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-19-2б

Клименко А.А.

Проверил

ассистент кафедры ВММБ

Нетбай Г.В.

Пермь, 2022

Нейронная сеть — это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг. Это тип процесса машинного обучения, называемый глубоким обучением, который использует взаимосвязанные узлы или нейроны в слоистой структуре, напоминающей человеческий мозг.

Цель работы: сформировать у студентов способность построения нейросетевой интеллектуальной системы для решения задач распознавания образов.

Задача:

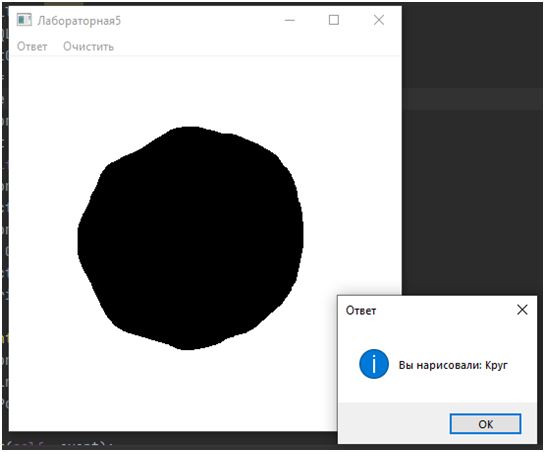
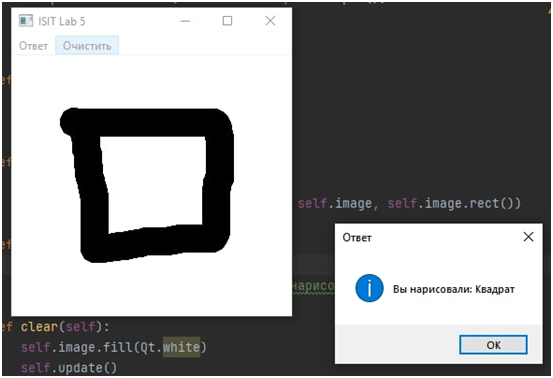
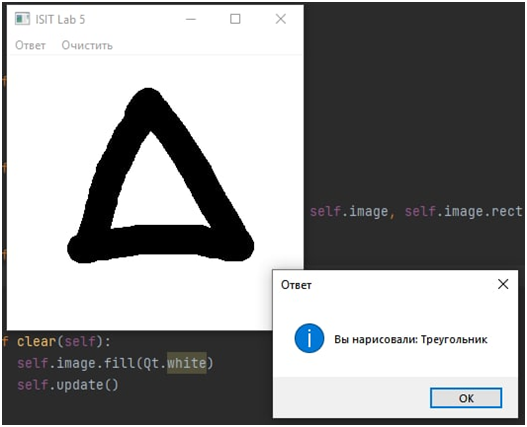
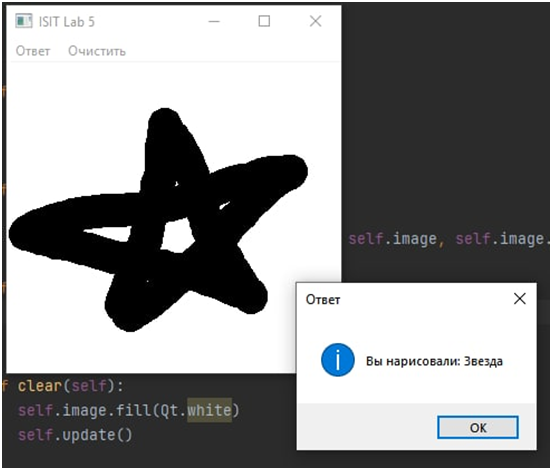
Реализовать приложение, решающее задачу распознавание образов с использованием искусственных нейронных сетей.

Архитектура сети

1. Слой свертки, размер ядра 3х3, количество карт признаков - 32 шт., функция активации ReLU.
2. Слой подвыборки, выбор максимального значения из квадрата 2х2
3. Слой свертки, размер ядра 3х3, количество карт признаков - 32 шт., функция активации ReLU.
4. Слой подвыборки, выбор максимального значения из квадрата 2х2
5. Слой свертки, размер ядра 3х3, количество карт признаков - 64 шт., функция активации ReLU.
6. Слой подвыборки, выбор максимального значения из квадрата 2х2
7. Слой преобразования из двумерного в одномерное представление
8. Полносвязный слой, 64 нейрона, функция активации ReLU.
9. Слой Dropout.
10. Выходной слой, 4 нейрона, функция активации sigmoid

Слои с 1 по 6 используются для выделения важных признаков в изображении, а слои с 7 по 10 - для классификации.

Ниже представлены листинг и работа программы.



Листинг программы:

Создание и обучение модели нейронной сети:

from keras.preprocessing.image import ImageDataGenerator  
from tensorflow.python.keras.models import Sequential  
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense  
  
train\_dir=r"C:\Users\Аня\Desktop\Work\ISIT\mymodel\train"  
test\_dir=r"C:\Users\Аня\Desktop\Work\ISIT\mymodel\test"  
val\_dir=r"C:\Users\Аня\Desktop\Work\ISIT\mymodel\val"  
  
img\_width, img\_height = 200, 200  
input\_shape = (img\_width, img\_height, 3)  
  
batch\_size = 20  
  
epochs = 20  
# Количество изображений для обучения  
nb\_train\_samples = 2706\*4  
# Количество изображений для проверки  
nb\_validation\_samples = 460\*4  
# Количество изображений для тестирования  
nb\_test\_samples = 561\*4  
  
datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.4)  
  
train\_generator = datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=(img\_width, img\_height),  
 batch\_size=batch\_size,  
 subset="training",  
 class\_mode='categorical')  
  
val\_generator = datagen.flow\_from\_directory(  
 val\_dir,  
 target\_size=(img\_width, img\_height),  
 batch\_size=batch\_size,  
 subset="validation",  
 class\_mode='categorical')  
  
test\_generator = datagen.flow\_from\_directory(  
 test\_dir,  
 target\_size=(img\_width, img\_height),  
 batch\_size=batch\_size,  
 class\_mode='categorical')  
  
  
model = Sequential()  
model.add(Conv2D(32, (3, 3), input\_shape=input\_shape, activation='relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Conv2D(32, (3, 3), activation='relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Conv2D(64, (3, 3), activation='relu'))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(Flatten())  
model.add(Dense(64, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(4, activation='softmax'))  
  
model.summary()  
  
model.compile(loss='categorical\_crossentropy',  
 optimizer='adam',  
 metrics=['accuracy'])  
  
  
x=train\_generator  
s\_p\_e=nb\_train\_samples // batch\_size  
v\_s=nb\_validation\_samples // batch\_size  
model.fit(x,  
 steps\_per\_epoch=s\_p\_e,  
 epochs=epochs,  
 validation\_data=val\_generator,  
 validation\_steps=v\_s)  
  
step=nb\_test\_samples // batch\_size  
  
scores = model.evaluate\_generator(test\_generator, step)  
  
print("Аккуратность на тестовых данных: %.2f%%" % (scores[1]\*100))  
model.save(r'C:\Users\Аня\Desktop\Work\ISIT\mymodel')

Тестирование модели нейронной сети:

import io  
from tkinter import Image  
  
import keras  
from PIL import ImageQt  
from keras.models import load\_model  
import numpy as np  
  
import sys  
from PyQt5.QtWidgets import \*  
from PyQt5.QtGui import \*  
from PyQt5.QtCore import \*  
import io  
from PIL import Image  
from tkinter.messagebox import showinfo  
  
model = keras.models.load\_model(r'C:\Users\Аня\Desktop\Work\ISIT\mymodel')  
name = np.array(['Круг', 'Квадрат', 'Звезда', 'Треугольник'])  
  
  
def predict\_digit(img):  
 buffer = QBuffer()  
 buffer.open(QBuffer.ReadWrite)  
 img.save(buffer, "PNG")  
 img = Image.open(io.BytesIO(buffer.data()))  
 img = img.convert('RGB')  
 img = img.resize((200, 200))  
 img = np.array(img)  
 x = np.asarray([np.array(img)])  
 predict\_result = model.predict(x, verbose=2)  
 result\_ind = np.argmax(predict\_result)  
 return name[result\_ind]  
  
class App(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Лабораторная5")  
 self.setGeometry(400, 400, 400, 400)  
 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 = 40  
 self.brushColor = Qt.black  
 self.lastPoint = QPoint()  
 mainMenu = self.menuBar()  
 recognizeAction = QAction("Ответ", self)  
 mainMenu.addAction(recognizeAction)  
 recognizeAction.triggered.connect(self.recognize)  
 clearAction = QAction("Очистить", self)  
 mainMenu.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):  
 o = predict\_digit(self.image)  
 showinfo(title="Ответ", message="Вы нарисовали: " + o)  
  
 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()