Report

Author: Zhengqi Yang
Date: 12/07/2019

Introduction

In this project, we were to tackle image forgery detection problems. In other words, we were trying to find out if an image was modified and where it was modified. In this project, there'll be at most one modified region by copy-pasting with resampling. Our task was to separate the two regions. In order to achieve the best results, we tried to use different filters based algorithms including Color Filter Array (CFA) [1], Error Level Analysis (ELA) [2], and PCA-based Noise Level Estimation (NOI5). Among these approaches, we found that the CFA generated the best mask for most of the test images overall even though there still were one modified image left undetected and one modified image generated with unexpected mask.

Methodology

Concept and Pre-processing:

The CFA based filter works well for this set of test images since this method uses only one threshold to make a decision about the testing image. The main idea of this method is that an imaging manipulation alters CFA demosaicing artifacts. For example, the lack or weak of CFA artifacts might imply the presence of tampering. [1] In the paper, "Image Tamper Detection based on Demosaicing Artifacts", A. Dirik and N. Memon introduced two methods to achieve such tamper detection based on artifacts created by CFA processing. These two methods are CFA pattern number estimation and CFA based noise analysis. [1] In this project, after comparing with the overall results generated by CFA based noise analysis, we finally chose to use the CFA pattern number estimation as the criteria to separate the modified region and the rest. In order to identify the CFA pattern of the test images, each test image was re-interpolated with 4 candidate CFA patterns. The reason we chose only 4 out of 36 potential CFA pattern candidates was that digital cameras in the market generally use one of the 4 bayer CFA arrangements shown in Figure 1 and implementation shown in Figure 2. Since the patterns/filters were chosen, the re-interpolated images would be generated. (See next subsection). Once the re-interpolated images were generated, the MSE between the original image and re-interpolated image was calculated. By observing the MSEs calculated via 4 different patterns/filters, we could conclude that whether or not the test image was tampered. Since for an image that is not tampered, it is expected that one of the MSE values should be significantly smaller than the others because the MSE computed for the actual CFA pattern used for the image re-interpolation should be smaller than the others.

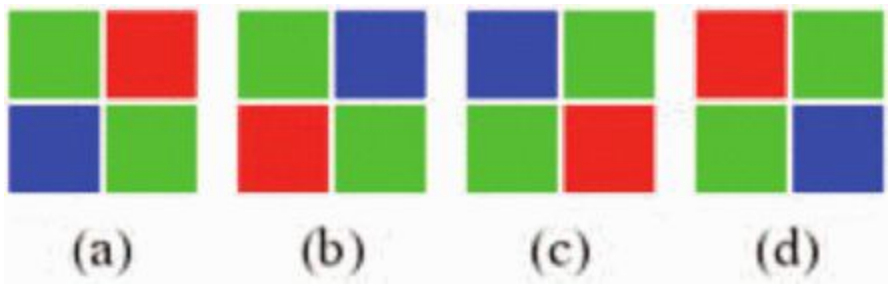

Figure 1: 4 Bayer CFA arrangements.

```
% list of possible CFA arrangements
   CFAList={[2 1;3 2] [2 3;1 2] [3 2;2 1] [1 2;2 3]};
```

Figure 2: Implementation of 4 Bayer CFA arrangements. Note: 1 means Red channel, 2 means Green channel, and 3 means Blue channel.

Re-interpolated Image Computation and MSE:

Once CFA arrangements were set, the re-interpolation with each test image was performed. We now describe the re-interpolation in more details. First of all, each test image was divided into 16✖16 sub-blocks and only the non-smooth blocks were used in the CFA feature computation. Note: In this way, re-interpolation errors in non-smooth blocks were much higher than errors in smooth blocks. The demosaicing algorithm, f, was set to be bilinear. Let each non-smooth block be B_i, where i =1, ... , N, N is the number of non-smooth blocks. Zero-padding was performed in this step. The re-sampled blocks B_i and each of the 4 CFA patterns/filters were computed via convolution and returned $\hat{B}_{ik}$, which consists of the re-interpolated images. The MSE calculated from re-interpolated image equation was shown in Figure 3 and the implementation was shown in Figure 4.

$$E_i(k,c) = \frac{1}{W \times W} \sum_{x=1}^{W} \sum_{y=1}^{w} (B_i(x,y,c) - \hat{B}_{i,k}(x,y,c))^2$$

```
E_{i}(k, c)={1\over W\times
W}\sum_{x=1}^{W}\sum_{y=1}^{w}(B_{i}(x,y,c)-\hat{B}_{i,k}(x,y,c))^{2}
```

This method depends on an estimation procedure for the CFA interpolation pattern

Figure 3: MSE calculation.

```
MeanError=inf(length(CFAList),1);
for TestArray=1:length(CFAList)

    BinFilter=[];
    ProcIm=[];
    CFA=CFAList{TestArray};
    R=CFA==1;
    G=CFA==2;
    B=CFA==3;
    BinFilter(:,:,1)=repmat(R,size(im,1)/2,size(im,2)/2);
    BinFilter(:,:,2)=repmat(G,size(im,1)/2,size(im,2)/2);
    BinFilter(:,:,3)=repmat(B,size(im,1)/2,size(im,2)/2);
    CFAIm=double(im).*BinFilter;
    BilinIm=bilinInterp(CFAIm,BinFilter,CFA);


    ProcIm(:,:,1:3)=im;
    ProcIm(:,:,4:6)=double(BilinIm);

    ProcIm=double(ProcIm);
    BlockResult=blockproc(ProcIm,[W1 W1],@eval_block);

    Stds=BlockResult(:,:,4:6);
    BlockDiffs=BlockResult(:,:,1:3);
    NonSmooth=Stds>StdThresh;

    MeanError(TestArray)=mean(mean(mean(BlockDiffs(NonSmooth))));
    BlockDiffs=BlockDiffs./repmat(sum(BlockDiffs,3),[1 1 3]);

    Diffs(TestArray,:)=reshape(BlockDiffs(:,:,2),1,numel(BlockDiffs(:,:,2)));
    F1Maps{TestArray}=BlockDiffs(:,:,2);
end
```

Figure 4: Implementation of MSE calculation.

Results

Test Image 1: For this test image, we noticed that other methods such as PCA-based noise level estimation generally separated the gate from the rest of the image shown in Figure 5.

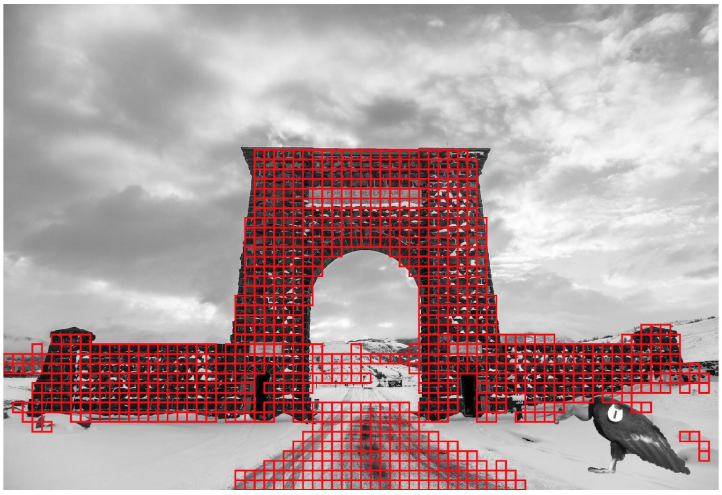

Figure 5: NOI5 Result.

However, the expected separated region should be the bird located at the bottom right corner. The CFA pattern number estimation successfully separated the bird from the rest of the image shown in Figure 6. The reason we thought the CFA pattern number estimation gave us expected result was that since the color/noise of the gate in the original image has the similar weights as the bird does comparing with the white background, preprocessing color filter would lower the weights of such items. Hence, the CFA can separate the bird from the rest of the image.

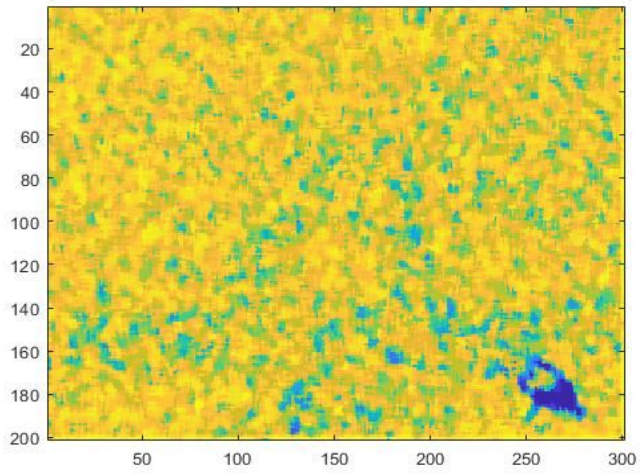

Figure 6: CFA Result.

Test Image 2: As expected, this image was proved with no modification shown in Figure 7.

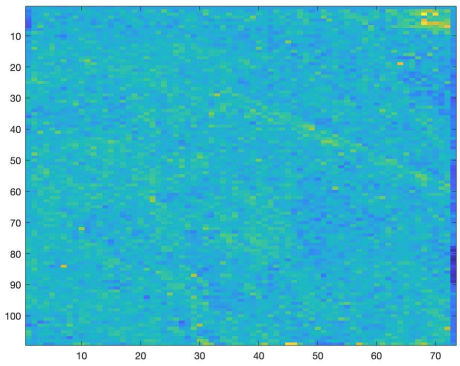

Figure 7: CFA Result.

Test Image 3: The CFA pattern number estimation successfully recognized the manipulation region of the original image, which is the wolf, and the PCA-based noise level estimation also found this modification shown in Figure 8. Comparing the test image with the test image 1, the main difference would be the weights of modification part. Because of this, without color filtering, noise level estimation also achieved the expected result.

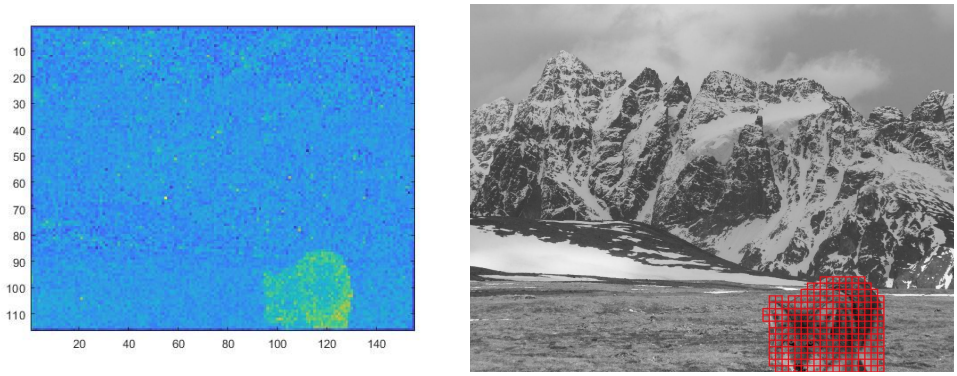

Figure 8: CFA and NOI5 results.

Test Image 4: The modified animal in this image was able to be detected by CFA pattern number estimation as well shown in Figure 9. However, the interesting thing in this result was that the detector also found some noise around the connection between sky and waterfall. By reviewing the test image, we thought this result made sense since there exists very sharp intensity change in this part.

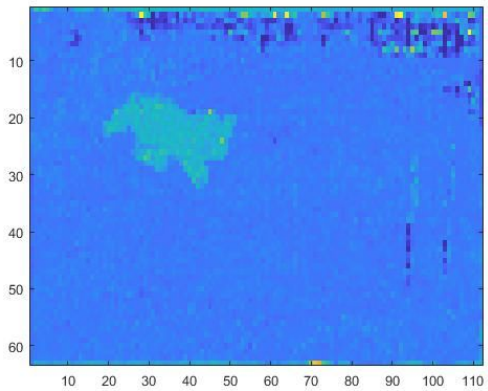

Figure 9: CFA result.

Test Image 5: This modified animal was nicely separated from the rest of the image shown in Figure 10. However, our question was that this image does have the similar feature as the test image 4 does, which is the sky and the land. At this part, we can notice the obvious intensity change but we did not see any noise in the results. The reason this happened was that this test image is much larger than the previous one so that the estimation would be more general. Hence, there does not exist noise at that region.

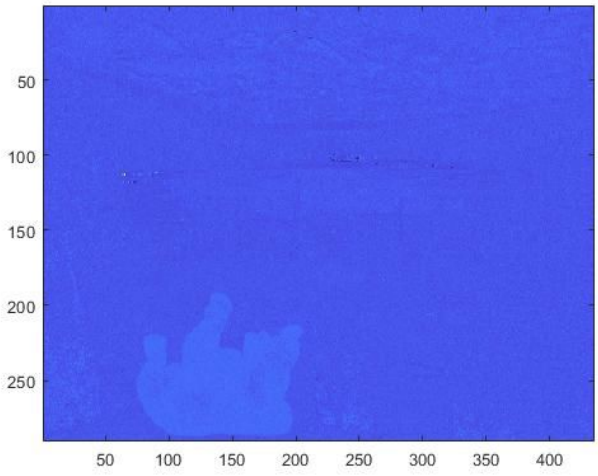

Figure 10: CFA result.

Test Image 6: Unexpected, the image was proved with no modification shown in Figure 11. However, the expected detected modification region should be the bird at bottom left corner. We thought the reason might be this image does not have enough pixels for CFA pattern number estimation. In other words, we can say this estimation was not converged at a good point.

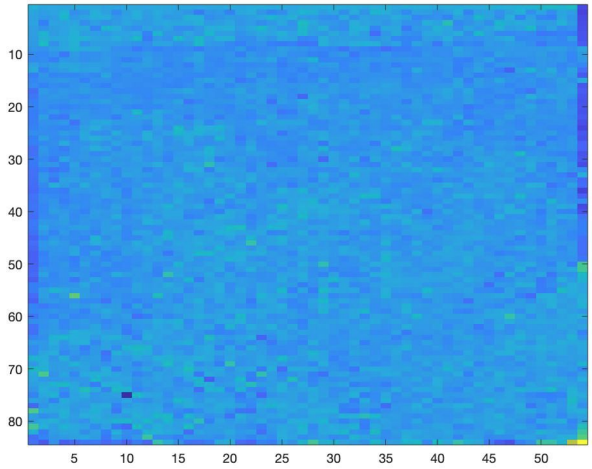

Figure 11: CFA result

Test Image 7: Unexpected, this image was proved to be pristine as well, even though the modified animal was nearly detected shown in Figure 12. By these two failure cases, we conclude that the number of pixels had trivial influence on the performance of CFA pattern number estimation. However, preprocessing with color filters, the intensity changes in the image could play a major role in the performance of the detector since the similarity of both images is the small intensity change.

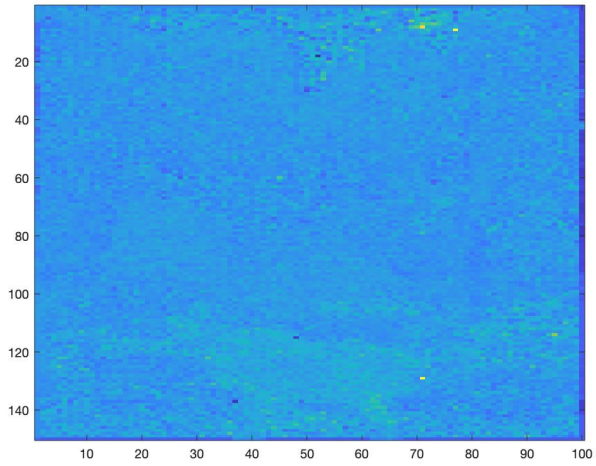

Figure 12: CFA result.

Test Image 8: The modified animal in this image was detected successfully by CFA shown in Figure 13. Following the assumption that we made in the previous part, since this image does have relatively large intensity change, the CFA returned good results.

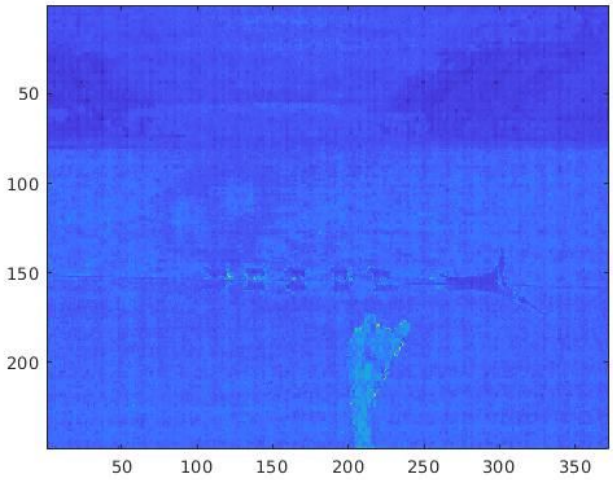

Figure 13: CFA result.

Test Image 9: This image was proved to be pristine shown in Figure 14.

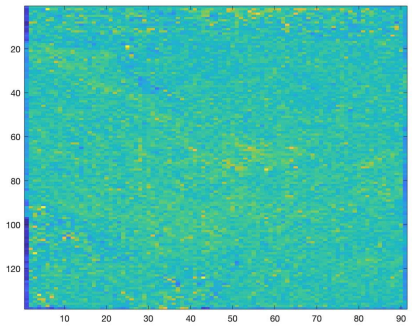

Figure 14: CFA result.

Test Image 10: This image was proved to be pristine shown in Figure 15.

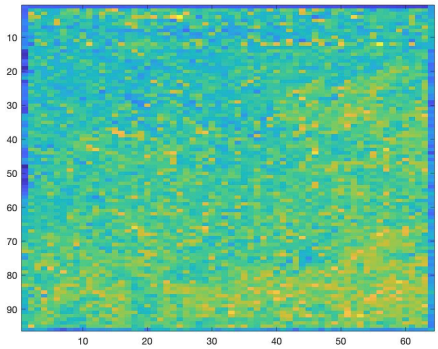

Figure 15: CFA Result.

Reference

[1] A. Dirik and N. Memon, *Image Tamper Detection based Demosaicing Artifacts*, 2009 16th IEEE

[2] http://fotoforensics.com

Appendix

```
close all; clear all;
im1='10.jpg'
[OutputMap, Feature_Vector, coeffArray] = cal(im1);
imagesc(OutputMap);

function [F1Map,CFADetected, F1] = CFATamperDetection_F1(im)
   StdThresh=5;
   Depth=3;

   im=double(im(1:round(floor(end/(2^Depth))*(2^Depth)),1:round(floor(end/(2^Depth))*(2^Depth)),:));

   % list of possible CFA arrangements
   SmallCFAList={[2 1;3 2] [2 3;1 2] [3 2;2 1] [1 2;2 3]};

   CFAList=SmallCFAList;

   %block size
   W1=16;

   if size(im,1)<W1 || size(im,2)<W1
      F1Map=zeros([size(im,1), size(im,2)]);
      CFADetected=[0 0 0 0];
      return
   end

   MeanError=inf(length(CFAList),1);
   for TestArray=1:length(CFAList)

      BinFilter=[];
      ProcIm=[];
      CFA=CFAList{TestArray};
      R=CFA==1;
```

```
        G=CFA==2;
        B=CFA==3;
        BinFilter(:,:,1)=repmat(R,size(im,1)/2,size(im,2)/2);
        BinFilter(:,:,2)=repmat(G,size(im,1)/2,size(im,2)/2);
        BinFilter(:,:,3)=repmat(B,size(im,1)/2,size(im,2)/2);
        CFAIm=double(im).*BinFilter;
        BilinIm=bilinInterp(CFAIm,BinFilter,CFA);


        ProcIm(:,:,1:3)=im;
        ProcIm(:,:,4:6)=double(BilinIm);

        ProcIm=double(ProcIm);
        BlockResult=blockproc(ProcIm,[W1 W1],@eval_block);

        Stds=BlockResult(:,:,4:6);
        BlockDiffs=BlockResult(:,:,1:3);
        NonSmooth=Stds>StdThresh;

        MeanError(TestArray)=mean(mean(mean(BlockDiffs(NonSmooth))));
        BlockDiffs=BlockDiffs./repmat(sum(BlockDiffs,3),[1 1 3]);

        Diffs(TestArray,:)=reshape(BlockDiffs(:,:,2),1,numel(BlockDiffs(:,:,2)));
        F1Maps{TestArray}=BlockDiffs(:,:,2);
    end

    Diffs(isnan(Diffs))=0;

    [~,val]=min(MeanError);
    U=sum(abs(Diffs-0.25),1);
    F1=median(U);
    CFADetected=CFAList{val}==2;
    F1Map=F1Maps{val};

end

function [ Out ] = eval_block( block_struc )
    im=block_struc.data;
    Out(:,:,1)=mean2((double(block_struc.data(:,:,1))-double(block_struc.data(:,:,4))).^2);
    Out(:,:,2)=mean2((double(block_struc.data(:,:,2))-double(block_struc.data(:,:,5))).^2);
    Out(:,:,3)=mean2((double(block_struc.data(:,:,3))-double(block_struc.data(:,:,6))).^2);

    Out(:,:,4)=std(reshape(im(:,:,1),1,numel(im(:,:,1))));
    Out(:,:,5)=std(reshape(im(:,:,2),1,numel(im(:,:,2))));
    Out(:,:,6)=std(reshape(im(:,:,3),1,numel(im(:,:,3))));
end



function [F1Map,CFADetected, F1] = cal( imPath )
    im=CleanUpImage(imPath);
    [F1Map,CFADetected, F1] = CFATamperDetection_F1(im);
end


function [ Out_Im ] = bilinInterp( CFAIm,BinFilter,CFA )
```

```
MaskMin=1/4*[1 2 1;2 4 2;1 2 1];
MaskMaj=1/4*[0 1 0;1 4 1;0 1 0];

if ~isempty(find(diff(CFA)==0)) || ~isempty(find(diff(CFA')==0))
   MaskMaj=MaskMaj.*2;
end

Mask=repmat(MaskMin,[1,1,3]);
[a,Maj]=max(sum(sum(BinFilter)));
Mask(:,:,Maj)=MaskMaj;

Out_Im=zeros(size(CFAIm));

for ii=1:3
   Mixed_im=zeros([size(CFAIm,1),size(CFAIm,2)]);
   Orig_Layer=CFAIm(:,:,ii);
   Interp_Layer=imfilter(Orig_Layer,Mask(:,:,ii));
   Mixed_im(BinFilter(:,:,ii)==0)=Interp_Layer(BinFilter(:,:,ii)==0);
   Mixed_im(BinFilter(:,:,ii)==1)=Orig_Layer(BinFilter(:,:,ii)==1);
   Out_Im(:,:,ii)=Mixed_im;
end

Out_Im=uint8(Out_Im);
```