NN – Домашнее задание 2

Дедлайн: 23:59, 27 июля

Вам необходимо:

Принять участие в командном соревновании на Kaggle:

В рамках этого соревнования перед вами стоит две задачи:

1. Написать и обучить самописную архитектуру модели на **PyTorch**.
2. Получить высокий **score** на финальном leaderboard.

Требования к `ipynb`:

- В нем должен быть сохранен **output** ячеек (не очищен).
- Внутри ноутбука указать:
    a. Ваши ники на Kaggle и скриншот с итоговой позицией на leaderboard *(важно — после завершения соревнования)*.
    b. Ваши ФИО.

Оценивание соревнования:

ДЗ 2 — макс. 10 баллов

- Корректная реализация и обучение самописной архитектуры: **5 баллов**.

Дополнительно баллы за ноутбук:

1. Логичность и корректность кода и комментариев.
2. Корректная валидация модели на отложенной выборке.
3. Применение дополнительных способов обработки данных.
4. Использование открытых моделей.

Баллы за итоговый score команды:

- Позиция **[1; 3]** на итоговом leaderboard: **+5 баллов**.
- Позиция **[4; 5]** на итоговом leaderboard: **+3 балла**.
- Позиция **[6; 7]** на итоговом leaderboard: **+1 балл**.

Итог:

- Зачет на 4: **7 баллов**.
- Зачет на 5: **8–10 баллов**.

Дополнительно (челлендж «Турнир братства»):

- Топ-3 команд на итоговом leaderboard: **+5 баллов**.

- Побили baseline метрику — можно написать отчет в общий чат по шаблону: **+2 балла**.

Шаблон для +2 баллов:

1. Итоговая позиция в соревновании.
2. Intro.
3. Анализ того, что пробовали.
4. Анализ причин неудач (или потенциальных проблем).
5. Анализ применимости решения в бизнес-процессах.
6. Outro.
7. Тегнуть Арину, чтобы начислила баллы.

Как отправить результат:

1. Ссылка на файл с ноутбуком в облачном хранилище (Google Диск и т.д.).
2. Ссылка на файл с ноутбуком в GitHub.

Важно: ссылки должны быть рабочие и с открытым доступом для скачивания.

Желаем удачи!

Информация об участнике соревнования:

1) Ник на Kaggle - **Vladimir Smirnovve** (участвовал в соревновании самостоятельно, без команды) 2) Скриншот с итоговой позицией на leaderboard (находится по ссылке)

- **9 место**
- **Private Score: 0.60702**
- **Public score: 0.62245** 4) ФИО - **Смирнов Владимир Евгеньевич**

Аннотация

В итоговом рейтинге соревнования я занял (если бы сдавал задание до окончания дедлайна; о причинах поздней сдачи написал в сообщении; для читающих ноутбук в двух словах - брал перерыв от учебы для поступления в магистратуру) **9 место** (по обоим лидербордам) с результатами:

- **Private Score: 0.60702**
- **Public score: 0.62245**

Использованная архитектура

Siamese Neural Network с ResNet18.

Что я пробовал

- **Базовый пайплайн**: Siamese Neural Network с ResNet18 в качестве энкодера.
- Загрузил предобученный ResNet18 с ImageNet, заморозив его веса для быстрого обучения.
- Использовал torch.cat для объединения признаков.

- Обучал простой MLP-классификатор на объединенных признаках.

Что пошло не так / могло пойти не так

- Отсутствие кросс-валидации - для получения стабильного результата я не смог провести полноценную k-fold кросс-валидацию, что могло бы улучшить финальный результат и дать более надежную оценку качества модели.

Применимость в бизнес-процессах

Моё решение может быть использовано в различных бизнес-процессах, связанных с визуальным контентом:

- Контроль качества изображений - автоматическая фильтрация и оценка качества фотографий, например, в e-commerce или на платформах для публикации контента;
- Генерация изображений - сравнение качества сгенерированных изображений с эталонными.

Самописная архитектура находится в самом низу ноутбука!

Библиотеки

```
!pip install --upgrade scikit-learn
!pip install nltk razdel pymorphy3 wordcloud
```

```
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (1.7.1)
Requirement already satisfied: numpy>=1.22.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.8.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.22.0->scikit-learn) (2025.2.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (2022.2.0)
Requirement already satisfied: mkl-service in
```

```
/usr/local/lib/python3.11/dist-packages (from numpy>=1.22.0->scikit-
learn) (2.4.1)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.22.0-
>scikit-learn) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.22.0-
>scikit-learn) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.22.0->scikit-learn) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy>=1.22.0->scikit-learn) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.22.0->scikit-learn) (2024.2.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (3.9.1)
Requirement already satisfied: razdel in
/usr/local/lib/python3.11/dist-packages (0.5.0)
Requirement already satisfied: pymorphy3 in
/usr/local/lib/python3.11/dist-packages (2.0.4)
Requirement already satisfied: wordcloud in
/usr/local/lib/python3.11/dist-packages (1.9.4)
Requirement already satisfied: click in
/usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk) (4.67.1)
Requirement already satisfied: dawg2-python>=0.8.0 in
/usr/local/lib/python3.11/dist-packages (from pymorphy3) (0.9.0)
Requirement already satisfied: pymorphy3-dicts-ru in
/usr/local/lib/python3.11/dist-packages (from pymorphy3)
(2.4.417150.4580142)
Requirement already satisfied: numpy>=1.6.1 in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (11.2.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.11/dist-packages (from wordcloud) (3.7.2)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
```

```
(1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy>=1.6.1->wordcloud) (2025.2.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(2022.2.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy>=1.6.1->wordcloud)
(2.4.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(4.58.4)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(25.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->wordcloud)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib->wordcloud) (1.17.0)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.6.1-
>wordcloud) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl->numpy>=1.6.1-
>wordcloud) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.6.1->wordcloud) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath->numpy>=1.6.1-
>wordcloud) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy>=1.6.1->wordcloud) (2024.2.0)

# --- Системные и общие ---
import os
import math
import random
import re
import warnings
from collections import Counter
from functools import lru_cache

import numpy as np
import pandas as pd
from scipy.sparse import hstack, issparse
from tqdm.auto import tqdm
from tqdm.notebook import tqdm as tqdm_notebook
from bs4 import BeautifulSoup
from IPython.display import display

warnings.filterwarnings('ignore')

# --- Визуализация ---
import matplotlib.pyplot as plt
import seaborn as sns
from graphviz import Digraph
from wordcloud import WordCloud

# --- NLP и лингвистика ---
import nltk
from nltk.corpus import stopwords
from razdel import tokenize as razdel_tokenize
import pymorphy3

# --- PyTorch ---
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torch.utils.tensorboard import SummaryWriter

# --- Torchvision ---
import torchvision
import torchvision.transforms as transforms
from torchvision.models import resnet18, ResNet18_Weights

# --- Hugging Face / Transformers ---
from transformers import (
    AutoTokenizer,
```

```
    AutoModel,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments
)
from datasets import Dataset as HFDataset

# --- Scikit-learn ---
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, confusion_matrix,
classification_report

# --- Константы и настройка среды ---
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
torch.manual_seed(RANDOM_STATE)

# --- Инициализация лидерборда ---
leaderboard = pd.DataFrame(columns=['Метод', 'ROC_AUC_Score'])


import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
import io
from tqdm.notebook import tqdm
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

# Для воспроизводимости результатов
torch.manual_seed(42)
np.random.seed(42)
```

```
2025-08-21 07:21:40.838866: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
```

```
E0000 00:00:1755760900.863747     100 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1755760900.871273     100 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```
# Определяем устройство
if torch.backends.mps.is_available():
    device = torch.device("mps")
    torch.mps.manual_seed(RANDOM_STATE)
elif torch.cuda.is_available():
    device = torch.device("cuda")
    torch.cuda.manual_seed_all(RANDOM_STATE)
else:
    device = torch.device("cpu")

print(f"Используемое устройство: {device}")
```

```
Используемое устройство: cuda
```

```
# Пути к файлам
train_path = '/kaggle/input/teta-nn-2-2025/train.parquet'
test_path = '/kaggle/input/teta-nn-2-2025/test.parquet'

# Загружаем данные из файлов соревнования
if os.path.exists(train_path) and os.path.exists(test_path):
    full_train_df = pd.read_parquet(train_path)
    test_df = pd.read_parquet(test_path)

print("\nПример данных из обучающего набора (df_train):")
display(full_train_df.head())

# Посмотрим на одно из сгенерированных изображений
print("\nПример первого изображения из первой строки:")
display(Image.open(io.BytesIO(full_train_df.iloc[0]['image_1'])))
```

```
Пример данных из обучающего набора (df_train):
```

	image_1	\
0	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...	
1	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...	
2	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...	
3	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...	
4	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...	

	image_2	is_image1_better
0	b'RIFF\x06%\x01\x00WEBPVP8 \xfa$\x01\x00\xb0A\...	1

```
1  b'RIFFL\x99\x01\x00WEBPVP8 @\x99\x01\x00p\xbb\...                    0
2  b'RIFF8@\x02\x00WEBPVP8 ,@\x02\x00\x106\t\x9d\...                    1
3  b'RIFFv\n\x01\x00WEBPVP8 j\n\x01\x00p\xbe\x04\...                    0
4  b'RIFF\xde|\x00\x00WEBPVP8 \xd2|\x00\x00PH\x03...                    0
```

Пример первого изображения из первой строки:

```
class ImagePairDataset(Dataset):
    """
    Класс Dataset для загрузки и предобработки пар изображений.
    """
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

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

    def __getitem__(self, idx):
        # Получаем строку из DataFrame по индексу
        row = self.dataframe.iloc[idx]

        # Декодируем байты в изображения
        # io.BytesIO создает в памяти файлоподобный объект из байтовой
строки
        try:
            image1 =
Image.open(io.BytesIO(row['image_1'])).convert('RGB')
            image2 =
Image.open(io.BytesIO(row['image_2'])).convert('RGB')
        except Exception as e:
            print(f"Ошибка загрузки изображения по индексу {idx}:
{e}")
            # В реальном проекте здесь может быть логика для возврата
"пустого" тензора
            raise

        # Применяем трансформации, если они заданы
        if self.transform:
            image1 = self.transform(image1)
            image2 = self.transform(image2)

        # Метка: 1.0, если первое изображение лучше, иначе 0.0
        label = torch.tensor([float(row['is_image1_better'])],
dtype=torch.float32)

        return image1, image2, label

class SiameseComparator(nn.Module):
    def __init__(self, freeze_encoder=True):
        super(SiameseComparator, self).__init__()

        # Загружаем предобученный ResNet18
        self.encoder =
models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)

        num_features = self.encoder.fc.in_features
```

```
        # Fine-tuning: размораживаем часть слоев
        if not freeze_encoder:
            # Размораживаем только последний блок ResNet
            for param in self.encoder.layer4.parameters():
                param.requires_grad = True
            for param in self.encoder.fc.parameters():
                param.requires_grad = True

        # Заменяем последний классификационный слой
        self.encoder.fc = nn.Identity()

        # Создаем новую "голову" для классификации с учетом diff и mul
        # Входной размер: num_features * 4 (features1, features2,
diff, mul)
        self.classifier_head = nn.Sequential(
            nn.Linear(num_features * 4, 1024),
            nn.BatchNorm1d(1024),
            nn.GELU(),
            nn.Dropout(0.5),
            nn.Linear(1024, 256),
            nn.BatchNorm1d(256),
            nn.GELU(),
            nn.Dropout(0.4),
            nn.Linear(256, 1)
        )

    def forward(self, img1, img2):
        # Пропускаем каждое изображение через энкодер
        features1 = self.encoder(img1)
        features2 = self.encoder(img2)

        # Вычисляем разность и произведение признаков
        # Данную идею заимствовал у коллег по курсу - не сказать, что
мне дало значительный прирост
        diff_features = features1 - features2
        mul_features = features1 * features2

        # Конкатенируем все четыре вектора
        combined_features = torch.cat((features1, features2,
diff_features, mul_features), dim=1)

        output = self.classifier_head(combined_features)
        return output

def train_model(model, dataloader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0
```

```
    for images1, images2, labels in tqdm(dataloader, desc="Обучение"):
        images1, images2, labels = images1.to(device),
images2.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(images1, images2)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images1.size(0)
        preds = torch.sigmoid(outputs) > 0.5
        correct_predictions += (preds == labels).sum().item()
        total_samples += labels.size(0)

    epoch_loss = running_loss / total_samples
    epoch_acc = correct_predictions / total_samples
    return epoch_loss, epoch_acc

def evaluate_model(model, dataloader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0


    all_preds_logits = []
    all_labels = []

    with torch.no_grad():
        for images1, images2, labels in tqdm(dataloader,
desc="Оценка"):
            images1, images2, labels = images1.to(device),
images2.to(device), labels.to(device)

            outputs = model(images1, images2)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * images1.size(0)
            preds = torch.sigmoid(outputs) > 0.5
            correct_predictions += (preds == labels).sum().item()
            total_samples += labels.size(0)

            all_preds_logits.append(outputs.cpu())
            all_labels.append(labels.cpu())

    epoch_loss = running_loss / total_samples
```

```
    epoch_acc = correct_predictions / total_samples

    all_preds_logits = torch.cat(all_preds_logits)
    all_labels = torch.cat(all_labels)

    return epoch_loss, epoch_acc, all_preds_logits, all_labels

# Трансформации для тренировочного набора
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2, hue=0.1),
    transforms.GaussianBlur(kernel_size=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
])

# Трансформации для валидационного/тестового набора (без аугментации)
val_test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
])

# --- 2. Разделение данных и создание DataLoaders ---
# Разбиваем исходный train на обучающую и валидационную выборки
train_df, val_df = train_test_split(
    full_train_df,
    test_size=0.2,
    random_state=RANDOM_STATE,
    stratify=full_train_df['is_image1_better']
)

train_df = train_df.reset_index(drop=True)
val_df = val_df.reset_index(drop=True)

test_df = pd.read_parquet(test_path)

# Создаем экземпляры Dataset
train_dataset = ImagePairDataset(train_df, transform=train_transforms)
val_dataset = ImagePairDataset(val_df, transform=val_test_transforms)


# --- Подготовка данных ---

test_df = pd.read_parquet(test_path)
```

```
class TestImagePairDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

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

    def __getitem__(self, idx):
        row = self.dataframe.iloc[idx]

        # Декодируем байты в изображения
        image1 = Image.open(io.BytesIO(row['image_1'])).convert('RGB')
        image2 = Image.open(io.BytesIO(row['image_2'])).convert('RGB')

        # Применяем трансформации
        if self.transform:
            image1 = self.transform(image1)
            image2 = self.transform(image2)

        return image1, image2

# Гиперпараметры
BATCH_SIZE = 16
LEARNING_RATE = 1e-5
NUM_EPOCHS = 30

test_dataset = TestImagePairDataset(test_df,
transform=val_test_transforms)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=0)

# 3. Инициализируем модель, функцию потерь и оптимизатор
model = SiameseComparator().to(device)

criterion = nn.BCEWithLogitsLoss()

optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

Downloading: "https://download.pytorch.org/models/resnet18-
f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-
f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 167MB/s]
```

```
history = {'train_loss': [], 'train_acc': []}

for epoch in range(NUM_EPOCHS):
    print(f"\n--- Эпоха {epoch + 1}/{NUM_EPOCHS} ---")

    train_loss, train_acc = train_model(model, train_loader,
criterion, optimizer, device)

    print(f"Эпоха {epoch + 1} | Потери на обучении: {train_loss:.4f} |
Точность на обучении: {train_acc:.4f}")

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
```

```
--- Эпоха 1/30 ---
```


```
{"model_id":"f9b4dfc13a4a4e0db6769e99e7bf4f1a","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 1 | Потери на обучении: 0.6908 | Точность на обучении: 0.5518

--- Эпоха 2/30 ---
```


```
{"model_id":"22e86c603d224a40b95ee62e7442ec3f","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 2 | Потери на обучении: 0.6622 | Точность на обучении: 0.6007

--- Эпоха 3/30 ---
```


```
{"model_id":"a3cdeadc8c07498a909a49d310596bd6","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 3 | Потери на обучении: 0.6474 | Точность на обучении: 0.6250

--- Эпоха 4/30 ---
```


```
{"model_id":"def5b2011d144fe7bea43b97fe88e2fc","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 4 | Потери на обучении: 0.6316 | Точность на обучении: 0.6431

--- Эпоха 5/30 ---
```


```
{"model_id":"10e3341147654cde80b281cdbafd8ba7","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 5 | Потери на обучении: 0.6186 | Точность на обучении: 0.6622

--- Эпоха 6/30 ---
```

{"model_id":"83c2f6e409e341feac752b25a60ba2e2","version_major":2,"version_minor":0}

```
Эпоха 6 | Потери на обучении: 0.6138 | Точность на обучении: 0.6655

--- Эпоха 7/30 ---
```

{"model_id":"65143046e7a94a6e90542136f393d0af","version_major":2,"version_minor":0}

```
Эпоха 7 | Потери на обучении: 0.6047 | Точность на обучении: 0.6792

--- Эпоха 8/30 ---
```

{"model_id":"7d82e956912a4f2998789867f18587bd","version_major":2,"version_minor":0}

```
Эпоха 8 | Потери на обучении: 0.5904 | Точность на обучении: 0.6861

--- Эпоха 9/30 ---
```

{"model_id":"29d2af8ae81243cd80eaf795b144fc1e","version_major":2,"version_minor":0}

```
Эпоха 9 | Потери на обучении: 0.5848 | Точность на обучении: 0.6907

--- Эпоха 10/30 ---
```

{"model_id":"c21a0286210f46ee901a7b90148767a8","version_major":2,"version_minor":0}

```
Эпоха 10 | Потери на обучении: 0.5719 | Точность на обучении: 0.7002

--- Эпоха 11/30 ---
```

{"model_id":"5d5085816052419d834e024999119fc5","version_major":2,"version_minor":0}

```
Эпоха 11 | Потери на обучении: 0.5659 | Точность на обучении: 0.7118

--- Эпоха 12/30 ---
```

{"model_id":"81658088dbfa46d8a9806df86385fb3b","version_major":2,"version_minor":0}

```
Эпоха 12 | Потери на обучении: 0.5556 | Точность на обучении: 0.7130

--- Эпоха 13/30 ---
```

{"model_id":"9f8ca0f6b55b4049bd37b9efeb7526e3","version_major":2,"version_minor":0}

```
Эпоха 13 | Потери на обучении: 0.5423 | Точность на обучении: 0.7252

--- Эпоха 14/30 ---

{"model_id":"21a0df61117048bb8c01a84472066841","version_major":2,"vers
ion_minor":0}

Эпоха 14 | Потери на обучении: 0.5252 | Точность на обучении: 0.7329

--- Эпоха 15/30 ---

{"model_id":"7aa547ef00974cdbb30b7b8a8fd8cbfb","version_major":2,"vers
ion_minor":0}

Эпоха 15 | Потери на обучении: 0.5212 | Точность на обучении: 0.7430

--- Эпоха 16/30 ---

{"model_id":"e7f7f675f85441a2a2debbdf2ed273a1","version_major":2,"vers
ion_minor":0}

Эпоха 16 | Потери на обучении: 0.4989 | Точность на обучении: 0.7556

--- Эпоха 17/30 ---

{"model_id":"afe8ca131983432691599 77f828137e7","version_major":2,"vers
ion_minor":0}

Эпоха 17 | Потери на обучении: 0.4890 | Точность на обучении: 0.7661

--- Эпоха 18/30 ---

{"model_id":"b34c4ed6f85c4d54a0349f128a5c7f90","version_major":2,"vers
ion_minor":0}

Эпоха 18 | Потери на обучении: 0.4730 | Точность на обучении: 0.7688

--- Эпоха 19/30 ---

{"model_id":"fc8871ef83194588869cf34e158bd06c","version_major":2,"vers
ion_minor":0}

Эпоха 19 | Потери на обучении: 0.4566 | Точность на обучении: 0.7816

--- Эпоха 20/30 ---

{"model_id":"c20122c66359488e9ec7201503ee7fac","version_major":2,"vers
ion_minor":0}

Эпоха 20 | Потери на обучении: 0.4454 | Точность на обучении: 0.7905

--- Эпоха 21/30 ---
```

{"model_id":"245a19ce775345c692b7fa88058e9b68","version_major":2,"version_minor":0}

```
Эпоха 21 | Потери на обучении: 0.4239 | Точность на обучении: 0.8045

--- Эпоха 22/30 ---
```

{"model_id":"bb65d4e930d74fdf92a46dcf5234e2f1","version_major":2,"version_minor":0}

```
Эпоха 22 | Потери на обучении: 0.4117 | Точность на обучении: 0.8098

--- Эпоха 23/30 ---
```

{"model_id":"5d7aa89f9d5947898fdd2856de70b707","version_major":2,"version_minor":0}

```
Эпоха 23 | Потери на обучении: 0.3916 | Точность на обучении: 0.8193

--- Эпоха 24/30 ---
```

{"model_id":"0e4f450f00574c9fbd8e49817c07f9d4","version_major":2,"version_minor":0}

```
Эпоха 24 | Потери на обучении: 0.3762 | Точность на обучении: 0.8319

--- Эпоха 25/30 ---
```

{"model_id":"ff9774479223429f97488880e3e98436","version_major":2,"version_minor":0}

```
Эпоха 25 | Потери на обучении: 0.3593 | Точность на обучении: 0.8396

--- Эпоха 26/30 ---
```

{"model_id":"dae69de448054053b460bd5fd466dd8f","version_major":2,"version_minor":0}

```
Эпоха 26 | Потери на обучении: 0.3622 | Точность на обучении: 0.8396

--- Эпоха 27/30 ---
```

{"model_id":"94ac168e52794c858da4505b8f0a7c87","version_major":2,"version_minor":0}

```
Эпоха 27 | Потери на обучении: 0.3378 | Точность на обучении: 0.8536

--- Эпоха 28/30 ---
```

{"model_id":"38a798793bb74aba9d24831c01ec87d0","version_major":2,"version_minor":0}

```
Эпоха 28 | Потери на обучении: 0.3304 | Точность на обучении: 0.8532

--- Эпоха 29/30 ---

{"model_id":"74ee477a9013462f8630fd2668bf1b5e","version_major":2,"vers
ion_minor":0}

Эпоха 29 | Потери на обучении: 0.3254 | Точность на обучении: 0.8575

--- Эпоха 30/30 ---

{"model_id":"c371423cf9024074869ef91486c8abbb","version_major":2,"vers
ion_minor":0}

Эпоха 30 | Потери на обучении: 0.3180 | Точность на обучении: 0.8665

print("\n--- Генерация предсказаний на тестовом наборе ---")

model.eval()

all_preds_probs = []

with torch.no_grad():
    for images1, images2 in tqdm(test_loader, desc="Генерация
предсказаний"):
        images1 = images1.to(device)
        images2 = images2.to(device)

        outputs = model(images1, images2)

        # Преобразуем логиты в вероятности
        probs = torch.sigmoid(outputs)

        all_preds_probs.append(probs.cpu())

# Объединяем предсказания в один массив
submission_preds = torch.cat(all_preds_probs).numpy().flatten()


--- Генерация предсказаний на тестовом наборе ---

{"model_id":"db583adb0fd84da7bf2cdf3a6ee47827","version_major":2,"vers
ion_minor":0}

# submission
print("Создание DataFrame для submission...")

submission_df = pd.DataFrame({
    'index': test_df.index,
    'is_image1_better': submission_preds
})
```

```
submission_df.to_csv('submission.csv', index=False)
print("Файл submission.csv успешно создан!")
```

```
Создание DataFrame для submission...
Файл submission.csv успешно создан!
```

Архитектура

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()

        # Свёрточный блок 1
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        # Свёрточный блок 2
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        # Свёрточный блок 3
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        # Полносвязные слои (MLP)
        self.fc = nn.Sequential(
            nn.Linear(64 * 28 * 28, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 1)
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)

        x = x.view(x.size(0), -1)
```

```
        x = self.fc(x)
        return x

class CustomSiameseModel(nn.Module):
    def __init__(self):
        super(CustomSiameseModel, self).__init__()

        # Используем нашу самописную CNN в качестве энкодера
        self.encoder = SimpleCNN()

        # Важно! Нам нужно определить размерность выходного вектора
нашего энкодера
        # Выходной слой SimpleCNN имеет размерность 512, но на конце у
нас уже есть
        # выходной слой на 1 нейрон. Поэтому мы должны изменить
архитектуру.
        # Более правильный подход - это получить эмбеддинги перед
финальным слоем

        num_features = 512

        # Создаем новую "голову" для классификации объединенных
признаков
        self.classifier_head = nn.Sequential(
            nn.Linear(num_features * 2, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 1)
        )

    def forward(self, img1, img2):
        features1 = self.encoder(img1)
        features2 = self.encoder(img2)

        combined_features = torch.cat((features1, features2), dim=1)

        output = self.classifier_head(combined_features)
        return output

# Трансформации
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(15),
    transforms.ToTensor(),
])

val_test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
```

```
])

# Инициализация Dataset и DataLoader
train_dataset = ImagePairDataset(train_df, transform=train_transforms)
val_dataset = ImagePairDataset(val_df, transform=val_test_transforms)
test_dataset = TestImagePairDataset(test_df,
transform=val_test_transforms)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

# Гиперпараметры
BATCH_SIZE = 16
LEARNING_RATE = 1e-5
NUM_EPOCHS = 20

best_val_auc = 0.0
best_model_path = 'best_model.pth'

model = SiameseComparator().to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

# Цикл обучения
for epoch in range(NUM_EPOCHS):
    train_loss, train_acc = train_model(model, train_loader,
criterion, optimizer, device)

    val_loss, val_acc, val_preds_logits, val_labels =
evaluate_model(model, val_loader, criterion, device)

    val_preds_probs = torch.sigmoid(val_preds_logits).numpy()
    val_roc_auc = roc_auc_score(val_labels.numpy(), val_preds_probs)

    print(f"Эпоха {epoch + 1} | Тренировка: Потери={train_loss:.4f} |
Точность={train_acc:.4f}")
    print(f"Эпоха {epoch + 1} | Валидация: Потери={val_loss:.4f} |
Точность={val_acc:.4f} | ROC-AUC={val_roc_auc:.4f}")

    if val_roc_auc > best_val_auc:
        best_val_auc = val_roc_auc
        torch.save(model.state_dict(), best_model_path)
        print(f"Лучшая модель сохранена с ROC-AUC:
{best_val_auc:.4f}")
```

{"model_id":"0db96607619c4d3d8f57b8ffb2b3ca77","version_major":2,"version_minor":0}

{"model_id":"326062dd3059488d90b0c302c405be33","version_major":2,"version_minor":0}

```
Эпоха 1 | Тренировка: Потери=0.6732 | Точность=0.5890
Эпоха 1 | Валидация: Потери=0.6585 | Точность=0.6200 | ROC-AUC=0.5654
Лучшая модель сохранена с ROC-AUC: 0.5654
```

{"model_id":"56bd387badb849db93e3801825cbb3ad","version_major":2,"version_minor":0}

{"model_id":"c455a5c0ac4b4d4692581ee266eab3e7","version_major":2,"version_minor":0}

```
Эпоха 2 | Тренировка: Потери=0.6474 | Точность=0.6378
Эпоха 2 | Валидация: Потери=0.6508 | Точность=0.6240 | ROC-AUC=0.5911
Лучшая модель сохранена с ROC-AUC: 0.5911
```

{"model_id":"333d100fd1c94a828685c46508836c42","version_major":2,"version_minor":0}

{"model_id":"0d52868c91894c0483e36afc159521c6","version_major":2,"version_minor":0}

```
Эпоха 3 | Тренировка: Потери=0.6295 | Точность=0.6449
Эпоха 3 | Валидация: Потери=0.6460 | Точность=0.6338 | ROC-AUC=0.6051
Лучшая модель сохранена с ROC-AUC: 0.6051
```

{"model_id":"5d879249a99c4b7dbe6c7d88627a8f72","version_major":2,"version_minor":0}

{"model_id":"bb156e06e58a4f1eb73385bb7c5ec25b","version_major":2,"version_minor":0}

```
Эпоха 4 | Тренировка: Потери=0.6146 | Точность=0.6650
Эпоха 4 | Валидация: Потери=0.6456 | Точность=0.6263 | ROC-AUC=0.6094
Лучшая модель сохранена с ROC-AUC: 0.6094
```

{"model_id":"64d2ffa3be304fb9a665e06375afc51a","version_major":2,"version_minor":0}

{"model_id":"1dd3ac3a7acc47e9aeeb3010ef5c1209","version_major":2,"version_minor":0}

```
Эпоха 5 | Тренировка: Потери=0.6026 | Точность=0.6748
Эпоха 5 | Валидация: Потери=0.6432 | Точность=0.6320 | ROC-AUC=0.6211
Лучшая модель сохранена с ROC-AUC: 0.6211
```

{"model_id":"f527d8c48dc4410f95eb656b54460e80","version_major":2,"version_minor":0}

{"model_id":"5554a12b50a74c4e8490342399c3cadd","version_major":2,"version_minor":0}

```
Эпоха 6 | Тренировка: Потери=0.5851 | Точность=0.6893
Эпоха 6 | Валидация: Потери=0.6414 | Точность=0.6435 | ROC-AUC=0.6281
Лучшая модель сохранена с ROC-AUC: 0.6281
```

{"model_id":"b600ab0b58ed4953a24dcf710b2bc1ca","version_major":2,"version_minor":0}

{"model_id":"715487612094407dba719e02478938af","version_major":2,"version_minor":0}

```
Эпоха 7 | Тренировка: Потери=0.5650 | Точность=0.7097
Эпоха 7 | Валидация: Потери=0.6462 | Точность=0.6406 | ROC-AUC=0.6311
Лучшая модель сохранена с ROC-AUC: 0.6311
```

{"model_id":"adbc595de0a94cee93225a709afb642c","version_major":2,"version_minor":0}

{"model_id":"586e7889391542f3a2b374f8ef93fa7b","version_major":2,"version_minor":0}

```
Эпоха 8 | Тренировка: Потери=0.5559 | Точность=0.7120
Эпоха 8 | Валидация: Потери=0.6521 | Точность=0.6389 | ROC-AUC=0.6398
Лучшая модель сохранена с ROC-AUC: 0.6398
```

{"model_id":"e7e859131f9247a785fb1a424a6ce406","version_major":2,"version_minor":0}

{"model_id":"190b343515444b239d17519b10b1a03c","version_major":2,"version_minor":0}

```
Эпоха 9 | Тренировка: Потери=0.5377 | Точность=0.7216
Эпоха 9 | Валидация: Потери=0.6578 | Точность=0.6326 | ROC-AUC=0.6401
Лучшая модель сохранена с ROC-AUC: 0.6401
```

{"model_id":"f9acd85c076d49a684a63595c50896f3","version_major":2,"version_minor":0}

{"model_id":"eb19a9319d014659b77b4de9e1d835fe","version_major":2,"version_minor":0}

```
Эпоха 10 | Тренировка: Потери=0.5149 | Точность=0.7511
Эпоха 10 | Валидация: Потери=0.6796 | Точность=0.6096 | ROC-AUC=0.6387
```

{"model_id":"75f7a03783bf49ab876331858ce51366","version_major":2,"version_minor":0}

{"model_id":"45a4cf7caad94ffa83f2b2f75f8b7991","version_major":2,"version_minor":0}

```
Эпоха 11 | Тренировка: Потери=0.4909 | Точность=0.7665
Эпоха 11 | Валидация: Потери=0.6802 | Точность=0.6148 | ROC-AUC=0.6362
```


```
{"model_id":"4c6f59e2246a4ce5ae8b4204e19af10d","version_major":2,"vers
ion_minor":0}

{"model_id":"5da151a99bed497cab04be225eae6678","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 12 | Тренировка: Потери=0.4615 | Точность=0.7830
Эпоха 12 | Валидация: Потери=0.6973 | Точность=0.6183 | ROC-AUC=0.6346
```


```
{"model_id":"d917aa3e4e044c66a3a18aead6a066d1","version_major":2,"vers
ion_minor":0}

{"model_id":"5303d4a5b5b047348d0608fef083c9ea","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 13 | Тренировка: Потери=0.4296 | Точность=0.8078
Эпоха 13 | Валидация: Потери=0.7516 | Точность=0.6010 | ROC-AUC=0.6286
```


```
{"model_id":"0d673af3f1884022b53d3a68a1294d29","version_major":2,"vers
ion_minor":0}

{"model_id":"4b687393cb724b72834a11a8350c5c91","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 14 | Тренировка: Потери=0.4139 | Точность=0.8096
Эпоха 14 | Валидация: Потери=0.7732 | Точность=0.5970 | ROC-AUC=0.6224
```


```
{"model_id":"5750fa016a3b489dbd524e4326f66d20","version_major":2,"vers
ion_minor":0}

{"model_id":"c757c05b12d04b409e8265c1d265bb1e","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 15 | Тренировка: Потери=0.3699 | Точность=0.8352
Эпоха 15 | Валидация: Потери=0.8148 | Точность=0.5999 | ROC-AUC=0.6249
```


```
{"model_id":"bf87831f20524e4a87fc25d063c042c4","version_major":2,"vers
ion_minor":0}

{"model_id":"ad00bb19ac7b4099b6cfc2104d2affee","version_major":2,"vers
ion_minor":0}
```


```
Эпоха 16 | Тренировка: Потери=0.3480 | Точность=0.8525
Эпоха 16 | Валидация: Потери=0.8403 | Точность=0.5947 | ROC-AUC=0.6208
```


```
{"model_id":"c89d45115a6a4b198ecc8cdbb33c4141","version_major":2,"vers
ion_minor":0}

{"model_id":"d78def059d27416cabdaa049430bcfd2","version_major":2,"vers
ion_minor":0}
```

```
Эпоха 17 | Тренировка: Потери=0.3257 | Точность=0.8611
Эпоха 17 | Валидация: Потери=0.9111 | Точность=0.5798 | ROC-AUC=0.6202

{"model_id":"bc7cc5ee4d2f42008fc804cbf473ccc3","version_major":2,"vers
ion_minor":0}

{"model_id":"a72871d54c6e44b1a44c82d5187e5085","version_major":2,"vers
ion_minor":0}

Эпоха 18 | Тренировка: Потери=0.3088 | Точность=0.8733
Эпоха 18 | Валидация: Потери=0.9088 | Точность=0.5913 | ROC-AUC=0.6144

{"model_id":"9fef2052bed84c26b458de5b7fd4b969","version_major":2,"vers
ion_minor":0}

{"model_id":"7833ce6d29974f599373d8c4054e9b37","version_major":2,"vers
ion_minor":0}

Эпоха 19 | Тренировка: Потери=0.2951 | Точность=0.8789
Эпоха 19 | Валидация: Потери=0.9357 | Точность=0.6039 | ROC-AUC=0.6182

{"model_id":"1a8626e763494219a4ea64b6205d47a0","version_major":2,"vers
ion_minor":0}

{"model_id":"0f956639c3c446ba98bf4132e73b0334","version_major":2,"vers
ion_minor":0}

Эпоха 20 | Тренировка: Потери=0.2762 | Точность=0.8911
Эпоха 20 | Валидация: Потери=0.9993 | Точность=0.5896 | ROC-AUC=0.6133
```

```python
def create_test_predictions(model, dataloader, device):
    """
    Генерирует предсказания для тестового набора данных.

    Args:
        model (torch.nn.Module): Обученная модель.
        dataloader (DataLoader): DataLoader для тестового набора.
        device (torch.device): Устройство (CPU, CUDA, MPS).

    Returns:
        numpy.ndarray: Массив с вероятностями предсказаний.
    """

    model.eval()
    all_preds_logits = []

    with torch.no_grad():
        for images1, images2 in tqdm(dataloader, desc="Генерация
предсказаний"):
            images1 = images1.to(device)
            images2 = images2.to(device)
```

```
            outputs = model(images1, images2)
            all_preds_logits.append(outputs.cpu())

    all_preds_logits = torch.cat(all_preds_logits)
    all_preds_probs =
torch.sigmoid(all_preds_logits).numpy().flatten()

    return all_preds_probs

best_model = SiameseComparator().to(device)
best_model.load_state_dict(torch.load('best_model.pth'))

submission_preds = create_test_predictions(best_model, test_loader,
device)

# submission
submission_df = pd.DataFrame({
    'index': test_df.index,
    'is_image1_better': submission_preds
})

submission_df.to_csv('submission_self_made.csv', index=False)
print("Файл submission.csv успешно создан!")
```

{"model_id":"f0d12b969c1e4d19adbd3238f40d4029","version_major":2,"version_minor":0}

```
Файл submission.csv успешно создан!
```