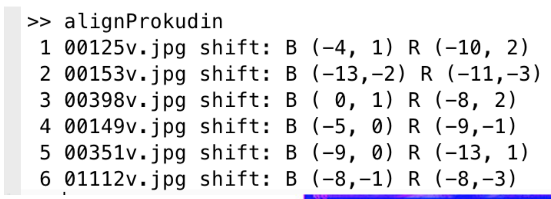
EECS 442 Project 1 Writeup

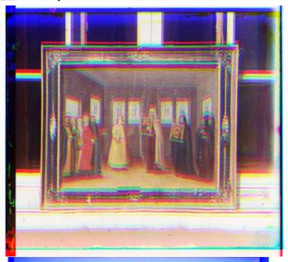
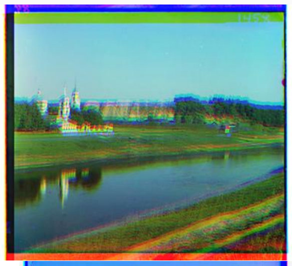
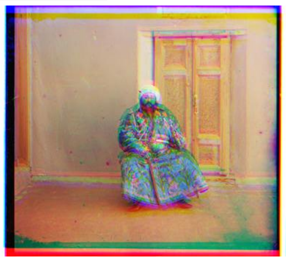
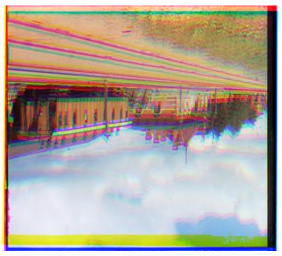
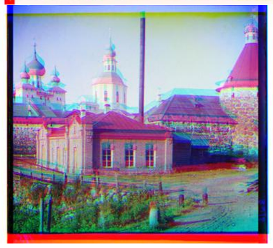
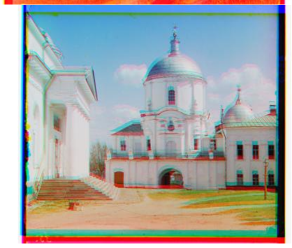
Part 1

I developed the algorithm to align the channels. For the ten colorful pictures, the results are perfect except figure 6(Candy.jpg). I did some analysis to this algorithm.

The basic concept we suppose is that if we turn RGB channels into vectors, the vectors should be paralleled at beginning to get the max dot product. However, in figure 6, I wonder the vectors are not paralleled. In order to verify my suppose, I wrote the cos.m script to test the angles between the vectors of RGB channels. As a result, I find if we turn R and G channels into vectors, the angles between them in candy.jpeg 26°, where the cat.jpg is only 4.8°. That's why my algorithm doesn't work well for candy.jpeg.
Below are my results for Prokudin alignment:



```
>> alignProkudin
 1 00125v.jpg shift: B (-4, 1) R (-10, 2)
 2 00153v.jpg shift: B (-13,-2) R (-11,-3)
 3 00398v.jpg shift: B ( 0, 1) R (-8, 2)
 4 00149v.jpg shift: B (-5, 0) R (-9,-1)
 5 00351v.jpg shift: B (-9, 0) R (-13, 1)
 6 01112v.jpg shift: B (-8,-1) R (-8,-3)
```

As you can see the boundaries perform bad. We only shift the picture by integer. That is just a approximation which leads to the monochromatic boundaries.

Part2

Here is my result for the demosacing: ---

#	image	baseline	nn	linear	adagrad
1	balloon.jpeg	0.179239	0.043685	0.012852	0.011166
2	cat.jpg	0.099966	0.032004	0.013985	0.014203
3	ip.jpg	0.231587	0.123587	0.018290	0.015558
4	puppy.jpg	0.094093	0.019812	0.006277	0.005876

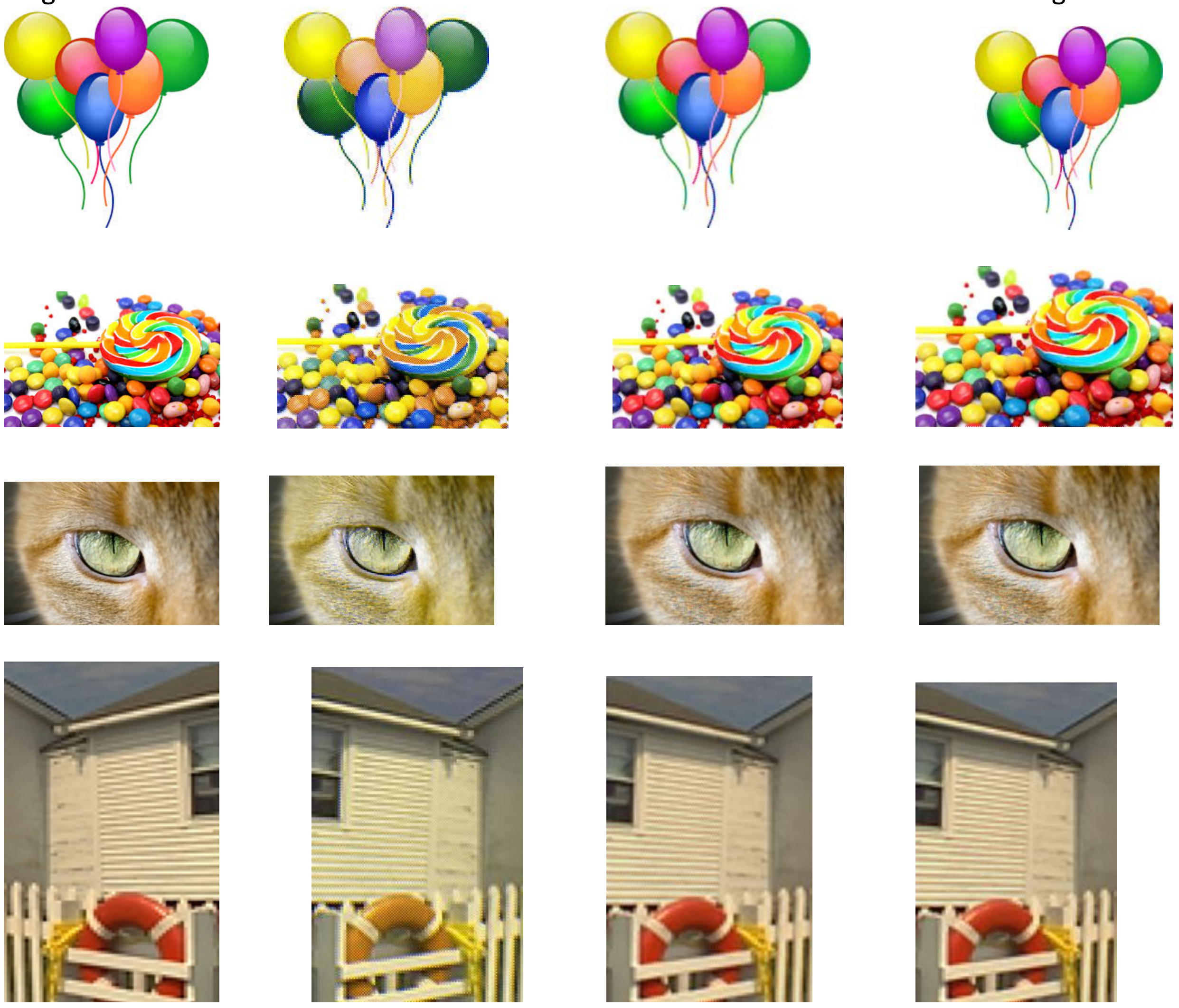
5	squirrel.jpg	0.121964	0.037154	0.023120	0.024095
6	candy.jpeg	0.206359	0.071343	0.023098	0.021503
7	house.png	0.117667	0.037203	0.017312	0.015969
8	light.png	0.097868	0.030218	0.017495	0.016839
9	sails.png	0.074946	0.024708	0.013652	0.012856
10	tree.jpeg	0.167812	0.037438	0.015209	0.012590
	average	0.139150	0.045715	0.016129	0.015066

For nn method, I used the circshift to rotate the matrix and sum them up to make every slot has the nearest value. And update the boundary to make sure that there is no overlapping or covering. However it performs much more bad than the other two algorithm.

For linear and adagrad algorithm, I used loop to set each slot's value.

Below are some sets of pictures

Origin nn linear adagrad



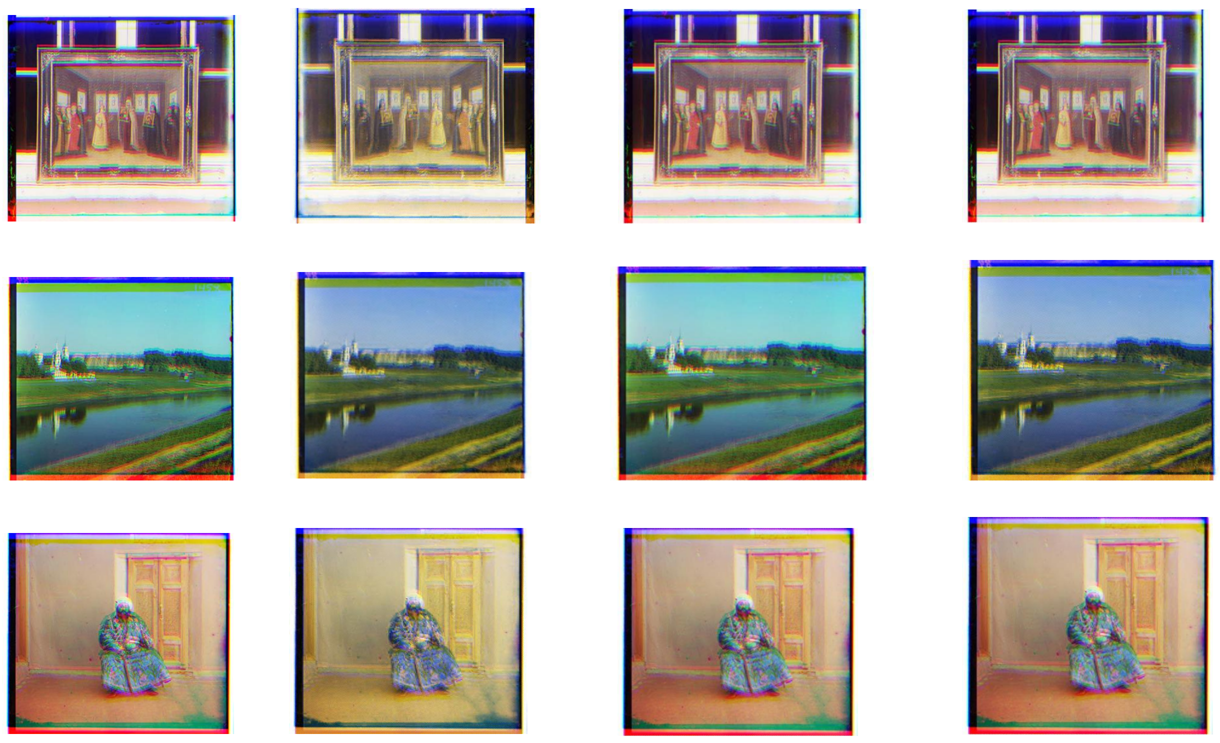
From our raw eyes, the red channel performs bad in nn-method. And nn-method also left a lot of mosaics in the picture where the other algorithms make it more similar with the original photos.

Extra Part:

Digital Prokudin Gorskii:
I wrote the script evalProkudinDemosac.m to finish the extra credit. Here are my result.

```
>> evalProkudinDemosaic
----------------------------------------------------------------------------------------------------
#         image              baseline          nn                  linear    adagrad
----------------------------------------------------------------------------------------------------
1         00125-aligned.jpg       0.155038         0.044777         0.010148         0.009906
2         00153-aligned.jpg       0.109485         0.040287         0.009891         0.009614
3         00398-aligned.jpg       0.142259         0.038732         0.013309         0.012717
4         00149-aligned.jpg       0.169005         0.042305         0.014992         0.013784
5         00351-aligned.jpg       0.157338         0.045718         0.015396         0.015253
6         01112-aligned.jpg       0.121177         0.041385         0.011755         0.011218
----------------------------------------------------------------------------------------------------
          average            0.142384         0.042201         0.012582         0.012082
```

Origin   nn   linear   adagrad



The conclusion is the same as that in part 2. The color saturation of the pictures made by nn-method are bad due to the red channels. For the other two methods, the error rate is extremely small and the distinction can hardly be observed by raw eyes.