Lab 1. Del 2- Laboration
Svarsdokument

Spara detta dokument som .pdf dokument innan ni lägger ut det på Lisam.

Studenternas namn och LiU-ID: (Max 2 studenter per grupp):
1. Adna Maric, adnma186
2. Rebecca Arkhed, rebar310
Inlämningsdatum: 4/2-2025
Version 1

Observera att ibland krävs det att ni ska skriva ett antal MATLAB kommandon som svar. Detta kan enkelt göras genom kopiera-klistra-in från MATLAB till svarsdokumentet.

Observera att ni kan skriva matriserna antingen genom att använda ”Equation i words” och skapa matriser där, eller skriva dem precis som man definierar en matris i MATLAB, eller på ett annat lämpligt sätt för att tydligt visa en matris.

1) Variabeltyper och bildvisning

Uppgift 1.1)
Varför är *b/4* inte helt korrekt?

När a =5 konverteras till uint8 så fås b och eftersom 5 är inom intervallet för uint8 ([0, 255]) så bevaras dess värde. Sedan när man utför b/4 så sker divisionen med heltal enligt uint8 formatet och ger således att det kommer avrundas till närmsta heltal i stället för det exakta decimaltalet.

Vad får vi ut genom *b/12*? Och Varför?
På samma sätt som i (a) kommer b/12 avrundas till närmsta heltal, vilket i detta fall blir 0 då $\frac{5}{12} \approx 0.4167.$

Uppgift 1.2)
Hur mycket mer minne behöver *k2* jämfört med *k* och varför?

För k gäller (1byte/pixel**):**

- **Bildens upplösning:** $512\ x\ 512\ pixlar = 262\ 144\ pixlar$
- **Varje pixel kräver** $262\ 144\ pixlar\ x\ 1\frac{byte}{pixel} = 262\ 144\ bytes$

För k2 gäller (8 byte/pixel**):**

- **Varje pixel kräver** $262\ 144\ pixlar\ x\ 8\frac{byte}{pixel} = 2\ 097\ 152\ bytes$

Skillnad i minne mellan k och k2:

$$2\ 097\ 152 - 262\ 144 = 1\ 835\ 008\ bytes$$

SVAR: k2 kräver 1 835 008 bytes mer minne än bild k.

Uppgift 1.3)
Förklara varför medan bilden *k* visas som en korrekt bild, visas *k2* som en helvit bild.
imshow(k); visar bilden korrekt.

För uint8-bilder: imshow tolkar bildens värden mellan 0 och 255,
där 0 motsvarar svart och 255 motsvarar vitt. Värdena i bilden kommer att tolkas och visas i det intervallet.

imshow(k2); visar en helvit bild.

För double-bilder: imshow tolkar bildens värden mellan 0 och 1, där 0 motsvarar svart och 1 motsvarar vitt. Om värdena i en double-bild är större än 1, kommer dessa värden att tolkas som vitt, vilket innebär att bilder med värden större än 1 kommer att visas som helvita.

Uppgift 1.4)
Förklara varför medan *imshow(k/255)* visas som en helt svart bild, visas *imshow(k2/255)* som en korrekt bild.

imshow(k/255) denna blir helt svart.
När du delar k med 255, kommer divisionen att ge resultat i decimalformat.
Men eftersom k är en uint8-bild, utförs divisionen också på ett sätt som resulterar i heltalsvärden efter avrundning. Detta betyder att värden som är mindre än 255 (men större än 0) kommer att avrundas till 0, och endast de pixelvärden som är exakt 255 kommer att bli 1. Alla andra värden kommer att avrundas till 0.

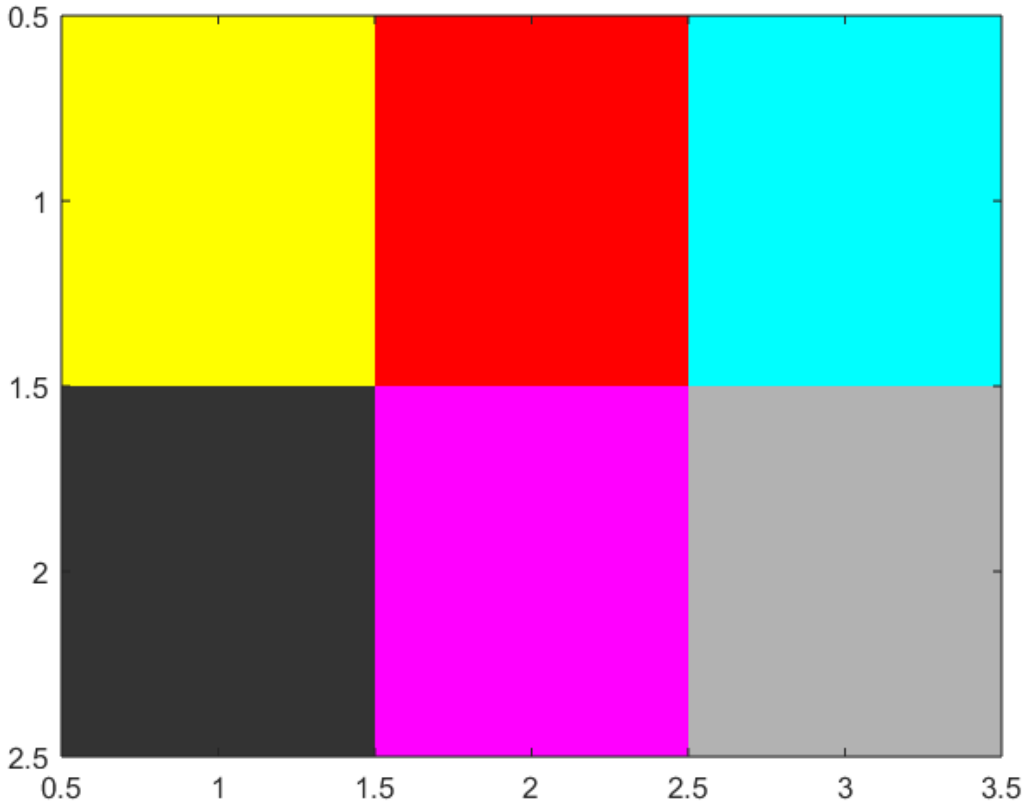
imshow(k2/255) denna blir nu rätt bild.
När man konverterade k till double, bildades en ny bild k2 där varje pixelvärde är ett tal mellan 0 och 255 (i double-format). När du delar k2 med 255, sker divisionen utan att avrundas till heltal eftersom k2 är av datatypen double.
Därmed behålls alla decimaler som resulterar från divisionen.

2) RGB färger och colormap

Uppgift 2.1)
Er MATLAB kod (**tre rader**) här:

```
map = [1 1 0; % Gult (Röd + Grön), upp vänster
1 0 0; % Röd, upp-mitten
0 1 1; % Cyan (Grön + Blå), upp höger
0.2 0.2 0.2; % Mörkgrå, ner vänster
1 0 1; % Magenta (Röd + Blå), ner-mitten
0.7 0.7 0.7]; % Ljusgrå, ner höger
image([1, 2, 3; 4, 5, 6]) % Rutmönster (2x3)
colormap(map)
```

Infoga bilden här:



3) Matriser och punktvis operation

Uppgift 3.1)

Det första kommandot >>N(1:3:end,1) kommer ge att var tredje rad (med början från första raden) i matrisen väljs men endast från kolumn 1. Det vill säga först 1 och sedan 10 som svar.

>> N(1:3:end, 1)=
Ans =

$$\begin{bmatrix} 1 \\ 10 \end{bmatrix}$$

Det andra kommandot >>N(1:3:end,:) kommer ge att var tredje rad (med början från första raden) i matrisen väljs men till skillnad från det tidigare kommandot så tas raderna från alla kolumner. Det vill säga 1, 2, 3, 10, 11, 12 kommer väljas ut.

>> N(1:3:end, :)=
Ans=

$$\begin{bmatrix} 1 & 2 & 3 \\ 10 & 11 & 12 \end{bmatrix}$$

Uppgift 3.2)

Förklara anledningen om du får ett felmeddelande i någon av raderna.
s1= 1
s2= Denna funkar ej då vektorerna inte är lika stora och då kan man inte använda elementvis multiplikation.

4) Logiska operationer

Uppgift 4.1)

$$u1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$u2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$u3 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$u4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

5) Färgbilder

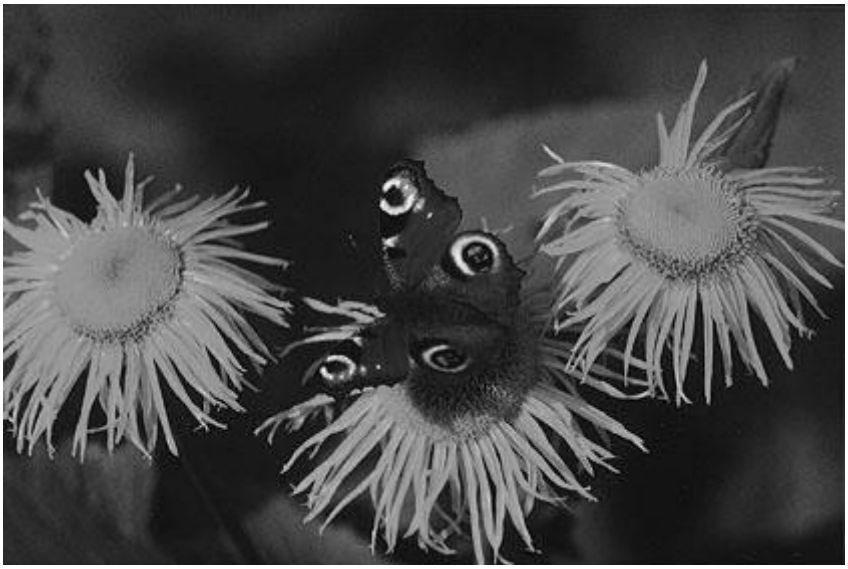
Uppgift 5.1)
MATLAB kommandon ni har använt för att skapa *mygray* här under (5-6 rader kod):

```
fargbild = imread('Butterfly.tif'); % Läs in bilden
fargbild = im2double(fargbild); % Konvertera till double och normera mellan 0 och 1
% Konvertera RGB till gråskalebild (genom att ta medelvärdet av R, G och B)
mygray = (fargbild(:, :, 1) + fargbild(:, :, 2) + fargbild(:, :, 3)) / 3;

% Visa den gråskalebild som skapats
imshow(mygray);

% spara den gråskalebild som en .png-fil
imwrite(mygray, 'mygray.png');
```

Infoga *mygray* här (skala inte bilden efter att ni har infogat):



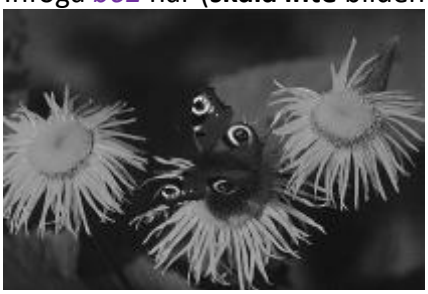
6) Nedsampling och uppsampling

Uppgift 6.1)
MATLAB kommandot här: (går med bara en rad)

```
b61 = mygray(1:2:end, 1:2:end);
```

Infoga *b61* här (**skala inte** bilden efter att ni har klistrat in i words):

Uppgift 6.2)
Infoga *b62* här (**skala inte** bilden efter att ni har klistrat in i words):



Uppgift 6.3)
MATLAB kommandon här (det krävs bara en rad per bild):

```
b63_nearest = imresize(b61, size(mygray), 'nearest');
b63_linear = imresize(b61, size(mygray), 'bilinear');
```

b63_cubic = imresize(b61, size(mygray), 'bicubic');

Infoga *b63_nearest*: (**skala inte** bilden efter att ni har klistrat in i words)

Infoga *b63_linear*: (**skala inte** bilden efter att ni har klistrat in i words)

Infoga *b63_cubic*: (**skala inte** bilden efter att ni har klistrat in i words)

Diskutera vilken av dessa tre bilder och på vilket sätt ser bättre ut och liknar originalet mest:

Uppgift 6.4)
MATLAB kommandon här: (max två rader)

```
b64_down = imresize(mycolorimage, 0.5, 'nearest');
b64_up = imresize(b64_down, 2, 'nearest');
```

Infoga *b64*:

Beskriv de tydliga skillnaderna mellan *mycolorimage* och *b64*:

mycolorimage:Eftersom detta är originalbilden, kommer den att ha alla sina detaljer intakta. Bilden kommer att vara skarp och innehålla mycket information i varje pixel, vilket gör att detaljer som färgnyanser, texturer och kanter bevaras.

b64: Efter att ha använt "nearest neighbor" interpolation för nedprovning, kommer b64 att vara suddigare och mindre detaljerad. "Nearest neighbor"-metoden fungerar genom att helt enkelt tilldela varje pixel i den nya bildens grid det värde från den närmaste pixeln i den ursprungliga bilden. Detta gör att övergångar mellan färger och detaljer inte är lika mjuka som i originalbilden. Den nedsamplade bilden kommer därför att ha ett mer pixligt utseende, där vissa fina detaljer går förlorade.

Uppgift 6.5)
MATLAB kommandon här: (max 10 rader men fullt möjligt med 6 rader)

```
R = mycolorimage(:, :, 1);
G = mycolorimage(:, :, 2);
B = mycolorimage(:, :, 3);

% Sampla ner R och B med faktor 0.5
R2 = imresize(R, 0.5, 'nearest');
B2 = imresize(B, 0.5, 'nearest');

% Sampla upp R2 och B2 med faktor 2 för att få tillbaka originalstorleken
R2 = imresize(R2, 2, 'nearest');
B2 = imresize(B2, 2, 'nearest');

% Återskapa den nyskapade bilden
b65 = cat(3, R2, G, B2);
```

Infoga *b65* här:

Vilken av bilderna *b64* och *b65* liknar originalet mest? Förklara varför!

B64: Nearest neighbor"-interpolation bevarar inte gradvisa färgövergångar, vilket kan skapa en mer blockig eller pixlig bild, särskilt när man samplar upp en nedsamplad bild. Därför kan detaljer i den ursprungliga bilden gå förlorade och övergångar mellan färger kan bli hårdare.

B65: Eftersom den gröna kanalen behålls i full upplösning, kommer b65 att ha mer detaljer och skärpa jämfört med b64, även om nedsamplingen och uppsamplingen av R och B kan ge en viss förlust av detaljer.

Alltså så liknar b65 originalet mest.

Uppgift 6.6)

Hur mycket minne utryckt i megabytes (MB) krävs för att spara en 4000×2000 pixlar stor färgbild i uint8-format? (skriv hur du räknat)

$4000\ x\ 2000 = 8\ 000\ 000\ pixlar$
$8\ 000\ 000\ pixlar\ x\ 3\ kanaler\ x\ 1\ byte = 24\ 000\ 000\ bytes = 24\ MB$

Hur mycket minne krävs för den komprimerade bilden om vi samplar ner två av bildens färgkanaler till hälften så stor i varje led men behåller den tredje som den var? (skriv hur du räknat)

G-kanalen: $4000\ x\ 2000\ pixlar\ x\ 1\ byte = 8\ 000\ 000\ bytes$
R- och B-kanalerna (samplade ner till hälften i varje led):
$2000\ x\ 1000\ pixlar\ x\ 1\ byte = 2\ 000\ 000\ bytes\ per\ kanal$
$8\ 000\ 000 + 2\ 000\ 000 + 2\ 000\ 000 = 12\ 000\ 000\ bytes = 12\ MB$

Uppgift 6.7)

MATLAB kommandon här: (max 7 rader men fullt möjligt med 3 rader)

```
% Sampla ner R och G med faktor 0.5
R3 = imresize(R, 0.5, 'nearest');
G3 = imresize(G, 0.5, 'nearest');

% Sampla upp R3 och G3 med faktor 2 för att få tillbaka originalstorleken
R3 = imresize(R3, 2, 'nearest');
G3 = imresize(G3, 2, 'nearest');

% Återskapa den nyskapade bilden
b67 = cat(3, R3, G3, B);
```

Infoga *b67* här:

Förklara varför *b65* ser bättre ut än *b67* (är mer lik originalet).

I b65 behålls G-kanalen i originalstorlek, medan R och B samplas ner och upp.
Eftersom G-kanalen bär mycket av detaljinformationen i bilden (då det är mycket grönt i bilden), är det att föredra att bevara den intakt.

I b67, däremot, samplas både R och G-kanalerna ner och upp, medan endast B-kanalen behålls intakt. Eftersom G-kanalen innehåller mycket av detaljinformation, som nämnt tidigare, så förloras viktiga detaljer i b67. Detta resulterar i att den inte liknar originalbilden lika mycket som b65 gör.

Uppgift 6.8)
MATLAB kommandon här: (max 13 rader men fullt möjligt med 9 rader)

```
% Skapa de tre nya bilderna
bild1 = R + G + B;
bild2 = R - G;
bild3 = R + G - 2*B;

% Sampla ner bild2 och bild3 med faktor 0.5
bild2_down = imresize(bild2, 0.5, 'nearest');
bild3_down = imresize(bild3, 0.5, 'nearest');

% Sampla upp bild2 och bild3 med faktor 2 för att återställa till originalstorlek
bild2_up = imresize(bild2_down, 2, 'nearest');
bild3_up = imresize(bild3_down, 2, 'nearest');

% Återskapa R, G och B-kanalerna
R_new = (bild1/3) - (bild2_up/2) + (bild3_up/6);
G_new = (bild1/3) - (bild2_up/2) + (bild3_up/6);
B_new = (bild1/3) - (bild3_up/3);

% Kombinera de återställda R, G och B-kanalerna för att skapa den nya färgbilden
```

b68 = cat(3, R_new, G_new, B_new);

Infoga *b68* här:

Ser ni att *b68* ser bättre ut än *b65* och är nästan identisk originalbilden?

Uppgift 6.9)

Hur mycket minne krävs för B1 (utryckt i MB)? (skriv hur du räknat)

Originalbilden $B1$ har storleken $4000\ x\ 12\ 000\ pixlar$ och består av tre kanaler (RGB), där varje pixel i varje kanal tar 8 bitar (1 byte).

Totalt antalet pixlar $= 4000\ x\ 12\ 000 = 48\ 000\ 000\ bytes$

Totalt minne för B1 $= 48\ 000\ 000\ x\ 3 = 144\ 000\ 000\ bytes = 144\ MB$

Hur mycket minne krävs för B2 (utryckt i MB)? (skriv hur du räknat)

För bild B2, som är nedsamplad med faktor 0.25 (det vill säga att bildens storlek blir $4000\ x\ 12\ 000\ x\ 0.25 = 1000\ x\ 3000\ pixlar$), gäller följande:

Efter nedsampling:

$4000\ x\ 0.25 = 1000\ rader$

$12\ 000\ x\ 0.25 = 3000\ kolumner$

Totalt antalet pixlar $= 1000\ x\ 3000 = 3\ 000\ 000\ pixlar$

Totalt minne $= 3\ 000\ 000\ x\ 3 = 9\ 000\ 000\ bytes = 9\ MB$

Hur mycket minne krävs för B3 (utryckt i MB)? (skriv hur du räknat)

Eftersom bilinjär interpolation inte påverkar mängden data som lagras (den ändrar bara interpoleringen mellan pixlar), kommer storleken på B3 vara samma som för B2. (Beräknat på samma sätt som ovan).

Totalt minne för B3$= 9\ 000\ 000\ bytes = 9\ MB$

Glöm inte att spara dokumentet som *.pdf* innan ni lägger ut det på Lisam.