Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Кафедра вычислительной математики и механики

Лабораторная работа № 5

по дисциплине: «Интеллектуальные ИСИТ»

Выполнила:

студентка группы ИСТ-19-1б

Дорогина И.В.

Проверил:

ассистент кафедры ВММБ

Нетбай Г. В.

Пермь, 2022

**Описание метода**

Сверточная интеллектуальная нейронная сеть (ИНС) входит в состав технологий глубокого обучения.

Чтобы передать изображение в ИНС, нужно преобразовать его в матрицу размером , где каждый элемент матрицы имеет значение 1 или 0, исходя из контуров распознаваемого символа. Далее каждый пиксель изображения надо передать на вход сети. Для того чтобы ИНС могла распознать образы, ее нужно соответствующем образом обучить.

**Постановка задания**

В рамках лабораторной работы необходимо реализовать приложение, решающее задачу распознавания рукописных цифр с использованием искусственных нейронных сетей.

**Ход работы**

Для выполнения лабораторной работы использован набор данных *MNIST*, который является одним из самых популярных. Он содержит 60 000 тренировочных изображений написанных от руки цифр от 0 до 9, а также 10 000 картинок для тестирования. В наборе есть 10 разных классов. Изображения с цифрами представлены в виде матриц , где каждая ячейка содержит определенный оттенок серого.

Для глубокого обучения использована библиотека *Keras*. Графический интерфейс реализован с применением *PyQt5*.

На первом этапе были импортированы все модули, необходимые для тренировки модели (рис. 1). Некоторые из них уже содержатся в *Keras*, в том числе *MNIST*. С помощью метода *mnist.load\_data()* все тренировочные данные, их метки и тестовые данные с метками были разделены на набор для обучения и тестовый набор.

|  |
| --- |
| import keras  from keras.datasets import mnist  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten  from keras.layers import Conv2D, MaxPooling2D  from keras import backend as K  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  print(x\_train.shape, y\_train.shape) |
| Рисунок 1 – Импорт необходимых данных |

На этапе предварительной обработки данных (рис. 2) было выполнено преобразование их к необходимой для модели сверточной нейронной сети размерности – (60000, 28, 28, 1).

|  |
| --- |
| num\_classes = 10  x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)  x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)  input\_shape = (28, 28, 1)  # преобразование векторных классов в бинарные матрицы  y\_train = keras.utils.to\_categorical(y\_train, num\_classes)  y\_test = keras.utils.to\_categorical(y\_test, num\_classes)  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')  x\_train /= 255  x\_test /= 255  print('Размерность x\_train:', x\_train.shape)  print(x\_train.shape[0], 'Размер train')  print(x\_test.shape[0], 'Размер test') |
| Рисунок 2 – Подготовка данных для модели ИНС |

Следующий этап – создание модели сверточной ИНС (рис. 3). Она преимущественно состоит из сверточных и слоев подвыборки. Модель лучше работает с данными, представленными в качестве сеточных структур. Именно поэтому такая сеть хорошо подходит для задач по классификации изображений. Слой исключения используется для отключения отдельных нейронов и во время тренировки. Он уменьшает вероятность переобучения. Затем происходит компиляция модели с помощью оптимизатора *Adadelta*.

|  |
| --- |
| batch\_size = 128  epochs = 10  model = Sequential()  model.add(Conv2D(32, kernel\_size=(3, 3),activation='relu',input\_shape=input\_shape))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))  model.add(Dropout(0.25))  model.add(Flatten())  model.add(Dense(256, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(num\_classes, activation='softmax'))  model.compile(loss=keras.losses.categorical\_crossentropy,  optimizer=keras.optimizers.Adadelta(),metrics=['accuracy']) |
| Рисунок 3 – Создание сверточной нейронной сети |

Важным этапом является тренировка модели (рис. 4). Функция *model.fit()* в *Keras* начинает тренировку модели. Она принимает тренировочные, валидационные данные, эпохи (*epoch*) и размер батча (*batch*).

Тренировка модели занимает некоторое время. После этого веса и определение модели сохраняются в файле *mnist.h5*.

|  |
| --- |
| hist = model.fit(x\_train, y\_train, batch\_size = batch\_size, epochs=epochs, verbose=1, validation\_data=(x\_test, y\_test))  print("Модель успешно обучена")  model.save('mnist.h5')  print("Модель сохранена как mnist.h5") |
| Рисунок 4 – Тренировка ИНС |

Для создания графического интерфейса привлечена библиотека *PyQt5*. Основные компоненты:

1) *QImage* – полотно, для рисования цифр с помощью тачпада;

2) *QLabel* – поле для вывода результата работы ИНС;

3) Кнопки *Recognize* и *Clear* для управления приложением. *Recognize* – запуск процесса обработки введенной цифры, *Clear* – очищение полотна.

Для обеспечения рисования на полотне реализованы обработчики событий *mousePressEvent()*, *mouseMoveEvent()*, *mouseReleaseEvent()* (рис. 5).

|  |
| --- |
| def mousePressEvent(self, event):  if event.button() == Qt.LeftButton:  self.drawing = True  self.lastPoint = event.pos()  def mouseMoveEvent(self, event):  if (event.buttons() & Qt.LeftButton) & self.drawing:  painter = QPainter(self.image)  painter.setPen(QPen(self.brushColor, self.brushSize, Qt.SolidLine, Qt.RoundCap, Qt.RoundJoin))  painter.drawLine(self.lastPoint, event.pos())  self.lastPoint = event.pos()  self.update()  def mouseReleaseEvent(self, event):  if event.button == Qt.LeftButton:  self.drawing = False |
| Рисунок 5 – Реализация опции рисования |

Чтобы передать нарисованную цифру нейронной сети в функции *predict\_digit()* реализована подготовка картинки – изменение её размера, сохранение в 8-битном формате оттенков серого. Результат предсказания возвращается и отображается в приложении.

На рисунке 6 представлено приложение, реализованное в рамках лабораторной работы.

Полный листинг графического приложения в Приложении 1.

|  |
| --- |
|  |
| Рисунок 6 – Интерфейс приложения по распознаванию цифр |

**Примеры работы программы**

Ниже представлены примеры работы программы по распознаванию рукописных цифр с помощью нейронной сети:

|  |  |  |
| --- | --- | --- |
| 1. |  |  |
| 2. |  |  |
| 3. |  |  |

**Заключение**

В рамках лабораторной работы была сформирована способность построения нейросетевой интеллектуальной системы для решения задачи распознавания цифр.

В результате лабораторной работы разработана реализация графического приложения, решающего задачу распознавания образов.

Приложение 1

**Листинг приложения**

from keras.models import load\_model

import numpy as np

import sys

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtCore import \*

model = load\_model('mnist.h5')

def predict\_digit(img):

imm = img.scaled(28, 28, Qt.IgnoreAspectRatio, Qt.SmoothTransformation)

incomingImage = imm.convertToFormat(QImage.Format\_Grayscale8)

width = incomingImage.width()

height = incomingImage.height()

ptr = incomingImage.bits()

ptr.setsize(incomingImage.byteCount())

img = np.array(ptr).reshape(height, width, 1) # Copies the data

# изменение размерности для поддержки модели ввода и нормализации

img = img.reshape(1,28,28,1)

img = img/255.0

# предсказание цифры

res = model.predict([img])[0]

return np.argmax(res), max(res)

class App(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.top = 400

self.left = 400

self.width = 300

self.height = 300

self.setWindowTitle("ISIT Lab 5")

self.setGeometry(self.top, self.left, self.width, self.height)

self.image = QImage(self.size(), QImage.Format\_RGB32)

self.image.fill(Qt.white)

self.label = QLabel("", self)

self.label.setGeometry(125, 100, 100, 100)

self.drawing = False

self.brushSize = 10

self.brushColor = Qt.black

self.lastPoint = QPoint()

mainMenu = self.menuBar()

fileMenu = mainMenu.addMenu("Options")

recognizeAction = QAction("Recognize", self)

recognizeAction.setShortcut("Ctrl+R")

fileMenu.addAction(recognizeAction)

recognizeAction.triggered.connect(self.recognize)

clearAction = QAction("Clear", self)

clearAction.setShortcut("Ctrl+X")

fileMenu.addAction(clearAction)

clearAction.triggered.connect(self.clear)

def mousePressEvent(self, event):

if event.button() == Qt.LeftButton:

self.drawing = True

self.lastPoint = event.pos()

def mouseMoveEvent(self, event):

if (event.buttons() & Qt.LeftButton) & self.drawing:

painter = QPainter(self.image)

painter.setPen(QPen(self.brushColor, self.brushSize, Qt.SolidLine, Qt.RoundCap, Qt.RoundJoin))

painter.drawLine(self.lastPoint, event.pos())

self.lastPoint = event.pos()

self.update()

def mouseReleaseEvent(self, event):

if event.button == Qt.LeftButton:

self.drawing = False

def paintEvent(self, event):

canvasPainter = QPainter(self)

canvasPainter.drawImage(self.rect(), self.image, self.image.rect())

def recognize(self):

digit, acc = predict\_digit(self.image)

self.clear()

self.label.setText(str(digit) + ', ' + str(int(acc \* 100)) + '%')

def clear(self):

self.image.fill(Qt.white)

self.update()

def main():

app = QApplication(sys.argv)

window = App()

window.show()

app.exec()

if \_\_name\_\_ == '\_\_main\_\_':

main()