ДОДАТОК А

Слайди презентації

Атестаційна робота магістра

Дослідження методів комп'ютерного зору для вирішення задач навчання ознак для реідентифікації об'єктів

Виконав: ст. гр. ІПЗм-18-3 Танасюк Д.О.
Керівник роботи: к.т.н., доц. Турута О.П.

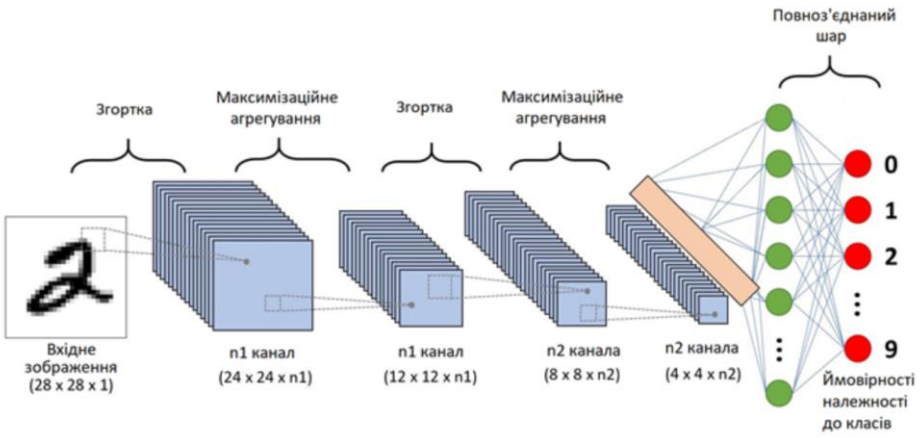
Рисунок А.1 – Слайд 1

Актуальність роботи

- Метою даної роботи є дослідження, аналіз та покращення якості роботи алгоритмів, що використовуються для реідентифікації об’єктів на зображеннях.
- Об’єктом дослідження є архітектури глибоких нейронних згорткових мереж для реідентифікації об’єктів на зображеннях за допомогою методів навчання ознак.
- Предметом дослідження є оптимізація метрик якості на різних підзадачах реідентифікації.

Рисунок А.2 – Слайд 2

Класифікаційна згорткова нейронна мережа



Джерело: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

3

Рисунок А.3 – Слайд 3

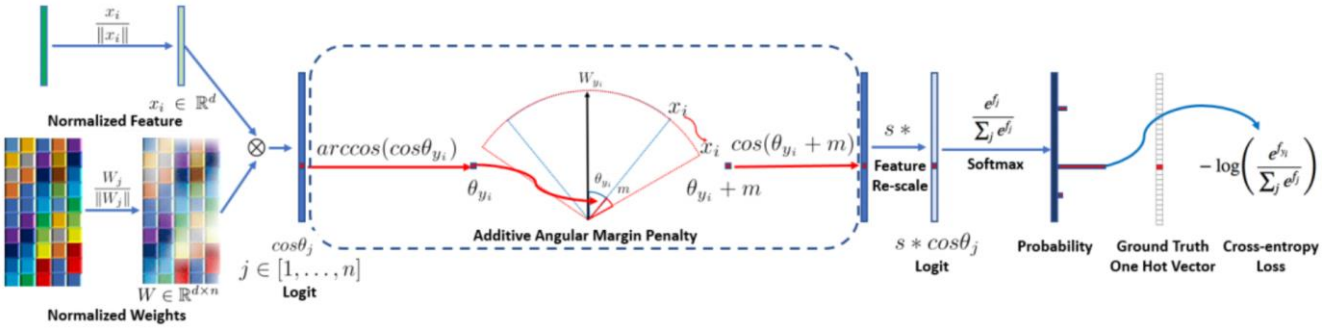
Недоліки переходу до класифікації

- динамічне змінення кількості класів;
- мала вибірка;
- лінійна роздільність.

4

Рисунок А.4 – Слайд 4

Нейронна мережа ArcFace



Джерело: https://arxiv.org/pdf/1801.07698.pdf

5

Рисунок А.5 – Слайд 5

Нейронна мережа ArcFace

$$L = -log\left(\frac{e^{\|x_i\| \cos\left(\theta_{y_i,i} + m\right)}}{e^{\|x_i\| \cos\left(\theta_{y_i,i} + m\right)} + \sum_{j \neq y_i} e^{\|x_i\| \cos\left(\theta_{y_j,i}\right)}}\right)$$

де x – вектор ознак,

y – цільовий клас,

θ – кут між векторами,

m – адитивний коефіцієнт кутового розділення.

6

Рисунок А.6 – Слайд 6

Нейронна мережа ArcFace

(a) Softmax (b) ArcFace

Джерело: https://arxiv.org/pdf/1801.07698.pdf

7

Рисунок А.7 – Слайд 7

Постановка задачі

Три основні задачі реідентифікації:

- реідентифікація з відкритим набором даних;
- реідентифікація з закритим набором даних;
- попарна реідентифікація.

8

Рисунок А.8 – Слайд 8

Реідентифікація з відкритим набором даних

датасет MegaFace
метрика - точність

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

де *acc* – точність,

TP – true positive – правильно ідентифікований клас,

TN – true negative – правильно відхилений клас,

FP – false positive – помилка першого роду – помилкове спрацювання,

FN – false negative – помилка другого роду – пропуск класу.

9

Рисунок А.9 – Слайд 9

Реідентифікація з закритим набором даних

датасет CIFAR-10
метрика - *FI*-micro

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f1 = \frac{2 * presicion * recall}{precision + recall}$$

де *TP, FP, FN* – див. слайд 9,

precision – позитивно прогнозована величина, *recall* – чутливість,

f1 – метрика *F1*-micro.

10

Рисунок А.10 – Слайд 10

Попарна реідентифікація

датасет Trillion-Pairs
метрика - false positive rate

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

де *TN, FP* – див. слайд 9,

N – кількість зразків,

FPR – false positive rate.

11

Рисунок А.11 – Слайд 11

Методи оптимізації

- Триплетний відбір зразків (стратегії: випадкова; всі позитивні, все негативні; найпростіший позитивний, найтяжчий негативний).
- Нормалізація пакетів даних (стратегії: нормалізація пакетів, нормалізація ваг, нормалізація шарів).
- Шар виключення (параметри ймовірності: p=0, p=0.2, p=0.5, p=0.7).
- Аугментація зображень (типи перетворень: горизонтальний поворот, складна аугментація).

12

Рисунок А.12 – Слайд 12

Аналіз отриманих результатів. Нормалізація пакетів даних

Стратегія	MegaFace, точність	CIFAR-10, f1-micro	Trillion-Pairs, FPR
без змін	0.775	0.863	0.848
нормалізація пакетів	**0.787**	**0.885**	**0.860**
нормалізація ваг	0.763	0.784	0.832
нормалізація шарів	0.776	0.873	0.850

13

Рисунок А.13 – Слайд 13

Аналіз отриманих результатів. Шар виключення

Ймовірність	MegaFace, точність	CIFAR-10, f1-micro	Trillion-Pairs, FPR
p=0	0.775	0.863	0.848
p=0.2	0.783	0.877	0.852
p=0.5	**0.795**	**0.894**	**0.870**
p=0.7	0,770	0.843	0.843

14

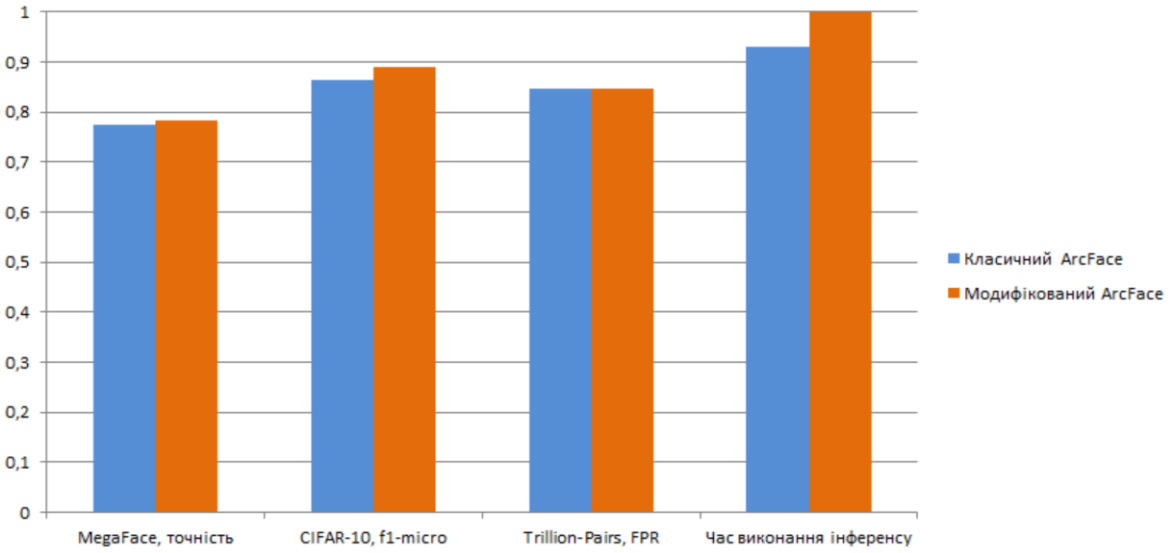
Рисунок А.14 – Слайд 14

Аналіз отриманих результатів. Фінальна архітектура

15

Рисунок А.15 – Слайд 15

Аналіз отриманих результатів. Порівняння метрик



16

Рисунок А.16 – Слайд 16

Висновки

У ході виконання магістерської роботи було проведено:

- аналіз стану досліджень;
- модифікацію метода навчання ознак ArcFace;
- реідентифікацію за допомогою розробленого методу з відкритим та закритим наборами даних, попарну реідетифікацію;
- аналіз результатів дослідження;
- аналіз подальшого використання отриманих результатів в науковій діяльності.

17

Рисунок А.17 – Слайд 17

ДОДАТОК Б

Наукові публікації

MONOGRAFIA
POKONFERENCYJNA

SCIENCE,
RESEARCH, DEVELOPMENT #28

TECHNICS AND TECHNOLOGY.

Baku

29.04.2020 - 30.04.202

U.D.C. 004+62+54+66+082
B.B.C. 94
Z 40

Zbiór artykułów naukowych recenzowanych.
(1) Z 40 Zbiór artykułów naukowych z Konferencji Miedzynarodowej Naukowo-Praktycznej (on-line) zorganizowanej dla pracowników naukowych uczelni, jednostek naukowo-badawczych oraz badawczych z państw obszaru byłego Związku Radzieckiego oraz byłej Jugosławii.

(*30.04.2020*) - Warszawa, 2020. - 72 str.
ISBN: 978-83-66401-48-8

Wydawca: Sp. z o.o. «Diamond trading tour»
Adres wydawcy i redakcji: 00-728 Warszawa, ul. S. Kierbedzia, 4 lok.103
e-mail: info@conferenc.pl

W artykułach naukowych zachowano oryginalną pisownię.

Wszystkie artykuły naukowe są recenzowane przez dwóch członków Komitetu Naukowego.

W przypadku cytowań obowiązkowe jest odniesienie się do monografii.

Publikacja elektroniczna.

ISBN: 978-83-66401-48-8

SPIS/СОДЕРЖАНИЕ

МЕТОДЫ METRIC LEARNING ДЛЯ РЕШЕНИЯ ЗАДАЧИ РЕИДЕНТИФИКАЦИИ ОБЪЕКТОВ НА ИЗОБРАЖЕНИИ

Танасюк Д.О.
Студент,
Харьковский национальный университет радиоэлектроники

Ключевые слова: metric learning, компьютерное зрение, машинное обучение, реидентификация

Keywords: metric learning, computer vision, machine learning, reidentification

Искусственный интеллект приобретает все большую популярность. Благодаря ему человечество решает, как более повседневные задачи, вроде распознания лиц или отпечатка пальца для разблокировки мобильного телефона, так и более глобальные. В настоящее время искусственный интеллект широко используется в различных сферах жизни человека: промышленности (регулировка нагрузки на оборудовании), политике, маркетинга (анализ рекламы, прогнозирования спроса и т.д.), медицине (определение заболеваний), образовании, интернете вещей, машинном переводе, голосовых помощниках и т.д.

Компьютерное зрение – это направление в области искусственного интеллекта, и технологии получения изображений объектов, с ним связаны, а также обработка этих изображений и использования полученных данных, для решения прикладных задач без участия человека. Благодаря компьютерному зрения можно решить такие задачи, как сегментация изображений, классификация изображений, распознавания лиц, генерация изображений и тому подобное. Компьютерное зрение используется для работы беспилотных автомобилей, распознавание изображений с камер слежения для дальнейшего использования (например распознавания номеров машин, лиц), сортировка или поиск брака на производстве, картографические системы, анализ эмоционального состояния человека, считывания штрихкодов, технологии дополненной и виртуальной реальности, конвертация книг и документов из бумажного формата в цифровой.

Распознавание и реидентификация объектов, как одна из самых распространенных задач компьютерного зрения, широко изучалась десятилетиями.

Сверточная нейронная сеть – алгоритм глубокого обучения, который может принимать входное изображение, присваивать важность (учебные веса и смещение) различным объектам на картинке и имеет возможность отличать их друг от друга. Предварительная обработка, необходимая для рабо-

ты с этим типом сетей, значительно ниже по сравнению с другими алгоритмами классификации, тогда как ранние исследования базируются на построении неглубоких модели с низкоуровневыми признаками объекта, созданными вручную. В ходе работы изображение множество раз проходит через слои с сверточными фильтрами по итогу превращаясь в вектор в пространстве признаков, после чего линейно классифицируется.

Реидентификация объектов обычно включает три этапа: выявление объекта, изъятие особенностей и классификация. Первый этап выполняется отдельно как шаг предварительной обработки изображения с помощью моделей детекции, но на практике не всегда возникает необходимость в этапе выявления.

Задачу реидентификации объектов можно свести к задаче классификации с помощью глубокой сверточной нейронной сети, но с динамическим количеством классов, то есть со временем количество классов может меняться, но такой подход имеет несколько основных недостатков:

– изменение количества классов для распознавания влечет за собой изменение архитектуры нейронной сети, а значит ее придется переучивать;

– обычно в задаче реидентификации мы имеем очень маленькую выборку, например, мы хотим распознавать сотрудников нашей фирмы на изображениях с камер и у нас есть всего несколько фотографий каждого из них, тогда как для обучения хорошей сверточной сети необходимы десятки тысяч экземпляров;

– обучение классической сверточной сети решает проблему линейной разделимости изображений в полученном пространстве признаков, но не отдаляет классы между собой, т.е. экземпляры разных классов могут находиться довольно близко в полученном пространстве.

Очевидно, что классическая сверточная нейронная сеть для классификации имеет достаточно весомые недостатки, поэтому для решения задач реидентификации и распознавания применяют другой подход, основанный на сверточных нейронных сетях – методы metric learning.

Методы metric learning в компьютерном зрении заключается в поиске подходящего пространства признаков, в котором сходства между парами изображений сохраняют соответствующую структуру расстояния, то есть дистанция между изображениями одного класса или одного объекта значительно меньше, чем дистанция между изображениями разных классов. Такое пространство признаков также может улучшить эффективность поиска изображений, в частности когда количество категорий или объектов очень большое или неизвестно.

Суть заключается в переходе от задачи классификации к задаче регрессии, то есть вместо оптимизации метрик, связанных с распределением вероятностей целевых классов, оптимизируются определенные метрики

расстояния между векторами признаков изображений.

Это направление является достаточно новым, относительно классических задач компьютерного зрения и стремительно развивается последние годы, потому что появляется все больше крупных датасетов разных доменов. Архитектура таких моделей базируются на рассмотренных сверточных нейронных сетях, но после сверточной части сети следует не линейная классификация, а другие конструкции для решения задачи регрессии.

Одним с первых методов metric learning является сиамская нейронная сеть. Суть работы заключается в том, что экземпляры поступают в сеть парами, с каждого экземпляра выделяют вектор признаков с помощью глубокой нейронной сети (обычно сверточной), а затем вычисляется эвклидово расстояние между ними. Полученное расстояние можно сравнивать с некоторым порогом, подобранным на валидационной выборке, для определения принадлежат оба изображения к одному классу или нет.

Таким образом, сиамская нейронная сеть с помощью перехода от классификации к регрессии позволяет реидентифицировать объекты на изображениях без перечисленных выше недостатков классических сверточных сетей: архитектура не зависит от количества классов; выборка может содержать не только наши экземпляры, но и другие изображения из похожего домена для лучшего обучения сети; вместо линейной разделимости сеть будет максимизировать эвклидово расстояние между классам.

Список литературы:

1. Hoffer E., Ailon N. Deep metric learning using triplet network // [Материалы конференции] International Workshop on Similarity-Based Pattern Recognition, p.84-92, 2015.
2. Utkin L., Kovalev M. and Kasimov E. An explanation method for Siamese neural networks // arXiv.org e-Print article

ДОДАТОК В

Лістинг коду

```
import json
import pandas as pd
import numpy as np
import random
import os
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchvision import transforms
from google.colab import drive
from tqdm import tqdm
import datetime
from datetime import date
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,
     classification_report
from pprint import pprint
import seaborn as sns
import matplotlib.pyplot as plt
import types
import math
from sklearn.neighbors import KNeighborsClassifier

%matplotlib inline

def seed_everything(seed=357, gpu_deterministic=False):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = gpu_deterministic

seed_everything()

def smooth_loss(loss, rate=2):
  result = []
  l = list(zip(*loss))
  for i in range(len(loss) // rate - 1):
    result += [(l[0][i * rate + rate], sum(l[1][i * rate:i * rate + rate])
      / rate)]
  return result

is_tqdm = True

device = torch.device("cuda")
loader_workers = 64
batch_size = 512
```

```
pretrained_weights = 'resnet50'

previous = None
is_load_optimizer = False

transform = transforms.Compose([
          transforms.ToTensor(),
          transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
     0.224, 0.225]),
])

ds_train = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
ds_val = torchvision.datasets.CIFAR10(root='./data', train=False,
                                      download=True, transform=transform)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
prod_to_target = { name : id for id, name in enumerate(classes) }
target_to_prod = { id : name for id, name in enumerate(classes) }
num_classes = len(classes)

class ArcMarginHead(nn.Module):
  def __init__(self, num_classes, emb_size=512, easy_margin=False,
      margin_m=0.5, margin_s=30.0):
    super(ArcMarginHead, self).__init__()

    self.weight = torch.nn.Parameter(torch.FloatTensor(num_classes,
     emb_size))
    nn.init.xavier_uniform_(self.weight)

    self.easy_margin = easy_margin
    self.m = margin_m
    self.s = margin_s

    self.cos_m = math.cos(self.m)
    self.sin_m = math.sin(self.m)
    self.th = math.cos(math.pi - self.m)
    self.mm = math.sin(math.pi - self.m) * self.m

  def forward(self, input, label):
    x = F.normalize(input)
    W = F.normalize(self.weight)
    cosine = F.linear(x, W)
    sine = torch.sqrt(1.0 - torch.pow(cosine, 2))
    phi = cosine * self.cos_m - sine * self.sin_m  # cos(theta + m)
    if self.easy_margin:
        phi = torch.where(cosine > 0, phi, cosine)
    else:
        phi = torch.where(cosine > self.th, phi, cosine - self.mm)
    one_hot = torch.zeros(cosine.size(), device=device)
    one_hot.scatter_(1, label.view(-1, 1).long(), 1)
    output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
    output *= self.s

    return output

class ArcfaceModel(nn.Module):
```

```
  def __init__(self, encoder, custom, head):
    super(ArcfaceModel, self).__init__()

    self.encoder = encoder
    self.custom = custom
    self.head = head

  def forward(self, x, label):
    x = self.encoder(x)
    x = self.custom(x)

    if label is not None:
      x = self.head(x, label)

    return x

class VanilaBlock(nn.Module):
  def __init__(self, x1=512, x2=512):
    super(VanilaBlock, self).__init__()

    self.fc = nn.Linear(in_features=x1, out_features=x2, bias=True)

  def forward(self, x):
    x = self.fc(x)

    return x

class BNBlock(nn.Module):
  def __init__(self, x1=512, x2=512):
    super(BNBlock, self).__init__()

    self.bn1 = nn.BatchNorm1d(512)
    self.fc = nn.Linear(in_features=x1, out_features=x2, bias=True)
    self.bn2 = nn.BatchNorm1d(512)

  def forward(self, x):
    x = self.bn1(x)
    x = self.fc(x)
    x = self.bn2(x)

    return x

class LNBlock(nn.Module):
  def __init__(self, x1=512, x2=512):
    super(LNBlock, self).__init__()

    self.bn1 = nn.LayerNorm(512)
    self.fc = nn.Linear(in_features=x1, out_features=x2, bias=True)
    self.bn2 = nn.LayerNorm(512)

  def forward(self, x):
    x = self.bn1(x)
    x = self.fc(x)
    x = self.bn2(x)

    return x

class DOBlock(nn.Module):
```

```
  def __init__(self, x1=512, x2=512, p=0.5):
    super(DOBlock, self).__init__()

    self.do = nn.Dropout(p=p)
    self.fc = nn.Linear(in_features=x1, out_features=x2, bias=True)

  def forward(self, x):
    x = self.do(x)
    x = self.fc(x)

    return x

class FinalBottleneck(nn.Module):
  def __init__(self, x1=512, x2=512, p=0.5):
    super(FinalBottleneck, self).__init__()

    self.bn1 = nn.LayerNorm(512)
    self.do = nn.Dropout(p=p)
    self.fc = nn.Linear(in_features=x1, out_features=x2, bias=True)
    self.bn2 = nn.LayerNorm(512)

  def forward(self, x):
    x = self.bn1(x)
    x = self.do(x)
    x = self.fc(x)
    x = self.bn2(x)

    return x

def forward(self, x):
  x = self.conv1(x)
  x = self.bn1(x)
  x = self.relu(x)
  x = self.maxpool(x)
  x = self.layer1(x)
  x = self.layer2(x)
  x = self.layer3(x)
  x = self.layer4(x)
  x = self.avgpool(x)
  x = torch.flatten(x, 1)
  return x

encoder = torch.hub.load('pytorch/vision', pretrained_weights,
     pretrained=True)
encoder.forward = types.MethodType(forward, encoder)

head = ArcMarginHead(num_classes)

block = FinalBottleneck()

model = ArcfaceModel(encoder, block, head)
model = model.to(device)

def validate(dataloader, criterion):
  model.eval()
  tlen = len(dataloader)

  tloss = 0.0
```

```
  preds = []
  labels = []

  with torch.no_grad():
    it = tqdm(dataloader, total=tlen) if is_tqdm else dataloader
    for x, y in it:
      x = x.to(device)
      y = y.to(device)
      output = model(x, y)
      loss = criterion(output, y)
      tloss += loss.item()
      _, pred = torch.max(output, 1)

      preds += pred.cpu()
      labels += y.cpu()

  tloss = tloss / tlen
  print(f'Loss: {tloss}')
  print(classification_report(labels, preds, target_names=classes))

  return tloss, confusion_matrix(labels, preds)

def train_epoch(dl_train, batch_num, criterion, scheduler, acc_step):
  model.train()

  optimizer.zero_grad()

  tloss = []
  it = tqdm(dl_train, total=batch_num) if is_tqdm else dl_train
  for idx, (x, y) in enumerate(it):
    x = x.to(device)
    y = y.to(device)

    output = model(x, y)
    loss = criterion(output, y)

    tloss.append(loss.item())

    loss.backward()

    if (idx + 1) % acc_step == 0 or (idx + 1) == batch_num:
      optimizer.step()
      optimizer.zero_grad()

  return np.array(tloss)

def train(dl_train, dl_val, scheduler, criterion, epochs, finished_epoches,
      val_interval=1, acc_step=1):
  batch_num = len(dl_train)

  train_loss_log = []
  val_loss_log = []

  ft = '{:.6f}'.format
  for epoch in range(finished_epoches, finished_epoches + epochs):
    if scheduler:
      scheduler.step()
      print(f'LR: {scheduler.get_lr()}')
```

```
    tloss = train_epoch(dl_train, batch_num, criterion, scheduler,
     acc_step)
    print(f'{datetime.datetime.now()} Epoch {epoch+1}/{finished_epoches +
     epochs}: train loss={ft(np.mean(tloss))}')
    train_loss_log += list(zip(range(epoch * batch_num, epoch * batch_num +
     batch_num), tloss))
    torch.cuda.empty_cache()

    if epoch % val_interval == 0 or (epoch == finished_epoches + epochs -
     1):
      val_loss, _ = validate(dl_val, criterion)
      val_loss_log += [(epoch * batch_num + batch_num - 1, val_loss)]
      torch.cuda.empty_cache()

  torch.save({
    'epoch': finished_epoches + epochs,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    # 'scheduler_state_dict': scheduler.state_dict(),
  }, f'{model_filename()}_{finished_epoches + epochs}.pth')
  print('model saved')
  del scheduler

  return train_loss_log, val_loss_log

plt.figure(figsize=(20,10))
plt.plot(*zip(*smooth_loss(train_losses, rate=10)), label = "train")
plt.plot(*zip(*val_losses), label = "validation")
plt.xlabel('batch')
plt.ylabel('loss')
plt.title('Loss')
plt.legend()
plt.show()

@torch.no_grad()
def inference(x):
    return model(x.to(device), None).cpu().numpy()

def getXY(dl):
    features = np.array([]).reshape(0, 512)
    targets = np.array([])

    tlen = len(dl)
    for x, y in tqdm(dl, total = tlen):
        features = np.concatenate((features, inference(x)), axis=0)
        targets = np.concatenate((targets, y.numpy()), axis=0)

    return (features, targets)

model.eval();
features, targets = getXY(dl_train)
val_f, val_t = getXY(dl_val)

for k in range (1, 11):
    neigh = KNeighborsClassifier(n_neighbors=k, metric='cosine')
    neigh.fit(features, targets)
    print(k, (neigh.predict(val_f) == val_t).sum(), 'from', len(val_t))
```

ДОДАТОК Г

Відгук керівника

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
Факультет комп’ютерних наук

ВІДГУК

на атестаційну роботу магістра
Танасюк Дмитро Олегович, ІПЗм-18-3,
спеціальність 121 – Інженерія програмного забезпечення
освітньо-наукова програма «Інженерія програмного забезпечення»

Тема атестаційної роботи Дослідження методів комп'ютерного зору для вирішення задач навчання ознак для реідентифікації об'єктів

Студент Танасюк Д.О. виконував атестаційну роботу магістра за темою дослідження методів методів комп'ютерного зору для вирішення задач навчання ознак для реідентифікації об'єктів. Темою аналізу зображень займається протягом останніх 2 років, приймає участь у конференціях присвячених комп’ютерному зору та машинному навчанню.

Під час виконання роботи Танасюк Д.О. самостійно провів аналіз предметної галузі, виділив задачу реідентифікації об’єктів в великому наборі даних та запропонував підхід для зменшення кількості ознак для классифікації.

До виконання роботи ставився сумлінно, регулярно інформував наукового керівника про результати дослідження та незалежно визначав наступні кроки роботи. В цілому, здатен проводити дослідження самостійно займатися науковою діяльністю, займатися розробкою програмного забезпечення та самостійно підвищувати кваліфікацію.

Магістрант гр. ІПЗм-18-3 Танасюк Д.О. готов до самостійної інженерної діяльності. Атестаційну роботу можна подати до захисту в ЕК за спеціальністю 121 – «Інженерія програмного забезпечення», освітньо-науковою програмою «Інженерія програмного забезпечення».

«_____» _____ 2020 р.

Керівник атестаційної роботи магістра
доц. Турута О.П.