Semestrální projekt

Segmentace a prahování obrazu

Vypracoval : Lískovský David

Předmět : 2GZAV

Datum : 04.04.2024

Obsah

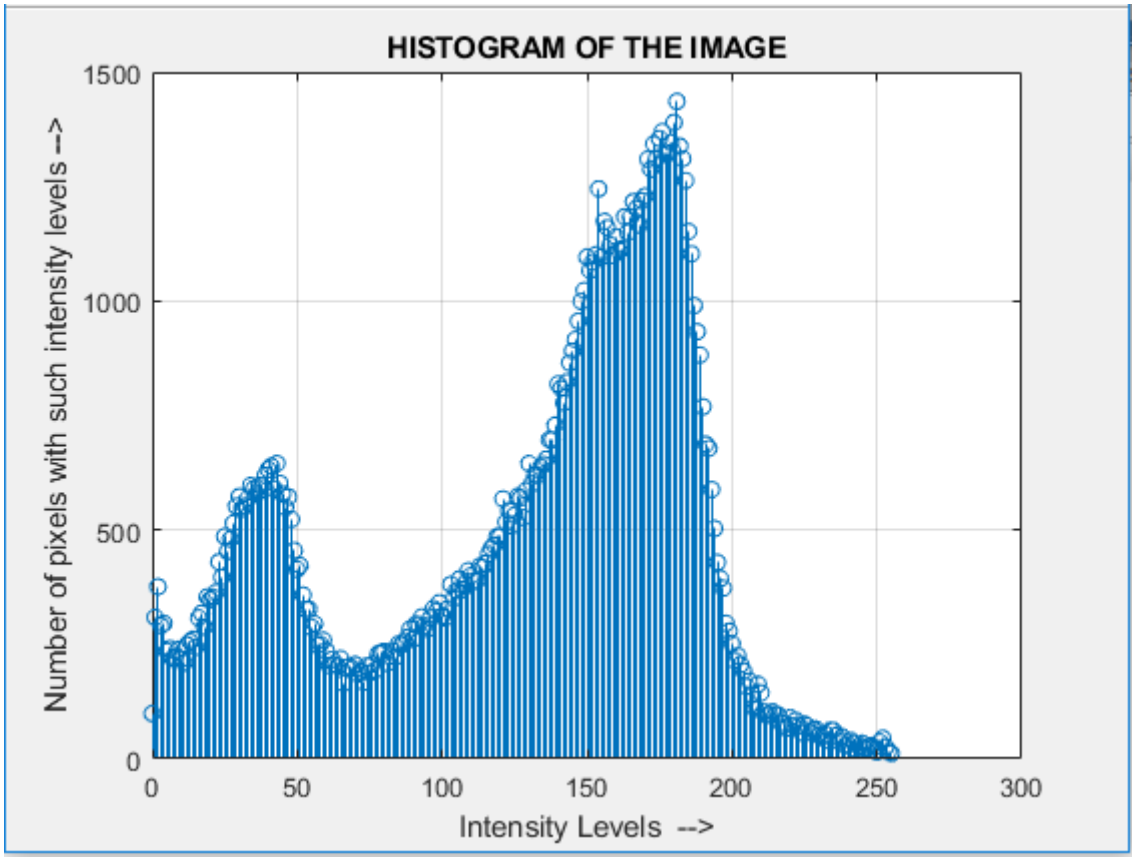
Segmentace obrazu

Jde o techniku rozdělování obrazu do dílčích oblastí nebo odlišných objektů. Míra podrobnosti, do které je dělení provedeno, závisí na řešeném problému. To znamená, že segmentace by se měla zastavit, když byly detekovány objekty nebo oblasti zájmu v aplikaci.

Segmentace netriviálních snímků je jedním z nejobtížnějších úkolů při zpracování obrazu. Přesnost segmentace určuje případný úspěch nebo selhání postupů počítačové analýzy. Segmentační postupy se obvykle provádějí pomocí dvou přístupů – detekce diskontinuity v obrazech a propojení hran za účelem vytvoření oblasti (známé jako segmentace na základě hran) a zjišťování podobnosti mezi pixely na základě úrovní intenzity (známé jako segmentace založená na prahu).

Prahování

Thresholding je jedna z technik segmentace, která generuje binární obraz (binární obraz je ten, jehož pixely mají pouze dvě hodnoty – 0 a 1, a proto vyžaduje pouze jeden bit k uložení intenzity pixelů) z daného obrázku ve stupních šedi jeho rozdělením do dvou oblastí. na základě prahové hodnoty. Pixely, které mají hodnoty intenzity vyšší než uvedený práh, budou tedy ve výstupním obrazu považovány za bílé nebo 1 a ostatní budou černé nebo 0.



Obr. 1 - Histogram obrázku

Předpokládejme, že výše uvedený diagram je histogram obrázku ***f(x,y)***. Můžeme vidět jeden vrchol poblíž úrovně 40 a druhý na úrovni 180. Existují tedy dvě hlavní skupiny pixelů – jedna skupina se skládá z pixelů s tmavším odstínem a ostatních se světlejším odstínem. V pozadí tedy může být zasazen objekt zájmu. Pokud použijeme vhodnou prahovou hodnotu, řekněme 90, rozdělí celý obraz na dvě odlišné oblasti.

Jinými slovy, pokud máme práh ***T***, pak se segmentovaný obrázek ***g(x,y)*** vypočítá, jak je uvedeno níže:

$$\boldsymbol{g(x,y) = 1, pokud\ \ f(x,y) > T}$$

$$\boldsymbol{g(x,y) = 0, pokud\ \ f(x,y) \leq T}$$

Výstupní segmentovaný obrázek má tedy pouze dvě třídy pixelů – jedna má hodnotu 1 a další mají hodnotu 0. Pokud je práh ***T*** konstantní při zpracování v celé oblasti obrazu, říká se, že jde o globální prahování. Pokud se ***T*** v oblasti obrazu mění, říkáme, že jde o proměnné prahování.

Multiple-thresholding klasifikuje obraz do tří oblastí – jako dva odlišné objekty na pozadí. Histogram v takových případech ukazuje tři vrcholy a dvě údolí mezi nimi. Segmentovaný obraz může být dokončen pomocí dvou vhodných prahových hodnot ***Ti a T2*** .

$$\boldsymbol{g(x,y) = a, pokud\ \ f(x,y) > T2}$$

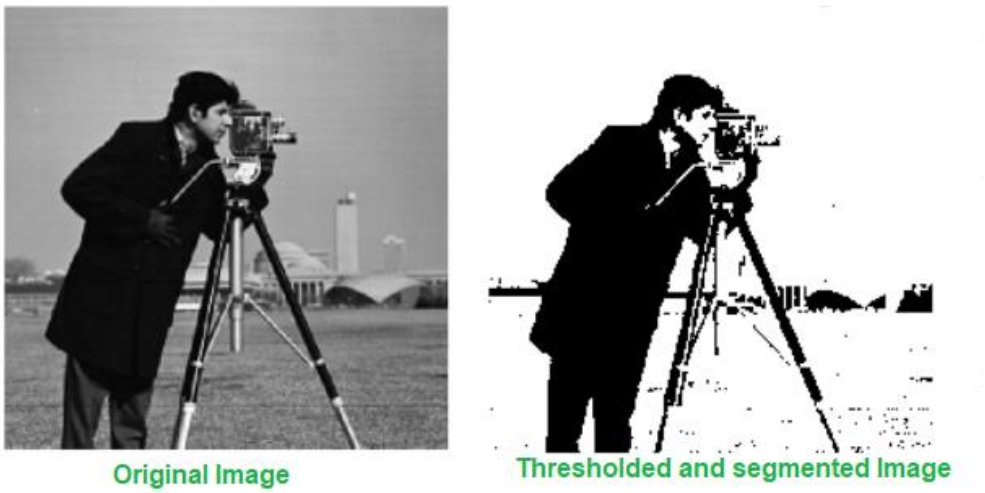
$$\boldsymbol{g(x,y) = b, pokud\ \ T1 < f(x,y) \leq T2}$$

$$\boldsymbol{g(x,y) = c, pokud\ \ f(x,y) \leq T1}$$

kde ***a, b a c*** jsou tři různé hodnoty intenzity.

Z výše uvedených ukázek můžeme intuitivně vyvodit, že úspěch prahování intenzity je přímo spojen s šířkou a hloubkou údolí oddělujících módy histogramu. Klíčovými faktory ovlivňujícími vlastnosti těchto údolí jsou rozdělení mezi vrcholy, obsah šumu v obraze a relativní velikosti objektů a pozadí. Čím více jsou dva vrcholy v histogramu odděleny, tím lépe budou fungovat algoritmy pro prahování, a tedy segmentaci obrazu. Šum v obraze často degraduje tuto široce oddělenou dvouvrcholovou distribuci histogramu a vede k obtížím při adekvátním prahování a segmentaci. Když je přítomen šum, je vhodné použít nějaký filtr na vyčištění obrazu a poté aplikovat segmentaci. Relativní velikosti objektů hrají roli při určování přesnosti segmentace.

Globální prahování



Obr. 2 - Rozdíl prahového nastavení

Když je rozložení intenzity objektů a pozadí dostatečně zřetelné, je možné použít jedinou nebo globální prahovou hodnotu použitelnou na celý snímek. Základní algoritmus globálního prahování iterativně najde nejlepší prahovou hodnotu, takže segmentuje.

Algoritmus

1. Vyberte počáteční odhad prahové hodnoty T.
2. Segmentujte obrázek pomocí T tak, abyste vytvořili dvě skupiny $G_1 a\ G_2$: G_1 se skládá ze všech pixelů s hodnotami intenzity $> T$ *a* G_2 se skládá ze všech pixelů s hodnotami intenzity $\leq T$.
3. Vypočítejte průměrné hodnoty intenzity $m_1\ a\ m_2$ pro skupiny $G_1 a\ G_2 * \sigma$
4. Vypočítejte novou hodnotu prahu T jako $T = (m_1 + m_2)/2$
5. Opakujte kroky 2 až 4, dokud nebude rozdíl v následné hodnotě T menší než předem definovaná hodnota δ.
6. Segmentujte obrázek jako $g(x,y) = 1, pokud\ \ f(x,y) > T$
$g(x,y) = 0, pokud\ \ f(x,y) \leq T$

Tento algoritmus funguje dobře pro obrázky, které mají v histogramu jasné údolí. Čím větší je hodnota δ, tím menší bude počet iterací. Počáteční odhad T může být roven průměrné intenzitě pixelů celého obrazu.

Výše uvedené jednoduchého globálního prahování lze optimalizovat pomocí Otsuovy metody. Otsuova metoda je optimální v tom smyslu, že maximalizuje rozptyl mezi třídami. Základní myšlenkou je, že dobře nastavené třídy nebo skupiny by měly být odlišné s ohledem na hodnoty intenzity jejich pixelů a naopak, práh poskytující nejlepší oddělení mezi třídami z hlediska jejich hodnot intenzity by byl nejlepší nebo optimální práh.

Variabilní prahování

Existují zhruba dva různé přístupy k místnímu prahování. Jedním z přístupů je rozdělení obrazu na nepřekrývající se obdélníky. Poté jsou na každý z dílčích obrazů aplikovány techniky globálního prahování nebo Otsuova metoda. V technice dělení obrazu se tedy metody globálního prahování aplikují na každý obdélník dílčího obrazu za předpokladu, že každý takový obdélník je samostatný obraz sám o sobě. Tento přístup je oprávněný, když jsou vlastnosti histogramu dílčího obrazu vhodné (mají dva píky s širokým údolím mezi nimi) pro aplikaci prahovacích technik, ale celý histogram obrazu je poškozen šumem, a proto není ideální pro globální prahování.

Druhou možností je vypočítat proměnnou prahovou hodnotu v každém bodě z vlastností sousedních pixelů. Řekněme, že máme okolí $\boldsymbol{S_{xy}}$ pixelu se souřadnicemi (x,y). Pokud je průměr a směrodatná odchylka intenzit pixelů v tomto okolí $\boldsymbol{m_{xy}}$ $\boldsymbol{a}$ $\boldsymbol{\sigma_{xy}}$, pak lze práh v každém bodě vypočítat jako:

$$\boldsymbol{T_{xy} = a_{xy} + bm_{xy}}$$

kde $\boldsymbol{a}$ a $\boldsymbol{b}$ jsou libovolné konstanty. Výše uvedená definice proměnné prahové hodnoty je pouze příkladem. Podle potřeby lze použít i jiné definice.

Segmentovaný obrázek se vypočítá takto:

$$\boldsymbol{g(x,y) = 1, pokud\ \ f(x,y) > T_{xy}}$$
$$\boldsymbol{g(x,y) = 0, pokud\ \ f(x,y) \leq T_{xy}}$$

Jako prahové hodnoty lze také použít klouzavé průměry. Tato technika prahování obrazu je nejobecnější a lze ji aplikovat na velmi různé případy.

Praktická část

1. Načtení obrázku

Tato část kódu umožňuje uživateli vybrat obrázek z disku. Po načtení je obrázek načten pomocí OpenCV a zobrazen v grafickém rozhraní.

```
def select_image(self):
    self.clear_image_frame()
    self.load_image()

def load_image(self):
    path = filedialog.askopenfilename()
    if path:
        self.image_path = path
        self.image = cv2.imread(path)
        if self.image is not None:
            self.display_histogram()
            self.show_threshold_widgets()
        else:
            messagebox.showerror("Error", "Failed to load image.")
```

Obr. 3 - Blok pro načtení obrázku

2. Zobrazení histogramu

Tento kód vytváří histogram obrazu, který zobrazuje distribuci intenzit pixelů v obraze. Histogram je užitečný nástroj pro analýzu jasu a kontrastu v obraze a často se používá při úpravách obrázků v počítačovém vidění.

```
def display_histogram(self):
    if self.histogram_canvas:
        self.histogram_canvas.get_tk_widget().destroy()
    gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    plt.hist(gray.ravel(), 256, [0, 256])
    plt.xlabel('Intensity Value')
    plt.ylabel('Pixel Count')
    plt.title('Histogram of the Image')
    fig = plt.gcf()
    fig.set_size_inches(4, 4)
    self.histogram_canvas = FigureCanvasTkAgg(fig, master=self.histogram_frame)
    self.histogram_canvas.draw()
    self.histogram_canvas.get_tk_widget().pack()
```

Obr. 4 - Blok pro zobrazení histogramu

Pro nás je asi nejdůležitější řádek:

gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)

Tento krok totiž konvertuje načtený barevný obrázek do stupňů šedi, čímž se sníží složitost dat a zjednoduší se jejich zpracování. A další řádek:

plt.hist(gray.ravel(), 256, [0, 256])

Kde necháme zobrazit distribuci intenzit pixelů v obraze. Tento krok je užitečný pro analýzu jasu a kontrastu v obraze.

3. Segmentace obrázku

Segmentace je proces rozdělení obrazu na jednotlivé části nebo objekty. Tato metoda provádí segmentaci pomocí prahování, kde je pixel označen jako černý nebo bílý na základě hodnoty prahu. V tomto případě je prahová hodnota určena uživatelem pomocí grafického rozhraní.

```
def threshold_segmentation(self):
    if self.image is None:
        return None
    try:
        threshold_value = int(self.threshold_entry.get())
    except ValueError:
        messagebox.showerror("Error", "Invalid threshold value.")
        return None

    gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    _, binary_image = cv2.threshold(gray, threshold_value, 255, cv2.THRESH_BINARY)
    return binary_image
```

Obr. 5 - Blok pro segmentaci obrázku

Pro nás je zajímavá část kódu:

binary_image = cv2.threshold(gray, threshold_value, 255, cv2.THRESH_BINARY)

Tento krok segmentuje obrázek pomocí prahování, kde se pixel označí jako černý nebo bílý na základě hodnoty prahu. To může pomoci oddělit objekty od pozadí.

4. Zobrazení segmentovaného obrázku

Tato část kódu zobrazuje segmentovaný obrázek v grafickém rozhraní. V počítačovém vidění je vizualizace výsledků segmentace důležitá pro kontrolu a validaci výstupů algoritmů segmentace. Tyto části kódu tvoří základní funkčnost aplikace pro segmentaci obrazu a poskytují užitečné nástroje pro práci s obrazovými daty.

```
    def show_segmented_image(self):
        if self.segmented_image_label:
            self.segmented_image_label.destroy()
        segmented_image = self.threshold_segmentation()
        if segmented_image is not None:
            segmented_image = cv2.cvtColor(segmented_image, cv2.COLOR_GRAY2RGB)
            segmented_image = Image.fromarray(segmented_image)
            resized_image = segmented_image.resize((self.image_frame.winfo_width(), self.image_frame.winfo_height()))
            photo = ImageTk.PhotoImage(image=resized_image)

            # Zobrazení upraveného obrázku
            self.segmented_image_label = tk.Label(self.image_frame, image=photo)
            self.segmented_image_label.image = photo
            self.segmented_image_label.pack_propagate(False)
            self.segmented_image_label.pack(fill=tk.BOTH, expand=True)
```

Obr. 6 - Blok pro zobrazení obrázku

Kód k aplikaci:

```
import cv2

import tkinter as tk

from tkinter import filedialog, messagebox

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

from PIL import Image, ImageTk


class ImageSegmentationApp:

    def __init__(self, root):

        self.root = root

        self.root.title('Image Segmentation')

        self.root.geometry("800x600")


        self.top_frame = tk.Frame(root)

        self.top_frame.pack(side=tk.TOP, fill=tk.X)
```

```
        self.select_button = tk.Button(self.top_frame, text='Select Image', command=self.select_image)
        self.select_button.pack(side=tk.LEFT, pady=10, padx=10)

        self.save_button = None

        self.threshold_frame = tk.Frame(self.top_frame)

        self.threshold_label = tk.Label(self.threshold_frame, text='Threshold Value:')
        self.threshold_label.grid(row=0, column=0)

        self.threshold_entry = tk.Entry(self.threshold_frame)
        self.threshold_entry.grid(row=0, column=1)

        self.confirm_button = tk.Button(self.threshold_frame, text='Confirm Threshold',
command=self.show_segmented_image)
        self.confirm_button.grid(row=0, column=2, padx=(10, 0))

        self.histogram_frame = tk.Frame(root, width=200, height=200)
        self.histogram_frame.pack(side=tk.LEFT, padx=10, pady=10)

        self.image_frame = tk.Frame(root, bg='white', width=400)
        self.image_frame.pack(side=tk.RIGHT, padx=10, pady=10, fill=tk.BOTH, expand=True)

        self.histogram_canvas = None
        self.segmented_image_label = None
        self.image = None
        self.image_path = None

    def select_image(self):
        self.clear_image_frame()
```

```
    self.load_image()

def load_image(self):
    path = filedialog.askopenfilename()
    if path:
        self.image_path = path
        self.image = cv2.imread(path)
        if self.image is not None:
            self.display_histogram()
            self.show_threshold_widgets()
        else:
            messagebox.showerror("Error", "Failed to load image.")

def clear_image_frame(self):
    for widget in self.image_frame.winfo_children():
        widget.destroy()

def display_histogram(self):
    if self.histogram_canvas:
        self.histogram_canvas.get_tk_widget().destroy()
    gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    plt.hist(gray.ravel(), 256, [0, 256])
    plt.xlabel('Intensity Value')
    plt.ylabel('Pixel Count')
    plt.title('Histogram of the Image')
    fig = plt.gcf()
    fig.set_size_inches(4, 4)
    self.histogram_canvas = FigureCanvasTkAgg(fig, master=self.histogram_frame)
    self.histogram_canvas.draw()
    self.histogram_canvas.get_tk_widget().pack()
```

```
    def show_threshold_widgets(self):
        self.threshold_frame.pack(side=tk.LEFT, padx=10, pady=10)

    def threshold_segmentation(self):
        if self.image is None:
            return None
        try:
            threshold_value = int(self.threshold_entry.get())
        except ValueError:
            messagebox.showerror("Error", "Invalid threshold value.")
            return None

        gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        _, binary_image = cv2.threshold(gray, threshold_value, 255, cv2.THRESH_BINARY)
        return binary_image

    def show_segmented_image(self):
        if self.segmented_image_label:
            self.segmented_image_label.destroy()
        segmented_image = self.threshold_segmentation()
        if segmented_image is not None:
            segmented_image = cv2.cvtColor(segmented_image, cv2.COLOR_GRAY2RGB)
            segmented_image = Image.fromarray(segmented_image)
            resized_image = segmented_image.resize((self.image_frame.winfo_width(),
self.image_frame.winfo_height()))
            photo = ImageTk.PhotoImage(image=resized_image)

            # Zobrazení upraveného obrázku
            self.segmented_image_label = tk.Label(self.image_frame, image=photo)
            self.segmented_image_label.image = photo
            self.segmented_image_label.pack_propagate(False)
```

```
            self.segmented_image_label.pack(fill=tk.BOTH, expand=True)


            self.show_save_button()


    def show_save_button(self):
        if self.save_button is None:
            self.save_button = tk.Button(self.top_frame, text='Save Image', command=self.save_image)
            self.save_button.pack(side=tk.LEFT, pady=10, padx=10)


    def save_image(self):
        if self.segmented_image_label and self.image_path:
            path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG Image", "*.jpg"),
("PNG Image", "*.png"), ("All Files", "*.*")])
            if path:
                segmented_image = self.threshold_segmentation()
                if segmented_image is not None:
                    cv2.imwrite(path, segmented_image)
                    print("Image saved successfully.")
                else:
                    messagebox.showerror("Error", "Failed to save image.")


def main():
    root = tk.Tk()
    app = ImageSegmentationApp(root)
    root.mainloop()


if __name__ == '__main__':
    main()
```

Seznam obrázků