Python Mini Projekt

Rumena Marinova-Pedersen

Projektbeskrivelse

Til dette project valgte jeg at bruge Python til billedredigering, og at lave et tekst-baseret billedredigeringsprogram. Mit program skal kunne de følgende populære/almindelige redigeringsfunktioner:

1. Resize
2. Crop (også center crop)
3. Rotate
4. Flip (lodret og vandret)
5. Greyscale
6. Darken/lighten brightness
7. Ændre på kontrasten
8. Blur
9. Sharpen
10. Og filtre: Outline, 2-bit Image og Enhance edges

Brugeren skal kunne vælge et billede ud fra et udvalgt galleri, manipulere billedet ved at vælge en funktion ud fra menuen (og evt. indtaste input såsom størrelse, eller blur-factor) og efterfølgende gemme den redigerede kopi, hvis han/hun vil det.

Planlægning/tackling af projektet

1. I starten brugte jeg et stykke tid for at researche/kigge på teorien: primært læste jeg om Python PIL (Python Imaging Library) modulet, og hvordan den kan bruges for at åbne, redigere og gemme billeder i Python. Her er nogle af kilderne jeg brugte:

- Python Pillow modul dokumentation:

https://pillow.readthedocs.io/en/latest/handbook/index.html

- Mere om Pillow:

https://neptune.ai/blog/pil-image-tutorial-for-machine-learning

- NumPy modul dokumentation:

https://numpy.org/doc/stable/reference/generated/numpy.array.html

- NumPy shape method (bliver brugt i greyscaling):

https://www.digitalocean.com/community/tutorials/python-shape-method

- Mere om billederedigering:

https://scikit-image.org/docs/stable/auto_examples/index.html

- Lambda function call (brugt i outline-filter funktionen):

https://opensource.com/article/19/10/python-programming-paradigms

Det vigtigste at huske i starten var, at hver billede uanset af dets størrelse (100 x 100, 480 x 640, 1024 x 768, 1920 x 1080, eller noget helt andet) i komputerens øjne består bare af en 2D matrice, hvor hver pixel har sine egne x- og y-koordinater. For eksempel vil et billede med størrelse 5 x 5 pixels blive gemt i hukommelsen som en sådan matrice:

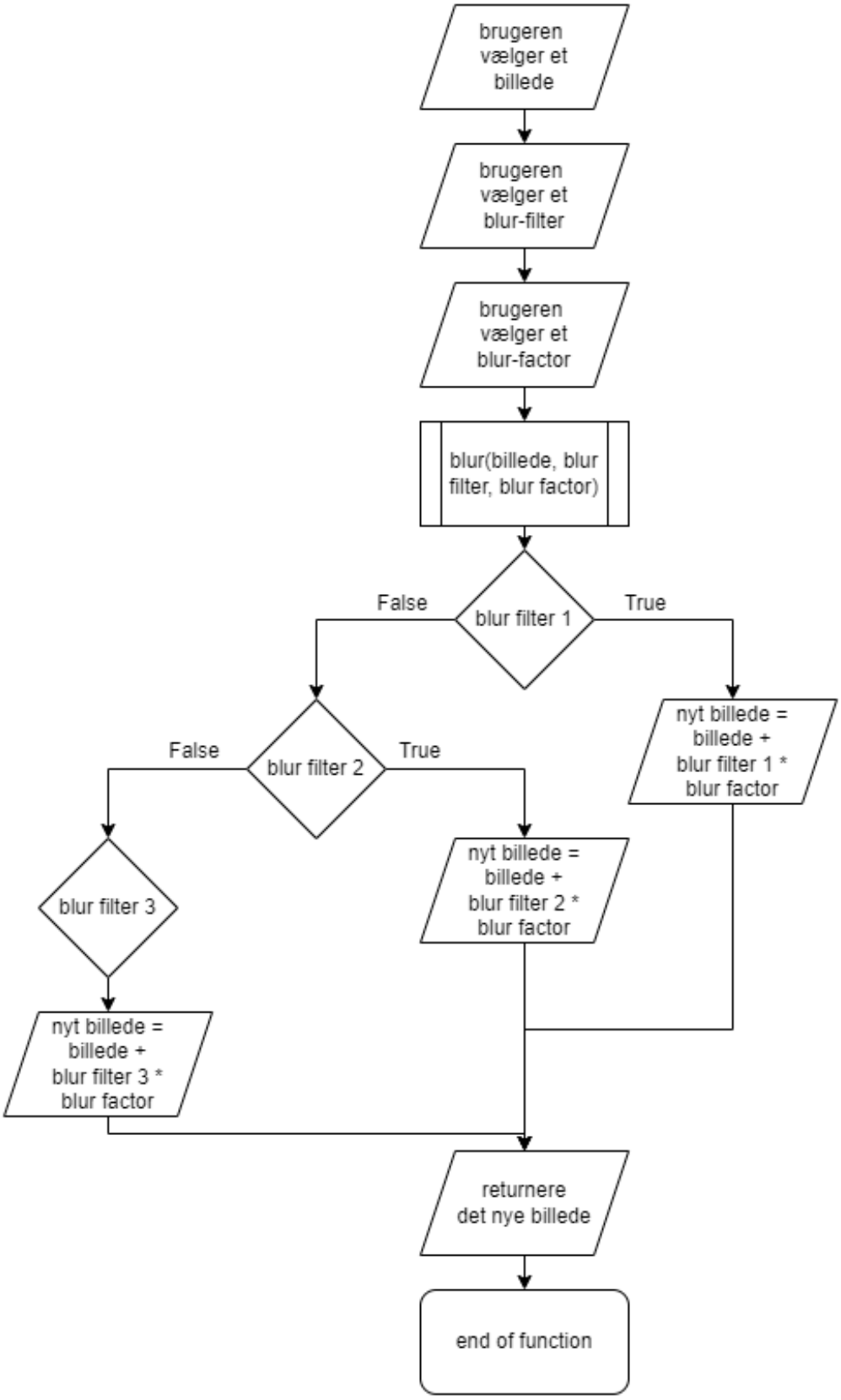
	A	B	C	D	E
1	pixel	pixel	pixel	pixel	pixel
2	pixel	pixel	pixel	pixel	pixel
3	pixel	pixel	pixel	pixel	pixel
4	pixel	pixel	pixel	pixel	pixel
5	pixel	pixel	pixel	pixel	pixel

Et farvebillede har faktisk 3 lag af 2D-matricer. Disse lag kaldes farvekanaler eller farvebånd. Hvilket betyder, at hver pixel har 3 værdier fra 3 kanaler (R,G og B). På denne måde får farvebilleder deres farve. Hvordan adskiller gråtonede billeder sig så? Et gråtonebillede har kun 1 farvelag, så det er kun et enkelt 2D-matrice.

Nogle af de indbyggede funktioner i PIL modulet krævede arrays fra NumPy modulet. NumPy-arrays har en fast størrelse ved oprettelsen, i modsætning til Python-lister (som er mutable og kan vokse dynamisk). Numpy-arrays letter avancerede matematiske og andre typer operationer på et stort antal data, derfor foretrækkes de ved billederedigering. Typisk udføres sådanne operationer mere effektivt og med mindre kode, end det er muligt ved brug af Pythons indbyggede sekvenser (lister osv.).

Lambda-funktioner er små, navnløse funktioner, der kan tage et vilkårligt antal argumenter, men kun 'spytter' én værdi ud. De er nyttige, når de bruges som argument for en anden funktion eller ligger inde i en anden funktion; derfor er de beregnet til kun at blive brugt i et sted ('one instance only') ad gangen.

Efter jeg fik teorien på plads begyndte jeg at skrive nogle forskellige redigeringsfunktioner i form af Python funktioner. Her er et eksempel på hvordan jeg skrev en funktion, som tilføjer et ud af 3 forskellige blur-filtre til det billede brugeren har valgt:



Efter et stykke tid havde jeg skrevet 16 funktioner:

- **read_img()** - hvor input er path, altså stien til billedet valgt af brugeren, og output er billedet returneret som PIL objekt
- **save_img()** - input er vores redigerede billede, og output er i formen af tekst, som notifikerer os om programmet har gemt billedet, eller ej
- **resize_img()** - input er billedet, ønskede højde og bredde i pixels, output er billedet i den ændrede størrelse
- **crop_img()** - input er billedet, og de 4 størrelser i pixels (venstre, top, højre, bund) af vores crop; funktionen returnerer det beskærede billede
- **center_img()** - input er vores billede, output er 4 tal (venstre, top, højre, bund), returneret som tuple. Hvis man bruger de 4 tal i crop-funktionen, får man et centrebeskærede billede, hvor vi har skåret 25% væk i hver retning.
- **rotate_img()** - input er vores billede og den ønskede vinkelstørrelse; funktionen returnerer det drejede billede (obs, at billedet drejes *mod* uret).
- **greyscale_img()** - input er billedet, output er billedet i greyscale.
- **binary_img()** - input er billedet og en threshold (grænse). Funktionen konverterer billedet til greyscale og bruger en array (genereret via numpy) fra det nye billede. Efterfølgende gennem en dobbel for-løkke går vi igennem hver eneste pixel (element) i vores array, og tjekker om den er større end vores grænse (i det tilfælde bliver den konverteret til ren hvid farve), eller mindre (i det tilfælde bliver vores pixel en ren sort farve). Funktionen returnerer et nyt billede, lavet ud af vores array som nu kun består af enten ren sort eller hvid farve.
- **flip_horiz_img()** - input er billedet, output er billedet vendt vandret.
- **flip_vert_img()** - input er billedet, output er billedet vendt lodret.
- **brightness_img()** - input er billedet og et float-tal, som bestemmer hvor lysere (factor > 1.0) eller mørkere (0.0 <= factor < 1.0) det nye billede bliver. En factor af 1.0 giver det originale billede. Funktionen returnerer billedet med den ændrede lysstyrke.
- **contrast_img()** - input er billedet og et float-tal, som bestemmer hvor meget stærkere (factor > 1.0) eller blødere (0.0 <= factor < 1.0) kontrasten bliver. En factor af 1.0 giver det originale billede. Funktionen returnerer billedet med den ændrede kontrast.
- **blur_img()** - Funktionen givet i mit eksempel på d. sidste side. Input er billedet, og to integers - den første (1, 2 eller 3) bestemmer hvilken blur bliver brugt i funktionen, og den anden integer bestemmer hvor 'kraftig' blur'en bliver. Det nye billede bliver returneret af funktionen.
- **shapren_img()** - input er billedet og et float-tal som bruges til vores *sharpness factor* (1.0 giver det oprindelige billede). Funktionen returnerer det redigerede billede.
- **outlines_filter()** - input er billedet og en integer threshold (grænse). Først bruger funktionen den indbyggede i PIL funktion FIND_EDGES, og efterfølgende vi bruger .split() for at dele billedet op i dets farvebånd (R, G og B). Derefter bruger vi det første element af båndene (bands[0]) med en lambda function call for at sætte hver pixel (hver element i det bånd) til enten sort eller hvid, som giver en virkelig ren

outline, der ligner en skitse-filter. Funktionen returnerer det outlinede billede til sidst. Funktionen og måden på hvilken den fungerer lyder lidt som binary_img()-funktionen, men i virkeligheden er deres output markant forskelligt - nedenunder kan vi se output fra binary (top) og outlines (bund) begge med threshold = 100:

- **edges_img()** - input er billedet, output er det redigedere billede, hvor vi har brugt EDGE_ENHANCE filteret fra PIL-modulet.

Jeg fik alle de individuelle funktioner testet, og efter jeg konfirmerede, at de fungerede som de skulle, var jeg klar til trin 2:

2. I dette trin skulle jeg prøve at organisere funktionerne i en menu, der giver mening for brugeren.

Først delte jeg de ovennævnte funktioner op ifølge denne logik:

	A	B	C	D
1	**Functional**	**Edits**	**Manipulations**	**Filters**
2	read_img()	resize_img()	greyscale_img()	binary_img()
3	save_img()	crop_img()	brightness_img()	outlines_filter()
4		center_img()	contrast_img()	edges_img()
5		rotate_img()	blur_img()	
6		flip_horiz_img()	sharpen_img()	
7		flip_vert_img()		

Derefter lavede jeg en plan i form af pseudokode, og ud fra den blev der udarbejdet programmets endelige struktur, og selve Python koden. Her er en del af programmet i pseudokode. Den blev også brugt til flowcharten, som er vedlagt denne opgave.

- Programme starts
- Let the user choose an image from a menu
- The chosen image determines the image path
- Read the image as a PIL object from the path
- Pass the image to the menu options: Edit, Manipulate or Filter
- User choses one of the options
- (F.ex the user has chosen Filters):
- Ask the user if he wants to use: binary filter, outlines filter, edge enhancement filter or go back to the previous menu
- (F.ex. the user choses binary filter):
- Ask the user to input the threshold value
- Run the binary_img() function with the selected image + threshold as input
- Save the output of the functions in a new variable
- Show the image
- Run the save_img() function
- …
- …
- End of programme

Konklusion

Vores mini Python projekt var en virkelig sjov og lærerig udfordring, og en rigtig god måde at koble alle de ting vi har lært i løbet af de sidste 3 måneder sammen, og afslutte semesteret på.

Det sværste for mig var nok at vælge kun én af mine ideer i starten, da jeg havde rigtig mange af dem. Dette kommer selvfølgelig an på personen, men jeg fungerer bedre når der er strengere rammer/ mere definerede regler fra starten af, og så kan jeg vælge hvordan at være kreativ i dem.

Som ethvert stykke software blev mit program ikke 'færdiggjort', derfor er denne udgivelse version 0.1. Jeg har allerede lavet en liste over de ting jeg gerne vil tilføje til softwaren:

- Fortæl brugeren om de mulige værdier når han/hun bliver spurgt om at taste ind input.
- Forklaringer for værdierne! F.eks. 0 = sort/mørk, 255 = hvid/lys osv.
- Tjek invalid/out of range input fra brugeren *alle steder*. F.eks. hvis værdien skal være mellem 0 og 255, og brugeren taster ind -10, eller et ord... osv.
- Implementere muligheden for at bruge flere edits på det samme billede - f.eks. skær billedet, og så ændre på kontrast og lysstyrke.
- Implementere Sobel filter (https://en.wikipedia.org/wiki/Sobel_operator). Jeg begyndte at researche den, men jeg skal læse lidt mere grundigt før jeg kan prøve at skrive min egen Sobel filter function.
- Implementere muligheden for custom input billede - altså at brugeren kan selv uploade/vælge et billede, som ikke findes i programmet endnu.
- Scalable resize/crop - så at brugeren ikke skal taste ind alle værdierne, men kan f.eks. bare skriv, at det redigerede billede skal være 1000 px i bredden, eller 30% af den originale bredde osv.
- GUI!!! Måske det hårdeste, men vigtigste at implementere.

Jeg glæder mig til at forstætte med at arbejde på mit lille projekt i min fritid, og jeg er sikker på, at inden længe bliver version 1.0 uploadet på min GitHub profil. :)