SHADOW REMOVAL

A PROJECT REPORT

Course:
COMPUTER VISION

Submitted by:
Neelkumar Patel
1101357
Hrishika Vachhani
1111692

Guided by: **Dr. Shan Du**

Lakehead University
Department of Computer Science
Thunder Bay, ON

9th Dec, 2019

Abstract

Most of the time, shadows create disturbance in detection of objects during the understanding of an image process performed by the computer. The critical part of dark shadow, it result in loss of the color values of the surface underlying the shadow. It causes problem in image segmentation, classification or detection in various form. Hence, it becomes necessary to remove the shadow.

The project is about the detection and removal of shadows. The interface allows the user to select and image and performs shadow removal process upon user's request. Various methods were developed and upgraded due to their limitation. The final algorithm performs well compare to the other various of algorithms developed by the group. Python language is utilized for the development and various libraries such as OpenCV, tkinter, numpy etc. The Project GUI is similar to a Windows software.

Contents

List of Figures

Chapter 1

Introduction

Computer vision is all about image understanding. For this, various algorithms related to object detection, segmentation, classification, etc. are needed to be implemented. Shadows present in the images significantly affect the algorithms which may result to misclassification or may results in wrong detections. Shadows are created when an object lies in path of light source. In other words, Obstacle in the way of light causes shadow. It contains loss of information. The surface secured by the shadow district present issues for picture upgrade capacities like picture coordinating, change recognition, etc. There are two types of shadows such as self-shadow and cast shadow. A self-shadow is considered to be when the object itself is shaded. Cast shadows are the actual shadow of an object due to direct light that falls on other surfaces.

1.1 Aim and Objective of Project

The main aim of the project is to remove shadows efficiently without disturbing the entire image appearance. Having less computation and conduct faster execution so that the algorithm is time and memory efficient.

1.2 Problem Specification

Human eye are easily able to observe shadows but to make computer identify shadow is quite difficult. Also, shadow causes object's shape distortion and confusion in boundaries. Thus, it becomes necessary to remove shadow from image as in the pre-processing steps before performing object detection, segmentation or classification.

1.3 H/W and S/W Requirement

The development of this project required the following hardware and software specification. Shadow images are necessary to perform "Shadow Detection and Removal Algorithm".

1.3.1 Hardware Requirements

- Laptop or Computer.

- Basic RAM and Hard disk requirements.

1.3.2 Software Requirements

- Windows XP/Vista/7/10.
- Sublime Text for easy interaction with Web Files.

Chapter 2

Research

2.1 Literature Review

Most of the proposed approaches take into account of the shadow models. The texture-based method of removing shadows was proposed in [1]. It involves two steps: The selection of shadow candidate pixels and secondly, the classification of pixels either as foreground or shadow. This is performed based on the correlation of pixels with the texture. An existing foreground segmentation algorithm uses a particular threshold value to separate the foreground and background. It can also be used for images having stationary object and moving background[2]. But it becomes difficult to separate shadows, this can be accomplished by converting the RGB image to HSV color space and using Otsu's threshold, gradient magnitude and it's direction can be calculated to obtain the differences of pixels between the reference frame and the foreground frame. Later, matting technique is used to remove the shadow [3]. Considering shadows only as darker portions is not adequate as this will result in considering the darker objects as shadows. thus, for accurately detecting shadow, A color to NIR map is additionally computed, which ensures that the candidate are actually shadows rather than objects and then multiplied with shadow candidate map to form a shadow map [4]. Another approach assumes that a non-point light source creates three distinct illuminated areas such as non-lit areas(umbra), partially lit areas(penumbra) and completely lit areas where light falls directly. The penumbra have less intensity than the umbra in terms of darkness. Shadow removal can be performed by lightening umbra and penumbra regions using the shadow probability map [5]. In some research, cast shadow contains two sections: umbra and obscuration. The umbra is made since the immediate light has been totally blocked, while the obscuration is made by fractional hindering of direct light. Artificial Intelligence is used to create decision tree. Choice tree incorporates a root hub, branches and leaf hubs. Criteria checking is performed at inside hubs and the leaf hubs hold the class marks. Choice tree pursues the separation and win procedure with no backtracking. At the point when a basis is checked there are two results yes and no. If there should be an occurrence of indeed the left subtree is produced and again at the hub criteria check is performed and dependent on the result the tree is created. The decision tree continues developing for every paradigm checked until all branches arrive at the leaf hubs. The principle restriction of utilizing a decision tree is that over fitting of information happens and once it is utilized for the total arrangement of information, prompts poor outcomes[7]. Pruning is performed to remove shadows. Directed AI calculations require enormous

examples of preparing datasets for highlight conveyance of classes. So as to conquer the constraints of a decision tree, the creators propose the calculation dependent on random forests. The random forests produces a decision tree however not one. They make numerous decision trees dependent on irregular criteria. Every single tree that is produced will cast a ballot and an official choice is taken dependent on the share of the votes[6].

2.2 Comparison

Figure 2.1 shows the comparison between the oldest method used for shadow removal in 2001 with the latest method used in 2019.

Previous (2001)	Updated (2019)
3 algorithms is used: 1. Statistical non-parametric (SNP) 2. Deterministic non-model based with color exploitation (DNM1) 3. Deterministic non-model based with spatial redundancy exploitation (DNM2)	**2 evaluation indicators are used:** 1. Structural Similarity(SSIM) 2. Peak signal-to-noise ratio(PSNR)
DNM1 method is the most robust to niose.	Rsnet model is more robust algorithm.
The capacity to deal with different shadow size and strength is high in both the SNP and the DNM1. However, the higher flexibility is achieved by the DNM2 algorithm.	The SSIM index values obtained in the RSnet model were superior to the SSIM values of other algorithms, regardless of shadow/output or target/output. The value of the RSnet model was higher than other algorithms, especially in the RSDB database proposed and the PSNR values are higher.
DNM2 seems the more time consuming, due to the amount of processing necessary. On the other hand, the SNP is very fast.	The RSnet model is 1.47 times more stable and the image shadow processing effect is better.

Figure 2.1: Comparison between 2001 and 2019 method

Figure 2.2 shows the comparison of the pre-trained networks used by researches to generate the shadow mask by using shadow and shadow-free images [8], [9] and [10].

Paper	Method	Peak-Signal to Nosie Ratio
Paired Regions for Shadow Detection and Removal	Unary SVM + Pairwise	29.9656
Stacked Conditional Generative Adversarial Networks for Jointly Learning Shadow Detection and Shadow Removal	STacked Conditional Generative Adversarial Network (ST-CGAN)	31.1917
Image Shadow Removal Using End-to-End Deep Convolutional Neural Networks	RSNet	**33.6863**

Figure 2.2: PSNR Comparison of Pre-trained Models

Chapter 3

Design and Methodologies

3.1 Project Design

Most of the time, while performing various processing steps for analyzing images, shadows become the major concern. The project can be useful in avoiding such issues and will also increase the accuracy of the respective algorithm used for image analysis. A window application will be developed where the user can select any image and perform the shadow removal operation. Later, it displays result to the user which can also be saved as a jpeg file.

3.1.1 Technologies

Following we be the technologies used for the development of the project:

1. Python 3.6 or 3.6+
2. Libraries such as OpenCV, numpy, etc.
3. Sublime Text Editor or Jupyter Notebook

3.1.2 Procedure

- User selects an image from the computer.
- Click on a button provided to perform the 'Shadow Removal' operation.
- The algorithm will process the inputed image and tries to remove shadows.
- The result will be displayed to the user in a new window.

3.2 Possible Methodologies

Any algorithm or methodologies should be able to overcome the major following challenges during shadow removal processing:

- Preserve the texture of the shaded region.
- Retain the color information of the surface.

- Make the shadow edges unnoticeable.

Various methods of shadow removal are classified based on five major methods:

1. Reintegration Methods - This method nullifies the image gradient along with the shadow edges and integrate the modified gradient back to the image.
2. Relighting Methods - To find a factor that can be used to enchance the lightness of the shadow pixels
3. Patch-based Methods - This methods try to remove shadows by operating on patches rather than on single pixels.
4. Color transfer Methods - Transfer of color information of lit regions into non-lit region to remove shadows.
5. Interactive Methods - Providing the user a platform to interact with the shadow removal system can lead to improved results.

Chapter 4

Implementation & Results

4.1 System Flow

The project is a window software that allows the user to perform operations based on the button clicked. When user selects "Select an Image", the folder option pops-up and let the user to select a shadow image. When "Remove Shadow" button is clicked, the algorithm detects the shadow first and then removes the shadow.

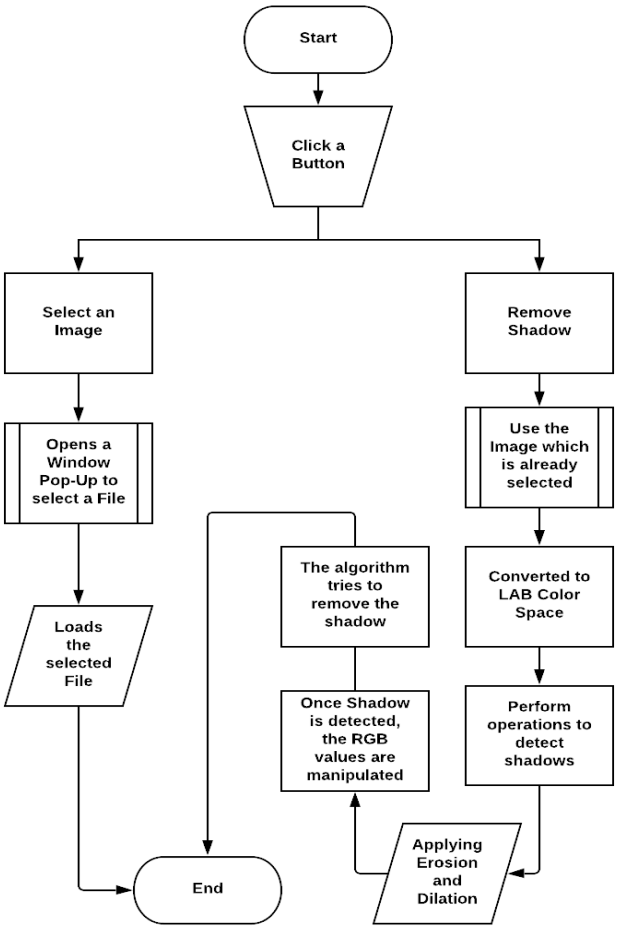

Figure 4.1: Flowchart

4.2 Graphical User Interface

The GUI is important feature to be taken into consideration for any software development. It allows the user to easy interact with the software. Figure 4.2 is a simple software made using python-tkinter library.

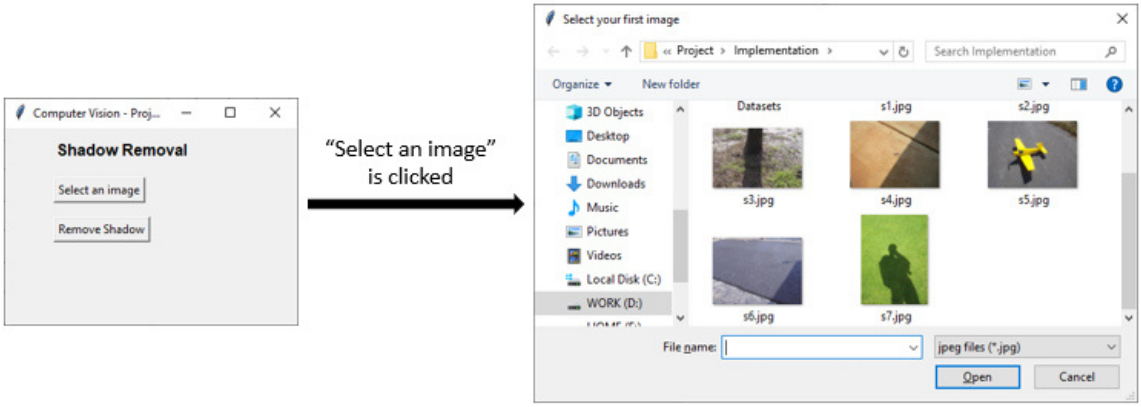

Figure 4.2: Graphical User Interface

4.3 Various Implementations

Our group have performed various methods to remove shadow. Shadow removal is very difficult to perform on JPEG files as the are already in a compressed version. Also, if the shadow is to dark then the information of pixel's color values are lost. At that time, the shadow can only be lighten and the underlying surface cannot be recovered appropriately. Also, the information cannot be traced by LAB or HSV Color Space. Following are some of the implementations performed.

4.3.1 Method 1

```
#Calculating mean of only detected and non-detected area pixels
totalShadowPixels = [0,0,0]
totalNonShadowPixels = [0,0,0]
labShadowPixels = [[],[],[]]
labNonShadowPixels = [[],[],[]]
for i in range(labImageCopy.shape[0]):
   for j in range(labImageCopy.shape[1]):
      for k in range(labImageCopy.shape[2]):
         if mask_lab[i,j] == 0:
            labShadowPixels[k].append(labImage[i,j,k])
            totalShadowPixels[k] +=1
         else:
            labNonShadowPixels[k].append(labImage[i,j,k])
            totalNonShadowPixels[k] +=1

meanLShadowPixels = np.mean(labShadowPixels[0])
meanAShadowPixels = np.mean(labShadowPixels[1])
meanBShadowPixels = np.mean(labShadowPixels[2])
```

```
print("Shadow")
print(meanLShadowPixels,meanAShadowPixels,meanBShadowPixels)
print(totalShadowPixels[0],totalShadowPixels[1],
totalShadowPixels[2])
meanLNonShadowPixels = np.mean(labNonShadowPixels[0])
meanANonShadowPixels = np.mean(labNonShadowPixels[1])
meanBNonShadowPixels = np.mean(labNonShadowPixels[2])
print("Non-Shadow")
print(meanLNonShadowPixels,meanANonShadowPixels,meanBNonShadowPixels)
print(totalNonShadowPixels[0],totalNonShadowPixels[1],totalNonShadowPixels
    ↪ [2])
shadedDiffNonShadedL = abs(meanLShadowPixels - meanLNonShadowPixels)
shadedDiffNonShadedA = abs(meanAShadowPixels - meanANonShadowPixels)
shadedDiffNonShadedB = abs(meanLShadowPixels - meanBNonShadowPixels)
shadedRatioNonShadedL = meanLShadowPixels/meanLNonShadowPixels
shadedRatioNonShadedA = meanAShadowPixels/meanANonShadowPixels
shadedRatioNonShadedB = meanBShadowPixels/meanBNonShadowPixels
print("Difference")
print(shadedDiffNonShadedL,shadedDiffNonShadedA,shadedDiffNonShadedB)
print("Ratio")
print(shadedRatioNonShadedL,shadedRatioNonShadedA,shadedRatioNonShadedB)

cv2.imshow('Before',labImage)
for i in range(labImageCopy.shape[0]):
   for j in range(labImageCopy.shape[1]):
      if mask_lab[i,j] == 0:
         print("Before: {},{},{}".format(labImage[i,j,0],labImage[i,j,1],
             ↪ labImage[i,j,2]))
         labImage[i,j,0]=((meanLNonShadowPixels+labImage[i,j,0])/(
             ↪ meanLNonShadowPixels-labImage[i,j,0]))*labImage[i,j,0]
         labImage[i,j,1]=(shadedRatioNonShadedA*labImage[i,j,1])-(labImage[i
             ↪ ,j,1]-meanANonShadowPixels)
         labImage[i,j,2]=(shadedRatioNonShadedB*labImage[i,j,2])-(labImage[i
             ↪ ,j,2]-meanBNonShadowPixels)
         if labImage[i,j,1] > meanANonShadowPixels:
         labImage[i,j,1]=(shadedRatioNonShadedA*labImage[i,j,1])-(labImage[i
             ↪ ,j,1]-meanANonShadowPixels)
         if labImage[i,j,2] > meanBNonShadowPixels:
            labImage[i,j,2]=(shadedRatioNonShadedB*labImage[i,j,2])-(
                ↪ labImage[i,j,2]-meanBNonShadowPixels)
print("After: {},{},{}".format(labImage[i,j,0],labImage[i,j,1],labImage[i,j
    ↪ ,2]))

cv2.imshow('After',labImage)
finalImage = cv2.cvtColor(labImage,cv2.COLOR_Lab2BGR)
cv2.imshow("Final",finalImage)
```

Initially, we use to perform the removal of shadow using the LAB color space, where L represents the Lightness, A and B are the color components. The mean of Lightness, A and B of all shadow and non-shadow pixels is computed. Using the mean, the ratio of shadow with respect to the non-shadow is calculated and a formula is generated.

For example of A components of image, (shadedRatioNonShadedA*labImage[i,j,1])-(labImage[i,j,1]-meanANonShadowPixels). Similarly, it is calculated for all components of the image at pixel-level. But, this implementation was not able to preserve the color information of the surface underlying the shadow.

4.3.2 Method 2

```
totalShadowPixels = [0,0,0]
totalNonShadowPixels = [0,0,0]
bgrShadowPixels = [[],[],[]]
bgrNonShadowPixels = [[],[],[]]
for i in range(image.shape[0]):
   for j in range(image.shape[1]):
      for k in range(image.shape[2]):
      if mask_lab[i,j] == 0:
         bgrShadowPixels[k].append(image[i,j,k])
         totalShadowPixels[k] +=1
      else:
         bgrNonShadowPixels[k].append(image[i,j,k])
         totalNonShadowPixels[k] +=1

meanBShadowPixels = np.mean(bgrShadowPixels[0])
meanGShadowPixels = np.mean(bgrShadowPixels[1])
meanRShadowPixels = np.mean(bgrShadowPixels[2])
sumMeanShadow = meanBShadowPixels+meanGShadowPixels+meanRShadowPixels
print("Shadow")
print(meanBShadowPixels,meanGShadowPixels,meanRShadowPixels)
print("Addition Mean: {}".format(sumMeanShadow))
meanBNonShadowPixels = np.mean(bgrNonShadowPixels[0])
meanGNonShadowPixels = np.mean(bgrNonShadowPixels[1])
meanRNonShadowPixels = np.mean(bgrNonShadowPixels[2])
sumMeanNonShadow = meanBNonShadowPixels+meanGNonShadowPixels+
    ↪ meanRNonShadowPixels
print("Non-Shadow")
print(meanBNonShadowPixels,meanGNonShadowPixels,meanRNonShadowPixels)
print("Addition Mean: {}".format(sumMeanNonShadow))

newMeanAfterIteration = [[],[],[]]
for i in range(image.shape[0]):
   for j in range(image.shape[1]):
   if mask_lab[i,j] == 0:
      # print("Before: {},{},{}".format(image[i,j,0],image[i,j,1],image[i,j
          ↪ ,2]))
      image[i,j,0] = image[i,j,0] + ((image[i,j,0]*meanBNonShadowPixels)/
          ↪ sumMeanNonShadow)
      image[i,j,1] = image[i,j,1] + ((image[i,j,1]*meanGNonShadowPixels)/
          ↪ sumMeanNonShadow)
      image[i,j,2] = image[i,j,2] + ((image[i,j,2]*meanRNonShadowPixels)/
          ↪ sumMeanNonShadow)
cv2.imshow("Final",image)
```

Based on the limitation of the Method 1, as the color information was lost. We decided to used the LAB color space for shadow detection and RGB color space for the shadow removal. The mean of R, G and B of all shadow and non-shadow pixels is computed. Using the mean, a formula is generated. For example of B components of image, image[i,j,0] = image[i,j,0] + ((image[i,j,0]*meanBNonShadowPixels)/sumMeanNonShadow). Here, the sumMeanNonShadow variable is the sum of the mean of R, G and B values of Non-shadow Pixels. Similarly, it is calculated for all components of the image at pixel-level. This implementation didn't removed the shadow but slightly lightens it along with preserving the color information.

4.3.3 Method 3

```
iterate = 1
while True:
   print("Iteration:",iterate)
   iterate+=1
   totalShadowPixels = [0,0,0]
   totalNonShadowPixels = [0,0,0]
   bgrShadowPixels = [[],[],[]]
   bgrNonShadowPixels = [[],[],[]]
   for i in range(image.shape[0]):
      for j in range(image.shape[1]):
         for k in range(image.shape[2]):
         if mask_lab[i,j] == 0:
            bgrShadowPixels[k].append(image[i,j,k])
            totalShadowPixels[k] +=1
         else:
            bgrNonShadowPixels[k].append(image[i,j,k])
            totalNonShadowPixels[k] +=1

   meanBShadowPixels = np.mean(bgrShadowPixels[0])
   meanGShadowPixels = np.mean(bgrShadowPixels[1])
   meanRShadowPixels = np.mean(bgrShadowPixels[2])
   sumMeanShadow = meanBShadowPixels+meanGShadowPixels+meanRShadowPixels
   print("Shadow")
   print(meanBShadowPixels,meanGShadowPixels,meanRShadowPixels)
   print("Addition Mean: {}".format(sumMeanShadow))
   meanBNonShadowPixels = np.mean(bgrNonShadowPixels[0])
   meanGNonShadowPixels = np.mean(bgrNonShadowPixels[1])
   meanRNonShadowPixels = np.mean(bgrNonShadowPixels[2])
   sumMeanNonShadow = meanBNonShadowPixels+meanGNonShadowPixels+
      ↪ meanRNonShadowPixels
   print("Non-Shadow")
   print(meanBNonShadowPixels,meanGNonShadowPixels,meanRNonShadowPixels)
   print("Addition Mean: {}".format(sumMeanNonShadow))

   newMeanAfterIteration = [[],[],[]]
   for i in range(image.shape[0]):
      for j in range(image.shape[1]):
         if mask_lab[i,j] == 0:
```

```
            # print("Before: {},{},{}".format(image[i,j,0],image[i,j,1],
                ↪ image[i,j,2]))
            image[i,j,0] = image[i,j,0] + ((image[i,j,0]*
                ↪ meanBNonShadowPixels)/sumMeanNonShadow)
            image[i,j,1] = image[i,j,1] + ((image[i,j,1]*
                ↪ meanGNonShadowPixels)/sumMeanNonShadow)
            image[i,j,2] = image[i,j,2] + ((image[i,j,2]*
                ↪ meanRNonShadowPixels)/sumMeanNonShadow)
            newMeanAfterIteration[0].append(image[i,j,0])
            newMeanAfterIteration[1].append(image[i,j,1])
            newMeanAfterIteration[2].append(image[i,j,2])
   # print("After: {},{},{}".format(image[i,j,0],image[i,j,1],image[i,j,2])
       ↪ )
   newMeanAfterIterationAddition = np.mean(newMeanAfterIteration[0])+np.
       ↪ mean(newMeanAfterIteration[1])+np.mean(newMeanAfterIteration[2])
   print(newMeanAfterIterationAddition)
   if newMeanAfterIterationAddition >= (sumMeanNonShadow/2):
      break
cv2.imshow('After',image)
```

In method 2, the shadow was slightly lightened preserving the color information under the shadow. So, in method 3, Our group decided to iterate the whole process of method 2 multiple times to see whether the shadow gets lighter. But, the shadow pixels transformed into Red, Blue and Green patches. The iteration was then performed one by one and it was noticeably that after some iteration only the patches starts to appear. Hence, we came up with a formula of iterating the loop such that the sum of new mean of R, G and B of shaded region is less than the half of the overall mean of non-shadow region. So, we achieved the a much lighter shadow with color preserving method.

4.3.4 Method 4

```
alpha = 0.5
beta = 65
for x in range(image.shape[0]):
   for y in range(image.shape[1]):
      if mask_lab[x,y] == 0:
         image[x,y,0] = np.clip(alpha*image[x,y,0] + beta, 0, 255)
         image[x,y,1] = np.clip(alpha*image[x,y,1] + beta, 0, 255)
         image[x,y,2] = np.clip(alpha*image[x,y,2] + beta, 0, 255)

cv2.imshow('Increased',image)
```

Along with method 3, as the shadow was lighter, adjusting the contrast and brightness might remove the shadow and enhance the color intensity to match with the non-shaded region. Alpha is used to increase the contrast and beta is used for brightness. In computer vision, the multiplication adjust the contrast and addition adjusts the brightness. The value of alpha ranges from 0 to 3.0 and for brightness, it ranges from 0 to 100. We tried all possible combinations of alpha and beta on non-shaded regions but no such combination that reduces the shadow and increases the color intensity

was found.

4.3.5 Method 5

```
labelArray = np.zeros((image.shape[0],image.shape[1]))
for i in range(mask_lab.shape[0]):
   for j in range(mask_lab.shape[1]):
      if mask_lab[i][j] == 0:
         #shadow
         labelArray[i][j] = 1111
      else:
         #non-shadow
         labelArray[i][j] = 2222
print(image.shape,labelArray.shape)
image= cv2.copyMakeBorder(image,60,60,60,60,cv2.BORDER_CONSTANT,value
    ↪ =[0,0,0])
labelArray= cv2.copyMakeBorder(labelArray,60,60,60,60,cv2.BORDER_CONSTANT,
    ↪ value=3333)
#print(image,labelArray)
numberOfBlocks = 3
newH,newW = int(image.shape[0]/numberOfBlocks),int(image.shape[1]/
    ↪ numberOfBlocks)

for i in range(numberOfBlocks):
   # meanB,meanG,meanR = 0,0,0
   for j in range(numberOfBlocks):
      startR,endR,startC,endC = newH*i,(newH*(i+1))-1,newW*j,(newW*(j+1))-1
      tempImage = image[startR:endR,startC:endC]
      # tempMask = mask_lab[startR:endR,startC:endC]
      # image[startR:endR,startC:endC] = cv2.inpaint(tempImage,tempMask,3,
          ↪ cv2.INPAINT_TELEA)
      tempLabel = labelArray[startR:endR,startC:endC]
      bAvg,gAvg,rAvg = [],[],[]
      for p in range(tempImage.shape[0]):
         for q in range(tempImage.shape[1]):
            if tempLabel[p][q] == 2222.0:
               bAvg.append(image[p,q,0])
               gAvg.append(image[p,q,1])
               rAvg.append(image[p,q,2])
            if bAvg==[] or gAvg == [] or rAvg == []:
               meanB,meanG,meanR = 0,0,0
            else:
               meanB,meanG,meanR = np.mean(bAvg),np.mean(gAvg),np.mean(rAvg)
print(meanB,meanG,meanR)
for r in range(tempImage.shape[0]):
   for s in range(tempImage.shape[1]):
      if tempLabel[r][s] == 1111.0:
         tempImage[r,s] = [meanB,meanG,meanR]
         tempLabel[r][s] = 2222.0
         image[startR:endR,startC:endC] = tempImage
```

```
            labelArray[startR:endR,startC:endC] = tempLabel

cv2.imshow('Final',image)
```

As per the limitation of the previous method, a new technique was developed in which, the shadow pixels are labeled as 1111 and non-shadow pixels are label as 2222. Based on this information, the image is divided into equal number of blocks and the block is accessed one by one. If the block contains 1111, i.e shadow pixels, then the R, B and G values are replaced with the mean of the non-shadow pixels (2222) present in the same block. But, this created blocks on the shaded regions and also the averaging the values were resulting to different colors. We have also, tried to perform Inpainting methods on the block and its respective block of mask. But, it distorted the shaded region.

4.4 Final Implementation

Based on considering the above all limitation, we have developed this method. In this method, the shaded pixel is selected and its nearby vertical lines(columns) and horizontal lines(rows) are selected. Then the non-shaded pixels present in this rows and columns are collected. The euclidean distance is calculated between the shaded pixel with all other pixel and the nearest pixel from the shaded pixel is selected and its R, G and B values are replaced with the non-shaded pixel to remove shadow.

```
def openFile():
   global image,height,width
   filePath = filedialog.askopenfilename(title='Select your first image',
      ↪ filetypes=(('jpeg files','*.jpg'),('bmp files','*.bmp')))
   if filePath == "":
      print('No image is selected!')
      image = []
   else:
      image = cv2.imread(filePath)
      image = cv2.resize(image,(height,width))
      cv2.imshow('Original',image)
```

The above function is used to open the file dialog box, allowing the user to select and shaded image for further operation. If the file dialog is closed, without selecting the image, a error message "No image is selected!" displayed to the user. Once, the image is selected by the user, it is loaded into the memory, resized and displayed to the user.

```
#Function to calculate i and j values such as [i-3,i-2,i-1,i,i+1,i+2,i+3],
   ↪ [j-3,j-2,j-1,j,j+1,j+2,j+3]
def getRange(i,j,height,width):
   lineThreshold = 3
   I,J = [],[]
   for p in range(0,lineThreshold+1):
      tmpp,tmps = int(i+p),int(i-p)
      if tmpp >= 0 and tmpp < height:
         I.append(tmpp)
      if tmps >= 0 and tmps < height and p != 0:
```

```
            I.append(tmps)
      for q in range(0,lineThreshold+1):
         tmpp,tmps = int(j+q),int(j-q)
         if tmpp >= 0 and tmpp < width:
            J.append(tmpp)
         if tmps >= 0 and tmps < width and q != 0:
            J.append(tmps)
      return I,J
```

The function getRange() is used to selected the rows and columns around the shaded pixel. It is necessary to check that the rows and columns don't exceed beyond the height and width respectively. This function is called for all shaded pixel and its corresponding neighbor lines are returned back.

```
def shadowRemovalFunction():
   global image,height,width
   if image == []:
      print("Select an Image")
   else:
      print("Performing Shadow Removal Technique..")
      labImage = cv2.cvtColor(image,cv2.COLOR_BGR2Lab)
      grayImage = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
      lightness,a,b = cv2.split(labImage)
      mean,std = np.mean(lightness),np.std(lightness)

      maskLabImage = np.copy(labImage)
      for i in range(maskLabImage.shape[0]):
         for j in range(maskLabImage.shape[1]):
            if lightness[i,j] > (mean-(std/3)):
               #Consider to be non-shadow
               grayImage[i,j] = 0

      binaryImage = grayImage.copy()
      for i in range(grayImage.shape[0]):
         for j in range(grayImage.shape[1]):
            if grayImage[i,j] != 0:
               binaryImage[i,j] = 0
            else:
               binaryImage[i,j] = 255

      kernel = np.array([[1, 1, 1],[1, 1, 1],[1, 1, 1]],np.uint8)
      dilate = cv2.dilate(binaryImage,kernel,iterations=2)
      mask_lab = dilate.copy()

      cv2.imshow('Original',image)
      cv2.imshow('Mask',mask_lab)

      labelMatrix = np.zeros((image.shape[0],image.shape[1]))
      for i in range(mask_lab.shape[0]):
         for j in range(mask_lab.shape[1]):
            if mask_lab[i][j] == 0:
```

```
            #shadow
            labelMatrix[i][j] = 1111
         else:
            #non-shadow
            labelMatrix[i][j] = 2222

imageMatrixB = image[:,:,0]
imageMatrixG = image[:,:,1]
imageMatrixR = image[:,:,2]
imageMatrixA = labImage[:,:,1]

indexOfShadowPixels = []
indexOfNonShadowPixels = []
# print(len(indexOfShadowPixels))
# print(len(indexOfNonShadowPixels))

for i in range(labelMatrix.shape[0]):
   for j in range(labelMatrix.shape[1]):
      if labelMatrix[i][j] == 1111:
         shadowPixel = labImage[i,j,1]
         rangeOfI, rangeOfJ = getRange(i,j,height,width)
nonShadowNearPixels = []
for fullRow in range(0,height):
   for rj in rangeOfJ:
      if labelMatrix[fullRow][rj] == 2222:
         nonShadowNearPixels.append([fullRow,rj,labImage[fullRow,rj
            ↪ ,1]])
for ri in rangeOfI:
   for fullCol in range(0,width):
      if labelMatrix[ri][fullCol] == 2222:
         nonShadowNearPixels.append([ri,fullCol,labImage[ri,fullCol
            ↪ ,1]])
#Computing the Distance with Near Pixels
distance = []
for nsi in range(len(nonShadowNearPixels)):
   tmp = (int(nonShadowNearPixels[nsi][2])-int(shadowPixel))**2
   #               tmp = math.sqrt((int(nonShadowNearPixels[nsi
      ↪ ][2][0])-int(shadowPixel[0]))**2 + (int(nonShadowNearPixels[
      ↪ nsi][2][1])-int(shadowPixel[1]))**2 + (int(
      ↪ nonShadowNearPixels[nsi][2][2])-int(shadowPixel[2]))**2)
   if tmp >= 2:
      distance.append([tmp,nonShadowNearPixels[nsi][0],
         ↪ nonShadowNearPixels[nsi][1]])
distance.sort()
image[i,j] = image[distance[0][1],distance[0][2]]

cv2.imshow('Final Image',image)
print("Done")
cv2.waitKey(0)
cv2.destroyAllWindows()
mainWindow.destroy()
```

The function shadowRemovalFunction() detects the shadow and tries to remove it. The detection is performed based on the mean and standard deviation of the Lightness component of the LAB color space of image. The dark objects are slightly less compared to the shadow in the Lightness space. Hence, the detection of shadow is performed accurately. Also, to remove misclassified pixels, morphological operations are performed. After the detection is performed, the rows and columns are used to select the nearest pixel from the shaded pixel and replaced th R, G and B values with the nearest pixel.

4.5 Results

Following are the results obtained during various implementation of methods:

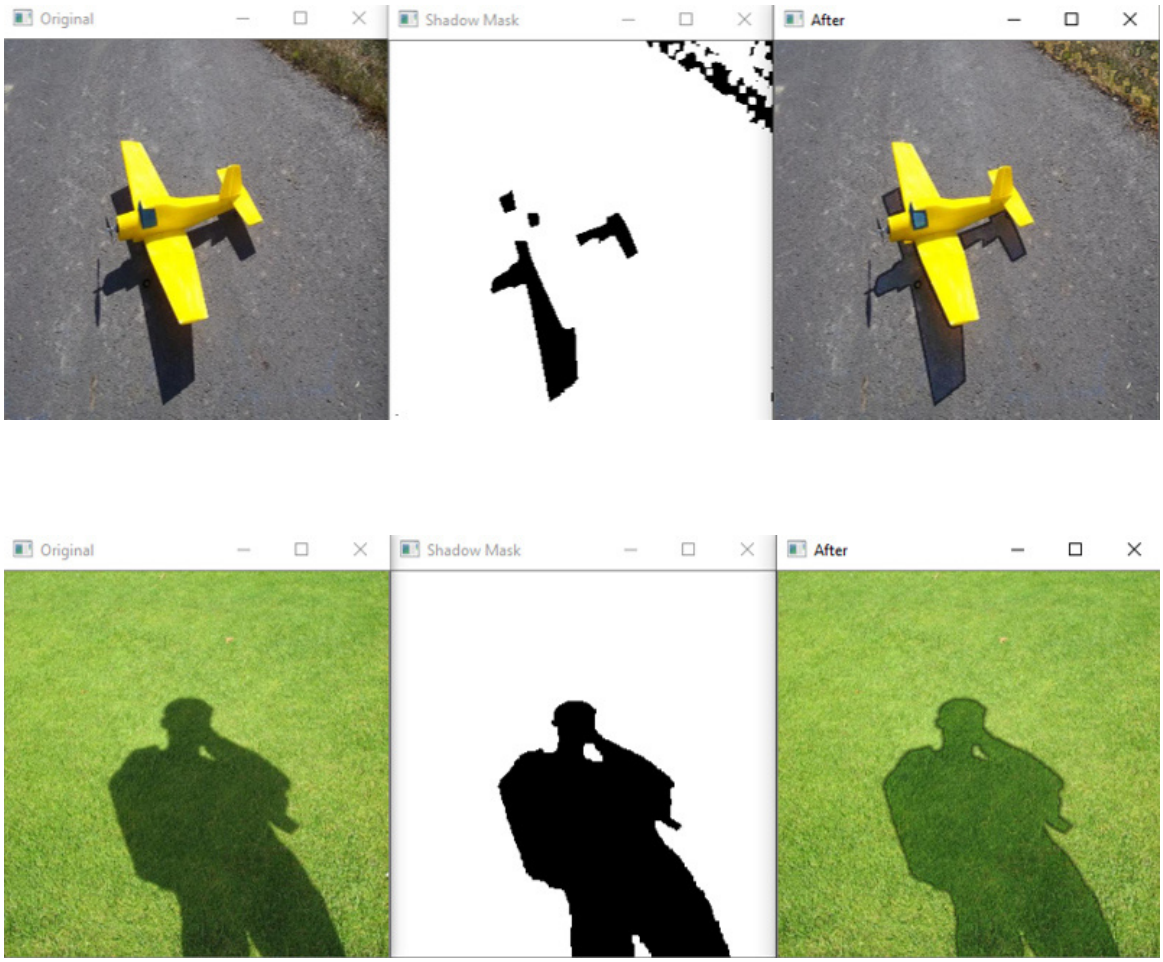

Figure 4.3: Shadow was lighten, not completely removed

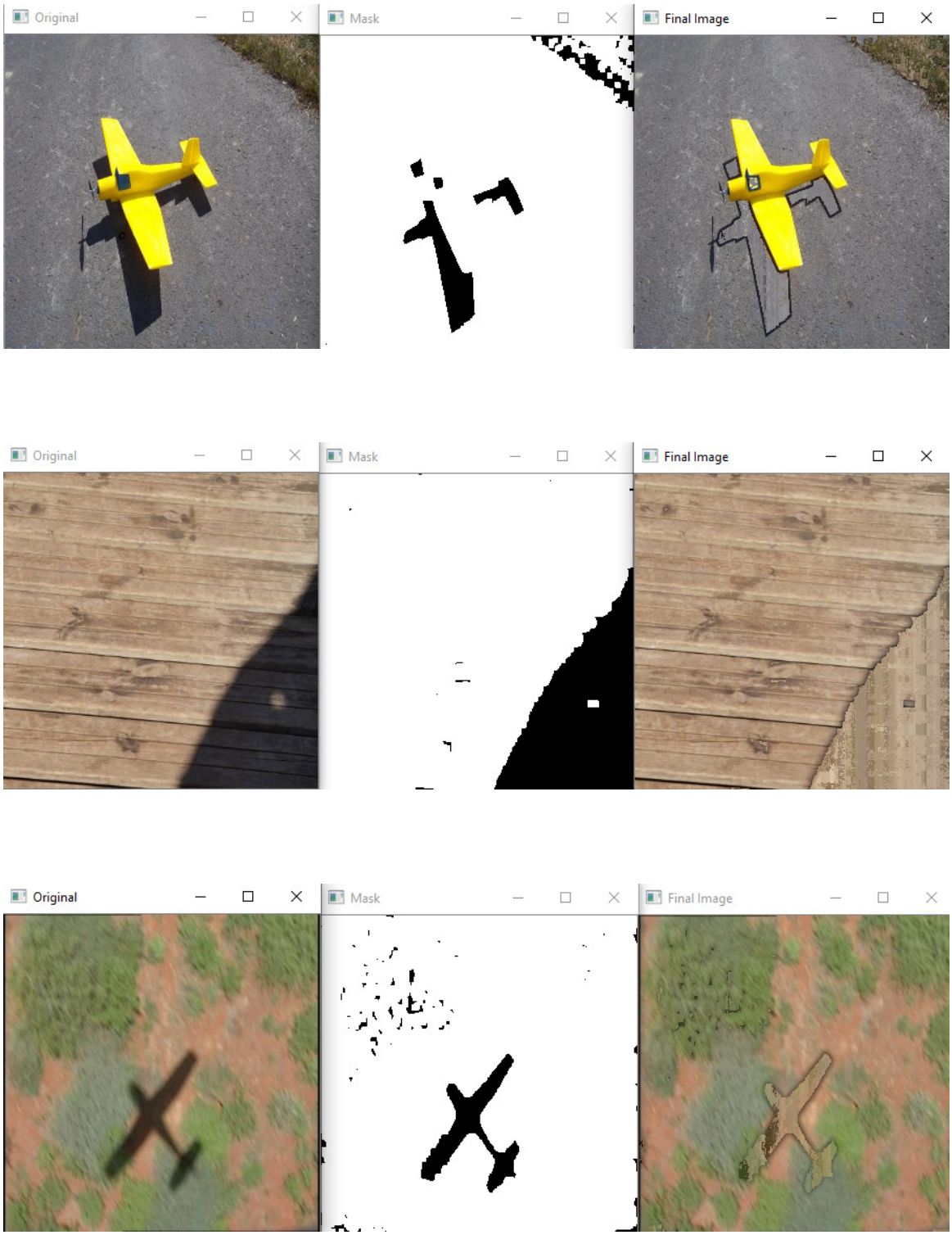

Figure 4.4: Final results achieved

Chapter 5

Discussion

Based on research performed, various algorithms have perform masking of shadow and regarding the shadow removal stated that morphological cleaning operation is performed. No more information is provided for shadow removal in most of the papers. Some of them have limitations that the algorithm only performs well on camera clicked photos as various camera parameters are needed to be set. Pre-trained models are easier to implement and they are readily available. Neural networks used in any research paper have at least used one pre-trained model such as VGG16 or DesNet. JPEG file is a compressed version of images. Thus, results in loss of information. Hence, we decided to develop an algorithm which can perform masking and also removes shadow. Based on the results displayed, we are somehow near to shadow removal. Our group have performed various methods based on our thinking and knowledge. Many limitations that occurred during the development of algorithms are overcomed and came to a conclusion were upto certain amount of shadow is removed. There may be a reason that the algorithm is not able to remove shadow appropriately due to the usage of JPEG files. The advantages of algorithm over others, it doesn't require a pre-trained model or weights. The algorithm is easier to understand and can be enhanced based on the requirement.

Chapter 6

Conclusion

Convolutional Neural Network involves maintaining a large number of parameters and execution time is also comparatively large. Most of the neural networks, doesn't remove shadows, the network is modeled to generate only the shadow masks. Whereas, the proposed algorithm doesn't uses any pre-trained model or weights. The algorithm can be further enhanced for removing shadow borders and removal of shadow can be more flawless.

Bibliography

[1] A. Prati, R. Cucchiara, I. Mikic, and M. Trivedi, "Analysis and detection of shadows in video streams: a comparative evaluation," IEEE Conf. Computer Vision and Pattern Recognition, volume 2, pages 571–576, 2001.

[2] Mikael A. Mousse, Eugene C. Ezin, and Cina Motamed, "Foreground-Background Segmentation Based on Codebook and Edge Detector," Universite d'Abomey-Calavi, Benin BP 613 Porto-Novo, France.

[3] Akmalbek Abdusalomov, and Adiljon Djurayev, "Robust Shadow Removal Technique For Improving Image Enhancement Based On Segmentation Method," IOSR Journal of Electronics and Communication Engineering (IOSR-JECE) Volume 11, Issue 5, Ver. III (Sep.-Oct .2016), PP 17-21.

[4] Dominic Rufenacht, Clement Fredembach, and Sabine Susstrunk, "Automatic and accurate shadow detection using near-infrared information," IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE.

[5] N. Salamati, A. Germain, and S. Susstrunk, "Removing shadows from Images using Color AND Near-Infrared," School of Computer and Communication Sciences, Switzerland.

[6] Vicky Nair, Parimala Geetha Kosal Ram, and Sundaravadivelu Sundararaman, "Shadow detection and removal from images using machine learning and morphological operations," The Journal of Engineering.

[7] Zhu, Z., Woodcock, C.: "Object-based cloud and cloud shadow detection in Landsat imagery," Remote Sens. Environ., 2012, 118, pp. 83–94, doi: 10.1016/j.rse.2011.10.028

[8] Guo, R.; Dai, Q.; Hoiem, D. Paired Regions for Shadow Detection and Removal. IEEE Trans. Pattern Anal. Mach. Intell. 2013, 35, 2956–2967

[9] Wang, J.; Li, X.; Hui, L.; Yang, J. Stacked Conditional Generative Adversarial Networks for Jointly Learning Shadow Detection and Shadow Removal. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018.

[10] Fan, Hui & Han, Meng & Li, Jinjiang, "Image Shadow Removal Using End-to-End Deep Convolutional Neural Networks," Applied Sciences, 2019.