ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4 по дисциплине «Мобильные системы компьютерного зрения»

Выполнили:
Шишкин Никита Дмитриевич
Давыдов Иван Денисович
Группа P42143

Преподаватель:
Денисов А.К.

Санкт-Петербург
2022 год

Оглавление

Задание

Цель работы

Создать прикладную систему компьютерного зрения на базе Jetson Nano.

Тема

Классификация телефонов

Задание

1. Выбрать и зафиксировать в Google таблице курса тему проекта.
2. Разработать требования к системе.
3. Разработать архитектуру системы.
4. Реализовать систему обработки изображений с использованием ИИ и алгоритмов компьютерного зрения
5. Оценить потребление ресурсов при функционировании системы и технические характеристики

Выполнение

Требования к системе

В ходе обсуждения лабораторной были выявлены следующие требования:

1) Наличие user-интерфейса
2) Система развернута на Jetson nano
3) Система должна быть способна определять телефоны следующих брендов: Apple, Huawei, LG, Nokia, OnePlus, Oppo, Realme, Samsung, Sony, Vivo, Xiaomi.

Так же были составлены следующие технические требования:

1) Объем памяти, занимаемый моделью классификации, не должен превышать полтора гигабайта.
2) Время отклика не уточнялось
3) Точность не уточнялась

Архитектура и реализация системы

Подсистема обработки изображений состоит из трех частей:

1.) Распознавание объектов на изображении

Данное действие производилось при помощи модели SSD-Mobilenet-v2 тренированной на MS COCO датасете. Модель умеет распознавать 91 класс объектов, в который входит так же класс телефона. Сам выбор модели обусловлен её относительно небольшим размером, который позволит использовать её на Jetson Nano, а также встроенной поддержкой этой модели в jetson-inference библиотеке, предназначенной специально для Jetson продуктов.

Таким образом, используя SSD-Mobilenet-v2 на изображении выявлялись объекты с классом телефона. Полученные части изображения извлекались для использования в следующей части.

2.) Классификация изображения

Результатом работы предыдущей части подсистемы являлось изображение с телефоном. Для его классификации была использована модель Vision Transformer (ViT) (а именно - vit-base-patch16-224-in21k). Обучение производилось при помощи датасета изображений телефонов, содержащего информацию о ~50.000 разных телефонах с изображением и размеченным брендом. (https://www.kaggle.com/datasets/lasaljaywardena/mobile-smartphone-images-dataset)

Обучение производилось при помощи GoogleColab, однако из-за ограничений бесплатного использования ресурсов данной системы, удалось использовать только ~3.000 изображений. Итоговая оценка аккуратности составила 52.5%

Обученная модель была загружена в систему и использована для классификации при помощи библиотеки transformers.

3.) Совмещение результатов

По результатам работы прошлых двух частей были получены данные о bounding box предполагаемых телефонов на сцене, а так же метки для них. При помощи утилит из jetson-inference данная информация помещается на исходное изображение, что и является итоговым результатом подсистемы обработки изображений.

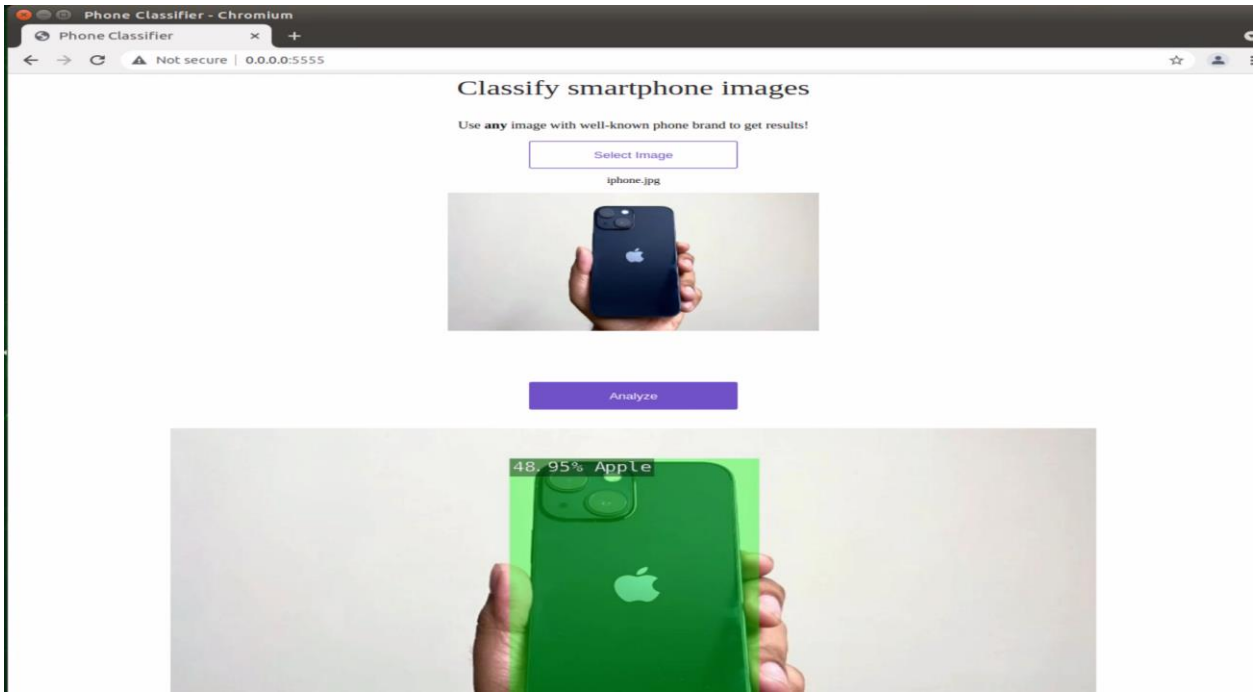
WebApp

Для использования классификатора в приложении, подсистема используется в рамках веб-сервера, написанном на python с применением FastAPI модуля. Клиентская сторона написана на чистом HTML + JS и выдается веб-сервером в виде статичных файлов.

При запуске веб сервера в систему загружаются обе модели.

При загрузке изображения на бэкенд изображение конвертируется из бинарного в объект PIL Image, который уже воспринимаем моделями. В результате обработки подсистема так же возвращает PIL Image, который уже конвертируется в base64 и передается клиенту для отображения.

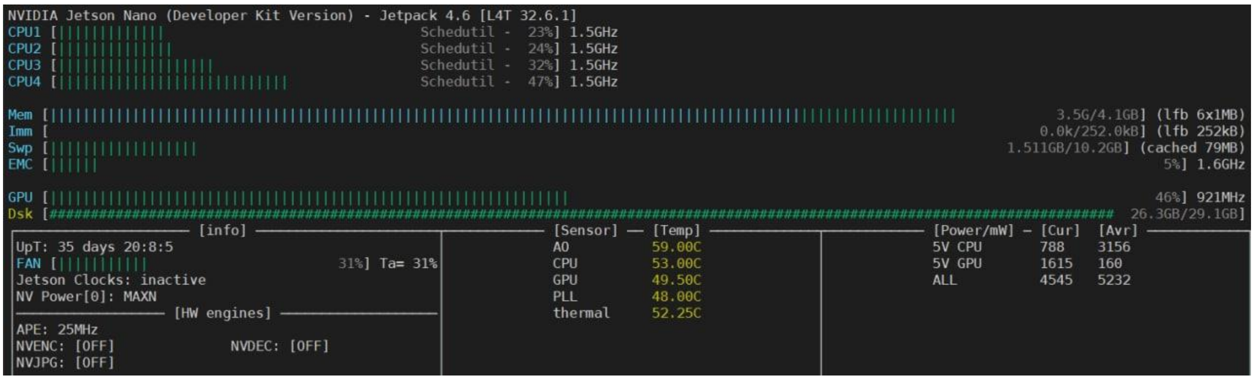
Пример использования веб-приложения:



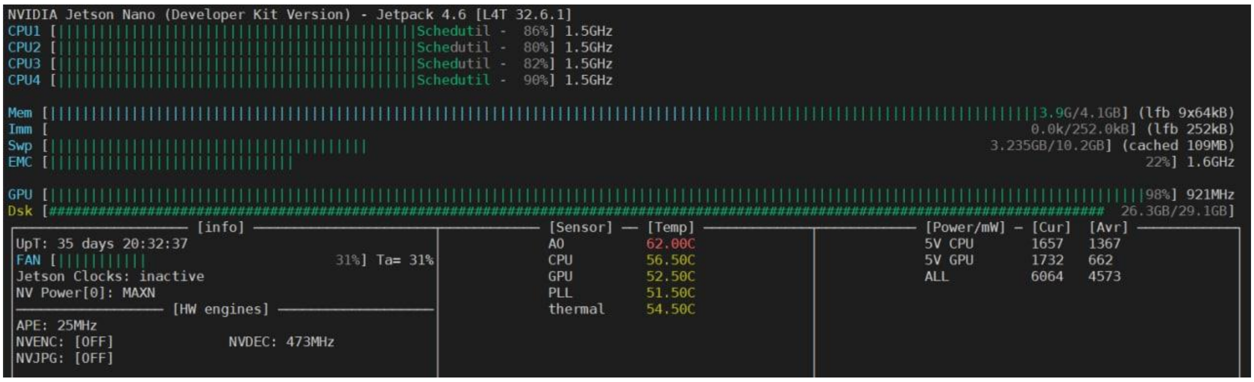
Пример использования моделей для обработки видеопотока

Оценка

Показатели использования ресурсов системы во время выполнения:



При обработке видео потока:



Временные показатели на тестовом скрипте:

Average detect time: 0.08378200183312098
Average classify time: 0.719015551937951
Average util time: 0.08180002123117447
Average exec time: 0.4352128550410271

Возможность применения реализованной системы в real-time приложениях:

$$\frac{\mathbf{1}}{\mathbf{0.4352}} = \mathbf{2.2977}\ \boldsymbol{fps}$$

В виду того, что в исходном датасете данные были не лучшего качества (преобладала продукция марки apple, остальные бренды были представлены в куда меньшем количестве), итоговая модель классификатора в большинстве случаев помечает изображение как телефон марки apple (в некоторых случаях как Samsung). Каким-либо образом повлиять

положительным образом на точность модели в данных условиях не представляется возможным, поскольку перечисленные негативные факторы влияющие на качество модели не зависят от разработчиков системы.

Вывод

В рамках лабораторной работы была обучена модель ViT для классификации моделей телефонов. Получившаяся модель была объединена в пайплайн с моделью detect-net и развернуты в рамках веб-сервера для использования внутри веб-приложения.

Список литературы

1. Transformers, URL: https://huggingface.co/transformers/v3.0.2/index.html
2. Jetson dusty-nv, URL: https://github.com/dusty-nv/jetson-inference
3. Датасет, URL: https://www.kaggle.com/datasets/lasaljaywardena/mobile-smartphone-images-dataset

Листинг

```
import sys
import math
import numpy as np
import base64

import jetson.inference
import jetson.utils

from transformers import pipeline

from PIL import Image
from fastapi import FastAPI, Request
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.responses import FileResponse
from fastapi.responses import PlainTextResponse

from io import BytesIO
import uvicorn, asyncio, requests #aiohttp


cellphone_class_ids = [
    77
]

app = FastAPI()
app.mount("/static", StaticFiles(directory="static"))
templates = Jinja2Templates(directory="templates")

detection_net = None
classifier = None
# font = None

# analyze image and draw phone boxes
def AnalyzeImage(input_pil):
    input_array = np.array(input_pil)
    input_cuda_image = jetson.utils.cudaFromNumpy(input_array)

    # detect objects in the image (with overlay)
    detections = detection_net.Detect(input_cuda_image, overlay="none")

    # print the detections
    print("detected {:d} objects in image".format(len(detections)))

    for detection in detections:
        print(detection)
        print(detection_net.GetClassDesc(detection.ClassID))
        if detection.ClassID in cellphone_class_ids:
            # Crop the image to use it in
```

```
            roi = (
                    math.floor(detection.Left),
                    math.floor(detection.Top),
                    math.ceil(detection.Right),
                    math.ceil(detection.Bottom))

            cropped_cuda_image = jetson.utils.cudaAllocMapped(
                    width=roi[2] - roi[0],
                    height=roi[3] - roi[1],
                    format=input_cuda_image.format)

            jetson.utils.cudaCrop(input_cuda_image, cropped_cuda_image, roi)

            cropped_array = jetson.utils.cudaToNumpy(cropped_cuda_image)

            cropped_pil_image = Image.fromarray(cropped_array, 'RGB')

            # Classify the image
            result = classifier(cropped_pil_image)
            print(result)
            confidence = result[0]['score']
            label = result[0]['label']

            # Overlay the bouding box
            jetson.utils.cudaDrawRect(
                    input_cuda_image,
                    (roi[0], roi[1], roi[2], roi[3]),
                    (0, 255, 0, 100))
            font = jetson.utils.cudaFont()
            # Overlay the text label
            font.OverlayText(
                    input_cuda_image,
                    roi[2] - roi[0],
                    roi[3] - roi[1],
                    "{:05.2f}% {:s}".format(confidence * 100, label),
                    roi[0] + 5,
                    roi[1] + 5,
                    font.White,
                    font.Gray40)

    output_array = jetson.utils.cudaToNumpy(input_cuda_image)
    output_pil = Image.fromarray(output_array, 'RGB')
    output_pil.save('./res.jpg')
    return output_pil


@app.on_event('startup')
async def setup_learner():
    global detection_net
    global classifier
```

```
    # global font
    print('Setup')

    # load the object detection detection_network
    detection_net = jetson.inference.detectNet("ssd-mobiledetection_net-v2",
sys.argv, 0.3) #opt.detection_network, sys.argv, opt.threshold)
    classifier = pipeline("image-classification",
model="/home/nano7/l4/checkpoint", device=0)

    # create utils
    # font = jetson.utils.cudaFont()

    print('Finish setting up nets')


@app.get('/')
async def root(request: Request):
    return templates.TemplateResponse("index.html", {'request':request})


@app.post('/analyze', response_class=PlainTextResponse)
async def analyze(request: Request):
    print("analyze")
    data = await request.form()
    if data['file'] == 'undefined':
        return {'result': 'undefined'}

    img_bytes = await (data['file'].read())
    img = Image.open(BytesIO(img_bytes))
    res = AnalyzeImage(img)
    # res.save('./res.jpg')
#    res = img

    buffer = BytesIO()
    res.save(buffer, format="JPEG")
    img_str = base64.b64encode(buffer.getvalue())
    return img_str


if __name__ == '__main__':
    if 'serve' in sys.argv: uvicorn.run(app=app, host='0.0.0.0', port=5555)
```