Linear Algebra's Applications to Image Analysis

An Introduction to Image Manipulation and Analysis with Matlab

Tyler Eley

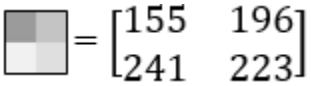
ENGR 2300.004

Dr Peter Blakey

May 6, 2017

Linear Algebra has many applications to image manipulation and analysis. One can use linear algebra techniques to preform image manipulation, image compression, and facial recognition.

An image can be represented by a matrix, which means that one can utilize linear algebra to analyze and manipulate images. A simple greyscale image with a resolution of $m \times n$ can be represented by a $m \times n$ matrix where the element of each index of the matrix is a number within the range $[0,256)$ or $[0, 2^8]$. The image below is an example of a simple image with a resolution of 2×2 as image and as a matrix.[1]



$$= \begin{bmatrix} 155 & 196 \\ 241 & 223 \end{bmatrix}$$

Using this, one can determine how much memory is required to represent a single picture. Each index of the matrix, and by extension each pixel of the picture, can have a maximum value of 256. This means that each pixel of a greyscale image requires 8 bits. If a picture has a resolution of $m \times n$, then the picture can be represented by (8) $(m \times n)$ bits. A colored image requires three color scales, which means that a single colored image requires $(3)(8)(m \times n)$ bits. Since greyscale images require less memory than colored images, it is easier to compress images that are greyscale. [1] The properties of these matrices enables more advance image manipulation and analysis.

Matlab provides powerful image manipulation tools. An image from a file can be represented as a matrix by using the command *imageObject = imread('<Image_File_Name>');* Once an image is imported to Matlab, one can use Matlab's graphing and morphology tools to edit an image. [5] To see an example of Matlab's image editing capabilities, look the Color Segment program in the appendix.

One can use the Singular Value Decomposition to preform image reconstruction. Image reconstruction is the process of converting a blurred or otherwise damaged image back into its original image prior to being distorted. The algorithm for image reconstruction takes two inputs, a matrix representing a greyscale image, and an integer, *k*. First, perform the Singular Value Decomposition on the matrix representing the image. Next, for every k^{th} row, reconstruct the image matrix, calculate the ratio of compression, and the root means squared error. The compression ratio and root means squared error can be used to measure how accurately the algorithm reconstructs the image. The end result of this process is a reconstructed image. [6]

Matlab also has assets for computer vision. The *CascadeObjectDetector* object can be used to detect objects in a picture. By default, the *CascadeObjectDetector* detects faces that are looking directly at the camera, however it is possible to train the *CascadeObjectDetector* to detect any object with fixed aspect ratio, such as street signs or vehicles. This can be used in conjunction with the *webcam* addon to detect if a person is facing a webcam. The Facial Tracking program in the Appendix builds on this concept by take several greyscale images from a webcam and drawing a box around each detected face to create a simple video demonstrates basic facial tracking.

Image manipulation and facial recognition are only a few of the techniques made possible with Matlab. From the examples provided in this report, it is not hard to see how Matlab provides tools to enables anyone with a basic understanding of Linear Algebra to easily find solutions for problems that once seemed impossible.

Appendix – Programs

Program 1: Color Segment

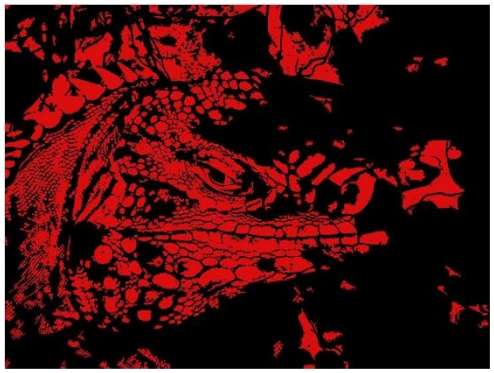
```
%PROGRAM: Color Segment
%DESCRIPTION: Takes a image, and separates it into two layers, a foreground
%and a background. The output is an image where the background is black and
%the foreground is a color chosen by the user. [3][4]

%This is the sample images I used.
lizard = imread('C:\Users\Tyler\Desktop\Assets for Lin Alg Report\Source
Images\lizardbro.jpg');

%This block of code contains variables that the user can change
imageToUse = lizard %This variable represents the image to use.
radius = 20 %As the value of radius increases, more of the image will be
colored. Any pixels that are uncolored will be black.
red = 219; %This determines the red value of the images color
green = 15;%This determines the green value of the images color
blue = 15; %This determines the blue value of the images color

red = red/255
green = green/255
blue = blue/255
imageGrey = rgb2gray(imageToUse) %Converts the image to greyscale
j = 0;
background = imopen(imageGrey,strel('disk',radius));%The background is
composed of several disks with a size equal to radius
figure
surf(double(background(1:8:end,1:8:end))),zlim([0 255]);
set(gca,'ydir','reverse'); %The 3 lines above are used for preprocessing
image2 = imageGrey - background; %Subtracts the background from the image.
bwImage = imbinarize(image2);%Cleans up background. Without this, the program
runs, however the background is partially colored as well
bwImage = bwareaopen(bwImage, 50);
rgbImage = cat(3, red*double(bwImage), green*double(bwImage),
blue*double(bwImage));%Colors all white pixels the selected color
imshow(rgbImage) %Display Final Image
```

Input Image [8]



Output Image

Program #2 Facial Tracking
(Requires Function #1 [cam2ImgArray], Function #2 [frameToPicArray], and the webcam addon for Matlab)

```
%Program: facialTrack
%Description: Takes 10 frames of images from an attached webcam. After that,
the program draws a bounding box around each face that is detected in each
image. Once image has been analyzed, it plays back a greyscale version of
each picture in order, essentially making a simple 10 frame video with motion
tracking
%Required for the facialTrack program

%Requires the MatLab webcam addon. You can get this through the Add-On
browser located in the Environment Section of the Home Menu in Matlab
clear %Clears the workspace. This is to ensure that there is only one webcam
object

pause(1) %Waits 1 second. This gives the user enough time to be in clear view
of the webcam before it takes photos.
cam = webcam %Creates a web camera object. It uses any detected web camera
that is attached to the computer.
cam.Resolution = '320x240';%Limits the resolution of the camera to 320 by
240. The program works with any resolution, however by using a small
resolution, runtime is reduced.
faceDetector = vision.CascadeObjectDetector; %Creates the facial recognition
object. The default setting detects faces that look directly at the camera
imgArray = cam2ImgArray(cam,faceDetector)% Takes 10 pictures, detects faces,
and outputs an array of 10 photos with bounding boxes around each detected
face. [2]
mplay(imgArray)%Plays back the 10 frames of images as a video.
clear cam
```

Output

Function #1: cam2ImgArray

```
%Function: cam2ImgArray(<webcam>, <CascadeObjectDetector>)
%Description: Takes 10 frames worth of photos from the webcam.
%Required for the facialTrack program
function [ out ] = cam2ImgArray(cam,faceDetector)
image1=cam.snapshot
image2=cam.snapshot
image3=cam.snapshot
image4=cam.snapshot
image5=cam.snapshot
image6=cam.snapshot
image7=cam.snapshot
image8=cam.snapshot
image9=cam.snapshot
image10=cam.snapshot
clear cam
%Converts the 10 frames into a single array
imgArray=cat(3,image1,image3,image4,image5...
,image6,image7,image8,image9,image10)

out = frameToPicArray(imgArray,faceDetector)%Converts the array of images
into an array of frames with a bounding box around any detected faces
end
```

Function #2: frameToPicArray

```
%Function: frameToPicArray(<imgArray>,
<CascadeObjectDetector>)
%Description:Converts the array of images
into a array of frames with a bounding box
around any detected face
%Required for the facialTrack program and
cam2ImgArray function
function [ picArray ] = frameToPicArray(
imgArray,faceDetector )
i =1;

i=i+1
image1 =imgArray(:,:,i); %Takes image1 from
the image array
bboxes = step(faceDetector,image1) %Looks
for faces in the picture
faces = insertObjectAnnotation(image1,
'rectangle', bboxes, 'Face Detected');%Draws
a bounding box around each face
figure, imshow(faces), title('Output')
%Displays the image
f = getframe;
[pic1,map1] = frame2im(f) %Converts the
frame (The graph that is generated with a
bounding box around each face) into a image
object
%This process is continued for each of the
10 photos

i=i+1
image2 =imgArray(:,:,i);
bboxes = step(faceDetector,image2)
faces = insertObjectAnnotation(image2,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic2,map2] = frame2im(f)
i=i+1
image3 =imgArray(:,:,i);
bboxes = step(faceDetector,image3)
faces = insertObjectAnnotation(image3,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic3,map3] = frame2im(f)
i=i+1
image4 =imgArray(:,:,i);
bboxes = step(faceDetector,image4)
faces = insertObjectAnnotation(image4,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic4,map4] = frame2im(f)
i=i+1
image5 =imgArray(:,:,i);
bboxes = step(faceDetector,image5)
faces = insertObjectAnnotation(image5,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic5,map5] = frame2im(f)
i=i+1
image6 =imgArray(:,:,i);
bboxes = step(faceDetector,image6)
faces = insertObjectAnnotation(image6,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic6,map6] = frame2im(f)
i=i+1
image7 =imgArray(:,:,i);
bboxes = step(faceDetector,image7)
faces = insertObjectAnnotation(image7,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic7,map7] = frame2im(f)
i=i+1
image8 =imgArray(:,:,i);
bboxes = step(faceDetector,image8)
faces = insertObjectAnnotation(image8,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic8,map8] = frame2im(f)
i=i+1
image9 =imgArray(:,:,i);
bboxes = step(faceDetector,image9)
faces = insertObjectAnnotation(image9,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic9,map9] = frame2im(f)
i=i+1
image10 =imgArray(:,:,i);
bboxes = step(faceDetector,image10)
faces = insertObjectAnnotation(image10,
'rectangle', bboxes, 'Face Detected');
figure, imshow(faces), title('Output')
f = getframe;
[pic10,map10] = frame2im(f)

%At this point, we have 10 images with a
bounding box around each face that
%is detected.

%This block of code converts the images of
frames into an array of regular images
picArray=cat(3,pic1,pic2,pic3,pic4,pic5,pic6
,pic7,pic8,pic9,pic10)
end
```

Works Cited

[1] G. Strang, "Introduction to Linear Algebra, 5th Edition", Math.mit.edu, 2016. [Online]. Available: https://math.mit.edu/~gs/linearalgebra/. [Accessed: 22- Apr- 2017].

[2] User abcd on Stack Overflow, "Build an array of images in Matlab", Stackoverflow.com, 2011. [Online]. Available: http://stackoverflow.com/questions/6496696/build-an-array-of-images-in-matlab. [Accessed: 24- Apr- 2017].

[3] Image Analysis on Matlab Answers, "Changing all of one color in an image to another - MATLAB Answers - MATLAB Central", Mathworks.com, 2015. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/231831-changing-all-of-one-color-in-an-image-to-another. [Accessed: 24- Apr- 2017].

[4] Image Analysist on Matlab Answers, "convert gray image back to rgb - MATLAB Answers - MATLAB Central", Mathworks.com, 2013. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/67137-convert-gray-image-back-to-rgb. [Accessed: 24- Apr- 2017].

[5] "MATLAB Documentation - MathWorks United Kingdom", Mathworks.com. [Online]. Available: https://www.mathworks.com/help/matlab/. [Accessed: 24- Apr- 2017].

[6] S. Abdul Karima, M. Mohd Mustafa, B. Abdul Karim, M. Hasan, J. Sulaiman and M. Ismail, "Image reconstruction using singular value decomposition.", AIP Conference Proceedings, vol. 1522, no. 1, pp. 268 - 274, 2013.

[7] C. Eley, A Picture of a Lizard taken at the Little Cayman Beach Resort in Blossom Village, Cayman Islands. 2016.