Základy a aplikace počítačového vidění (KIP/2GZAV)
Semestrální úkol – distanční a kombinované studium

Aplikace pokročilých filtrů pro zapracování obrazu pomocí OpenCL

Petr Velčovský velcpe68@osu.cz | Aplikovaná informatika | LS 2024/25
https://github.com/petrvelcovsky/2GZAV_OpenCv-Filters

Zadání:

Vypracujte tyto body zadání:

1. Základní filtry (jako základ pro porovnání s pokročilými filtry)
    a. průměrování pro odstranění šumu
    b. rozmazání pomocí Gaussova rozmazání
    c. odstranění impulzního šumu (bílé a černé tečky) pomocí mediánového filtru
2. Pokročilé filtry a efekty
    a. bilaterální filtr
    b. rozmazání pohybu
    c. filtr napodobující malířské styly
3. Implementace filtrů napodobující malířské styly na obrázky a na video stream v reálném čase

---

Vypracování:

1a.

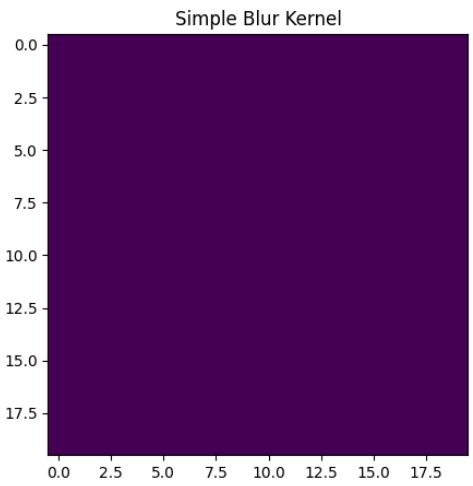
Součástí knihovny OpenCV je několik základních filtrů pro úpravu a vylepšení obrazu. Použití těchto základních filtrů v OpenCV je jednoduché.

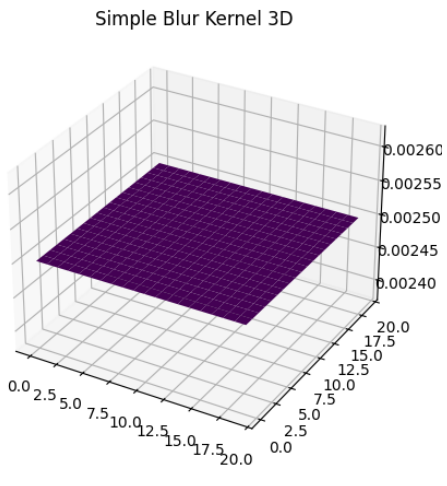
Základní konvoluční filtr používá pro výpočet hodnoty aktuálního pixelu průměrování pixelů okolních hodnot. Jedná se o jednoduchý filtr, který rozmazává okolí bez ohledu na to, zda jde o hrany či plochy.

Příklad: pro matici 3x3 ... $K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

V OpenCV je filtr realizován funkcí: `cv2.blur(image, kernel_size, kernel_size)`,

kde parametr kernel_size je počet řádků/sloupců konvoluční (v našem případě čtvercové) matice. Tento parametr ovlivňuje míru rozmazání – čím větší konvoluční matice, tím větší rozmazání (příklad: 20x20)





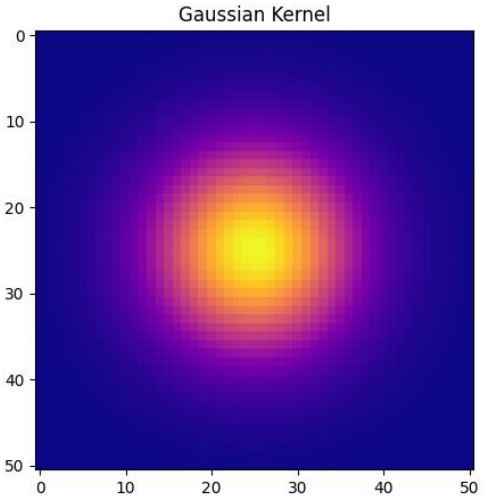
Výsledky použití průměrovacího filtru: (bez filtru; 20x20; 50x50):

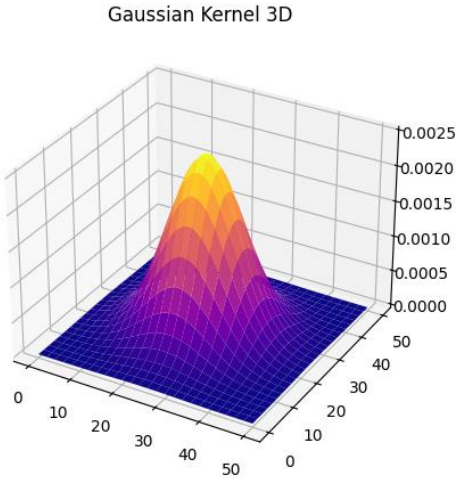
1b.

Gaussovské rozmazání funguje v podstatě na stejném principu jako předchozí typ, kdy k výpočtu pixelu použijeme hodnot okolních pixelů, a korát s tím rozdílem, že váhy matice jsou rozděleny podle Gaussovy křivky – největší váhu má pixel, který je uprostřed a odpovídá právě počítanému pixelu.

V OpenCV je filtr realizován funkcí: `cv2.GaussianBlur(image, (n, n), sigma)`,

kde n je šířka/výška konvoluční matice – hodnoty musí být liché, jin kprogram zahlásí chybu – v takovém případě nelze vypočítat střední bod, sigma je směrodatná odchylka, která určuje tvar gaussovy plochy (větší číslo – plocha je „tlustší“) – při 0 je velikost vypočítána klasická gaussova plocha podle velikosti jádra.





Výsledky použití gaussova filtru: (bez filtru; n = 51, sigma = 0; n = 51, sigma = 20):

1c.

Posledním základním filtrem, který jsem vybral, je mediánový filtr, který je účinný proti šumu (salt and pepper) v obraze. Funguje tak, že pro každý pixel se opět vezme jeho okolí a vypočítá se z nich medián (střední hodnota seřazených prvků). A tato hodnota nahradí původní pixel.

V OpenCV je filtr realizován funkcí: `cv2.medianBlur(image, n)`,

kde n je velikost okolí, ze kterého se počítá medián.

Výsledky použití mediánového filtru: (původní obrázek bez vad; náhodně přidané bílé a černé pixely s pravděpodobností 5%; aplikace filtru pro n = 5):

2a.

Bilaterální filtr patří již k pokročilým filtrům a používá se zejména, pokud chceme odstranit šum, ale nechceme, aby došlo k rozmazání hran. Na rozdíl od základních filtrů, jako je Gaussův nebo průměrovací filtr, bere bilaterální filtr v úvahu nejen polohu pixelů (prostorová blízkost), ale i jejich barevnou podobnost (intenzitu). Díky tomu dokáže velmi efektivně zachovat hrany, které by jinak běžné rozostřovací filtry rozmazaly.

V OpenCV je filtr realizován funkcí: `cv2.bilateralFilter(image, d, sigma_color, sigma_space)`,

kde d je velikost okolí, sigma_color je tolerance rozdílu barev (čím větší tolerance, tím pravděpodobnější rozmazání hran), sigma space je rozsah prostoru (čím větší tolerance, tím více pixelů se zohlední).

Výsledky použití bilaterálního filtru: (původní obrázek; d = 25, sigma_color = 100, sigma_space = 100):

2b.

Pro poslední filtr jsem si na webu našel zajímavý nápad – rozmazání obrázku, aby v pozorovateli vyvolalo pocit, že snímek byl pořízen v pohybu. Opět se jedná pouze o aplikaci filtru. Tentokrát konvoluční matice rozmazává snímek pouze v požadovaném směru. Konvoluční pro horizontální a vertikální směr vypadají takto (pokud je matice větší, vždy je pouze střední řádek/sloupec roven jedné):

$$K_{horizontal} = \frac{1}{3}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad K_{vertical} = \frac{1}{3}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

V OpenCV jsou filtry realizovány funkcemi:

`cv2.filter2D(image, -1, k1)` resp. `cv2.filter2D(image, -1, k1)`,

kde -1 je hloubka výstupního obrazu (-1 zachovává původní počet kanálů)

k1:

```
k1 = np.zeros((pixels, pixels))
k1[int(pixels/2), :] = np.ones(pixels)
k1 = k1 / pixels
```

k2:

```
k2 = np.zeros((pixels, pixels))
k2[ :, int(pixels/2)] = np.ones(pixels)
k2 = k2 / pixels
```

a pixels je délka/šířka matice pixelů použitých pro rozmazání (průměrování).

Výsledky použití 2D filtru: (původní obrázek; horizontální rozmazání k1, pixels = 25, vertikální rozmazání k1, pixels = 25):

3.

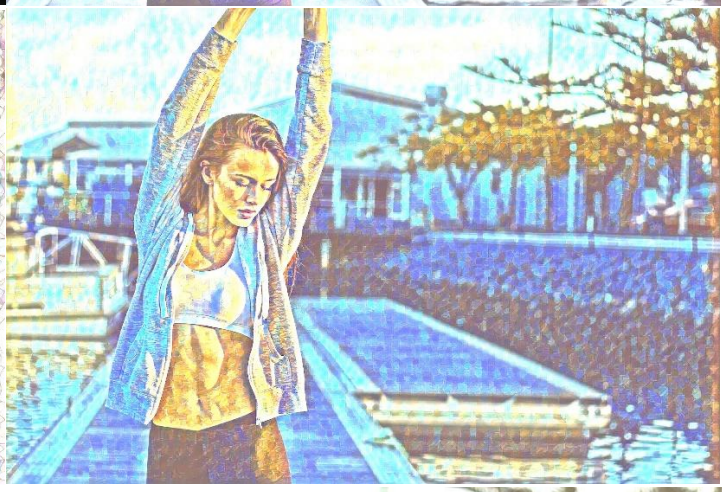
Na závěr jsem využil možnosti OpenCV pracovat s TensorFlow knihovnami. Jedna z oblíbených funkcí TensorFlow ja přenos malířského stylu na libovolný snímek. Samotné OpenCV nepodporuje možnost trénování libovolného malířského stylu. Nicméně na Githubu se dá najít množství předtrénovaných modelů uložených ve formátech, které OpenCV podporuje. Tyto lze načíst pomocí modulu `cv2.dnn()` bez nutnosti instalace TensorFlow. To umožňuje efektivní aplikaci stylu v reálném čase i na běžném počítači. Pomocí těchto modelů lze snadno přenášet různé malířské styly na snímky z kamery nebo uložené fotografie. Celý proces probíhá v několika krocích: načtení snímku, vytvoření vstupního „blobu" (standardizovaného formátu a velikosti snímku, se kterým OpenCV dokáže pracovat), výpočet pomocí modelu a převedení výstupu zpět do zobrazitelného formátu.

Pro svou práci jsem našel na Githubu (*https://github.com/jcjohnson/fast-neural-style*) tyto čtyři modely:

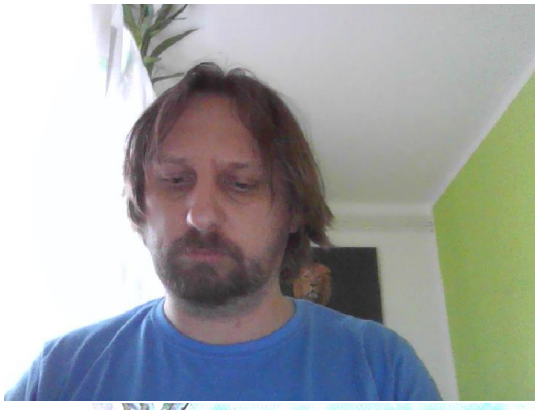
Candy (obraz malířky Natashi Wescoat– Červnový strom), Mosaic (snímek mozaiky), Rain princess (obraz Dešťová princezna malíře Leonida Afremova) a Udnie (obraz Udnie malíře Francise Picabia)

Všechny modely byly uloženy ve formátu *.onnx, který je rovněž OpenCV podporován:
`cv2.dnn.blobFromImage(image, 1.0, (w, h),(103.939, 116.779, 123.680), swapRB=False, crop=False)`
`cv2.dnn.readNetFromONNX(model_path)` , kde číselné hodnoty jsou normalizacemi obrazů BGR

Výsledky (použité na snímky):



Výsledky (použité na video stream):



Závěr.

Jak jsem uvedl na začátku, práce s filtry a obecně s úpravami obrázků v OpenCV je velmi snadná a intuitivní. Parametry a krátké vysvětlení ke každému filtru se dají dohledat na stránkách dokumentace k OpenCV (*https://docs.opencv.org/4.x/*). Pro pochopení a zíkání základů mi velmi dobře posloužil úvodní kurz , který jsem byl schopen zvládnout během několika hodin.
Znalosti získané během kurzu mi umožnily nejen experimentovat s různými základními filtry a efekty, ale také si vyzkoušet pokročilejší funkce využívající základy umělé inteligence prostřednictvím knihovny TensorFlow (respektive alternativně přes modely *.onnx použitelné v OpenCV).
Díky tomu, že práce s filtry implementovanými přímo do OpenCV je snadná , tak nejvíce práce mi zabrala práce spojená s vyhledáváním a následným propojením modelů pro změnu stylu obrázků a videí.
Z původního plánu porovnat různé parametry nastavení pro jednotlivé filtry jsem nakonec ustoupil, jednak protože by prezentace byla zbytečně dlouhá, ale především proto, že vhodné parametry bývají vždy závislé na konkrétním typu snímku, zamýšleném použití a zejména na osobních preferencích ohledně očekávaného výsledku.