МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

Институт кибербезопасности и цифровых технологий

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Отчёт по практической работе № 1

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Выполнил:

ББМО–02–22

Шмарковский М. Б.

Проверил:

Спирин А. А.

Москва, 2024

Практическая работа №1

Выполнил Шмарковский М.Б. ББМО-02-22

Основы работы с моделями нейронных сетей

Пункт 1 Импорт библиотек, необходимых для работы.

```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu121)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: torch==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.1.0+cu121)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (4.5.0
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0->torchvision) (2.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2023.11.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch==2.1.0->torchvision)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch==2.1.0->torchvision) (1.3
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import DataLoader, Dataset

from torchvision import datasets
from torchvision.transforms import ToTensor, Resize

import numpy as np

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Пункт 2 Загрузка открытых датасетов из пакета datasets фреймворка pytorch

```
# Загрузка обучающего датасета.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Загрузка тестового датасета.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-im
100%|██████████| 26421880/26421880 [00:01<00:00, 14928937.16it/s]
Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-la
100%|██████████| 29515/29515 [00:00<00:00, 270127.00it/s]
Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-imag
100%|██████████| 4422102/4422102 [00:00<00:00, 4902470.74it/s]
Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labe
100%|██████████| 5148/5148 [00:00<00:00, 5721324.06it/s]Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionM
```

```
# Загрузка классов данных, присутствующих в датасете
test_data.classes
#test_data.data
```

```
['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']
```

Пункт 3 После того, как датасеты были загружены, данные из них необходимо упаковать в пакеты (батчи). Основное преимущество нейросетей заключается в параллельной обработке данных, выполняющейся за счет того, что данные подаются на вход группами (батчами), то есть порциями. Чем больше размер пакета (батча), тем выше скорость обучения нейронной сети, но также возрастает ресурсоемкость сети. Таким образом, необходимо **выбирать размер пакета** таким образом, чтобы сохранялся **баланс между скоростью обучения и ресурсоемкостью**.
Для формирования пакетов используется класс DataLoader. С его помощью можно сформировать батчи необходимой длины, перемешать данные или добавить особую функцию (collate function), которая будет формировать пакеты.
В ячейке ниже предоставлены входные данные (тензор размерности [N, C, H, W], где N - размер пакета, C - количество каналов (1 - для ч/б изображений, 3 или 4 в зависимости от изображения RGB или RGBA), H и W - height и width, высота и ширина изображения). И выходные данные с указанием метки класса.

```
batch_size = 64

# Создание загрузчиков данных.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64]) torch.int64
```

Пункт 4 Описание модели. Модель нейронной сети, создается фреймворком pytorch с помощью пакета torch.nn, который мы импортировали на шаге 1.

Перед этим, необходимо выбрать устройство, на котором будет размещаться наша модель. Обычно существует два варианта: GPU/CUDA (видеокарта) или CPU (процессор). В google colab есть выбор запуска среды как с CUDA так и на CPU.

Для того, чтобы описать модель, необходимо создать класс, в нашем случае это класс NeuralNetwork, который будет наследовать интерфейс нейронной сети от класса nn.Module. Согласно этому интерфейсу в классе NeuralNetwork необходимо описать слои нейронной сети в конструкторе класса (***init***), а также порядок прохождения и обработки информации через слои с помощью перегрузки метода *forward*.

Подробное описание слоев предоставлено в комментариях в коде в ячейке ниже.

```
# Выбираем девайс GPU, если недоступна, то CPU.
device = (
    "cuda" if torch.cuda.is_available() else "cpu"
)
print(f"Используемое устройство: {device}")

# Описание нейронной сети
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
```

```
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```
Используемое устройство: cuda
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Пункт 5 Выбор функции ошибок, алгоритмов оптимизации и определения функции обучения.

```
#  в данном примере рассматривается многоклассовая классификация, поэтому наиболее
#  рациональным решением будет использование перекрестной энтропии (кросс-энтропии),
#  а в качестве примера алгоритм оптимизации Стохастичный Градиентный Спуск (SGD).
loss_fn = nn.CrossEntropyLoss()
#  параметр lr (learning_rate - шаг обучения) определяет размер шага, которым
#  алгоритм будет достигать максимума градиента.
#  При увеличении lr увеличивается скорость, но ухудшается точность, при уменьшении
#  увеличивается точность, но уменьшается скорость обучения.
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)


def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        #  вычисляем ошибку предсказания модели
        pred = model(X)
        loss = loss_fn(pred, y)

        #  реализация метода обратного распространения ошибки
        #  backward вычисляет градиенты, а step производит обновление весов
        loss.backward()
        optimizer.step()
        #  метод zero_grad производит обнуление градиентов, чтобы это не мешало
        #  дальнейшему обучению сети
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Пункт 6 Обучение нейронной сети.

Эпоха, это итерация, на которой модель использует данные из обучающей выборки. Согласно гипотезе, чем больше эпох, тем лучше, так как чем больше человек повторит какой либо материал, тем лучше он запомнит его.

Но в случае с нейронной сетью это не так. При увеличении числа эпох, в какой-то момент произойдет "переобучение сети", из-за чего точность на тестовой выборке начнет падать. Природа данного эффекта заключается в том, что модель слишком сильно настраивается на данные обучающей выборки, из-за чего теряется способность обобщающая способность нейронной сети. Более подробно см: https://proproprogs.ru/neural_network/pereobuchenie-chto-eto-i-kak-etogo-izbezhat-kriterii-ostanova-obucheniya?ysclid=lmlrjx3fbg719891347

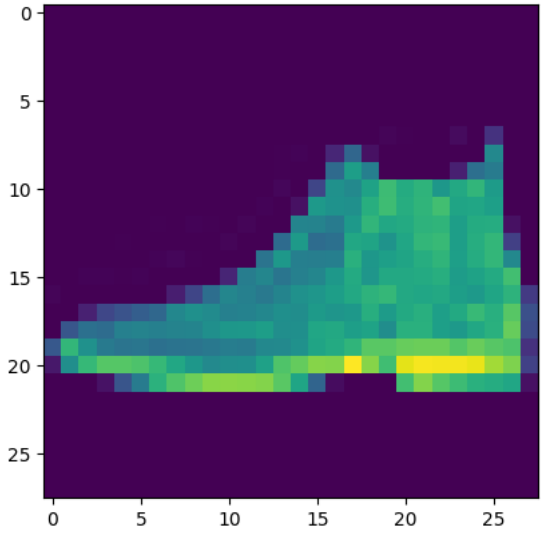
```
epochs = 5
for t in range(epochs):
    print(f"Эпоха {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Готово!")
```

```
 loss: 2.151437  [ 6464/60000]
 loss: 2.089043  [12864/60000]
 loss: 2.117546  [19264/60000]
 loss: 2.069018  [25664/60000]
 loss: 1.994506  [32064/60000]
 loss: 2.048756  [38464/60000]
 loss: 1.955088  [44864/60000]
 loss: 1.970015  [51264/60000]
 loss: 1.900724  [57664/60000]
 Test Error:
  Accuracy: 56.3%, Avg loss: 1.893831

 Эпоха 3
 -------------------------------
 loss: 1.914874  [   64/60000]
 loss: 1.894247  [ 6464/60000]
 loss: 1.771136  [12864/60000]
 loss: 1.831579  [19264/60000]
 loss: 1.716821  [25664/60000]
 loss: 1.655013  [32064/60000]
 loss: 1.705674  [38464/60000]
 loss: 1.581850  [44864/60000]
 loss: 1.620687  [51264/60000]
 loss: 1.516504  [57664/60000]
 Test Error:
  Accuracy: 61.1%, Avg loss: 1.525324

 Эпоха 4
 -------------------------------
 loss: 1.581709  [   64/60000]
 loss: 1.552731  [ 6464/60000]
 loss: 1.393811  [12864/60000]
 loss: 1.487792  [19264/60000]
 loss: 1.360162  [25664/60000]
 loss: 1.348939  [32064/60000]
 loss: 1.383316  [38464/60000]
 loss: 1.285551  [44864/60000]
 loss: 1.334485  [51264/60000]
 loss: 1.231044  [57664/60000]
 Test Error:
  Accuracy: 62.7%, Avg loss: 1.254177

 Эпоха 5
 -------------------------------
 loss: 1.324581  [   64/60000]
 loss: 1.309241  [ 6464/60000]
 loss: 1.137177  [12864/60000]
 loss: 1.262167  [19264/60000]
 loss: 1.132387  [25664/60000]
 loss: 1.154783  [32064/60000]
 loss: 1.186323  [38464/60000]
 loss: 1.105650  [44864/60000]
 loss: 1.158466  [51264/60000]
 loss: 1.066743  [57664/60000]
 Test Error:
  Accuracy: 64.3%, Avg loss: 1.088297

 Готово!
```

```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]
```

```
model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
print("Входное изображение:")
plt.imshow(np.transpose(x.detach().cpu().numpy(), (1,2,0)))
```

```
Predicted: "Ankle boot", Actual: "Ankle boot"
Входное изображение:
<matplotlib.image.AxesImage at 0x7f523ab86d70>
```



Визуализация работы ИИ

Пункт 1 Установка дополнительного пакета *torchcam* для визуализации зон изображений, на основе которых модель принимает решение.

```
!pip install torch==1.13.1 torchvision==0.14.1 torchaudio==0.13.1
!pip install torchcam
```

```
Collecting torch==1.13.1
  Downloading torch-1.13.1-cp310-cp310-manylinux1_x86_64.whl (887.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 887.5/887.5 MB 1.6 MB/s eta 0:00:00
Collecting torchvision==0.14.1
  Downloading torchvision-0.14.1-cp310-cp310-manylinux1_x86_64.whl (24.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.2/24.2 MB 10.6 MB/s eta 0:00:00
Collecting torchaudio==0.13.1
  Downloading torchaudio-0.13.1-cp310-cp310-manylinux1_x86_64.whl (4.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 15.7 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch==1.13.1) (4.5.0)
Collecting nvidia-cuda-runtime-cu11==11.7.99 (from torch==1.13.1)
  Downloading nvidia_cuda_runtime_cu11-11.7.99-py3-none-manylinux1_x86_64.whl (849 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 849.3/849.3 kB 23.4 MB/s eta 0:00:00
Collecting nvidia-cudnn-cu11==8.5.0.96 (from torch==1.13.1)
  Downloading nvidia_cudnn_cu11-8.5.0.96-2-py3-none-manylinux1_x86_64.whl (557.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 557.1/557.1 MB 2.2 MB/s eta 0:00:00
Collecting nvidia-cublas-cu11==11.10.3.66 (from torch==1.13.1)
  Downloading nvidia_cublas_cu11-11.10.3.66-py3-none-manylinux1_x86_64.whl (317.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.1/317.1 MB 4.0 MB/s eta 0:00:00
Collecting nvidia-cuda-nvrtc-cu11==11.7.99 (from torch==1.13.1)
  Downloading nvidia_cuda_nvrtc_cu11-11.7.99-2-py3-none-manylinux1_x86_64.whl (21.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.0/21.0 MB 44.2 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision==0.14.1) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision==0.14.1) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision==0.14.1) (9.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from nvidia-cublas-cu11==11.10.3.66->torch==1
Requirement already satisfied: wheel in /usr/local/lib/python3.10/dist-packages (from nvidia-cublas-cu11==11.10.3.66->torch==1.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision==0.14
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision==0.14.1) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision==0.14.1) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision==0.14.1) (2
Installing collected packages: nvidia-cuda-runtime-cu11, nvidia-cuda-nvrtc-cu11, nvidia-cublas-cu11, nvidia-cudnn-cu11, torch, torch
  Attempting uninstall: torch
    Found existing installation: torch 2.1.0+cu121
    Uninstalling torch-2.1.0+cu121:
      Successfully uninstalled torch-2.1.0+cu121
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.16.0+cu121
    Uninstalling torchvision-0.16.0+cu121:
      Successfully uninstalled torchvision-0.16.0+cu121
  Attempting uninstall: torchaudio
    Found existing installation: torchaudio 2.1.0+cu121
    Uninstalling torchaudio-2.1.0+cu121:
      Successfully uninstalled torchaudio-2.1.0+cu121
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
torchdata 0.7.0 requires torch==2.1.0, but you have torch 1.13.1 which is incompatible.
torchtext 0.16.0 requires torch==2.1.0, but you have torch 1.13.1 which is incompatible.
Successfully installed nvidia-cublas-cu11-11.10.3.66 nvidia-cuda-nvrtc-cu11-11.7.99 nvidia-cuda-runtime-cu11-11.7.99 nvidia-cudnn-cu
WARNING: The following packages were previously imported in this runtime:
  [torch,torchgen,torchvision]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

```
Collecting torchcam
  Downloading torchcam-0.4.0-py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 890.1 kB/s eta 0:00:00
Collecting torch<3.0.0,>=2.0.0 (from torchcam)
  Downloading torch-2.1.2-cp310-cp310-manylinux1_x86_64.whl (670.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 670.2/670.2 MB 756.9 kB/s eta 0:00:00
Requirement already satisfied: numpy<2.0.0,>=1.17.2 in /usr/local/lib/python3.10/dist-packages (from torchcam) (1.23.5)
Requirement already satisfied: Pillow!=9.2.0,>=8.4.0 in /usr/local/lib/python3.10/dist-packages (from torchcam) (9.4.0)
Requirement already satisfied: matplotlib<4.0.0,>=3.7.0 in /usr/local/lib/python3.10/dist-packages (from torchcam) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam) (0
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torchcam)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4.0.0,>=3.7.0->torch
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (4
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 35.7 MB/s eta 0:00:00
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 66.0 MB/s eta 0:00:00
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 47.2 MB/s eta 0:00:00
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 1.7 MB/s eta 0:00:00
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.6 MB/s eta 0:00:00
```

```
                                     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.0 MB/s eta 0:00:00
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.6/121.6 MB 8.5 MB/s eta 0:00:00
Collecting nvidia-curand-cu12==10.3.2.106 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 MB 10.4 MB/s eta 0:00:00
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.2/124.2 MB 6.0 MB/s eta 0:00:00
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.0/196.0 MB 2.4 MB/s eta 0:00:00
Collecting nvidia-nccl-cu12==2.18.1 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_nccl_cu12-2.18.1-py3-none-manylinux1_x86_64.whl (209.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 209.8/209.8 MB 5.2 MB/s eta 0:00:00
Collecting nvidia-nvtx-cu12==12.1.105 (from torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 14.9 MB/s eta 0:00:00
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch<3.0.0,>=2.0.0->torchcam) (2.1.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch<3.0.0,>=2.0.0->torchcam)
  Downloading nvidia_nvjitlink_cu12-12.3.101-py3-none-manylinux1_x86_64.whl (20.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.5/20.5 MB 17.9 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib<4.0.0,>=3
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch<3.0.0,>=2.0.0->torchca
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch<3.0.0,>=2.0.0->torchcam) (
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvi
  Attempting uninstall: torch
    Found existing installation: torch 1.13.1
    Uninstalling torch-1.13.1:
      Successfully uninstalled torch-1.13.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
torchaudio 0.13.1 requires torch==1.13.1, but you have torch 2.1.2 which is incompatible.
torchdata 0.7.0 requires torch==2.1.0, but you have torch 2.1.2 which is incompatible.
torchtext 0.16.0 requires torch==2.1.0, but you have torch 2.1.2 which is incompatible.
torchvision 0.14.1 requires torch==1.13.1, but you have torch 2.1.2 which is incompatible.
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runti
WARNING: The following packages were previously imported in this runtime:
  [torch,torchgen]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

Пункт 2 Сама визуализация

```
from torchvision.io import read_image
from torchvision.transforms.functional import normalize, resize, to_pil_image
from torchvision.models import resnet18
from torchcam.methods import SmoothGradCAMpp
from torchvision import transforms

model = resnet18(pretrained=True).eval()
cam_extractor = SmoothGradCAMpp(model)
# Открываем картинку
img = read_image("input.jpg")
# Предобрабатываем картинку для входа нашей нейронной сети
input_tensor = normalize(resize(img, (224, 224)) / 255., [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

# Получаем выходные данные на основе модели
out = model(input_tensor.unsqueeze(0))
# Получаем карту активации нейроннов последнего слоя модели resnet18
activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)
plt.title("Карта последнего сверточного слоя ResNet18")
plt.imshow(np.transpose(activation_map[0].detach().cpu().numpy(), (1, 2, 0)))
```

```
WARNING:root:no value was provided for `target_layer`, thus set to 'layer4'.
<matplotlib.image.AxesImage at 0x7f523a69de70>
```

Пункт 3 Наложение маски на исходное изображение

Открываем изображение с помощью cv2

```
import cv2
img = cv2.imread("input.jpg")
```

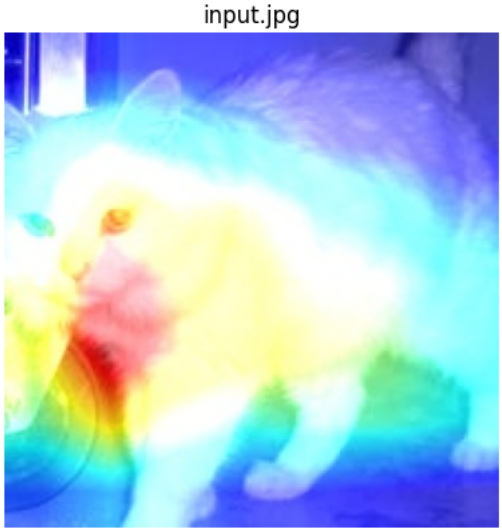
Растяжка карты до размеров исходного изображения

```
heatmap = cv2.resize(activation_map[0].squeeze(0).numpy(), (img.shape[1], img.shape[0]))
```

Преобразование карты в RGB, наложение на изображение.

```
# Загрузить фото кота в локальную среду выполнения ноутбука (сессионное хранилище)
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
hif = .8
superimposed_img = heatmap * hif + img
plt.imshow(superimposed_img)
output = 'output.jpg'
cv2.imwrite(output, superimposed_img)
img=mpimg.imread(output)
plt.imshow(img)
plt.axis('off')
plt.title("input.jpg")
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers
Text(0.5, 1.0, 'input.jpg')
```



Применение state-of-the-art решений на примере датасета FGVAircraft

Пункт 1 Аугментация данных.

Аугментация данных - это умышленное изменение/искажение исходных данных, для улучшения обобщающих способностей при обучении нейронной сети. В качестве аугментации изображений применяют различные повороты, наклоны изображений, изменение цветовых параметров пикселей и т.д.

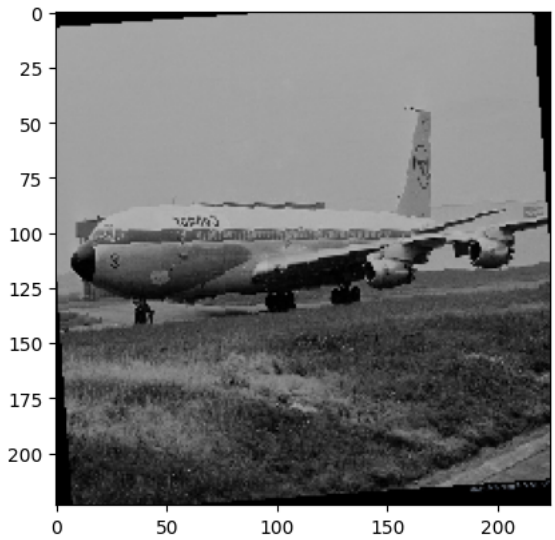
```
# описание функции трансформации изображений для реализации аугментаци
augmentation = transforms.Compose([
    # приведение матрицы к тензору pytorch
    ToTensor(),
    # уменьшение размера до 224 на 224 (требование большинства архитектур resnet, vgg ....)
    transforms.Resize((224, 224)),
    # случайное отражение по горизонтали
    transforms.RandomHorizontalFlip(),
    # слаучайное отражение по вертикали
    transforms.RandomVerticalFlip(),
    # случайный поворот изображения до 20 градусов
    transforms.RandomRotation(20),
    # случайное изменение яркости и оттенка изображения
    transforms.ColorJitter(brightness=.3, hue=.2)
])
# подробнее обо всех методах модуля transforms можно узнать в документации https://pytorch.org/vision/stable/transforms.html
```

```
train = datasets.FGVCAircraft(
      "data",
      annotation_level="manufacturer",
      split="train",
      download=True,
      transform=augmentation
    )

test = datasets.FGVCAircraft(
      "data",
      annotation_level="manufacturer",
      split="test",
      download=True,
      transform=transforms.Compose([ToTensor(), transforms.Resize((224, 224))])
    )
```

```
Downloading https://www.robots.ox.ac.uk/~vgg/data/fgvc-aircraft/archives/fgvc-aircraft-2013b.tar.gz to data/fgvc-aircraft-2013b.tar
100%|██████████| 2753340328/2753340328 [01:34<00:00, 29276790.04it/s]
Extracting data/fgvc-aircraft-2013b.tar.gz to data
```

```
# изображение после аугментации
plt.imshow(np.transpose(train[0][0], (1, 2, 0)))
```

```
<matplotlib.image.AxesImage at 0x7f51f8154d90>
```




```
# создание загрузчиков данных, которые будут формировать и подавать данные пакетам на вход модели
train_dl = DataLoader(train, batch_size=16, shuffle=True)
test_dl = DataLoader(test, batch_size=4, shuffle=False)
```

Пункт 2 Определение модели. Будем использовать готовую модель архитектуры resnet с 50 слоями.

```
from torchvision.models import resnet50

# создание экземпляра модели и инициализация весами, полученными при обучении на датасете ImageNet
model = resnet50(pretrained=True)
model
```

```
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
      (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
      (fc): Linear(in_features=2048, out_features=1000, bias=True)
    )
```

```
#  так как в филнальном размер выходного вектора больше необходимого нам количества классов, изменим его!
"Кол-во классов в датасете", len(train.classes), "Кол-во классов предсказываемых моделью:", model.fc.out_features
```

```
('Кол-во классов в датасете',
 30,
 'Кол-во классов предсказываемых моделью:',
 1000)
```

```
#  изменение размерности выходного вектора и загрузка модели на устройство (cpu или gpu)
model.fc = nn.Linear(model.fc.in_features, len(train.classes))
model = model.to(device)
```

Пункт 3 Обучение нейронной сети. Функция для обучения и теста написаны по аналогии с такими же в разделе 1.

```
def train_fn(model, loader, criterion, optimizer):
    total_loss, total_acc = 0, 0

    model.train()
```

```
    for batch, (x, y) in enumerate(loader):
      optimizer.zero_grad()
      x,y = x.to(device), y.to(device)
      out = model(x)

      loss = criterion(out, y)

      # рассчет точности
      total_acc += (out.argmax(-1) == y).sum() / out.size(0)
      total_loss += loss.item()

      loss.backward()
      optimizer.step()

      if (batch + 1) % 100 == 0:
        print("-" * 90)
        print("loss:", total_loss / (batch + 1), "acc:", total_acc.item() / (batch + 1), "| batch:", batch+1, "/", len(loader), "|")
    return total_loss / len(loader), total_acc / len(loader)


def evaluate(model, loader, criterion):
    total_acc, total_loss = 0, 0
    model.eval()

    with torch.no_grad():
      for batch, (x,y) in enumerate(loader):
        x,y = x.to(device), y.to(device)
        out = model(x)

        loss = criterion(out, y)

        total_acc += (out.argmax(-1) == y).sum() / out.size(0)
        total_loss += loss.item()

        if (batch + 1) % 100 == 0:
          print("-" * 90)
          print("loss:", total_loss / (batch + 1), "acc:", total_acc.item() / (batch + 1), "| batch:", batch+1, "/", len(loader), "|")
    return total_loss / len(loader), total_acc / len(loader)


# Уменьшил количество эпох для быстродействия
epochs = 1

train_accs, train_losses = [], []
test_accs, test_losses = [], []

# Перекрёстная энтропия для рассчета ошибки мультиклассовой классификации
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

print(f"Process started on device: {device}")

for epoch in range(1, epochs + 1):
    loss, acc = train_fn(model, train_dl, criterion, optimizer)
    train_accs.append(acc.detach().cpu().item())
    train_losses.append(loss)
    print("-" * 90)
    print(f"| epoch: {epoch} | train_acc: {acc} | train_loss: {loss} |")
    eloss, eacc = evaluate(model, test_dl, criterion)
    test_accs.append(eacc.detach().cpu().item())
    test_losses.append(eloss)
    print(f"| epoch: {epoch} | test_acc: {eacc}  | test_loss: {eloss}  |")
```

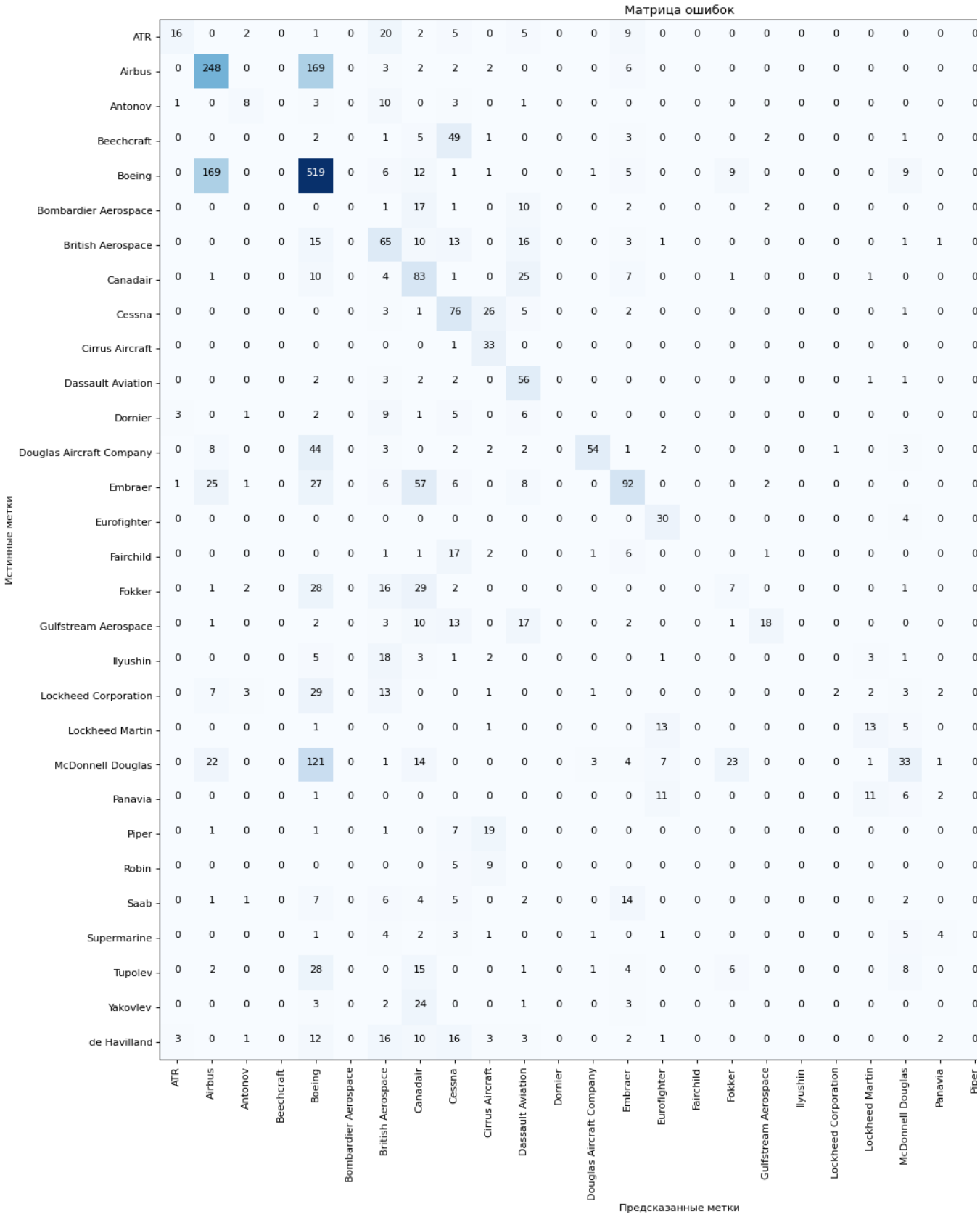
```
 Process started on device: cuda
 ------------------------------------------------------------------------------------------
 loss: 2.439289706945419 acc: 0.284375 | batch: 100 / 209 |
 ------------------------------------------------------------------------------------------
 loss: 2.2066433918476105 acc: 0.33125 | batch: 200 / 209 |
 ------------------------------------------------------------------------------------------
 | epoch: 1 | train_acc: 0.3319377899169922 | train_loss: 2.197934219141326 |
 ------------------------------------------------------------------------------------------
 loss: 0.7325132682919502 acc: 0.74 | batch: 100 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 0.7734098334610462 acc: 0.6925 | batch: 200 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 0.8915949960549673 acc: 0.6366666666666667 | batch: 300 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 1.070887122415006 acc: 0.601875 | batch: 400 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 1.2152543957233428 acc: 0.5685 | batch: 500 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 1.3379709196587404 acc: 0.5375 | batch: 600 / 834 |
 ------------------------------------------------------------------------------------------
 loss: 1.4522059928945132 acc: 0.5032142857142857 | batch: 700 / 834 |
```

```
      ----------------------------------------------------------------------------------------------
      loss: 1.602223882675171 acc: 0.4609375 | batch: 800 / 834 |
      | epoch: 1 | test_acc: 0.4433453381061554  | test_loss: 1.6529644391805434  |
```

```
import itertools
from sklearn.metrics import classification_report, confusion_matrix


def accuracy(pred, y):
    """функция для расчета точности"""
    vals = (pred.argmax(-1) == y)
    return vals.sum() / pred.size(0)


def build_classification_report(pred, y):
    """функция для построения отчета о классификации, выводит для каждого класса
    полноту, точность, Ф1 меру, поддержку, и макро и средне взвешенные метрики
    для всех классов и выводит это в файл"""
    report = classification_report(y, pred, target_names=train.classes)
    with open('classification_report.txt', 'w') as file:
        file.write(report)
    return report


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Матрица ошибок',
                          cmap=plt.cm.Blues):
    """функция для отображения матрицы ошибок классификации"""
    plt.figure(figsize=(20,20), dpi=80)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Истинные метки')
    plt.xlabel('Предсказанные метки')


#  прогон модели на тестовой выборке для получения меток для дальнейшего расчета метрик
preds = []
ys = []
model.eval()
with torch.no_grad():
  for batch, (x, y) in enumerate(test_dl):
    predicted = model(x.to(device))
    preds.extend(predicted.argmax(-1).detach().cpu().tolist())
    ys.extend(y.tolist())
```

```
 /usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias
```

```
plot_confusion_matrix(confusion_matrix(ys, preds), train.classes)
```




```
build_classification_report(preds, ys)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
'                         precision    recall  f1-score   support\n\n                    ATR       0.67      0.24      0.35
0.54       433\n                Antonov       0.42      0.24      0.31        33\n              Beechcraft       0.00      0.00
0.71      0.59       734\n    Bombardier Aerospace       0.00      0.00      0.00        33\n       British Aerospace       0.30
0.27      0.62      0.38       133\n                  Cessna       0.32      0.57      0.41       134\n         Cirrus Aircraft
Aviation       0.35      0.84      0.50        67\n                 Dornier       0.00      0.00      0.00        33\nDouglas Aircra
Embraer       0.56      0.39      0.46      ...'
```

```
# график изменения ошибки и точности от эпохи
fig = plt.figure(num = 2)
fig1 = fig.add_subplot(2,1,1)
fig2 = fig.add_subplot(2,1,2)
fig1.plot(train_losses, label = 'training loss')
fig1.plot(train_accs, label = 'training accuracy')
fig2.plot(test_losses, label = 'validation loss')
fig2.plot(test_accs, label = 'validation accuracy')
plt.legend()
plt.show()
```

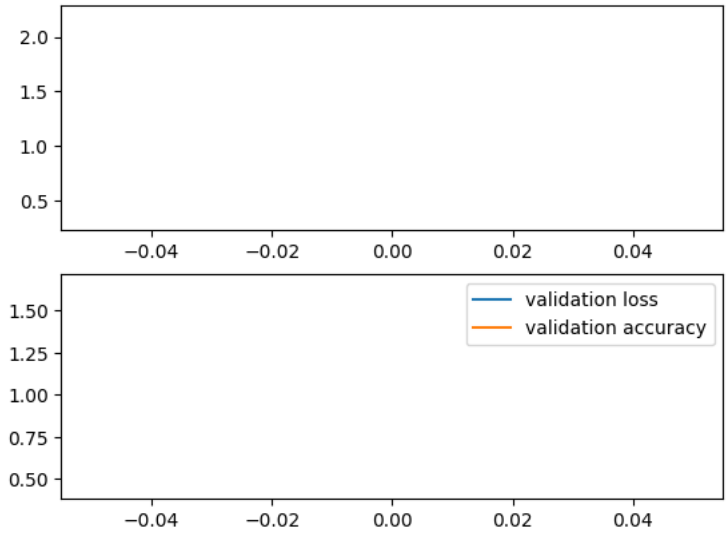


График не отображается из-за всего одной эпохи

Задание создайте тепловую карту последнего сверточного слоя модели resnet50, посмотрите и проанализируйте, корректно ли модель делает акцент для принятия решения или использует косвенные признаки

```
unique = []
for x, y in train:
  if y not in unique:
    unique.append(y)
len(unique)
```

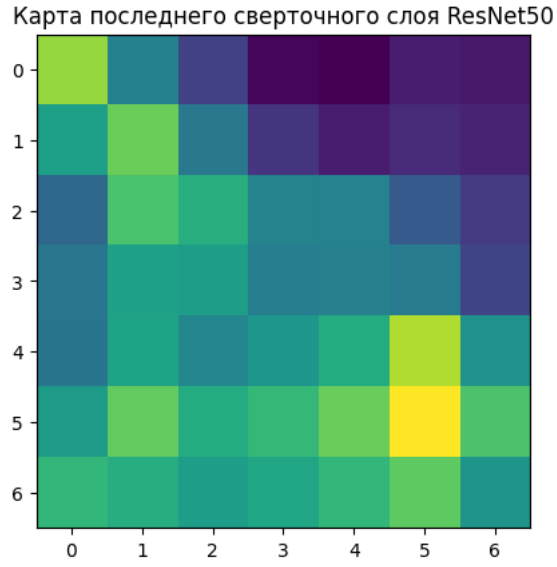
```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias
30
```

```
torch.save(model.cpu().state_dict(), "resnet50.pt")
```

```
model = resnet50(pretrained=False)
model.fc = nn.Linear(2048, len(unique))
model = model.eval()
model.load_state_dict(torch.load("resnet50.pt"))
cam_extractor = SmoothGradCAMpp(model)
# Открываем картинку
img = read_image("input.jpg")
# Предобрабатываем картинку для входа нашей нейронной сети
input_tensor = normalize(resize(img, (224, 224)) / 255., [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

# Получаем выходные данные на основе модели
out = model(input_tensor.unsqueeze(0))
# Получаем карту активации нейроннов последнего слоя модели resnet18
```

```
activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)
plt.title("Карта последнего сверточного слоя ResNet50")
plt.imshow(np.transpose(activation_map[0].detach().cpu().numpy(), (1, 2, 0)))
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
WARNING:root:no value was provided for `target_layer`, thus set to 'layer4'.
<matplotlib.image.AxesImage at 0x7f514e3f8100>
```



Классификация дорожных знаков

```
def augmentation_fn_builder(img_size):
    augmentation = transforms.Compose([
        ToTensor(),
        transforms.Resize((img_size, img_size)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20)
    ])
    return augmentation


train = datasets.GTSRB(
    "data",
    split="train",
    download=True,
    transform=augmentation_fn_builder(224)
)

test = datasets.GTSRB(
    "data",
    split="test",
    download=True,
    transform=transforms.Compose([ToTensor(), transforms.Resize((224, 224))])
)


unique = []
for x, y in train:
  if y not in unique:
    unique.append(y)
len(unique)
```

```
43
```

```
train_dl = DataLoader(train, batch_size=16, shuffle=True)
test_dl = DataLoader(test, batch_size=14, shuffle=False)


model = resnet50(pretrained=True)
model.fc = nn.Linear(2048, len(unique))
model = model.to(device)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
```

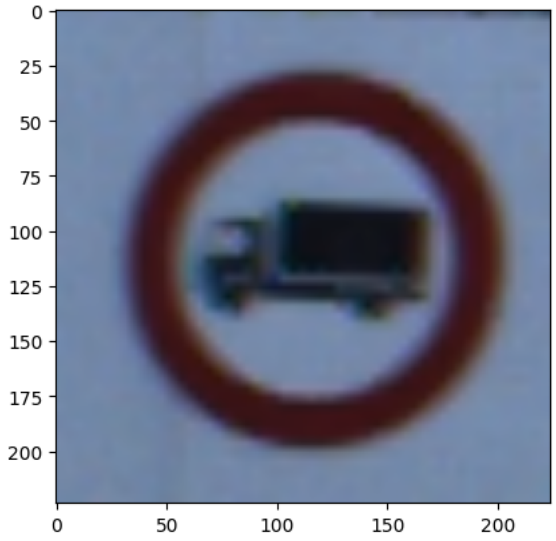
```
def train_fn(model, loader, criterion, optimizer):
    total_loss, total_acc = 0, 0

    model.train()

    for batch, (x, y) in enumerate(loader):
      optimizer.zero_grad()
      x,y = x.to(device), y.to(device)
      out = model(x)

      loss = criterion(out, y)

      # рассчет точности
      total_acc += (out.argmax(-1) == y).sum() / out.size(0)
      total_loss += loss.item()

      loss.backward()
      optimizer.step()

      if (batch + 1) % 100 == 0:
        print("-" * 90)
        print("loss:", total_loss / (batch + 1), "acc:", total_acc.item() / (batch + 1), "| batch:", batch+1, "/", len(loader), "|")
    return total_loss / len(loader), total_acc / len(loader)


def evaluate(model, loader, criterion):
    total_acc, total_loss = 0, 0
    model.eval()

    with torch.no_grad():
      for batch, (x,y) in enumerate(loader):
        x,y = x.to(device), y.to(device)
        out = model(x)

        loss = criterion(out, y)

        total_acc += (out.argmax(-1) == y).sum() / out.size(0)
        total_loss += loss.item()

        if (batch + 1) % 100 == 0:
          print("-" * 90)
          print("loss:", total_loss / (batch + 1), "acc:", total_acc.item() / (batch + 1), "| batch:", batch+1, "/", len(loader), "|")
    return total_loss / len(loader), total_acc / len(loader)


epochs = 1

train_accs, train_losses = [], []
test_accs, test_losses = [], []

# перекрестная энтропия для рассчета ошибки мультиклассовой классификации
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

print(f"Process started on device: {device}")

for epoch in range(1, epochs + 1):
    loss, acc = train_fn(model, train_dl, criterion, optimizer)
    train_accs.append(acc.detach().cpu().item())
    train_losses.append(loss)
    print("-" * 90)
    print(f"| epoch: {epoch} | train_acc: {acc} | train_loss: {loss} |")
    eloss, eacc = evaluate(model, test_dl, criterion)
    test_accs.append(eacc.detach().cpu().item())
    test_losses.append(eloss)
    print(f"| epoch: {epoch} | test_acc: {eacc}  | test_loss: {eloss}  |")
```

```
Process started on device: cuda
------------------------------------------------------------------------------------------
loss: 2.2149759870767594 acc: 0.41375 | batch: 100 / 1665 |
------------------------------------------------------------------------------------------
loss: 1.5988258320093154 acc: 0.5765625 | batch: 200 / 1665 |
------------------------------------------------------------------------------------------
loss: 1.2486161604026953 acc: 0.6689583333333333 | batch: 300 / 1665 |
------------------------------------------------------------------------------------------
loss: 1.0358413204550743 acc: 0.7234375 | batch: 400 / 1665 |
------------------------------------------------------------------------------------------
loss: 0.8980437335819006 acc: 0.758625 | batch: 500 / 1665 |
------------------------------------------------------------------------------------------
```

```
loss: 0.7896821880278488 acc: 0.7869791666666667 | batch: 600 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.7157095301710069 acc: 0.8059821428571429 | batch: 700 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.6553399487980641 acc: 0.821328125 | batch: 800 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.6044665013077772 acc: 0.8347222222222223 | batch: 900 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.5611902165459469 acc: 0.846 | batch: 1000 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.5280421591499312 acc: 0.8544886363636364 | batch: 1100 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.49817177282335856 acc: 0.8622916666666667 | batch: 1200 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.47398378051889056 acc: 0.8683173076923076 | batch: 1300 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.4525308783851298 acc: 0.8740625 | batch: 1400 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.43385665422615904 acc: 0.8785416666666667 | batch: 1500 / 1665 |
--------------------------------------------------------------------------------------------
loss: 0.4169292020337889 acc: 0.8830078125 | batch: 1600 / 1665 |
--------------------------------------------------------------------------------------------
| epoch: 1 | train_acc: 0.885998547077179 | train_loss: 0.4063833852961391 |
--------------------------------------------------------------------------------------------
loss: 0.23326215484179558 acc: 0.9271427917480469 | batch: 100 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.24683781957020984 acc: 0.9253579711914063 | batch: 200 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.25303178894954426 acc: 0.9242862955729166 | batch: 300 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.26150078534847127 acc: 0.9214277648925782 | batch: 400 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.25808289848896676 acc: 0.9222839965820312 | batch: 500 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.256774781003575 acc: 0.92214111328125 | batch: 600 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.25486342299654746 acc: 0.9222432163783482 | batch: 700 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.25308763455097505 acc: 0.9223199462890626 | batch: 800 / 903 |
--------------------------------------------------------------------------------------------
loss: 0.2551166010669355 acc: 0.9220623101128472 | batch: 900 / 903 |
| epoch: 1 | test_acc: 0.9220839142799377  | test_loss: 0.255120088815638  |
```

```
plt.imshow(np.transpose(test[0][0], (1, 2, 0)))
```

```
<matplotlib.image.AxesImage at 0x7f51f838a7a0>
```



Создаем тепловую карту последнего сверточного слоя сети

```
to_pil_image(test[0][0]).save("sign.png")
torch.save(model.cpu().state_dict(), "resnet50.pt")
```

```
model = resnet50(pretrained=False)
model.fc = nn.Linear(2048, len(unique))
model = model.eval()
model.load_state_dict(torch.load("resnet50.pt"))
cam_extractor = SmoothGradCAMpp(model)
# Открываем картинку
img = read_image("sign.png")
# Предобрабатываем картинку для входа нашей нейронной сети
input_tensor = normalize(resize(img, (224, 224)) / 255., [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]
```

```
# Получаем выходные данные на основе модели
out = model(input_tensor.unsqueeze(0))
# Получаем карту активации нейронов последнего слоя модели resnet18
activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)
plt.title("Карта последнего сверточного слоя ResNet50")
```

```
WARNING:root:no value was provided for `target_layer`, thus set to 'layer4'.
<matplotlib.image.AxesImage at 0x7f51f83885b0>
```