Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Мобильные системы компьютерного зрения

Лабораторная работа №4

ResNet-18, классификация изображений

Преподаватель: Денисов Алексей Константинович

Выполнили: Демьяновский Савелий Игоревич,
Юров Максим Алексеевич,
группа P4115

Санкт-Петербург
2024

Содержание

Описание решаемой задачи

Цель работы:

Изучить основы реализации глубоких нейронных сетей на мобильных системах, а также методы их оптимизации.

Задание:

1. Изучить принципы построения глубоких нейронных сетей, их разновидности, архитектуры (применительно к обработке изображений и видео).
2. Изучить способы реализации нейросетевых вычислений (CPU, GPU).
3. Реализовать систему обработки изображений на основе нейронной сети (назначение и архитектуру сети выбрать самостоятельно, это может быть предобученная сеть для детектирования объектов, сегментации, классификации, построения карты глубины, вычисления оптического потока). Реализация обучения сети не требуется. Приложение должно принимать на вход реальное изображение (изображения) и выводить результат (обработанное изображение или полученную из него информацию).
4. Оптимизировать выбранную сеть с помощью TensorRT.
5. Оценить следующие характеристики:
    5.1 Время выполнения программы и количество используемой памяти при использовании сети без оптимизации.
    5.2 Производительность и потребление памяти при использовании TensorRT.
    5.3 Изменение выхода сети (числовых значений) при использовании TensorRT при одинаковых входных данных.
    5.4 Возможность применения реализованной системы в real-time приложениях.

Теоретическая база

Классификация изображений: Задача классификации изображений заключается в том, чтобы отнести изображение к одной из заранее определенных категорий. Для этого используется сверточная нейронная сеть, которая на основе обученных весов извлекает признаки из входного изображения, а затем относит его к соответствующему классу.

ResNet (Residual Networks): ResNet — это архитектура глубоких нейронных сетей, предложенная для решения задачи распознавания изображений. Ключевая идея ResNet заключается во введении "остаточных

связей" (skip connections), которые позволяют передавать информацию через несколько слоев, избегая проблемы затухания градиентов и улучшая обучение глубоких сетей.

TensorRT: Это библиотека, предназначенная для ускорения выполнения нейронных сетей на графических процессорах (GPU) путем применения различных оптимизаций, таких как квантование, fusing (схлопывание) слоев и использование FP16. TensorRT может значительно ускорить работу модели без значительных потерь в точности.

Описание разработанной системы

Разработанная система предназначена для классификации изображений с использованием предобученной модели ResNet18. Система поддерживает два режима работы: обычный режим с использованием стандартной модели ResNet18 и режим с оптимизацией с помощью TensorRT, что позволяет ускорить обработку и повысить производительность на устройствах с поддержкой CUDA.

Основной принцип работы системы заключается в автоматической загрузке изображений из заранее определенной папки img, их предобработке и последующей классификации. Результаты классификации добавляются на изображения, после чего они сохраняются в выходную папку output. Система предоставляет два ключевых этапа обработки изображений:

1. **Предобработка изображений**: изображения подготавливаются для подачи в модель путем изменения их размера до фиксированных 224x224 пикселей и нормализации. Это обеспечивает совместимость с входным форматом модели ResNet18 и стандартными методами глубокого обучения для задач классификации.
2. **Классификация изображений**: изображение передаётся на вход модели, после чего результат обработки — это индекс класса, который соответствует определенному объекту на изображении.

Если пользователь запускает систему с флагом -trt, происходит оптимизация модели с использованием TensorRT. Это включает в себя следующие этапы:

- Преобразование стандартной модели ResNet18 в формат, совместимый с TensorRT.
- Сохранение оптимизированной модели для последующего использования.

- При следующем запуске с флагом -trt система загружает уже оптимизированную модель.

Ниже представлена часть кода, связанная с загрузкой модели, обработкой и классификацией изображения:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Обработка и классификация изображений
def batch_classify_images(images: list, model, class_dict: dict):
    print("Starting image processing...")
    timeimg = time.time()

    for image in images:
        index = classify_image(image, model)
        output_text = f"{index}: {class_dict[index]}"
        draw_and_save_image(image, output_text)

    print(f"Image processing time: {time.time() - timeimg:.2f} seconds")
    print(f'Memory allocated: {torch.cuda.memory_allocated()} bytes')

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
])

# Классфикация одного изображения
def classify_image(image: Image, model) -> int:
    image_tensor = transform(image).unsqueeze(0).to(device)
    output = model(image_tensor)
    return output.data.cpu().numpy().argmax()

# Подпись с результатами, сохранение изображения
def draw_and_save_image(image: Image, output_text: str):
    draw = ImageDraw.Draw(image)
       draw.rectangle((0, image.height - 20, image.width, image.height),
fill=(255, 255, 255))
       draw.text((50, image.height - 15), output_text, fill=(0, 0, 0),
font=ImageFont.load_default())
    image.save(output_path / Path(image.filename).name)

def main():
    # Проверка флага TensorRT
```

```
trt_enabled = '-trt' in sys.argv
if trt_enabled:
    sys.argv.remove('-trt')

class_dict = class_dict_init('./imagenet/classes.csv')

# Проверка и загрузка файлов из директории ./img
img_dir = Path('./img')
if not img_dir.exists() or not img_dir.is_dir():
    print(f"{img_dir} directory does not exist.")
    sys.exit(1)

image_extensions = ['*.jpg', '*.jpeg', '*.png', '*.bmp']
images = []
for ext in image_extensions:
    images.extend(img_dir.glob(ext))

image_list = []
for img_path in images:
    try:
        image_list.append(Image.open(img_path))
    except FileNotFoundError:
        print(f"{img_path} not found")

if not image_list:
    print(f"No images found in {img_dir}")
    sys.exit(1)

# Загрузка модели
load_time_start = time.time()

model = resnet18(pretrained=True).eval().to(device)

# TensorRT - оптимизация при наличии флага
if trt_enabled:
    print("Optimizing model with TensorRT...")
    x = torch.ones((1, 3, 224, 224)).to(device)
    model_trt = torch2trt(model, [x])  # Оптимизация модели
    #torch.save(model_trt.state_dict(), 'resnet18_trt.pth')
    model = model_trt  # Use the optimized model
else:
    print("Using regular ResNet18 model.")

# Проверка наличия существующей TensorRT модели
```

```
    #try:
        #model_trt = TRTModule()
            #model_trt.load_state_dict(torch.load('resnet18_trt.pth'))    #
Загрузка оптимизированной модели
        #model = model_trt
        #print("Loaded existing TensorRT model.")
    #except FileNotFoundError:
            #print("No existing TRT model found, proceeding with regular
model.")

    load_time_end = time.time()
    print(f"Model load time: {load_time_end - load_time_start:.2f} seconds")

    # Обработка изображений
    batch_classify_images(image_list, model, class_dict)

if __name__ == "__main__":
    main()
```

Результаты работы и тестирования системы

Для проведения тестирования системы был использован набор из 15 тестовых изображений. В результате проведения работы они были классифицированы моделью ResNet-18 в рамках набора данных ImageNet. Ниже приведены результаты корректно классифицированных изображений (рис. 1-2), а также изображений, классы для которых были определены неверно (рис. 3-5):

Рисунок 1. Корректно классифицированное изображение морской звезды.

Рисунок 2. Корректно классифицированное изображение ламы.

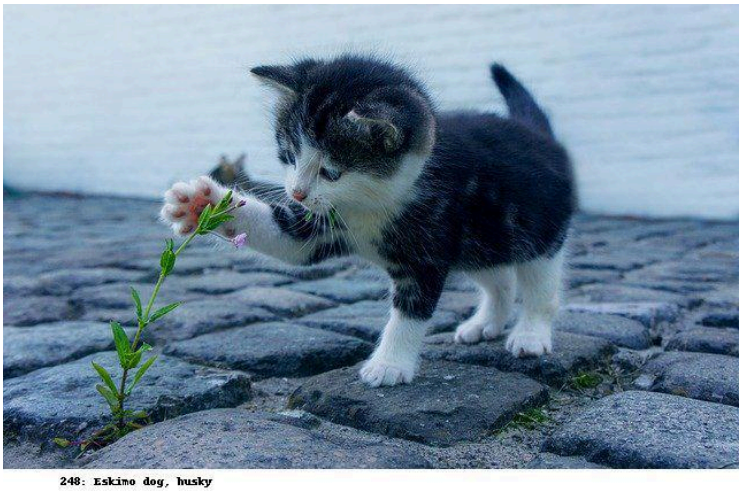

Рис. 3-5. Примеры некорректно определенных изображений.

Время выполнения программы и количество используемой памяти

В результате выполнения программы без оптимизации, с использованием TensorRT и загрузкой готовой модели, были получены следующие результаты (представлены средние значения времени выполнения):

```
Using regular ResNet18 model.
No existing TRT model found, proceeding with regular model.
Model load time: 14.69 seconds
Starting image processing...
Image processing time: 23.55 seconds
Memory allocated: 46861312 bytes

Optimizing model with TensorRT...
Model load time: 96.37 seconds
Starting image processing...
Image processing time: 3.20 seconds
Memory allocated: 47463424 bytes

Loaded existing TensorRT model.
```

```
Model load time: 12.48 seconds
Starting image processing...
Image processing time: 3.22 seconds
Memory allocated: 0 bytes
```

Полученные значения представлены в таблице 1 (данные об использовании памяти приведены с помощью *torch.cuda.memory_allocated()*):

	Загрузка модели, с	**Обработка изображений, с**	**Использование памяти, байт**
Без оптимизации	*14,69*	*23,55*	*46861312*
TensorRT	*96,37*	*3,20*	*47463424*
TensorRT (загрузка модели)	*12,48*	*3,22*	*0*

Таблица 1. Время выполнения программы и количество используемой памяти программы без оптимизации, с использованием TensorRT и загрузкой готовой модели.

В случае использования обычной модели ResNet18, процесс загрузки модели занимает около 14.69 секунд, а обработка изображений — 23.55 секунд. После оптимизации модели с использованием TensorRT, время загрузки увеличивается до 96.37 секунд, так как происходит процесс преобразования модели и её оптимизация для ускорения инференса. Однако время обработки изображений существенно снижается — до 3.20 секунд. Это достигается за счёт использования TensorRT, который позволяет значительно ускорить вычисления.

Использование TensorRT демонстрирует значительное улучшение производительности по времени обработки изображений. Хотя первоначальная оптимизация требует больше времени и памяти, при повторном использовании оптимизированной модели время загрузки снижается, а использование памяти становится более эффективным.

Изменение выхода сети (числовых значений) при использовании TensorRT при одинаковых входных данных

При одинаковых входных данных в рамках данной лабораторной работы изменений выхода сети, а в частности классов, определяемых на изображениях, обнаружено не было. Небольшие изменения в результатах вычислениях возможны из-за оптимизации вычислений и сокращения точности TensorRT.

Возможность применения реализованной системы в real-time приложениях

Реализованная система демонстрирует хорошие перспективы для использования в реальных приложениях благодаря значительному снижению времени инференса при оптимизации с использованием TensorRT. Время обработки изображений сократилось с 23.55 секунд до 3.2 секунд, что приближает систему к требованиям реального времени.

Потребление памяти также остается на приемлемом уровне: около 46 MB без оптимизации и 47 MB с TensorRT. Это позволяет системе эффективно обрабатывать изображения, не перегружая ресурсы.

Устойчивость к задержкам при инференсе делает систему подходящей для задач распознавания изображений. Несмотря на достигнутые улучшения, для полного соответствия требованиям реального времени в условиях высоких нагрузок может потребоваться дополнительная оптимизация и специализированное аппаратное обеспечение.

Вывод

В результате выполнения данной работы мы ознакомились с основами реализации глубоких нейронных сетей на мобильных системах и изучили методы их оптимизации. Мы рассмотрели архитектуру ResNet18 и применили её для классификации изображений с использованием библиотек PyTorch и TensorRT. В процессе работы были реализованы подходы к оптимизации, которые позволили значительно ускорить процесс инференса без значительных потерь в точности.

Источники

https://pytorch.org/vision/main/models/resnet.html
https://habr.com/ru/companies/vk/articles/359214/