UNIVERSITY OF TEXAS AT AUSTIN

NEU 380E Color Prediction

Haoqi Wang (haoqiwang@utexas.edu, ID: hw9335)

May 10, 2020

1 INTRODUCTION

This homework aims at applying neural network to recognize hand-written digits. This report illustrated the basic theory of neural network, including forward propagation, backward propagation, gradient descent, etc. Then, we built a neural network from scratch using sigmoid as activation function and softmax for multiclass classification. After that, we evaluated loss of training and testing data. We also changed the number of units of hidden layer to measure its influence on accuracy. Moreover, we utilized RuLU as another activation function to train neural network. Based on several expriments, we found that around 100 neurons in hidden layer can perform pretty well with accuracy more than 95%. Comparing activation function of sigmoid and ReLU, we noticed that sigmoid behaves better.

2 METHODS

2.1 Import data

We import MNIST data.

2.2 Neural network

2.2.1 Forward propagation

There are basically two equations for forward propagation.

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \tag{2.1}$$

,where $z^{[l]} \in \mathbb{R}^{n^{[l]} \times 1}$, $W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$, $b^{[l]} \in \mathbb{R}^{n^{[l]} \times 1}$, $[l]$ is the index of layer and we have L layers in total, $n^{[l]}$ is the number of units in lth layer. And

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{2.2}$$

, where $g^{[l]}$ is the activation function of lth layer. There are many choices activation functions, including sigmoid funciton

$$g(z) = \frac{1}{1+e^{-z}} \tag{2.3}$$

Rectified Linear Unit (ReLU)

$$g(z) = max(0, z) \tag{2.4}$$

For the last layer, we have the equations

$$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]} \tag{2.5}$$

$$\widehat{y} = a^{[L]} = g^{[L]}(z^{[L]}) \tag{2.6}$$

$$\mathscr{L} = f(\widehat{y}) \tag{2.7}$$

, where f is loss function. Suppose we have m samples in total, the loss function for ith sample can be written as

$$\mathscr{L}^{(i)} = f(a^{[L](i)}) \tag{2.8}$$

and the cost function C is the average of the loss functions of the entire training set.

$$C = \frac{1}{m} \sum_{i=1}^{m} \mathscr{L}^{(i)} \tag{2.9}$$

There are mnay kinds of loss functions, like quadratic loss function

$$\mathscr{L} = \frac{1}{2} \sum_{k=1}^{n^{[L]}} (\widehat{y}_k - y_k)^2 \tag{2.10}$$

and cross entropy loss

$$\mathscr{L} = -\sum_{k=1}^{n^{[L]}} y_k \log \widehat{y}_k \tag{2.11}$$

where y_k is ground truth.

2.2.2 Backward propagation

After computing cost function, we will use backward propagation to update parameters. For each term, equations (2.1) and (2.2) can be written as

$$z_i^{[l]} = \sum_{j}^{n^{[l-1]}} W_{ij}^{[l]} a_j^{[l-1]} + b_i^{[l]} \tag{2.12}$$

$$a_i^{[l]} = g^{[l]}(z_i^{[l]}) \tag{2.13}$$

Apparently, we can have the folowing partial derivatives

$$\begin{cases} \frac{\partial z_i^{[l]}}{\partial W_{ij}^{[l]}} = a_j^{[l-1]} \\ \frac{\partial z_i^{[l]}}{\partial a_j^{[l-1]}} = W_{ij}^{[l]} \\ \frac{\partial z_i^{[l]}}{\partial b_i^{[l]}} = 1 \\ \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} = g^{[l]'}(z_i^{[l]}) \end{cases} \tag{2.14}$$

, which will be utilized in the following derivations.
Here we define

$$\delta_i^{[l]} = \frac{\partial C}{\partial z_i^{[l]}} \tag{2.15}$$

So

$$\delta_i^{[L]} = \frac{\partial\, C}{\partial z_i^{[L]}} = \frac{\partial\, C}{\partial a_i^{[L]}} \frac{\partial a_i^{[L]}}{\partial z_i^{[L]}} = \frac{\partial\, C}{\partial a_i^{[L]}} g^{[L]'}(z_i^{[L]}) \tag{2.16}$$

, which can be vectorized as

$$\delta^{[L]} = \nabla_{a^{[L]}} C \odot g^{[L]'}(z^{[L]}) \tag{2.17}$$

And

$$\begin{aligned} \delta_i^{[l]} &= \frac{\partial C}{\partial z_i^{[l]}} \\ &= \frac{\partial C}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} \\ &= (\sum_{j=1}^{n^{[l+1]}} \frac{\partial C}{\partial z_j^{[l+1]}} \frac{\partial z_j^{[l+1]}}{\partial a_i^{[l]}}) \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} \\ &= (\sum_{j=1}^{n^{[l+1]}} \delta_j^{[l+1]} W_{ji}^{[l+1]}) g^{[l]'}(z_i^{[l]}) \end{aligned} \tag{2.18}$$

, which can be vectorized as

$$\delta^{[l]} = ({W^{[l+1]}}^T \delta^{[l+1]}) \odot g^{[l]'}(z^{[l]}) \tag{2.19}$$

And

$$\begin{aligned} \frac{\partial C}{\partial b_i^{[l]}} &= \frac{\partial C}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial b_i^{[l]}} \\ &= \delta_i^{[l]} \end{aligned} \tag{2.20}$$

, which can be vectorized as

$$\frac{\partial C}{\partial b^{[l]}} = \delta^{[l]} \tag{2.21}$$

And

$$\begin{aligned} \frac{\partial C}{\partial W_{ij}^{[l]}} &= \frac{\partial C}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial W_{ij}^{[l]}} \\ &= \delta_i^{[l]} a_j^{[l-1]} \end{aligned} \tag{2.22}$$

, which can be vectorized as

$$\frac{\partial C}{\partial W^{[l]}} = \delta^{[l]} a^{[l-1]^T} \tag{2.23}$$

(2.17), (2.19), (2.21), (2.23) are 4 basic equations in back propagation. For each iteration, the parameters are updated as

$$W^{[l]} = W^{[l]} - \eta \frac{\partial C}{\partial W^{[l]}} \tag{2.24}$$

$$b^{[l]} = b^{[l]} - \eta \frac{\partial C}{\partial b^{[l]}} \tag{2.25}$$

, where η is learning rate.

2.2.3 Algorithm

The algorithm of training a neural network is the following.

Algorithm 1 Neural network

1. **Initialization**: Initialize $W^{[l]}$, $b^{[l]}$ and activation functions for each layer
2. **Forward propagation**: For $l = 1, 2, ..., L$, apply (2.1) and (2.2) toward (2.9)
3. **Output error**: Compute cost of last layer according to (2.17)
4. **Backpropagate error**: Compute cost for each layer according to (2.19)
5. **Update parameters**: Update $W^{[l]}$, $b^{[l]}$ according to (2.21), (2.23), (2.24), (2.25)

2.3 Gradient descent

Basically there are 3 ways of gradient descent. The equation (2.9) is gradient descent, in which case we add all the loss of the entire training set. This method can give us the optimal move of descent, while it can be time consuming. If we only consider one sample, i.e. $m = 1$, this is stochastic gradient descent (SGD). For batch gradient descent, we consider some samples a time for cost function. Generally we have two ways to judge whether the iteration stops or not, one is setting an iterations limit, another is we setting a threshold in advance and it stops when the cost meets the requirement.

2.4 Softmax and cross entropy

We applied softmax function for multiclass classification, which is

$$\widehat{y}_i = a_i^{[L]} = \frac{e^{z_i^{[L]}}}{\sum_j^{n^{[L]}} e^{z_j^{[L]}}} \tag{2.26}$$

So we have

$$\frac{\partial a_i^{[L]}}{\partial z_j^{[L]}} = a_i^{[L]}(\delta_{ij} - a_j^{[L]}) \tag{2.27}$$

, where δ_{ij} is Kronecker delta. So

$$
\begin{aligned}
\frac{\partial \mathscr{L}}{\partial z_j^{[L]}} &= -\frac{\partial}{\partial z_j^{[L]}} \sum_{i=1}^{n^{[L]}} y_i \log a_i^{[L]} \\
&= -\sum_{i=1}^{n^{[L]}} y_i \frac{\partial \log a_i^{[L]}}{\partial a_i^{[L]}} \frac{\partial a_i^{[L]}}{\partial z_j^{[L]}} \\
&= -\sum_{i=1}^{n^{[L]}} y_i (\delta_{ij} - a_j^{[L]}) \\
&= a_j^{[L]} - y_j
\end{aligned}
\tag{2.28}
$$

So

$$
\frac{\partial \mathscr{L}}{\partial z^{[L]}} = a^{[L]} - y \tag{2.29}
$$

$$
\delta^{[L]} = \frac{\partial C}{\partial z^{[L]}} = \sum_{i=1}^{m} (a^{[L](i)} - y^{(i)}) \tag{2.30}
$$

Combing softmax and cross entropy, the $\delta^{[L]}$ is quite a simple form.

2.5 Prediction

After the neural network is trained, we input test images. The netwok will output the probabilities for each digit and the digit with the highest probability would be the prediction. Then it would be compared with the labels to acquire accuracy of our model.

3 Results

3.1 pre-test

3.1 3.2 3.3 3.4 natural forest recover3.5

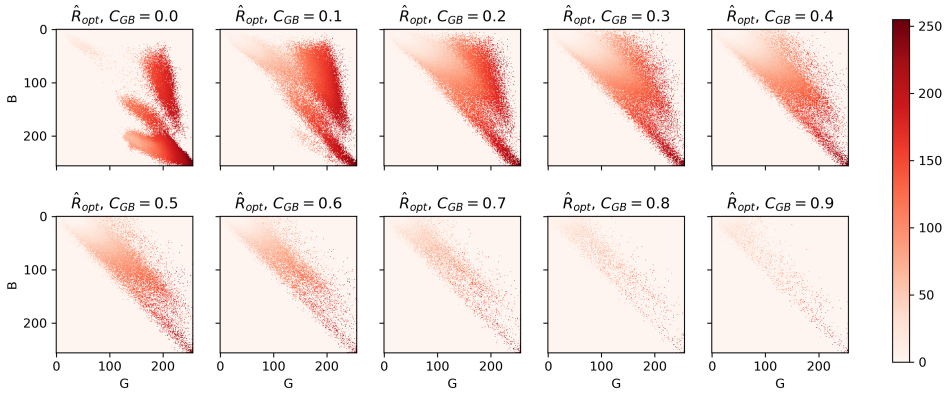


Figure 3.1: natural forest predicted red

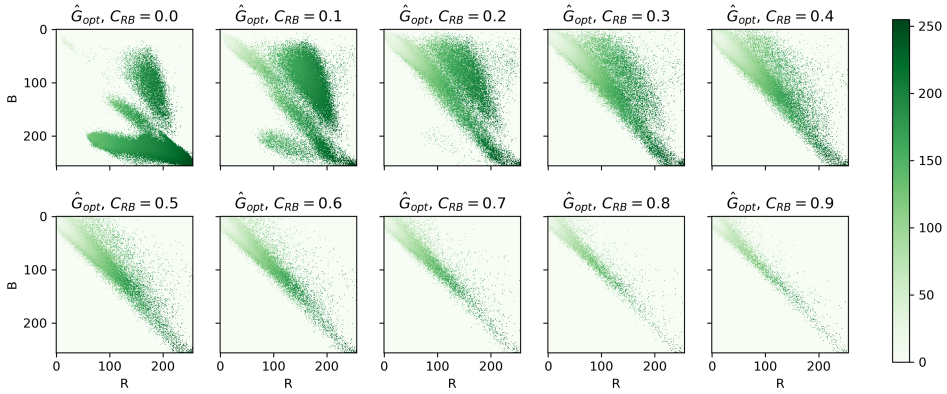


Figure 3.2: natural forest predicted green

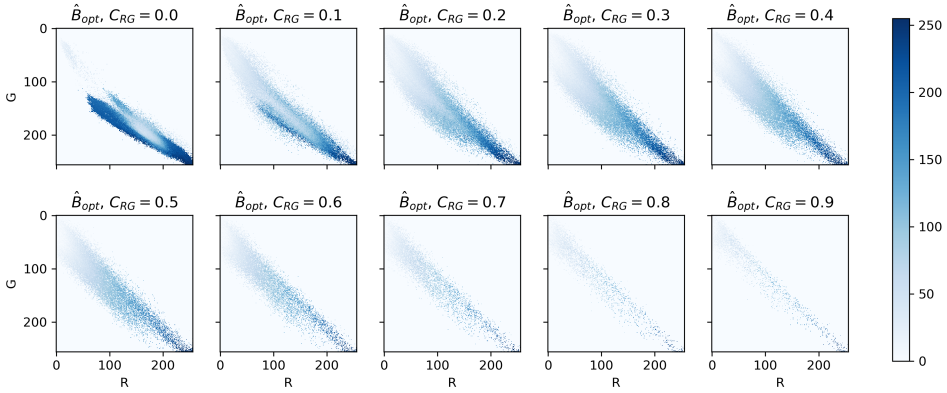
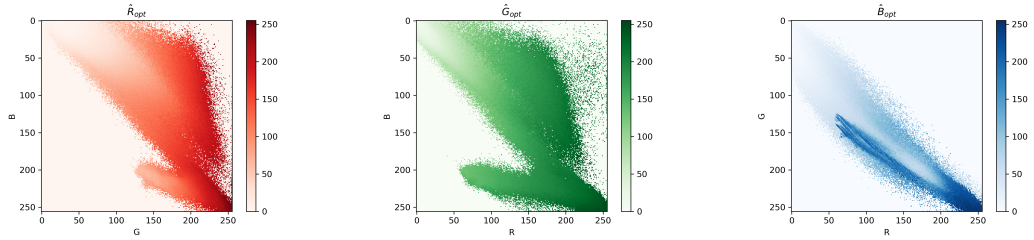


Figure 3.3: natural forest predicted blue



(a) predicted red no contrast (b) predicted green no contrast pre (c) predicted blue no contrast pre

Figure 3.4: predicted color no contrast pre

3.2 Record visualization

Estimation of a missing color channel. Plots show the optimal estimate of the missing color value given the observed color values on the horizontal and vertical axes.
predicted red 3.6 predicted green3.7 predicted blue3.8
predicted colors no contrast3.9 predicted colors no contrast surface plot3.10

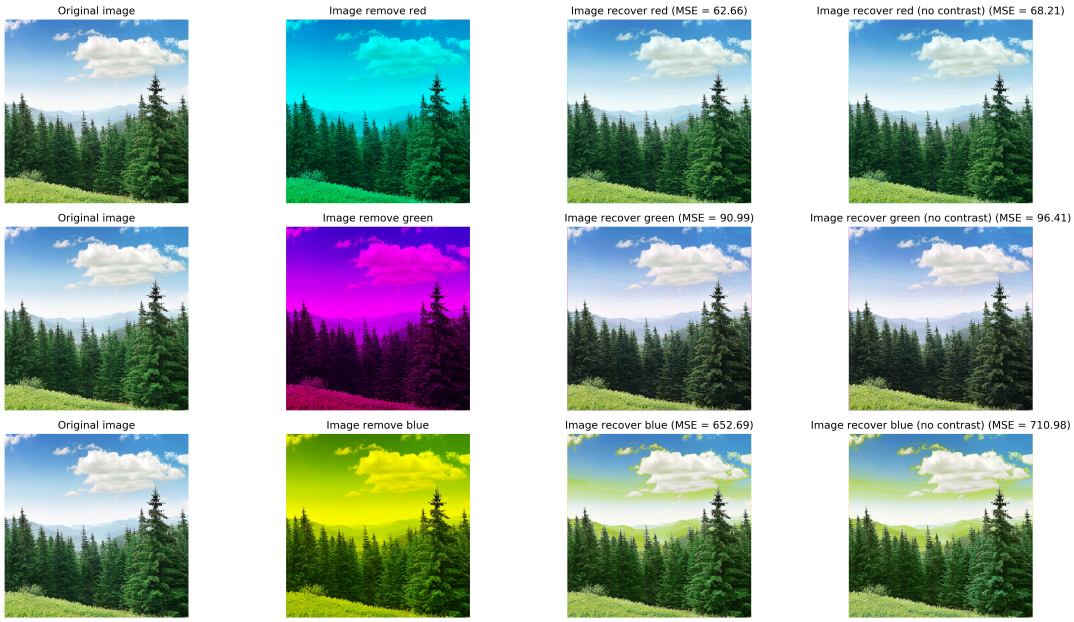
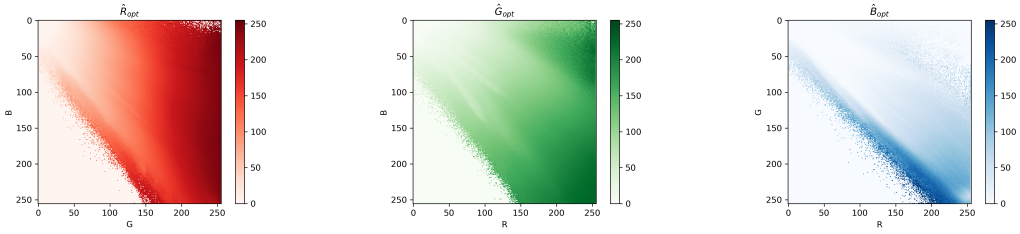
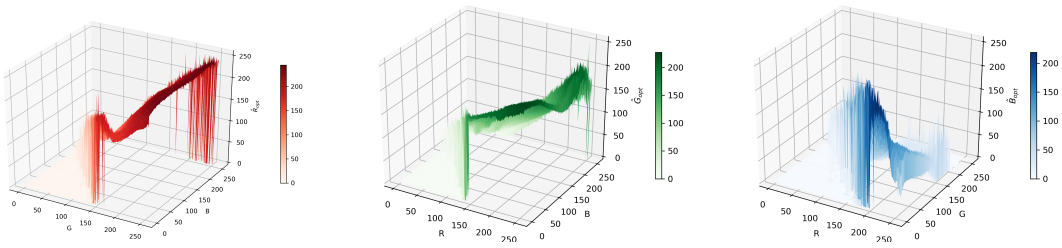


Figure 3.5: natural forest recover



(a) predicted red no contrast (b) predicted green no contrast (c) predicted blue no contrast

Figure 3.9: predicted color no contrast



(a) predicted red no contrast surface plot (b) predicted green no contrast surface plot (c) predicted blue no contrast surface plot

Figure 3.10: predicted color no contrast

3.3 Recover images

cps201004281214 recover 3.11 cps201004291795 recover 3.12

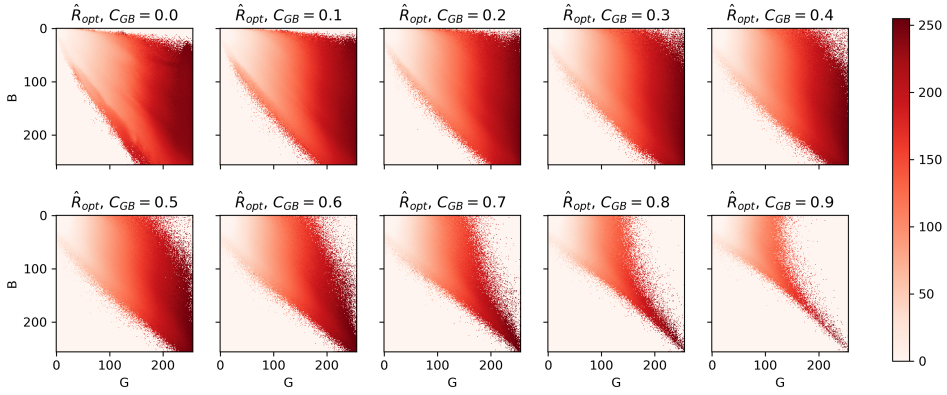


Figure 3.6: predicted red

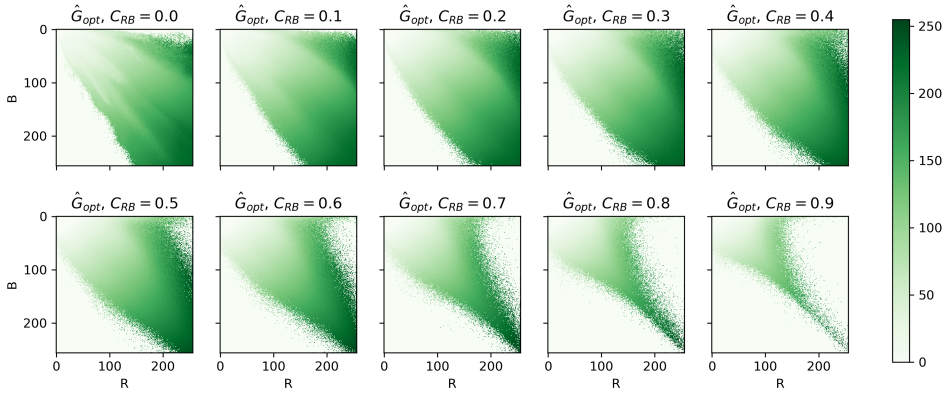


Figure 3.7: predicted green

3.3.1 Experiment 3

In this experiment we set learning rate as 0.01 and the number of units in hidden layer is 100 and smaller initiation of W and b ($W_{ij}, b_i \sim N(0, 0.01)$). The result is shown as **??** and **??**. We can find that the loss converges after several thousands iterations but it fluctuate drastically after that. The accuracy is 92.09%.

3.3.2 Accuracy

We set iteration of 60000 times and learning rate as 0.1. We try different number of units in hidden layer to see its influence on accuracy.
The accuracy is shown as **??**, from which we can find arond 100 neurons performs reletivaly well.

3.3.3 Accuracy of ReLU

Here we built a neural network with one hidden layer and applied SGD and ReLU as activation function. We iterate 60000 times, i.e. go over all training images once. earning rate as 0.1 and $W_{ij}, b_i \sim N(0, 0.1)$. The result of accuracy is shown as **??**. We can find the accuracy is less than that of sigmoid and unstable.

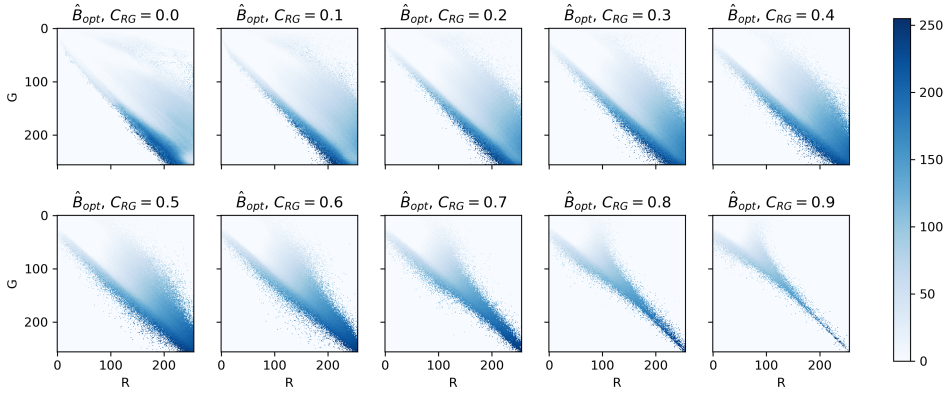


Figure 3.8: predicted blue

4 Conclusion

In this report, we built a neural network with only 1 hidden layer and we can see the accuracy can approach more than 95%. However, there are some problems need to be solved, like the loss obtained from SGD is not stable. What's more, comparing sigmoid and ReLU, we find that sigmoid behaves better than ReLU and the reason is unclear.

NEXT STEP SGD is not stable, I may use gradient descent and batch gradient descent to see if they are more stable. I can try other activation functions and add more hidden layers. In addition, I write this neural network by myself. I should learn some packages like tensorflow, pytorch to build networks.

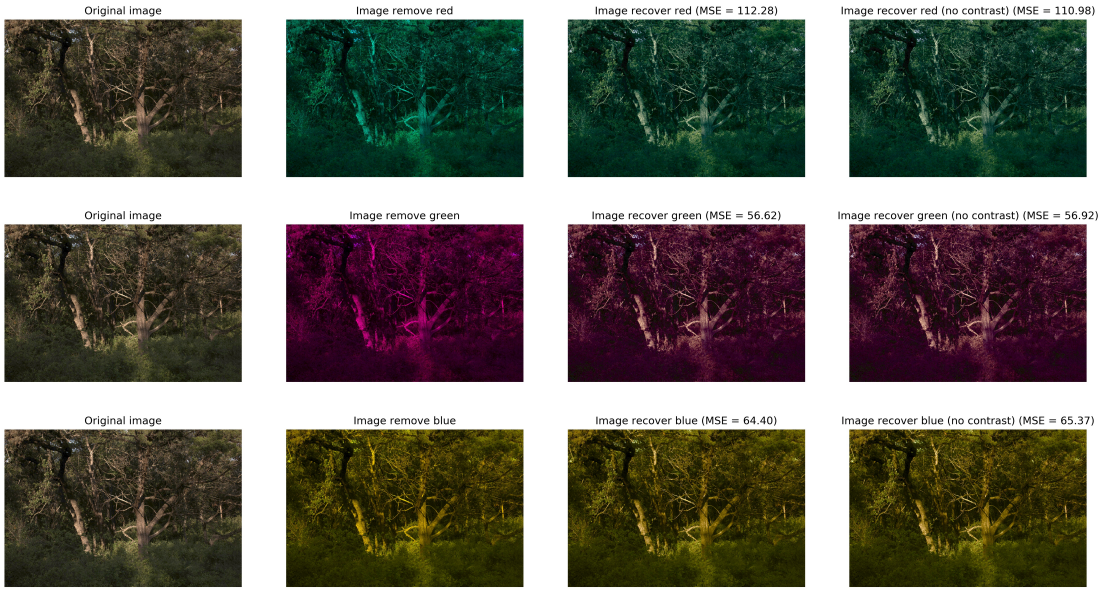


Figure 3.11: cps201004281214 recover

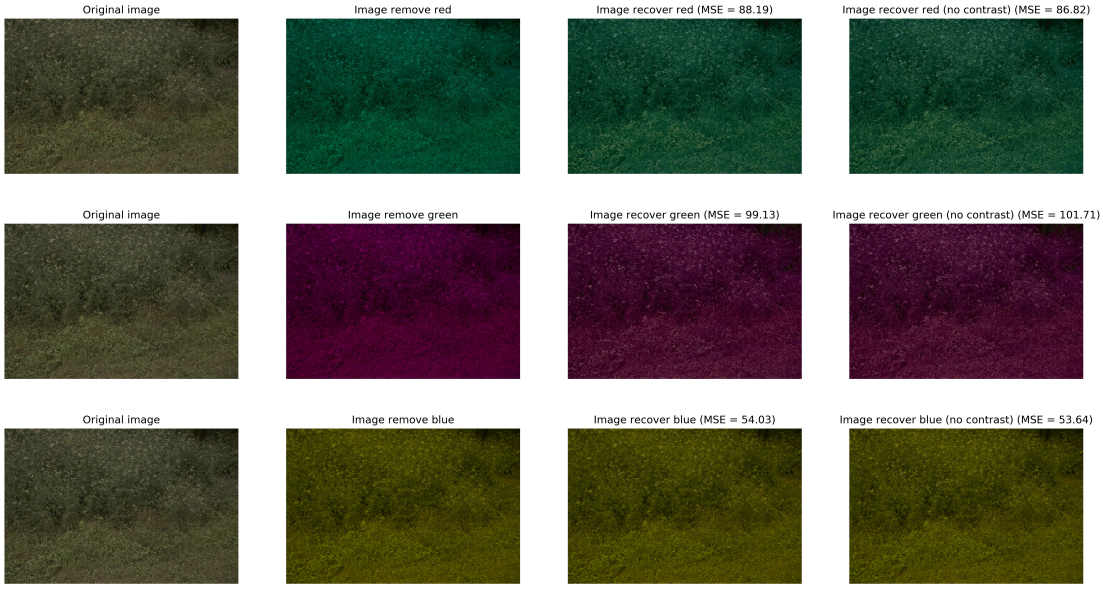


Figure 3.12: cps201004291795 recover