MTH 261: Midterm Project Write-Up
Chioma Opara

Q1 - Q5

Q1: The purpose of double and uint8 in code
Converting the image array to a double allows us to perform the multiplication needed to convert the image to grayscale with greater precision. We convert the array back to 8-bit integers (between 0-255) using uint8 to allow us to store and display the transformed image.

Q2: What does the filter matrix [1 0 0; 0 0 0; 0 0 0] do?
It keeps the red channel in the image as it was originally and eliminates the blue and green channels. The overall effect is that the image appears entirely in shades of red, since all color information now comes only from the red channel.

Q3: What does this transformation [0 0 1; 0 1 0; 1 0 0] do?
It swaps the intensity of the blue and red channels. The red is rendered with the intensity of the blue channel and blue channel is rendered with the original red intensity. The green channel remains unchanged. This creates a noticeable shift in color where blue areas appear reddish and red areas appear bluish.

Q4: What does each command do:

- `ImJPG1 = ImJPG(100:m - 100, 100: n-70);`

Crops the image and converts it to grayscale

- `ImJPG_flip = flip(ImJPG);`

Flips the image

- `ImJPG_90 = rot90(ImJPG);`

Rotates the image 90 degrees counterclockwise

- `ImJPG255 = 255-ImJPG;`

Produces a photo-negative version of the image

- `ImJPG50 = ImJPG-50;`

Darkens the image

- `ImJPG_uint = uint8(1.25 * ImJPG);`

Increases the brightness and intensity of the image by 25%

Q5: Explain what the initial transformation does to the image. What does the inverse transformation do with the colors?

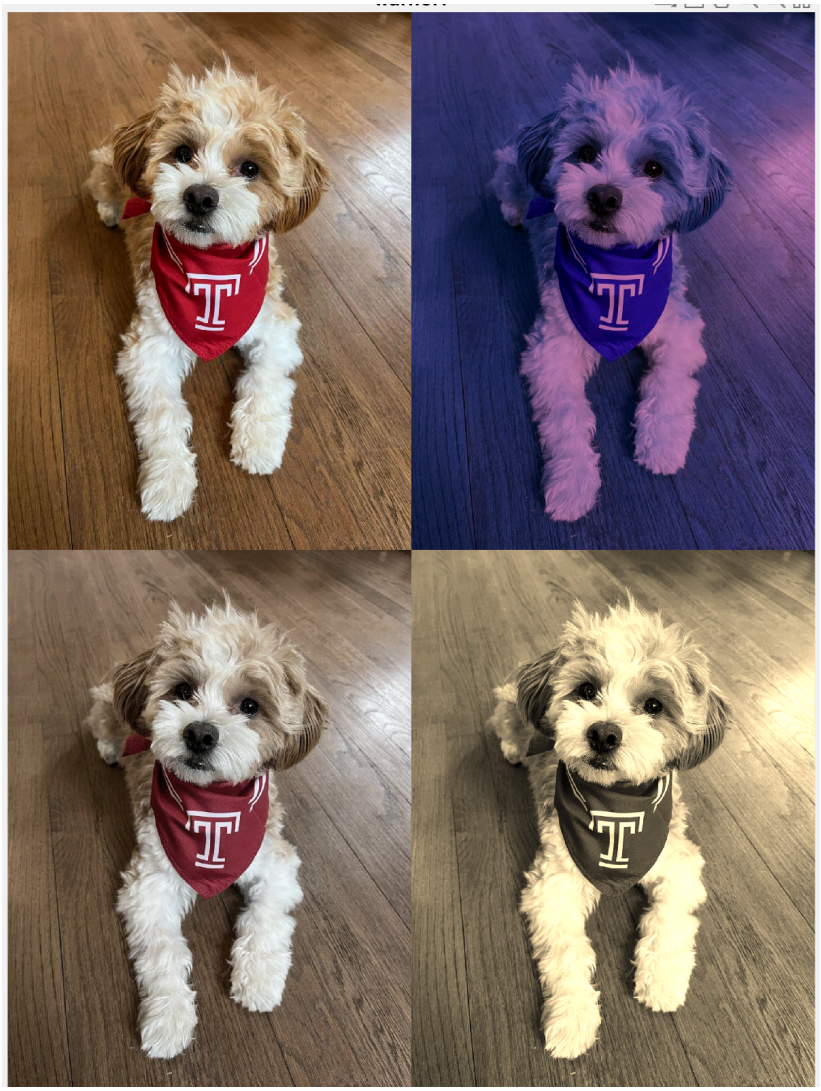
The initial transformation causes each of the RGB channels to be composed of mostly the original intensity of that channel and 15% of the intensity of the other two channels. This

reduces the contrast between all three channels. The result is that the image looks mostly the same, but the dominant colour (red in this case) appears to be slightly muted as its intensity decreases. The inverse transformation reverses this effect to produce the original image. There may be slight differences due to rounding

Warhol Images

Initially, I tried creating warhol collages from images produced from transforming the orientation of the original image e.g. flipping, rotating etc. However, I kept having issues with the dimensions not being uniform. So I could not concatenate the matrices of the images. I decided to stick to colour transformations.

Image 1



Description:

- Top-left: Original image

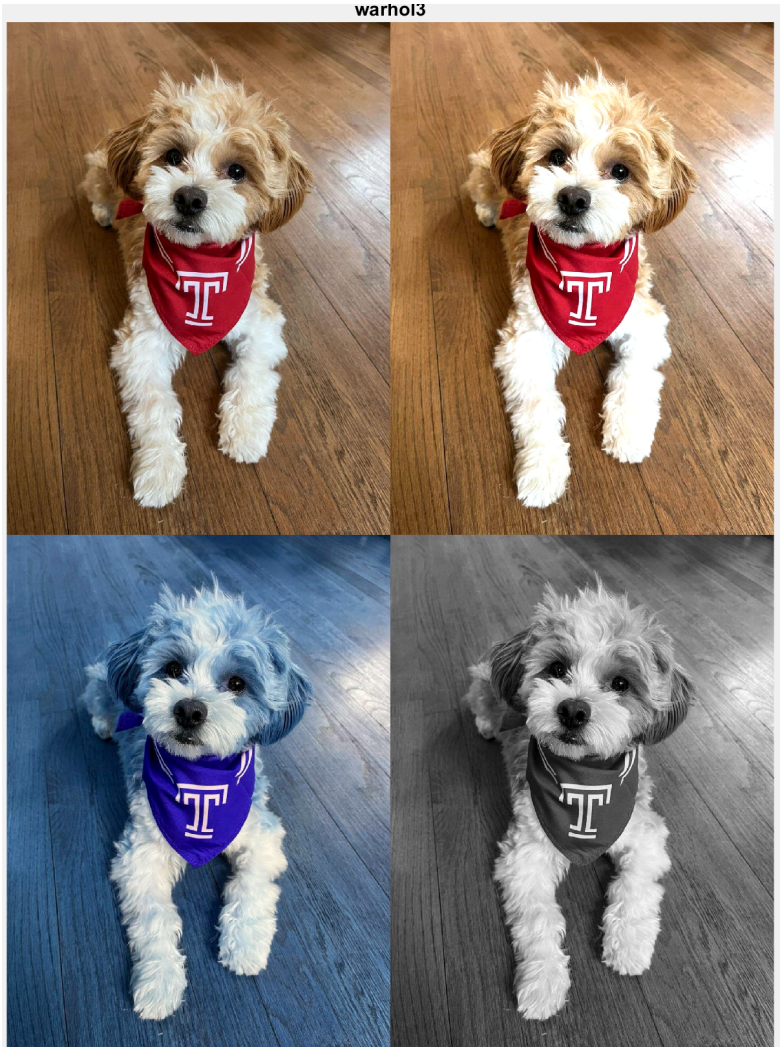
- Top-right: Custom filter (Section 3.7) that remaps red and blue channels, giving the image a bluish tone
- Bottom-left: Image processed with a transformation matrix (section 4.1) that reduces the contrast between RGB channels, slightly muting the dominant colors– red in this case.
- Bottom-right: Sepia filter that gives the image a warm, vintage look

Image 2

Description:

- Top-left: Original image
- Top-right: Darkened version (ImJPG - 50) that reduces intensity across all color channels
- Bottom-left: Negative image (255 - ImJPG) that inverts all colors
- Bottom-right: Vertically flipped image, giving a mirrored visual distortion

Image 3



Description:

- Top-left: Original image
- Top-right: Brightened image using 1.25 * ImJPG to increase color intensity
- Bottom-left: The red and blue color channels in the original image are swapped to produce this effect (section 3.5)
- Bottom-right: Grayscale image based on equal-weight RGB averaging

Free Exploration Section

Acknowledgement: ChatGPT was used to brainstorm ideas and help break down the overall concept into implementable steps. It was also used for debugging when the code produced unexpected errors.

Description: This free exploration project implements an animated ASCII art sequence. The user provides one or two images, which are converted into ASCII text and animated using a series of visual transformations. These include scrolling, flipping, wave motion, and a transition between images. An unsuccessful attempt at creating a zoom transformation was also made. The goal was to creatively manipulate image representations in a non-traditional, text-based format.

Delegation: I worked independently.

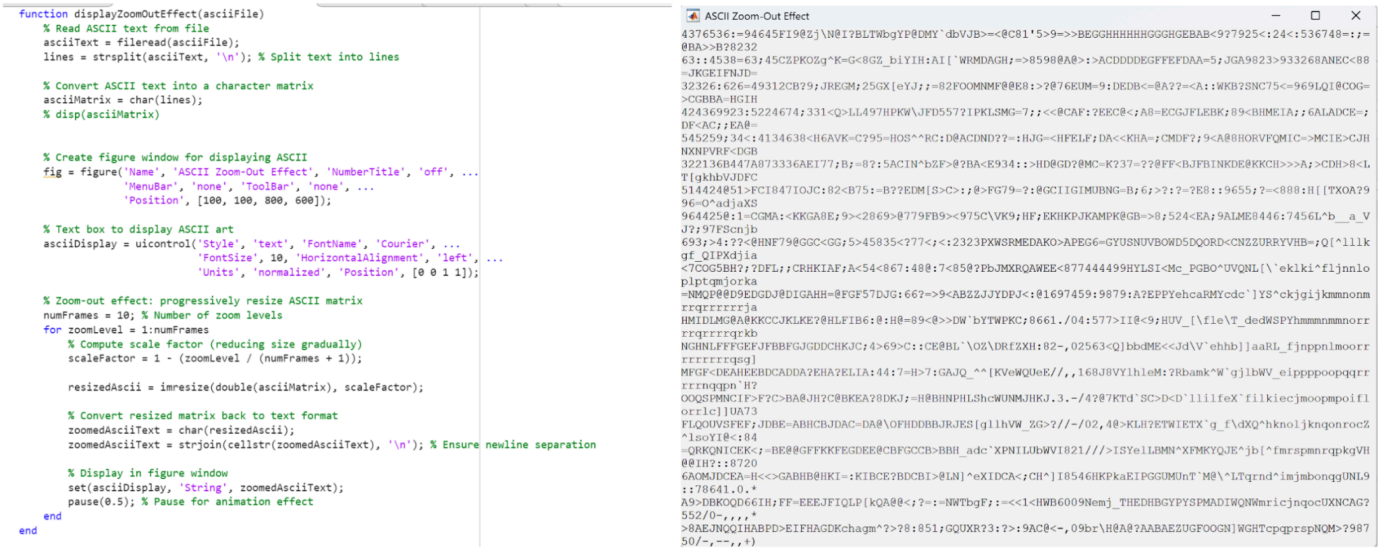
Step 1: Convert Image to ASCII (using img2txt.m)
I found an online script that converts images into ASCII and outputs a .txt file. The ASCII image uses characters with different levels of brightness to approximate grayscale shading.

Step 2: Displaying the ASCII Art
I attempted to display the ASCII image as plain text in a user interface. This turned out to be more difficult than expected. The dimensions of the display area and the image's original size had to be coordinated precisely for the image to appear correctly.

The biggest challenge was that the bottom part of the image was being cropped. I suspected this was due to the font size. When the font size was set above zero, part of the image disappeared. When it was set to zero, the text wouldn't display at all. I ended up adjusting the display settings for the images I worked with manually. These settings worked well for two other images I tried, but there is no guarantee that it will work well for most images.

Step 3: **Implementing a Zoom Feature**
Initial Approach:
I attempted to convert the ASCII text into a matrix of numbers representing character brightness levels, apply a scale factor, and then map the resized matrix back into ASCII characters.

However, this failed because scaling the matrix changed the pixel values. As a result, the new values didn't correspond to the original character set, and the image lost its intended appearance.



```
function displayZoomOutEffect(asciiFile)
    % Read ASCII text from file
    asciiText = fileread(asciiFile);
    lines = strsplit(asciiText, '\n'); % Split text into lines

    % Convert ASCII text into a character matrix
    asciiMatrix = char(lines);
    % disp(asciiMatrix)


    % Create figure window for displaying ASCII
    fig = figure('Name', 'ASCII Zoom-Out Effect', 'NumberTitle', 'off', ...
                 'MenuBar', 'none', 'ToolBar', 'none', ...
                 'Position', [100, 100, 800, 600]);

    % Text box to display ASCII art
    asciiDisplay = uicontrol('Style', 'text', 'FontName', 'Courier', ...
                             'FontSize', 10, 'HorizontalAlignment', 'left', ...
                             'Units', 'normalized', 'Position', [0 0 1 1]);

    % Zoom-out effect: progressively resize ASCII matrix
    numFrames = 10; % Number of zoom levels
    for zoomLevel = 1:numFrames
        % Compute scale factor (reducing size gradually)
        scaleFactor = 1 - (zoomLevel / (numFrames + 1));

        resizedAscii = imresize(double(asciiMatrix), scaleFactor);

        % Convert resized matrix back to text format
        zoomedAsciiText = char(resizedAscii);
        zoomedAsciiText = strjoin(cellstr(zoomedAsciiText), '\n'); % Ensure newline separation

        % Display in figure window
        set(asciiDisplay, 'String', zoomedAsciiText);
        pause(0.5); % Pause for animation effect
    end
end
```

ASCII Zoom-Out Effect

```
4376536:=94645FI9@Zj\N@I?BLTWbgYP@DMY`dbVJB>=<@C81'5>9=>>BEGGHHHHHHGGGHGEBAB<9?7925<:24<:536748=:;=
@BA>>B?8232
63::4538=63;45CZPKOZg^K=G<8GZ_biYIH:AI[`WRMDAGH;=>8598@A@>:>ACDDDDEGFFEFDAA=5;JGA9823>933268ANEC<88
=JKGEIFNJD=
32326:626=49312CB79;JREGM;25GX[eYJ;;=82FOOMNMF@@E8:>?@76EUM=9:DEDB<=@A7?=<A::WKB?SNC75<=969LQI@COG=
>CGBBA=HGIH
424369923:5224674;331<Q>LL497HPKW\JFD557?IPKLSMG=7;;<<@CAF:?EEC@<;A8=ECGJFLEBK;89<BHMEIA;;6ALADCE=;
DF<AC;;EA@=
545259;34<:4134638<H6AVK=C?95=HOS^^RC:D@ACDND??=:HJG=<HFELF;DA<<KHA=;CMDF?;9<A@8HORVFQMIC=>MCIE>CJH
NXNPVRF<DCB
322136B447A873336AEI77;B;=8?:5ACIN^bZF>@?BA<E934::>HD@GD?@MC=K?37=??@FF<BJFBINKDE@KKCH>>>A;>CDH>8<L
T[gkhbVJDFC
514424@51>FCI847IOJC:82<B75:=B??EDM[S>C>:;@>FG79=?:@GCIIGIMUBNG=B;G;>?:?=?E8::9655;?=<888:H[[TXOA?9
96=O^adjaXS
964425@:1=CGMA:<KKGA8E;9><2869>@779FB9><975C\VK9;HF;EKHKPJKAMPK@GB=>8;524<EA;9ALME8446:7456L^b__a_V
J?;97FScnjb
693;>4:??<@HNF79@GGC<GG;5>45835<?77<;<:2323PXWSRMEDAKO>APEG6=GYUSNUVBOWD5DQORD<CNZZURRYVHB=;Q[^lllk
gf_QIPXdjia
<7COG5BH?;?DFL;;CRHKIAF;A<54<867:48@:7<85@?PbJMXRQAWEE<877444499HYLSI<Mc_PGBO^UVQNL[\`eklki^fljnnlo
plptqmjorka
=NMQP@@D9EDGDJ@DIGAHH=@FGF57DJG:66?=>9<ABZZJJYDPJ<:@1697459:9879:A?EPPYehcaRMYcdc`]YS^ckjgijkmmnonm
rrqrrrrrrja
HMIDLMG@A@KKCCJKLKE?@HLFIB6:@:H@=89<@=>>DW`bYTWPKC;8661./04:577>II@<9;HUV_[\fle\T_dedWSPYhmmmmmmnorr
rrqrrrrqrkb
NGHNLFFFGEFJFBBFGJGDDCHKJC;4>69>C::CE@BL`\OZ%DRfZXH:82-,02563<Q]bbdME<<Jd\V`ehhb]]aaRL_fjnppnlmoorr
rrrrrrrqsq]
MPGF<DEAHEEBDCADDA?EHA?ELIA:44:7=H>7:GAJQ_^^[KVeWQUeE//,,168J8VYlhleM:?Rbamk^W`gjlbWV_eipppoopqqrr
rrrnqqpn`H?
OOQSPMNCIF>F?C>BA@JH?C@BKEA?8DKJ;=H@BHNPHLShcWUNMJHKJ.3.-/4?@7KTd`SC>D<D`llilfeX`filkiecjmoopmpoifl
orrlc]]UA73
FLQOUVSFEF;JDBE=ABHCBJDAC=DA@\OFHDDBBJRJES[qllhVW_ZG>?//-/02,4@>KLH?ETWIETX`g_f\dXQ^hknoljknqonroc%
^lsoYI@<:84
=QRKQNICEK<;=BE@@GFFKKFEGDEE@CBFGCCB>BBH_adc`XPNILUbWVI821///>ISYelLBMN^XFMKYQJE^jb[^fmrspmnrqpkgVH
@@IH?::8720
6AOMJDCEA=H<<>GABHB@HKI=:KIBCE?BDCBI>@LN]^eXIDCA<;CH^]I8546HKPkaEIPGGUMUnT`M@\^LTqrnd^imjmbonqgUNL9
::78641.0.*
A9>DBKOQD66IH;FF=EEEJFIQLP[kQA@@<;?=:=NWTbgF;:=<<1<HWB6009Nemj_THEDHBGYPYSPMADIWQNWmricjnqocUXNCAG?
552/0-,,,,*
>8AEJNQQIHABPD>EIFHAGDKchagm^?>?8:851;GQUXR?3:?>:9AC@<-,09br\H@A@?AABAEZUGFOOGN]WGHTcpqprspNQM>?987
50/-,--,,+)
```

Approach 2:
To simulate a zoom-out effect, I tried dividing the ASCII image into blocks and replacing each block with the most frequent character inside it. My aim was to create a zoom-out animation by increasing the block size over time,

This approach ran into issues with the blockResize function, which threw a dimension mismatch error.

```
Unable to perform assignment because the size of the left side is 1-by-1 and
the size of the right side is 2-by-1.

Error in ZoomOut>blockResize (line 61)
                reducedAscii(i, j) = uniqueChars(maxIdx);

Error in ZoomOut (line 15)
        reducedAscii = blockResize(asciiMatrix, blockSize);
```

Step 4: Scrolling Animation
I started with the left scroll, which was fairly straightforward. To create the scrolling effect, I removed characters from the left side of each line and added spaces to the right. This made it look like the image was moving to the left. One problem is that if too many characters are removed, the whole image can disappear off the screen at once. So it works best when the scroll amount is small.

After getting the left scroll to work, I used the same idea to create scrolling in other directions — right, up, and down. All I had to do was change how I edited the ASCII matrix, either by cutting from the other side or shifting rows instead of columns.

Step 5: “Dancing image”

While experimenting with right-scroll, I accidentally flipped the ASCII matrix horizontally in a loop. This resulted in a flickering effect that resembled a "dancing" image. I kept this in as a stylistic transformation.

Step 6: Flipping into another image
To make the animation more dynamic, I added the ability to transition between two ASCII images. Instead of transforming a single image repeatedly, the final frames shift into the second image. This creates a visual flip or morph between scenes. An additional step I considered adding, but didn't due to time constraints was to allow the second image to appear gradually like a fade in effect or to allow one frame scroll into the next when transitioning to a new image.

Future Work:

- ASCII Music Synchronization: Incorporate audio into the ASCII animation by syncing visual transitions with sound. For example, changes in the beat or volume could trigger specific transformations (like zooms, flips, or flashes) in the ASCII "video."
- Interactive Animation Tool: Transform the current animation into an interactive program where users can select which transformations to apply
- Live Webcam Input to ASCII: Extend the project to support real-time ASCII conversion from a webcam feed.