УДК 004.8

Обучения модели для определения класса изображений на примере CIFAR-10

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети для определения определенных классов изображений на примере CIFAR-10. В работе набор данных CIFAR-10 и использовалась модель сверточная нейронная сеть для определения классов изображения. В результате работы модель способен определять классы изображения на примере CIFAR-10, а также в процессе обучения была получена метрика функция потерь и точность предсказания модели.
Ключевые слова: CIFAR-10, классификация данных, сверточная нейронная сеть, Torch.

Training the model to determine the class of images on the example of CIFAR-10

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using a neural network model to determine certain classes of images using CIFAR-10 as an example. In the work, the CIFAR-10 dataset was used and a convolutional neural network model was used to determine image classes. As a result of the work, the model is able to determine image classes on the example of CIFAR-10, and also in the learning process, the metric was obtained, the loss function and the prediction accuracy of the model.
Keywords: CIFAR-10, data classification, convolutional neural network, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в том что оно дает практический опыт начинающим специалистам по данным и инженерам по машинному обучению, которые стремятся работать с крупномасштабными и сложными наборами данных изображений. Работая с набором данных, таким как CIFAR-10, они могут лучше понять сложности, связанные с классификацией изображений. Кроме того, точная классификация изображений является важной задачей в нескольких отраслях, которые

полагаются на анализ и обработку визуальных данных. Поэтому хорошее понимание и практический опыт классификации изображений имеет решающее значение для тех, кто хочет работать в этих областях.

1.2. Цель исследования

Целью работы тренировка модели для определения классы изображений на примере CIFAR-10, и тестирования модели на реальных данных.

1.3. Обзор исследований

Автор работ Б. Речт и др. исследует эффективность классификации сверточных нейронных сетей (CNN) в известном наборе данных CIFAR-10. Авторы оценивают точность каждого метода, используя несколько показателей, таких как точность и время обучения. В конечном итоге результаты показывают, что предварительно обученные CNN в ImageNet обеспечивают наилучшую производительность классификации, а их точная настройка в CIFAR-10 еще больше улучшает результаты. Однако обучение CNN с нуля на CIFAR-10 также позволяет достичь высокой точности [1].

Авторы Ю. Абуэльнаги и др. представили ансамбль классификаторов, основанный на алгоритме K ближайших соседей (KNN) для набора данных CIFAR-10. Подход включает в себя обучение нескольких классификаторов KNN на разных подмножествах данных и объединение их прогнозов посредством голосования. Полученный ансамбль обеспечивает более высокую точность, чем отдельные классификаторы KNN, и превосходит современные методы CIFAR-10. Исследование предполагает, что ансамблевые методы могут повысить производительность простых алгоритмов, таких как KNN, особенно для сложных наборов данных с большим количеством классов [2].

Д. Банкман и др. представляет новый двоичный процессор CNN со смешанными сигналами, который является энергоэффективным и имеет возможность непрерывно обрабатывать данные. Процессор имеет всю память на кристалле и разработан с использованием 28-нм технологии CMOS. Схема способна достичь точности 86% при классификации изображений CIFAR-10 с потребляемой мощностью 3,8 мкДж на вывод, что делает ее подходящей для встроенных приложений, где энергопотребление является проблемой. Использование двоичных весов и активаций снижает требования к памяти модели CNN и помогает в достижении этих энергоэффективных и высокоточных результатов [3].

Ф. О. Джусте, Дж. К. Визкаррой изучают эффективность использования ансамблей признаков для классификации изображений в наборе данных Cifar-10. Исследователи использовали десять различных функций, включая локальные бинарные шаблоны и цветовые гистограммы, для обучения и тестирования различных моделей классификации. Их результаты показывают, что использование комбинации признаков с помощью ансамблей может повысить точность классификации. Кроме того, они обнаружили, что использование нескольких классификаторов в сочетании с наборами функций может еще больше повысить производительность. Эти результаты показывают, что использование ансамблей признаков и нескольких

классификаторов может повысить точность классификации изображений для сложных наборов данных изображений, таких как Cifar-10 [4].

Т. М. Х. Хсу, Х. Ци, М. Браун исследовали влияние неидентичного распределения данных на объединенную визуальную классификацию. Авторы работ использовали набор данных, состоящий из изображений из нескольких источников, и сравнили точность централизованной модели с федеративной моделью, в которой обучающие данные были распределены по нескольким устройствам. Они обнаружили, что неидентичное распределение данных негативно повлияло на точность федеративной модели, что привело к снижению производительности. Кроме того, они обнаружили, что корректировка веса данных, используемых для обучения на каждом устройстве, повышает точность федеративной модели. В этом исследовании подчеркивается важность рассмотрения распределения данных в федеративном обучении и дается представление о том, как смягчить его негативное влияние на производительность модели [5].

Н. Шарма, В. Джайн, А. Мишра провели тщательный анализ сверточных нейронных сетей (CNN) для классификации изображений. Они сравнили различные архитектуры CNN, включая VGG16, ResNet50 и InceptionV3, на наборе данных из 10 000 изображений из десяти различных категорий. Авторы работ оценили производительность каждой сети на основе точности, достоверности, полноты и F1-показателя. Они также изучили влияние различных гиперпараметров, таких как количество слоев, скорость обучения и размер партии. В целом работы авторов дает ценную информацию об использовании CNN для классификации изображений. Их анализ различных архитектур и гиперпараметров может помочь исследователям и практикам в выборе наиболее подходящей модели для их конкретных задач классификации [6].

2. Рабочий процесс

2.1. Набор данных

Исходный данный представлено CIFAR-10 [7], а внутри наборе данных содержится изображения размера 32x32 с определенным классам, всего содержится классов 10. Обработка изображения выполнялась только преобразования в тензор для дальнейшей тренировки нейронным сетям.

Таблица 1. Список данных по классам

Номер класса	Названия класса	Рисунок
0	airplane	
1	automobile	
2	bird	

3	cat	
4	deer	
5	dog	

6	frog	
7	horse	
8	ship	

9	truck	

2.2. Модель

Модель классификации Cifar-10 включает использование сверточных нейронных сетей (CNN) и методов глубокого обучения. Он состоит из нескольких слоев свертки, объединения и полностью связанных слоев.

Входной слой принимает изображения размером 32x32 пикселя и 3 каналов и пропускает их через серию сверточных слоев для извлечения признаков. Полученные признаков затем обрабатываются путем объединения слоев для увеличения количество каналов. Выходные данные последнего объединяющего слоя затем выравниваются и передаются в линейный слои для создания распределения вероятностей по десяти классам изображений Cifar-10.

Листинг 2.1. Структура модели.

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [128, 32, 32, 32]             896
            ReLU-2         [128, 32, 32, 32]               0
          Conv2d-3         [128, 64, 32, 32]          18,496
            ReLU-4         [128, 64, 32, 32]               0
       MaxPool2d-5         [128, 64, 16, 16]               0
          Conv2d-6        [128, 128, 16, 16]          73,856
            ReLU-7        [128, 128, 16, 16]               0
          Conv2d-8        [128, 128, 16, 16]         147,584
            ReLU-9        [128, 128, 16, 16]               0
      MaxPool2d-10         [128, 128, 8, 8]               0
         Conv2d-11         [128, 256, 8, 8]         295,168
           ReLU-12         [128, 256, 8, 8]               0
         Conv2d-13         [128, 256, 8, 8]         590,080
           ReLU-14         [128, 256, 8, 8]               0
      MaxPool2d-15         [128, 256, 4, 4]               0
        Flatten-16              [128, 4096]               0
         Linear-17              [128, 1024]       4,195,328
           ReLU-18              [128, 1024]               0
         Linear-19               [128, 512]         524,800
           ReLU-20               [128, 512]               0
         Linear-21                [128, 10]           5,130
================================================================
Total params: 5,851,338
Trainable params: 5,851,338
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 419.01
```

```
Params size (MB): 22.32
Estimated Total Size (MB): 442.83
----------------------------------------------------------------
```

```
Cifar10CnnModel(
        (network): Sequential(
                (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1): ReLU()
                (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (3): ReLU()
                (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (6): ReLU()
                (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (8): ReLU()
                (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (11): ReLU()
                (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (13): ReLU()
                (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (15): Flatten(start_dim=1, end_dim=-1)
                (16): Linear(in_features=4096, out_features=1024, bias=True)
                (17): ReLU()
                (18): Linear(in_features=1024, out_features=512, bias=True)
                (19): ReLU()
                (20): Linear(in_features=512, out_features=10, bias=True)
        )
)
```

Листинг 2.2. Параметры для обучения модели

```
1  self.criterion = nn.CrossEntropyLoss().to(self.device)
2  self.lr = 0.001
3  self.model = {
4          "cnn": Cifar10CnnModel().to(self.device)
5  }
6  self.optimizer = {
7          "cnn": torch.optim.Adam(self.model["cnn"].parameters(), lr=self.lr)
8  }
```

Строка 1. Метрика CrossEntropyLoss для классификация класса.
Строка 2. Скорость обучения.
Строка 3 — 5. Модель.
Строка 6 — 8. Оптимизатор Adam.

2.3. Обучение

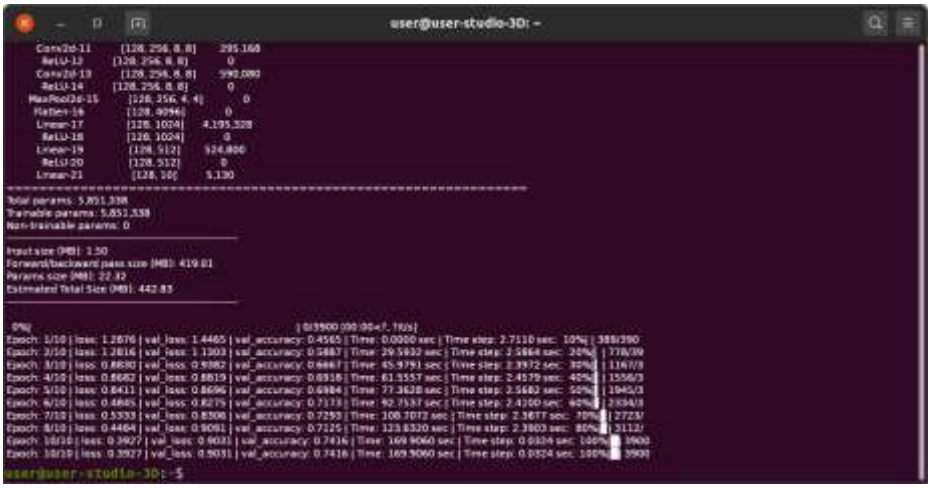

Рисунок 1. Обучение модели

В результате тренировки нейронных сеть, для оценки прогнозирования тестовых признаков удалось достичь по точности (accuracy) 74,16, а функция потери (loss) — 0.903, а функция потерь по обучающий признаков — 0,393. Обучения выполнялось 10 эпох, 3900 циклов. Для оценки обучения модели использовалась метрика Cross Entropy.

Листинг 2.3. Функция для обучение модель

```
1  def step_train(self, sel: any, index :int) -> None:
2          super().step_train(sel, index)
3          images, labels = sel
4          images = images.to(self.device)
5          labels = labels.to(self.device)
6
7          outputs = self.model["cnn"](images)
8          loss = self.criterion(outputs, labels)
9          loss.backward()
10         self.optimizer["cnn"].step()
11         self.optimizer["cnn"].zero_grad()
12
13         if self.metric.Step % (len(self.datasets[0].dataLoader) - 1) == 0:
14                 self.model["cnn"].eval()
15                 m = self.test_model()
16                 m_data = {key: [dic[key] for dic in m] for key in m[0]}
17
18                 self.metric.val_accuracy = torch.stack(m_data["val_acc"]).mean().item()
19                 self.metric.val_loss = torch.stack(m_data["val_loss"]).mean().item()
20                 self.model["cnn"].train()
21         self.metric.loss = loss.item()
```

Строка 2, заглушка.

Строка 3 — 5, загрузка данных, images — изображений, labels — метки (номер класса, число).

Строка 7 — 11. обучаем модель.

Строка 13 — 20. оцениваем модель по тестовому признаку, это происходит до конца эпохи обучения (достижения 389 цикл обучения).

Листинг 2.4. Оценка модели по тестовому признаку и оценка точность (accuracy) модели.

```
1  def test_model(self):
2          ls = deque()
3          for images, labels in self.datasets[1].dataLoader:
4                  images = images.to(self.device)
5                  labels = labels.to(self.device)
6                  out = self.model["cnn"](images)
7                  loss = self.criterion(out, labels)
8                  acc = self.accuracy(out, labels)
9                  ls.append({'val_loss': loss.detach(), 'val_acc': acc})
10         return ls
11 def accuracy(self, outputs, labels):
12         _, preds = torch.max(outputs, dim=1)
13         return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

Строка 2 — список, пара метрика функция потерь и точность (accuracy).

Строка 3 итерация по тестовому признаки.

Строка 4 — 5 загрузка данные в устройства (в данной случае использовался в GPU).

Строка 6 предсказания модели.

Строка 7 — 8 оценка функция потерь и точность (accuracy).

Строка 9 — добавление в список

Строка 10 возврат значение в виде списка.

Строка 11 функция для оценки сравнения прогнозируемый (полученный результат модели) и исходный данный.

2.4. Оценка модели

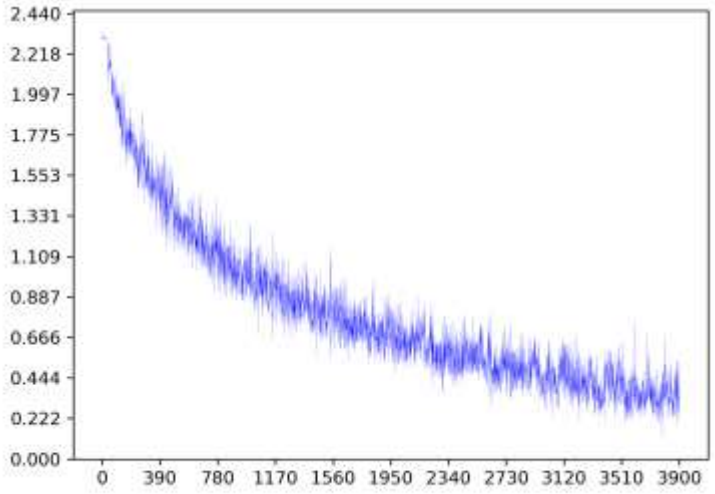

Рисунок 2. Функция потерь (метрика Cross Entropy)

На графике содержится значение функция потерь по осью Y, а по осью X — цикл обучения. Точка минимума достигла функция потерь 0,126, цикл обучения — 3782. Последний цикл обучения была достигнута функция потерь — 0,393.

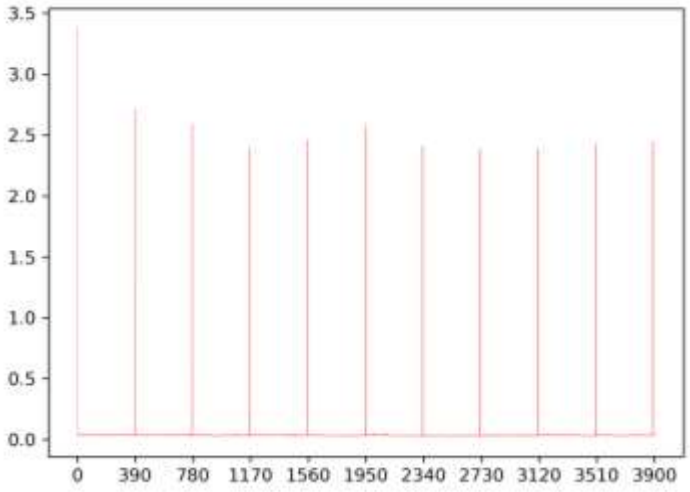

Рисунок 3. Время обучения

Время обучения в секундах представлен на графике с обработкой признаков, в первый выполнялось обработка и загрузка тренировочные признаки, а последующие задержки выполнялось оценка и тестирования модели, каждый раз выполнялось загрузка и обработка тестовых признаков.

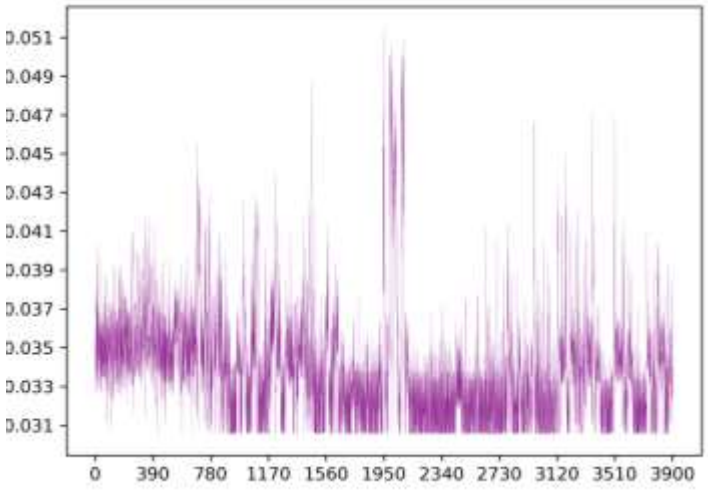

Рисунок 4. Время обучения без обработки признаков

На графике представлен время обучения без обработки признаков с отсечение порога больше 1 секунда. Время обучения с отсечение 131,893 секунда (2 минута, 11,893 секунда). В среднем 0,03391 секунда.

Полная время обучения 160,049 (2 минута, 40,049 секунда). В среднем — 0,04104 секунда.

Время задержки обработкой признаков 28,156 секунда. В среднем — 2,5596 секунда.

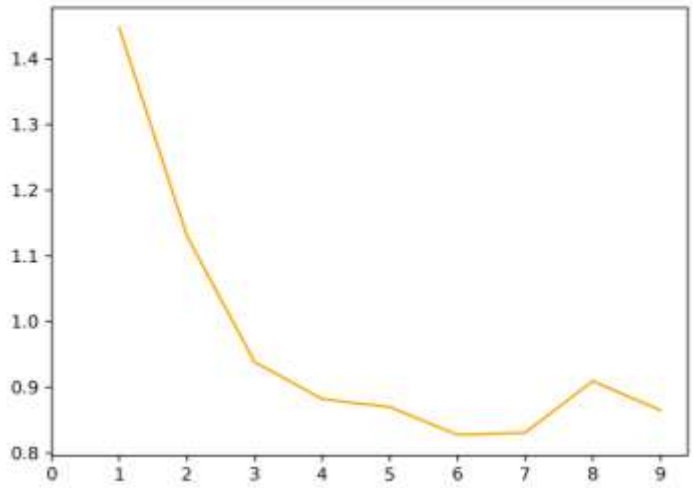

Рисунок 5. Функция потерь в тестовых признаков

По оси X — эпоха, а по оси Y — функция потерь, оценка выполнялось по тестовому признаку, метрика использовалась Cross Entropy, минимальное значение потерь 0,57 в эпохе 6. Последняя эпоха достигнута — 0,865.

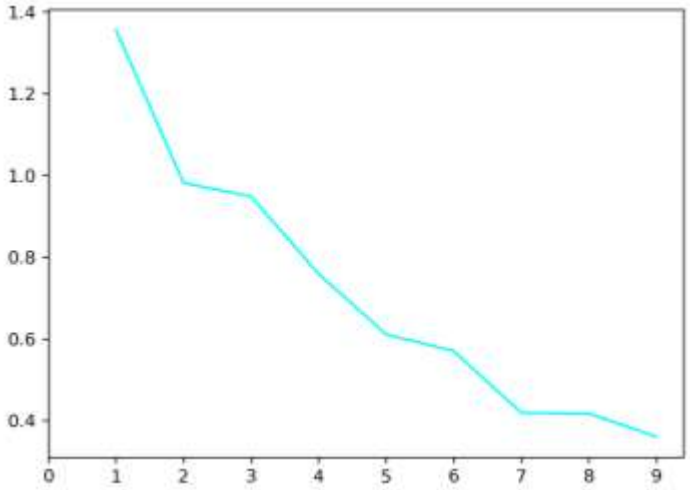

Рисунок 6. Функция потерь в обучающий признаков

Минимальное значение потерь 0,361 в обучающий признаков последний эпоха.

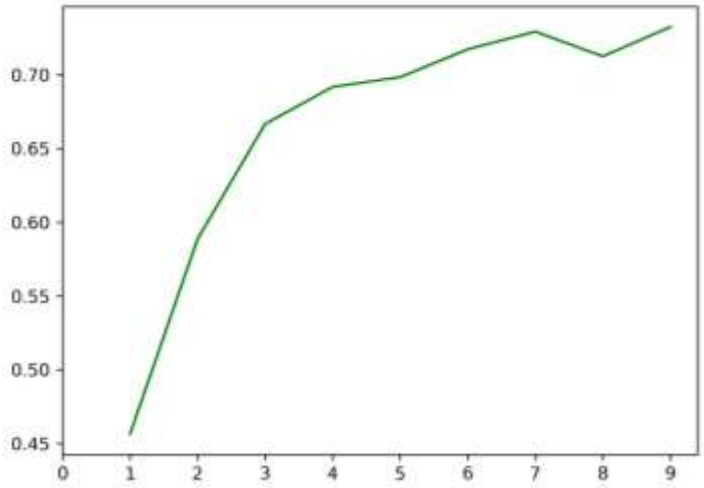

Рисунок 7. Точность (accuracy) в тестовых признаков

Оценка выполнялось по тестовому признаку, для оценки модели использовалась сравнения прогнозируемые и тестовая признаки а затем получили среднее значение, максимальное значение удалось достичь — 0,732 в последний эпоха.

2.5. Предсказание

Рисунок 8. 32 изображений для тестирования

Набор данных взят с сайта wallhaven [8]. В наборе данных содержится 32 изображения, и мы вручную разметили данных каждого изображения для оценки модели в строке 2 — 33 (листинг 2.5).

Листинг 2.5. Функция для предсказания модели

```
1  def predict(self, img):
2          self.model["cnn"].to("cpu").eval()
3          img = self.transform(transforms.Resize([32, 32])(img)).unsqueeze(0)
4          _, preds = torch.max(self.model["cnn"](img), dim=1)
5          return self.label_class(preds)
```

Строка 1. функция выполняет предсказание модели

Строка 2. переходит модель на CPU и режим выполнение.

Строка 3. изменения размера изображения до 32x32 и преобразования в тензор.

Строка 4. выполняет, определение класса изображения.

Строка 5. возвращает название класса.

Листинг 2.6. Использование для предсказания на реальных данных

```
1  cl_data = {
2          "/wallhaven/wallhaven-48gk7k.jpg":"2",
3          "/wallhaven/wallhaven-nkype1.jpg":"2",
4          "/wallhaven/wallhaven-48jj2k.jpg":"2",
5          "/wallhaven/wallhaven-odqg69.jpg":"2",
6          "/wallhaven/wallhaven-392r63.jpg":"2",
7          "/wallhaven/wallhaven-zmdj5v.jpg":"2",
8          "/wallhaven/wallhaven-eooryk.jpg":"2",
9          "/wallhaven/wallhaven-0wm8mx.jpg":"2",
10         "/wallhaven/wallhaven-x18ywz.jpg":"4",
11         "/wallhaven/wallhaven-lqw1qr.jpg":"4",
12         "/wallhaven/wallhaven-01r6q1.jpg":"4",
13         "/wallhaven/wallhaven-2kxypy.jpg":"4",
14         "/wallhaven/wallhaven-45z227.jpg":"5",
15         "/wallhaven/wallhaven-4377py.jpg":"5",
16         "/wallhaven/wallhaven-gjo3o3.jpg":"5",
17         "/wallhaven/wallhaven-eol9p8.jpg":"5",
18         "/wallhaven/wallhaven-zx6q3o.jpg":"5",
19         "/wallhaven/wallhaven-k91537.jpg":"6",
20         "/wallhaven/wallhaven-4xpmqo.jpg":"6",
21         "/wallhaven/wallhaven-4dk55l.jpg":"8",
22         "/wallhaven/wallhaven-nzqxgj.jpg":"8",
23         "/wallhaven/wallhaven-n6ld1x.jpg":"9",
24         "/wallhaven/wallhaven-j58lep.jpg":"9",
25         "/wallhaven/wallhaven-q6pr9q.jpg":"9",
26         "/wallhaven/wallhaven-v9k8l5.jpg":"9",
27         "/wallhaven/wallhaven-285me6.jpg":"9",
28         "/wallhaven/wallhaven-lmolml.jpg":"3",
29         "/wallhaven/wallhaven-4yx56l.jpg":"3",
30         "/wallhaven/wallhaven-47pv5e.jpg":"3",
31         "/wallhaven/wallhaven-wyml1p.jpg":"3",
32         "/wallhaven/wallhaven-r22zyj.jpg":"3",
33         "/wallhaven/wallhaven-9558qd.jpg":"1"
34 }
35 l_answer = deque()
36 for i in cl_data:
37         img = app.load_image(url_image + "/" + i)
38         y = app.predict(img)
39         label = app.label_class(int(cl_data[i]))
40         print("src: %s, correct: %s, model: %s, correct answer: %s" % (
41                 i, label, y, label == y
42         ))
43         l_answer.append(label == y)
44 n_answer = np.array(l_answer)
45 print("accuracy: %s" % (n_answer[n_answer == True].sum() / len(n_answer)))
```

В строке 36 — 43, выполняет предсказания модели для определения класса, а в строке 44 — 45, выводим оценку количество правильных ответов соотношение всего признаков, оценка точность (accuracy).

Таблица 2. Вывод отчета для оценки модели

Src	Correct	Model	Correct answer
/wallhaven-48gk7k.jpg	bird	deer	False
/wallhaven-nkype1.jpg	bird	bird	True
/wallhaven-48jj2k.jpg	bird	bird	True
/wallhaven-odqg69.jpg	bird	bird	True
/wallhaven-392r63.jpg	bird	cat	False
/wallhaven-zmdj5v.jpg	bird	bird	True
/wallhaven-eooryk.jpg	bird	bird	True
/wallhaven-0wm8mx.jpg	bird	airplane	False
/wallhaven-x18ywz.jpg	deer	cat	False
/wallhaven-lqw1qr.jpg	deer	deer	True
/wallhaven-01r6q1.jpg	deer	deer	True
/wallhaven-2kxypy.jpg	deer	deer	True
/wallhaven-45z227.jpg	dog	horse	False
/wallhaven-4377py.jpg	dog	horse	False
/wallhaven-gjo3o3.jpg	dog	cat	False
/wallhaven-eol9p8.jpg	dog	bird	False
/wallhaven-zx6q3o.jpg	dog	cat	False
/wallhaven-k91537.jpg	frog	bird	False
/wallhaven-4xpmqo.jpg	frog	frog	True
/wallhaven-4dk55l.jpg	ship	ship	True
/wallhaven-nzqxgj.jpg	ship	ship	True
/wallhaven-n6ld1x.jpg	truck	truck	True
/wallhaven-j58lep.jpg	truck	deer	False
/wallhaven-q6pr9q.jpg	truck	truck	True
/wallhaven-v9k8l5.jpg	truck	truck	True
/wallhaven-285me6.jpg	truck	truck	True
/wallhaven-lmolml.jpg	cat	cat	True
/wallhaven-4yx56l.jpg	cat	cat	True
/wallhaven-47pv5e.jpg	cat	dog	False
/wallhaven-wyml1p.jpg	cat	dog	False
/wallhaven-r22zyj.jpg	cat	dog	False

/wallhaven-9558qd.jpg	automobile	truck	False

Src — путь изображений, correct — правильный ответ (размеченный признак), model — ответ полученный от модели, correct answer — утверждение правильность ответа, False — модель неверно ответил, True — модель верно ответил (таблица 2).

В результате выполнения тестирования реальных данных, оценка точность (accuracy) получена 53,125%.

3 Выводы

В данное статье была использована модель для классификация данных на примере CIFAR-10 а так же была выполнено тесты на примере случайные подобранные и вручную размеченный изображений, в результате работы была получена оценка функция потерь на тренировочных данных 0,393, а тестовых данных была получена 0,903, а для оценки правильность ответа модели была получена 74,16% на тестовых данных, а для оценки на реальных данных подобранных случайный изображения и вручную размеченный была достигнута точность 53,125%. И в заключение модель способен определять всего 10 класс изображений.

Библиографический список

1. Recht B. et al. Do cifar-10 classifiers generalize to cifar-10? //arXiv preprint arXiv:1806.00451. 2018.
2. Abouelnaga Y. et al. Cifar-10: Knn-based ensemble of classifiers //2016 International Conference on Computational Science and Computational Intelligence (CSCI). – IEEE, 2016. С. 1192-1195.
3. Bankman D. et al. An Always-On 3.8$\mu $ J/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS //IEEE Journal of Solid-State Circuits. 2018. Т. 54. №. 1. С. 158-172.
4. Giuste F. O., Vizcarra J. C. Cifar-10 image classification using feature ensembles //arXiv preprint arXiv:2002.03846. 2020.
5. Hsu T. M. H., Qi H., Brown M. Measuring the effects of non-identical data distribution for federated visual classification //arXiv preprint arXiv:1909.06335. 2019.
6. Sharma N., Jain V., Mishra A. An analysis of convolutional neural networks for image classification //Procedia computer science. – 2018. – Т. 132. – С. 377-384.
7. CIFAR-10 and CIFAR-100 datasets // CIFAR-100 datasets URL: https://www.cs.toronto.edu/~kriz/cifar.html (дата обращения: 2023-05-09).
8. Awesome Wallpapers - wallhaven.cc // wallhaven.cc URL: https://wallhaven.cc (дата обращения: 2023-05-11).

4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import cv2
from torchsummary import summary
from torchvision import transforms, datasets
import torchvision.transforms.functional as F
from typing import Optional, Callable, TypeVar, Type, Union
from PIL import Image
from Lib.AppMain import *
import Lib.Transforms as Trans
from torch.cuda import amp
import torch.nn.functional as nnf
import matplotlib.pyplot as plt
import os
from collections import deque

class Cifar10CnnModel(nn.Module):
	def __init__(self):
		super().__init__()
		self.network = nn.Sequential(
			nn.Conv2d(3, 32, kernel_size=3, padding=1),
			nn.ReLU(),
			nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
			nn.ReLU(),
			nn.MaxPool2d(2, 2), # output: 64 x 16 x 16
			nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
			nn.ReLU(),
			nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
			nn.ReLU(),
			nn.MaxPool2d(2, 2), # output: 128 x 8 x 8
			nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
			nn.ReLU(),
			nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
			nn.ReLU(),
			nn.MaxPool2d(2, 2), # output: 256 x 4 x 4
			nn.Flatten(),
			nn.Linear(256*4*4, 1024),
			nn.ReLU(),
			nn.Linear(1024, 512),
			nn.ReLU(),
			nn.Linear(512, 10)
		)
	def forward(self, xb):
		return self.network(xb)
class App(AppMain):
	def label_class(self, label):
		maping = {
			0:"airplane",
			1:"automobile",
			2:"bird",
			3:"cat",
			4:"deer",
			5:"dog",
			6:"frog",
			7:"horse",
			8:"ship",
			9:"truck"
		}
		in_map = (label.item() if type(label) == torch.Tensor else label)
		return maping[in_map]
	def main(self):
		self.dir_prefix = "./Data"
		self.dirs = {
			"dataset": "cifar-10",
			"fig": "Fig",
			"viewt": "View",
			"view_test": "View_test"
		}
		self.profile_name = "default"
```

```
75                      self.datasets = deque()
76                      self.model = {}
77                      self.optimizer = {}
78                      self.auto_save_exit = False
79                      self.init_dirs()
80                      self.disp_metric = ["loss", "val_loss", "val_accuracy"]
81                      self.metric.loss  = None
82                      self.metric.val_accuracy = None
83                      self.metric.val_loss = None
84                      self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
85                      self.criterion = nn.CrossEntropyLoss().to(self.device)
86                      self.lr = 0.001
87                      self.model = {
88                              "cnn": Cifar10CnnModel().to(self.device)
89                      }
90                      self.optimizer = {
91                              "cnn": torch.optim.Adam(self.model["cnn"].parameters(), lr=self.lr)
92                      }
93                      self.transform = transforms.ToTensor()
94                      self.cache_data = True
95                      self.init_data()
96              def init_data(self):
97                      train_data = datasets.CIFAR10(root=app.get_path("dataset"), train=True,
98                                                                      download=False, transform=self.transform)
99                      test_data = datasets.CIFAR10(root=app.get_path("dataset"), train=False,
100                                                                     download=False, transform=self.transform)
101                     self.datasets.append(Datasets_batch(train_data, batch_size=128))
102                     self.datasets.append(Datasets_batch(test_data, batch_size=128))
103                     self.init_datasets(num_workers=0)
104                     self.cache_data = True
105             def start_train(self) -> None:
106                     self.model["cnn"].train()
107             def test_model(self):
108                     ls = deque()
109                     for images, labels in self.datasets[1].dataLoader:
110                             images = images.to(self.device)
111                             labels = labels.to(self.device)
112                             out = self.model["cnn"](images)
113                             loss = self.criterion(out, labels)
114                             acc = self.accuracy(out, labels)
115                             ls.append({'val_loss': loss.detach(), 'val_acc': acc})
116                     return ls
117             def accuracy(self, outputs, labels):
118                     _, preds = torch.max(outputs, dim=1)
119                     return torch.tensor(torch.sum(preds == labels).item() / len(preds))
120             def step_train(self, sel: any, index :int) -> None:
121                     super().step_train(sel, index)
122                     images, labels = sel
123                     images = images.to(self.device)
124                     labels = labels.to(self.device)
125                     outputs = self.model["cnn"](images)
126                     loss = self.criterion(outputs, labels)
127                     loss.backward()
128                     self.optimizer["cnn"].step()
129                     self.optimizer["cnn"].zero_grad()
130                     if self.metric.Step % (len(self.datasets[0].dataLoader) - 1) == 0:
131                             self.model["cnn"].eval()
132                             m = self.test_model()
133                             m_data = {key: [dic[key] for dic in m] for key in m[0]}
134                             self.metric.val_accuracy = torch.stack(m_data["val_acc"]).mean().item()
135                             self.metric.val_loss = torch.stack(m_data["val_loss"]).mean().item()
136                             self.model["cnn"].train()
137                     self.metric.loss = loss.item()
138             def predict(self, img):
139                     self.model["cnn"].to("cpu").eval()
140                     img = self.transform(transforms.Resize([32, 32])(img)).unsqueeze(0)
141                     _, preds = torch.max(self.model["cnn"](img), dim=1)
142                     return self.label_class(preds)
143             def load_image(self, path :str) -> Image.Image:
144                     with open(path, "rb") as f:
145                             img = Image.open(f)
146                             return img.convert("RGB")
147             def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
148                     pic = None
149                     with open(path, "wb") as f:
150                             if type(img) != Image.Image:
151                                     pic = Image.fromarray(img, "RGB")
152                             else:
```

```
                                        pic = img
                                pic.save(f)
app = App()
app.main()
i = 0
for images, labels in app.datasets[0].dataLoader:
        p = app.get_path("view")
        for j in range(len(images)):
                cl = p + "/" + app.label_class(int(labels[j])).split("/")[0]
                if not os.path.exists(cl):
                        os.mkdir(cl)
                img_d = F.to_pil_image(images[j])
                app.save_image(cl + "/" + "%s.jpg" % i, img_d)
                i += 1
i = 0
for images, labels in app.datasets[1].dataLoader:
        p = app.get_path("view_test")
        for j in range(len(images)):
                cl = p + "/" + app.label_class(int(labels[j])).split("/")[0]
                if not os.path.exists(cl):
                        os.mkdir(cl)
                img_d = F.to_pil_image(images[j])
                app.save_image(cl + "/" + "%s.jpg" % i, img_d)
                i += 1
summary(app.model["cnn"], input_size=(3, 32, 32), batch_size=128)
app.fit(10)
app.save_model("class-cifar-10")
#----------------------------
app.load_model("class-cifar-10")
app.model["cnn"].to("cpu").eval()
url_image = "./images"
cl_data = {
        "/wallhaven/wallhaven-48gk7k.jpg":"2",
        "/wallhaven/wallhaven-nkype1.jpg":"2",
        "/wallhaven/wallhaven-48jj2k.jpg":"2",
        "/wallhaven/wallhaven-odqg69.jpg":"2",
        "/wallhaven/wallhaven-392r63.jpg":"2",
        "/wallhaven/wallhaven-zmdj5v.jpg":"2",
        "/wallhaven/wallhaven-eooryk.jpg":"2",
        "/wallhaven/wallhaven-0wm8mx.jpg":"2",
        "/wallhaven/wallhaven-x18ywz.jpg":"4",
        "/wallhaven/wallhaven-lqw1qr.jpg":"4",
        "/wallhaven/wallhaven-01r6q1.jpg":"4",
        "/wallhaven/wallhaven-2kxypy.jpg":"4",
        "/wallhaven/wallhaven-45z227.jpg":"5",
        "/wallhaven/wallhaven-4377py.jpg":"5",
        "/wallhaven/wallhaven-gjo3o3.jpg":"5",
        "/wallhaven/wallhaven-eol9p8.jpg":"5",
        "/wallhaven/wallhaven-zx6q3o.jpg":"5",
        "/wallhaven/wallhaven-k91537.jpg":"6",
        "/wallhaven/wallhaven-4xpmqo.jpg":"6",
        "/wallhaven/wallhaven-4dk55l.jpg":"8",
        "/wallhaven/wallhaven-nzqxgj.jpg":"8",
        "/wallhaven/wallhaven-n6ld1x.jpg":"9",
        "/wallhaven/wallhaven-j58lep.jpg":"9",
        "/wallhaven/wallhaven-q6pr9q.jpg":"9",
        "/wallhaven/wallhaven-v9k8l5.jpg":"9",
        "/wallhaven/wallhaven-285me6.jpg":"9",
        "/wallhaven/wallhaven-lmolml.jpg":"3",
        "/wallhaven/wallhaven-4yx56l.jpg":"3",
        "/wallhaven/wallhaven-47pv5e.jpg":"3",
        "/wallhaven/wallhaven-wyml1p.jpg":"3",
        "/wallhaven/wallhaven-r22zyj.jpg":"3",
        "/wallhaven/wallhaven-9558qd.jpg":"1"
}
l_answer = deque()
log_ls = deque()
for i in cl_data:
        img = app.load_image(url_image + "/" + i)
        app.save_image(url_image + "/Out/%s" % (os.path.basename(i)), img)
        y = app.predict(img)
        label = app.label_class(int(cl_data[i]))
        print("src: %s, correct: %s, model: %s, correct answer: %s" % (
                i, label, y, label == y
        ))
        l_answer.append(label == y)
        log_ls.append({
                "src": i,
```

```
                        "correct": label,
                        "model": y,
                        "correct answer": label == y
                })
n_answer = np.array(l_answer)
print("accuracy: %s" % (n_answer[n_answer == True].sum() / len(n_answer)))
pd.DataFrame(log_ls).to_csv(url_image + "/out.csv")
# plot
df = pd.DataFrame(app.metric._data)
df_m = df[df["val_accuracy"].notna()].drop_duplicates("Epoch").iloc[1:]
df_time = df[df["Time"] < 1]
df_time.drop_duplicates("Step")["Time"].sum()
df.drop_duplicates("Step")["Time"].sum()
x_step = 390
#Time
fig, ax = plt.subplots()
ax.plot(df["Step"], df["Time"], label ='Time', color="red", linewidth=0.2)
ax.set_xticks(np.arange(0, max(df["Step"])+2, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("Time")), dpi = 300)
y_step = np.round((df_time["Time"].max() - df_time["Time"].min()) / 10.0, 3)
fig, ax = plt.subplots()
ax.plot(df_time["Step"], df_time["Time"], label ='Time', color="purple", linewidth=0.1)
ax.set_xticks(np.arange(0, max(df_time["Step"])+2, x_step))
ax.set_yticks(np.arange(
        np.round(min(df_time["Time"]), 3),
        np.round(max(df_time["Time"]), 3),
        y_step
        ))
fig.savefig(app.get_path("fig", "{0}.png".format("Time2")), dpi = 300)
#loss
y_step = (df["loss"].max() - df["loss"].min()) / 10.0
fig, ax = plt.subplots()
ax.plot(df["Step"], df["loss"], label ='loss', color="blue", linewidth=0.1)
ax.set_xticks(np.arange(0, max(df["Step"])+2, x_step))
ax.set_yticks(np.arange(0, max(df["loss"]) + y_step * 1.0, y_step))
fig.savefig(app.get_path("fig", "{0}.png".format("loss")), dpi = 300)
df.iloc[df["loss"].idxmin()]
# Test
x_step = 1
fig, ax = plt.subplots()
ax.plot(df_m["Epoch"]-1, df_m["val_accuracy"], label ='Accuracy', color="green")
ax.set_xticks(np.arange(0, max(df_m["Epoch"]), x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("val_accuracy")), dpi = 300)
fig, ax = plt.subplots()
ax.plot(df_m["Epoch"]-1, df_m["val_loss"], label ='val_loss', color="orange")
ax.set_xticks(np.arange(0, max(df_m["Epoch"]), x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("val_loss")), dpi = 300)
fig, ax = plt.subplots()
ax.plot(df_m["Epoch"]-1, df_m["loss"], label ='loss', color="cyan")
ax.set_xticks(np.arange(0, max(df_m["Epoch"]), x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("train_loss")), dpi = 300)
df_m.loc[df_m["val_loss"].idxmin()]
df_m.loc[df_m["loss"].idxmin()]
df_m.loc[df_m["val_accuracy"].idxmax()]
```