Университет Иннополис

Институт дополнительного образования

Отчёт

по результатам итоговой аттестации

Создание и тестирование приложения для классификации изображений в промышленности с помощью машинного обучения

Студент: Шаверин Никита Владимирович

Томск - 2024 г.

СОДЕРЖАНИЕ

1. Введение 3
2. Постановка задачи 3
3. Обзор моделей 3
4. Описание процесса решения 7
5. Результаты 10

Программный код, модели, отчёт и презентация находится по адресу: https://github.com/ShaoYoung/57_Innopolis_final_certification.

Telegram-бот - https://t.me/phonebook_221025_bot.

1. Введение

Машинное обучение (ML) – это направление искусственного интеллекта, сосредоточенное на создании систем, которые обучаются и развиваются на основе получаемых ими данных. В настоящее время используются два основных типа машинного обучения: контролируемое обучение (обучение с учителем) и самостоятельное обучение (обучение без учителя). Обучение с учителем используется наиболее часто.

Обучение с учителем подразумевает наличие результатов, которые должен получить алгоритм ML. Алгоритм практикуется на специальным образом маркированных наборах данных с предопределёнными результатами. Для данного типа обучения используются такие алгоритмы как линейная и логистическая регрессии, бинарная и мультиклассовая классификации, метод опорных векторов.

Классификация изображений является важной задачей в компьютерном зрении, поскольку она используется для идентификации объекта, который появляется на изображении. Эта задача состоит в маркировке входных изображений с вероятностью присутствия определённого класса объектов.

2. Постановка задачи

Целью настоящей работы является разработка, обучение, сравнение и тестирование моделей машинного обучения, а также создание, настройка и тестирование приложения для классификации изображений.

3. Обзор моделей

Для решения поставленной задачи были выбраны три модели: линейная (Linear Model), свёрточная (CNN) и ResNet-модель (ResNet50).

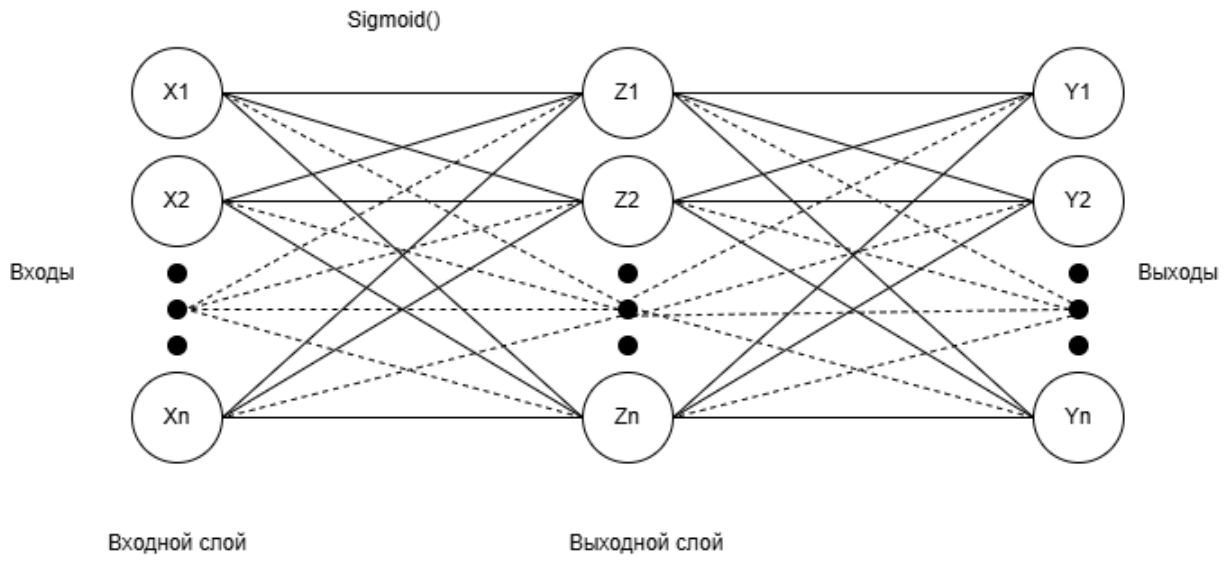
Линейная модель – это модель машинного обучения, основанная на предположении, что зависимость в наблюдаемых данных можно описать простой прямой:

$Y = a + b*X$, где a – свободный член, b – угловой коэффициент

Линейные модели относительно просты и дают легко интерпретируемую математическую формулу для скоринга. Однако они хорошо работают только на очень простых зависимостях.

В настоящей работе использована следующая двухслойная линейная модель:

```
class LinearModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc_1 = nn.Linear(3*params_dict['image_height'] * params_dict['image_width'], 100)
        self.activation = nn.Sigmoid()
        self.fc_2 = nn.Linear(100, len(class_names))
    def forward(self, x: torch.Tensor):
        x = x.view((-1, 3 * params_dict['image_height'] * params_dict['image_width']))
        x = self.fc_1(x)
        x = self.activation(x)
        x = self.fc_2(x)
        return x
```



Изображение преобразуется в вектор размерностью 3 * высота изображения * ширина изображения. Размерность тензора на выходе первого слоя 100. Функция активации Sigmoid(). На выходе второго слоя тензор, размерность которого равна количеству классов в датасете.

Свёрточная модель (свёрточная нейронная сеть) – это класс нейронных сетей, который специализируется на обработке изображений и видео. Такие сети хорошо улавливают локальный контекст, когда информация в пространстве непрерывна, т.е. её носители находятся рядом. Например, пиксели изображения, которые расположены близко друг к другу и содержат визуальные данные: яркость и цвет.

Свёрточная сеть состоит из нескольких слоёв. Чем больше слоёв, тем мощнее архитектура и лучше обучаемость нейросети. Основные элементы CNN: свёрточный слой, пулинг, нормализация по батчу, полносвязный слой.

Свёрточные нейронные сети обладают рядом неоспоримых преимуществ, таких как инвариантность к сдвигу (CNN может распознавать объекты независимо от их местоположения), общее использование параметров (один и тот же набор параметров применяется для всех частей входного изображения), иерархические представления (позволяют моделировать сложные структуры данных), устойчивость к изменениям, возможность изучения различных уровней входного изображения и т.д.

К недостаткам CNN можно отнести необходимость большого объёма размеченных данных для обучения и, как следствие, высокие требования к вычислительной мощности. Глубокие архитектуры CNN могут страдать от переобучения.

В настоящей работе использована следующая CNN:

```
class CNNModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.activation = nn.Tanh()
        self.pool = nn.MaxPool2d(kernel_size=(2, 2))
        self.conv_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.conv_3 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
        self.fc = nn.Linear(56*56*32, len(class_names))
    def forward(self, x: torch.Tensor):
```

```
x = self.conv_1(x)
x = self.activation(x)
x = self.pool(x)
x = self.conv_2(x)
x = self.activation(x)
x = self.pool(x)
x = self.conv_3(x)
x = self.activation(x)
x = x.view((-1, 56*56*32))
x = self.fc(x)
return x
```

CNN состоит из трёх свёрточных слоёв (размер свёрточного фильтра 3x3) и одного линейного (выходного). После каждого свёрточного слоя применяется функция активации Tahn() и пулинг MaxPoll2d() с размером ядра 2x2. Перед линейным слоем тензор преобразуется в вектор.

ResNet (Residual Network, остаточная CNN) – модель, использующая идею residual learning (остаточное обучение), согласно которому к выходу каждого блока прибавляется его вход таким образом, что блок выучивает не полноценное новое преобразование входа, а его добавочное преобразование. Решает проблему затухания градиента с ростом глубины свёрточной сети, считается революционным прорывом в области глубокого обучения.

В зависимости от количества слоёв различают следующие типы ResNet сетей:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

В настоящей работе в качестве эталонной модели использована нейросеть resnet50 из модуля torchvision.models фреймворка pytorch.

4. Описание процесса решения

Для решения задачи обучения и тестирования моделей был выбран датасет с ресурса https://www.kaggle.com/datasets/anthonytherrien/image-classification-64-classes-animal/data, содержащий изображения животных, птиц и насекомых, разделённые по классам. Из данного датасета были выделены изображения животных (класс Animals) и птиц (класс Birds). Изображения были размещены на ресурсе https://drive.google.com, который был подключен к среде разработки https://colab.research.google.com, где создавался, тестировался и запускался программный код приложения на языке Python. При этом изображения были сгруппированы по директориям, каждая из которых является классом, а её название – именем класса.

Во время работы над созданием приложения была предпринята попытка классификации изображений людей. Датасетом для данной задачи был специально созданный фотоальбом семьи автора данной работы.

Сводная информация по датасетам приведена в таблице:

Датасет	Кол-во классов датасета	Кол-во файлов датасета	train	test	Размер, гБ	Высота и ширина изображения
Birds	11	2376	1900	476	0,91	512x512
Animals	46	10490	8392	2098	4,22	512x512
Family	5	103	82	21	0,018	1600x900

Основные параметры для работы с датасетом и моделями были собраны в отдельный словарь Python:

```
params_dict = {'num_epochs' : 10,       # количество эпох
               'batch_size' : 64,       # размер batch
               'learning_rate' : 0.01,  # learning_rate
               'device' : torch.device('cuda' if torch.cuda.is_available() else 'cpu'),  # device
               'loss_function' : nn.CrossEntropyLoss(),   # функция потерь
               'image_height' : 224,    # высота image после нормализации
```

'image_width' : 224, # ширина image после нормализации

}

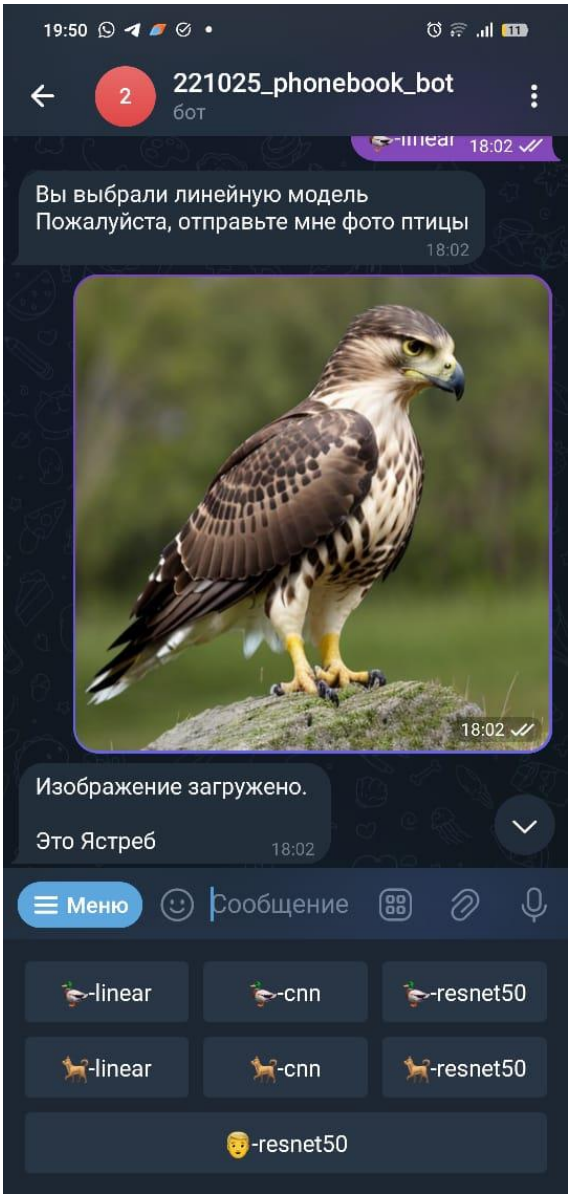
С помощью класса torchvision.datasets.ImageFolder изображения загружались в среду разработки, предварительно проходя процесс нормализации.

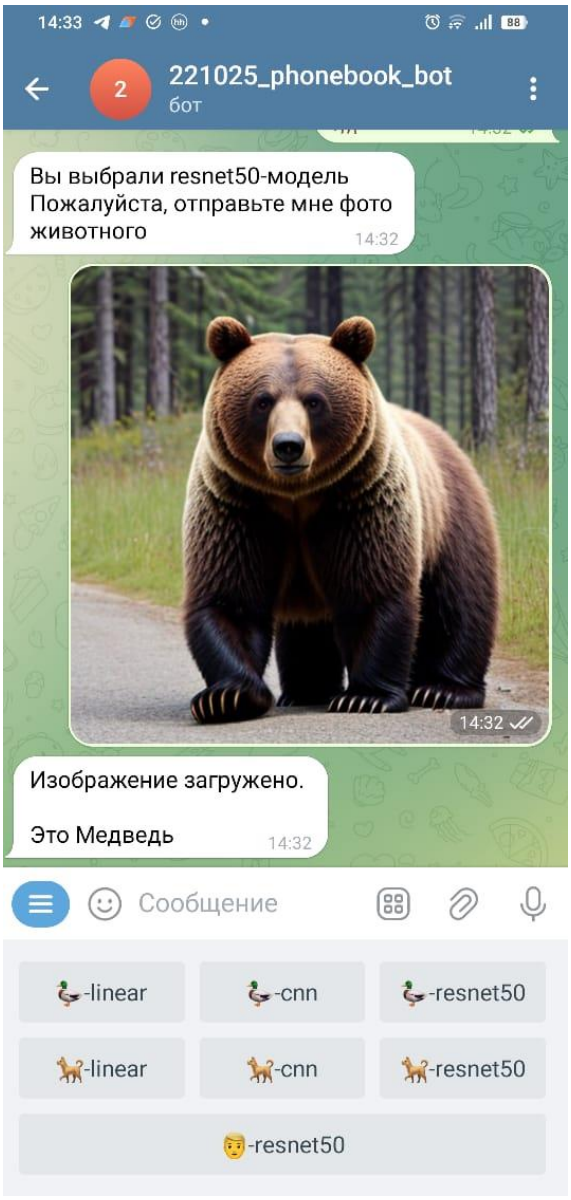
Полученный датасет в пропорции 4:1 был разделён на train и test части, которые были загружены в train_loader и test_loader.

Обучение каждой моделей происходило поочередно на всех датасетах. Оптимайзер – torch.optim.SGD(), реализующий метод градиентного спуска. Одновременно с обучением производился расчёт значений функции потерь (loss_function) и accuracy в конце каждой эпохи. По обученным моделям были собраны метрики, сами модели были сохранены в файлы на ресурсе https://colab.research.google.com.

Пользовательское приложение было реализовано в виде Telegram-бота на языке Python с применением фреймворка aiogram.

При запуске приложения происходит загрузка предварительно обученных моделей в ОЗУ. После выполнения команды /start пользователю предлагается выбрать тип изображений (Animals, Birds, Family) и рабочую модель путём нажатия соответствующей кнопки ReplyKeyboard. По получению от пользователя сообщения о выбранном типе изображений и рабочей модели выбираются соответствующая модель и имена классов и предлагается пользователю загрузить изображение для его классификации.





По получению изображения (отправленного боту из предсохранённого файла или сделанного в online-режиме фото) программа сохраняет его в файл и вызывает функцию определения класса сохранённого изображения. Результа-

том работы является ответное сообщение пользователю в Telegram об определённом приложением классе.

5. Результаты

Результаты обучения моделей приведены в таблицах:

Датасет Birds (device: cuda):

Модель	Эпоха									
	loss_1	loss_2	loss_3	loss_4	loss_5	loss_6	loss_7	loss_8	loss_9	loss_10
	acc_1	acc_2	acc_3	acc_4	acc_5	acc_6	acc_7	acc_8	acc_9	acc_10
Linear	1,86	1,57	1,43	1,33	1,24	1,16	1,1	1,04	0,98	0,93
	47,74	63,79	68,63	70,53	72,47	74,53	76,68	78,47	80,63	81,21
CNN	1,5	0,84	0,67	0,56	0,49	0,41	0,39	0,32	0,28	0,24
	52,42	72,63	77,74	81,58	83,89	87,63	87,68	90,53	91,32	92,58
ResNet50	1,54	0,05	0,02	0,01	0,01	0,01	0,01	0,01	0	0
	78,58	99,42	99,89	99,95	99,89	100	100	100	100	100

Датасет Animals (device: cpu):

Модель	Эпоха									
	loss_1	loss_2	loss_3	loss_4	loss_5	loss_6	loss_7	loss_8	loss_9	loss_10
	acc_1	acc_2	acc_3	acc_4	acc_5	acc_6	acc_7	acc_8	acc_9	acc_10
Linear	3,37	3,07	2,89	2,74	2,61	2,48	2,37	2,27	2,16	2,07
	27,32	40,16	45,66	49,01	52,3	55,74	58,57	60,8	63,94	65,65
CNN	2,14	0,94	0,67	0,53	0,44	0,36	0,29	0,25	0,19	0,16
	43,82	73,53	80,86	84,9	87,09	89,92	91,83	93,01	94,64	97,1
ResNet50	0,98	0,03	0,02	*	*	*	*	*	*	*
	86,87	99,8	99,9	*	*	*	*	*	*	*

* - количество эпох при обучении модели ResNet50 - 3

Датасет Family (device: cpu):

Модель	Эпоха									
	loss_1	loss_2	loss_3	loss_4	loss_5	loss_6	loss_7	loss_8	loss_9	loss_10
	acc_1	acc_2	acc_3	acc_4	acc_5	acc_6	acc_7	acc_8	acc_9	acc_10
ResNet50	7,16	2,27	0,49	0,12	0,1	*	*	*	*	*
	3,66	65,85	98,8	100	100	*	*	*	*	*

Параметры обученных моделей приведены в таблицах:

Датасет Birds (device: cuda):

Модель	Estimated total size, Mb	FileSize, Mb	Accuracy, train	Accuracy, test	Learning time
Linear	58	58,8	0,837	0,67	30:36
CNN	27,04	4,4	0,924	0,821	11:29
ResNet50	384,62	100,3	1	0,994	15:16

Датасет Animals (device: cpu):

Модель	Estimated total size, Mb	FileSize, Mb	Accuracy, train	Accuracy, test	Learning time
Linear	58	58,8	0,66	0,62	1:56:55
CNN	40,44	18,1	0,975	0,83	3:37:19
ResNet50	384,62	100,5	1	0.999	7:79:44*

* - время обучения на трёх эпохах

Датасет Family (device: cpu):

Модель	Estimated total size, Mb	FileSize, Mb	Accuracy, train	Accuracy, test	Learning time
ResNet50	384.6	100,2	1	0,95	08:09

В ходе выполнения работы были сделаны следующие выводы:

- Из трёх тестируемых моделей худшие результаты показала линейная модель. Даже десяти эпох обучения не хватило для получения удовлетворительных метрик. Она наименее предназначена для решения задачи классификации изображений.
- Лучшие результаты ожидаемо показала нейронная сеть ResNet50, однако её обучение потребовало значительных вычислительных ресурсов сервиса Google Colab. Размер файла сохранённой модели также значительно превышает другие модели. Обучение модели с очень неплохими метриками может быть обеспечено за две-три эпохи. Оптимальным решением из рассмотренных по соотношению метрик и вычислительных затрат на обучение представляется свёрточная нейросеть.
- Использование Telegram-бота в качестве пользовательского интерфейса позволяет оперативно проводить классификацию изображений с помощью обученных моделей в определённых типах изображений, т.к. приложение Telegram и фотокамера присутствует в смартфонах подавляющего большинства потенциальных потребителей подобных услуг.
- Использование фреймворка PyTorch при должной подготовке изображений обучающего датасета позволяет использовать разработанное приложение для решения задач классификации иных наборов изображений.
- Развитие направления классификации изображений видится в увеличении обучающей выборки фотографий людей и дополнительном обучении моделей. Полагаю, что для расширения возможностей использования нейросетей в задачах классификации фотографий людей, необходимо предварительно решить задачу кластеризации по выделению из предоставленного изображения области лица человека. Решением этой задачи мне было бы интересно заняться после окончания курса обучения, если, конечно, удастся найти точку приложения сил.