Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №1

по курсу

«Технологии распределенных вычислений и анализа данных»

|  |  |
| --- | --- |
| Выполнила: | магистрант группы 355841  А.В. Деркач |
| Проверил: | к.т.н., доцент каф. ЭВМ  Д.Ю. Перцев |

Минск 2024

**1 ЗАДАНИЕ**

На вход подаются папки с множеством видеофайлов. На выходе, в каждой проанализированной папке, для каждого видеофайла, создается некоторый набор миниатюр.

Бонусные задачи: видеофайлы расположены на разных устройствах, присутствует распределенная система генерации.

Допускается использовать: любую операционную систему, любой язык программирования, любые технологии и библиотеки алгоритмов.

**2 ВЫПОЛНЕНИЕ РАБОТЫ**

Реализована пользовательская утилита, которая запускается с помощью командной строки и поддерживает параметры, представленные на рисунке 2.1.

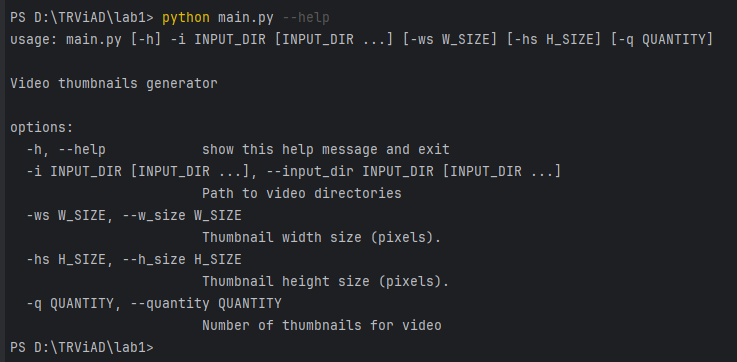


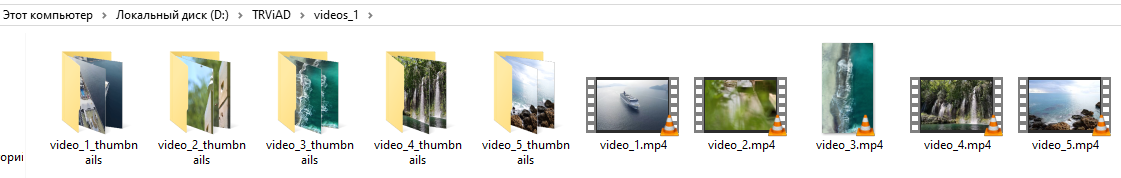
Рисунок 2.1 – Параметры для настройки генерации миниатюр

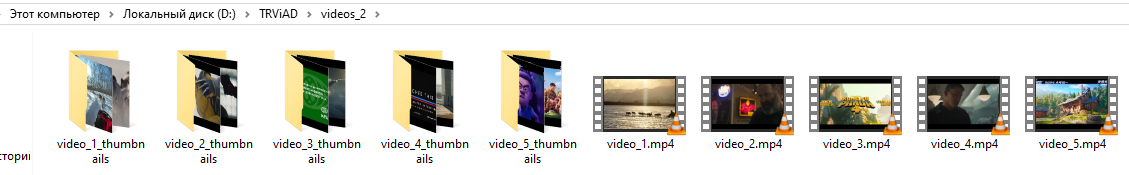
Для запуска утилиты с указанием трех директорий видеофайлов (суммарно 15 видео), десяти миниатюр для генерации размером 854x480 (суммарно 150 миниатюр) выполняется команда:

python main.py -i '../videos\_1' '../videos\_2' '../videos\_3' -ws 854 -hs 480 -q 10

Утилита анализирует все переданные папки и для каждого найденного видеофайла генерирует заданное количество миниатюр с заданным размером. Каждое видео разбивается на фреймы и случайным образом отбираются фреймы для сохранения в качестве миниатюры.

Результат генерации видеофайлов в трех директориях представлен на рисунке 2.2. На рисунке 2.3 представлен результат генерации миниатюр для одного видео.





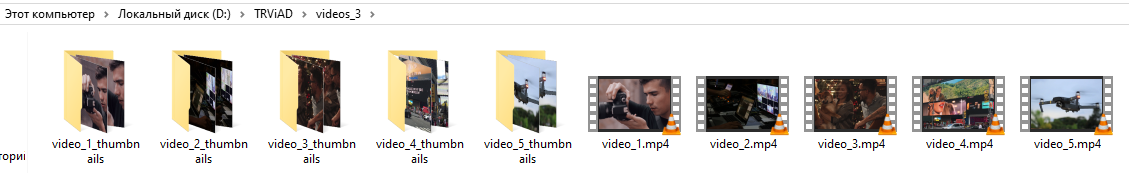


Рисунок 2.2 – Результат генерации миниатюр для видеофайлов в трех директориях

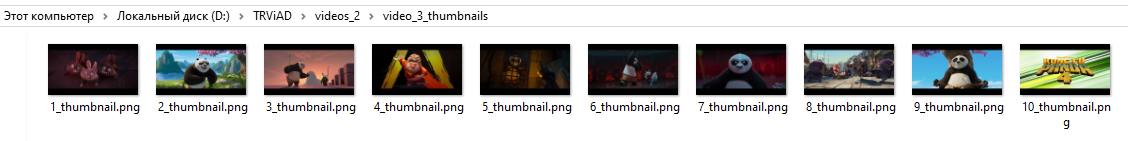


Рисунок 2.3 – Результат генерации миниатюр для одного видеофайла

В утилите предусмотрена распределенная система генерации с балансировщиком нагрузки. Для этого указываются адреса узлов (серверов), которые через REST API принимают видео и возвращают список миниатюр для этого видео. Все узлы работают параллельно, поэтому чем больше узлов – тем быстрее обработаются видеофайлы. Узлы могут распологаться на разных серверах, доступных утилите.

Для тестирования балансировщика нагрузки было добавлено дополнительной логирование, демонстрирующее сервис, на который осуществлялась отправка задачи по обработке одного видео (см. рисунок 2.4).



Рисунок 2.4 – Логирование балансировщика нагрузки узлов генерации

Код реализации пользовательского приложения на языке Python:

001. import argparse

002. import os

003. import shutil

004. from concurrent.futures.thread import ThreadPoolExecutor

005.

006. import cv2

007. import numpy as np

008. import requests

009.

010. VIDEO\_FORMATS = ['.WEBM', '.MPG', '.MP2', '.MPEG', '.MPE', '.MPV', '.OGG', '.MP4', '.M4P', '.M4V',

011. '.AVI', '.WMV', '.MOV', '.QT', '.FLV', '.SWF', '.AVCHD']

012.

013. WORKER\_NODES = [

014. "http://localhost:8080",

015. "http://localhost:8081",

016. "http://localhost:8082"

017. ]

018.

019.

020. def get\_videos\_from\_directory(video\_directory\_path: str):

021. directory\_video\_files = []

022. directories\_to\_process = [video\_directory\_path]

023.

024. while directories\_to\_process:

025. current\_directory = directories\_to\_process.pop(0)

026. directory\_entries = os.listdir(current\_directory)

027.

028. for entry in directory\_entries:

029. full\_entry\_path = os.path.join(current\_directory, entry)

030.

031. if os.path.isdir(full\_entry\_path):

032. directories\_to\_process.append(full\_entry\_path)

033. else:

034. \_, file\_extension = os.path.splitext(full\_entry\_path)

035. if file\_extension.upper() in VIDEO\_FORMATS:

036. directory\_video\_files.append(full\_entry\_path)

037.

038. return directory\_video\_files

039.

040.

041. def save\_video\_thumbnails(video\_path: str, thumbnails: list[np.ndarray]):

042. video\_thumbnails\_directory = os.path.join(

043. os.path.dirname(video\_path), "%s\_thumbnails" % (os.path.splitext(os.path.basename(video\_path))[0])

044. )

045.

046. if os.path.exists(video\_thumbnails\_directory):

047. shutil.rmtree(video\_thumbnails\_directory)

048.

049. os.mkdir(video\_thumbnails\_directory)

050.

051. for i, thumbnail in enumerate(thumbnails):

052. cv2.imwrite('%s/%d\_thumbnail.png' % (video\_thumbnails\_directory, i + 1), thumbnail)

053.

054.

055. def process\_videos\_parallel(videos: list[str], thumbnail\_width: int, thumbnail\_height: int, num\_thumbnails: int):

056. with ThreadPoolExecutor(max\_workers=len(WORKER\_NODES)) as executor:

057. futures = []

058. for video\_path in videos:

059. worker\_url = WORKER\_NODES[len(futures) % len(WORKER\_NODES)]

060.

061. futures.append(executor.submit(process\_video, video\_path, thumbnail\_width, thumbnail\_height, num\_thumbnails,

062. worker\_url))

063.

064. for future in futures:

065. future.result()

066.

067.

068. def process\_video(video\_path: str, thumbnail\_width: int, thumbnail\_height: int, num\_thumbnails: int, worker\_url: str):

069. with open(video\_path, 'rb') as file:

070. files = {'video': file}

071. params = {

072. 'thumbnail\_width': thumbnail\_width,

073. 'thumbnail\_height': thumbnail\_height,

074. 'num\_thumbnails': num\_thumbnails

075. }

076.

077. response = requests.post(worker\_url + '/process\_video', files=files, data=params)

078. response.raise\_for\_status()

079.

080. response\_data = response.json()

081.

082. thumbnails\_json = response\_data['thumbnails']

083. thumbnails = [np.array(thumbnail) for thumbnail in thumbnails\_json]

084.

085. save\_video\_thumbnails(video\_path, thumbnails)

086.

087.

088. def main():

089. parser = argparse.ArgumentParser(description='Video thumbnails generator')

090. parser.add\_argument('-i', '--input\_dir', type=str, nargs='+', required=True, help='Path to video directories')

091. parser.add\_argument('-ws', '--w\_size', type=int, default=640, help='Thumbnail width size (pixels).')

092. parser.add\_argument('-hs', '--h\_size', type=int, default=640, help='Thumbnail height size (pixels).')

093. parser.add\_argument('-q', '--quantity', type=int, default=5, help='Number of thumbnails for video')

094. args = parser.parse\_args()

095.

096. try:

097. input\_directories = args.input\_dir

098. thumbnail\_width = args.w\_size

099. thumbnail\_height = args.h\_size

100. num\_thumbnails = args.quantity

101.

102. for video\_directory\_path in input\_directories:

103. directory\_videos = get\_videos\_from\_directory(video\_directory\_path)

104.

105. for i in range(0, len(directory\_videos), len(WORKER\_NODES)):

106. group = directory\_videos[i:i + len(WORKER\_NODES)]

107. process\_videos\_parallel(group, thumbnail\_width, thumbnail\_height, num\_thumbnails)

108.

109. print("Thumbnails successfully generated.")

110. except Exception as e:

111. print("Oops! Something went wrong... :( \n{}".format(e))

112.

113.

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

115. main()

Код реализации узла генерации изображений на языке Python (каждый узел дублируется и задается свой порт):

01. import os

02. import tempfile

03.

04. from flask import Flask, request, jsonify

05.

06. from video\_thumbnails\_generator import VideoThumbnailsGenerator

07.

08. app = Flask(\_\_name\_\_)

09.

10.

11. @app.route('/process\_video', methods=['POST'])

12. def process\_video():

13. video\_file = request.files['video']

14. thumbnail\_width = int(request.form['thumbnail\_width'])

15. thumbnail\_height = int(request.form['thumbnail\_height'])

16. num\_thumbnails = int(request.form['num\_thumbnails'])

17.

18. with tempfile.TemporaryDirectory() as temp\_dir:

19. try:

20. video\_path = os.path.join(temp\_dir, video\_file.filename)

21. video\_file.save(video\_path)

22.

23. generator = VideoThumbnailsGenerator(video\_path, thumbnail\_width, thumbnail\_height, num\_thumbnails)

24. generated\_thumbnails = generator.generate()

25.

26. thumbnails\_data = [thumbnail.tolist() for thumbnail in generated\_thumbnails]

27. return jsonify({'thumbnails': thumbnails\_data})

28. except Exception as e:

29. return 'Something went wrong... {}'.format(e), 500

30.

31.

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

33. app.run(debug=True, port=8080)

Код реализации алгорима генарации на языке Python:

01. import random

02.

03. import cv2

04. import numpy as np

05.

06.

07. class VideoThumbnailsGenerator:

08. VIDEO\_FORMATS = ['.WEBM', '.MPG', '.MP2', '.MPEG', '.MPE', '.MPV', '.OGG', '.MP4', '.M4P', '.M4V',

09. '.AVI', '.WMV', '.MOV', '.QT', '.FLV', '.SWF', '.AVCHD']

10.

11. def \_\_init\_\_(self, video\_path: str, thumbnail\_width=640, thumbnail\_height=640, num\_thumbnails=5):

12. self.video\_path = video\_path

13. self.thumbnail\_width = thumbnail\_width

14. self.thumbnail\_height = thumbnail\_height

15. self.num\_thumbnails = num\_thumbnails

16.

17. def generate(self):

18. video\_frames\_number = self.get\_video\_frames\_number(self.video\_path)

19. video\_thumbnails\_frames\_indexes = self.select\_random\_video\_frames(video\_frames\_number)

20. video\_thumbnails\_frames = self.get\_frames\_from\_video(self.video\_path, video\_thumbnails\_frames\_indexes)

21.

22. return self.convert\_frames\_to\_thumbnails(video\_thumbnails\_frames)

23.

24. def select\_random\_video\_frames(self, video\_frames\_number: int):

25. frames\_number\_to\_select = self.num\_thumbnails

26.

27. if frames\_number\_to\_select >= video\_frames\_number:

28. frames\_number\_to\_select = video\_frames\_number

29.

30. return random.sample(range(video\_frames\_number), frames\_number\_to\_select)

31.

32. def convert\_frames\_to\_thumbnails(self, frames: list[np.ndarray]):

33. thumbnails = []

34. for frame in frames:

35. frame\_hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

36. thumbnail = cv2.resize(frame\_hsv, (self.thumbnail\_width, self.thumbnail\_height))

37. result\_thumbnail = cv2.cvtColor(thumbnail, cv2.COLOR\_HSV2BGR)

38.

39. thumbnails.append(np.array(result\_thumbnail))

40.

41. return thumbnails

42.

43. @staticmethod

44. def get\_frames\_from\_video(video\_path: str, frame\_indexes: list[int]):

45. cap = cv2.VideoCapture(video\_path)

46. if not cap.isOpened():

47. raise ValueError("Unable to open video file '{}'".format(video\_path))

48.

49. frames = []

50. for index in frame\_indexes:

51. cap.set(cv2.CAP\_PROP\_POS\_FRAMES, index)

52.

53. ret, frame = cap.read()

54. if not ret:

55. raise ValueError("Failed to read frame from video file '{}'".format(video\_path))

56.

57. frames.append(frame)

58.

59. cap.release()

60.

61. return frames

62.

63. @staticmethod

64. def get\_video\_frames\_number(video\_path: str):

65. cap = cv2.VideoCapture(video\_path)

66. if not cap.isOpened():

67. raise ValueError("Unable to open video file '{}'".format(video\_path))

68.

69. video\_frames\_number = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

70.

71. cap.release()

72. return video\_frames\_number

**3 ВЫВОДЫ**

В ходе выполнения лабораторной работы была подготовлена пользовательская утилита для генерации миниатюр фидеофайлов в заданных директориях операционной системы (ОС). Утилита реализована с применением распределительной системы генерации и балансировщиком нагрузки между несколькими REST API «узлов» генерации. Тестирование производилось с использованием трех «узлов» на ОС Microsoft Windows 10 и 11.