УДК 004.8

Распознавание рукописных цифр

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети для распознавание рукописные числа. В работе использовался набор данных MNIST содержащий рукописные числа и использовалась модель сверточная нейронная сеть для распознавание рукописные числа. В результате работы модель способен распознавать рукописные числа на примере MNIST введенный пользователем.
Ключевые слова: MNIST, распознавания, сверточная нейронная сеть, Torch.

Handwritten digit recognition

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using a neural network model for handwritten number recognition. The work used the MNIST dataset containing handwritten numbers and used a convolutional neural network model to recognize handwritten numbers. As a result of the work, the model is able to recognize handwritten numbers on the example of MNIST entered by the user.
Keywords: MNIST, recognition, convolutional neural network, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в том что MNIST широко используется для распознавания рукописных цифр в различных приложениях, начиная от почтовых служб, автоматического чтения документов, обработки банковских чеков и проверки подписи. Распознавание рукописных символов, особенно цифр, является сложной задачей, поскольку у разных людей существует множество различий в стилях письма. Набор данных MNIST состоит из 60 000 обучающих изображений и 10 000 тестовых изображений рукописных цифр, что достаточно для обучения и тестирования моделей глубокого обучения.

Недавние достижения в архитектуре нейронных сетей и алгоритмах обучения показали многообещающие результаты в достижении высокой точности набора данных MNIST. Самая современная точность набора данных

MNIST была достигнута с помощью сверточных нейронных сетей (CNN), которые обычно используются для задач распознавания изображений.

Поэтому набор данных MNIST и распознавание рукописных цифр остается актуальной темой исследований в области машинного обучения и компьютерного зрения. Достижения в области методов глубокого обучения и повышенная доступность крупномасштабных наборов данных будут и впредь способствовать прогрессу в этой области и поддерживать разработку новых приложений.

1.2. Цель исследования

Целью работы является разработка приложения по распознавания рукописные числа на примере MNIST и тренировка нейронный сеть для распознавания рукописные числа.

1.3. Обзор исследований

Авторы исследования Х. Ченг и др, предлагают новый тип импульсной нейронной сети под названием LISNN, которая включает в себя латеральные взаимодействия, и показывают, что она превосходит традиционные импульсные нейронные сети в некоторых задачах распознавания объектов. Исследование дает интересное представление о потенциале импульсных нейронных сетей для решения сложных задач и предлагает многообещающее направление для будущих исследований в этой области. Однако технический язык, использованный в исследовании, может вызвать затруднения у тех, кто не знаком с предметом исследования [1].

Л. М. Сенг и др. предоставляет глубокий анализ различных архитектур сверточных нейронных сетей (CNN) для распознавания рукописных цифр в популярном наборе данных MNIST. Авторы представляют подробное сравнение показателей производительности, включая точность, точность, полноту и оценку F1 для каждой архитектуры, предоставляя ценные рекомендации по выбору лучшей модели CNN для данного сценария. Исследование хорошо структурировано, с четким и лаконичным языком и подкреплено подробными экспериментальными результатами и визуализациями. В целом, исследование является ценным вкладом в области машинного обучения и компьютерного зрения и рекомендуется к прочтению всем, кто интересуется выбором архитектуры CNN для задач классификации изображений [2].

Л. Цзян, З. Чжан исследует эффективность среды глубокого обучения PyTorch в классификации изображений. Авторы представили новую модель классификации, названную сверточной нейронной сетью с двойным путем (D-PCNN), и провели обширный эксперимент, чтобы сравнить точность и производительность D-PCNN с другими существующими моделями классификации изображений. Эмпирические результаты показали, что предложенная D-PCNN превосходит существующие модели, такие как AlexNet, VGG и ResNet, с точки зрения точности и времени обучения. Кроме того, исследователи также провели сравнительный анализ различных оптимизаторов и графиков скорости обучения, чтобы оптимизировать предложенную модель D-PCNN. Исследование предполагает, что PyTorch

является эффективной и действенной платформой для построения моделей глубокого обучения для задач классификации изображений [3].

Х. Хуанг разрабатывает платформу с открытым исходным кодом для распознавания лиц с использованием PyTorch. Эта структура включает в себя модули предварительной обработки, обучения и тестирования как для обнаружения, так и для распознавания лиц. Исследование представляет собой тщательную оценку платформы с использованием различных наборов данных и сравнивает производительность различных алгоритмов распознавания лиц. Результаты показывают, что предложенная структура повышает точность и эффективность задач распознавания лиц, что делает ее весьма перспективным инструментом для будущих исследований в этой области [4].

С. Ан представляет эффективный и точный подход к распознаванию цифр с использованием комбинации простых сверточных нейронных сетей (CNN). Исследователи предложили ансамблевую структуру обучения, которая объединяет несколько моделей CNN для достижения более высокой производительности, чем отдельные модели. Результаты эксперимента показывают, что предложенный подход обеспечивает высокую точность набора данных MNIST по сравнению с другими современными моделями, сводя при этом к минимуму вычислительную сложность и обеспечивая простоту конструкции модели. В целом исследование представляет собой многообещающее решение задач распознавания цифр, которое можно легко воспроизвести в практических приложениях [5].

Ф. Лапорте, Й. Дамбре, П. Биенстман предлагает новый подход к моделированию и оптимизации фотонных схем с использованием среды глубокого обучения PyTorch. Авторы демонстрируют эффективность своего метода на различных примерах, показывая, что он позволяет значительно сократить время моделирования при сохранении точности. В целом исследование предлагает инновационный и многообещающий подход к моделированию и оптимизации фотонных схем, который заслуживает дальнейшего изучения [6].

2. Рабочий процесс

2.1. Набор данных

Исходный данный представлено MNIST [7], а внутри наборе данных содержится рукописные изображения размера 28x28 с разметкой числа от 0 до 9. Обработка изображений выполнялась только преобразования в тензор для дальнейшей тренировки нейронным сетям. А для использования модели, обработка выполнялась преобразования в градация серого и уменьшение размера до 28x28 пикселях, преобразования в тензор.

Рисунок 1. Набор рукописных изображений

2.2. Модель

Модель сверточной нейронной сети (CNN) для распознавания рукописного ввода из набора данных MNIST состоит из нескольких слоев фильтров и операций объединения. Входными данными для модели является изображение в градациях серого размером 28x28 пикселей, которое представляет собой рукописную цифру.

Первый слой представляет собой сверточный слой, который применяет набор фильтров к входному изображению. Каждый фильтр представляет собой небольшую матрицу (обычно 3x3 или 5x5), которая скользит по входному изображению и вычисляет скалярное произведение между фильтром и соответствующим участком пикселей во входном изображении. Результатом каждого вычисления является новое значение на карте объектов, которая сохраняет пространственную информацию входного изображения [8].

После каждого сверточного слоя к картам объектов применяется функция активации ReLU. ReLU устанавливает все отрицательные значения в ноль и оставляет все положительные значения такими, какие они есть. Эта нелинейная функция активации вносит нелинейность в модель и делает ее способной изучать более сложные закономерности и взаимосвязи [9].

Затем к картам объектов применяется объединяющий слой. Операция объединения уменьшает размерность данных и делает модель более эффективной. Max pooling — это распространенный тип операции объединения, который выбирает максимальное значение в пределах каждого участка пикселей и отбрасывает другие значения [10].

Выходные данные последнего слоя объединения объединяются в вектор и передаются в линейный слой. Где каждый нейрон связан с каждым нейроном в предыдущем и следующем слоях. Плотный слой использует информацию,

полученную от сверточных слоев и слоев объединения, для классификации входного изображения в одну из 10 возможных цифр (0-9).

Наконец, к выходным данным плотного слоя применяется функция активации Adam, чтобы получить распределение вероятностей по 10 возможным цифрам. Цифра с наибольшей вероятностью считается окончательным прогнозом модели.

Во время обучения модель учится улучшать свои прогнозы, минимизируя функцию потерь, которая измеряет разницу между прогнозируемыми вероятностями и метками истинности. Параметры модели оптимизируются с использованием алгоритма, называемого обратным распространением, который регулирует веса и смещения в сети, чтобы минимизировать функцию потерь [11].

В целом, модель CNN для распознавания рукописного ввода из набора данных MNIST использует слои свертки и объединения для извлечения значимых признаков из входного изображения, а также плотный слой для классификации изображения на основе этих признаков. Эта модель достигла очень высокой точности в наборе данных MNIST и служит основой для многих других приложений компьютерного зрения.

Листинг 2.1. Структура модели

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [16, 32, 28, 28]             320
     BatchNorm2d-2         [16, 32, 28, 28]              64
            ReLU-3         [16, 32, 28, 28]               0
          Conv2d-4         [16, 64, 28, 28]          18,496
     BatchNorm2d-5         [16, 64, 28, 28]             128
            ReLU-6         [16, 64, 28, 28]               0
       MaxPool2d-7         [16, 64, 14, 14]               0
          Conv2d-8         [16, 128, 14, 14]         73,856
     BatchNorm2d-9         [16, 128, 14, 14]            256
           ReLU-10         [16, 128, 14, 14]              0
      MaxPool2d-11          [16, 128, 7, 7]               0
          Dropout-12             [16, 6272]               0
          Linear-13              [16, 128]         802,944
     BatchNorm1d-14              [16, 128]             256
           ReLU-15              [16, 128]               0
          Dropout-16            [16, 128]               0
          Linear-17             [16, 64]            8,256
     BatchNorm1d-18             [16, 64]              128
           ReLU-19              [16, 64]               0
          Dropout-20            [16, 64]               0
          Linear-21             [16, 10]             650
================================================================
Total params: 905,354
Trainable params: 905,354
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 39.91
Params size (MB): 3.45
Estimated Total Size (MB): 43.41
----------------------------------------------------------------
```

```
Net(
          (conv_block): Sequential(
```

```
                (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                (5): ReLU(inplace=True)
                (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
                (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
                (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                (9): ReLU(inplace=True)
                (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        )
        (linear_block): Sequential(
                (0): Dropout(p=0.5, inplace=False)
                (1): Linear(in_features=6272, out_features=128, bias=True)
                (2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                (3): ReLU(inplace=True)
                (4): Dropout(p=0.5, inplace=False)
                (5): Linear(in_features=128, out_features=64, bias=True)
                (6): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
                (7): ReLU(inplace=True)
                (8): Dropout(p=0.5, inplace=False)
                (9): Linear(in_features=64, out_features=10, bias=True)
        )
)
```

Листинг 2.2. Параметры для обучения модели

```
1  self.criterion = nn.CrossEntropyLoss().to(self.device)
2  self.lr = 0.003
3  self.model = {
4          "cnn": Net().to(self.device)
5  }
6  optimizer = optim.Adam(self.model["cnn"].parameters(), lr=self.lr)
7  self.optimizer = {
8          "cnn": optimizer,
9          "exp_lr_scheduler": optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
10 }
```

Строка 1. Метрика CrossEntropyLoss для классификации признаков.
Строка 2. Скорость обучения.
Строка 3 — 5. Модель.
Строка 6. Оптимизатор Adam.
Строка 9. Снижает скорость обучения по гамме 0.1 и каждый шаг 7.

2.3. Обучение

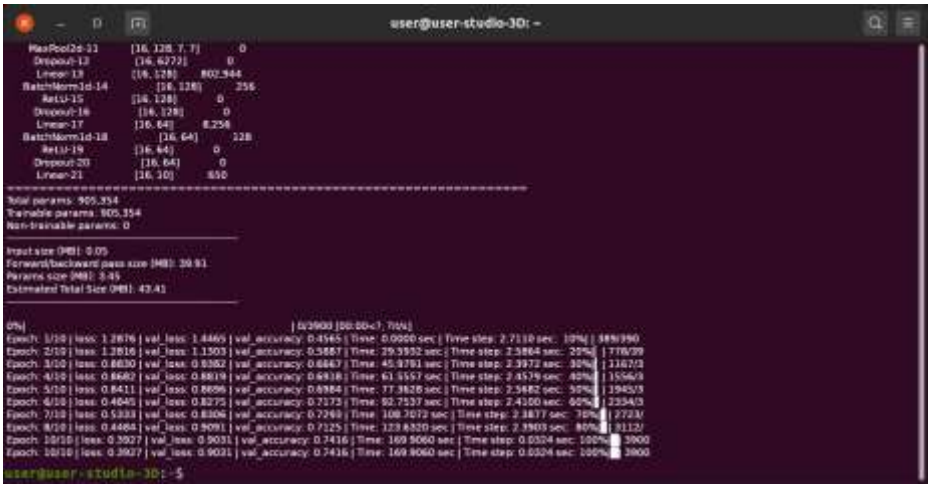

Рисунок 2. Обучение модели

В результате выполнения обучения мы получили оценки модели функция потерь и точность (только тестовых признаков), в результате обучение последняя эпоха получена функция потерь — 0,01597, а точность достигла (только тестовых признаков) — 99,479%, а функция потерь для тестовых признаков достигла — 0,01967. Обучения проходил 25 эпох, 11 700 циклов. Для обучения использовалась метрика CrossEntropyLoss.

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          self.optimizer["cnn"].zero_grad()
7          outputs = self.model["cnn"](images)
8          loss = self.criterion(outputs, labels)
9          loss.backward()
10         self.optimizer["cnn"].step()
11         if self.metric.Step % (len(self.datasets[0].dataLoader) - 1) == 0:
12                 self.model["cnn"].eval()
13                 m = self.test_model()
14                 m_data = {key: [dic[key] for dic in m] for key in m[0]}
15                 self.metric.val_accuracy = torch.stack(m_data["val_acc"]).mean().item()
16                 self.metric.val_loss = torch.stack(m_data["val_loss"]).mean().item()
17                 self.model["cnn"].train()
18         self.metric.loss = loss.item()
```

Строка 2, заглушка.

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

Строка 6 — 10. обучаем модель.

Строка 11 — 17. оцениваем модель по тестовому признаку, это происходит до конца эпохи 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

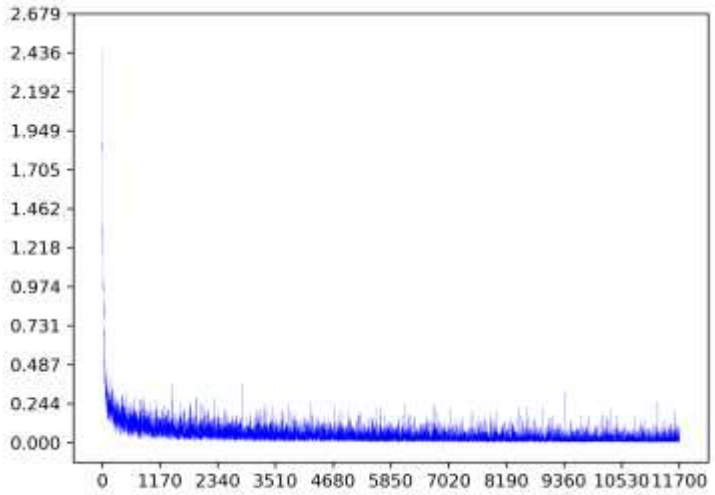

Рисунок 3. Функция потерь (метрика Cross Entropy)

На графике содержится значение функция потерь по осью Y, а по осью X — цикл обучения. Точка минимума достигла функция потерь 0,000295, цикл обучения — 9823, эпоха 21.

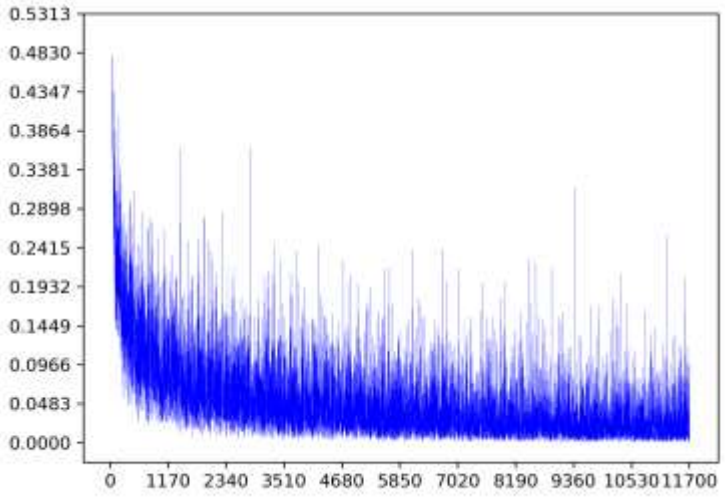

Рисунок 4. Функция потерь

На графике представлено с отсечение до 0.5 функция потерь.

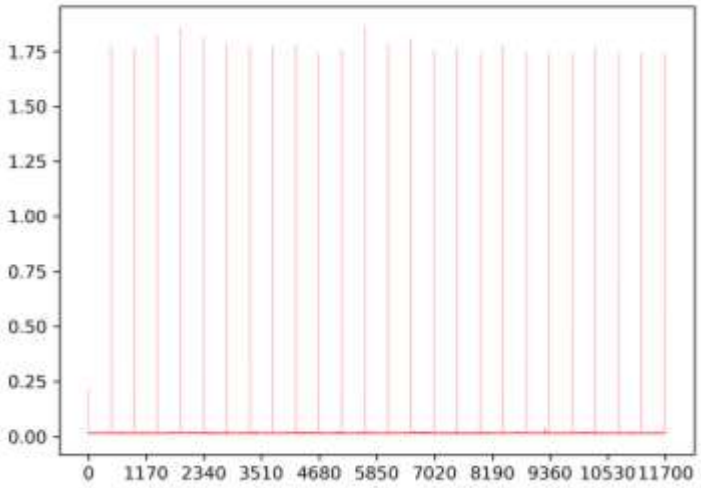

Рисунок 5. Время обучения

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.

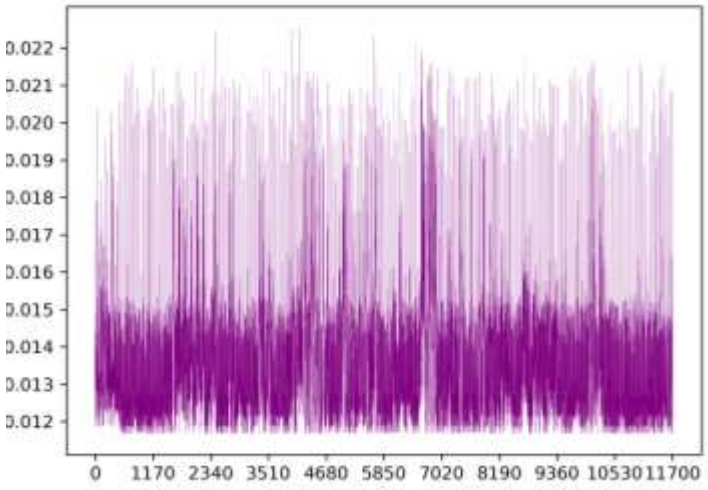

Рисунок 6. Время обучения без обработки признаков

На графике представлен время обучения без обработки признаков с отсечение порога не больше 0.04 секунда. Время обучения с отсечение 158,673 секунд (2 минута, 38,673 секунд). В среднем 0,013593 секунда.

Полная время обучения 160,049 (2 минута, 40,049 секунда). В среднем — 0,013593 секунда.

Время задержки обработкой признаков 44,615 секунда. В среднем — 1,785 секунда.

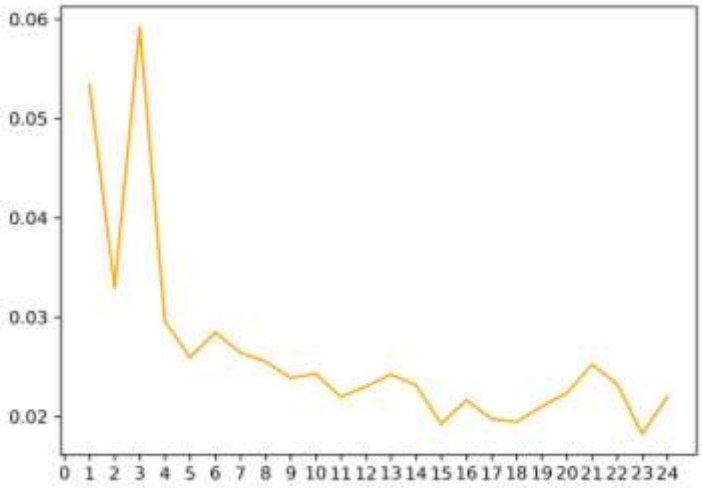

Рисунок 7. Функция потерь в тестовых признаках

По оси X — эпоха, а по оси Y — функция потерь, оценка выполнялось по тестовому признаку, метрика использовалась Cross Entropy, минимальное значение потерь 0,01824 в эпохе 23. Последняя эпоха достигнута — 0,022014.

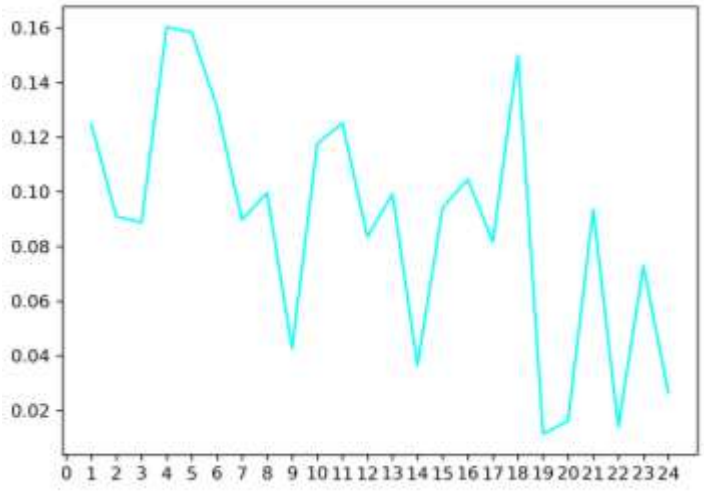

Рисунок 8. Функция потерь в обучающих признаках

Минимальное значение потерь 0,011340 в обучающий признаков 19 эпоха.

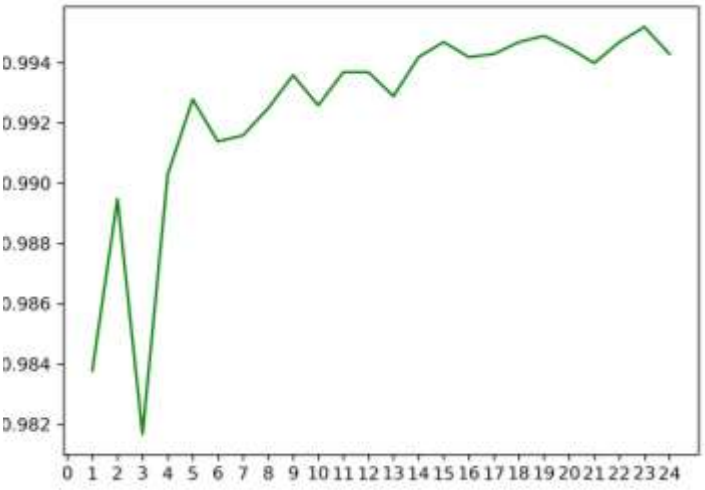

Рисунок 9. Точность (accuracy) в тестовых признаках

Оценка выполнялось по тестовому признаку, для оценки модели использовалась сравнения прогнозируемые и тестовая признаки, а затем получили среднее значение, максимальное значение удалось достичь — 99,519% в 23 эпоха.

2.5. Предсказание

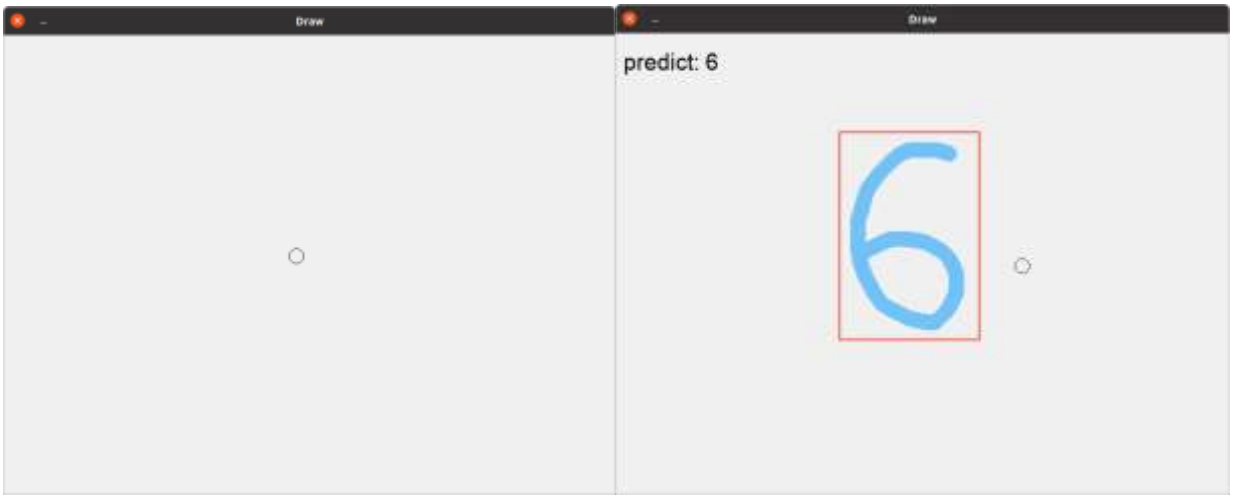

Рисунок 10. Программа рукописный распознавание изображений

Программа написано на языке Python использует библиотеку Pygame, для использования, левая кнопка мыши — рисовать, правая кнопка мыши — стирать, прокрутка вверх и вниз изменяет размер кисти, программа принимает клавиатурный ввод, d — клавиша для распознавание рисунков выделенный красный область, c — полностью стереть рисунок. Красный выделенный область — размер изображений для входной модели предсказания.

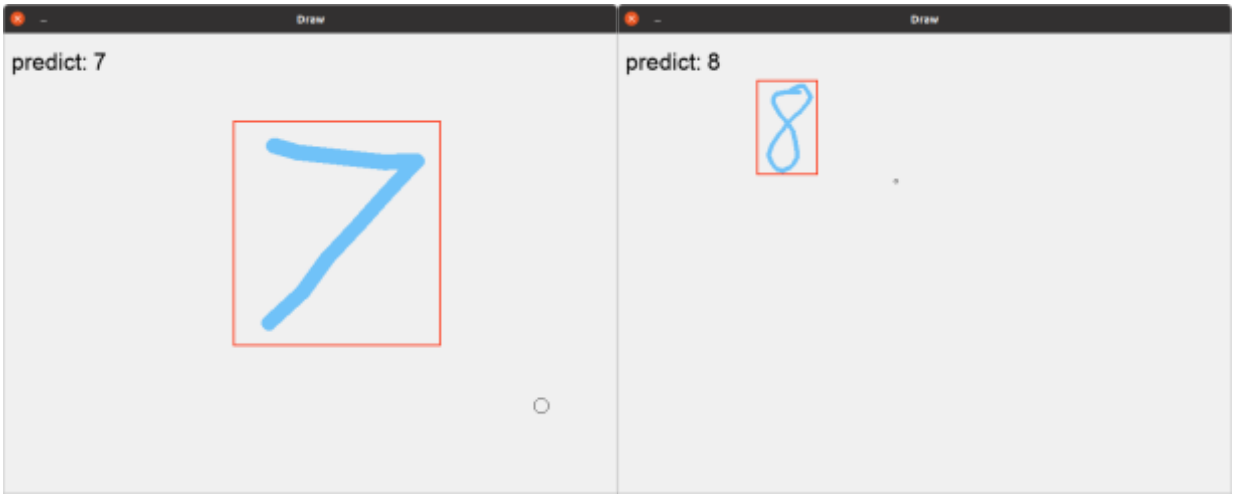

Рисунок 11. Пример нарисованный число 7 и 8

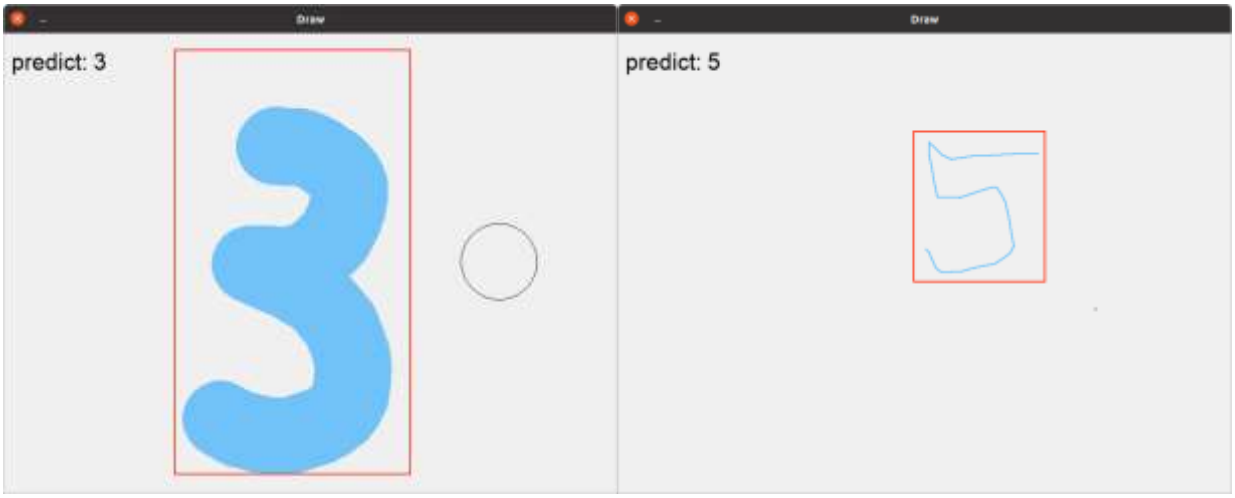

Рисунок 12. Пример нарисованный число 3 и 5

3 Выводы

В данной статьи была использована модель для распознавание рукописных цифр, в результате работы получили метрику функция потерь — 0,01597 по обучающий признаков, а по тестовому признаку 0,01967, а по точности предсказание 99,479%, в результате модель неплохо распознает рукописные изображений, а так же мы разработали программу для рукописный ввод и получили возможность произвольно использовать модель для распознавания рукописный ввод.

Библиографический список

1. Cheng X. et al. LISNN: Improving spiking neural networks with lateral interactions for robust object recognition //IJCAI. – 2020. – С. 1519-1525.
2. Seng L. M. et al. MNIST handwritten digit recognition with different CNN architectures //J. Appl. Technol. Innov. – 2021. – Т. 5. – №. 1. – С. 7-10.
3. Jiang L., Zhang Z. Research on image classification algorithm based on pytorch //Journal of Physics: Conference Series. – IOP Publishing, 2021. – Т. 2010. – №. 1. – С. 012009.
4. Huang X. et al. A research on face recognition open source development framework based on PyTorch //2021 International Symposium on Computer Technology and Information Science (ISCTIS). – IEEE, 2021. – С. 346-350.
5. An S. et al. An ensemble of simple convolutional neural network models for mnist digit recognition //arXiv preprint arXiv:2008.10400. – 2020.
6. Laporte F., Dambre J., Bienstman P. Photontorch: simulation and optimization of large photonic circuits using the deep learning framework PyTorch //2019 IEEE Photonics Society Summer Topical Meeting Series (SUM). – IEEE, 2019. – С. 1-2.
7. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges // mnist URL: http://yann.lecun.com/exdb/mnist/ (дата обращения: 2023-05-13).

8. Convolutional neural network - Wikipedia // Wikipedia URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (дата обращения: 2023-05-14).
9. ReLU — PyTorch 2.0 documentation // ReLU URL: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html (дата обращения: 2023-05-15).
10. Max Pooling Definition | DeepAI URL: https://deepai.org/machine-learning-glossary-and-terms/max-pooling (дата обращения: 2023-05-15).
11. Loss Function Definition | DeepAI URL: https://deepai.org/machine-learning-glossary-and-terms/loss-function (дата обращения: 2023-05-15).

4. Приложения

Листинг 4.1. Исходный код программы для тренировки нейронный сеть.

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

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
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

#https://www.kaggle.com/code/franklemuchahary/mnist-digit-recognition-using-pytorch
class Net(nn.Module):
	def __init__(self):
		super(Net, self).__init__()
		self.conv_block = nn.Sequential(
			nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
			nn.BatchNorm2d(32),
			nn.ReLU(inplace=True),
			nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
			nn.BatchNorm2d(64),
			nn.ReLU(inplace=True),
			nn.MaxPool2d(kernel_size=2, stride=2),
			nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
			nn.BatchNorm2d(128),
			nn.ReLU(inplace=True),
			nn.MaxPool2d(kernel_size=2, stride=2)
		)
		self.linear_block = nn.Sequential(
			nn.Dropout(p=0.5),
			nn.Linear(128*7*7, 128),
			nn.BatchNorm1d(128),
			nn.ReLU(inplace=True),
			nn.Dropout(0.5),
			nn.Linear(128, 64),
			nn.BatchNorm1d(64),
			nn.ReLU(inplace=True),
			nn.Dropout(0.5),
			nn.Linear(64, 10)
		)
	def forward(self, x):
		x = self.conv_block(x)
		x = x.view(x.size(0), -1)
		x = self.linear_block(x)
		return x

class App(AppMain):
	def main(self):
		self.dir_prefix = "./Data"
		self.dirs = {
			"dataset": "MNIST",
			"fig": "Fig",
			"view": "View",
			"view_test": "View_test",
			"view_raw": "View_raw"
		}
		self.profile_name = "default"
		self.datasets = deque()
		self.model = {}
		self.optimizer = {}
		self.auto_save_exit = False
		self.init_dirs()

		self.disp_metric = ["loss", "val_loss", "val_accuracy"]
		self.metric.loss  = None
```

```
75                      self.metric.val_accuracy = None
76                      self.metric.val_loss = None
77
78                      self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
79                      self.criterion = nn.CrossEntropyLoss().to(self.device)
80                      self.lr = 0.003
81                      self.model = {
82                              "cnn": Net().to(self.device)
83                      }
84                      optimizer = optim.Adam(self.model["cnn"].parameters(), lr=self.lr)
85                      self.optimizer = {
86                              "cnn": optimizer,
87                              "exp_lr_scheduler": optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
88                      }
89                      self.transform = transforms.ToTensor()
90                      self.transform_predict = transforms.Compose([
91                              transforms.Grayscale(num_output_channels=1),
92                              transforms.Resize([28, 28]),
93                              transforms.ToTensor()
94                      ])
95                      self.cache_data = True
96                      self.init_data()
97              def init_data(self):
98                      train_data = datasets.MNIST(root=self.get_path("dataset") + "/../", train=True,
99                                                                              download=True, transform=self.transform)
100                     test_data = datasets.MNIST(root=self.get_path("dataset") + "/../", train=False,
101                                                                             download=True, transform=self.transform)
102                     self.datasets.append(Datasets_batch(train_data, batch_size=128))
103                     self.datasets.append(Datasets_batch(test_data, batch_size=128))
104                     self.init_datasets(num_workers=0)
105                     self.cache_data = True
106             def start_train(self) -> None:
107                     self.model["cnn"].train()
108                     self.optimizer["exp_lr_scheduler"].step()
109             def test_model(self):
110                     ls = deque()
111                     for images, labels in self.datasets[1].dataLoader:
112                             images = images.to(self.device)
113                             labels = labels.to(self.device)
114                             out = self.model["cnn"](images)
115                             loss = self.criterion(out, labels)
116                             acc = self.accuracy(out, labels)
117                             ls.append({'val_loss': loss.detach(), 'val_acc': acc})
118                     return ls
119             def accuracy(self, outputs, labels):
120                     _, preds = torch.max(outputs, dim=1)
121                     return torch.tensor(torch.sum(preds == labels).item() / len(preds))
122             def step_train(self, sel: any, index :int) -> None:
123                     super().step_train(sel, index)
124                     images, labels = sel
125                     images = images.to(self.device)
126                     labels = labels.to(self.device)
127                     self.optimizer["cnn"].zero_grad()
128                     outputs = self.model["cnn"](images)
129                     loss = self.criterion(outputs, labels)
130                     loss.backward()
131                     self.optimizer["cnn"].step()
132                     if self.metric.Step % (len(self.datasets[0].dataLoader) - 1) == 0:
133                             self.model["cnn"].eval()
134                             m = self.test_model()
135                             m_data = {key: [dic[key] for dic in m] for key in m[0]}
136                             self.metric.val_accuracy = torch.stack(m_data["val_acc"]).mean().item()
137                             self.metric.val_loss = torch.stack(m_data["val_loss"]).mean().item()
138                             self.model["cnn"].train()
139                     self.metric.loss = loss.item()
140             def predict(self, img):
141                     self.model["cnn"].to("cpu").eval()
142                     img = self.transform_predict(img).unsqueeze(0)
143                     _, preds = torch.max(self.model["cnn"](img), dim=1)
144                     return int(preds)
145             def load_image(self, path :str) -> Image.Image:
146                     with open(path, "rb") as f:
147                             img = Image.open(f)
148                             return img.convert("RGB")
149             def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
150                     pic = None
151                     with open(path, "wb") as f:
152                             if type(img) != Image.Image:
```

```
153                                     pic = Image.fromarray(img, "RGB")
154                             else:
155                                     pic = img
156                             pic.save(f)
157 if __name__ == '__main__':
158         app = App()
159         app.main()
160         i = 0
161         for images, labels in app.datasets[0].dataLoader:
162                 p = app.get_path("view")
163                 for j in range(len(images)):
164                         cl = p + "/" + str(int(labels[j]))
165                         if not os.path.exists(cl):
166                                 os.mkdir(cl)
167                         img_d = F.to_pil_image(images[j])
168                         app.save_image(cl + "/" + "%s.jpg" % i, img_d)
169                         i += 1
170         i = 0
171         for images, labels in app.datasets[1].dataLoader:
172                 p = app.get_path("view_test")
173                 for j in range(len(images)):
174                         cl = p + "/" + str(int(labels[j]))
175                         if not os.path.exists(cl):
176                                 os.mkdir(cl)
177                         img_d = F.to_pil_image(images[j])
178                         app.save_image(cl + "/" + "%s.jpg" % i, img_d)
179                         i += 1
180         for images, labels in app.datasets[0].dataLoader:
181                 p = app.get_path("view_raw")
182                 for j in range(len(images)):
183                         img_d = F.to_pil_image(images[j])
184                         app.save_image(p + "/" + "%s.jpg" % i, img_d)
185                         i += 1
186         summary(app.model["cnn"], input_size=(1, 28, 28), batch_size=16)
187         app.fit(25)
188         app.save_model("class-mnist")
189
190         #-----------------------------------
191
192         app.load_model("class-mnist")
193         app.model["cnn"].to("cpu").eval()
194         y = app.predict(img)
195
196         #-----------------------------------
197
198         app.load_model("class-mnist")
199         # plot
200
201         df = pd.DataFrame(app.metric._data)
202         df_m = df[df["val_accuracy"].notna()].drop_duplicates("Epoch").iloc[1:]
203         df_time = df[df["Time"] < 0.04]
204         df_loss = df[df["loss"] < 0.5]
205
206         df_time.drop_duplicates("Step")["Time"].sum()
207         df.drop_duplicates("Step")["Time"].sum()
208
209         x_step = 1170
210
211         #Time
212         fig, ax = plt.subplots()
213         ax.plot(df["Step"], df["Time"], label ='Time', color="red", linewidth=0.2)
214         ax.set_xticks(np.arange(0, max(df["Step"])+2, x_step))
215         fig.savefig(app.get_path("fig", "{0}.png".format("Time")), dpi = 300)
216
217         y_step = np.round((df_time["Time"].max() - df_time["Time"].min()) / 10.0, 3)
218
219         fig, ax = plt.subplots()
220         ax.plot(df_time["Step"], df_time["Time"], label ='Time', color="purple", linewidth=0.1)
221         ax.set_xticks(np.arange(0, max(df_time["Step"])+2, x_step))
222         ax.set_yticks(np.arange(
223                 np.round(min(df_time["Time"]), 3),
224                 np.round(max(df_time["Time"]), 3),
225                 y_step
226                 ))
227         fig.savefig(app.get_path("fig", "{0}.png".format("Time2")), dpi = 300)
228
229         #loss
230         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)

            y_step = (df_loss["loss"].max() - df_loss["loss"].min()) / 10.0
            fig, ax = plt.subplots()
            ax.plot(df_loss["Step"], df_loss["loss"], label ='loss', color="blue", linewidth=0.1)
            ax.set_xticks(np.arange(0, max(df_loss["Step"])+2, x_step))
            ax.set_yticks(np.arange(0, max(df_loss["loss"]) + y_step * 1.0, y_step))
            fig.savefig(app.get_path("fig", "{0}.png".format("loss2")), 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()]
```

Листинг 4.2. Исходный код программы для распознавания рукопись.

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

import pygame
import pygame.locals as lgm
import pygame.draw as pyd
import numpy as np
from Main_app import *

app = App()
app.main()
app.load_model("class-mnist")
app.model["cnn"].to("cpu").eval()

pygame.init()
pygame.font.init()
disp_font = pygame.font.SysFont('arial', 30)
text_value = ""
vis_text_value = disp_font.render(text_value, True, (0, 0, 0))

screen_size = [800, 600]
screen = pygame.display.set_mode(screen_size)
clean_color = [0.9, 0.9, 0.9, 1.0]
fps = 24
background = pygame.Surface(screen_size)
background.fill(pygame.Color('#F0F0F0'))

cursor_0_rect = pygame.Rect(0, 0, 10, 10)
cursor_0 = pygame.Surface(screen_size, pygame.SRCALPHA)
cursor_0.fill(pygame.Color("#FFFFFF00"))

image_0 = pygame.Surface(screen_size)
image_0.fill(pygame.Color("#000000"))
image_0_view = pygame.Surface(screen_size, pygame.SRCALPHA)
image_0_view.fill(pygame.Color("#00000000"))

image_0_view2 = pygame.Surface(screen_size, pygame.SRCALPHA)
```

```
image_0_view2.fill(pygame.Color("#00000000"))

radius = np.array(cursor_0_rect).max()
pyd.circle(
        cursor_0,
        pygame.Color("#000000F0"),
        #np.array(cursor_0.get_size()) / 2.0,
        np.array([radius, radius]),
        radius, 1)

draw_status = 0
draw_start_point = [0, 0]
coord = np.array([0, 0])
prev_coord = np.array([0, 0])
crop_image = pygame.Rect(0, 0, 1, 1)
image_clear = True

#https://stackoverflow.com/questions/597369/how-to-create-ms-paint-clone-with-python-and-pygame
def roundline(srf, color, start, end, radius=1):
        dx = int(end[0])-int(start[0])
        dy = int(end[1])-int(start[1])
        distance = max(abs(dx), abs(dy))
        for i in range(int(distance)):
                x = int( start[0]+float(i)/distance*dx)
                y = int( start[1]+float(i)/distance*dy)
                pygame.draw.circle(srf, color, (x, y), radius)

def fn_update():
        screen.blit(image_0_view, [0, 0, *screen_size])
        screen.blit(image_0_view2, [0, 0, *screen_size])
        screen.blit(cursor_0, cursor_0_rect)
        vis_text_value = disp_font.render(text_value, True, (0, 0, 0))
        screen.blit(vis_text_value, (10, 20))

def fn_event(event):
        global draw_status,coord, draw_start_point, crop_image, image_clear
        global text_value,radius,cursor_0_rect,cursor_0
        if event.type == pygame.MOUSEMOTION:
                prev_coord = coord
                coord = np.array(pygame.mouse.get_pos()) - np.array(cursor_0_rect.size) / 2.0
                cursor_0_rect.update(coord - radius / 2.0, (radius, radius))
                if draw_status == 1 or draw_status == 2:
                        if coord[0] < crop_image.left:
                                crop_image.width += crop_image.left - coord[0] + radius * 2.0
                                crop_image.left = coord[0] - radius * 2.0
                        if coord[0] > (crop_image.left + crop_image.width):
                                crop_image.width = coord[0] - crop_image.left + radius * 2.0

                        if coord[1] < crop_image.top:
                                crop_image.height += crop_image.top - coord[1] + radius * 2.0
                                crop_image.top = coord[1] - radius * 2.0
                        if coord[1] > (crop_image.top + crop_image.height):
                                crop_image.height = coord[1] - crop_image.top + radius * 2.0

                        r_from = prev_coord + radius / 2.0
                        r_to = coord + radius / 2.0
                        if draw_status == 1:
                                roundline(image_0, pygame.Color("#FFFFFF"), r_from, r_to, radius)
                                roundline(image_0_view, pygame.Color("#0099FF88"), r_from, r_to, radius)
                        elif draw_status == 2:
                                roundline(image_0, pygame.Color("#000000"), r_from, r_to, radius)
                                roundline(image_0_view, pygame.Color("#00000000"), r_from, r_to, radius)

                        image_0_view2.fill(pygame.Color("#00000000"))
                        pyd.rect(image_0_view2, pygame.Color("#FF2200F0"), crop_image, 2)
        if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                        draw_status = 1
                elif event.button == 3:
                        draw_status = 2
                draw_start_point = coord
                if image_clear:
                        crop_image = crop_image.move(coord[0], coord[1])
                image_clear = False
        elif event.type == pygame.MOUSEBUTTONUP:
                draw_status = 0
        if event.type == pygame.MOUSEWHEEL:
                prev_coord = coord
```

```
                        coord = np.array(pygame.mouse.get_pos()) - np.array(cursor_0_rect.size) / 2.0
                        cursor_0_rect.update(coord - radius / 2.0, (radius, radius))

                        cursor_0_rect.width += event.y
                        cursor_0_rect.height += event.y
                        if cursor_0_rect.width < 1:
                                cursor_0_rect.width = 1
                                cursor_0_rect.height = 1
                        if cursor_0_rect.width > 50:
                                cursor_0_rect.width = 50
                                cursor_0_rect.height = 50

                        radius = np.array(cursor_0_rect.size).max()
                        cursor_0.fill(pygame.Color("#FFFFFF00"))
                        pyd.circle(
                                cursor_0,
                                pygame.Color("#000000F0"),
                                np.array([radius, radius]),
                        radius, 1)
                if event.type == pygame.KEYUP:
                        if event.key == pygame.K_c:
                                image_0.fill(pygame.Color("#000000"))
                                image_0_view.fill(pygame.Color("#00000000"))
                                image_0_view2.fill(pygame.Color("#00000000"))
                                image_clear = True
                                crop_image.x = 0
                                crop_image.y = 0
                                crop_image.width = 1
                                crop_image.height = 1
                        if event.key == pygame.K_d:
                                img_data = pygame.image.tostring(image_0, "RGB")
                                image_pil = Image.frombytes('RGB', image_0.get_size(), img_data)
                                image_pil = image_pil.crop((crop_image.left, crop_image.top, crop_image.right, crop_image.bottom))
                                print("predict: ", app.predict(image_pil))
                                text_value = "predict: " + str(app.predict(image_pil))

if __name__ == "__main__":
        clock = pygame.time.Clock()
        gdw = pygame.draw
        pygame.display.set_caption("Draw")
        screen.fill(np.array(clean_color) * 255)
        cycles = True
        while cycles:
                time_delta = clock.tick(fps) / 1000.0
                screen.blit(background, (0, 0))
                fn_update()
                for event in pygame.event.get():
                        pygame.display.update()
                        if event.type == lgm.QUIT or (event.type == lgm.KEYDOWN and event.key == lgm.K_ESCAPE):
                                cycles = False
                        fn_event(event)

                pygame.display.flip()
        pygame.quit()
```