**Міністерство освіти і науки України**

**Національний університет «Запорізька політехніка»**

Кафедра програмних засобів

**ЗВІТ**

Дисципліна «Розробка прикладних програм»

Робота №5

Тема «Машинне навчання»

**Виконав варіант 19**

Студент КНТ-122 Онищенко О. А.

**Прийняли**

Викладач Дейнега Л. Ю.

2024

Мета роботи

Ознайомитись з основними пакетами, які використовуються для машинного навчання в програмах, написаних мовою Python.

Навчитися розробляти сучасні інтелектуальні системи з використанням методів машинного навчання.

Індивідуальне завдання

Розробити програмне забезпечення, яке дозволяє виділити в даному зображені набір основних кольорів. Кількість кольорів може бути вибрана попередньо з заданого діапазону. Алгоритм кластеризації обрати самостійно. Для обраного зображення визначити також набір зображень з подібними кольорами. Банк зображень повинен формуватися автоматизовано.

Текст файлу

from sklearn.cluster import KMeans

import matplotlib.image as img

from nicegui import ui, app

import numpy as np

import os

class Color:

def \_\_init\_\_(

self,

r: int,

g: int,

b: int,

):

self.red=int(r)

self.green=int(g)

self.blue=int(b)

def \_\_str\_\_(self):

return f'{self.red} {self.green} {self.blue}'

def \_\_repr\_\_(self):

return f'{self.red} {self.green} {self.blue}'

def get\_html\_rgb\_value(self):

return f'rgb({self.red},{self.green},{self.blue})'

def get\_average(self):

return (self.red+self.green+self.blue)//3

def get\_n\_most\_common\_colors(

image\_path:str,

colors\_count:int,

):

'''

< image\_path: str

full image path as string

< colors\_count: int

number of most common colors to return for a given image

> palette: list[Color]

has length of colors\_count

each element is a Color object with values for red, green and blue as ints

'''

# Read an image, get it into an object to work with

image=img.imread(image\_path)

# Read image's width, height, depth

w,h,d=tuple(image.shape)

# Get a pixel size by reshaping the image into rows

pixel=np.reshape(image,(w\*h,d))

# Make a K-Means algorithm model, fit it into a pixel

model=KMeans(n\_clusters=colors\_count,random\_state=40).fit(pixel)

# Make a colors palette by extracting cluster centers from the model

palette=np.uint8(model.cluster\_centers\_)

# Make a list for storing Colors data

res=[]

# Go through each color in palette, colors\_count times

for c in palette:

# Extract the red, green and blue values for each color

r,g,b=c

# Make and add a Color object for the colors

res.append(Color(r,g,b))

# Return the produced list of Colors

return res

def get\_colors\_dataset(

images\_folder:str,

image\_files:list[str],

):

colors\_dataset=dict()

for image\_file\_name in image\_files:

image\_path=os.path.join(images\_folder,image\_file\_name)

this\_image\_colors=get\_n\_most\_common\_colors(

image\_path=image\_path,

colors\_count=colors\_count

)

colors\_dataset[image\_file\_name]=this\_image\_colors

return colors\_dataset

def get\_average\_colors(

colors\_dataset:dict,

):

def get\_average\_color(

colors:list[Color]

):

return round(sum([c.get\_average() for c in colors])/len(colors))

return {name:get\_average\_color(colors) for name,colors in colors\_dataset.items()}

ROOT\_FOLDER=os.path.dirname(os.path.abspath(\_\_file\_\_))

IMAGE\_FOLDER\_PATH=os.path.join(ROOT\_FOLDER,'images')

IMAGE\_FILE\_NAMES=os.listdir(IMAGE\_FOLDER\_PATH)

LABEL\_CLASSES='font-medium text-lg'

COLOR\_CLASSES='flex-1 py-7 rounded-md'

app.add\_media\_files('/images',IMAGE\_FOLDER\_PATH)

image\_file=IMAGE\_FILE\_NAMES[0]

colors\_count=3

image\_colors=get\_colors\_dataset(IMAGE\_FOLDER\_PATH,IMAGE\_FILE\_NAMES)

average\_image\_colors=get\_average\_colors(image\_colors)

def update\_image\_output():

results\_image.source=os.path.join(IMAGE\_FOLDER\_PATH,image\_file)

def update\_colors():

colors\_count\_label.text=f'The image has {colors\_count} most common color{"s" if colors\_count>1 else ""}'

image\_average\_color=average\_image\_colors[image\_file]

colors\_list=list(set((image\_name,color\_value) for image\_name,color\_value in average\_image\_colors.items() if color\_value<=image\_average\_color and image\_name!=image\_file))

closest\_colors=sorted(

colors\_list,

key=lambda color: color[1],

reverse=True

)

this\_image\_colors:list[Color]=image\_colors[image\_file]

closest\_color\_one.style(f'background: {this\_image\_colors[0].get\_html\_rgb\_value()};')

closest\_color\_two.style(f'background: {this\_image\_colors[1].get\_html\_rgb\_value()};')

closest\_color\_two.visible=colors\_count>=2

closest\_color\_three.style(f'background: {this\_image\_colors[2].get\_html\_rgb\_value()};')

closest\_color\_three.visible=colors\_count>=3

closest\_color\_image\_paths=[

os.path.join(IMAGE\_FOLDER\_PATH,this\_file\_name) for this\_file\_name,this\_color\_value in closest\_colors

]

similar\_image\_one.source=closest\_color\_image\_paths[0]

similar\_image\_two.source=closest\_color\_image\_paths[1]

similar\_image\_three.source=closest\_color\_image\_paths[2]

def update\_ui():

update\_image\_output()

update\_colors()

ui.label('Image to process').classes(LABEL\_CLASSES)

ui.select(

options=IMAGE\_FILE\_NAMES,

with\_input=True,

value=image\_file,

on\_change=update\_ui,

).classes('w-full').bind\_value(globals(),'image\_file')

ui.label('Number of colors').classes('mt-7 '+LABEL\_CLASSES)

ui.slider(

min=1,

max=3,

value=colors\_count,

on\_change=update\_ui,

).bind\_value(globals(),'colors\_count')

with ui.row().classes('flex justify-between w-full'):

ui.label(1)

ui.label(2)

ui.label(3)

ui.label(f'Results for {image\_file}').classes('mt-12 '+LABEL\_CLASSES)

results\_image=ui.image(os.path.join(IMAGE\_FOLDER\_PATH,image\_file)).classes('rounded-md object-center').style('max-height: 37vh;')

colors\_count\_label=ui.label(f'Most common colors ({colors\_count})').classes('mt-7 '+LABEL\_CLASSES)

with ui.row().classes('w-full flex gap-3'):

closest\_color\_one=ui.element('span').classes(COLOR\_CLASSES)

closest\_color\_two=ui.element('span').classes(COLOR\_CLASSES)

closest\_color\_three=ui.element('span').classes(COLOR\_CLASSES)

ui.label(f'Images with similar colors').classes('mt-7 '+LABEL\_CLASSES)

with ui.row().classes('w-full flex gap-3'):

similar\_image\_one=ui.image().classes('flex-1 rounded-md max-h-60 display-cover')

similar\_image\_two=ui.image().classes('flex-1 rounded-md max-h-60 display-cover')

similar\_image\_three=ui.image().classes('flex-1 rounded-md max-h-60 display-cover')

update\_ui()

ui.run(favicon='🖼️',title='Image Colors')

Результати виконання

Показ роботи програми у вигляді відео за [посиланням](https://youtube.com/shorts/liZ15Q6qdAo).

При вході на сторінку обрано перше зображення зі списку:

Зображення, що містить хмара, дерево, небо, текст

Автоматично згенерований опис

Рисунок 1.1 – Вигляд програми при вході

При зміні зображення оновлюється користувацький інтерфейс: зображення у секції результатів, кольори та схожі зображення. Приклад нижче:

Зображення, що містить небо, текст, просто неба, знімок екрана

Автоматично згенерований опис

Рисунок 1.2 – Оновлення інтерфейсу при обранні іншого зображення

При зміні кількості кольорів змінюється вивід у секції кольорів:

Зображення, що містить вода, дерево, озеро, текст

Автоматично згенерований опис

Рисунок 1.3 – Зміна кількості кольорів

Зображення, що містить текст, просто неба, небо, знімок екрана

Автоматично згенерований опис

Рисунок 1.4 – Зміна кількості кольорів на один

Опис алгоритма розв’язання задачі

Алгоритм працює наступним чином:

1. Завантажує зображення з наданого шляху;
2. Отримує розміри зображення: ширину, довжину та глибину;
3. Змінює форму зображення на двомірний масив з кожним рядком як піксель;
4. Створює модель алгоритму К-середніх. Як кількість кластерів приймає значення кількості кольорів, отримане від користувача у інтерфейсі;
5. Підганяє модель під піксель, отримує центри кластерів, які і є найпоширенішими кольорам;
6. Конвертує кластери до типу цілого числа, повертає значення як список змінних типу Color.

[Джерело](https://www.geeksforgeeks.org/extract-dominant-colors-of-an-image-using-python/)

Текст алгоритму:

def get\_n\_most\_common\_colors(

image\_path:str,

colors\_count:int,

):

'''

< image\_path: str

full image path as string

< colors\_count: int

number of most common colors to return for a given image

> palette: list[Color]

has length of colors\_count

each element is a Color object with values for red, green and blue as ints

'''

# Read an image, get it into an object to work with

image=img.imread(image\_path)

# Read image's width, height, depth

w,h,d=tuple(image.shape)

# Get a pixel size by reshaping the image into rows

pixel=np.reshape(image,(w\*h,d))

# Make a K-Means algorithm model, fit it into a pixel

model=KMeans(n\_clusters=colors\_count,random\_state=40).fit(pixel)

# Make a colors palette by extracting cluster centers from the model

palette=np.uint8(model.cluster\_centers\_)

# Make a list for storing Colors data

res=[]

# Go through each color in palette, colors\_count times

for c in palette:

# Extract the red, green and blue values for each color

r,g,b=c

# Make and add a Color object for the colors

res.append(Color(r,g,b))

# Return the produced list of Colors

return res

Результати тестування

Після проведення тестування (зміни зображень, зміни кількості кольорів) помилок не було виявлено, окрім однією: іноді показує те саме зображення у секції Схожих зображень. Проблему не було вирішено, але шматок коду для її потенційного вирішення:

colors\_list=list(set((image\_name,color\_value) for image\_name,color\_value in average\_image\_colors.items() if color\_value<=image\_average\_color and image\_name!=image\_file))

Перелік проблем під час виконання

Під час виконання роботи виникло кілька проблем. Нижче наведено опис кожної та її вирішення:

*1 Як змінити колір коробочки кольору для секції найпоширеніших кольорів?*

**Опис**: у секції результатів є підсекція яка виводить список найпоширеніших кольорів зображення. До елементів користувацького інтерфейсу було додано елементи span з відповідними стилями (закруглені края, трохи вертикального внутрішнього відступу, задання ширини на таку що покриває все вільне місце), але для зміни кольору потрібно:

*Використання кастомного кольору:* Після спроби застосувати кастомний колір як клас Tailwind (bg-[#customColorHexCode]) нічого не спрацювало. Тоді для вирішення цього потрібно прописати користувацький CSS код із вказанням кольору як значення CSS (background: rgb(custom, color, values);).

**Вирішення**: після пошуку шляхів вирішення проблеми Господь Ісус Христос підказав спосіб оновлення стилів елементу через зміну властивості styles. Проблему було вирішення застосування методу style для кожного елементу коробки кольору зі значенням кольору, отриманого від класу Color.

*2 Як зв’язати позицію елементу користувацького інтерфейсу зі змінною в коді аби використовувати його значення для динамічного відображення даних?*

**Опис**: ця проблема виникла рано під час розробки. Потрібно було отримати значення елементу повзунка (використовується для отримання від користувача кількості кольорів) для застосування його у обрахунку кольорів.

**Вирішення**: благодаттю Господа нашого Ісуса Христа було знайдено метод bind\_value(), який і було застосовано до глобальної змінної colors\_count. Після цього значення цієї змінної почало оновлюватися зі зміною позиції повзунка.

Висновки по роботі

В результаті виконання було розроблено веб-застосунок на мові програмування Python використовуючи бібліотеки scikit-learn та nicegui. Застосунок дозволяє обрати зображення із завантажених до файлів проєкту і для нього показує список найбільш використаних кольорів. Застосунок працює у браузері, що дозволяє користувачам мати доступ до нього з будь-якого приладу, який має доступ до веб-браузера. Вся слава Господу Ісусу Христу Всемогутньому БОГУ Амінь.