Одеський національний університет імені І. І. Мечникова

Факультет математики, фізики та інформаційних технологій

Кафедра математичного та комп'ютерного моделювання

Дипломна робота

на здобуття ступеня вищої освіти «магістр»

на тему: **«Оптичне розпізнавання та ідентифікація об'єктів»**

«Optical recognition and identification of objects»

Виконав: студент денної форми навчання
спеціальності 113 Прикладна математика
Куперман Антон Олександрович

Керівник: к. ф.-м. н., доц. Таірова М. С. _____
Рецензент: к. ф.-м. н., доц. Вербіцький В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ____ від _____ 2021 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ____ від _____ 2021 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2021 р.

Зміст

ВСТУП

Задача оптичного розпізнавання об'єктів є досить актуальною на сьогоднішній день. Вона використовується в роботі поліції, в автоматизації виробництва, перевірки безпеки, тощо.

У даній роботі розглядається розпізнавання деяких об'єктів з камер дорожнього спостереження. Результат цієї роботи може бути використаний для допомоги оцінки екологічного становища, пошука порушника ПДР або з метою отримання статистики.

Використання методів машинного навчання для вирішення задач оптичного розпізнавання об'єктів пов'язане з пошуком кращого рішення з метою економії часу та автоматизації процесів.

Для вирішення даної задачі оптичного розпізнавання об'єктів було реалізовано два методи машинного навчання. Було проведено тренування відповідних нейронних мереж на різних за розміром вибірках, які були попередньо відобрані та поділені на навчальну та тестувальну частини. Після тренування на зображеннях розглянуті методи було застосовано до розпізнавання об'єктів на відео з різною частотою кадрів. Для обробки відео наведені методи було модифіковано. Було проведено ідентифікацію наведених об'єктів з використанням окремої попередньо тренованої нейронної мережі.

Об'єктом дослідження даної роботи є задача оптичного розпізнавання об'єктів з камер дорожнього спостереження. Предметом дослідження є застосування методів машинного навчання, які найчастіше використовуються для вирішення задач оптичного розпізнавання об'єктів. Метою роботи є реалізація та порівняння цих методів.

ВИСНОВКИ

У цій роботі було розглянуто задачу оптичного розпізнавання об'єктів (зокрема світлофора, авто, людини, вантажівки та велосипеда). Було розглянуто та реалізовано метод і мережу SSD та метод і мережу YOLOv3. Було проведено тренування цих мереж з використанням різної кількості зображеннь. Отримані мережі було застовано до розпізнавання об'єктів на відео з різною частотою кадрів. Було проведено ідентифікацію наведених об'єктів, результати якої можно вважати задовільними.

Порівняння цих методів дало зрозуміти, що метод SSD підходить для нескладних задач, які потребують швидкого вирішення. Метод YOLOv3 краще підходить для вирішення задач багатокласової класифікації та задач досягнення максимальної точності. Крім того, підхід YOLOv3 дозволяє працювати з зображеннями будь-якої якості, що є досить корисним на практиці. Оптимізація вибірки та алгоритму роботи може покращити час тренування методу YOLOv3, але через складність її архітектури, час тренування YOLOv3 майже завжди буде довшим ніж в інших моделях.

Обидві моделі, попередньо треновані на зображеннях, було застосовано для розпізнавання об'єктів на відео. Як і у випадку з розпізнаванням зображень, метод YOLOv3 показав більшу точність, але обидва методи показали результат достатній для використовування їх як у розпізнаванні об'єктів на зображенях, так і у розпізнаванні об'єктів на відео.

Обидва методи показали достатній результат для початкової задачі. Отриманий результат може бути використаний на практиці для відстеження рівня пробок на дорогах, пошука вкрадених автівок, тощо. Можна вважати дану задачу оптимального розпізнавання об'єктів вирішеною.

СПИСОК ЛІТЕРАТУРИ

1. Uijlings, J.R., van de Sande, K.E., Gevers, T., Smeulders, A.W.: Selective search for object recognition. IJCV (2013)
2. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR.(2016)
4. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. In: ICLR. (2014)
5. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: NIPS. (2015)
6. Bell, S., Upchurch, P., Snavely, N., and Bala, K. Material recognition in the wild with the materials in context database. CoRR, abs/1412.0623, 2014.
7. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. In Proc. BMVC., 2014.
8. T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2117 – 2125, 2017.
9. M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303 – 338, 2010.
10. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770 – 778, 2016.
11. А. Т. Яровий Багатовимірний статистичний аналіз: навчально-методичний посібник. / А. Т. Яровий, Є. М. Страхов. – М:Астропринт, 2015. – 46с.

12. Yuyin Sun, Liefeng Bo and Dieter Fox. Attribute Based Object Identification. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 2096 - 2103, 2013.
13. А. О. Куперман, М. С. Таірова. Про методи оптичного розпізнавання об'єктів на відео. У рамках міжнародного наукового симпозіуму Інтелектуальні рішення-С, с. 51-52, 2021.

Додаток А

Реалізація методу SSD на Python 3

```
import torch
from matplotlib import pyplot as plt
import matplotlib.patches as patches
precision = 'fp32'
ssd_model = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_ssd', model_math=precision)
utils = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_ssd_processing_utils')
ssd_model.to('cuda')
ssd_model.eval()
uris = [
    'http://farm7.staticflickr.com/6154/6177305915_56793dd241_z.jpg']
inputs = [utils.prepare_input(uri) for uri in uris]
tensor = utils.prepare_tensor(inputs, precision == 'fp16')
with torch.no_grad():
    detections_batch = ssd_model(tensor)
results_per_input = utils.decode_results(detections_batch)
best_results_per_input = [utils.pick_best(results, 0.40) for results in results_per_input]
classes_to_labels = utils.get_coco_object_dictionary()
for image_idx in range(len(best_results_per_input)):
    fig, ax = plt.subplots(1)
    # Show original, denormalized image...
    image = inputs[image_idx] / 2 + 0.5
    ax.imshow(image)
    # ...with detections
    bboxes, classes, confidences = best_results_per_input[image_idx]
    for idx in range(len(bboxes)):
        left, bot, right, top = bboxes[idx]
        x, y, w, h = [val * 300 for val in [left, bot, right - left, top - bot]]
        rect = patches.Rectangle((x, y), w, h, linewidth=1, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        ax.text(x, y, "{} {:.0f}%".format(classes_to_labels[classes[idx] - 1], confidences[idx]*100),
        bbox=dict(facecolor='white', alpha=0.5))
plt.show()
```

```
import time
import torch.backends.cudnn as cudnn
import torch.optim
import torch.utils.data
from model import SSD300, MultiBoxLoss
from datasets import PascalVOCDataset
from utils import *

# Data parameters
data_folder = 'Диплом/'
keep_difficult = True


n_classes = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Learning parameters
checkpoint = None
batch_size = 8
iterations = 250
workers = 4
print_freq = 200
momentum = 0.9
weight_decay = 5e-4
grad_clip = None

cudnn.benchmark = True


def main():

    global start_epoch, label_map, epoch, checkpoint, decay_lr_at

    # Initialize model or load checkpoint
    if checkpoint is None:
        start_epoch = 0
        model = SSD300(n_classes=n_classes)
        biases = list()
        not_biases = list()
        for param_name, param in model.named_parameters():
            if param.requires_grad:
                if param_name.endswith('.bias'):
                    biases.append(param)
                else:
                    not_biases.append(param)
        optimizer = torch.optim.SGD(params=[{'params': biases, 'lr': 2 * lr}, {'params': not_biases}],
                                    lr=lr, momentum=momentum, weight_decay=weight_decay)

    else:
        checkpoint = torch.load(checkpoint)
        start_epoch = checkpoint['epoch'] + 1
        print('\nLoaded checkpoint from epoch %d.\n' % start_epoch)
        model = checkpoint['model']
```

```
        optimizer = checkpoint['optimizer']


    model = model.to(device)
    criterion = MultiBoxLoss(priors_cxcy=model.priors_cxcy).to(device)


    train_dataset = PascalVOCDataset(data_folder,
                                     split='train',
                                     keep_difficult=keep_difficult)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
                                               collate_fn=train_dataset.collate_fn, num_workers=workers,
                                               pin_memory=True)


    epochs = 250
    decay_lr_at = [x // (len(train_dataset) // 32) for x in decay_lr_at]

    # Epochs
    for epoch in range(start_epoch, epochs):

        # Decay learning rate at particular epochs
        if epoch in decay_lr_at:
            adjust_learning_rate(optimizer, decay_lr_to)

        # One epoch's training
        train(train_loader=train_loader,
              model=model,
              criterion=criterion,
              optimizer=optimizer,
              epoch=epoch)




def train(train_loader, model, criterion, optimizer, epoch):

    model.train()  # training mode enables dropout

    batch_time = AverageMeter()  # forward prop. + back prop. time
    data_time = AverageMeter()  # data loading time
    losses = AverageMeter()  # loss

    start = time.time()

    # Batches
    for i, (images, boxes, labels, _) in enumerate(train_loader):
        data_time.update(time.time() - start)

        # Move to default device
        images = images.to(device)  # (batch_size (N), 3, 300, 300)
        boxes = [b.to(device) for b in boxes]
        labels = [l.to(device) for l in labels]
```

```
        # Forward prop.
        predicted_locs, predicted_scores = model(images)  # (N, 8732, 4), (N, 8732, n_classes)

        # Loss
        loss = criterion(predicted_locs, predicted_scores, boxes, labels)  # scalar

        # Backward prop.
        optimizer.zero_grad()
        loss.backward()

        # Clip gradients, if necessary
        if grad_clip is not None:
            clip_gradient(optimizer, grad_clip)

        # Update model
        optimizer.step()

        losses.update(loss.item(), images.size(0))
        batch_time.update(time.time() - start)

        start = time.time()

        # Print status
        if i % print_freq == 0:
            print('Epoch: [{0}][{1}/{2}]\t'
                  'Batch Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
                  'Data Time {data_time.val:.3f} ({data_time.avg:.3f})\t'
                  'Loss {loss.val:.4f} ({loss.avg:.4f})\t'.format(epoch, i, len(train_loader),
                                                                  batch_time=batch_time,
                                                                  data_time=data_time, loss=losses))
    del predicted_locs, predicted_scores, images, boxes, labels

if __name__ == '__main__':
    main()

#Video recognising

import cv2
import numpy as np

filename = 'edmonton_canada.mp4'
file_size = (1920,1080)

output_filename = 'traffic_mobssd2.mp4'
output_frames_per_second = 20.0

RESIZED_DIMENSIONS = (300, 300) # Dimensions that SSD was trained on.
IMG_NORM_RATIO = 0.007843 # In grayscale a pixel can range between 0 and 255

# Load the pre-trained neural network
neural_network = cv2.dnn.readNetFromCaffe('MobileNetSSD_deploy.prototxt.txt',
        'MobileNetSSD_deploy.caffemodel')
```

```
# List of categories and classes
categories = { 0: 'background', 1: 'aeroplane', 2: 'bicycle', 3: 'bird',
               4: 'boat', 5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat',
               9: 'chair', 10: 'cow', 11: 'diningtable', 12: 'dog',
              13: 'horse', 14: 'motorbike', 15: 'person',
              16: 'pottedplant', 17: 'sheep', 18: 'sofa',
              19: 'train', 20: 'tvmonitor', 21: 'truck'}

classes =  ["background", "aeroplane", "bicycle", "bird", "boat", "bottle",
            "bus", "car", "cat", "chair", "cow",
           "diningtable",  "dog", "horse", "motorbike", "person",
           "pottedplant", "sheep", "sofa", "train", "tvmonitor", "truck"]

# Create the bounding boxes
bbox_colors = np.random.uniform(255, 0, size=(len(categories), 3))

def main():

  # Load a video
  cap = cv2.VideoCapture(filename)

  # Create a VideoWriter object so we can save the video output
  fourcc = cv2.VideoWriter_fourcc(*'mp4v')
  result = cv2.VideoWriter(output_filename,
                           fourcc,
                           output_frames_per_second,
                           file_size)

  # Process the video
  while cap.isOpened():

    # Capture one frame at a time
    success, frame = cap.read()

    # Do we have a video frame? If true, proceed.
    if success:

      # Capture the frame's height and width
      (h, w) = frame.shape[:2]

      # Create a blob. A blob is a group of connected pixels in a binary
      # frame that share some common property (e.g. grayscale value)
      # Preprocess the frame to prepare it for deep learning classification
      frame_blob = cv2.dnn.blobFromImage(cv2.resize(frame, RESIZED_DIMENSIONS),
                     IMG_NORM_RATIO, RESIZED_DIMENSIONS, 127.5)

      # Set the input for the neural network
      neural_network.setInput(frame_blob)

      # Predict the objects in the image
      neural_network_output = neural_network.forward()
```

```
      # Put the bounding boxes around the detected objects
      for i in np.arange(0, neural_network_output.shape[2]):

        confidence = neural_network_output[0, 0, i, 2]

        # Confidence must be at least 30%
        if confidence > 0.30:

          idx = int(neural_network_output[0, 0, i, 1])

          bounding_box = neural_network_output[0, 0, i, 3:7] * np.array(
            [w, h, w, h])

          (startX, startY, endX, endY) = bounding_box.astype("int")

          label = "{}: {:.2f}%".format(classes[idx], confidence * 100)

          cv2.rectangle(frame, (startX, startY), (
            endX, endY), bbox_colors[idx], 2)

          y = startY - 15 if startY - 15 > 15 else startY + 15

          cv2.putText(frame, label, (startX, y),cv2.FONT_HERSHEY_SIMPLEX,
            0.5, bbox_colors[idx], 2)

      # We now need to resize the frame so its dimensions
      # are equivalent to the dimensions of the original frame
      frame = cv2.resize(frame, file_size, interpolation=cv2.INTER_NEAREST)

            # Write the frame to the output video file
      result.write(frame)

    # No more video frames left
    else:
      break

  # Stop when the video is finished
  cap.release()

  # Release the video recording
  result.release()

main()
```

Додаток Б

Реалізація методу YOLOv3 на Python 3

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="plates")
trainer.setTrainConfig(object_names_array=[], batch_size=4, num_experiments=200,
train_from_pretrained_model="pretrained-yolov3.h5")
trainer.trainModel()



from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path ,
            "model.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=
os.path.join(execution_path , "1.jpg"), output_image_path=
os.path.join(execution_path , "imagenew.jpg"))

for eachObject in detections:
    print(eachObject["name"] ,
    " : " ,
    eachObject["percentage_probability"] )



#Video recognising

import cv2
import numpy as np
import argparse
import time

parser = argparse.ArgumentParser()
parser.add_argument('--play_video', help="Tue/False", default=True)
parser.add_argument('--video_path', help="Path of video file", default="videos/traffic.mp4")
parser.add_argument('--verbose', help="To print statements", default=True)
args = parser.parse_args()

#Load yolo
```

```
def load_yolo():
        net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
        classes = []
        with open("coco.names", "r") as f:
                classes = [line.strip() for line in f.readlines()]

        layers_names = net.getLayerNames()
        output_layers = [layers_names[i[0]-1] for i in net.getUnconnectedOutLayers()]
        colors = np.random.uniform(0, 255, size=(len(classes), 3))
        return net, classes, colors, output_layers

def detect_objects(img, net, outputLayers):
        blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0), swapRB=True,
        net.setInput(blob)
        outputs = net.forward(outputLayers)
        return blob, outputs

def get_box_dimensions(outputs, height, width):
        boxes = []
        confs = []
        class_ids = []
        for output in outputs:
                for detect in output:
                        scores = detect[5:]
                        class_id = np.argmax(scores)
                        conf = scores[class_id]
                        if conf > 0.3:
                                center_x = int(detect[0] * width)
                                center_y = int(detect[1] * height)
                                w = int(detect[2] * width)
                                h = int(detect[3] * height)
                                x = int(center_x - w/2)
                                y = int(center_y - h / 2)
                                boxes.append([x, y, w, h])
                                confs.append(float(conf))
                                class_ids.append(class_id)
        return boxes, confs, class_ids

def draw_labels(boxes, confs, colors, class_ids, classes, img):
        indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
        font = cv2.FONT_HERSHEY_PLAIN
        for i in range(len(boxes)):
                if i in indexes:
                        x, y, w, h = boxes[i]
                        label = str(classes[class_ids[i]])
                        color = colors[i]
                        cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
                        cv2.putText(img, label, (x, y - 5), font, 1, color, 1)
        cv2.imshow("Image", img)


def start_video(video_path):
        model, classes, colors, output_layers = load_yolo()
```

```
        cap = cv2.VideoCapture(video_path)
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        result = cv2.VideoWriter('traffic_yolo.mp4', fourcc, 20, (1920, 1080))
        while True:
                _, frame = cap.read()
                height, width, channels = frame.shape
                blob, outputs = detect_objects(frame, model, output_layers)
                boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
                draw_labels(boxes, confs, colors, class_ids, classes, frame)
                new_frame = cv2.resize(frame, (1920,1080), interpolation=cv2.INTER_NEAREST)
                # Write the frame to the output video file
                result.write(new_frame)
                key = cv2.waitKey(1)
                if key == 27:
                        break
        cap.release()
        result.release()

if __name__ == '__main__':
        video_play = args.play_video
        if video_play:
                video_path = args.video_path
                if args.verbose:
                        print('Opening '+video_path+" .... ")
                start_video(video_path)
        cv2.destroyAllWindows()
```