**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра автоматики и процессов управления**

**Контрольная работа**

**по дисциплине «Нейросетевое управление»**

**Тема: Решение задач компьютерного зрения в системах управления с использованием предобученных моделей сверточных нейронных сетей.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7391 |  | Посвященный Д.Е. |
| Преподаватель |  | Никонов А.Н. |

Санкт-Петербург

2022

Цель работы - решение задач компьютерного зрения в системах управления с использованием предобученных моделей сверточных нейронных сетей из библиотеки PyTorch.

Преимущество предобученных моделей - получение быстрого результата без погружения в детали реализации и выполнения трудоемкого процесса настройки на крупных библиотеках изображений.

Их недостатки - возможны ограничения лицензий использования как на использованный при обучении набор данных, так и на сами коэффициенты из библиотеки и исходный код модели. Ещё одним недостатком является то, что такие весовые коэффициенты рассчитываются на публичных библиотеках изображений и зачастую могут оказаться нерелеванты решаемой задаче.

Последнюю проблему обычно решают с помощью набора приемов, известных в англоязычной литературе как "Transfer learning". Суть его состоит в использовании библиотечных коэффициентов и модели как базы для настройки новой модели.

Ниже будет проиллюстрировано решение задач классификации и донастройки модели с использованием библиотеки PyTorch.

# **Инициализация преднастроенных моделей и функций.**

Для начала, скачаем предобученные модели: AlexNet, ResNet50, DenceNet, MobileNetV3 и SwinT на GPU (графический процессор) и CPU (центральный процессор), Рисунок 1.

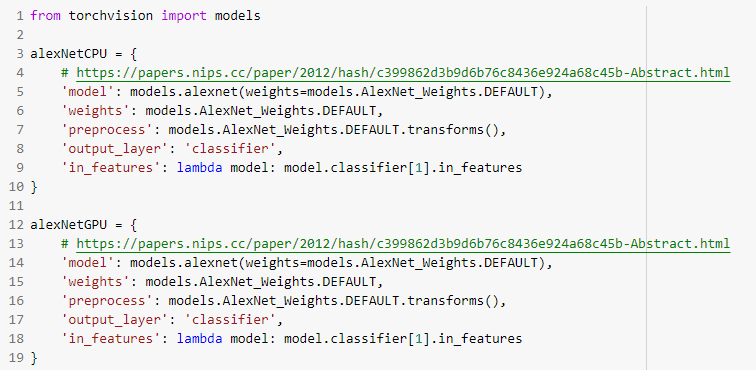


Рисунок 1 –загрузка моделей.

Далее определим вычислительную подсистему (CPU или GPU), Рисунок 2.

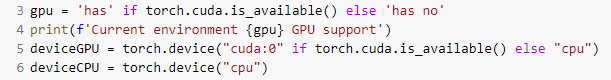


Рисунок 2 – определение вычислительной подсистемы.

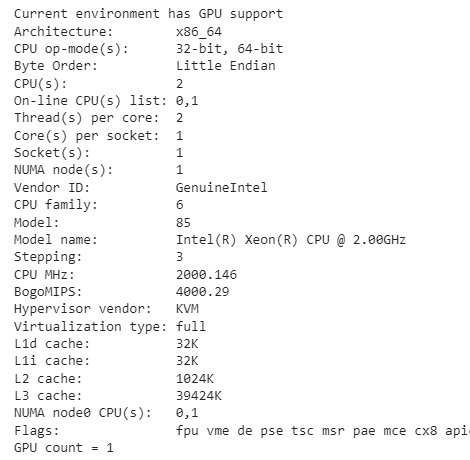


Рисунок 3 – текущие характеристики системы.

Опишем вспомогательные функции для работы с предобученными моделями.

Функция classify, для которой **входные** переменные – параметры модели, а **выходные** переменные – время распознавания картинки, флаг показателя правильно распознанной картинки и процент точности распознавания, Рисунок 4.

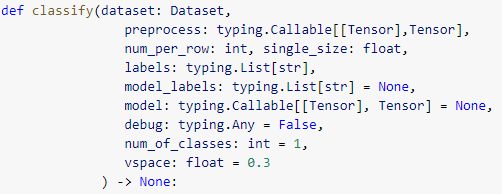


Рисунок 4 – функция classify.

Для создания датасета собрал 50 изображений микроскопа и 50 изображений лобзика.

Ссылки записал в файл simple.csv, Рисунок 5.

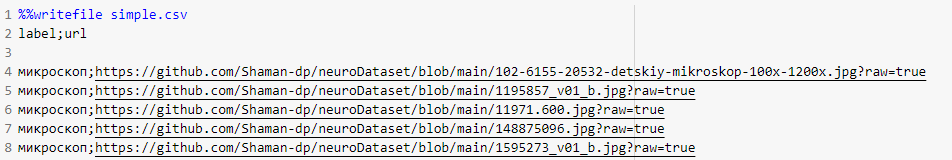


Рисунок 5 – ссылки на фотографии.

# **Расчет моделей**

Необходимо классифицировать изображения для каждой модели и сравнить быстродействие выполнения на GPU и CPU.

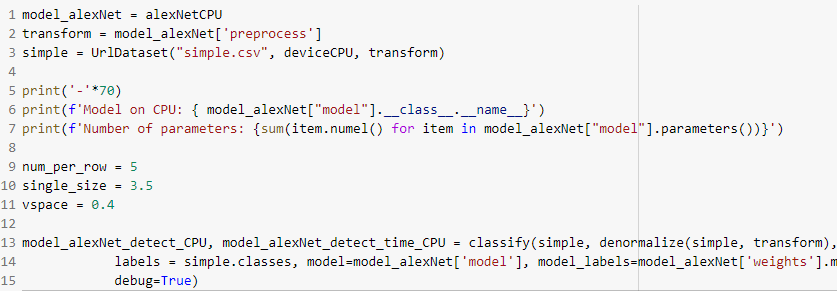


Рисунок 6 – выполнение классификации модели alexNet на CPU.

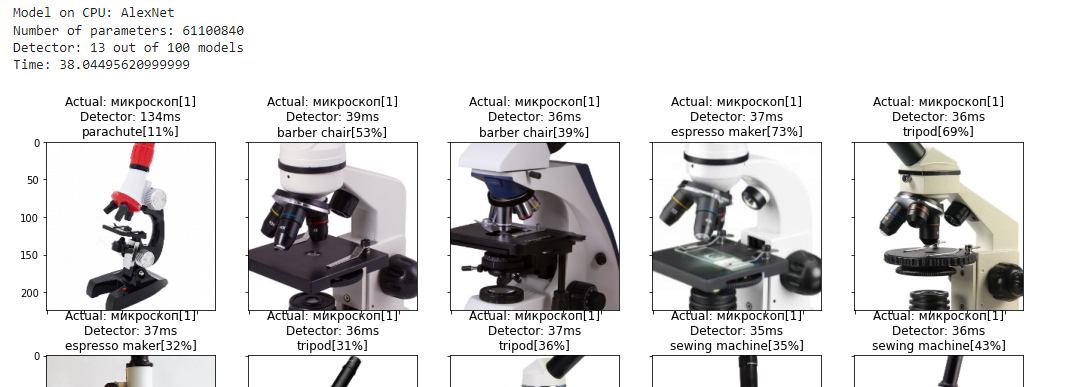


Рисунок 7– результат классификации.

Результат классификации (на рисунке 7) – ни одно изображение микроскопа не было распознано, но зато эспрессо-машина был распознан в пяти случаях.

Проведем классификацию всех моделей и сравним количество правильно распознанных объектов из датасета.

Получим график с распределением распознанных объектов и моделей, на Рисунке 8.

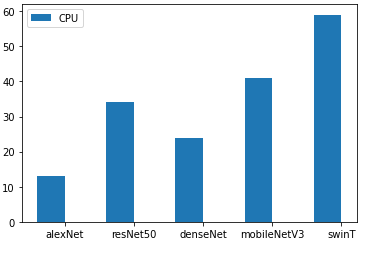


Рисунок 8 – проценты правильно распознанных объектов.

Глядя на график, можно сделать вывод, что предобученная модель SwinT показала наибольший процент правильно классифицированных объектов.

Сравнение скоростей классификации моделей на CPU и GPU показаны на Рисунке 9.

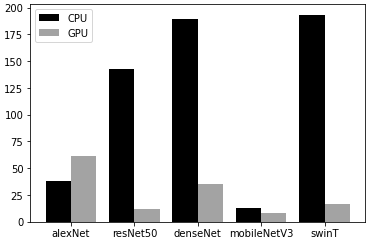


Рисунок 9 – сравнение скоростей классификации моделей.

Глядя на график, можно сделать вывод, что модели AlexNet и MobileNetV3\_small классифицируют объекты, независимо от выбранного процессора, практически с одинаковой скоростью. Модели ResNet50, DenceNet и SwinT, могут выполнять классификацию в параллельном режиме, следовательно, для них лучше использовать графический процессор.

# **Донастройка модели.**

Для донастройки модели собрал два датасета: train.csv с 80 изображениями микроскопа и лобзика, и test.csv с 20 изображениями тех же объектов.

Наиболее простой способ адаптации предобученных моделей к специфической задаче это переобучение выходного классификатора, который вычисляет принадлежность изображения к тому или иному классу. Это возможно благодаря особенности глубоких нейронных сетей, состоящей в том, что по мере преобразования между слоями происходит постепенный переход от конкретного изображения как набора пикселей к более обощённому представлению (пространство признаков), инвариантных к второстепенной информации (фон, повороты, растяжения, освещенность и т.д.). Таким образом, выходной слой лишь использует обобщённое пространство признаков для разделения на области, относящщиеся к разным классам объектов.

Пререобучим выходной классификатор, для распознавания нужных классов объектов.

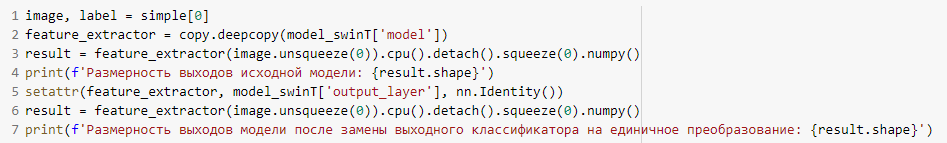


Рисунок 10 – переобучение выходного классификатора.

Используя код на Рисунке 10, получим:

Размерность выходов исходной модели: (1000)

Размерность выходов модели после замены выходного классификатора на единичное преобразование: (768).

Попробуем визуализировать пространство признаков, обозначая объекты целевых классов разными цветами. Общая гипотеза состоит в том, что изображения разных классов должны лежать в разных областях пространства признаков (т.е. они выглядят как сосредоточенные "облака" точек, которые в свою очередь расположены на значительном удалении друг от друга). В этом случае с помощью линейной гиперплоскости можно спроецировать точки в N-мерном пространстве признаков на эту плоскость и получить одно-, двух- или трехмерные проекции этих облаков, которые уже можно визуализировать на графике. Таким образом, для проверки гипотезы достаточно убедиться, что проекции "облаков" разных классов будут визуально изолированы. В этом случае перенастройка линейного классификатора на выходе нейронной сети приведет к решению задачи классификации на индивидуальной выборке объектов.

Чтобы получить двухмерную проекцию точек пространства признаков воспользуемся методом редукции размерности на основе метода линейного PCA (Principial Component Analysis). Суть его состоит в отыскании такой линейной гиперплоскости в N-мерном пространстве, расстояние до которой всех точек будет минимальным (т.е. она максимально аппроксимирует данные). Координаты спроецированных точек в пространстве этой гиперповерхности как раз и являются искомым двухмерным представлением "облаков", характеризующих разные классы объектов выборки.

Итак, первым делом преобразуем имеющиеся списки изображений в массивы признаков и лэйблов классов.



Рисунок 11 – преобразование списков изображений объектов.

Далее применим PCA-метод, который на основе N-мерных массивов признаков вычислит координаты точек на гиперплоскости (U), список собственных чисел (S) и собственных векторов (V) для преобразования координат между основным и редуцированным пространством. Для визуализации остается совершить трансформации осей массивов, чтобы получить ряды для координат абсцисс и ординат графика, а также разделить массивы точек разных классов, чтобы каждому задать свой цвет.

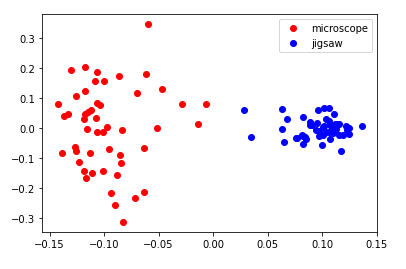


Рисунок 12 – визуализация пространства признаков объектов.

На Рисунке 12 видно, что даже на двухмерной проекции точки разных классов лежат в разных областях и могут быть легко разделены прямой. Это означает, что применение линейного классификатора в данном случае выглядит оправданным.

Заменим линейный классификатор с 768 параметрами и обучим его на распознавание микроскопа и лобзика. Получим, что наша модель определила все объекты правильно. На рисунке 13 показаны графики ошибок.

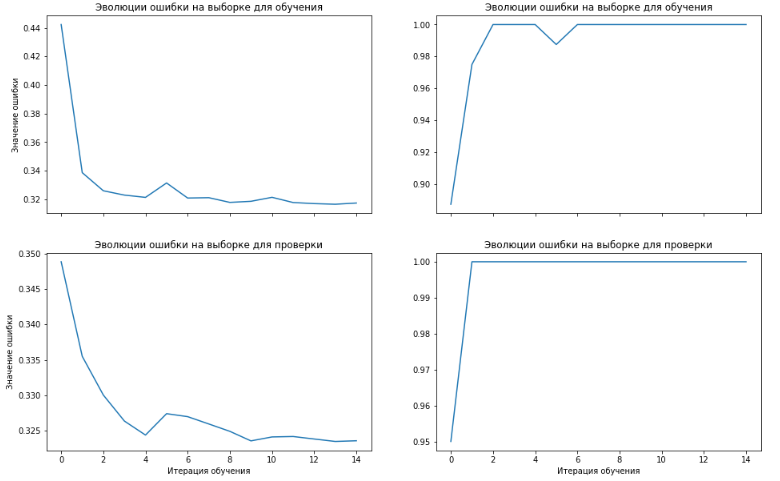


Рисунок 13 – эволюции ошибки.

Получим график с распределением распознанных объектов и моделей, на Рисунке 14.

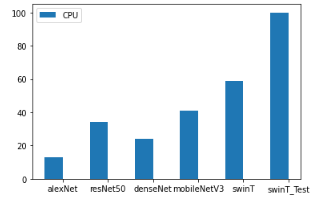


Рисунок 14 – сравнение процентов правильно распознанных объектов на разных моделях.

# **Вывод**

В данной работе были рассмотрены предобученные модели, произведен анализ скорости классификации моделей на различных процессорах, процент правильно распознанных объектов.

А также обучение последнего слоя нейронной сети – классификатора изображений, и его дальнейшая проверка на тестовом датасете.

# **Приложение**

# -\*- coding: utf-8 -\*-

"""kursach.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/13mas2kmAOCqnOIMxW5CZ8rNBU-EMv\_W7

# models

"""

from torchvision import models

alexNetCPU = {

# https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

'model': models.alexnet(weights=models.AlexNet\_Weights.DEFAULT),

'weights': models.AlexNet\_Weights.DEFAULT,

'preprocess': models.AlexNet\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[1].in\_features

}

alexNetGPU = {

# https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

'model': models.alexnet(weights=models.AlexNet\_Weights.DEFAULT),

'weights': models.AlexNet\_Weights.DEFAULT,

'preprocess': models.AlexNet\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[1].in\_features

}

resNet50CPU = {

# https://arxiv.org/abs/1512.03385

'model': models.resnet50(weights=models.ResNet50\_Weights.DEFAULT),

'weights': models.ResNet50\_Weights.DEFAULT,

'preprocess': models.ResNet50\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

resNet50GPU = {

# https://arxiv.org/abs/1512.03385

'model': models.resnet50(weights=models.ResNet50\_Weights.DEFAULT),

'weights': models.ResNet50\_Weights.DEFAULT,

'preprocess': models.ResNet50\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

denseNetCPU = {

# https://arxiv.org/abs/1608.06993

'model': models.densenet201(weights=models.DenseNet201\_Weights.DEFAULT),

'weights': models.DenseNet201\_Weights.DEFAULT,

'preprocess': models.DenseNet201\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

denseNetGPU = {

# https://arxiv.org/abs/1608.06993

'model': models.densenet201(weights=models.DenseNet201\_Weights.DEFAULT),

'weights': models.DenseNet201\_Weights.DEFAULT,

'preprocess': models.DenseNet201\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

mobileNetV3\_smallCPU = {

# https://arxiv.org/abs/1905.02244

'model': models.mobilenet\_v3\_small(weights=models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT),

'weights': models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT,

'preprocess': models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

mobileNetV3\_smallGPU = {

# https://arxiv.org/abs/1905.02244

'model': models.mobilenet\_v3\_small(weights=models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT),

'weights': models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT,

'preprocess': models.mobilenetv3.MobileNet\_V3\_Small\_Weights.DEFAULT.transforms(),

'output\_layer': 'classifier',

'in\_features': lambda model: model.classifier[0].in\_features

}

swinTCPU = {

# https://arxiv.org/abs/2103.14030

'model': models.swin\_t(weights=models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT),

'weights': models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT,

'preprocess': models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT.transforms(),

'output\_layer': 'head',

'in\_features': lambda model: model.head.in\_features

}

swinTGPU = {

# https://arxiv.org/abs/2103.14030

'model': models.swin\_t(weights=models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT),

'weights': models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT,

'preprocess': models.Swin\_T\_Weights.IMAGENET1K\_V1.DEFAULT.transforms(),

'output\_layer': 'head',

'in\_features': lambda model: model.head.in\_features

}

"""# preprocess"""

import torch

gpu = 'has' if torch.cuda.is\_available() else 'has no'

print(f'Current environment {gpu} GPU support')

deviceGPU = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

deviceCPU = torch.device("cpu")

!lscpu

print(f'GPU count = {torch.cuda.device\_count() if torch.cuda.is\_available() else "0"}')

for modelCPU in [resNet50CPU, alexNetCPU, denseNetCPU, mobileNetV3\_smallCPU, swinTCPU]:

modelCPU['model'].to(deviceCPU)

modelCPU['preprocess'].to(deviceCPU)

modelCPU['model'].eval()

for modelGPU in [resNet50GPU, alexNetGPU, denseNetGPU, mobileNetV3\_smallGPU, swinTGPU]:

modelGPU['model'].to(deviceGPU)

modelGPU['preprocess'].to(deviceGPU)

modelGPU['model'].eval()

"""# main"""

import os

import time

import copy

import math

import typing

from typing import Callable

from functools import partial

import numpy

import pandas

from pandas.core.arrays import boolean

import matplotlib.pyplot as plt

from requests import get

from torch import Tensor, nn

from torch.utils.data import DataLoader, Dataset

from torchvision.transforms import ToPILImage, Compose, Resize, CenterCrop

from torchvision.io import read\_image

strip\_chars = ' \t'

tmp\_file\_name = 'tmp\_file\_name\_for\_image\_download'

to\_image = ToPILImage()

def classify(dataset: Dataset,

preprocess: typing.Callable[[Tensor],Tensor],

num\_per\_row: int, single\_size: float,

labels: typing.List[str],

model\_labels: typing.List[str] = None,

model: typing.Callable[[Tensor], Tensor] = None,

debug: typing.Any = False,

num\_of\_classes: int = 1,

vspace: float = 0.3

) -> None:

detect\_model = 0

time\_detect = 0

num = len(dataset)

fig, axs = plt.subplots(math.ceil(num/num\_per\_row), num\_per\_row, figsize=(

single\_size\*num\_per\_row, (single\_size + vspace)\*(math.ceil(num/num\_per\_row))),

sharex=True, sharey=True)

for i in range(0, len(dataset)):

try:

image, label = dataset[i]

pred = None

if model is not None and model\_labels is not None:

start\_time = time.perf\_counter\_ns()

score = model(image.unsqueeze(0)).detach().squeeze(0).softmax(0)

pred\_index = numpy.flip(score.detach().cpu().argsort().numpy())[0]

end\_time = time.perf\_counter\_ns()

pred = f'Detector: {(end\_time - start\_time) / 1\_000\_000:.0f}ms\n{model\_labels[pred\_index]}[{score[pred\_index].item()\*100:.0f}%]'

#

time\_detect = ((end\_time - start\_time) / 1000000) + time\_detect

if model\_labels[pred\_index] == 'espresso maker' or model\_labels[pred\_index] == 'power drill':

detect\_model = detect\_model + 1

#

loc\_fig = axs[i//num\_per\_row, i % num\_per\_row]

loc\_fig.imshow(to\_image(preprocess(image)))

title =f'\nActual: {labels[label]}[{label}]\n{pred}'

loc\_fig.title.set\_text(title)

except Exception as ex:

if debug:

raise ex

print(f'Image {i} is failed to load: {str(ex)}')

#

print('Detector: ' + str(detect\_model) + ' out of 100 models')

print('Time: ' + str(time\_detect/100))

return detect\_model, time\_detect/100,

#

fig.subplots\_adjust(wspace=0.3)

plt.show()

def denormalize(dataset: Dataset, trans: typing.Any) -> Callable[[Tensor], Tensor]:

image, label = dataset[0]

std = torch.as\_tensor(trans.std, dtype=image.dtype, device=image.device).view(-1, 1, 1)

mean = torch.as\_tensor(trans.mean, dtype=image.dtype, device=image.device).view(-1, 1, 1)

return lambda img: img\*std + mean

class UrlDataset(Dataset):

def \_\_init\_\_(self, file: str, to\_device, transform = None) -> None:

self.file = file

self.transform = transform

self.dataset = pandas.read\_csv(file, sep=';')

self.classes = self.dataset['label'].unique()

self.classes.sort()

self.class\_to\_index = {self.classes[i] : i for i in range(len(self.classes))}

self.device = to\_device

def \_\_len\_\_(self) -> int:

return len(self.dataset)

def \_\_getitem\_\_(self, index: int) -> typing.Tuple[torch.Tensor, str]:

url = self.dataset.iloc[index]['url'].strip(strip\_chars)

with open(tmp\_file\_name, 'wb') as file:

file.write(get(url).content)

image = read\_image(tmp\_file\_name).to(self.device)

label = self.class\_to\_index[self.dataset.iloc[index]['label']]

if self.transform:

return self.transform(image), label

return image, label

def train\_model(model, dataloader: DataLoader, device: torch.device,

critery, optimizer, scheduler, num\_epochs=25):

since = time.time()

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_loss = 0.0

best\_epoch = -1

process = {'train': {'loss': [], 'accuracy': []}, 'validate': {'loss': [], 'accuracy': []}}

for epoch in range(num\_epochs):

print(f'Epoch {epoch}/{num\_epochs - 1}')

print('-' \* 10)

epoch\_loss = 0.0

epoch\_acc = 0.0

for item in dataloader:

if item['train'] == True:

model.train()

else:

model.eval()

running\_loss = 0.0

running\_corrects = 0

dataset\_size = 0

for inputs, labels in item['loader']:

dataset\_size = dataset\_size + 1

inputs = inputs.to(device)

labels = labels.to(device)

optimizer.zero\_grad()

with torch.set\_grad\_enabled(item['train'] == True):

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

loss = critery(outputs, labels)

if item['train'] == True:

loss.backward()

optimizer.step()

running\_loss += loss.item() \* inputs.size(0)

running\_corrects += torch.sum(preds == labels.data)

if item['train'] == True:

scheduler.step()

epoch\_loss = running\_loss / dataset\_size

epoch\_acc = running\_corrects.detach().cpu().double() / dataset\_size

if item['train'] == True:

ptype = 'train'

else:

ptype = 'validate'

process[ptype]['loss'].append(epoch\_loss)

process[ptype]['accuracy'].append(epoch\_acc)

print(f'[{epoch}][train={item["train"]}] Loss: {epoch\_loss:.4f} Acc: {epoch\_acc:.4f}')

# deep copy the model

if item['train'] == True and 1/epoch\_loss > best\_loss:

best\_loss = 1/epoch\_loss

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_epoch = epoch

print()

time\_elapsed = time.time() - since

print(f'Training complete in {time\_elapsed // 60:.0f}m {time\_elapsed % 60:.0f}s')

print(f'Best val loss: {1/best\_loss:4f} at epoch {best\_epoch}')

# load best model weights

model.load\_state\_dict(best\_model\_wts)

return model, process

"""# simple file"""

# Commented out IPython magic to ensure Python compatibility.

# %%writefile simple.csv

# label;url

#

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/102-6155-20532-detskiy-mikroskop-100x-1200x.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/1195857\_v01\_b.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/11971.600.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/148875096.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/1595273\_v01\_b.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/1\_big.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/23337.600.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/28046.970.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/3.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/3870.970.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/41c4a7f010.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/45610044.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/4bc158b1f08b03162be756c495e7900c.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/4d2d3456c037c165f82b823320dc3338.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/6049003855.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/68ba85225ca2946d610cce170e3655fb.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/69051\_levenhuk-rainbow-50l-plus-moonstone\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/73985\_levenhuk-trinocular-microscope-med-10t\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/74004\_levenhuk-med-40b-binocular-microscope\_01.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/74008\_levenhuk-med-45b-binocular-microscope\_02.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/74319\_bresser-mikroskop-junior-biolux-sel-40-1600x-zelenyj\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/74319\_bresser-mikroskop-junior-biolux-sel-40-1600x-zelenyj\_09.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/74322\_bresser-mikroskop-junior-biolux-sel-40-1600x-sinij\_09.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/75419\_levenhuk-microscope-400m\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/75425\_levenhuk-microscope-500b\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/75435\_levenhuk-d400t-3-1m-digital-trinocular-microscope\_00.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/7858388fc9e608c70424b5dd1567283f.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/8d709d7256a9e54454dc59065a33b39b3ad821ca14d1dc89924e030a33ca8c80.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/Micromet3m.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/ad95bd0455da1afcac911f74c37cf19d0bdb889e0b278e067fee385d97ccd775.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/binokmikroskop1-700x700.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/binokulyarnyj-mikroskop-n-300m.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/bnucmchh9zw52o1ktlftmmabmq3uh6zd.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/bresser\_junior\_8855000.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/df1c5bea2624439ccf79b5f948f68d85.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/editor5094.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/editor5898.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/editor8740.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/editor8742.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/ef960df9a09887d20c2bef899d9615b6.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/img\_5.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/m\_logo.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/micmed-5\_1od\_enl.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/microscope-levenhuk-720b.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop-levenhuk-labzz-m2.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop\_biologicheskiy\_mikromed\_s\_13\_1.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop\_discovery\_nano\_terra\_s\_knigoy\_1637599206\_1.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskopy\_3260f41c541737d\_800x600.jpg?raw=true

# микроскоп;https://github.com/Shaman-dp/neuroDataset/blob/main/shop\_items\_catalog\_image37040.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_1.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_2.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_3.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_4.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_5.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_6.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_7.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_8.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_9.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_10.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_11.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_12.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_13.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_14.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_15.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_16.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_17.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_18.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_19.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_20.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_21.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_22.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_23.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_24.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_25.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_26.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_27.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_28.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_29.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_30.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_31.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_32.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_33.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_34.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_35.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_36.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_37.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_38.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_39.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_40.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_41.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_42.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_43.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_44.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_45.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_46.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_47.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_48.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_49.jpg?raw=true

# лобзик;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_50.jpg?raw=true

"""# alexNet"""

model\_alexNet = alexNetCPU

transform = model\_alexNet['preprocess']

simple = UrlDataset("simple.csv", deviceCPU, transform)

print('-'\*70)

print(f'Model on CPU: { model\_alexNet["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_alexNet["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_alexNet\_detect\_CPU, model\_alexNet\_detect\_time\_CPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_alexNet['model'], model\_labels=model\_alexNet['weights'].meta["categories"],

debug=True)

model\_alexNet = alexNetGPU

transform = model\_alexNet['preprocess']

simple = UrlDataset("simple.csv", deviceGPU, transform)

print('-'\*70)

print(f'Model on GPU: { model\_alexNet["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_alexNet["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_alexNet\_detect\_GPU, model\_alexNet\_detect\_time\_GPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_alexNet['model'], model\_labels=model\_alexNet['weights'].meta["categories"],

debug=True)

"""# resNet50"""

model\_resNet50 = resNet50CPU

transform = model\_resNet50['preprocess']

simple = UrlDataset("simple.csv", deviceCPU, transform)

print('-'\*70)

print(f'Model on CPU: { model\_resNet50["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_resNet50["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_resNet50\_detect\_CPU, model\_resNet50\_detect\_time\_CPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_resNet50['model'], model\_labels=model\_resNet50['weights'].meta["categories"],

debug=True)

model\_resNet50 = resNet50GPU

transform = model\_resNet50['preprocess']

simple = UrlDataset("simple.csv", deviceGPU, transform)

print('-'\*70)

print(f'Model on GPU: { model\_resNet50["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_resNet50["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_resNet50\_detect\_GPU, model\_resNet50\_detect\_time\_GPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_resNet50['model'], model\_labels=model\_resNet50['weights'].meta["categories"],

debug=True)

"""# denseNet"""

model\_denseNet = denseNetCPU

transform = model\_denseNet['preprocess']

simple = UrlDataset("simple.csv", deviceCPU, transform)

print('-'\*70)

print(f'Model on CPU: { model\_denseNet["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_denseNet["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_denseNet\_detect\_CPU, model\_denseNet\_detect\_time\_CPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_denseNet['model'], model\_labels=model\_denseNet['weights'].meta["categories"],

debug=True)

model\_denseNet = denseNetGPU

transform = model\_denseNet['preprocess']

simple = UrlDataset("simple.csv", deviceGPU, transform)

print('-'\*70)

print(f'Model on GPU: { model\_denseNet["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_denseNet["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_denseNet\_detect\_GPU, model\_denseNet\_detect\_time\_GPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_denseNet['model'], model\_labels=model\_denseNet['weights'].meta["categories"],

debug=True)

"""# mobileNetV3\_small"""

model\_mobileNetV3\_small = mobileNetV3\_smallCPU

transform = model\_mobileNetV3\_small['preprocess']

simple = UrlDataset("simple.csv", deviceCPU, transform)

print('-'\*70)

print(f'Model on CPU: { model\_mobileNetV3\_small["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_mobileNetV3\_small["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_mobileNetV3\_small\_detect\_CPU, model\_mobileNetV3\_small\_detect\_time\_CPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_mobileNetV3\_small['model'], model\_labels=model\_mobileNetV3\_small['weights'].meta["categories"],

debug=True)

model\_mobileNetV3\_small = mobileNetV3\_smallGPU

transform = model\_mobileNetV3\_small['preprocess']

simple = UrlDataset("simple.csv", deviceGPU, transform)

print('-'\*70)

print(f'Model on GPU: { model\_mobileNetV3\_small["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_mobileNetV3\_small["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_mobileNetV3\_small\_detect\_GPU, model\_mobileNetV3\_small\_detect\_time\_GPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_mobileNetV3\_small['model'], model\_labels=model\_mobileNetV3\_small['weights'].meta["categories"],

debug=True)

"""# swinT"""

model\_swinT = swinTGPU

transform = model\_swinT['preprocess']

simple = UrlDataset("simple.csv", deviceGPU, transform)

print('-'\*70)

print(f'Model on GPU: { model\_swinT["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_swinT["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_swinT\_detect\_GPU, model\_swinT\_detect\_time\_GPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_swinT['model'], model\_labels=model\_swinT['weights'].meta["categories"],

debug=True)

model\_swinT = swinTCPU

transform = model\_swinT['preprocess']

simple = UrlDataset("simple.csv", deviceCPU, transform)

print('-'\*70)

print(f'Model on CPU: { model\_swinT["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_swinT["model"].parameters())}')

num\_per\_row = 5

single\_size = 3.5

vspace = 0.4

model\_swinT\_detect\_CPU, model\_swinT\_detect\_time\_CPU = classify(simple, denormalize(simple, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace = vspace,

labels = simple.classes, model=model\_swinT['model'], model\_labels=model\_swinT['weights'].meta["categories"],

debug=True)

"""# сравнение"""

model\_title = ['alexNet', 'resNet50', 'denseNet', 'mobileNetV3', 'swinT']

object\_detect\_CPU = [model\_alexNet\_detect\_CPU,

model\_resNet50\_detect\_CPU,

model\_denseNet\_detect\_CPU,

model\_mobileNetV3\_small\_detect\_CPU,

model\_swinT\_detect\_CPU]

# object\_detect\_GPU = [model\_alexNet\_detect\_GPU,

# model\_resNet50\_detect\_GPU,

# model\_denseNet\_detect\_GPU,

# model\_mobileNetV3\_small\_detect\_GPU,

# model\_swinT\_detect\_GPU]

width = 0.4

x = numpy.arange(5)

fig, ax = plt.subplots()

rects1 = ax.bar(x - width/2, object\_detect\_CPU, width, label = 'CPU')

# rects2 = ax.bar(x + width/2, object\_detect\_GPU, width, label = 'GPU')

ax.set\_xticks(x)

ax.set\_xticklabels(model\_title)

ax.legend()

time\_detect\_CPU = [model\_alexNet\_detect\_time\_CPU,

model\_resNet50\_detect\_time\_CPU,

model\_denseNet\_detect\_time\_CPU,

model\_mobileNetV3\_small\_detect\_time\_CPU,

model\_swinT\_detect\_time\_CPU]

time\_detect\_GPU = [model\_alexNet\_detect\_time\_GPU,

model\_resNet50\_detect\_time\_GPU,

model\_denseNet\_detect\_time\_GPU,

model\_mobileNetV3\_small\_detect\_time\_GPU,

model\_swinT\_detect\_time\_GPU]

width = 0.4

x = numpy.arange(5)

fig, ax = plt.subplots()

rects1 = ax.bar(x - width/2, time\_detect\_CPU, width, label = 'CPU', color='black')

rects2 = ax.bar(x + width/2, time\_detect\_GPU, width, label = 'GPU', color='#A3A3A3')

ax.set\_xticks(x)

ax.set\_xticklabels(model\_title)

ax.legend()

"""# train file"""

# Commented out IPython magic to ensure Python compatibility.

# %%writefile train.csv

# label;url

#

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/102-6155-20532-detskiy-mikroskop-100x-1200x.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/1195857\_v01\_b.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/11971.600.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/148875096.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/1595273\_v01\_b.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/1\_big.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/23337.600.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/28046.970.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/3.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/3870.970.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/41c4a7f010.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/45610044.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/4bc158b1f08b03162be756c495e7900c.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/4d2d3456c037c165f82b823320dc3338.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/6049003855.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/68ba85225ca2946d610cce170e3655fb.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/69051\_levenhuk-rainbow-50l-plus-moonstone\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/73985\_levenhuk-trinocular-microscope-med-10t\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/74004\_levenhuk-med-40b-binocular-microscope\_01.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/74008\_levenhuk-med-45b-binocular-microscope\_02.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/74319\_bresser-mikroskop-junior-biolux-sel-40-1600x-zelenyj\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/74319\_bresser-mikroskop-junior-biolux-sel-40-1600x-zelenyj\_09.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/74322\_bresser-mikroskop-junior-biolux-sel-40-1600x-sinij\_09.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/75419\_levenhuk-microscope-400m\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/75425\_levenhuk-microscope-500b\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/75435\_levenhuk-d400t-3-1m-digital-trinocular-microscope\_00.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/7858388fc9e608c70424b5dd1567283f.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/8d709d7256a9e54454dc59065a33b39b3ad821ca14d1dc89924e030a33ca8c80.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/Micromet3m.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/ad95bd0455da1afcac911f74c37cf19d0bdb889e0b278e067fee385d97ccd775.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/binokmikroskop1-700x700.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/binokulyarnyj-mikroskop-n-300m.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/bnucmchh9zw52o1ktlftmmabmq3uh6zd.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/bresser\_junior\_8855000.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/df1c5bea2624439ccf79b5f948f68d85.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/editor5094.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/editor5898.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/editor8740.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/editor8742.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/ef960df9a09887d20c2bef899d9615b6.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_1.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_2.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_3.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_4.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_5.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_6.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_7.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_8.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_9.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_10.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_11.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_12.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_13.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_14.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_15.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_16.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_17.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_18.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_19.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_20.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_21.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_22.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_23.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_24.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_25.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_26.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_27.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_28.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_29.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_30.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_31.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_32.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_33.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_34.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_35.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_36.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_37.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_38.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_39.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_40.jpg?raw=true

"""# test file"""

# Commented out IPython magic to ensure Python compatibility.

# %%writefile test.csv

# label;url

#

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/img\_5.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/m\_logo.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/micmed-5\_1od\_enl.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/microscope-levenhuk-720b.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop-levenhuk-labzz-m2.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop\_biologicheskiy\_mikromed\_s\_13\_1.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskop\_discovery\_nano\_terra\_s\_knigoy\_1637599206\_1.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/mikroskopy\_3260f41c541737d\_800x600.jpg?raw=true

# microscope;https://github.com/Shaman-dp/neuroDataset/blob/main/shop\_items\_catalog\_image37040.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_41.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_42.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_43.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_44.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_45.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_46.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_47.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_48.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_49.jpg?raw=true

# jigsaw;https://github.com/Shaman-dp/neuroDataset/blob/main/image\_50.jpg?raw=true

"""# Test"""

transform = model\_swinT['preprocess']

train = UrlDataset("train.csv", deviceCPU, transform)

test = UrlDataset('test.csv', deviceCPU, transform)

num\_per\_row = 5

single\_size = 3.5

vspace = 0.3

print('-'\*70)

print(f'Model: { model\_swinT["model"].\_\_class\_\_.\_\_name\_\_}')

print(f'Number of parameters: {sum(item.numel() for item in model\_swinT["model"].parameters())}')

print('-'\*30 + ' Train dataset ' + '-'\*30)

classify(train, denormalize(train, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace=vspace,

labels = train.classes, model=model\_swinT['model'], model\_labels=model\_swinT['weights'].meta["categories"])

print('-'\*30 + ' Test dataset ' + '-'\*30)

classify(test, denormalize(test, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace=vspace,

labels = test.classes, model=model\_swinT['model'], model\_labels=model\_swinT['weights'].meta["categories"])

image, label = simple[0]

feature\_extractor = copy.deepcopy(model\_swinT['model'])

result = feature\_extractor(image.unsqueeze(0)).cpu().detach().squeeze(0).numpy()

print(f'Размерность выходов исходной модели: {result.shape}')

setattr(feature\_extractor, model\_swinT['output\_layer'], nn.Identity())

result = feature\_extractor(image.unsqueeze(0)).cpu().detach().squeeze(0).numpy()

print(f'Размерность выходов модели после замены выходного классификатора на единичное преобразование: {result.shape}')

output = []

for image, label in torch.utils.data.ConcatDataset([train, test]):

result = {'label': label, 'feature': feature\_extractor(image.unsqueeze(0)).cpu().detach().squeeze(0).numpy()}

output.append(result)

features = numpy.concatenate([numpy.expand\_dims(record['feature'], 0) for record in output])

labels = [record['label'] for i, record in enumerate(output)]

totalU, totalS, totalV = torch.pca\_lowrank(torch.from\_numpy(features), q=2)

series = {}

for key in ['microscope', 'jigsaw']:

points = numpy.concatenate([numpy.expand\_dims(totalU[i,:].cpu().detach().numpy(), 0)

for i, label in enumerate(labels) if train.classes[label] == key])

series[key] = numpy.moveaxis(points, 1, 0)

plt.figure()

plt.plot(series['microscope'][0], series['microscope'][1], 'ro', series['jigsaw'][0], series['jigsaw'][1], 'bo')

plt.legend(['microscope', 'jigsaw'])

plt.show()

tunned\_model = copy.deepcopy(model\_swinT['model'])

for param in tunned\_model.parameters():

param.requires\_grad = False

print(getattr(tunned\_model, model\_swinT['output\_layer']))

in\_features = model\_swinT['in\_features'](tunned\_model)

print(in\_features)

setattr(tunned\_model, model\_swinT['output\_layer'], torch.nn.Sequential(torch.nn.Linear(in\_features, 2), torch.nn.Softmax(dim=1)))

print(f"num of adjustable parameters = {sum(i.numel() for i in getattr(tunned\_model, model\_swinT['output\_layer']).parameters())}")

dataloader = [

{

'train': True,

'loader': DataLoader(train, batch\_size=1, shuffle=True, num\_workers=0)

},

{

'train': False,

'loader': DataLoader(test, batch\_size=1, shuffle=True, num\_workers=0)

}

]

tunned\_model.to(deviceCPU)

critery = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(getattr(tunned\_model, model\_swinT['output\_layer']).parameters(), lr=0.001, momentum=0.9)

lr\_scheduler = torch.optim.lr\_scheduler.StepLR(optimizer, step\_size=5, gamma=0.5,verbose=True)

model\_conv, process = train\_model(model=tunned\_model, dataloader=dataloader, device=deviceCPU,

critery=critery, optimizer=optimizer, scheduler=lr\_scheduler,

num\_epochs=9)

fig, axs = plt.subplots(2,2, figsize=(16,10), sharex=True)

axs[0,0].plot(process['train']['loss'])

axs[0,0].title.set\_text('Эволюции ошибки на выборке для обучения')

axs[0,1].plot(process['train']['accuracy'])

axs[0,1].title.set\_text('Эволюции ошибки на выборке для обучения')

axs[1,0].plot(process['validate']['loss'])

axs[1,0].title.set\_text('Эволюции ошибки на выборке для проверки')

axs[1,1].plot(process['validate']['accuracy'])

axs[1,1].title.set\_text('Эволюции ошибки на выборке для проверки')

axs[1,0].set\_xlabel('Итерация обучения')

axs[1,1].set\_xlabel('Итерация обучения')

axs[0,0].set\_ylabel('Значение ошибки')

axs[1,0].set\_ylabel('Значение ошибки')

print('-'\*30 + ' Train dataset ' + '-'\*30)

classify(train, denormalize(train, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace=vspace,

labels = train.classes, model=tunned\_model, model\_labels=train.classes, debug=True)

print('-'\*30 + ' Test dataset ' + '-'\*30)

classify(test, denormalize(test, transform), num\_per\_row=num\_per\_row, single\_size=single\_size, vspace=vspace,

labels = test.classes, model=tunned\_model, model\_labels=test.classes)