ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра «Електронних обчислювальних машин»

Кваліфікаційна робота
на тему:
Технологія розпізнавання шрифту Брайля на базі нейронної мережі

Виконав ст.гр СПм-22-2 Врублевський В.О.
Керівник:доц. Бовчалюк С.Я.

Мета і задачі дослідження

Часткові завдання дослідження:

- Аналіз існуючих технологій розпізнавання шрифту Брайля
- Аналіз сучасних технологій комп'ютерного зору
- Визначення найбільш оптимального алгоритму бінаризації
- Огляд сучасних середовищ та інструментів розробки
- Розробка та опис технології
- Аналіз ефективності технології

Існуючі методи розпізнавання шрифта Брайля

Алгоритми OBR можливо поділити на наступні кроки.

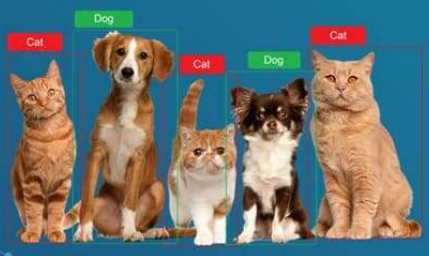
- Знайти точки
- Відновити уявну сітку
- Порівняти знайдені точки з вузлами сітки
- Розпізнати символи
- Конвертувати послідовність символів у звичайний текст

3

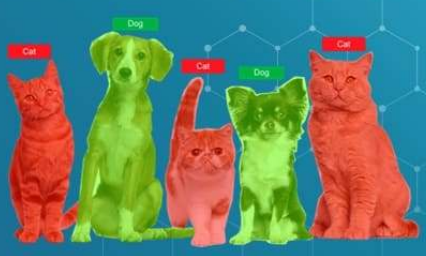
Нейронні мережі у комп’ютерному зорі

Класифікація

Детекція

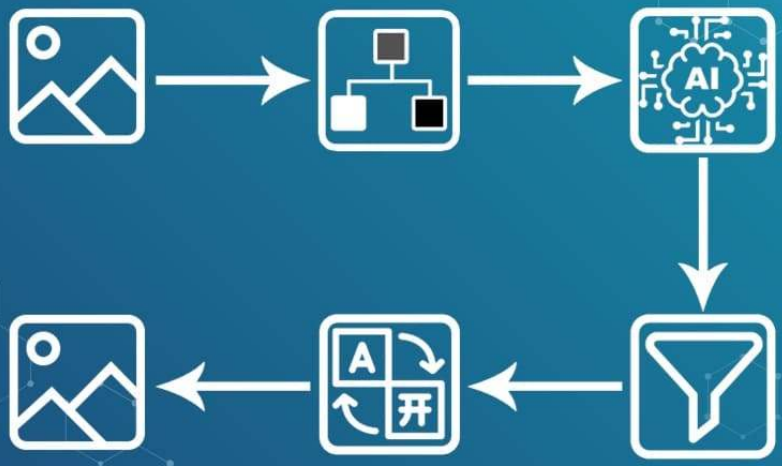


Сегментація



4

Алгоритм роботи технології



5

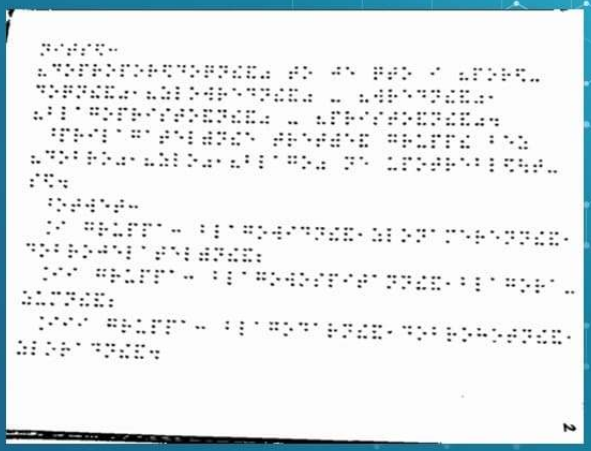
Бінаризація зображень

Переваги:

- Виділення об’єктів
- Зтиснення даних

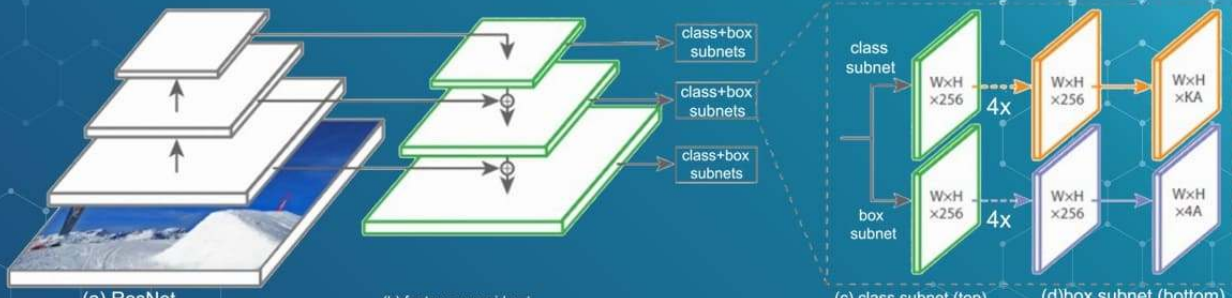
Недоліки:

- Втрата інформації
- Чутливість до шуму



6

Архітектура нейромережі RetinaNet

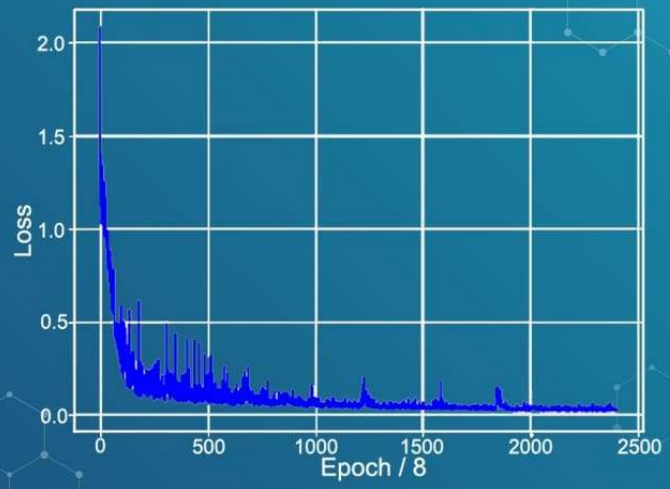


7

Створення датасету

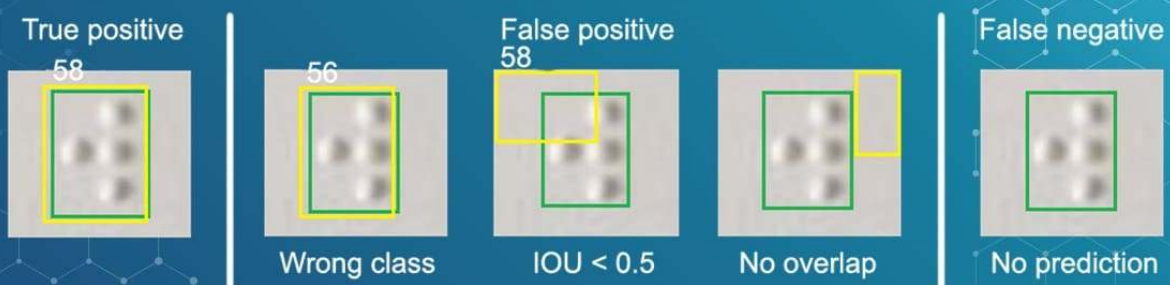
8

Навчання нейромережі



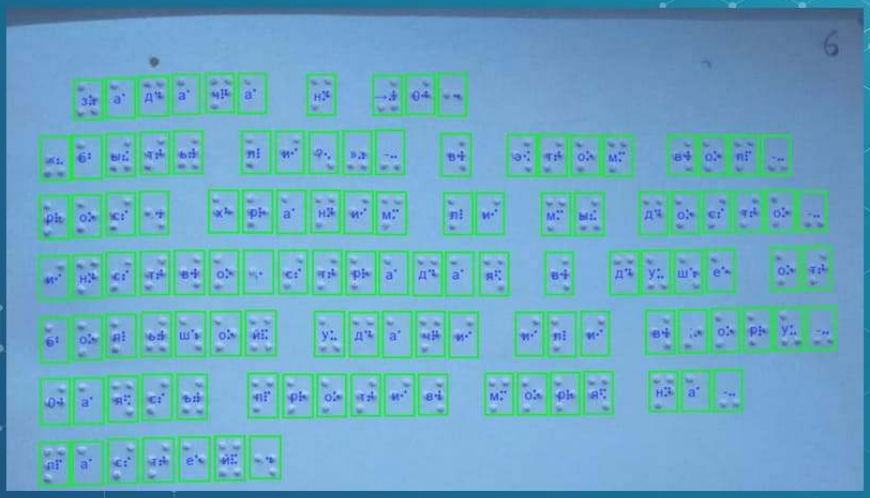
9

Тестування нейронної мережі



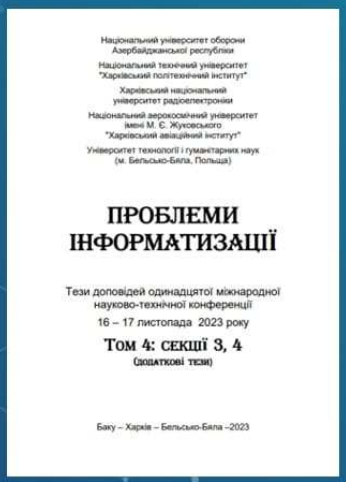
10

Результат роботи технології



11

Апробація результатів дослідження



Національний університет оборони
Азербайджанської республіки

Національний технічний університет
"Харківський політехнічний інститут"

Харківський національний
університет радіоелектроніки

Національний аерокосмічний університет
імені М. Є. Жуковського
"Харківський авіаційний інститут"

Університет технології і гуманітарних наук
(м. Бельсько-Біла, Польща)

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

Тези доповідей одинадцятої міжнародної
науково-технічної конференції
16 – 17 листопада 2023 року

Том 4: СЕКЦІЇ 3, 4
(додаткові тези)

Баку – Харків – Бельсько-Біла –2023

Problems of Informatization the eleventh international scientific and technical conference

ТЕХНОЛОГІЯ РОЗПІЗНАВАННЯ ШРИФТУ БРАЙЛЯ НА БАЗІ НЕЙРОННОЇ МЕРЕЖІ

Бончалюк С. Я., Ярублєвський В. О.
Харківський національний університет радіоелектроніки, Харків, Україна

Шрифт Брайля – це рельєфно-крапковий тактильний шрифт, який дає змогу людям з обмеженими можливостями зору читати і писати. Існує багато методів розпізнавання шрифту брайля, які об'єднані під терміном OBR (Optical Braille Recognition) [1]. У наш час бурхливого розвитку нейронних мереж і штучного інтелекту, з'являються дедалі нові сфери їх застосування, однією з популярних сфер застосування є машинний зір. У машинному зорі таким методом є конвертація зображень до відтінків сірого, як і повна бінаризація пікселів [2].

Метою доповіді є визначення оптимальних методів бінаризації зображень для зменшення кількості інформації про зображення без помітної втрати якості даних, а також виділення ключових ознак, необхідних для подальшої роботи із зображенням.

Бінаризація – це процес перетворення зображення з градацій сірого на двокольорні (бінарні) зображення, пікселі якого можуть мати лише два значення: чорний або білий. У контексті обробки зображень, бінаризація має на увазі використання порогового значення, для диференціювання класів пікселів чорного (об'єкт) і білого (фон). Методи знаходження порогу бінаризації можна розділити на дві умовні категорії - глобальні та локальні. У глобальних методах обробляється все зображення, використовуючи задалегідь підібраний поріг бінаризації, за допомогою якого відбувається диференціювання на чорний і білий колір. Методи бінаризації, за яких зображення ділиться на частини або береться вікно сплучення пікселя називають локальними.

Для того, щоб визначити найбільш ефективний алгоритм бінаризації, було обрано різні комбінації з пар: режим конвертації зображення в градації сірого та методу бінаризації. За результатами отриманих зображень можна зробити висновок, що найбільш оптимальними виявилися алгоритми Брейді, оскільки кінцевий результат містить найменшу кількість шуму, збережена ключова інформація з вихідного зображення, а обчислювальна складність поступається лише методам глобальної бінаризації.

Список літератури

1. Isayed, S. A review of optical Braille recognition / S. Isayed, R. Tahboub //2015 2nd World Symposium on Web Applications and Networking (WSWAN). – IEEE, 2015. – С. 1–6.

2. Romano Ho, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. В 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (с. 770–778).

12


Висновки
В ході даної роботи було розглянуто актуальність завдання перекладу шрифту Брайля на зображенні, розглянуті існуючі методи вирішення завдання, а також готові комерційні рішення. Були проаналізовані існуючі підходи машинного навчання, завдання комп'ютерного зору для яких ефективно застосування нейронних мереж, різні технології, які використовуються в сучасних архітектурах нейронних мереж, та найбільш сучасні архітектури, які активно застосовуються у цій галузі. У роботі наведено огляд сучасних середовищ розробки та засобів для розробки технологій на базі нейронної мережі, наведено обґрунтування обраних технологій. Проведено аналіз існуючих методів бінаризації. Знайдено найбільш оптимальний алгоритм для бінаризації зображень, що містять шрифт Брайля. Була виконана оптимізація бистродії бінаризації зображень, завдяки реалізації алгоритму на більш низькорівневому язику програмування. Розроблено технологію на базі нейронної мережі RetinaNet, яка виконує детекцію та переклад символів Брайля на зображеннях. Завдяки використанню найбільш сучасних хмарних провайдерів було здійснено швидкісне навчання нейронної мережі та її тестування. При аналізі ефективності, розроблена технологія показала 90% точності детекції та 98% точності класифікації символів шрифту Брайля на зображеннях.
13

ДОДАТОК Б

Наукові публікації за темою кваліфікаційної роботи

Національний університет оборони
Азербайджанської республіки

Національний технічний університет
"Харківський політехнічний інститут"

Харківський національний
університет радіоелектроніки

Національний аерокосмічний університет
імені М. Є. Жуковського
"Харківський авіаційний інститут"

Університет технології і гуманітарних наук
(м. Бельсько-Бяла, Польща)

ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ

Тези доповідей одинадцятої міжнародної
науково-технічної конференції

16 – 17 листопада 2023 року

Том 4: секції 3, 4

(додаткові тези)

Баку – Харків – Бельсько-Бяла –2023

ТЕХНОЛОГІЯ РОЗПІЗНАВАННЯ ШРИФТУ БРАЙЛЯ НА БАЗІ НЕЙРОННОЇ МЕРЕЖІ

Бовчалюк С. Я., Врублевський В. О.
Харківський національний університет радіоелектроніки, Харків, Україна

Шрифт Брайля – це рельєфно-крапковий тактильний шрифт, який дає змогу людям з обмеженими можливостями зору читати і писати. Існує багато методів розпізнавання шрифту брайля, які об'єднані під терміном OBR (Optical Braille Recognition) [1]. У наш час бурхливого розвитку нейронних мереж і штучного інтелекту, з'являються дедалі новіші сфери їх застосування, однією з популярних сфер застосування є машинний зір. У машинному зорі таким методом є конвертація зображень до відтінків сірого, як і повна бінаризація пікселів [2].

Метою доповіді є визначення оптимальних методів бінаризації зображень для зменшення кількості інформації про зображення без помітної втрати якості даних, а також виділення ключових ознак, необхідних для подальшої роботи із зображенням.

Бінаризація – це процес перетворення зображення з градацій сірого на двоколірне (бінарне) зображення, пікселі якого можуть мати лише два значення: чорний або білий. У контексті обробки зображень, бінаризація має на увазі використання порогового значення, для диференціювання класів пікселів чорного (об'єкт) і білого (фон). Методи знаходження порогу бінаризації можна розділити на дві умовні категорії – глобальні та локальні. У глобальних методах обробляється все зображення, використовуючи заздалегідь підібраний поріг бінаризації, за допомогою якого відбувається диференціювання на чорний і білий колір. Методи бінаризації, за яких зображення ділиться на частини або береться якесь оточення пікселя називають локальними.

Для того, щоб визначити найбільш ефективний алгоритм бінаризації, було обрано різні комбінації з пар: режим конвертації зображення в градації сірого та методу бінаризації. За результатами отриманих зображень можна зробити висновок, що найбільш оптимальним виявився алгоритм Бредлі, оскільки кінцевий результат містить найменшу кількість шуму, збережена ключова інформація з вихідного зображення, а обчислювальна складність поступається лише методам глобальної бінаризації.

Список літератури

1. Isayed, S. A review of optical Braille recognition / S. Isayed, R. Tahboub //2015 2nd World Symposium on Web Applications and Networking (WSWAN). – IEEE. 2015. – С. 1–6.

2. Romano He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. В 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (с. 770–778).

ДОДАТОК В

Програмний код розробленої технології

В.1 utils.py

```
from collections import OrderedDict
from PIL import ImageDraw, ImageFont, Image
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
import os
import json
import torch

proj_path = os.getcwd()
model_path = os.path.join(proj_path, "weights", "model_299.pth")
sources_path = os.path.join(proj_path, "data", "sources")
dataset_path = os.path.join(proj_path, "data", "labels")

with open(os.path.join(proj_path, "braille_letters.txt"),
encoding='utf-8') as file:
    braille = file.read()

with open(os.path.join(proj_path, "braille_to_ru.json"),
encoding='utf-8') as file:
    braille_to_ru = json.loads(file.read())

with open(os.path.join(proj_path, "braille_to_eng.json"),
encoding='utf-8') as file:
    braille_to_eng = json.loads(file.read())


def flatten(l):
    result = []

    for i in l:
        result += i

    return result


def targets_to_lines(boxes, labels, scores):
    min_height = min(map(lambda x: x[3] - x[1], boxes)) * 0.8
    predictions = []
    for i in range(len(boxes)):
        predictions.append({"box": boxes[i], "label": labels[i],
"score": scores[i]})
```

```
    lines = {}
    for pred in predictions:
        y = pred["box"][1]
        key = list(filter(lambda x: abs(x - y) < min_height,
lines.keys()))
        key = None if len(key) == 0 else key[0]
        if key is None:
            lines[y] = [pred]
        else:
            lines[key].append(pred)

    for key in lines:
        lines[key].sort(key=lambda x: x["box"][0])

    sorted_keys = sorted(lines.keys())
    sorted_lines = OrderedDict()
    for key in sorted_keys:
        sorted_lines[key] = lines[key]

    return sorted_lines



def filter_res(boxes, labels, scores):
    min_height = min(map(lambda x: x[3] - x[1], boxes)) * 0.8
    predictions = [{"box": box, "label": label, "score": score}
for box, label, score in zip(boxes, labels, scores)]

    lines = {}

    for pred in predictions:
        y = pred["box"][1]
        key = list(filter(lambda x: abs(x - y) < min_height,
lines.keys()))
        key = None if len(key) == 0 else key[0]
        if key is None:
            lines[y] = [pred]
        else:
            lines[key].append(pred)

    for key in lines:
        lines[key].sort(key=lambda x: x["box"][0])

    nms_predictions = {}

    for key, line in lines.items():
        nms_predictions[key] = []

        for i in range(0, len(line)):
            left_x1 = line[i]["box"][0]
            left_y1 = line[i]["box"][1]
            left_x2 = line[i]["box"][2]
            left_y2 = line[i]["box"][3]
```

```
            left_width = left_x2 - left_x1
            left_height = left_y2 - left_y1
            score_left = line[i]["score"]
            duplicate_i = None
            for k in range(len(nms_predictions[key])):
                right_x1 = nms_predictions[key][k]["box"][0]
                right_y1 = nms_predictions[key][k]["box"][1]
                right_x2 = nms_predictions[key][k]["box"][2]
                right_y2 = nms_predictions[key][k]["box"][3]
                right_width = right_x2 - right_x1
                right_height = right_y2 - right_y1
                if ((right_x1 - left_x1) ** 2 + (right_y1 -
left_y1) ** 2) ** 1/2 < min(left_height, left_width, right_height,
right_width) / 2:
                    duplicate_i = k
                    break

            if duplicate_i is not None:
                score_right =
nms_predictions[key][duplicate_i]["score"]
                if score_left > score_right:
                    nms_predictions[key][duplicate_i] = line[i]
            else:
                nms_predictions[key].append(line[i])

    sorted_lines = {key: lines[key] for key in
sorted(lines.keys())}

    values =
flatten(OrderedDict(sorted(nms_predictions.items())).values())

    return list(map(lambda x: x['box'], values)), list(map(lambda
x: x['label'], values)), list(map(lambda x: x['score'], values)),
sorted_lines

def draw(img, boxes, labels, scores):
    head, tail = os.path.split(img)
    img = Image.open(os.path.join(sources_path, tail))
    draw = ImageDraw.Draw(img)

    color = (0, 255, 0)
    width = 2

    text_color = (50, 50, 255)
    braile_font =
ImageFont.truetype("C:\\Users\\Drakosha\\Downloads\\BlistaBraille.
ttf", 15)
    font = ImageFont.truetype("arial.ttf", 15)


    for box, score, label in zip(boxes, scores, labels):
        if score < 0.5:
            continue
```

```
        x1, y1, x2, y2 = box
        width = x2 - x1
        height = y2 - y1
        draw.rectangle((x1, y1, x2, y2), outline=color, width=2)

        label_text = braille_to_ru.get(braille[label],
braille[label])

        my_text_size_x, my_text_size_y = draw.textsize(label_text,
font=font)
        braille_text_size_x, braille_text_size_y =
draw.textsize(braille[label], font=braile_font)
        centred_x = x1 + width / 2 - my_text_size_x
        centred_y = y1 + height / 2 - my_text_size_y / 2
        draw.text((centred_x, centred_y), label_text,
fill=text_color, font=font)
        draw.text((x1 + width / 2, y1 + height / 2 -
braille_text_size_y / 2), braille[label], fill=text_color,
font=braile_font)

    img.show()


def load_data():
    transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])

    labels_path = os.path.join(proj_path, "data", "labels")
    targets = []
    for label_file in os.listdir(labels_path):
        with open(os.path.join(labels_path, label_file)) as file:
            labels_dict = json.load(file)
            boxes = []
            labels = []
            for label in labels_dict["labels"]:
                boxes.append(label["box"])
                labels.append(label["class"])
            targets.append({"image": labels_dict["image"],
"targets": {"boxes": boxes, "labels": labels}})


    trainset_len = round(len(targets) * 0.9)

    trainset = BrailleDataset(targets[:trainset_len], transform)
    testset = BrailleDataset(targets[trainset_len:], transform)

    trainloader = DataLoader(trainset, batch_size=1, shuffle=True,
num_workers=1, collate_fn=collate_fn)
    testloader = DataLoader(testset, batch_size=1, shuffle=False,
num_workers=1, collate_fn=collate_fn)

    return trainloader, testloader
```

```
def collate_fn(batch):
    images = [item[0] for item in batch]
    targets = [item[1] for item in batch]

    return images, targets


class BrailleDataset(Dataset):
    def __init__(self, data, transform=None):
        self.data = data
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        img_path = self.data[idx]['image']
        image = {"image": Image.open(img_path), "path": img_path}
        targets = self.data[idx]['targets']

        if self.transform:
            image['image'] = self.transform(image['image'])
            targets['boxes'] = torch.tensor(targets['boxes'])
            targets['labels'] = torch.tensor(targets['labels'],
dtype=torch.int)

        return image, targets
```

B.2 model.py

```
import torch
from torchvision.models.detection.retinanet import RetinaNet,
RetinaNetClassificationHead, RetinaNetRegressionHead,
LastLevelP6P7
from torchvision.models.resnet import Bottleneck, ResNet
from torchvision.models.detection.backbone_utils import
BackboneWithFPN, LastLevelMaxPool
from torchvision.models.detection.anchor_utils import
AnchorGenerator
from functools import partial

class RetinaNetHead(torch.nn.Module):
    def __init__(self, in_channels, num_anchors, num_classes,
norm_layer = None):
        super().__init__()
        self.classification_head = RetinaNetClassificationHead(
            in_channels, num_anchors, num_classes,
norm_layer=norm_layer
        )
        self.regression_head =
RetinaNetRegressionHead(in_channels, num_anchors,
norm_layer=norm_layer)

    def compute_loss(self, targets, head_outputs, anchors,
matched_idxs):
        return {
            "classification":
self.classification_head.compute_loss(targets, head_outputs,
matched_idxs),
            "bbox_regression":
self.regression_head.compute_loss(targets, head_outputs, anchors,
matched_idxs),
        }

    def forward(self, x):
        return {"cls_logits": self.classification_head(x),
"bbox_regression": self.regression_head(x)}

def anchorgen():
    anchor_sizes = tuple((x, int(x * 2 ** (1.0 / 3)), int(x * 2 **
(2.0 / 3))) for x in [32, 64, 128, 256, 512])
    aspect_ratios = ((0.5, 1.0, 2.0),) * len(anchor_sizes)
    anchor_generator = AnchorGenerator(anchor_sizes,
aspect_ratios)
    return anchor_generator

def resnet_fpn_extractor(
    backbone, trainable_layers: int, returned_layers = None,
extra_blocks = None, norm_layer = None,
) -> BackboneWithFPN:
```

```
    layers_to_train = ["layer4", "layer3", "layer2", "layer1",
"conv1"][:trainable_layers]
    if trainable_layers == 5:
        layers_to_train.append("bn1")
    for name, parameter in backbone.named_parameters():
        if all([not name.startswith(layer) for layer in
layers_to_train]):
            parameter.requires_grad_(False)

    if extra_blocks is None:
        extra_blocks = LastLevelMaxPool()

    if returned_layers is None:
        returned_layers = [1, 2, 3, 4]

    return_layers = {f"layer{k}": str(v) for v, k in
enumerate(returned_layers)}

    in_channels_stage2 = backbone.inplanes // 8
    in_channels_list = [in_channels_stage2 * 2 ** (i - 1) for i in
returned_layers]
    out_channels = 256
    return BackboneWithFPN(
        backbone, return_layers, in_channels_list, out_channels,
extra_blocks=extra_blocks, norm_layer=norm_layer
    )

def create_retinanet(
    *,
    num_classes = 64,
    trainable_backbone_layers = 5,
    **kwargs,
) -> RetinaNet:

    backbone = ResNet(Bottleneck, [3, 4, 6, 3])
    backbone = resnet_fpn_extractor(
        backbone, trainable_backbone_layers, returned_layers=[2,
3, 4], extra_blocks=LastLevelP6P7(2048, 256)
    )
    anchor_generator = anchorgen()
    head = RetinaNetHead(
        backbone.out_channels,
        anchor_generator.num_anchors_per_location()[0],
        num_classes,
        norm_layer=partial(torch.nn.GroupNorm, 32),
    )
    head.regression_head._loss_type = "giou"
    model = RetinaNet(backbone, num_classes,
anchor_generator=anchor_generator, head=head,
detections_per_img=800, topk_candidates=1600, **kwargs)

    return model
```

B.3 train.py

```
import torch
from torchvision.models.detection.retinanet import
retinanet_resnet50_fpn_v2

from object_detection.RetinaNet.utils import load_data
from object_detection.RetinaNet.model import create_retinanet

epochs = 100

def train():
    torch.cuda.empty_cache()

    trainloader, testloader = load_data()

    model = create_retinanet(num_classes=64,
trainable_backbone_layers=5, detections_per_img=800,
topk_candidates=1600).cuda()

    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    model.train()

    for epoch in range(epochs):
        i = 0
        train_loader_len = len(trainloader)
        for i, data in enumerate(trainloader, 0):
            images, targets = data
            optimizer.zero_grad()

            # targets = [x.cuda() for x in targets]
            images = [x.cuda() for x in images]
            for target in targets:
                target['boxes'] = target['boxes'].cuda()
                target['labels'] = target['labels'].cuda()

            loss = model(images, targets)

            loss = sum(loss for loss in loss.values())
            loss.backward()
            optimizer.step()

            if i % 10 == 0:
                print(f"Epoch {epoch + 1}/100, Batch
{i}/{train_loader_len}, Loss: {loss.item()}")
                with open("results.txt", "+w") as file:
                    file.write(f"Epoch {epoch + 1}/100, Batch
{i}/{train_loader_len}, Loss: {loss.item()}")

    torch.save(model, f"model_{epoch}.pth")
```

B.4 validate.py

```
from object_detection.RetinaNet.train import load_data
from object_detection.RetinaNet.load_model import load_model
from object_detection.RetinaNet.utils import filter_res, draw,
targets_to_lines
import torch


def calculate_precision(predictions, targets):
    correct_detections = 0
    correct_classifications = 0
    for key_p, key_t in zip(predictions.keys(), targets.keys()):
        for num_p, box_p in enumerate(predictions[key_p]):
            min_distance = [10000, 0]
            for num_t, box_t in enumerate(targets[key_t]):
                distance = abs(box_p['box'][0] - box_t['box'][0])
                if distance < min_distance[0]:
                    min_distance = [distance, num_t]
            iou_value = calculate_iou(box_p['box'],
targets[key_t][min_distance[1]]['box'])
            if iou_value > 0.5:
                correct_detections += 1
                if box_p['label'] ==
targets[key_t][min_distance[1]]['label']:
                    correct_classifications += 1

    return correct_detections, correct_classifications


def calculate_iou(targets, predictions):
    xA = max(targets[0], predictions[0])
    yA = max(targets[1], predictions[1])
    xB = min(targets[2], predictions[2])
    yB = min(targets[3], predictions[3])

    inter_area = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    boxA_area = (targets[2] - targets[0] + 1) * (targets[3] -
targets[1] + 1)
    boxB_area = (predictions[2] - predictions[0] + 1) *
(predictions[3] - predictions[1] + 1)

    return inter_area / float(boxA_area + boxB_area - inter_area)


def validate():
    model = load_model().eval().cuda()

    total_detections = 0
    total_classifications = 0
    total_correct_detections = 0
```

```
    total_correct_classifications = 0


    trainloader, testloader = load_data()

    with torch.no_grad():
        for images_dict, targets in testloader:
            images = [x['image'].cuda() for x in images_dict]
            for target in targets:
                target['boxes'] = target['boxes'].cuda()
                target['labels'] = target['labels'].cuda()

            print(f"Processing {images_dict[0]['path']}")

            outputs = model(images)[0]
            boxes = outputs['boxes'].tolist()
            labels = outputs['labels'].tolist()
            scores = outputs['scores'].tolist()
            boxes, labels, scores, lines = filter_res(boxes,
labels, scores)
            draw(images_dict[0]["path"], boxes, labels, scores)

            targets_lines =
targets_to_lines(targets[0]['boxes'].tolist(),
targets[0]['labels'].tolist(), [1.0 for x in
range(len(targets[0]["boxes"]))])

            total_detections += len(targets[0]["boxes"])
            correct_predictions, correct_classifications =
calculate_precision(lines, targets_lines)
            total_classifications += correct_predictions
            total_correct_detections += correct_predictions
            total_correct_classifications +=
correct_classifications

    print(f"total detections: {total_detections}, correct:
{total_correct_detections}, accuracy: {total_correct_detections /
total_detections}")
    print(f"total classifications: {total_classifications},
correct: {total_correct_classifications}, accuracy:
{total_correct_classifications / total_classifications}")
```

B.5 infer.py

```
import torchvision.transforms as transforms
import os
from preprocess_image.binarize import binarize_image
from PIL import ImageDraw, ImageFont, Image
import time

from object_detection.RetinaNet.load_model import load_model
from object_detection.RetinaNet.utils import filter_res,
braille_to_ru, braille

model = load_model().cuda()

model.eval()


def draw(img, boxes, labels, scores):
    draw = ImageDraw.Draw(img)

    color = (0, 255, 0)
    width = 2

    text_color = (50, 50, 255)
    braile_font =
ImageFont.truetype("C:\\Users\\Drakosha\\Downloads\\BlistaBraille.
ttf", 15)
    font = ImageFont.truetype("arial.ttf", 15)


    for box, score, label in zip(boxes, scores, labels):
        if score < 0.5:
            continue
        x1, y1, x2, y2 = box
        width = x2 - x1
        height = y2 - y1
        draw.rectangle((x1, y1, x2, y2), outline=color, width=2)

        label_text = braille_to_ru.get(braille[label],
braille[label])

        my_text_size_x, my_text_size_y = draw.textsize(label_text,
font=font)
        braille_text_size_x, braille_text_size_y =
draw.textsize(braille[label], font=braile_font)
        centred_x = x1 + width / 2 - my_text_size_x
        centred_y = y1 + height / 2 - my_text_size_y / 2
        draw.text((centred_x, centred_y), label_text,
fill=text_color, font=font)
        draw.text((x1 + width / 2, y1 + height / 2 -
braille_text_size_y / 2), braille[label], fill=text_color,
font=braile_font)
```

```
    # img.show()

    return img

def get_results(folder_path, result_path=""):
    for file in os.listdir(folder_path):
        print(f"processing {file}")
        img_path = os.path.join(folder_path, file)
        img = binarize_image(img_path)
        transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
        input_image = transform(img).unsqueeze(0).cuda()
        output = model(input_image)[0]
        boxes = output['boxes'].tolist()
        scores = output['scores'].tolist()
        labels = output['labels'].tolist()
        orig_img = Image.open(os.path.join(folder_path, file))
        filtred_boxes, filtred_labels, filtred_scores,
sorted_lines = filter_res(boxes, labels, scores)
        res_image = draw(orig_img, filtred_boxes, filtred_labels,
filtred_scores)
        res_image.save(os.path.join(result_path + file))
```

B.6 bin_module.cpp

```
#include <Python.h>
#include <algorithm>


extern "C" void bradley_binarization(PyObject *pixels, PyObject
*width, PyObject *height, PyObject *bradley_param){
    long w = PyLong_AsLong(width);
    long h = PyLong_AsLong(height);
    long bradley_p = PyLong_AsLong(bradley_param);

    bradley_p = std::clamp(bradley_p, 1L, 32L);

    const int S = w / bradley_p;
    int s2 = S / 2;
    const float t = 0.15;
    unsigned long* integral_image = (unsigned
long*)malloc(sizeof(unsigned long) * w * h);
    long sum = 0;
    int count = 0;
    int index;
    int x1, y1, x2, y2;


    for (int x = 0; x < w; x++){
        sum = 0;
        for (int y = 0; y < h; y++){
            index = y * w + x;
            sum += PyLong_AsLong(PyList_GetItem(pixels, index));
            if (x == 0)
                integral_image[index] = sum;
            else
                integral_image[index] = integral_image[index - 1]
+ sum;
        }
    }

    for (int x = 0; x < w; ++x) {
        for (int y = 0; y < h; ++y) {
            index = y * w + x;

            x1 = x - s2;
            x2 = x + s2;
            y1 = y - s2;
            y2 = y + s2;

            if (x1 < 0)
                x1 = 0;
            if (x2 >= w)
                x2 = w - 1;
            if (y1 < 0)
```

```
                y1 = 0;
            if (y2 >= h)
                y2 = h - 1;

            count = (x2 - x1) * (y2 - y1);

            sum = integral_image[y2 * w + x2] - integral_image[y1
* w + x2] -
                integral_image[y2 * w + x1] + integral_image[y1 *
w + x1];

            long p = PyLong_AsLong(PyList_GetItem(pixels, index));
            if ((p * count) < (long)(sum * (1.0 - t)))
                PyList_SetItem(pixels, index, PyLong_FromLong(0));
            else
                PyList_SetItem(pixels, index,
PyLong_FromLong(255));
        }
    }
}
```

B.7 bin_module.py

```
import ctypes as ct


DLL = ct.CDLL(path_to_dll)

def bradley_binarization(pixels, width, height, bradley_param):
    bradley_bin = ct.PYFUNCTYPE(None, ct.py_object, ct.py_object,
ct.py_object, ct.py_object)(('bradley_binarization', DLL))
    bradley_bin(pixels, width, height, bradley_param)
```