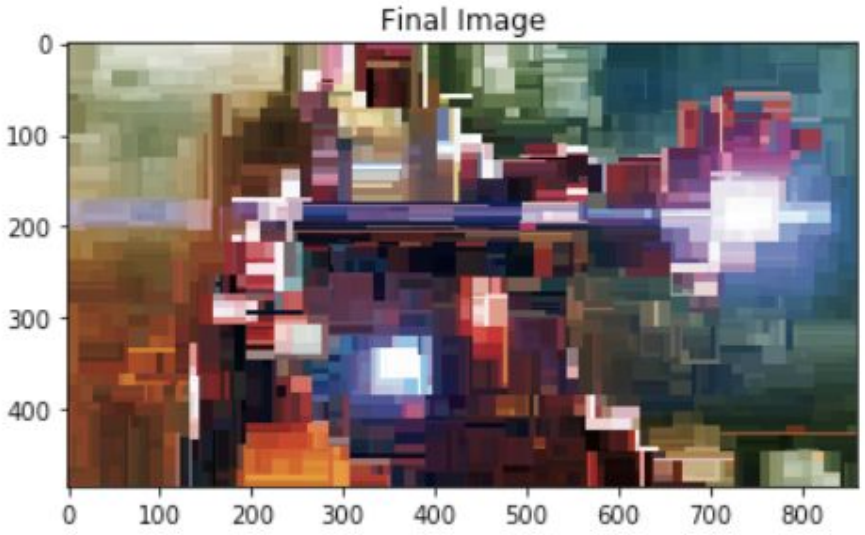
We chose the second option, regressing all three values at once by mapping (x, y) coordinates to (r, g, b) values. Besides, we also rescale the pixel intensities to make them lie between 0.0~1.0.

```
def rescaleSamples(samples):
    rescaledSamples = []
    for index in range(samples.shape[0]):
        rescaledSamples.append(samples[index] / float(255))
    return rescaledSamples
```

(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 predic- tion. You can then use imshow to view the result. (If you are using grayscale, try imshow(Y, cmap='gray') to avoid fake-coloring). You may use any imple- mentation of random forests, but you should understand the implementation and you must cite your sources.

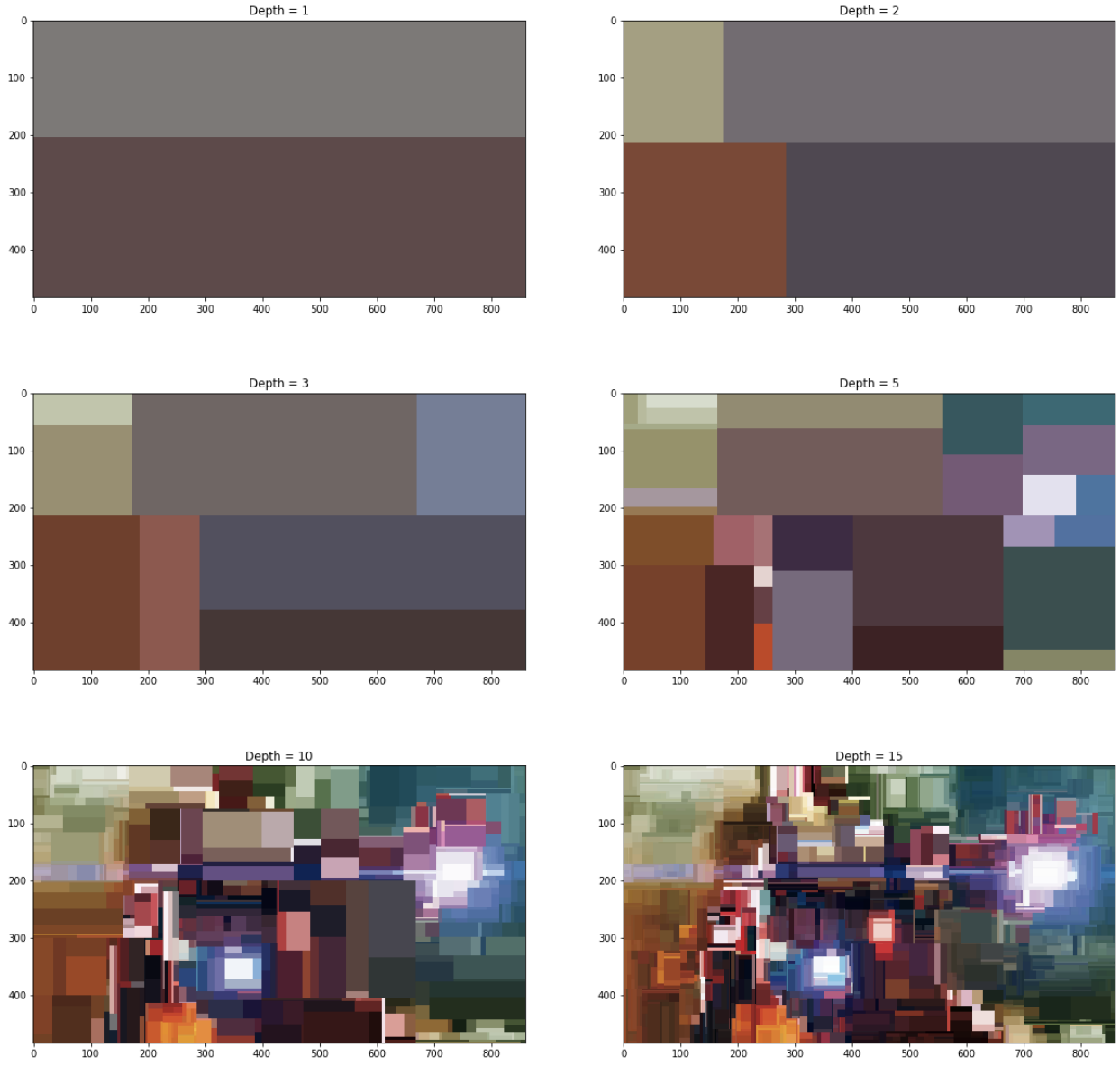
We applied the implementation of random forests provided by Sklearn

Ref: sklearn.ensemble.RandomForestRegressor



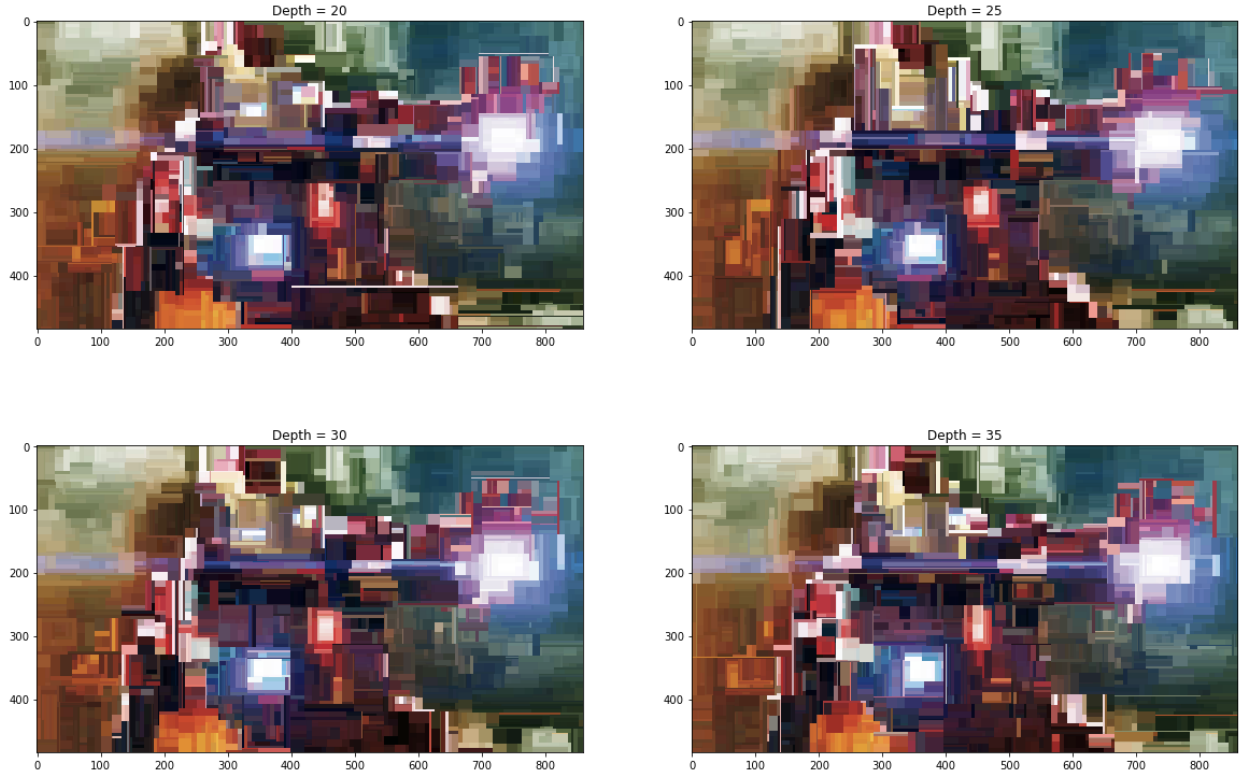
(e) Experimentation

(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.

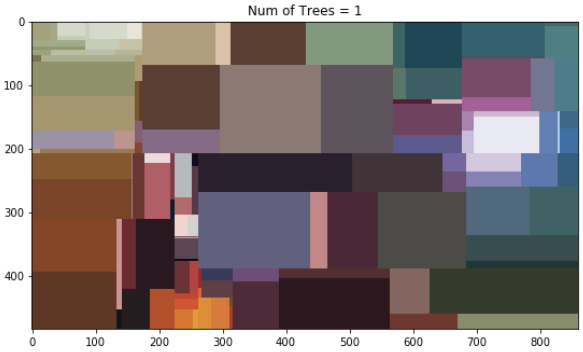


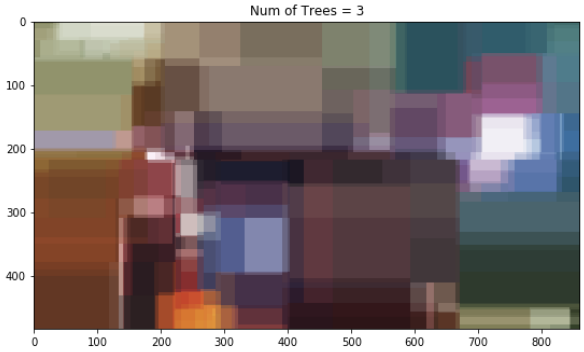
As number of depth increasing, we can get better approximation. When depth is very small, for example depth = 1. The decision tree is actually just binary, as we can observed from the output, the output is divided into two colors. When depth becomes bigger, the tree becomes more complicated and we have more details and more colors on the image.

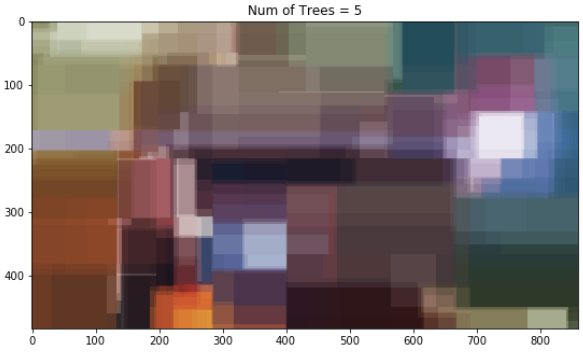
However, we also noticed that the performance can not be better when the depth value is more than 25. It even becomes worse when depth > 30.

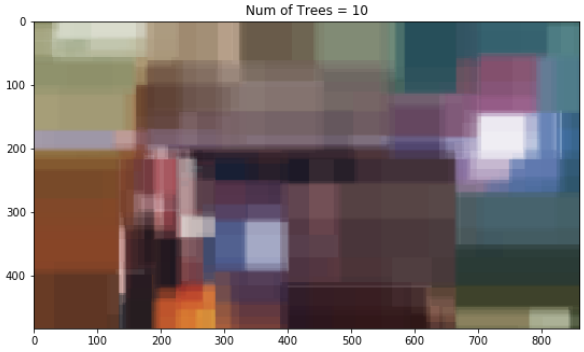


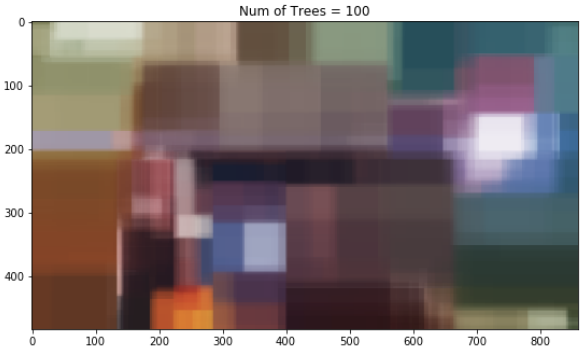
(ii) Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.





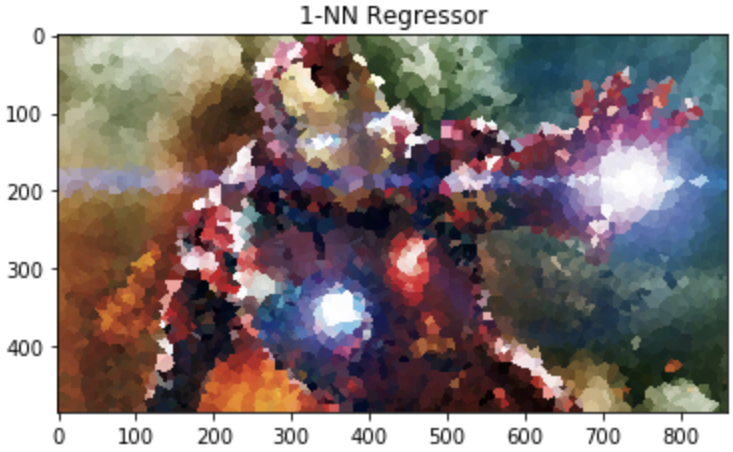






As the number of trees increasing, we have more blurred approximation. The reason why we observed this is because the output of a random forest is derived from averaging all the output of every trees in the forest. In this case, when we have more trees in the forest, we have less variant output, thus the image is more blurring.

(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?

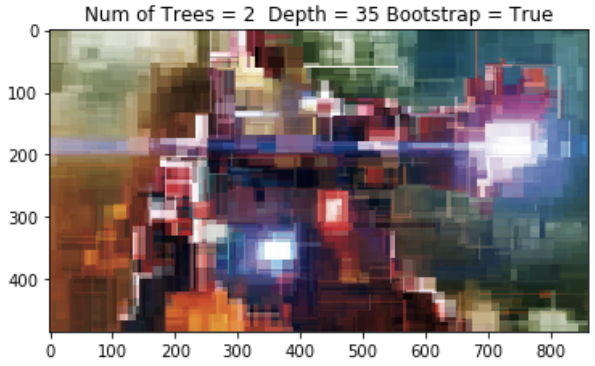


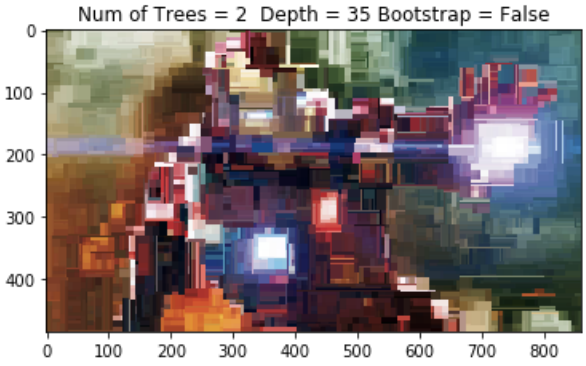
When we applying 1-NN algorithm, the color was predicted by considering its nearest neighbour's color. In this case, many small dots or clusters were created where all the pixels with this area have the same color, which make the image appear in a mosaic style.

(iv) Experiment with different pruning strategies of your choice.

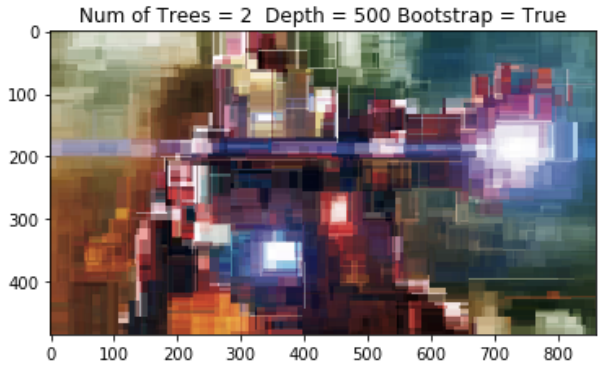
As we noticed in experiment (i), the performance will be even worse when the value of depth is very big. So in this experiment, we want to try to enable the 'Bootstrap' method for random forest regressor.

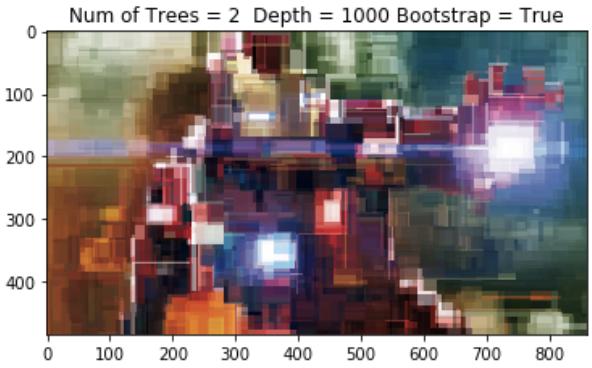
At first we found the performance of using Bootstrap is even worse when we user depth = 35



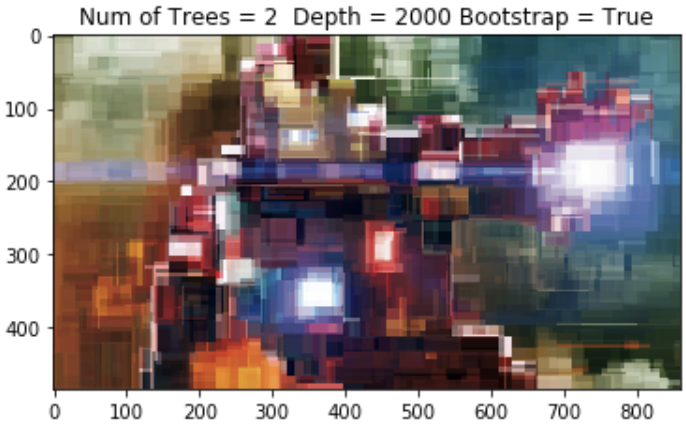


But when we tried to increase depth value, the bootstrap method turned to be better and better.





However, the performance also can not be improved after exceeding an upper limit value of depth. We believe that may due to the fact that we only have 5000 sample points. The performance can be further improved if we can have more sample points.



(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.

Pick a splitting variable j and split point s, which induces a pair of half planes:

$$Go \begin{cases} Left & x_j \geq s \\ Right & x_j < s \end{cases}$$

(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 patches of color are in rectangular shape. The reason is that on each tree node we will split the coordinates into two regions, either on x-axis or y a-axis, since the input is an 2-dimensional data (Pixel Coordinates). In this way, we will keep dividing the whole image into smaller rectangles, which eventually create the patches of color.

(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.

It depends on the depth of the tree. If given the depth is d, the maximum number of patches of color is 2^d

(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.

Assume the prediction is derived by taking average of all the outputs of the trees in the forest. Assume each tree's depth is d.

In this case, each tree has 2^d ***possible outputs. There are total n trees, which mean they can produce*** $(2^d)^n$ ***different sum values.***

So the maximum number of patches of color is 2^{d*n}

PART C - APPENDIX

CODE

Github

REFERENCES

A Solution Manual and Notes for: The Elements of Statistical Learning

https://www.slideshare.net/phvu/kmeans-em-and-mixture-models

https://blog.newey.me/manually-calculating-an-svms-support-vectors/

http://wallpaperwarrior.com/wp-content/uploads/2016/05/Iron-Man-Wallpaper-1.jpg

- END -