Лабораторная работа №2 - Встраивание моделей компьютерного зрения в веб-приложение (FastAPI)

Цель:

Разработать простое веб-приложение на FastAPI, которое позволит пользователю загружать изображения и получать предсказания от модели классификации изображений.

Шаги разработки

1. Подготовка окружения

Установите необходимые библиотеки:

```
pip install fastapi uvicorn pillow torch torchvision python-
        multipart
```

2. Структура проекта

Здесь отображена структура будущего проекта:

```
fastapi_image_classifier/
├── app.py
├── models.py
├── classifier/
│   └── resnet18.pth
└── templates/
    ├── upload.html
    └── result.html
```

3. Модель классификации изображений

В файле `models.py` добавьте код для загрузки предобученной модели (в примере используется PyTorch).

```
import torch
from torchvision import models, transforms
from PIL import Image

# Загрузка модели ResNet18
model = models.resnet18(pretrained=True)
model.eval()
```

```
# Преобразования для модели
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
        0.224, 0.225]),
])

def classify_image(image_path):
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0)

    with torch.no_grad():
        output = model(image)

    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    return probabilities
```

4. FastAPI приложение

Создайте основной файл приложения app.py.

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from starlette.requests import Request
import os
from models import classify_image

# Инициализация приложения
app = FastAPI()

# Подключение шаблонов
templates = Jinja2Templates(directory="templates")

# Хранилище загруженных файлов
app.mount("/uploads", StaticFiles(directory="uploads"),
        name="uploads")

# Обработка загрузки изображений и предсказаний
@app.post("/upload/")
async def upload_image(file: UploadFile = File(...)):
    file_location = f"uploads/{file.filename}"
```

```
    with open(file_location, "wb") as buffer:
        buffer.write(file.file.read())

    predictions = classify_image(file_location)
    class_probabilities = [(i, float(p)) for i, p in
        enumerate(predictions)]

    return templates.TemplateResponse("result.html", {"request":
        {}, "class_probabilities": class_probabilities})

# Страница для загрузки изображений
@app.get("/", response_class=HTMLResponse)
async def upload_form(request: Request):
    return templates.TemplateResponse("upload.html", {"request":
        request})
```

5. Шаблоны для загрузки и отображения результатов

5.1. Шаблон загрузки templates/upload.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Загрузка изображения</title>
</head>
<body>
    <h2>Загрузите изображение для классификации</h2>
    <form action="/upload/" enctype="multipart/form-data"
        method="post">
        <input name="file" type="file">
        <button type="submit">Загрузить</button>
    </form>
</body>
</html>
```

5.2. Шаблон отображения результатов templates/result.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Результаты классификации</title>
</head>
<body>
    <h2>Распределение по классам</h2>
    <ul>
```

```
        {% for class, prob in class_probabilities %}
        <li>Класс {{ class }}: {{ prob|round(2) }}</li>
        {% endfor %}
    </ul>
    <a href="/">Загрузить другое изображение</a>
</body>
</html>
```

6. Запуск приложения

Запустите сервер с помощью `uvicorn`:

```
uvicorn app:app --reload
```

Приложение будет доступно по адресу `http://127.0.0.1:8000`.

Задания

1. **Вывод топа классов:**
   Измените вывод результатов так, чтобы отображались только первые несколько классов с наивысшей вероятностью. Убедитесь, что классы сортируются по убыванию вероятности.

2. **Классификация нескольких изображений:**
   Реализуйте возможность загрузки и классификации нескольких изображений одновременно. Измените форму и обработку результатов, чтобы выводить предсказания для всех загруженных изображений.

3. **Анализ производительности:**
   Добавьте код для измерения времени, необходимого для классификации изображения, и отображайте его пользователю. Это позволит оценить производительность модели.

4. **Работа с автоматической документацией:**
   Изучите автоматическую документацию FastAPI. Проверьте доступные эндпоинты на /docs и /redoc. Попробуйте использовать API прямо из документации для загрузки изображений и получения результатов классификации.

5. **Логирование и мониторинг:**
   Добавьте логирование запросов и ответов. Используйте встроенные возможности FastAPI или сторонние библиотеки для логирования (например, loguru).