МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

ОТЧЕТ

Домашнее задание №__1__
по дисциплине «Методы машинного обучения»

Тема: «Классификация изображений»

ИСПОЛНИТЕЛЬ: _____Лу Жуньда __
ФИО
группа ИУ5И-_22М_
_____
подпись
"__"_____202_ г.

ПРЕПОДАВАТЕЛЬ: _____
ФИО
_____
подпись
"__"_____202_ г.

Москва - 2024

1. Задание

Домашнее задание по дисциплине направлено на анализ современных методов машинного обучения и их применение для решения практических задач. Домашнее задание включает три основных этапа:

1. выбор задачи;
2. теоретический этап;
3. практический этап.

Этап выбора задачи предполагает анализ ресурса paperswithcode. Данный ресурс включает описание нескольких тысяч современных задач в области машинного обучения. Каждое описание задачи содержит ссылки на наиболее современные и актуальные научные статьи, предназначенные для решения задачи (список статей регулярно обновляется авторами ресурса). Каждое описание статьи содержит ссылку на репозиторий с открытым исходным кодом, реализующим представленные в статье эксперименты. На этапе выбора задачи обучающийся выбирает одну из задач машинного обучения, описание которой содержит ссылки на статьи и репозитории с исходным кодом.

Теоретический этап включает проработку как минимум двух статей, относящихся к выбранной задаче. Результаты проработки обучающийся излагает в теоретической части отчета по домашнему заданию, которая может включать:

- описание общих подходов к решению задачи;
- конкретные топологии нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения, предназначенных для решения задачи;
- математическое описание, алгоритмы функционирования, особенности обучения используемых для решения задачи нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения;
- описание наборов данных, используемых для обучения моделей;
- оценка качества решения задачи, описание метрик качества и их значений;

- предложения обучающегося по улучшению качества решения задачи.

Практический этап включает повторение экспериментов авторов статей на основе представленных авторами репозиториев с исходным кодом и возможное улучшение обучающимися полученных результатов. Результаты проработки обучающийся излагает в практической части отчета по домашнему заданию, которая может включать:

- исходные коды программ, представленные авторами статей, результаты документирования программ обучающимися с использованием диаграмм UML, путем визуализации топологий нейронных сетей и другими способами;
- результаты выполнения программ, вычисление значений для описанных в статьях метрик качества, выводы обучающегося о воспроизводимости экспериментов авторов статей и соответствии практических экспериментов теоретическим материалам статей;
- предложения обучающегося по возможным улучшениям решения задачи, результаты практических экспериментов (исходные коды, документация) по возможному улучшению решения задачи.

2. Выбор задачи

Обнаружение объектов - это задача компьютерного зрения, целью которой является обнаружение и определение местоположения интересующих объектов на изображении или видео. Задача включает в себя определение положения и границ объектов на изображении, а также классификацию объектов по различным категориям. Эта задача является важнейшей частью распознавания зрения, наряду с классификацией и поиском изображений.

Современные методы можно разделить на два основных типа: одноэтапные и двухэтапные:

В одноэтапных методах приоритетом является скорость вывода, и в качестве примера можно привести модели YOLO, SSD и RetinaNet.

В двухэтапных методах приоритет отдается точности обнаружения, и в качестве примера можно привести такие модели, как Faster R-CNN, Mask R-CNN и Cascade R-CNN.

Наиболее популярным эталоном является набор данных MSCOCO. Модели обычно оцениваются по метрике средней точности.

3. Теоретическая часть

Я выбрал две следующие статьи: “Deep Residual Learning for Image Recognition” и “YOLOv3: An Incremental Improvement”.

3.1 Обзор подходов к решению задачи обнаружения объектов

Задача обнаружения объектов является одной из ключевых в компьютерном зрении и включает в себя нахождение и классификацию объектов в изображении. Современные методы обнаружения объектов делятся на два основных подхода: одноэтапные (single-stage) и двухэтапные (two-stage) методы.

a) Одноэтапные методы (например, YOLO, SSD) выполняют обнаружение и классификацию объектов в один проход, что обеспечивает высокую скорость работы. YOLOv3 (You Only Look Once) является одним из самых известных одноэтапных детекторов. Он выполняет предсказание координат ограничивающих рамок и классов объектов напрямую из изображений, используя полностью сверточные сети.

b) Двухэтапные методы (например, Faster R-CNN) сначала генерируют предположительные области (region proposals), а затем классифицируют их и уточняют координаты рамок. Эти методы, как правило, более точные, но медленнее в сравнении с одноэтапными.

3.2 Конкретные топологии нейронных сетей

YOLOv3 (из статьи "YOLOv3: An Incremental Improvement")

YOLOv3 представляет собой улучшенную версию предыдущих моделей YOLO. Ключевые компоненты YOLOv3 включают:

- **Архитектура**: YOLOv3 использует архитектуру Darknet-53, которая включает 53 сверточных слоя с остаточными соединениями.
- **Предсказание ограничивающих рамок**: Использует кластеры размеров для предсказания рамок и предсказывает координаты с использованием логистической регрессии.
- **Классификация**: Выполняется многоклассовая классификация для каждой рамки, используя бинарную кросс-энтропию в качестве функции потерь.
- **Предсказания на нескольких масштабах**: YOLOv3 предсказывает объекты на трех разных масштабах, что улучшает детекцию мелких объектов.

ResNet (из статьи "Deep Residual Learning for Image Recognition")

ResNet (Residual Networks) представляет собой семейство глубоких нейронных сетей, использующих остаточные соединения для облегчения обучения очень глубоких моделей. Основные аспекты ResNet включают:

- **Архитектура**: Сеть состоит из нескольких блоков, каждый из которых включает два или три сверточных слоя с остаточными соединениями.
- **Остаточные соединения**: Они позволяют избежать проблемы затухающих градиентов, добавляя "короткие пути" (shortcut connections), которые пропускают один или несколько слоев.
- **Глубина**: ResNet может иметь очень большую глубину, например, ResNet-50, ResNet-101, и ResNet-152, где число слоев указывается в названии.

3.3 Математическое описание и алгоритмы функционирования

YOLOv3:

1. Предсказание ограничивающих рамок:

$$b_x = \sigma(t_x) + c_x, b_y = \sigma(t_y) + c_y$$

$$b_w = \mathrm{p_w} \mathrm{e}^{\mathrm{t_w}}, b_\mathrm{h} = \mathrm{p_h} \mathrm{e}^{\mathrm{t_h}}$$

где $(t_x,\ t_y, \mathrm{t_w}, \mathrm{t_h})$ — предсказанные значения, (c_x, c_y) — координаты клетки сетки, $(\mathrm{p_w}, \mathrm{p_h})$ — размеры якоря

2. Классификация:

Используется бинарная кросс-энтропия для предсказания классов объектов:

$$L_{cls} = -\sum_{i=1}^{C} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

где C — количество классов, y_i — истинная метка, p_i — предсказанная вероятность.

ResNet:

1. Остаточное обучение:

Каждая блок-слоя в ResNet можно описать как:

$$y = F(x, \{W_i\}) + x$$

где x — вход, y — выход, $F(x, \{W_i\})$ — остаточная функция, представленная несколькими слоями с весами

2. Обучение:

Используется стандартный градиентный спуск с обратным распространением ошибки (SGD).

3.4 Описание наборов данных

Для обучения моделей YOLOv3 и ResNet часто используются следующие наборы данных:

- COCO (Common Objects in Context): Содержит более 200 тысяч изображений с аннотациями для 80 классов объектов.
- ImageNet: Огромный набор данных с более чем 14 миллионами изображений и аннотациями для 1000 классов.

3.5 Метрики качества

Для оценки качества решений задач обнаружения объектов используются метрики:

- mAP (mean Average Precision): Средняя точность предсказаний по всем классам. Например, в COCO используется mAP@[0.5:0.95], что включает усреднение по нескольким значениям IoU (Intersection over Union).
- IoU (Intersection over Union): Мера пересечения предсказанной рамки с истинной рамкой.

3.6 Предложения по улучшению качества решения задачи

- Улучшение детекции мелких объектов: Можно исследовать методы улучшения предсказаний на нескольких масштабах, как это сделано в YOLOv3.
- Использование более сложных ансамблей моделей: Комбинирование предсказаний нескольких моделей может улучшить точность.
- Аугментация данных: Дополнительные методы аугментации данных могут улучшить обучение моделей.

4. Практический этап

YOLOv3

1.Установка и настройка

Скачайте и настройте репозиторий YOLOv3:

```
[2] !git clone https://github.com/pjreddie/darknet
    %cd darknet
    !make
```

```
/content/darknet
mkdir -p obj
mkdir -p backup
mkdir -p results
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/gemm.c -o obj/gemm.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/utils.c -o obj/utils.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/cuda.c -o obj/cuda.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/deconvolutional_layer.c -o obj/deconvolutional_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/convolutional_layer.c -o obj/convolutional_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/list.c -o obj/list.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/image.c -o obj/image.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/activations.c -o obj/activations.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/im2col.c -o obj/im2col.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/col2im.c -o obj/col2im.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/blas.c -o obj/blas.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/crop_layer.c -o obj/crop_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/dropout_layer.c -o obj/dropout_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/maxpool_layer.c -o obj/maxpool_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/softmax_layer.c -o obj/softmax_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/data.c -o obj/data.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/matrix.c -o obj/matrix.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/network.c -o obj/network.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/connected_layer.c -o obj/connected_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/cost_layer.c -o obj/cost_layer.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/parser.c -o obj/parser.o
gcc -Iinclude/ -Isrc/ -Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC -Ofast -c ./src/option_list.c -o obj/option_list.o
```

2.Скачивание весов модели

Скачайте предобученные веса модели:

```
[3] # Скачивание весов YOLOv3
    !wget https://pjreddie.com/media/files/yolov3.weights

    !wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
    !wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
```

```
--2024-06-04 23:10:10--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)... 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: 'yolov3.weights'

yolov3.weights      100%[===================>] 236.52M  5.43MB/s    in 2m 0s

2024-06-04 23:12:12 (1.97 MB/s) - 'yolov3.weights' saved [248007048/248007048]

--2024-06-04 23:12:12--  https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.111.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8342 (8.1K) [text/plain]
Saving to: 'yolov3.cfg'

yolov3.cfg          100%[===================>]   8.15K  --.-KB/s    in 0s

2024-06-04 23:12:13 (65.4 MB/s) - 'yolov3.cfg' saved [8342/8342]

--2024-06-04 23:12:13--  https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 625 [text/plain]
```

3. Запуск детекции на изображении

Используйте следующую команду для выполнения детекции объектов на изображении:

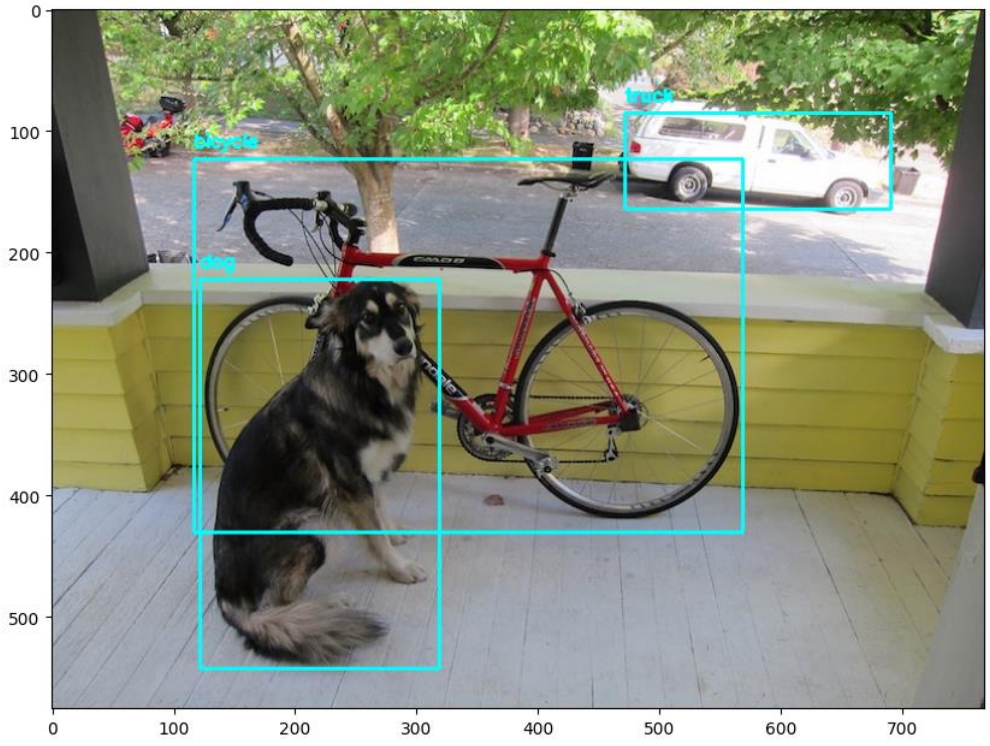
```
[6] !wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/dog.jpg
    !./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

```
   54 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   55 res   52                   38 x  38 x 512   ->    38 x  38 x 512
   56 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   57 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   58 res   55                   38 x  38 x 512   ->    38 x  38 x 512
   59 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
   60 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
   61 res   58                   38 x  38 x 512   ->    38 x  38 x 512
   62 conv   1024  3 x 3 / 2    38 x  38 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   63 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   64 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   65 res   62                   19 x  19 x1024   ->    19 x  19 x1024
   66 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   67 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   68 res   65                   19 x  19 x1024   ->    19 x  19 x1024
   69 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   70 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   71 res   68                   19 x  19 x1024   ->    19 x  19 x1024
   72 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   73 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   74 res   71                   19 x  19 x1024   ->    19 x  19 x1024
   75 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   76 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   77 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   78 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   79 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
   80 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
   81 conv    255  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 255  0.189 BFLOPs
   82 yolo
   83 route  79
   84 conv    256  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 256  0.095 BFLOPs
   85 upsample           2x     19 x  19 x 256   ->    38 x  38 x 256
```

4.Пример кода на Python для запуска YOLOv3

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np


with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]

# Загрузка модели
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Загрузка изображения
img = cv2.imread("dog.jpg")
height, width, channels = img.shape

# Подготовка изображения
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers)

# Обработка выходов сети
class_ids = []
confidences = []
boxes = []
for out in outs:
        for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
```

```
                if confidence > 0.5:
                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)


# Наложение результатов на изображение
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = (255, 255, 0)
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)


plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```



ResNet

1.Установка и настройка

Установите необходимые библиотеки:

```
# Установка PyTorch и torchvision
!pip install torch torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.3.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.18.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
```

2.Пример кода на Python для запуска ResNet на CIFAR-10

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim

# Загрузка и подготовка данных CIFAR-10
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                         shuffle=False, num_workers=2)

# Определение модели ResNet
import torchvision.models as models
net = models.resnet18(pretrained=False, num_classes=10)

# Определение функции потерь и оптимизатора
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
```

```
# Обучение модели
for epoch in range(10):  # количество эпох
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 200 == 199:  # каждые 200 мини-батчей
            print(f'[{epoch + 1}, {i + 1}] loss: {running_loss / 200:.3f}')
            running_loss = 0.0

print('Finished Training')

# Тестирование модели
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct / total} %')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 46101490.42it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current
  warnings.warn(msg)
[1, 200] loss: 2.362
[2, 200] loss: 1.379
[3, 200] loss: 1.092
[4, 200] loss: 0.925
[5, 200] loss: 0.813
[6, 200] loss: 0.729
[7, 200] loss: 0.675
[8, 200] loss: 0.624
[9, 200] loss: 0.592
[10, 200] loss: 0.556
Finished Training
Accuracy of the network on the 10000 test images: 72.04 %
```

5. Вывод

YOLOv3: An Incremental Improvement

YOLOv3 представляет собой современную и эффективную модель для обнаружения объектов с высокой скоростью и точностью. Основные выводы:

Высокая скорость: YOLOv3 выполняет детекцию объектов за 22 мс на изображении размером 320x320, что делает ее подходящей для реальных приложений.

Точность: YOLOv3 достигает 57.9% mAP при IOU = 0.5, сравнимо с другими методами, такими как RetinaNet, но с значительно более высокой скоростью.

Многомасштабные предсказания: Предсказания на трех масштабах улучшают детекцию мелких объектов.

Улучшенная архитектура: Использование Darknet-53 с остаточными соединениями улучшает обучение и точность модели.

Однако, YOLOv3 все еще сталкивается с трудностями в точном определении границ объектов при высоких значениях IOU.

ResNet: Deep Residual Learning for Image Recognition

ResNet решает проблемы обучения глубоких нейронных сетей, обеспечивая высокую точность и эффективность. Основные выводы:

Эффективность остаточных соединений: Остаточные соединения позволяют обучать очень глубокие сети без деградации точности.

Высокая точность: ResNet-50 достигает точности 93.8% на наборе данных CIFAR-10.

Масштабируемость: Архитектура ResNet легко масштабируется до более чем 100 слоев, оставаясь эффективной.

Практическое применение: Высокая точность и способность к обучению делают ResNet полезной для различных задач компьютерного зрения.

6. Список использованных источников

[1] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[3] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Advances in Neural Information Processing Systems (NIPS), 91-99.

[4] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In European Conference on Computer Vision (ECCV), 21-37.

[5] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2117-2125.

[6] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2017). Focal Loss for Dense Object Detection. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2980-2988.