Эта программа представляет собой простое графическое приложение на Python с использованием библиотек TensorFlow и Tkinter для распознавания написанных пользователем цифр.

Сначала программа загружает обученную модель нейронной сети, которая была предварительно обучена на наборе данных MNIST (набор цифровых изображений размером 28x28 пикселей). Затем пользователь может нарисовать цифру в окошке с помощью мыши, а затем нажать кнопку «Распознать», чтобы приложение попыталось определить, какая цифра была нарисована.

Если пользователь хочет использовать уже существующее изображение вместо рисования, он может нажать кнопку «Выбрать файл», чтобы выбрать изображение из файловой системы.

Если пользователь хочет обучить модель заново, то он может нажать кнопку «Обучить». При этом загрузятся данные из набора данных MNIST, затем данные будут преобразованы и нормализованы, и наконец, модель будет обучена на этих данных. Обученная модель будет сохранена в файл mnist.h5.

**Код проекта.**

import tensorflow as tf  
from tensorflow.keras.datasets import mnist  
from keras.datasets import mnist  
from tkinter import \*  
from tkinter import filedialog  
import tkinter as tk  
import win32gui  
from PIL import ImageGrab, Image  
import numpy as np  
  
  
# Загружаем сохраненную модель  
model = tf.keras.models.load\_model('mnist.h5')  
  
  
class App(tk.Tk):  
 # Графический интерфейс  
 def \_\_init\_\_(self):  
 tk.Tk.\_\_init\_\_(self)  
 self.x = self.y = 0  
 self.canvas = tk.Canvas(self, width=100, height=100, bg="white", cursor="cross")  
 self.label = tk.Label(self, text="0", font=("Helvetica", 24))  
 self.classify\_btn = tk.Button(self, text="Распознать", command=self.classify\_handwriting)  
 self.button\_clear = tk.Button(self, text="Очистить", command=self.clear\_all)  
 self.button\_load = tk.Button(self, text="Выбрать файл", command=self.open\_file\_dialog)  
 self.button\_edu = tk.Button(self, text="Обучить", command=self.train\_model)  
 self.canvas.grid(row=0, column=0, pady=2, sticky=W, )  
 self.label.grid(row=0, column=1, pady=2, padx=2)  
 self.classify\_btn.grid(row=1, column=1, pady=2, padx=2)  
 self.button\_clear.grid(row=1, column=0, pady=2)  
 self.button\_load.grid(row=2, column=1, pady=2)  
 self.button\_edu.grid(row=2, column=0, pady=2)  
 self.canvas.bind("<B1-Motion>", self.draw\_lines)  
 self.title("Беседин О.А.")  
 self.iconbitmap("ico.ico")  
 self.geometry("230x170")  
  
 def train\_model(self):  
 # Загружаем набор данных MNIST  
 (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
 # Преобразуем данные и нормализуем их  
 x\_train = x\_train.reshape((60000, 28, 28, 1))  
 x\_test = x\_test.reshape((10000, 28, 28, 1))  
 x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  
 # Определяем модель нейронной сети  
 model = tf.keras.models.Sequential([  
 # входной слой  
 tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),  
 tf.keras.layers.MaxPooling2D((2, 2)),  
 tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),  
 tf.keras.layers.MaxPooling2D((2, 2)),  
 tf.keras.layers.Flatten(),  
 # скрытый слой  
 tf.keras.layers.Dense(64, activation='relu'),  
 # выходной слой  
 tf.keras.layers.Dense(10, activation='softmax')  
 ])  
 # Компилируем модель  
 model.compile(optimizer='adam',  
 loss='sparse\_categorical\_crossentropy',  
 metrics=['accuracy'])  
 # Обучаем модель  
 model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))  
 # Сохраняем модель  
 model.save('mnist.h5')  
  
 def open\_file\_dialog(self):  
 filename = filedialog.askopenfilename()  
 im = Image.open(filename)  
 # преобразование изображения в черно-белое и массив numpy, и нормализация значений пикселей  
 image\_array = np.array(im.resize((28, 28)).convert('L')) / 255.0  
 # изменение размерности массива до формы (1, 28, 28, 1)  
 image\_array = image\_array.reshape((1, 28, 28, 1))  
 # Делаем предсказание  
 predictions = model.predict(image\_array)  
 digit = np.argmax(predictions)  
 self.label.configure(text=str(digit))  
  
 def clear\_all(self):  
 self.canvas.delete("all")  
  
 def classify\_handwriting(self, ):  
 HWND = self.canvas.winfo\_id()  
 rect = win32gui.GetWindowRect(HWND)  
 im = ImageGrab.grab(rect)  
 # преобразование изображения в черно-белое и массив numpy, и нормализация значений пикселей  
 image\_array = np.array(im.resize((28, 28)).convert('L')) / 255.0  
 # изменение размерности массива до формы (1, 28, 28, 1)  
 image\_array = image\_array.reshape((1, 28, 28, 1))  
 # Делаем предсказание  
 predictions = model.predict(image\_array)  
 digit = np.argmax(predictions)  
 self.label.configure(text=str(digit))  
  
 def draw\_lines(self, event):  
 self.x = event.x  
 self.y = event.y  
 r = 3  
 self.canvas.create\_oval(self.x - r, self.y - r, self.x + r, self.y + r, fill='black')  
  
  
app = App()  
mainloop()