Звіт

Лабораторна робота №3

Підготувала  
студентка факультету КНК

Групи ІПС-21  
Боровик Анастасія

# Багатопоточність для нейромереж

Використання багатопоточності дуже важливий аспект при роботі з нейромережамиі машинним навчанням, в цій сфері зазвичай потрібно багато обчислень і будь-яке спрощення може суттєво допомогти у роботі. Саме тому, для проєкту який використовує глибоку нейромережу для роботи із зображеннями, що неймовірно складна задача з точки зору обчислень, було прийняти рішення застосуватибагатопоточність.

Має програму для передбачення об’єктів на відео, яка окрім передбачення малює обмежувальні рамки навколо об’єктів і збирає нові кадри у вихідне відео. Запускмоделі проводиться виключно на GPU:

import cv2

import numpy as np

import tensorflow as tf

from threading import Thread

# Функція для завантаження нейромережі

def load\_model():

# Завантаження моделі (приклад)

model = tf.keras.applications.MobileNetV2(weights='imagenet', include\_top=True)

return model

# Функція для передбачення об'єктів на кадрі

def predict\_objects(frame, model):

# Перетворення кадру до формату, який приймає модель

resized\_frame = cv2.resize(frame, (224, 224))

input\_data = np.expand\_dims(resized\_frame, axis=0)

input\_data = tf.keras.applications.mobilenet\_v2.preprocess\_input(input\_data)

# Передбачення класу об'єкту та відповідних обмежувальних рамок

predictions = model.predict(input\_data)

# Отримання результуючих об'єктів та вірогідностей

# Тут можна використати власну логіку для обробки результатів передбачення

return predictions

# Функція для відображення кадру з обмежувальними рамками

def display\_frame\_with\_boxes(frame, boxes):

# Відображення кадру

# Додавання обмежувальних рамок

# Вивід на екран

# Функція для обробки відеопотоку

def process\_video(video\_path, output\_path, model):

# Відкриття відеофайлу

cap = cv2.VideoCapture(video\_path)

# Отримання параметрів відеопотоку

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

# Визначення відеокодеку та вихідного відеопотоку

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# Передбачення об'єктів на кадрі

predictions = predict\_objects(frame, model)

# Додавання обмежувальних рамок на кадр

frame\_with\_boxes = display\_frame\_with\_boxes(frame, predictions)

# Запис кадру з обмежувальними рамками у вихідний відеофайл

out.write(frame\_with\_boxes)

cv2.imshow('Frame', frame\_with\_boxes)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Закриття відеопотоків та відеозаписника

cap.release()

out.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

# Завантаження моделі на GPU

with tf.device('/GPU:0'):

model = load\_model()

# Шлях до вхідного та вихідного відеофайлів

input\_video\_path = 'input\_video.mp4'

output\_video\_path = 'output\_video.mp4'

# Запуск обробки відео

process\_video(input\_video\_path, output\_video\_path, model)

Має сенс запустити післяобробку зображення на паралельному процесі, що дозволить вже починати обробляти зображення ще під час післяобробки попереднього зображення, коли ми малюємо обмежувальні рамки.Відповідно маємо окремий клас для обробки кадрів на іншому потоці FrameProcessingThread.

import cv2

import numpy as np

import tensorflow as tf

from threading import Thread

# Функція для завантаження нейромережі

def load\_model():

# Завантаження моделі (приклад)

model = tf.keras.applications.MobileNetV2(weights='imagenet', include\_top=True)

return model

# Функція для передбачення об'єктів на кадрі

def predict\_objects(frame, model):

# Перетворення кадру до формату, який приймає модель

resized\_frame = cv2.resize(frame, (224, 224))

input\_data = np.expand\_dims(resized\_frame, axis=0)

input\_data = tf.keras.applications.mobilenet\_v2.preprocess\_input(input\_data)

# Передбачення класу об'єкту та відповідних обмежувальних рамок

predictions = model.predict(input\_data)

# Отримання результуючих об'єктів та вірогідностей

# Тут можна використати власну логіку для обробки результатів передбачення

return predictions

# Функція для обробки кадру в окремому потоці

def process\_frame(frame, model):

# Передбачення об'єктів на кадрі

predictions = predict\_objects(frame, model)

# Додавання обмежувальних рамок на кадр

frame\_with\_boxes = display\_frame\_with\_boxes(frame, predictions)

return frame\_with\_boxes

# Функція для відображення кадру з обмежувальними рамками

def display\_frame\_with\_boxes(frame, boxes):

# Відображення кадру

# Додавання обмежувальних рамок

# Вивід на екран

# Функція для обробки відеопотоку

def process\_video(video\_path, output\_path, model):

# Відкриття відеофайлу

cap = cv2.VideoCapture(video\_path)

# Отримання параметрів відеопотоку

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

# Визначення відеокодеку та вихідного відеопотоку

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# Обробка кадру в окремому потоці

frame\_with\_boxes = process\_frame(frame, model)

# Запис кадру з обмежувальними рамками у вихідний відеофайл

out.write(frame\_with\_boxes)

cv2.imshow('Frame', frame\_with\_boxes)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Закриття відеопотоків та відеозаписника

cap.release()

out.release()

cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

# Завантаження моделі на GPU

with tf.device('/GPU:0'):

model = load\_model()

# Шлях до вхідного та вихідного відеофайлів

input\_video\_path = 'input\_video.mp4'

output\_video\_path = 'output\_video.mp4'

# Запуск обробки відео

process\_video(input\_video\_path, output\_video\_path, model)

# Багатопоточність для збору даних

# Для проєкту, у рамках якого передбачався масштабний збір даних із відкритих ресурсів було застосовано використання багатопоточності для пришвидшення цього часозатратного процесу. У проєкті потрібно збирати інформацію використовуючи Selenium. Для багатопоточної реалізації передбачено відкриття багатьох вікон, кожне з яких на окремому потоці для паралельного збору даних:

def main(threads: int = 4, driver\_name: str = 'Safari', iteration\_to\_save\_data: int = 100) ->None:

users = get\_list\_of\_users()

if threads:

with ThreadPoolExecutor(threads) as executor:

for th in range(threads):

executor.submit(scrape\_followers,

[users[i] for i in range(len(users)) if i % threads == th],

driver\_name,

iteration\_to\_save\_data)

else:

pass

data = pd.read\_csv('../../data/twitter/scrapped\_tweets.csv')

with open('../../data/twitter/followers.json', 'r') as f:

followers: dict[str, int] = json.load(f)

followers\_data = add\_followers(data, followers)

followers\_data.to\_csv('../../data/twitter/scrapped\_tweets.csv')

if \_\_name\_\_ == '\_\_main\_\_':

main(threads=1, driver\_name='Safari', iteration\_to\_save\_data=20)

Відповідно, для запуску не паралельної версії достатньо зазничити у якості кількості потоків 1.

Для порівняння швидкодіїдля одних і тих самих вікон наведений код було запущено декілька разів. Нижче наведено результати, узагальнення та висновки.

Для версії без багатопоточності:

Час виконання: 32 секунди

Час виконання: 28 секунд

Час виконання 30 секунд

Час виконання: 29 секунд

Середній час виконання: 29.75 секунд

Для версії з багатопотоковістю (2 threads):

Час виконання: 18 секунд

Час виконання: 17 секунд

Час виконання: 19 секунд

Час виконання: 18 секунд

Середній час виконання: 18 секунд

Ми бачимо очевидне зменшення приблизно на 39,66%.

Окрім того, результати роботи у вигляді інформації в файлах залишилися тими самими.

Результати бенчмаркінгу свідчать про значне покращення часової ефективності при реалізації багатопотоковості. Варто зазначити, що ці результати можуть відрізнятися залежно від конкретного обладнання, програмного забезпечення та середовища, яке використовується для виконання.

Course

https://medium.com/analytics-vidhya/exploiting-multithreading-and-multiprocessing-in-python-as-a-data-scientist-e2c98b61997a

.

https://medium.com/analytics-vidhya/exploiting-multithreading-and-multiprocessing-in-python-as-a-data-scientist-e2c98b61997a