СОДЕРЖАНИЕ

РЕЗЮМЕ

Курсовой проект: 43 стр., 5 источников

ПОСТРОЕНИЕ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ РЕШЕНИЯ ЗАДАЧ КЛАССИФИКАЦИИ С ПОМОЩЬЮ БИБЛИОТЕКИ PYTORCH.

КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ОБУЧЕНИЕ НЕЙРОСЕТИ, СОЗДАНИЕ ВЕБ-ПРИЛОЖЕНИЯ.

Объект исследования: нейронные сети для распознавания и классификации.

Предмет исследования: особенности создания, обучения нейронной сети с помощью библиотеки PyTorch.

Цель работы: изучить способы создания и обучения нейронной сети с помощью библиотеки PyTorch, а также практического применения построенной модели по средством создания веб-приложения.

Методы исследования: графический, наблюдения, сравнительного анализа.

Исследования и разработки: изучены способы создания и обучения нейронной сети с помощью библиотеки PyTorch, а также практического применения построенной модели по средством создания веб-приложения.

SUMMARY

Course project: 43 pages, 5 sources.

CONSTRUCTION OF CONVOLUTIONAL NEURAL NETWORKS FOR SOLVING CLASSIFICATION PROBLEMS USING THE PYTORCH LIBRARY.

IMAGE CLASSIFICATION, CREATION AND TRAINING OF A NEURAL NETWORK, CREATION OF A WEB APPLICATION.

Object of study: neural networks for recognition and classification.

Subject of study: features of creating and training a neural network using the PyTorch library.

The purpose of the work: to study how to create and train a neural network using the PyTorch library, as well as the practical application of the built model by creating a web application.

Research methods: graphic, observation, comparative analysis.

R&D: Learned how to build and train a neural network using the PyTorch library, as well as the practical application of the constructed model by creating a web application.

ВВЕДЕНИЕ

Сверточные нейронные сети (CNN) являются мощным инструментом для анализа и распознавания изображений, видео и звука. Они могут быть применены во множестве сфер человеческой жизни, где требуется классификация объектов или процессов по определенным критериям. Рассмотрим несколько таких сфер.

Медицина: CNN используются для диагностики различных заболеваний по рентгенограммам, МРТ, УЗИ, сканам и другим медицинским изображениям. Также CNN могут быть использованы для распознавания микроскопических изображений клеток и тканей, прогнозирования риска заболевания у пациентов и многих других медицинских задач.

Автомобильная промышленность: CNN применяются для распознавания дорожных знаков, разметки дорог и пешеходов. Это помогает автомобильному транспорту и автопилотам уменьшать количество аварий на дорогах.

Финансы: CNN необходимы для анализа финансовых данных, таких как совокупный доход, расходы, активы и т.д. CNN могут также использоваться для обнаружения финансовых мошенничеств.

Робототехника: CNN незаменимы в задачах распознавания и классификации объектов для роботов. Они также могут быть использованы для управления роботами, например, для их навигации.

При всех равных мной было отдано предпочтение сфере **медицины**, так как выбор медицинского направления для реализации сверточных нейронных сетей и изучения направления глубокого обучения имеет несколько преимуществ:

- Во-первых, здесь доступен большой объем данных в виде клинических данных, медицинских изображений, электрокардиограмм, медицинских записей и т.д., которые можно использовать для обучения нейронных сетей.

- Во-вторых, медицинские данные содержат сложные взаимодействия, зависимости и вариации между различными переменными, что позволяет использовать высокоуровневые фреймворки глубокого обучения для их моделирования.
- В-третьих, использование сверточных нейронных сетей для анализа медицинских изображений и сигналов позволяет получать высокую точность диагностики, улучшает медицинскую практику и уменьшает ошибки количество и вероятность ошибок, связанных с человеческим фактором.
- И наконец, существует потребность в разработке инновационных медицинских приложений, которые могут помочь облегчить работу врачей, помочь в диагностике и лечении пациентов, улучшить качество жизни и уменьшить расходы на здравоохранение.

Таким образом, выбор медицинского направления для реализации сверточных нейронных сетей и изучения глубокого обучения открывает двери для инновационных медицинских приложений и облегчения работы врачей в лечении пациентов.

1.1. Выбор данных и описание цели исследования.

Итак, определившись со сферой применения, встаёт вопрос поиска и выбора необходимых данных для последующего обучения. И тут же необходимо учесть тот факт, что одним из наиболее важных аспектов при выборе данных для обучения нейронных сетей в медицинской сфере является выбор конкретной задачи. Другими словами, в медицинской сфере решение некоторых задач с использованием нейронных сетей будет более оправдано и полезно, а значит, выбор одной или нескольких задач из этого списка будет более оправданным действием. К примеру, в задачах медицинского обслуживания, таких как определение диагноза, описание болезни и

медицинской истории пациента, нейронные сети могут оказаться чрезвычайно полезными. А в других задачах, например, лечении болезней, нейронные сети могут внести лишь небольшой вклад, так как в этих случаях эффективность лечения зависит от подбора и применения правильных медицинских препаратов, и здесь роль человека все равно остаётся решающей. Считаю необходимым сосредоточиться на выполнении задачи по постановке правильного диагноза пациенту на основе полученных входных данных.

Учтя множество параметров для выбора необходимых данных, выдвинем к данным следующие требования:

1) Данные должны соответствовать поставленной задаче: по ним должно быть возможно постановить корректный диагноз.
2) Необходимый формат. Данные должны быть в формате изображений: рентгенограмм, МРТ, УЗИ, сканов и других медицинских изображений в силу специфики данной курсовой работы. Для построения сверточных нейронных сетей с целью решения задач классификации (объект исследования данной курсовой работы) такой тип данных, как изображения, идеально подходит.
3) Данных должно быть достаточное количество. Желательно просто много. Чем больше, тем лучше, так как при необходимости часть данных можно вовсе не использовать. Их можно удалить на этапе подготовки данных для обучения. Но вот недостаточное количество данных принесёт больше проблем: от недообучения (мало данных) до переобучения (слишком сложная модель для такого сравнительно малого количества данных)
4) Данные должны быть чистыми, качественными и точными, чтобы избежать ошибок в классификации. Например, медицинские изображения должны быть с высоким разрешением и правильно настроенными с медицинской точки зрения.

5) Доступность. Данные должны быть доступны для всех, кто заинтересован в обучении алгоритмов, чтобы повышать качество моделей и облегчение медицинской практики.

Для поиска качественных наборов данных, удовлетворяющих критериям, описанным выше, будем использовать платформу Kaggle. На данной платформе выбор пал на датасет Chest X-Ray Images (Pneumonia). Набор данных организован в 3 папки (train, test, val) и содержит подпапки для каждой категории изображений (пневмония/норма). Имеется 5863 рентгеновских изображения (JPEG) и 2 категории (пневмония/норма). Данный датасет удовлетворяет всем выше перечисленным требованиям, так как данные удовлетворяют объекту и предмету исследования, данные высокого уровня качества и находятся в необходимом формате, данных необходимое количество для полноценного обучения, и они общедоступны для всех желающих.

Теперь перейдём к постановке целей курсовой работы. Необходимо изучить способы создания и обучения сверточной нейронной сети для решения задач классификации с помощью библиотеки PyTorch, и как мы уже решили, в области медицины, на чем будет сконцентрировано основное внимание как на первичной цели. Опционально мы также разберём один из возможных вариантов практического применения построенной и обученной модели посредством создания веб-приложения с помощью Flask, CSS, HTML, JS.

1.2 Очистка и организация данных.

Теперь приступим к этапу загрузки, очистки и организации данных. После загрузки на этапе организации данных определим количество классов в датасете, количество изображений в каждой из выборок: тренировочной, валидационной, тестовой. Для выполнения поставленных задач будем использовать Kaggle Notebook - онлайн-редактор кода, встроенный в платформу Kaggle. Он идеально подходит для экспериментов над данными, так как нет необходимости в загрузке всех библиотек и датасета локально на компьютер. Также Kaggle Notebook предоставляет хоть и ограниченные, но все же внушительные мощности по CPU и памяти, поэтому является лучшим решением для датасетов Kaggle (в Colab меньше предоставляемой памяти и ее может не хватить в момент обучения модели, также сложнее импортировать датасет с Kaggle).

Для начала импортируем все необходимые для глубокого обучения библиотеки, такие как numpy, torch, torchvision, pandas, pathlib, matplotlib для построения графиков и другие:

Листинг 1:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dataset
import torchvision.models as models
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
```

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import random
from PIL import Image
import requests
from typing import Tuple, Dict, List
from pathlib import Path
from tqdm.auto import tqdm
```

Далее для удобства зададим переменные-константы, которые будут неизменны. А именно количество эпох обучения, скорость обучения, размер батча данных, размер изображения, которое будет поставляться непосредственно на вход модели, значение параметра начального состояния генератора случайных чисел, путь к данным и количество рабочих процессов - указывает на количество параллельных процессов, которые будут использоваться для извлечения данных – ускоряет процесс обучения.

Листинг 2:

```
EPOCHS = 50
LEARNING_RATE = 0.001
BATCH_SIZE = 128
INPUT_SHAPE = (60, 60)
SEED = 100
DATA_PATH_2 = Path("/kaggle/input/chest-xray-pneumonia/chest_xray")
NUM_WORKERS = os.cpu_count()
```

Укажем конкретный путь до каждой выборки. Но и здесь есть нюанс. В папке с валидационной выборкой всего лишь 16 изображений, и их мы оставим для тестирования готового приложения. Поэтому укажем конкретный путь до тестовой и тренировочной выборки.

Листинг 3:

Вввод:

```
TRAIN_PATH = DATA_PATH_2 / "train/"
ELECTED_PATH = DATA_PATH_2 / "val/"
TEST_PATH = DATA_PATH_2 / "test/"
print(f"Path to train part of dataset: {TRAIN_PATH}.\n")
print(f"Path to elected part of dataset: {ELECTED_PATH}.\n")
print(f"Path to test part of dataset: {TEST_PATH}\n")
```

Вывод:

```
Path to train part of dataset: /kaggle/input/chest-xray-pneumonia/chest_xray/train.
Path to elected part of dataset: /kaggle/input/chest-xray-pneumonia/chest_xray/val.
Path to test part of dataset: /kaggle/input/chest-xray-pneumonia/chest_xray/test
```

Теперь нам необходимо задать последовательность преобразований, используя объекты класса Compose из библиотеки torchvision.transforms. По умолчанию наши изображения имеют 3 канала (RGB) и для упрощения построения модели сделаем их черно-белыми (1 канал). Также уменьшим размер изображения до размера 60 на 60 пикселей. Также к выборкам применим аугментацию данных с помощью transforms.TrivialAugmentWide() для улучшения обобщающей способности модели, заставляя её учитывать различные варианты представления данных и увеличивая размер обучающего набора данных. Аугментация выполняется с использованием алгоритма TrivialAugment, который случайным образом изменяет яркость, контраст, насыщенность и цвет изображений. Параметр num_magnitude_bins отвечает за степень интенсивности применения случайных изменений данных (чем больше значение параметра, тем интенсивнее аугментация). Аугментацию целесообразно применять исключительно к тренировочной выборке. transforms.ToTensor() преобразует входное изображение или видео из формата PIL (Python Imaging Library) в формат тензора PyTorch. Он также нормирует тензор к диапазону значений от 0 до 1, деля все значения пикселей на 255.

Листинг 4:

```
train_transforms = transforms.Compose([
  transforms.Grayscale(num_output_channels=1),
```

```
    transforms.Resize((60, 60)),
    transforms.TrivialAugmentWide(num_magnitude_bins=31),
    transforms.ToTensor(),
])

val_transforms = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((60, 60)),
    transforms.ToTensor(),
])

test_transforms = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((60, 60)),
    transforms.ToTensor(),
])
```

Загрузим данные. При загрузке применим трансформацию. Данные датасета поделены на тренировочные, валидационные и тестовые. Однако разрыв в количестве изображений между тренировочным и валидационным датасетом слишком сильный. Для нивелирования данного эффекта, объединим тестовые и тренировочные данные, после чего поделим их в пропорции 7 : 2 : 1, и сразу же проверим результат. По итогу получим 4087 изображений для тренировочной выборки, 1168 для валидационной, и 585 для тестовой соответственно.

Листинг 5:

Вввод:

```
merged_set = datasets.ImageFolder(root=str(TRAIN_PATH),
transform=train_transforms) +  datasets.ImageFolder(root=str(TEST_PATH),
transform=test_transforms)                                    #обьединяем
тестовую и тренировочную выборку
dataset_size = len(merged_set)
train_size = int(dataset_size * 0.7)                  # 70% на обучение
val_size = int(dataset_size * 0.2)                    # 20% на валидацию
test_size = dataset_size - train_size - val_size      # 10% на тестирование
```

 # разбиваем датасет на три выборки

train_set, val_set, test_set = torch.utils.data.random_split(merged_set, [train_size, val_size, test_size])

 # создаем DataLoader для каждой выборки

train_loader = torch.utils.data.DataLoader(dataset=train_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)

val_loader = torch.utils.data.DataLoader(dataset=val_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)

test_loader = torch.utils.data.DataLoader(dataset=test_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)

print(f"Train data:\n{train_set}\nValidation data:\n{val_set}\nTest data:\n{test_set}\n")
print(f"Len train data: {len(train_set)}\nLen validation data: {len(val_set)}\nLen test data: {len(test_set)}\n")

Вывод:

Train data: <torch.utils.data.dataset.Subset object at 0x7d89d523bf70>

Validation data: <torch.utils.data.dataset.Subset object at 0x7d89d523b730>

Test data: <torch.utils.data.dataset.Subset object at 0x7d89d523bb80>

Len train data: 4087

Len validation data: 1168

Len test data: 585

1.3. Исследование данных.

1.3.1 Статистический анализ данных.

Статистический анализ данных – необходимый процесс обработки, интерпретации и анализа данных с помощью различных статистических методов, который является важным инструментом при работе с глубоким обучением. Каждая модель глубокого обучения требует большого объема данных для тестирования и обучения, и, чтобы обнаружить в данных некоторые паттерны и закономерности, необходимо проводить различные статистические анализы.

Для статистического анализа наших данных применим среднюю, медиану, стандартное отклонение, коэффициент корреляции и ковариации.

Mean, median, mode и std (standard deviation) – одни из основных статистических характеристик, которые используются для анализа данных. Mean – это среднее значение в выборке, представляющее собой сумму всех элементов выборки, разделенную на их количество, median – значение, делящее упорядоченный набор данных по полам и, в отличие от средней, нивелирует негативно искажающие выбросы (экстремальные значения), а std – стандартное отклонение – мера рассеяния (разброса) значений вокруг среднего значения, показывающая, насколько сильно значения отличаются от среднего значения. Более высокие значения std указывают на большую вариацию в данных, тогда как более низкие значения std указывают на меньшую вариацию.

Коэффициент корреляции и ковариации используются для анализа связей между двумя переменными. Коэффициент корреляции - это статистическая мера, определяющая взаимосвязь между переменными через измерение степени линейной связи. Коэффициент корреляции может принимать значения от -1 до 1. Значение 1 указывает на полную положительную корреляцию, а -1 – на полную отрицательную (обратную) корреляцию. Это означает, что, когда значения одной переменной

увеличиваются, значения другой переменной уменьшаются. Значение 0 указывает на отсутствие корреляции – переменные не связаны между собой. Значение 1 указывает на положительную корреляцию. Коэффициент ковариации – статистическая мера, определяющая силу и направление связи между двумя непрерывными переменными. Как и коэффициент корреляции, коэффициент ковариации может быть положительным или отрицательным. Он измеряет степень вариации двух переменных относительно среднего значения. Если значения переменных повышаются исправным образом, то ковариация будет положительной. Если одна переменная повышается, а вторая уменьшается, то ковариация будет отрицательной.

Для начала с помощью torch.stack() сложим все тензоры изображений (img_t) из обучающего набора данных в один большой тензор images_train, который имеет размерность (3, ширина_изображения, высота_изображения, количество_изображений). При этом dim=3 указывает, что мы хотим объединить тензоры изображений по четвертому измерению (количество изображений), чтобы получить общую выборку изображений. Аналогичную операцию применим и к валидационной и тестовой выборке.

Листинг 5:

Ввод:

```
images_train = torch.stack([img_t for img_t, _ in train_set], dim=3)
print(images_train.shape)

images_val = torch.stack([img_t for img_t, _ in val_set], dim=3)
print(images_val.shape)

images_test = torch.stack([img_t for img_t, _ in test_set], dim=3)
print(images_test.sha
```

Вывод:

```
torch.Size([1, 60, 60, 4087])
torch.Size([1, 60, 60, 1168])
```

torch.Size([1, 60, 60, 585])

Теперь вычислим значения среднего, стандартного отклонения, коэффициента корреляции и коэффициента ковариации для каждой из выборок. Используем метод view() для преобразования тензора images_train к одномерному массиву (flatten). Далее мы используем метод mean() с параметром dim=0 для вычисления среднего значения элементов по всему тензору вдоль 0-й размерности (размерности - соответствующая изображениям в данном случае), метод median() для вычисления медианы, метод std() для вычисления стандартного отклонения, метод corrcoef() и cov() для вычисления коэффициентов корреляции и ковариации для одного канала (ранее мы преобразовали наши тензоры изображения. Аналогичные операции применим и к валидационной и тестовой выборкам.

Листинг 6:

Ввод:

```
print(f"Train set mean: {  images_train.view(-1).mean(dim=0, keepdim=True)  }")
print(f"Val set mean: {  images_val.view(-1).mean(dim=0, keepdim=True)  }")
print(f"Test set mean: {  images_test.view(-1).mean(dim=0, keepdim=True)  }")
```

Вывод:

```
Train set mean: tensor([0.4463])
Val set mean: tensor([0.4457])
Test set mean: tensor([0.4453])
```

Ввод:

```
print(f"Train set median: {  images_train.view(-1).median(dim=0, keepdim=True)
}")
print(f"Val set median: {  images_val.view(-1).median(dim=0, keepdim=True)  }")
print(f"Test set median: {  images_test.view(-1).median(dim=0, keepdim=True)  }")
```

Вывод:

```
Train set mean: tensor([0.4824])
Val set mean: tensor([0.4784])
```

Test set mean: tensor([0.4824])

Ввод:

```
rounded_images_train = torch.round(images_train, decimals = 4)
rounded_images_val = torch.round(images_val, decimals = 4)
rounded_images_test = torch.round(images_test, decimals = 4)

print(f"Train set mode: {  rounded_images_train.view(3,-1).mode(dim=0, keepdim=True) }")
print(f"Val set mode: {  rounded_images_val.view(3,-1).mode(dim=0, keepdim=True)  }")
print(f"Test set mode: {  rounded_images_test.view(3,-1).mode(dim=0, keepdim=True)  }")
```

Вывод:

```
Train set median: torch.return_types.mode(
values=tensor([[0.0000, 0.0000, 0.0039,  ..., 0.0196, 0.1804, 0.0000]]),
indices=tensor([[2, 2, 2,  ..., 1, 0, 2]]))
Val set median: torch.return_types.mode(
values=tensor([[0.0000, 0.0078, 0.2157,  ..., 0.0000, 0.1020, 0.0431]]),
indices=tensor([[2, 2, 0,  ..., 2, 2, 2]]))
Test set median: torch.return_types.mode(
values=tensor([[0.6392, 0.0314, 0.0000,  ..., 0.0039, 0.0157, 0.0000]]),
indices=tensor([[0, 0, 0,  ..., 1, 1, 2]]))
```

Ввод:

```
print(f"Train set std: {  (images_train.view(-1).std(dim=0, keepdim=True))}")
print(f"Val set std: {  (images_val.view(-1).std(dim=0, keepdim=True))  }")
print(f"Test set std: {  (images_test.view(-1).std(dim=0, keepdim=True)) }")
```

Вывод:

```
Train set std: tensor([0.2716])
```

Val set std: tensor([0.2701])

Test set std: tensor([0.2713])

Ввод:

print(f"Train set corr coef: { images_train.view(3, -1).corrcoef() }")

print(f"Val set corr coef: { images_val.view(3, -1).corrcoef() }")

print(f"Test set corr coef: { images_test.view(3, -1).corrcoef() }")

Вывод:

Train set corr coef: tensor([[1.0000, 0.5763, 0.4888],
[0.5763, 1.0000, 0.6590],
[0.4888, 0.6590, 1.0000]])

Val set corr coef: tensor([[1.0000, 0.5806, 0.4971],
[0.5806, 1.0000, 0.6715],
[0.4971, 0.6715, 1.0000]])

Test set corr coef: tensor([[1.0000, 0.5666, 0.5192],
[0.5666, 1.0000, 0.6733],
[0.5192, 0.6733, 1.0000]])

Ввод:

print(f"Train set cov coef: { images_train.view(3, -1).cov() }")

print(f"Val set cov coef: { images_val.view(3, -1).cov() }")

print(f"Test set cov coef: { images_test.view(3, -1).cov() }")

Вывод:

Train set cov coef: tensor([[0.0598, 0.0368, 0.0364],
[0.0368, 0.0680, 0.0523],
[0.0364, 0.0523, 0.0925]])

Val set cov coef: tensor([[0.0595, 0.0366, 0.0365],
[0.0366, 0.0669, 0.0522],
[0.0365, 0.0522, 0.0905]])

Test set cov coef: tensor([[0.0597, 0.0356, 0.0381],
[0.0356, 0.0662, 0.0521],
[0.0381, 0.0521, 0.0903]])

Приступим к анализу полученных значений.

Значения среднего (mean) для выборок train, val и test близки к 0.45. Это говорит о том, что средний уровень яркости изображений находится примерно на середине диапазона значений пикселей [0, 1]. Значения среднего на всех выборках близки друг к другу и не сильно различаются, значит, средний уровень яркости не меняется сильно от выборки к выборке. На этом наборе данных нет сильной ярковой аномалии, и средний уровень яркости близок к однородному.

Уровень яркости изображений на всех выборках, рассчитанный как медиана, немного выше, чем при расчете среднего: примерно на 0,04. Такое различие, вероятно, связано с наличием выбросов в данных. Значит в выборках присутствуют изображения с яркими фонами или очень яркими/темными объектами, что оказало влияние на значение среднего уровня яркости. Медиана более устойчива к выбросам, так как она представляет собой значение, находящееся в середине упорядоченного ряда значений.

Стандартное отклонение 0,27 на всех выборках означает, что среднее значение отклоняется на 0,27 от среднего значения всех изображений. Это свидетельствует о том, что в наборе изображений есть меньше разнообразие, чем в другом наборе с большим стандартным отклонением. Общее невысокое значение стандартного отклонения указывает на невысокую изменчивость, что может быть хорошо, если предстоит применять алгоритмы машинного обучения для анализа изображений.

Значения коэффициента корреляции отображают степень связи между различными каналами цвета для каждой пары каналов. Значения корреляции можно интерпретировать следующим образом: Диагональные значения равны к 1, т.к. отображают максимальную связь двух одинаковых каналов – R/R G/G B/B (для расчёта коэффициентов корреляции и ковариации мы временно убрали метод transforms.Grayscale(num_output_channels=1**)**.

Далее можем выстроить цепочку по убыванию связности: G/B & B/G – 0,6; G/R & R/G – 0,58; B/R & R/B – 0,50. Можно сделать вывод, что зеленый канал

ближе к синему, чем к красному, красный ближе к зеленому, чем к синему. Наибольшее значение коэффициента корреляции находится между каналами G и B, что может указывать на наличие корреляции между синими и зелеными компонентами цвета. Значение коэффициента корреляции невысокое, и сильной связи между разными цветовыми каналами нет, что является нормальным для наборов данных с изображениями.

Значения коэффициента ковариации также отображают степень связи между различными каналами цвета для каждой пары каналов. Высокое значение коэффициента ковариации между двумя переменными может означать, что они сильно связаны между собой и могут быть скоррелированы. Это может привести к проблемам в анализе данных, таким как мультиколлинеарность. У нас же напротив значения сравнительно небольшие: в пределах от 0,03 до 0,09. Значения коэффициента ковариации близки к нулю 0, что свидетельствует о независимости переменных друг от друга и слабой связности.

Теперь, зная значения стандартного отклонения и средней, мы можем модернизировать наш код последовательности преобразований из листинга под номером 4 по средством нормализации. Нормализация изображений позволяет привести изображения к стандартной шкале и снизить вариацию значений пикселей на входе нейронной сети. Это уменьшает время обучения и позволяет сети более эффективно использовать веса, что в свою очередь увеличивает точность и скорость работы нейронной сети. Функция нормализации вычитает среднее значение канала из каждого пикселя в изображении и затем делит на стандартное отклонение канала. Таким образом, нормализация приводит распределение значений пикселей к среднему значению 0 и стандартному отклонению 1.

Листинг 7:

```
train_transforms = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((60, 60)),
    transforms.TrivialAugmentWide(num_magnitude_bins=31),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4465], std=[ 0.2716])
])

val_transforms = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((60, 60)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4471], std=[ 0.2701])
])

test_transforms = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((60, 60)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4536], std=[0 .2713])
])
```

1.3.2 Визуализация данных.

Визуализация данных незаменима как процесс преобразования данных из числовых или текстовых значений в графическое представление, которое более понятно и интуитивно для анализа. Это может быть график, карта, диаграмма, схема и т.д. Возможности визуализации данных становятся особенно важны при анализе больших массивов данных, таких как данные для глубокого обучения.

В датасете chest-xray-pneumonia все снимки поделены на два класса: на которых есть пневмония и на которых она, соответственно, отсутствует. Получим значения классов как в виде списка, так и в виде словаря:

Листинг 8:

Ввод:

```
class_names = train_set.classes
print(class_names)
class_dict = train_set.class_to_idx
print(class_dict)
```

Вывод:

```
['NORMAL', 'PNEUMONIA']
{'NORMAL': 0, 'PNEUMONIA': 1}
```

Следующим шагом необходимо проанализировать количественное распределение данных между классами. Сделаем это на примере тренировочной выборки. Для визуального отображения будем использовать гистограмму:

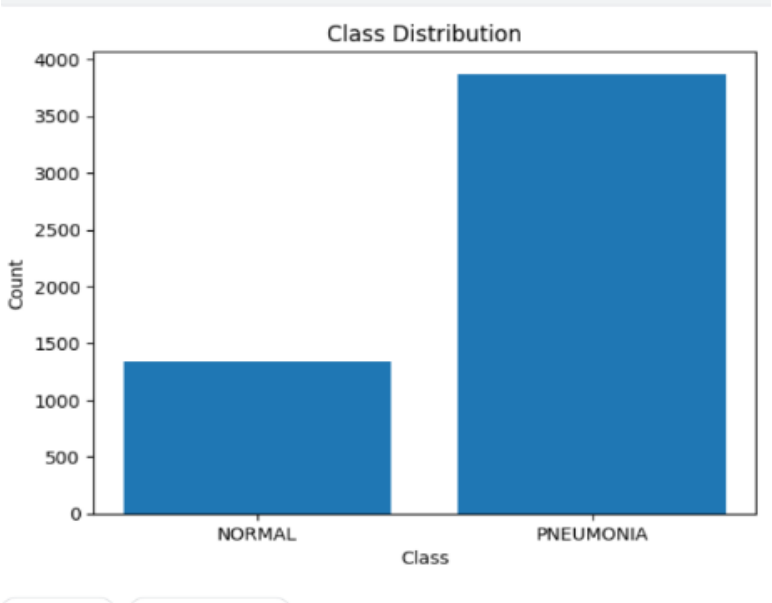
Листинг 9:

Ввод:

```
class_names = train_set.classes
class_counts = [0] * len(class_names)
for image, label in train_set:
```

```
    class_counts[label] += 1

# Рисуем гистограмму распределения классов
plt.bar(class_names, class_counts)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```

Вывод:



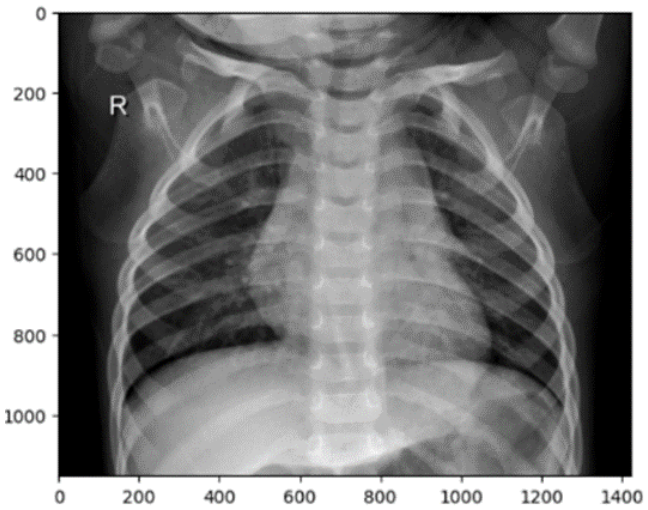
Как мы видим, классов NORMAL, больше чем классов PNEUMONIA в 2,5 раза. Загрузим и выведем одно изображение из тренировочной выборки:

Листинг 10:

Ввод:

```
_train_set = datasets.ImageFolder(root=str(TRAIN_PATH))
img, label = _train_set[1]
class_names = _train_set.classes
img, label, class_names[label]
plt.imshow(img)
plt.show()
```

Вывод:



Также создадим функцию для вывода произвольного количества рандомных изображений:

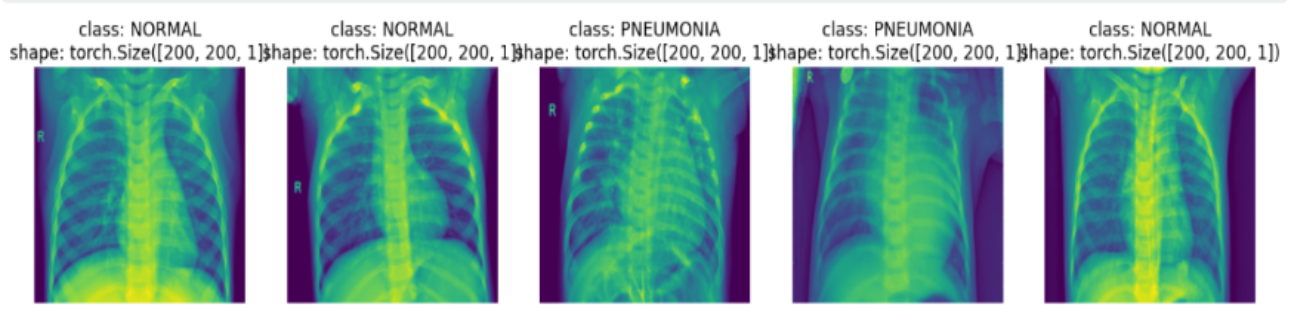
Листинг 11:

Ввод:

```
def display_random_images(dataset: torch.utils.data.dataset.Dataset,
                          classes: List[str] = None,
                          n: int = 10,
                          display_shape: bool = True,
                          seed: int = None):
 if n > 10:
    n = 10
    display_shape = False
    print(f"For display purposes, n shouldn't be larger than 10, setting to 10 and removing shape display.")

 if seed:
    random.seed(seed)
 random_samples_idx = random.sample(range(len(dataset)), k=n)
 plt.figure(figsize=(16, 8))

 for i, targ_sample in enumerate(random_samples_idx):
   targ_image, targ_label = dataset[targ_sample][0], dataset[targ_sample][1]
   targ_image_adjust = targ_image.permute(1, 2, 0)
   plt.subplot(1, n, i+1)
   plt.imshow(targ_image_adjust)
```

```
        plt.axis("off")
        if classes:
            title = f"class: {classes[targ_label]}"
            if display_shape:
                title = title + f"\nshape: {targ_image_adjust.shape}"
        plt.title(title)
    display_random_images(test_set, n=5,  classes=class_names, seed=None)
```

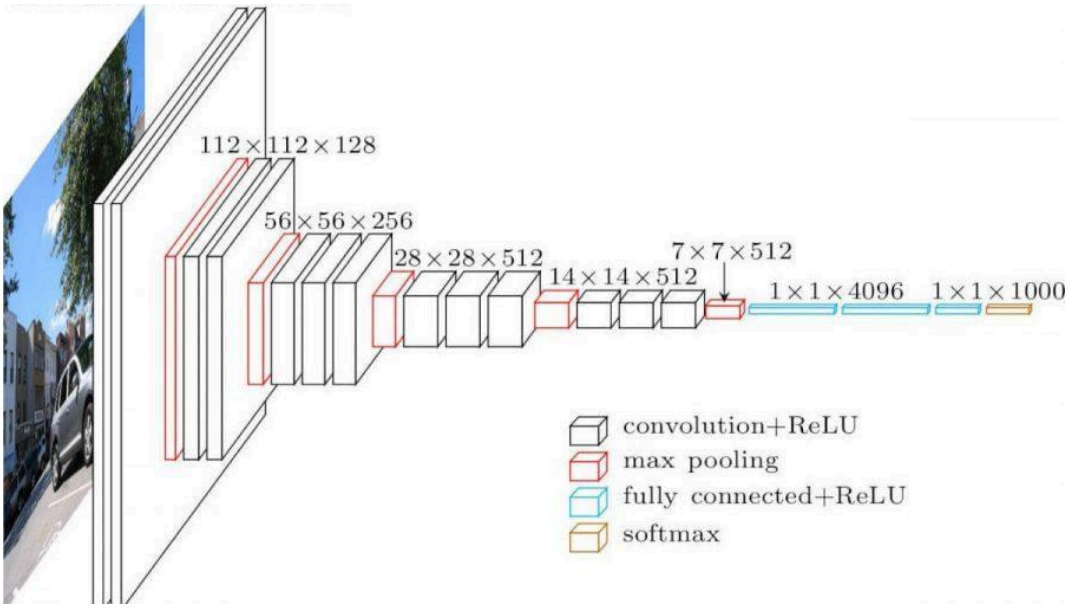
Вывод:



1.4 Процесс машинного обучения. Работа с моделью машинного обучения.

1.4.1 История создания модели машинного обучения.

Для глубокого обучения для бинарной классификации снимков мы будем использовать модель, созданную по подобию VGG16. VGG16 — модель сверточной нейронной сети, предложенная K. Simonyan и A. Zisserman из Оксфордского университета в статье “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Модель достигает точности 92.7% — топ-5, при тестировании на ImageNet в задаче распознавания объектов на изображении.



Концепция данной модели основывается на использовании небольших (3x3) сверток в качестве основных строительных блоков. Изначально это было сделано для улучшения точности классификации, а также снижения количества параметров в сети. Идея использования более мелких сверток вместо больших (например, 5x5 или 7x7) была взята из логики, что несколько сверток размера 3x3 дадут тот же эффект, что и одна свертка размера 5x5 или 7x7. При этом, использование более мелких сверток позволяет значительно уменьшить количество параметров в сети.

Архитектура модели VGG16 включает 16 слоев, включая 13 сверточных слоев и 3 полносвязных слоя. На вход слоя conv1 подаются RGB изображения размера 224x224 и преобразуются с помощью матричного умножения, используя фильтры (ядра), меньшего размера. Фильтры обнаруживают особенности изображения, такие как границы, углы и линии, и создают карты функций. В начале модели находятся 2 или 3 сверточных слоя (в зависимости от варианта модели) с размером фильтра 3x3 и шагом 1, затем следует операция max-pooling с размером фильтра 2x2 и шагом 2. Затем

последовательность сверточных слоев и операция max-pooling повторяется несколько раз - в итоге формируются блоки сверточных слоев с различным числом фильтров.

После нескольких блоков сверточных слоев, последний блок перед тремя полностью связанными слоями включает операцию max-pooling с размером фильтра 2x2 и шагом 2, в котором обработка изображения переходит от большинства пикселей к меньшему числу признаков, что позволяет уменьшить количество параметров в сети.

После стека сверточных слоев (который имеет разную глубину в разных архитектурах) идут три полносвязных слоя: первые два имеют по 4096 каналов, третий — 1000 каналов (так как в соревновании ILSVRC требуется классифицировать объекты по 1000 категориям; следовательно, классу соответствует один канал). Последним идет soft-max слой. Конфигурация полносвязных слоев одна и та же во всех нейросетях.

1.4.2 Математическое обоснование модели машинного обучения.

Математический подход к определению сверточных нейронных сетей включает использование свертки, операции max-pooling, активационных функций и алгоритма прямого и обратного распространения ошибки.

Свертка - это математическая операция, которая плавно перемещает одну функцию по другой и измеряет интеграл их точечного умножения. Она имеет глубинные связи с преобразованием Фурье и преобразованием Лапласа и интенсивно используется в обработке сигналов.

Свертка описывается следующей формулой:

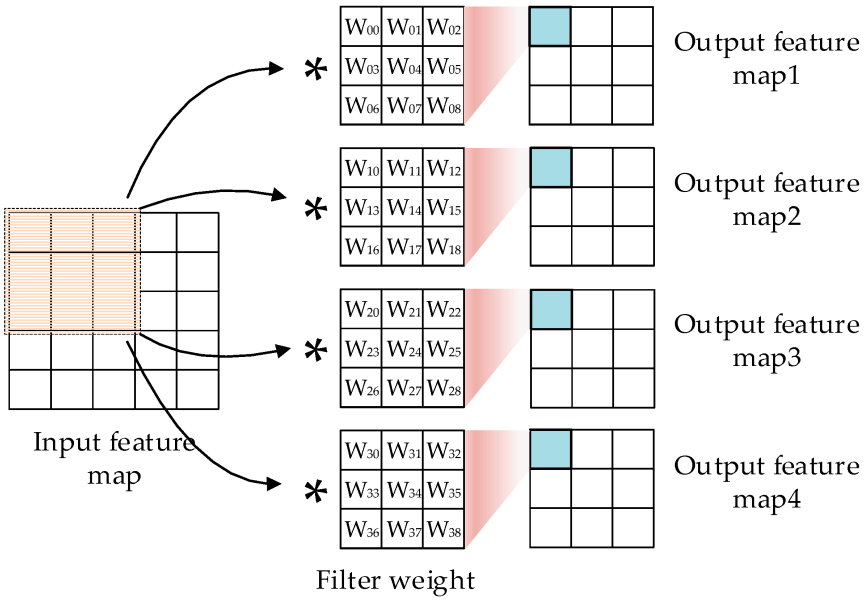
$$< f * g > (i, j) = \sum_{l=n_0}^{n_1} \sum_{k=m_0}^{m_1} f(i-l)(j-k) \cdot g(l,k)$$

где f - входной сигнал, g - ядро свертки (фильтр), (i, j) - позиция ядра на входном сигнале.

В PyTorch свертка реализована с помощью nn.Conv2d() со следующими параметрами:

- in_channels(int) - количество каналов во входном изображении.

- out_channels(int) – количество выходных каналов, созданных сверткой.
- kernel_size(int or tuple) — размер свертывающегося ядра/фильтра.
- stride(int or tuple, optional) — насколько большой шаг выполняет свертывающее ядро за один раз. По умолчанию: 1.
- padding(int, tuple, str) — padding добавляется ко всем четырем сторонам ввода. По умолчанию: 0.



Max-pooling - это операция в сверточных нейронных сетях, которая выполняет субдискретизацию изображения путем выбора максимального значения из каждого окна области на изображении и присваивания этого значения новому пикселю на выходе. Max-pooling обычно применяется после сверточных слоев, чтобы уменьшить количество параметров модели и предотвратить переобучение. Отрицательным эффектом max-pooling может быть потеря некоторой информации о местоположении объектов на изображении. Тем не менее, в целом, max-pooling справляется с анализом и преобразованием изображений, сохраняя при этом элементы, необходимые для точного определения характеристик изображения.

Операция max-pooling выполняется, выбрав окно определенного размера на изображении и выбрав максимальный элемент из этого окна с использованием следующей формулы:

$$maxpool(x,y) = max_{i,j}(input_{x+i, y+j})$$

Активационные функции в сверточных слоях обычно являются нелинейными, например, rectified linear unit (ReLU), которая обнуляет все отрицательные значения, а положительные оставляет без изменений:

$$ReLU(x)= max(0,x)$$

Алгоритм прямого распространения - это алгоритм машинного обучения, который используется в нейронных сетях для расчета выходных значений на основе входных данных. При работе нейронной сети, входные данные, которые представлены в виде вектора, передаются через каждый узел сети, где проходит вычисление значений. Результат вычислений передается на вход следующего узла, и процесс продолжается до вычисления выходного значения нейронной сети.

Шаги алгоритма прямого распространения:

1. Инициализация начальных весов: каждый узел вычисляет взвешенную сумму выходов всех входных узлов, умноженных на соответствующие им весовые коэффициенты.

2. Подача входных данных на вход нейронной сети: данные передаются нейронной сети, где каждый узел вычисляет свое выходное значение.

3. Расчет выходных значений для каждого нейрона слоя на основе входных данных и весов: Происходит вычисление выходных значений каждого узла, используя активационную функцию. Для извлечения нелинейных отношений между данными и их представлением узел может использовать различные функции активации, например, сигмоидальную функцию, функцию ReLU, softMax и т.д.

4. Передача выходных значений от первого слоя к следующему и повторение операций 3-4 для каждого слоя до получения конечного результата: процесс повторяется для каждого следующего слоя, где результат расчета предыдущего слоя становится входным значением для следующего слоя.

Алгоритм обратного распространения - это алгоритм машинного обучения, который используется для обучения нейронных сетей путем корректировки весов на основе вычисленной ошибки между предсказанными и реальными значениями.

Шаги алгоритма обратного распространения:

1. Инициализация начальных весов: веса инициализируются случайным образом, которые будут использоваться для вычисления выходных значений и расчета ошибки.

2. Подача входных данных на вход нейронной сети и расчет выходных значений: входные данные передаются через нейронную сеть и вычисляются значения выходных узлов.

3. Сравнение предсказанных значений с реальными значениями и расчет ошибки: Сравниваются уровни ошибки, полученные после обучения с реальными ответами. Ошибка может быть определена как разность между предсказанным значением и фактическим значением, возведенной в квадрат.

4. Обратное распространение ошибки от последнего слоя к первому:

- Вычисление ошибки слоя: ошибка вычисляется на основе разницы между полученным и ожидаемым выходом после прямого распространения на последнем слое.

- Обратное распространение ошибки до первого скрытого слоя: ошибка передается от последнего слоя к первому скрытому слою, где веса корректируются в соответствии с полученной ошибкой.

5. Обновление весов на основе расчета градиента ошибки и скорости обучения: Рассчитывается значение градиента функции ошибки по отношению к весам каждого узла, и они корректируются со скоростью обучения.

6. Повторение шагов 2-5 для каждого обучающего примера: шаги 2-5 повторяются для каждого примера в обучающем наборе, чтобы улучшить точность предсказания.

7. Остановка обучения при достижении заданного количества эпох или достижении минимальной ошибки: обучение останавливается, когда достигнуто максимальное количество эпох или когда достигнута минимальная ошибка.

Таким образом, математическое обоснование модели VGG16 заключается в использовании сверточных слоев, активационных функций, операции свертки и пулинга, а также прямого и обратного распространения ошибки для определения оптимальных параметров для классификации изображений.

1.4.3 Минимизация ошибок, оптимизаторы и оценка качества работы модели машинного обучения.

Для обучения модели существуют функции потерь, основной задачей которых является минимизация ошибок в процессе обучения. Таким образом, цель функций потерь - минимизировать ошибки между фактическими и предсказанными значениями, чтобы модель лучше работала на новых данных.

Для обновления параметров модели в процессе обучения с целью минимизации функции потерь необходимы особые алгоритмы - оптимизаторы. Они определяют, как модель должна обновлять веса и смещения в процессе обучения, чтобы достичь наилучшего качества предсказаний.

Оптимизаторы работают в тесном сочетании с функциями потерь. Функция потерь генерирует значение ошибки, на основе которого оптимизатор может рассчитать, как обновлять параметры модели.

Для оценки качества работы модели множество разных метрик. Оценка производится на тестовой и валидационной выборках и позволяет определить, насколько хорошо модель работает с тестовыми данными.

Познакомимся с существующими реализациями функций потерь, оптимизаторов и метрик оценки качества модели.

Реализации функций потерь:

Существует множество функций потерь в зависимости от задачи машинного обучения и типа предсказаний модели:

Mean Squared Error (MSE) - используется в задачах регрессии и измеряет среднеквадратичную разницу между фактическими и предсказанными значениями.

Mean Absolute Error (MAE) - также используется в задачах регрессии и измеряет среднюю абсолютную разницу между фактическими и предсказанными значениями.

Binary Cross-Entropy Loss - используется в задачах бинарной классификации, когда необходимо получить вероятность принадлежности к одному из двух классов.

Categorical Cross-Entropy Loss - используется в задачах многоклассовой классификации и измеряет расхождение между фактическим распределением

вероятностей и предсказанным распределением вероятностей для каждого класса.

Sparse Categorical Cross-Entropy Loss - аналогична Categorical Cross-Entropy Loss, используется для задач многоклассовой классификации, когда метки классов представлены в виде целых чисел.

Hinge Loss - используется в задачах классификации с потерями равными нулю до определенного порога (hinge point) и затем быстро возрастает.

Huber Loss - используется в задачах регрессии и уменьшая влияние выбросов.

Focal Loss - используется в задачах с несбалансированными классами и уменьшает влияние "легких" примеров.

В нашей модели будем использовать BCE Loss.

Реализации оптимизаторов:

Существует множество различных оптимизаторов, каждый со своими преимуществами и недостатками. Некоторые из наиболее распространенных оптимизаторов в машинном обучении включают в себя:

Стохастический градиентный спуск (SGD): наиболее простой и распространенный оптимизатор, который использует градиент для обновления параметров.

AdaGrad: адаптивный оптимизатор, который динамически настраивает скорость обучения для каждого параметра на основе используемых градиентов.

RMSprop: оптимизатор, который использует экспоненциальное скользящее среднее градиента для ускорения обучения и предотвращения переполнения.

Adam: оптимизатор, который сочетает в себе преимущества SGD и RMSprop, используя экспоненциальное скользящее среднее для адаптивного изменения скорости обучения и момента.

В нашей модели будем использовать SGD.

Реализации метрик оценивания качества модели:

Существует множество метрик, которые используются для оценки качества модели машинного обучения. Но для начала ознакомимся такими сокращениями, как TP, FP, FN и TN.

TP (True Positive) - это кол-во правильно определенных положительных примеров. Например, если мы строим модель для определения, болен ли конкретный пациент COVID-19 или нет и результаты показывают, что пациент действительно является положительным на COVID-19, то это считается TP.

TN (True Negative) - это кол-во правильно определенных отрицательных примеров. Используя тот же пример с COVID-19, если модель определяет, что пациент не заражен вирусом, а реально он не заражен, то это считается TN.

FP (False Positive) - это неправильно определенные положительные примеры. Если модель определяет, что человек заражен вирусом, но в действительности он не болен, то это считается FP.

FN (False Negative) - это неправильно определенные отрицательные примеры. В случае если модель не определяет, что пациент заразился вирусом, но в действительности он заражен, то это считается FN.

Теперь перейдем к самим метрикам оценки качества модели:

Accuracy (точность) - мера того, насколько хорошо модель правильно классифицирует данные. Она определяется как отношение правильно классифицированных примеров к общему количеству примеров.

Формула подсчета accuracy:

$$Accuracy = (TP+TN) / (TP+FP+FN+TN) = (TP+TN) / total$$

Recall (полнота) - мера того, насколько хорошо модель идентифицирует положительные примеры. Она определяется как отношение правильно идентифицированных положительных примеров к общему количеству положительных примеров.

Формула подсчета recall:

$$Recall = TP / Actual\ positive = TP / (TP+FN)$$

Specificity / True Negative Rate (TNR) - мера того, насколько хорошо модель идентифицирует отрицательные примеры. Как Recall имеет дело с положительным классом, так и Specificity имеет дело с отрицательным классом.

Формула подсчета specifity:

$$Specificity = TN / Actual\ negative = TN / (FP+TN)$$

Precison - мера того, насколько хорошо модель идентифицирует только правильные положительные примеры. Она определяется как отношение

правильно идентифицированных положительных примеров к общему количеству примеров, которые модель идентифицирует как положительные.

Формула подсчета precison:

$$Precision = TP / Predicted\ positive = TP / (TP+FP)$$

F1-score (F-мера) - это среднее гармоническое между precision и recall. Она может использоваться как общая мера качества модели машинного обучения.

Формула подсчета F1-score:

$$F1 = 2* (Precision*Recall) / (Precision+Recall)$$

ROC-кривая - представление кривой, которая показывает зависимость между долей правильно классифицированных положительных примеров (True Positive Rate) и долей неправильно классифицированных отрицательных примеров (False Positive Rate) при изменении порогового значения.

AUC-ROC - это площадь, занимаемая кривой ROC. AUC-ROC может быть использована для сравнения нескольких моделей машинного обучения. Чем выше значение AUC-ROC, тем лучше модель.

Выбор метрики зависит от типа задачи, которую мы решаем в конкретной модели машинного обучения. Например, для задач классификации чаще использоваться accuracy, а для задачи поиска выбросов может быть использована F1-score. Поэтому для курсовой выберем метрику accuracy.

1.4.4 Код для применения модели машинного обучения и оценки качества ее работы.

Теперь приступаем к построению собственной модели на основе уже рассмотренной ранее VGG16. По архитектуре она будет представлять собой уменьшенную и упрощенную копию своего «донора», чтобы модель не переобучалась из-за слишком не к месту усложненной и громоздкой архитектуры, которой по своей сути VGG16 и является для данного набора данных.

Начнем с инициализации модели. Для большей гибкости модели создадим подкласс nn.Module. Для него необходимо описание функции forward(). Создадим все экземпляры всех слоев, и затем воспользуемся всеми этими экземплярами один за другим в функции forward(). Всего модель EightModel содержит два сверточных слоя (Conv2d), и два полносвязных (Linear). В первом сверточном слое имеем только один входной канал, т.к. на вход подаются черно-белые снимки, размер ядра указываем как 3 на 3, padding присвоим 1, чтобы центр ядра попадал на каждый пиксель, в том числе и на крайние, размер шага также сделаем равным 1. Далее применяем операция батч-нормализации (BatchNorm2d), которая понизит риск переобучения и ускорит процесс обучения за счет более быстрого схождения данных. После применим функцию активации nn.ReLU() и наконец используем операцию Max Pooling для уменьшения размерности данных в 2 раза и извлечения наиболее значимых признаков из исходных данных. В полносвязном слое применим nn.Flatten() для выпрямления многомерного массива в одномерный вектор, nn.Dropout() для зануления (отключения) случайных нейронов на каждой итерации обучения для улучшения обобщающей способности модели, и также функцию активации nn.ReLU().

Таким образом входящее изображение размерностью проходит следующие преобразования: 1 * 60 * 60 => 8 * 30 * 30 => 8 * 15 * 15 => 1800 => 100 => 2.

Листинг 12:

```
class EightModel(nn.Module):
  def __init__(self) -> None:
    super(EightModel, self).__init__()


    self.layer_1 = nn.Sequential(
      ##Input = 3 x 60 x 60, Output = 8 x 30 x 30
      nn.Conv2d(in_channels=1,
            out_channels=8,
```

```
            kernel_size=(3, 3),
            stride=1,
            padding=1),
        nn.BatchNorm2d(8),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size = 2, stride = 2)
    )

    self.layer_2 = nn.Sequential(
        #Input = 8 x 30 x 50, Output = 8 x 15 x 15
        nn.Conv2d(in_channels=8,
            out_channels=8,
            kernel_size = (3,3),
            stride = 1,
            padding = 1),
        nn.BatchNorm2d(8),
        n.ReLU(),
        nn.MaxPool2d(kernel_size = 2, stride = 2)
    )

    self.fc_2 = nn.Sequential(
        nn.Flatten(),
        nn.Dropout(p=0.5),
        nn.Linear(in_features=1800, out_features=100),
        nn.ReLU()  )
    self.fc_4 = nn.Sequential(
        nn.Linear(in_features=100, out_features=2)
    )

def forward(self, x: torch.Tensor):
    out = self.layer_1(x)
    out = self.layer_2(out)

    out = self.fc_2(out)
    out = self.fc_4(out)
```

```
        return out

torch.manual_seed(100)
model_exemplar = EightModel().to(device)
```

После инициализации модели следующим шагом определим функцию для обучения модели в цикле на тренировочной выборке и вычисления значения функции потерь и метрики точности на каждой эпохе обучения по следующему алгоритму:

1. Сначала настроим модель в режим обучения.
2. На каждой эпохе, данные подаем в модель и полученное предсказание используется для вычисления значения функции потерь.
3. Затем градиенты обнуляются и выполняется обратное распространение ошибки.
4. Оптимизатор используется для обновления параметров модели.
5. Наконец, метрика точности вычисляется для каждого батча данных. После прохода по всем батчам данных для каждой метрики вычисляется среднее значение.

Функция возвращает среднее значение функции потерь и точности на всех батчах.

Листинг 13:

```
# create train loop function
def train_step_function(model: torch.nn.Module,
                        dataloader: torch.utils.data.DataLoader,
                        loss_fn: torch.nn.Module,
                        optimizer: torch.optim.Optimizer):

  model.train()
  train_loss, train_accuracy = 0, 0
```

```
    for (batch, (X, y)) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        y_predict = model(X)

        loss = loss_fn(y_predict, y)
        train_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        y_predict_class = torch.argmax(torch.softmax(y_predict, dim=1), dim=1)
        train_accuracy += (y_predict_class == y).sum().item()/len(y_predict)

    train_loss = train_loss / len(dataloader)
    train_accuracy = train_accuracy / len(dataloader)
    return train_loss, train_accuracy
```

Также определим функции для проверки модели на валидационных и тренировочных данных, чтобы оценить ее производительность на независимых выборках. Алгоритм:

1. Сначала настраивается модель в режиме оценки.
2. Затем, проход по каждому батчу данных из DataLoader.
3. Данные подаются в модель и используется полученное предсказание для вычисления значения функции потерь.
4. Точность модели также вычисляется для каждого батча данных.
5. После прохода по всем батчам данных для каждой метрики вычисляется среднее значение.

Функция возвращает среднее значение функции потерь и точности на всех батчах валидационных данных.

Листинг 14:

```
# create validation loop function
def validation_step_function(model: torch.nn.Module,
                             dataloader: torch.utils.data.DataLoader,
                             loss_fn: torch.nn.Module):

    model.eval()  # Put model in eval mode

    val_loss, val_accuracy = 0, 0

    with torch.inference_mode():
        for (batch, (X, y)) in enumerate(dataloader):
            X, y = X.to(device), y.to(device)

            # 1. Forward pass
            val_pred_logits = model(X)

            # 2. Calculate and accumulate loss
            loss = loss_fn(val_pred_logits, y)
            val_loss += loss.item()

            # Calculate and accumulate accuracy
            val_pred_labels = val_pred_logits.argmax(dim=1)
            val_accuracy += ((val_pred_labels == y).sum().item() / len(val_pred_labels))

    # Adjust metrics to get average loss and accuracy per batch
    val_loss = val_loss / len(dataloader)
    val_accuracy = val_accuracy / len(dataloader)
    return val_loss, val_accuracy

# create validation loop function
def test_step_function(model: torch.nn.Module,
                       dataloader: torch.utils.data.DataLoader,
                       loss_fn: torch.nn.Module):
```

```
    model.eval()

    test_loss, test_accuracy = 0, 0
    with torch.inference_mode():
        for (batch, (X, y)) in enumerate(dataloader):
            X, y = X.to(device), y.to(device)
            test_pred_logits = model(X)

            loss = loss_fn(test_pred_logits, y)
            test_loss += loss.item()

            test_pred_labels = test_pred_logits.argmax(dim=1)
            test_accuracy += ((test_pred_labels == y).sum().item() / len(test_pred_labels))

    test_loss = test_loss / len(dataloader)
    test_accuracy = test_accuracy / len(dataloader)
    return test_loss, test_accuracy
```

Следующим шагом определим функцию, которая выполняет обучение модели в цикле, вычисляя значения функции потерь и точности на тренировочных, валидационных и тестовых выборках для каждой эпохи обучения и возвращая в качестве результата словарь, содержащий значения функции потерь и точности на тренировочной, валидационной и тестовых данных для каждой эпохи обучения по следующему алгоритму:

1. На вход функции подаются необходимые параметры для обучения и тестирования модели, определение функции потерь, её параметры и число эпох обучения.
2. В цикле создается словарь для хранения результатов.
3. В каждой эпохе происходит обучение модели на тренировочных данных, проверка производительности модели на валидационных данных и оценка качества модели на тестовых данных.
4. После этого результаты выводятся на экран.

5. Обновляются результаты в словаре.

6. После прохода по числу эпох функция возвращает словарь с результатами.

Листинг 15:

```
def train_function(model: torch.nn.Module,
                   train_dataloader: torch.utils.data.DataLoader,
                   validation_dataloader: torch.utils.data.DataLoader,
                   test_dataloader: torch.utils.data.DataLoader,
                   optimizer: torch.optim.Optimizer,
                   loss_fn: torch.nn.Module = nn.CrossEntropyLoss(),
                   epochs: int = 50):          ):

  # 2. Create empty results dictionary
  results = { "train_loss": [],
              "train_accuracy": [],
              "val_loss": [],
              "val_accuracy": [],
              "test_loss": [],
              "test_accuracy": []
  }

  # 3. Loop through training and testing steps for a number of epochs
  for epoch in tqdm(range(epochs)):
     train_loss, train_accuracy = train_step_function(model=model,
                                                      dataloader=train_dataloader,
                                                      loss_fn=loss_fn,
                                                     optimizer=optimizer)

     val_loss, val_accuracy = validation_step_function(model=model,
                                                      dataloader=validation_dataloader,
                                                      loss_fn=loss_fn)

     test_loss, test_accuracy = test_step_function(model=model,
                                                   dataloader=test_dataloader,
```

```
                                        loss_fn=loss_fn)

    # 4. Print out results
    print(f"Epoch: {epoch + 1} | train_loss: {train_loss:.4f} | train_accuracy:
{train_accuracy:.4f} | val_loss: {val_loss:.4f} | val_accuracy: {val_accuracy:.4f} | test_loss:
{test_loss:.4f} | test_accuracy: {test_accuracy:.4f}")

    # 5. Update results dictionary
    results["train_loss"].append(train_loss)
    results["train_accuracy"].append(train_accuracy)
    results["val_loss"].append(val_loss)
    results["val_accuracy"].append(val_accuracy)
    results["test_loss"].append(test_loss)
    results["test_accuracy"].append(test_accuracy)

  # 6. Return results at the end of the epochs
  return results
```

Для начала обучения осталось лишь запустить тренировочную функцию. Тут же укажем количество эпох – 14, создадим экземпляр model_exemplar класса EightModel, установим функцию потерь (loss_fn) - кросс-энтропию и оптимизатор - стохастический градиентный спуск (SGD). Результаты работы функции train_function() присвоим переменной model_exemplar_results.

Листинг 16:

Ввод:

```
torch.manual_seed(100)
EPOCHS = 14

model_exemplar = EightModel()

loss_fn = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(params=model_exemplar.parameters(),
lr=LEARNING_RATE)

from timeit import default_timer as timer
start_time = timer()

model_exemplar_results = train_function(model=model_exemplar,
                        train_dataloader=train_loader,
                        validation_dataloader=val_loader,
                        test_dataloader=test_loader,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=EPOCHS)
end_time = timer()
print(f"Total training time: {end_time-start_time:.3f} seconds")
```

Вывод:

Epoch: 1 | train_loss: 0.5799 | train_accuracy: 0.7190 | val_loss: 0.5679 | val_accuracy: 0.7352 | test_loss: 0.6031 | test_accuracy: 0.7050

Epoch: 2 | train_loss: 0.5277 | train_accuracy: 0.7499 | val_loss: 0.5090 | val_accuracy: 0.7516 | test_loss: 0.5606 | test_accuracy: 0.7038

Epoch: 3 | train_loss: 0.4932 | train_accuracy: 0.7725 | val_loss: 0.4765 | val_accuracy: 0.7516 | test_loss: 0.5177 | test_accuracy: 0.7175

Epoch: 4 | train_loss: 0.4581 | train_accuracy: 0.7975 | val_loss: 0.4637 | val_accuracy: 0.7547 | test_loss: 0.5109 | test_accuracy: 0.7253

Epoch: 5 | train_loss: 0.4289 | train_accuracy: 0.8098 | val_loss: 0.4232 | val_accuracy: 0.7937 | test_loss: 0.4605 | test_accuracy: 0.7823

Epoch: 6 | train_loss: 0.4130 | train_accuracy: 0.8263 | val_loss: 0.4262 | val_accuracy: 0.7812 | test_loss: 0.4566 | test_accuracy: 0.7702

Epoch: 7 | train_loss: 0.3969 | train_accuracy: 0.8240 | val_loss: 0.3837 | val_accuracy: 0.8375 | test_loss: 0.4227 | test_accuracy: 0.7897

Epoch: 8 | train_loss: 0.3969 | train_accuracy: 0.8283 | val_loss: 0.3781 | val_accuracy: 0.8305 | test_loss: 0.4005 | test_accuracy: 0.8147

Epoch: 9 | train_loss: 0.3883 | train_accuracy: 0.8290 | val_loss: 0.3601 | val_accuracy: 0.8531 | test_loss: 0.3964 | test_accuracy: 0.8195

Epoch: 10 | train_loss: 0.3728 | train_accuracy: 0.8461 | val_loss: 0.3484 | val_accuracy: 0.8562 | test_loss: 0.3849 | test_accuracy: 0.8339

Epoch: 11 | train_loss: 0.3623 | train_accuracy: 0.8437 | val_loss: 0.3928 | val_accuracy: 0.8094 | test_loss: 0.4273 | test_accuracy: 0.7952

Epoch: 12 | train_loss: 0.3593 | train_accuracy: 0.8433 | val_loss: 0.3529 | val_accuracy: 0.8641 | test_loss: 0.3697 | test_accuracy: 0.8425

Epoch: 13 | train_loss: 0.3412 | train_accuracy: 0.8567 | val_loss: 0.3445 | val_accuracy: 0.8453 | test_loss: 0.3590 | test_accuracy: 0.8347

Epoch: 14 | train_loss: 0.3397 | train_accuracy: 0.8548 | val_loss: 0.3485 | val_accuracy: 0.8445 | test_loss: 0.3459 | test_accuracy: 0.8691

Total training time: 552.588 seconds

По итогу обучения получаем очень неплохую точность в примерно 85% на всех выборках. Обучение проходило плавно и модель не переобучилась, что можно увидеть на графиках:

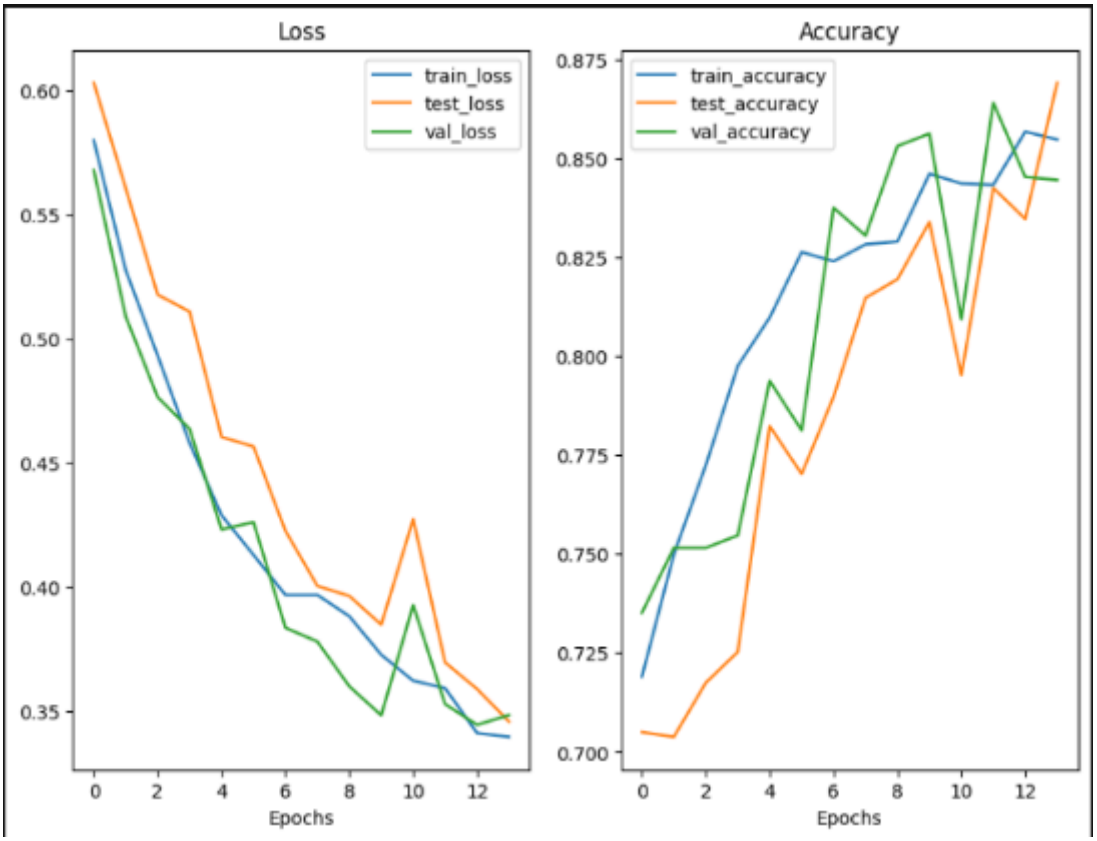
Листинг 17:

Ввод:

```
def plot_loss_accuracy(results: Dict[str, List[float]]):
    train_loss = results['train_loss']
    test_loss = results['test_loss']
    val_loss = results['val_loss']

    train_accuracy = results['train_accuracy']
    test_accuracy = results['test_accuracy']
    val_accuracy = results['val_accuracy']

    epochs = range(len(results['train_loss']))

    plt.figure(figsize=(15, 7))

    plt.subplot(1, 3, 1)
    plt.plot(epochs, train_loss, label='train_loss')
    plt.plot(epochs, test_loss, label='test_loss')
```

```
    plt.plot(epochs, val_loss, label='val_loss')

    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.legend()

    plt.subplot(1, 3, 2)
    plt.plot(epochs, train_accuracy, label='train_accuracy')
    plt.plot(epochs, test_accuracy, label='test_accuracy')
    plt.plot(epochs, val_accuracy, label='val_accuracy')

    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.legend();

plot_loss_accuracy(model_exemplar_results)
```

Вывод:



Попробуем выжать из нашей модели еще большую точность. Увеличим количество эпох еще на 26 и дообучим нашу модель. Результаты мы выведем с помощью таблицы датафрейма, использую Pandas:

Листинг 18:

Ввод:

```
import pandas as pd
model_exemplar_df = pd.DataFrame(model_exemplar_results)
print(model_exemplar_df)
```

Вывод:

	train_loss	train_accuracy	val_loss	val_accuracy	test_loss	test_accuracy
0	0.288994	0.881969	0.319322	0.843750	0.284480	0.887864
1	0.305446	0.870628	0.284455	0.872656	0.263974	0.880822
2	0.308987	0.868827	0.284669	0.885156	0.261540	0.904666
3	0.297187	0.878274	0.277775	0.899219	0.286975	0.885509
4	0.296289	0.874309	0.309599	0.860156	0.319827	0.854623
5	0.303274	0.874083	0.277126	0.900781	0.272449	0.889020
6	0.295852	0.874124	0.395497	0.843750	0.391460	0.845634
7	0.295951	0.877183	0.300239	0.885938	0.260923	0.910509
8	0.295588	0.872074	0.268762	0.882812	0.274539	0.883562
9	0.284123	0.881444	0.288995	0.869531	0.270239	0.876520
10	0.287058	0.881145	0.269996	0.892969	0.266982	0.885509
11	0.288603	0.877013	0.343488	0.829688	0.352273	0.833134
12	0.282838	0.884300	0.291724	0.899219	0.299855	0.901498
13	0.298381	0.875644	0.293942	0.890625	0.295506	0.881978
14	0.290893	0.871416	0.265116	0.895312	0.290788	0.850685
15	0.274518	0.878252	0.235879	0.912500	0.238690	0.906614
16	0.282342	0.881485	0.329287	0.876563	0.315328	0.891759
17	0.276873	0.883268	0.306757	0.872656	0.254698	0.897603
18	0.276263	0.887944	0.292161	0.878125	0.225362	0.914812
19	0.276330	0.886571	0.304021	0.887500	0.304147	0.886301
20	0.278331	0.887548	0.292647	0.868750	0.303910	0.858540
21	0.268764	0.885972	0.283627	0.870313	0.302372	0.864384
22	0.283853	0.879044	0.310679	0.853125	0.278000	0.866353
23	0.288692	0.875363	0.293620	0.874219	0.282479	0.885895
24	0.276108	0.886985	0.265347	0.881250	0.295415	0.872603
25	0.275990	0.884977	0.265981	0.903906	0.271671	0.894114

Таким образом мы обучили нашу модель до точности 88-90% на всех выборках, что можно считать приемлемой точностью. Это значит, что по 9 из

10 снимков мы получим правильно поставленный диагноз. Наша модель не смогла преодолеть потолок в 90% точности и за 26 эпох не было заметного прогресса. Для достижения большей точности необходимо еще больше экспериментировать как с оптимизаторами и функциями потерь, так и с архитектурой самой модели. Но мы остановимся на этом результате.

Осталось лишь сохранить нашу модель для дальнейшего использования. Далее, на основе уже обученной модели мы создадим простое веб-серверное приложение.

Листинг 19:

Ввод:

```
from pathlib import Path
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True,  exist_ok=True )
MODEL_NAME = "pytorch_computer_vision_model_3_0.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_exemplar_2.state_dict(),  f=MODEL_SAVE_PATH)
```

Вывод:

```
Saving model to: models/pytorch_computer_vision_model_3_0.pth
```

1.5 Создание веб серверного приложения на основе обученной модели для выполнения задач классификации.

Теперь приступим к последнему этапу выполнения курсовой работы и попробуем применить нашу обученную модель на практике и создать простой веб-сервис, в котором можно будет получить довольно точное предсказание диагноза по загруженному на сайт снимку. Для создания приложения будем использовать фреймворк Flask. Также будем использовать для верстки

HTML? CSS, JS но для разработки первого прототипа не будем уделять много внимания на дизайне, сосредоточась на функционале.

Создадим папку с проектом и перейдем в нее. Далее создадим новое виртуальное окружение Python и установим в него все необходимые библиотеки следующими командами:

python -m venv venv

venv\Scripts\activate

pip install Flask

pip install torch torchvision

Для инициализации приложения пропишем следующие команды:

set FLASK_APP=main.py

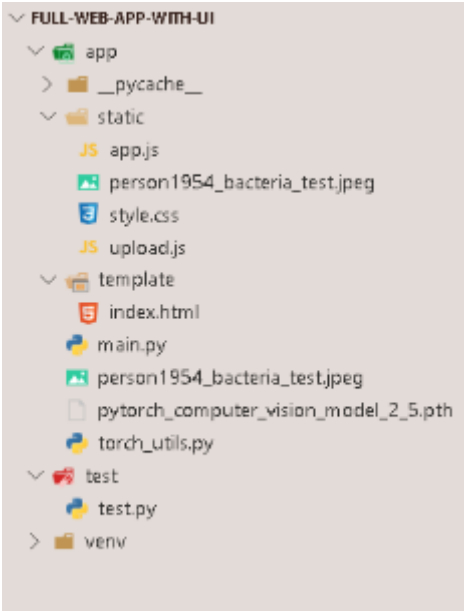
set FLASK_ENV=development

flask --app main.py --debug run

Для запуска приложения на локальном сервере используем команду с включённым дебагером:

flask --app main.py --debug run

Структура проекта имеет такой вид:



Код скрипта main.py, который в процессе обработки подвергся большому количеству изменений:

Листинг 20:

```
from flask import Flask, render_template, request, jsonify, url_for, redirect
from torch_utils import transform_image, get_prediction
import requests
from PIL import Image
import io
import os


app = Flask(__name__, template_folder='template')

UPLOAD_FOLDER = 'static/'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
  return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS



@app.route('/', methods=['GET', 'POST'])
def main():
   return render_template("index.html")

@app.route('/open', methods=['GET', 'POST'])
def open_file():
    return redirect(url_for('main'))

@app.route('/predict', methods=['GET', 'POST'])
def predict():
  if request.method == 'POST':

    if 'file' not in request.files:
          return jsonify({'error': 'File is not found'})

    file = request.files['file']

    if file is None or file.filename == "":
        return jsonify({'error': 'No file yet'})

    if not allowed_file(file.filename):
        return jsonify({'error': 'This file format is not supported. Please ch00se one
from this: png, jpg, jpeg'})

    try:

       img_path = "static/" + file.filename
       print("img_path", img_path)

       with open(file.filename, 'rb') as file:
          image_bytes = file.read()
          print("image_bytes", image_bytes[:20])

       tensor = transform_image(image_bytes)
       print("tensor", tensor)
       print("tensor.shape", tensor.shape)
       prediction = get_prediction(tensor)

       dictionary = {0 : 'NORMAL', 1 : 'PNEUMONIA'}
```

```
        if (dictionary[prediction.item()] == 'PNEUMONIA'):
            pred_text = "---RU--- Основываясь на полученном опыте в результате обучения
могу с высокой долей вероятности поставить диагноз пневмонии. На данном рентеновском
 снимке грудной клетки мной были распознаны аномальные затемнения в легких, которые
сведетельствуют о наличии бактериальной или вирусной пневмонии. Для более точного дианоза
незамедлительно обратитесь к вашему лечащему врачу.       --- EN ---    Based on the
experience gained as a result of the training, I can diagnose pneumonia with a high
degree of probability. In this X-ray of the chest, I recognized abnormal opacifications
in the lungs, which indicate the presence of bacterial or viral pneumonia. For a more
accurate diagnosis, contact your doctor immediately."

            return render_template("index.html", prediction = pred_text,  img_path =
img_path)


        elif (dictionary[prediction.item()] == 0):
            pred_text = "---RU---  Могу вас поздравить  с отсутствием пнемвонии. Данный
рентеновский  снимок является нормальным с высокой долей вероятности. Рентгенограмма
грудной клетки показывает чистые легкие без каких-либо областей аномального затемнения на
изображении. Для более точного дианоза в случае сомнений обратитесь к вашему лечащему
врачу.     /n --- EN ---   I can congratulate you on the absence of pneumonia. This x-ray
is normal with a high degree of probability. The chest x-ray shows clear lungs without
any areas of abnormal shading on the image. For a more accurate diagnosis, if in doubt,
contact your doctor."

            return  render_template("index.html", prediction = pred_text,  img_path =
img_path)


if __name__=="__main__":
  app.run(debug=True)
```

В этом скрипте создается объект Flask с названием приложения и указанием папки с шаблонами. Далее устанавливаются параметры загрузки файлов: путь для сохранения загруженных изображений - статическая переменная UPLOAD_FOLDER и разрешенные расширения файлов - статическая переменная ALLOWED_EXTENSIONS и определяется функция для проверки расширения загруженного файла.

Следующим шагом задаются три маршрута, которые отвечают за главную страницу приложения, открытие файлов и предсказание диагноза.

Если на маршрут "/predict" приходит запрос методом POST, то в форме передается файл, который проверяется наличием и разрешенным расширением. Если загруженный файл проходит проверку, то сначала создается путь для сохранения изображения в папке "static", затем происходит открытие файла и чтение его байтов.

Чтобы сделать предсказание диагноза, считанные байты преобразуются в тензор с помощью функции "transform_image" из модуля torch_utils.py, листинг которого будет приведен ниже, а затем передаются в функцию

"get_prediction", которая возвращает 0 или 1 в зависимости от диагноза (ее функционал также вынесен в модуль torch_utils.py.

Последним шагом создается словарь, который соотносит числовые значения предсказания с текстовыми значениями: 0 соответствует "NORMAL", 1 - "PNEUMONIA". Если модель диагностирует наличие пневмонии, то формируется соответствующий текст на двух языках, иначе - текст о том, что пневмонии нет. Текст отображается на главной странице. Если название файла или расширение не пройдут проверку, то на странице отобразится соответствующее сообщение об ошибке. Если же на маршрут "/open" приходит запрос методом POST, то пользователь перенаправляется на главную страницу. В конце скрипта приложение запускается в отладочном режиме.

Для большей гибкости и модульности приложения, как говорилось ранее, реализации функций трансформации и предсказания transform_image() и get_prediction() соответственно, а также загрузки предобученной модели были вынесены в отдельный скрипт torch_utils.py:

Листинг 21:

```
import io
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from PIL import Image

class EightModel(nn.Module):
    def __init__(self) -> None:
        super(EightModel, self).__init__()

        self.layer_1 = nn.Sequential(
            nn.Conv2d(in_channels=1,
                      out_channels=8,
                      kernel_size=(3, 3),
                      stride=1,
                      padding=1),
            nn.BatchNorm2d(8),
            nn.Dropout(p=0.2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2)
        )

        self.layer_2 = nn.Sequential(
            #Input = 16 x 50 x 50, Output = 16 x 25 x 25
            nn.Conv2d(in_channels=8,
                      out_channels=8,
                      kernel_size = (3,3),
                      stride = 1,
                      padding = 1),
            nn.BatchNorm2d(8),
            nn.Dropout(p=0.2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2)
```

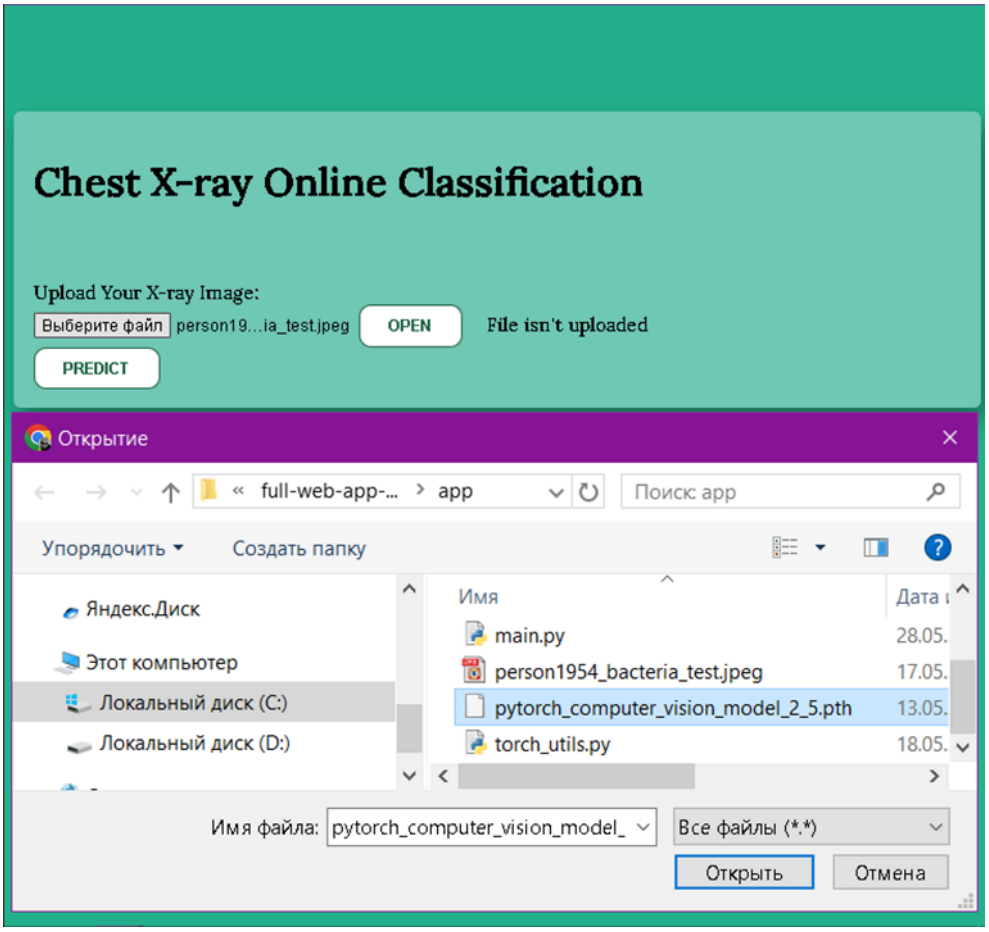
```
        )

        self.fc_2 = nn.Sequential(
            nn.Flatten(),
            nn.Dropout(p=0.5),
            nn.Linear(in_features=1800, out_features=100),
            nn.ReLU()
        )

        self.fc_4 = nn.Sequential(
            nn.Linear(in_features=100, out_features=2),

        )

    def forward(self, x: torch.Tensor):
        out = self.layer_1(x)
        out = self.layer_2(out)

        out = self.fc_2(out)
        out = self.fc_4(out)

        #print(x.shape)
        return out

torch.manual_seed(100)
model_exemplar = EightModel()
print(model_exemplar) #######----


MODEL_PATH_NAME = 'pytorch_computer_vision_model_2_5.pth'
model_exemplar.load_state_dict(torch.load(MODEL_PATH_NAME))
model_exemplar.eval()


#image -> tensor
def transform_image(image_bytes):
    transform_fn = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),
        transforms.Resize((60, 60)),
        transforms.ToTensor()
    ])

    image_for_prediction = Image.open(io.BytesIO(image_bytes))#.seek(0)
    return transform_fn(image_for_prediction).unsqueeze(dim=0)

def get_prediction(image_tensor):
    with torch.no_grad():
        output = model_exemplar(image_tensor)
        _, predicted = torch.max(output.data, 1)
        return predicted
```

Вначале мы определяем класс модели нейронной сети "EightModel" и создаем ее экземпляр. Далее мы загружаем сохраненную предобученную модель и устанавливаем ее экземпляр в режим обучения. В функции transform_image() мы преобразуем поданные на вход байты изображения, приводим их к черно-белому формату, изменяем размер изображения и в

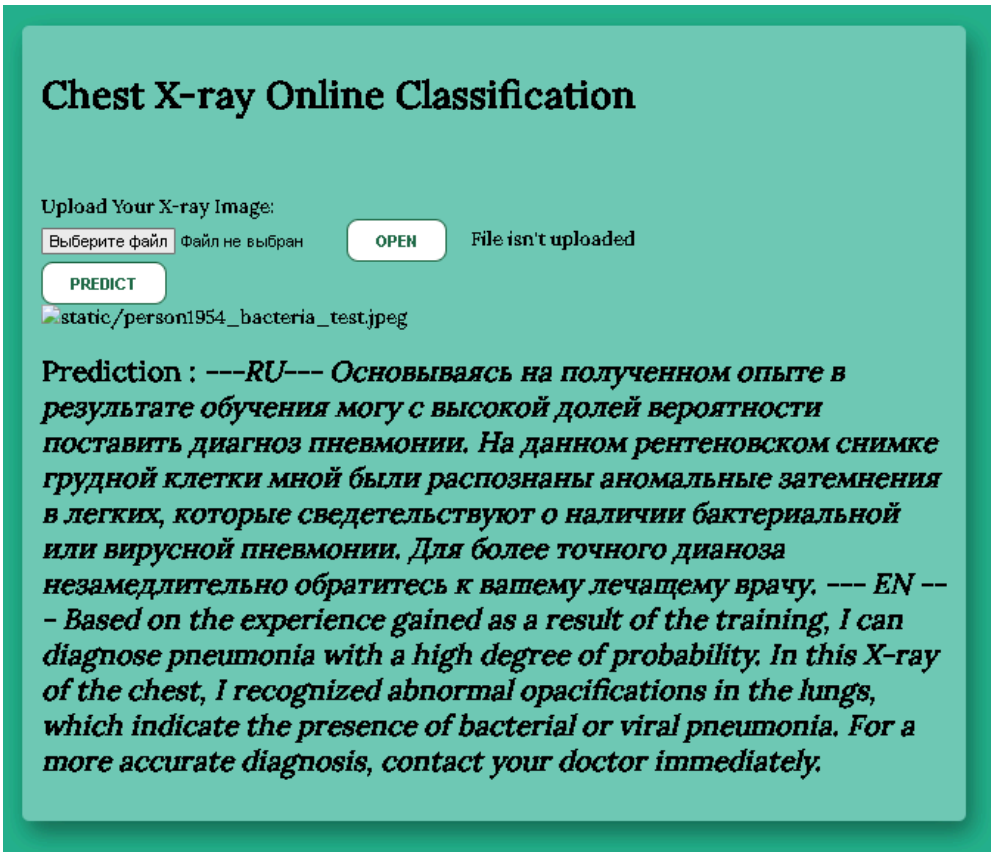
конце трансформируем в тензор. В конце создаем функцию get_prediction(), которая выдает предсказание на основе обработанного тензора изображения и загруженной модели. В функции get_prediction() происходит перевод тензора в набор прогнозов, выбирается прогноз с наивысшей оценкой в качестве окончательного предсказания и возвращается его индекс.

На основе этого индекса мы по итогу в главном скрипте и выбираем, какой диагноз вывести на основе полученного снимка.

Запустим наше приложение. Выберем изображение и нажмем предсказать. Дизайн сайта еще не финальный.



На выходе получим такое предсказание:


Chest X-ray Online Classification
Upload Your X-ray Image:
Выберите файл
Файл не выбран
OPEN
File isn't uploaded
PREDICT
static/person1954_bacteria_test.jpeg
Prediction : ---RU--- Основываясь на полученном опыте в результате обучения могу с высокой долей вероятности поставить диагноз пневмонии. На данном рентеновском снимке грудной клетки мной были распознаны аномальные затемнения в легких, которые сведетельствуют о наличии бактериальной или вирусной пневмонии. Для более точного дианоза незамедлительно обратитесь к вашему лечащему врачу. --- EN --- Based on the experience gained as a result of the training, I can diagnose pneumonia with a high degree of probability. In this X-ray of the chest, I recognized abnormal opacifications in the lungs, which indicate the presence of bacterial or viral pneumonia. For a more accurate diagnosis, contact your doctor immediately.

ЗАКЛЮЧЕНИЕ

Таким образом в ходе выполнения курсовой работы были изучены способы создания и обучения нейронной сети с помощью библиотеки PyTorch, а также способы практического применения построенной модели по средством создания веб-приложения с помощью фреймворка Flask. Были освоены различные способы загрузки, анализа и визуализации данных, построения архитектур нейронных сетей и их обучения, оценки результатов обучения и реального применения обученной модели через создание веб-приложения. В ходе этой работы, были изучены загрузка данных из директорий, их преобразование в PyTorch тензора, создание модели через построение архитектуры нейросети, определение функции потерь, использование оптимизатора, обучение модели, оценка ее действий и анализ результатов. Были усовершенствованы знания по визуализации данных и графиков, что помогало лучше понять закономерности в данных и обучении, работе модели. Фреймворк Flask был использован с целью создания веб-приложения, как мост между пользователем и обученной нейронной сетью. Веб-приложение включало в себя интерфейс, где пользователь мог загрузить изображение и получить ответ модели относительно анализируемого изображения. С помощью Flask мы смогли создать серверную сторону приложения, которая запускала обученную модель и возвращала её результат пользователю.

СПИСОК ЛИТЕРАТУРЫ:

1) Орелье Жерон - «Прикладное машинное обучение Scikit-Learn и TensorFlow - концепции, инструменты и техники для создания интеллектуальных систем.»

2) Посполит – «Основы искусственного интеллекта в примерах на Python.»

3) Питер Истман, Патрик Уолтерс – «Глубокое обучение в биологии и медицине».

4) Эли Стивенс, Лука Антига – «PyTorch.Освещая глубокое обучение.»

5) Джон Крон, Аглаэ Бассенс – «Глубокое обучение в картинках. Визуальный гид по искусственному интеллекту.»