

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА № 7**

**по дисциплине**

**«Проектирование интеллектуальных систем (часть ½)»**

Студент группы: ИКБО-04-21 Даурбеков М.И. *(Ф.И.О.студента)*

Руководитель Холмогоров В.В.

*(Ф.И.О.преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc167296791)

[1 ТЕОРИТИЧЕСКИЙ РАЗДЕЛ 4](#_Toc167296792)

[1.1 Описание предметной области 4](#_Toc167296793)

[1.2 Существующие подходы 4](#_Toc167296794)

[1.3 Метод решения 8](#_Toc167296795)

[2 АНАЛИТИЧЕСКИЙ РАЗДЕЛ 10](#_Toc167296796)

[2.1 Алгоритмический и математический анализ 10](#_Toc167296798)

[3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 11](#_Toc167296799)

[3.1 Описание программных сущностей 11](#_Toc167296801)

[3.2 Тестирование 11](#_Toc167296802)

[ЗАКЛЮЧЕНИЕ 12](#_Toc167296803)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc167296804)

[ПРИЛОЖЕНИЯ 14](#_Toc167296805)

# 

# АННОТАЦИЯ

В данной работе описано создание back-end составляющей для модели классификации музейных предметов на основе изображения.

# ВВЕДЕНИЕ

Сегодня музейным сотрудникам необходимо вручную присваивать информацию о новых экспонатах, что по мере их количества изнуряет людей и замедляет поступление информации на ресуры.

# ТЕОРИТИЧЕСКИЙ РАЗДЕЛ

* 1. Постановка задачи

Необходимо создать веб-сервис на основе искусственного интеллекта, который сможет определить, к какому классу музейных экспонатов относится предмет на изображении и сгенерировать для него описание.

* 1. Существующие подходы

В настоящее время Государственный каталог Музейного фонда Российской Федерации содержит более 45 млн музейных предметов со всех музеев России. Информация, описание и изображение по каждому музейному предмету вводится тысячами музейных работников по всей стране в «ручном» режиме в систему ФГИС «Госкаталог». Обеспечение качества данных в такой обширной базе является сложной задачей и не представляется возможной в «ручном» режиме.

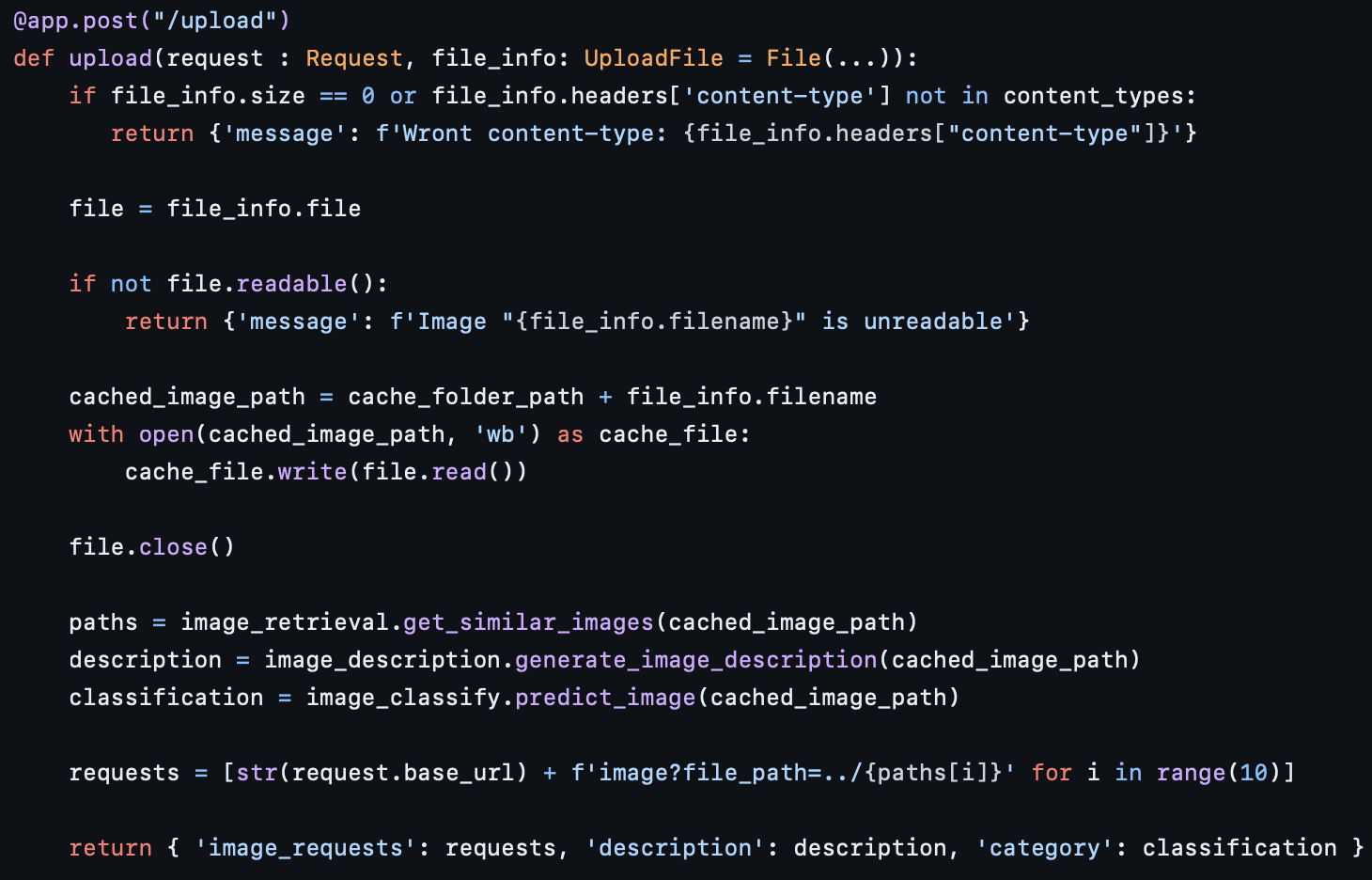
# АНАЛИТИЧЕСКИЙ РАЗДЕЛ

1. 1. Алгоритмический анализ

В ходе участия хакатона был разработан back-end на ЯП python с помощью фрейворка FastAPI. который имеет 2 метода:

Методы:

1. /upload
2. /image

****

**Рисунок 1 – Метод /upload**

Метод /upload принимает на вход файл изображения. Изображение сохраняется в временную папку для обработки. Далее изображение обрабатывается с помощью Clip модели и производится поиск наиболее похожих изображений с помощью косинусного расстояния и пред базы эмбеддингов. Далее с помощью Blip модели генерируется описание изображения. В конце, изображение обрабатывается до обученной моделью ResNet152 для классификации предмета на изображении.

****

**Рисунок 2 – Метод /image**

Метод /image используется для выгрузки файла, путь до которого выдал метод /upload в JSON формате.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы было осуществлено знакомство с фреймворком FastAPI и написание рабочего веб-сервиса, который классифицирует музейный экспонат, генерирует для него описание и вывод 10 наиболее похожих на него изображений. Были получены навыки работы в команде и публичной презентации продукта. Подтверждение участия представлено в приложении Б.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сорокин А. Б. Свёрточные нейронные сети: примеры реализаций – Москва: МИРЭА, 2020.

2. Цифровой прорыв - https://hacks-ai.ru/

3. FastAPI - https://fastapi.tiangolo.com/

4. Заказчик - https://gb.ru

# ПРИЛОЖЕНИЯ

Приложение А – Исходный код

Приложение Б – Сертификат

**Приложение А**

Исходный код

*Листинг 1 – код main.py*

from fastapi import FastAPI, UploadFile, File, Request

from fastapi.responses import FileResponse

from fastapi.middleware.cors import CORSMiddleware

from config import content\_types, cache\_folder\_path

import image\_retrieval

import image\_description

import image\_classify

app = FastAPI()

app.add\_middleware( CORSMiddleware, allow\_origins=['\*'] )

@app.post("/upload")

def upload(request : Request, file\_info: UploadFile = File(...)):

if file\_info.size == 0 or file\_info.headers['content-type'] not in content\_types:

return {'message': f'Wront content-type: {file\_info.headers["content-type"]}'}

file = file\_info.file

if not file.readable():

return {'message': f'Image "{file\_info.filename}" is unreadable'}

cached\_image\_path = cache\_folder\_path + file\_info.filename

with open(cached\_image\_path, 'wb') as cache\_file:

cache\_file.write(file.read())

file.close()

paths = image\_retrieval.get\_similar\_images(cached\_image\_path)

description = image\_description.generate\_image\_description(cached\_image\_path)

classification = image\_classify.predict\_image(cached\_image\_path)

requests = [str(request.base\_url) + f'image?file\_path=../{paths[i]}' for i in range(10)]

return { 'image\_requests': requests, 'description': description, 'category': classification }

@app.get("/image")

def image(file\_path : str):

return FileResponse(file\_path)

*Листинг 2 – код image\_retrieval.py*

import numpy as np

from transformers import CLIPModel, CLIPProcessor

from sklearn.metrics.pairwise import cosine\_similarity

from PIL import Image

device = 'cpu'

data = np.load('image\_embeddings.npz')

image\_filepaths = data['file\_names']

image\_embeddings = data['embeddings']

model\_id = 'openai/clip-vit-base-patch32'

model = CLIPModel.from\_pretrained(model\_id).to(device)

processor = CLIPProcessor.from\_pretrained(model\_id)

def get\_similar\_images(file\_path : str):

search\_image = Image.open(file\_path)

search\_inputs = processor(text=None, images=search\_image, return\_tensors="pt")

search\_pixel\_values = search\_inputs["pixel\_values"].to(device)

search\_features = model.get\_image\_features(pixel\_values=search\_pixel\_values).squeeze(0).cpu().detach().numpy()

distances = 1 - cosine\_similarity([search\_features], image\_embeddings).flatten()

path\_indexes = distances.argsort()[:10]

distances = sorted(distances)[:10]

return [image\_filepaths[path\_indexes[i]] for i in range(10)]

*Листинг 3 – код image\_features.py*

from transformers import CLIPTokenizerFast, CLIPModel, CLIPProcessor

import cv2

from tqdm import tqdm

import numpy as np

from sklearn.metrics.pairwise import cosine\_similarity

from config import nn\_classes

import torch

from torch.utils.data import Dataset, DataLoader

import pandas as pd

from PIL import Image

train\_folder\_path = "train\_dataset/train"

class TrainLoader(Dataset):

def \_\_init\_\_(self, path):

self.data = pd.read\_csv(path, sep=';')

def \_\_len\_\_(self):

return len(self.data)

def \_\_getitem\_\_(self, index):

object\_id = self.data.iloc[index]['object\_id']

img\_name = self.data.iloc[index]['img\_name']

image\_path = f'{train\_folder\_path}/{object\_id}/{img\_name}'

# image = cv2.imread(image\_path)

# image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

category\_index = nn\_classes.index(self.data.iloc[index]['group'])

return image\_path, category\_index

device = 'cpu'

train\_data = TrainLoader('train\_dataset/train.csv')

image\_embeddings = []

list\_of\_files = []

model\_id = 'openai/clip-vit-base-patch32'

model = CLIPModel.from\_pretrained(model\_id).to(device)

processor = CLIPProcessor.from\_pretrained(model\_id)

for image\_path, label in tqdm(train\_data):

list\_of\_files.append(image\_path)

image = Image.open(image\_path)

inputs = processor(text=None, images=image, return\_tensors="pt")

pixel\_values = inputs["pixel\_values"].to(device)

image\_features = model.get\_image\_features(pixel\_values=pixel\_values)

image\_features = image\_features.squeeze(0)

image\_features = image\_features.cpu().detach().numpy()

image\_embeddings.append(image\_features)

image\_arr = np.vstack(image\_embeddings)

np.savez('image\_embeddings.npz', embeddings=image\_arr, file\_names=list\_of\_files)

*Листинг 4 – код image\_description.py*

from PIL import Image

from transformers import BlipProcessor, BlipForConditionalGeneration, T5ForConditionalGeneration, T5Tokenizer

device = 'cpu'

processor = BlipProcessor.from\_pretrained('Salesforce/blip-image-captioning-base')

model\_blip = BlipForConditionalGeneration.from\_pretrained('Salesforce/blip-image-captioning-base').to(device)

model\_t5 = T5ForConditionalGeneration.from\_pretrained('utrobinmv/t5\_translate\_en\_ru\_zh\_small\_1024').to(device)

tokenizer\_t5 = T5Tokenizer.from\_pretrained('utrobinmv/t5\_translate\_en\_ru\_zh\_small\_1024')

def generate\_image\_description(file\_path: str):

image = Image.open(file\_path)

inputs = processor(images=image, return\_tensors="pt").to(device)

outputs = model\_blip.generate(\*\*inputs, max\_new\_tokens=20)

caption = processor.decode(outputs[0], skip\_special\_tokens=True)

input\_ids = tokenizer\_t5('translate to ru: ' + caption, return\_tensors="pt").input\_ids.to(device)

generated\_tokens = model\_t5.generate(input\_ids)

return tokenizer\_t5.decode(generated\_tokens[0], skip\_special\_tokens=True)

*Листинг 5 – код image\_classify.py*

from PIL import Image

from torchvision import transforms, models

import torch

import torch.nn as nn

from config import nn\_classes

device = 'cpu'

# Загрузка модели

loaded\_model = models.resnet152()

num\_ftrs = loaded\_model.fc.in\_features

loaded\_model.fc = nn.Linear(num\_ftrs, 15) # Указываем ту же архитектуру, что и при обучении

loaded\_model.load\_state\_dict(torch.load('ResNet152.pth', map\_location=torch.device(device)))

loaded\_model = loaded\_model.to(device)

loaded\_model.eval() # Переводим модель в режим оценки

# Трансформация изображения для предсказания

def transform\_image(image\_path):

transform = transforms.Compose([

transforms.Resize((64, 64)),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

image = Image.open(image\_path)

image = transform(image).unsqueeze(0) # Добавляем размерность батча

return image

# Функция для получения предсказания

def predict\_image(image\_path):

loaded\_model.eval()

image = transform\_image(image\_path)

image = image.to(device)

with torch.no\_grad():

output = loaded\_model(image)

\_, predicted = torch.max(output, 1)

idx = predicted.item()

return nn\_classes[idx]

**Приложение Б**

Сертификат

