## 题目一、基于机器学习的图片识别模型

### 引言

近年来，随着深度学习技术的迅速发展，计算机视觉领域取得了显著的进展，特别是在图像识别方面。图像识别作为人工智能领域的重要应用之一，对于识别和分类图像中的对象具有重要意义，其应用涵盖了许多领域，如医学影像诊断、智能安防、自动驾驶等。

本报告旨在基于深度学习技术，构建一个机器学习模型，实现对猫的图片进行识别，并开发相应的 UI 界面用于展示实验训练过程和测试集准确率，探索并验证深度学习在计算机视觉领域的应用效果。针对吴恩达课后编程作业的数据集，报告利用神经网络的方法，实现对猫的图片进行高效、准确的识别，以及提供一个用户友好的界面，使用户能够导入任意图片进行识别。

本报告将围绕深度学习模型的构建过程、实验步骤、UI 界面开发、实验结果和分析等内容展开，旨在全面呈现研究的整体过程和结果。

### **模型构建的步骤和方法**

本报告的pycharm工程项目一共包含三个python文件：lr\_utils.py，Deeplearningcat.py，UI.py。其中，lr\_utils.py是由题目给出的，用于解析训练集和测试集文件的内容；Deeplearningcat.py是用于构建单层神经网络模型的文件；UI.py将实验结果可视化，并为用户提供交互接口。

首先，【吴恩达课后编程作业】第二周的作业数据集有209张图片作为训练集，50张图片作为测试集，图片中有的是猫的图片，有的不是。每张图片的像素大小为64\*64，吴恩达把这两个图片集转换成训练集和测试集文件：train\_catvnoncat.h5（训练集），test\_catvnoncat.h5（测试集）。在作业给出的已知条件中有训练集文件、测试集文件和lr\_utils.py文件。

lr\_utils.py程序用来解析训练集和测试集文件的内容。训练集和测试集文件经过lr\_utils.py文件处理后返回以下参数：

* train\_set\_x\_orig ：保存的是训练集里面的图像数据（本训练集有209张64x64的图像）。
* train\_set\_y\_orig ：保存的是训练集的图像对应的分类值（【0 | 1】，0表示不是猫，1表示是猫）。
* test\_set\_x\_orig ：保存的是测试集里面的图像数据（本训练集有50张64x64的图像）。
* test\_set\_y\_orig ： 保存的是测试集的图像对应的分类值（【0 | 1】，0表示不是猫，1表示是猫）。
* classes ： 保存的是以bytes类型保存的两个字符串数据，数据为：[b’non-cat’ b’cat’]。

训练集和测试集的数据情况如下：

训练集的数量: m\_train = 209

测试集的数量 : m\_test = 50

每张图片的宽/高 : num\_px = 64

每张图片的大小 : (64, 64, 3)

训练集\_图片的维数 : (209, 64, 64, 3)

训练集\_标签的维数 : (1, 209)

测试集\_图片的维数: (50, 64, 64, 3)

测试集\_标签的维数: (1, 50)

在将数据加载到主程序里面之后，需要将训练集和测试集的数据维度分别降低并转置，然后进行归一化处理。这里所说的维度降低是把维度为（64，64，3）的numpy数组重新构造为（64 \* 64 \* 3 = 12288，1）的数组，要乘以3的原因是每张图片是由64x64像素构成的，而每个像素点由（R，G，B）三原色构成的。

在经过处理后，训练集和测试集的数据为：

训练集降维最后的维度： (12288, 209)

训练集\_标签的维数 : (1, 209)

测试集降维之后的维度: (12288, 50)

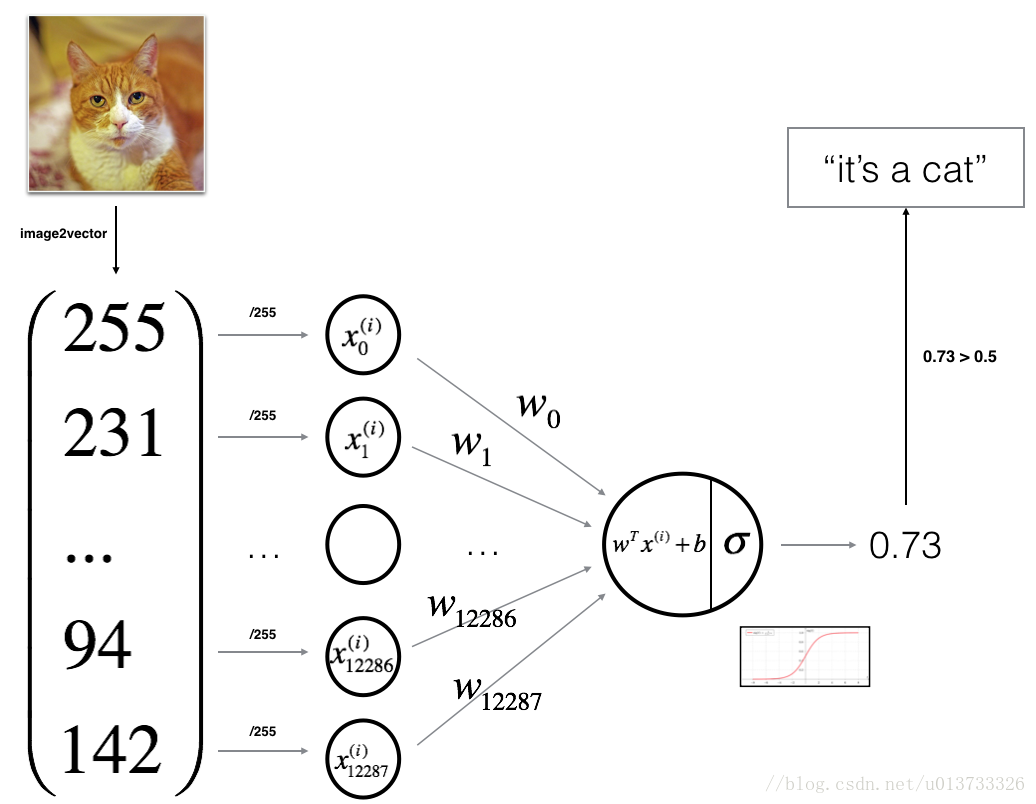
测试集\_标签的维数 : (1, 50)

在报告构建的多层线性神经网络模型中，输入信号:x，x，…，Xn权重值:w1，w，…，Wn。偏置值:b。输出信号由（1）式定义，损失函数由（3）定义，最后一层激活函数使用sigmoid函数，隐藏层使用Tanh函数。

然后通过对所有训练样例求和来计算成本:

（2）式中Sigmoid函数由sigmoid(z)程序定义如下：

隐藏层使用的Tanh激活函数如（6）式所示：



识别是非猫，报告采取构造多层线性神经网络模型。建立神经网络的主要步骤是：

0.数据读取与处理

1.初始化参数

2.前向线性传播

3.计算激活值

4.计算误差

5.反向传播

6.更新参数

7.预测

**Init\_params函数：**用于初始化神经网络的权重矩阵和偏置。

**Activation\_forward函数：**该函数用于封装一次前向传播过程——包含一个线性传播和激活函数的非线性激活，同时再最后一层使用Sigmoid激活函数。

**Forward\_propagation 函数：**用于实现神经网络的前向传播过程。该函数接收输入数据 X 和网络参数 parameters，计算每一层的激活值，最后返回输出层的激活值 Yhat 和缓存 caches。

**Compute\_cost 函数：**用于计算交叉熵损失，并返回损失值和损失对于输出层激活值 Yhat 的梯度。

**Linear\_backward函数：**用于计算线性层的反向传播。

**Activation\_backward函数：**该函数用于计算激活层的反向传播，其中包含了两种激活函数：TanH 和 Sigmoid。

**Backward\_propagation函数：**

参数：

dYhat：预测值的梯度

Yhat：预测值

Y：实际标签

caches：包含前向传播过程中存储的变量的元组

该函数的目标是计算损失函数对每个参数的梯度，以便进行参数更新。

返回grads 字典用于存储梯度矩阵：

L 表示神经网络的层数。

m 表示样本的数量。

针对输出层，利用 Activation\_backward 函数计算输出层的梯度，并将结果存储在 grads 中。通过一个 for 循环，从倒数第二层开始向前遍历各层，利用 Activation\_backward 函数分别计算各隐藏层的梯度，并将结果存储在 grads 中。最后，返回存储了各层梯度的 grads 字典。

**Update\_params函数：**使用梯度下降法更新神经网络参数。

参数：

parameters：包含神经网络参数的字典

grads：包含梯度信息的字典

learning\_rate：学习率，控制参数更新的步长

接下来调用Update\_params函数进行迭代次数为iterations次的参数优化，通过梯度下降算法来优化参数。对于第 l 层的权重矩阵，使用当前的学习率乘以该层的梯度来更新参数。对于第 l 层的偏置向量，同样使用当前的学习率乘以该层的梯度来更新参数。最后，返回更新后的参数 parameters。

**Train\_model函数：**训练模型的函数，通过反向传播算法训练神经网络模型，更新参数，并计算损失值。

参数：

X：输入特征

Y：实际标签

parameters：神经网络参数的字典

learning\_rate：学习率

iterations：迭代次数

threshold：误差阈值

主要步骤：

创建一个空列表 costs 用于存储每100次迭代的损失值，以便后续绘制折线图。

通过 for 循环进行指定次数的迭代，每次迭代包括以下步骤：

进行正向传播，得到预测结果 Yhat 和缓存值 caches。

计算损失值 cost 和关于预测值的导数 dYhat。

通过反向传播算法计算梯度 grads。

使用梯度下降法更新参数 parameters。

如果迭代次数能够整除100，将当前损失值添加到 costs 中并打印出来。

如果损失值低于阈值 threshold，同样将损失值添加到 costs 中并打印，并提前结束迭代。

最后返回更新后的参数 parameters、损失值列表 costs 和最终迭代次数 i。

**BP函数：**

参数：

X: 训练集特征

Y: 训练集标签

test\_data: 测试集特征

test\_label: 测试集标签

path: 参数存储路径

layers: 神经网络层的维度

iterations: 迭代次数

learning\_rate: 学习率

threshold: 误差阈值，默认为0.06

find\_wrong: 是否查找分类错误的样本

save\_params: 是否保存参数

load\_params: 是否加载参数

continue\_train: 是否继续训练

plot: 是否绘制损失曲线图

返回：

Parameters - 训练好的参数。

具体步骤如下：

调用 Init\_params 函数对参数进行初始化，并根据 load\_params 和 continue\_train 的值来决定是否加载之前保存的参数或者继续训练。

如果不是继续训练或者不加载参数，则调用 Train\_model 函数对模型进行训练，并根据 save\_params 的值来决定是否保存训练好的参数。

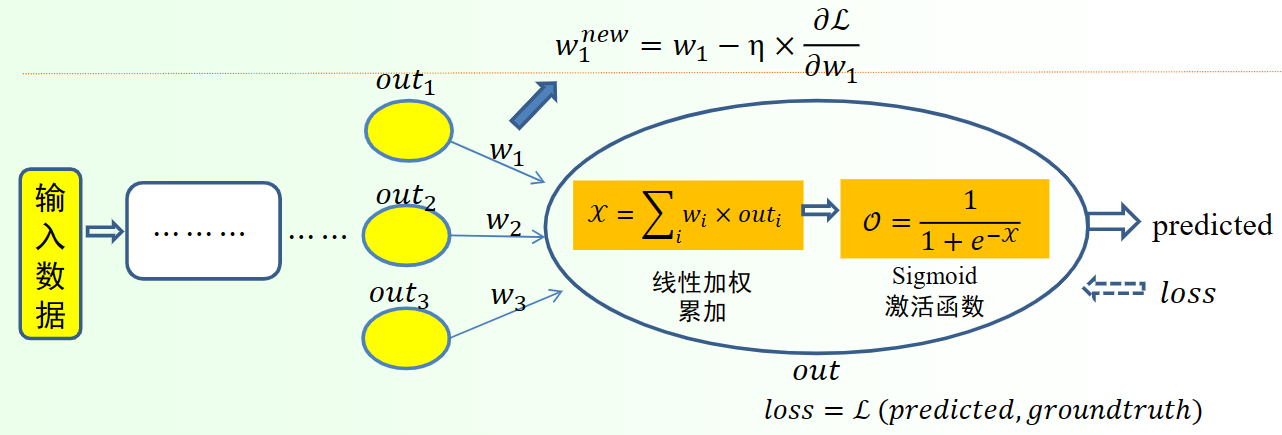
使用训练好的参数进行预测，并计算训练集和测试集的准确率。

如果 find\_wrong 为真，则调用 Find\_wrong 函数来查找分类错误的样本。

如果 plot 为真，则调用 Plot 函数绘制损失曲线图。

最后返回训练好的参数。

**参数优化，误差反向传播 (error back propagation, BP)：**



* BP算法: 将输出层误差反向传播给隐藏层进行参数更新的方法。
* 将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数。为了使损失函数𝓛取值减少（从而保证模型预测结果与实际结果之间的差距越来越小），需要求取损失函数𝓛相对于𝒘𝟏的偏导，然后按照损失函数梯度的反方向选取一个微小的增量，来调整𝒘𝟏的取值。
* 链式求导实现了损失函数对某个自变量求偏导，好比将损失误差从输出端向输入端逐层传播，通过这个传播过程来更新该自变量取值。梯度下降法告诉我们，只要沿着损失函数梯度的反方向来更新参数，就可使得损失函数下降最快。
* 参数𝑤1在下一轮迭代中的取值被调整为：

**switch选择函数:**

switch函数是Deeplearningcat.py留给UI.py文件接口，通过传递参数s来选择工作模式，s=1和s=0对应UI界面上的训练与测试按钮和识别图像按钮。

参数：

s - 1或者0，用来选择相应的程序（训练与测试和识别图像程序）

addr - 由用户在UI界面选择的本地图片地址，由识别图像程序所需要

d - d是识别图像程序所需要的训练好的参数集字典

当s=1，将执行以下步骤：

1. 加载训练集和测试集的数据。
2. 将训练集和测试集的维度降低并转置
3. 将训练集和测试集的数据归一化
4. 执行BP程序，获得返回值parameters
5. 返回parameters

当s=0，将执行以下步骤：

1. 获得图片地址参数addr
2. 根据地址打开图片
3. 将图像调整为 64x64 大小
4. 将图像数据转换为 NumPy 数组
5. 创建 HDF5 文件并保存图像数据，创建数据集output\_x
6. 将图像的维度降低并转置
7. 数据归一化
8. 利用训练好的参数进行预测
9. 根据predict返回值判断是不是猫并返回判断值

### **UI界面开发**

在UI.py程序中，我们从Deeplearningcat.py程序导入switch函数。Ui\_MainWindow类的作用是设置主窗口的布局，包括文本编辑器、标签和按钮等控件，并将按钮的点击事件与相应的方法进行关联。pushButton对应识别图像按钮，pushButton\_2对应训练与测试按钮，pushButton\_4对应选择图片按钮，pushButton\_5，pushButton\_6, pushButton\_7分别对应更新参数batch\_size，iterations，learning\_rate按钮。

关键的函数如下：

* def ClickButton1(self):  
   self.lineEdit.clear()  
   res = switch(0, imgNamepath,d)  
   if res == 1:  
   self.lineEdit.setText("是猫")  
   else:  
   self.lineEdit.setText("不是猫")

ClickButton1函数与pushButton的点击事件相关联。在点击识别图像按钮后触发ClickButton1函数。该函数调用switch函数的识别图像程序，并传入本地图片地址addr和训练好的参数集字典。最后对返回值进行判断，输出预测结果到文本框上。

* def ClickButton2(self):  
   global d  
   d=switch(1, imgNamepath,0)  
   img\_path = 'D:\硕士\研一\Python\\1 神经网络与深度学习\图片\准确率\plot.png' *# 图像路径* pixmap = QPixmap(img\_path)  
   self.label\_30.setPixmap(pixmap)  
   self.label\_30.setScaledContents(True)  
   self.pushButton.setEnabled(True)

ClickButton2函数与pushButton\_2的点击事件相关联。在点击训练与测试按钮后触发ClickButton2函数。函数调用switch的训练与测试程序进行模型构建，获得返回的d字典，将d字典设为全局变量。并将训练与测试程序所保存的模型效果图片调出粘贴在界面中部。最后使能pushButton按钮，表示训练好模型后，就可以对图像进行识别。

def ClickButton5(self):  
 global batch\_size  
 input\_value1 = self.lineEdit\_5.text()  
 batch\_size = int(input\_value1)  
 print(f'更新batch\_size为: {batch\_size}')  
def ClickButton6(self):  
 global iterations  
 input\_value2 = self.lineEdit\_6.text()  
 iterations = int(input\_value2)  
 print(f'更新迭代次数为: {iterations}')  
def ClickButton7(self):  
 global learning\_rate  
 input\_value3 = self.lineEdit\_7.text()  
 learning\_rate = float(input\_value3)  
 print(f'更新学习率为: {learning\_rate}')

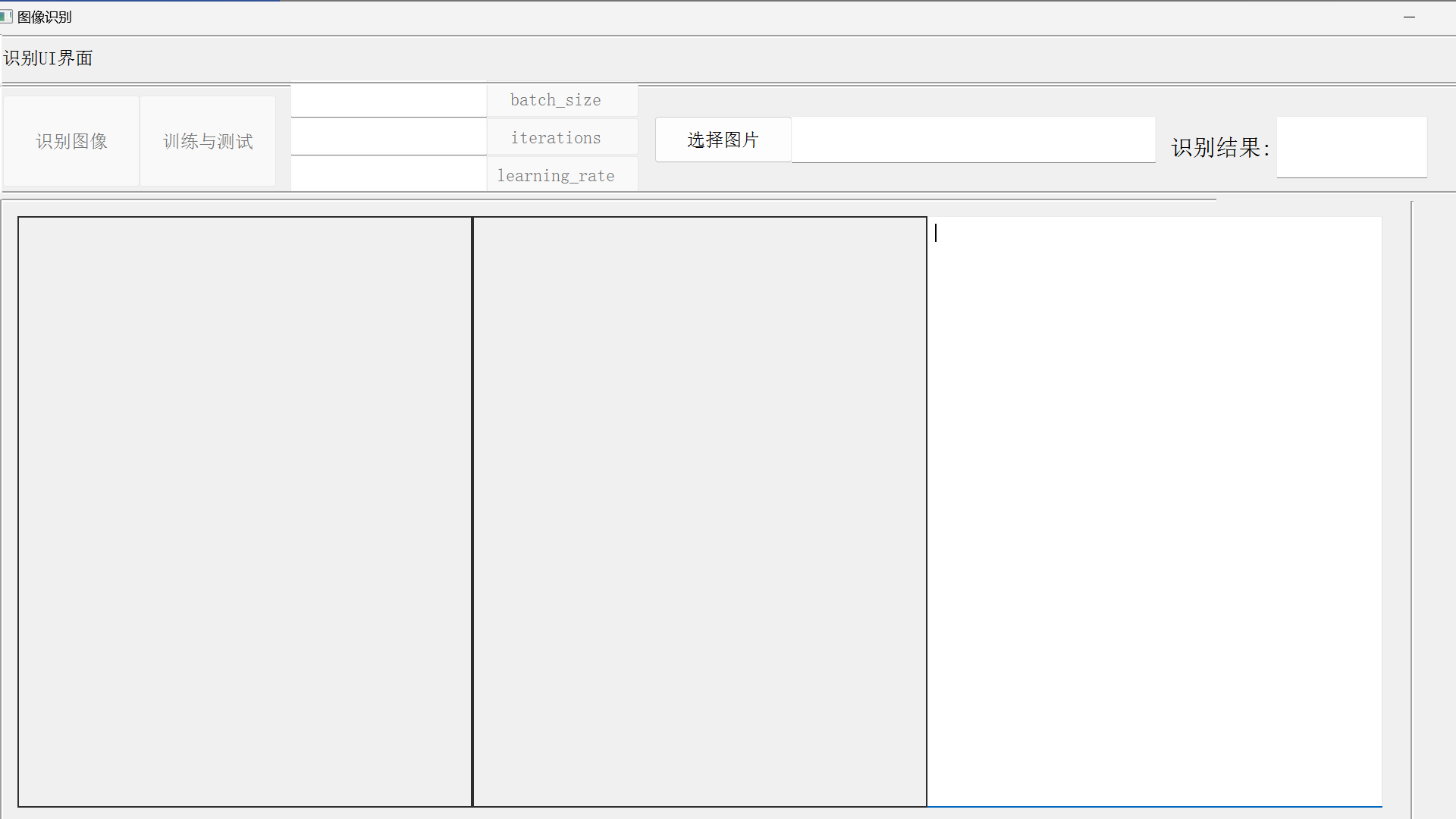
以上分别是更新参数batch\_size，iterations，learning\_rate的动作函数。ClickButton5函数与pushButton\_5的点击事件相关联。在点击batch\_size按钮后触发ClickButton5函数。函数获取用户在界面文本框lineEdit\_5输入的值，将之转换成int变量后传递给batch\_size变量。

* def openImage(self):  
   global imgNamepath *# 这里为了方便别的地方引用图片路径，将其设置为全局变量* imgNamepath, imgType = QFileDialog.getOpenFileName(self.centralwidget, "选择图片",  
   ".",  
   "All Files(\*);;\*.jpg;;\*.png")  
   *# 通过文件路径获取图片文件，并设置图片长宽为label控件的长、宽* img = QtGui.QPixmap(imgNamepath)  *# 在label控件上显示选择的图片* self.label\_3.setPixmap(img)  
   self.label\_3.setScaledContents(True)  
   *# 显示所选图片的路径* self.lineEdit\_4.setText(imgNamepath)  
   self.pushButton\_2.setEnabled(True)

openImage函数与pushButton\_4的点击事件相关联。在点击选择图片按钮后触发openImage函数。函数作用是由用户选择一张本地图片，将图片路径填写到选择图片后面的文本框中，并在界面左边显示该图片。

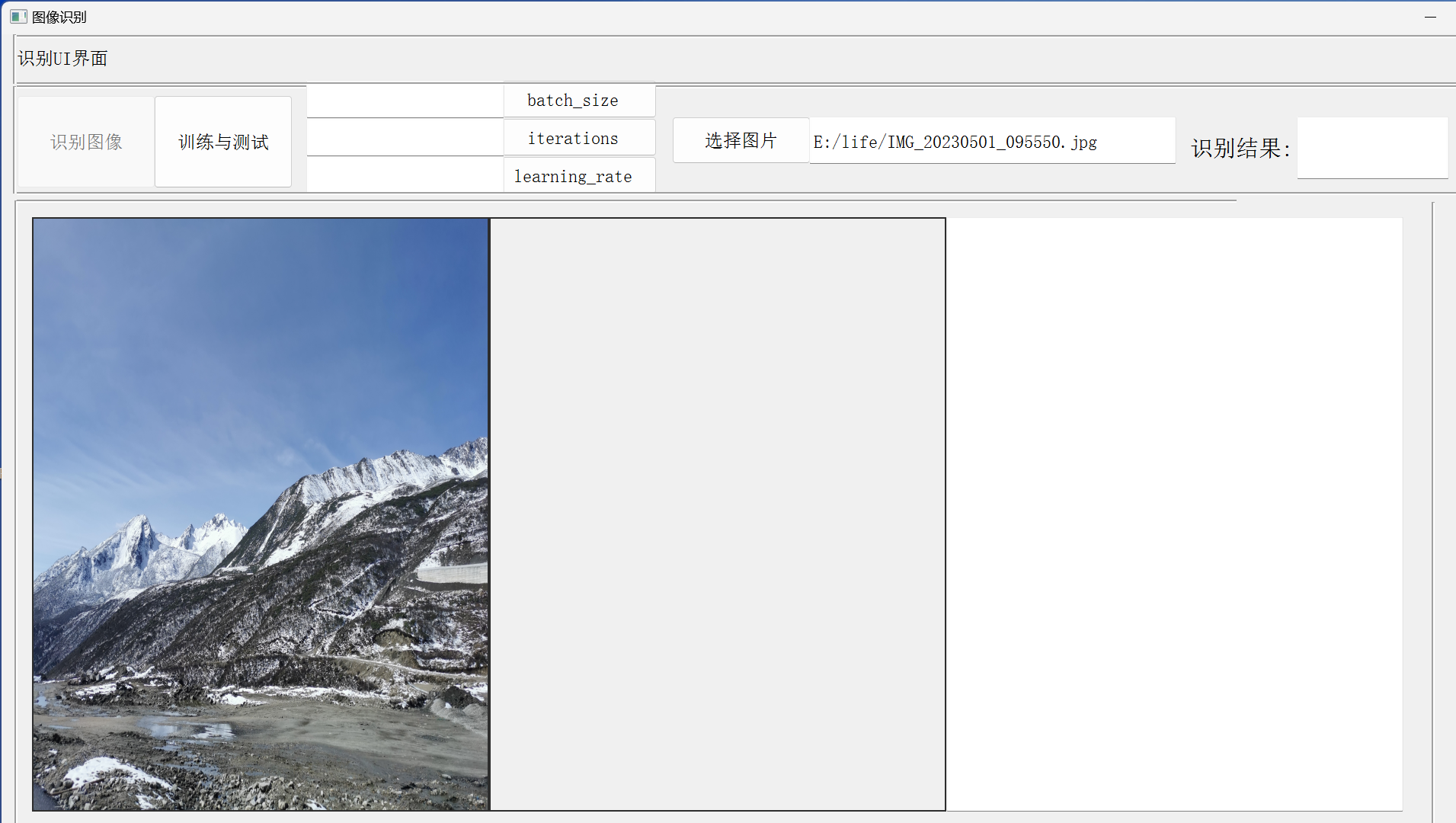
### 实验结果和分析

1. 初始UI界面如下图所示：



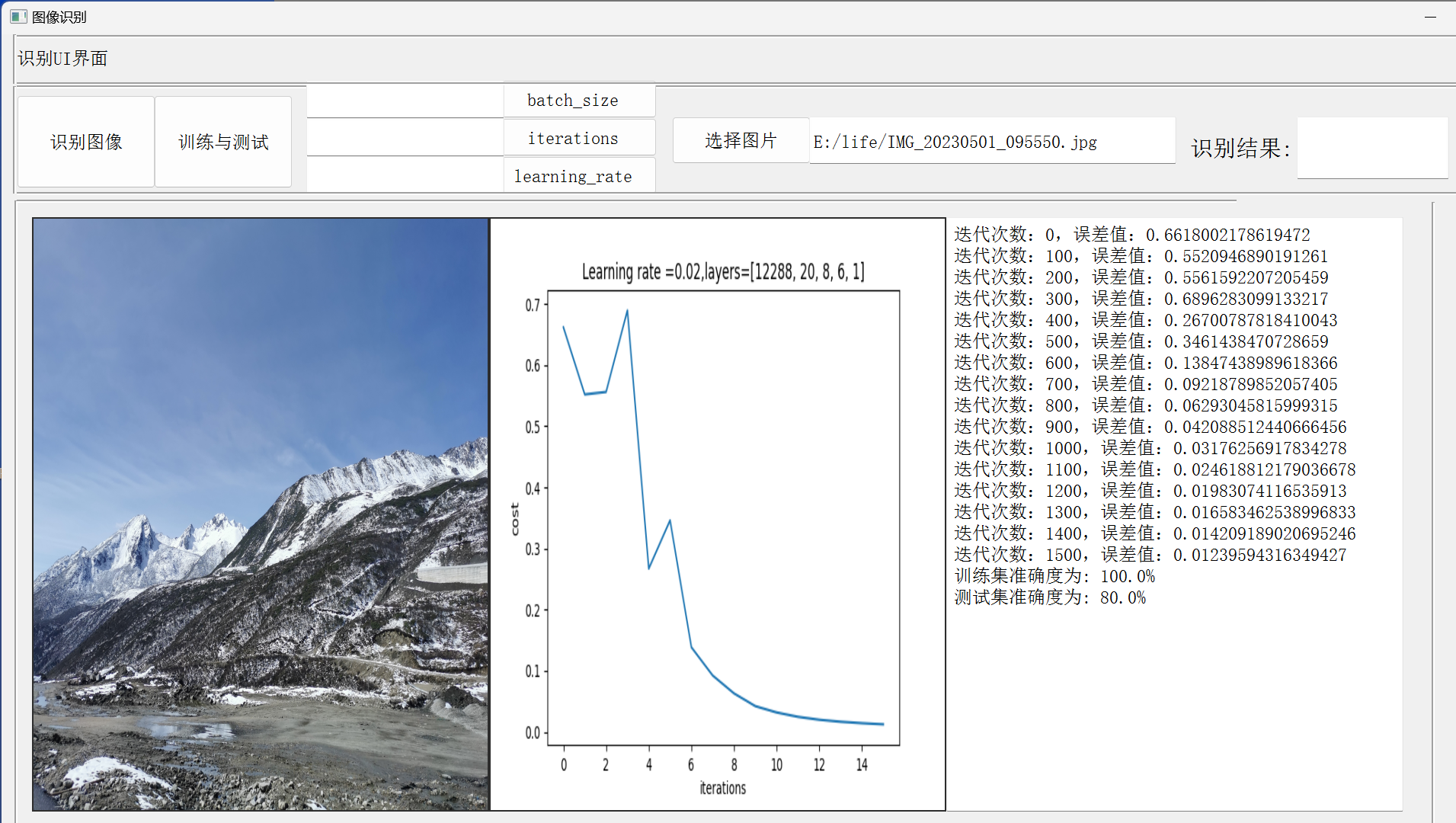
可见UI界面的元素有：文本框、标签、按钮、图像框等基本元素。

1. 首先我们需要选择一张本地图片。



可见选择图片按钮后面出现了图片路径，并且界面左侧展示图片。

1. 点击训练与测试按钮。



可见，在学习率为固定值0.02，迭代次数为1600情况下cost与iterations关系图像放在界面中间。模型训练过程和模型准确性打印在了界面右侧。训练集准确性达到了:100%，测试集准确性:80.0 %。

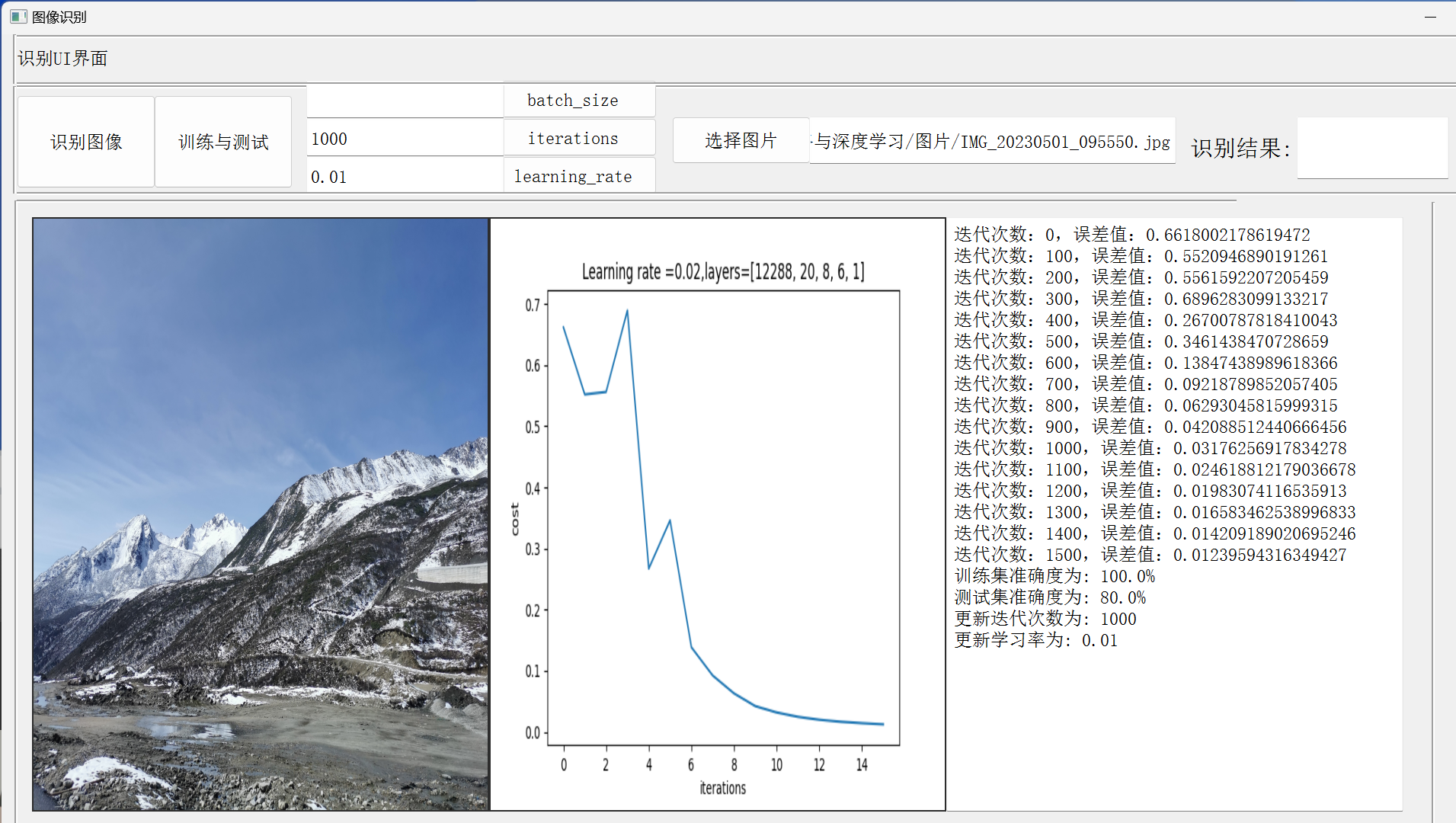
训练集的准确性高而测试集的准确性低，意味着模型存在过拟合的问题。过拟合指的是模型在训练数据上表现良好，但在未见过的测试数据上表现不佳的情况。

为了让渐变下降起作用，我们必须明智地选择学习速率。学习率α决定了我们更新参数的速度。如果学习率过高，我们可能会“超过”最优值。同样，如果它太小，我们将需要太多迭代才能收敛到最佳值。这就是为什么使用良好调整的学习率至关重要的原因。在报告中我们选择学习率α=0.002。

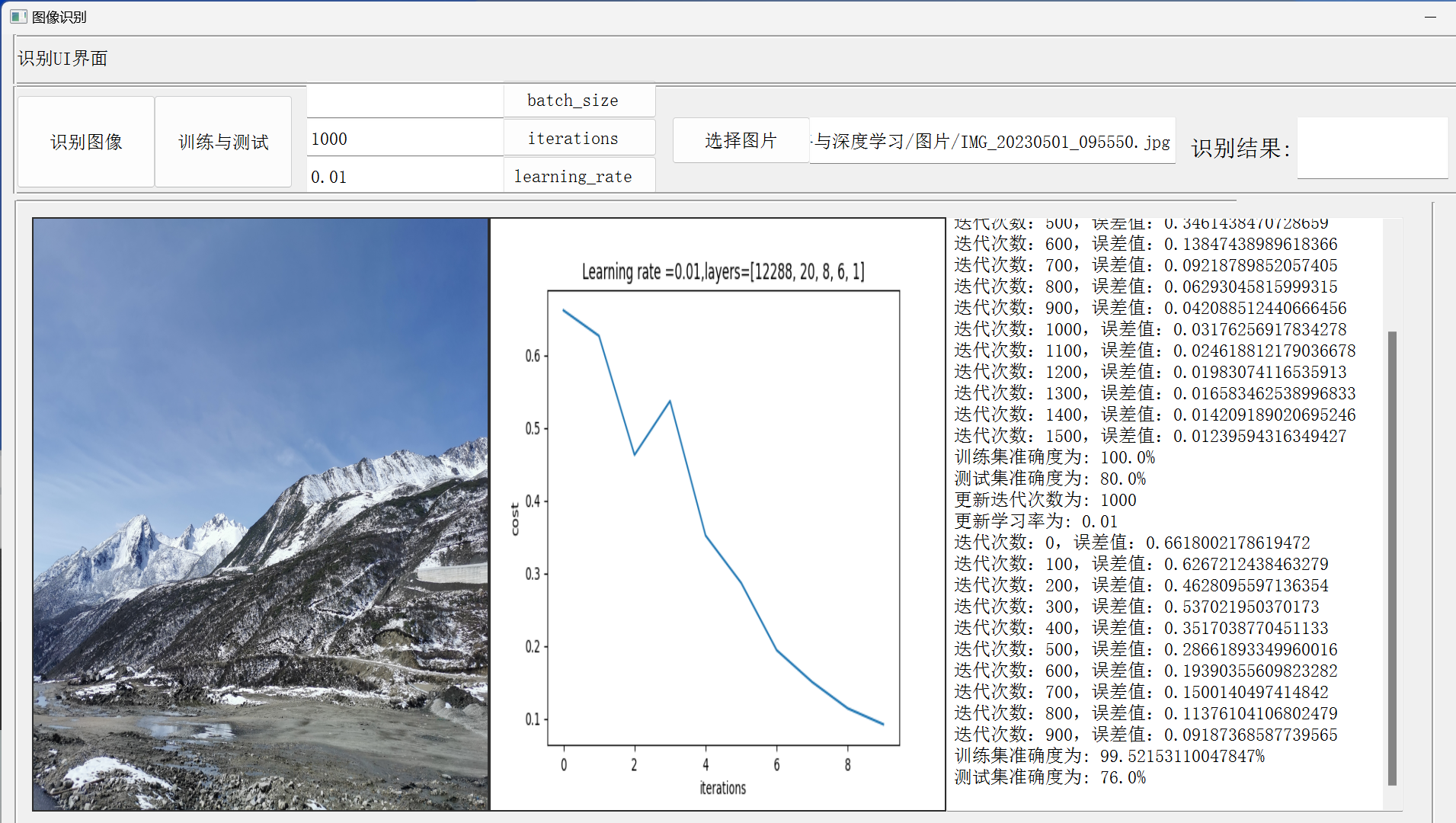
初始化后，batch\_size=64，iterations=1600，learning\_rate=0.02。为了方便地调整参数，在UI界面上展示batch\_size，iterations，learning\_rate的接口，用户可以根据训练集和测试集的结果来调整上述三个变量的值。

1. 更新参数

要将迭代次数更新为1000，学习率下降到0.01。将1000写入标签后点击iterations按钮，将0.01写入标签后点击learning\_rate按钮，如下图所示，交互界面打印更新迭代次数为：1000，更新学习率为：0.01。

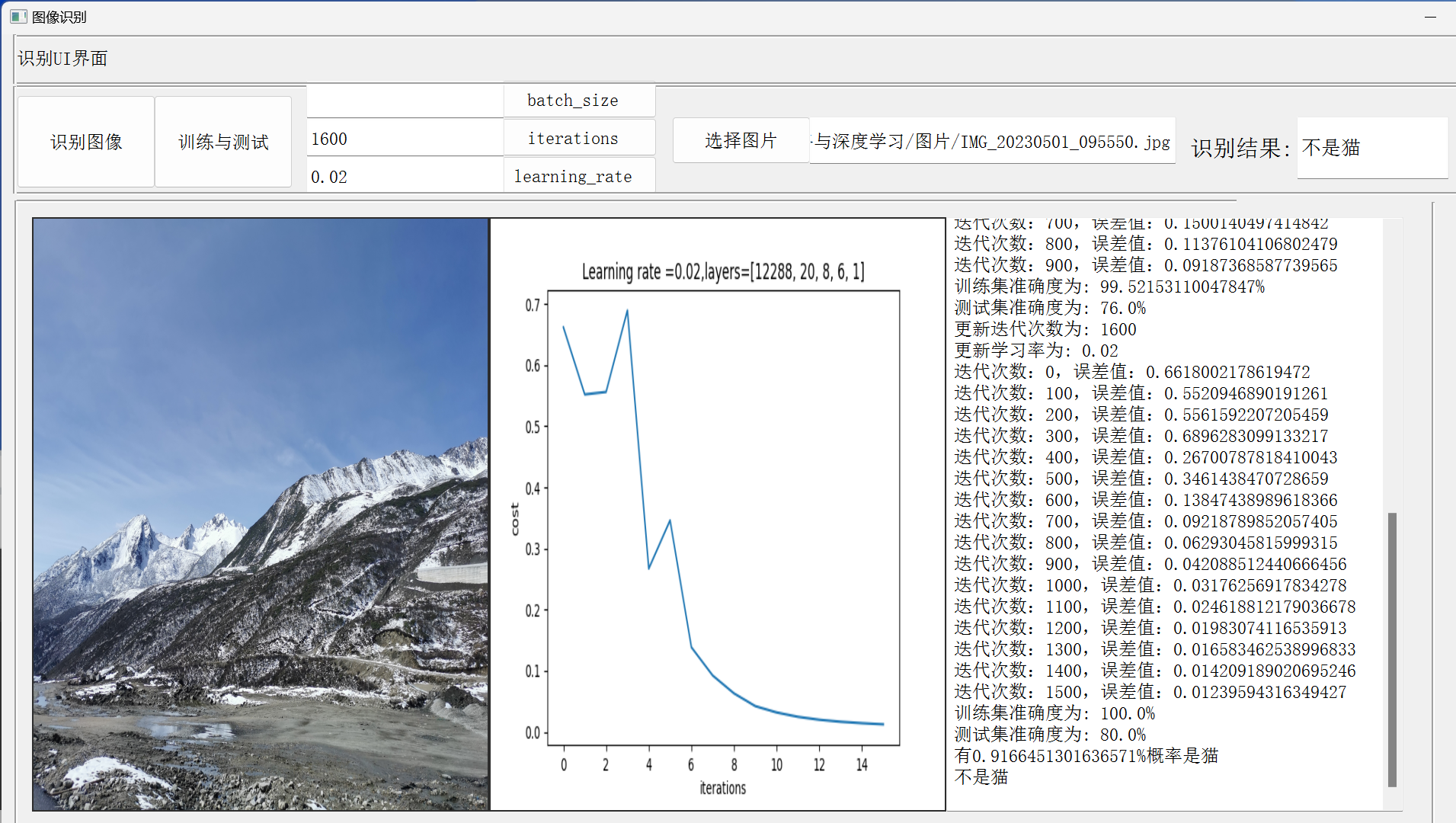


点击训练与测试按钮。



由上图可见，界面中间的cost随迭代次数变化的图像更新，图片上部learning\_rate=0.02，交互界面的训练集准确度和测试集准确度都下降了一些。

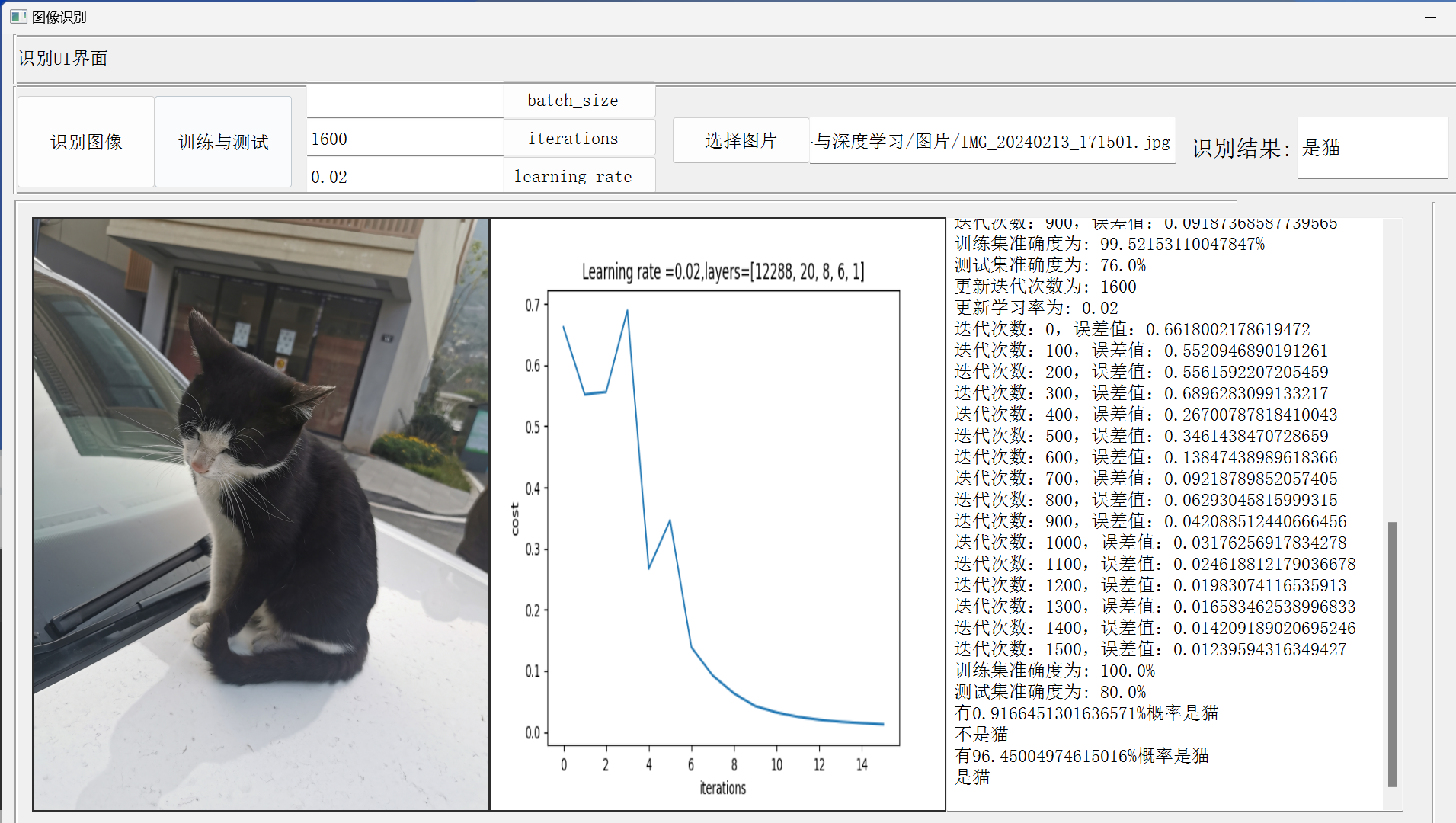
1. 在iterations=1600，learning\_rate=0.02的条件下，点击识别图像按钮

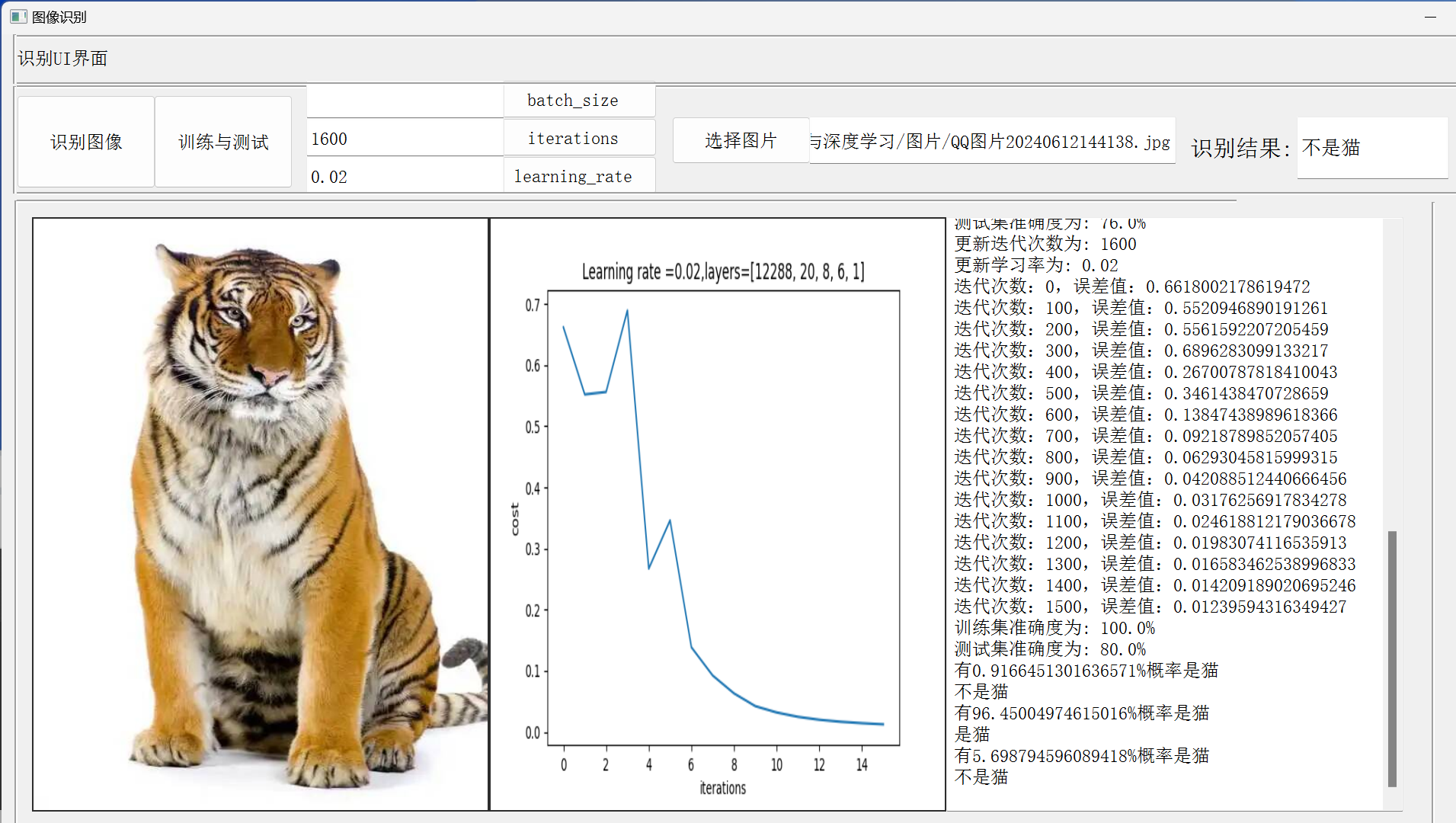


识别结果标签后的文字框和交互界面显示了识别结果为不是猫。

选择其他图片的效果展示：

重新选择图像后，再次点击识别图像按钮：





可见模型能以一定的正确概率识别图片是否是猫的图片。

### 总结

本报告构建了一个识别猫的机器学习模型和一个可视化界面，用户可以实现在界面调整模型参数、训练模型，通过上传本地任意图片来识别图片是否是猫，最终可以以一定的概率识别照片是否是猫。经过多次迭代更新模型的误差传输参数，最后训练集准确性达到了:100%，测试集准确性:80.0 %。在今后的工作中，若要提高测试集的准确性，我们可以：

* 增加数据量： 获取更多的训练数据可能是最直接有效的方法之一。更多的数据可以帮助模型更好地泛化，并且减少过拟合的可能性。
* 数据清洗和预处理： 确保数据集中没有噪声、异常值或缺失值，并对数据进行适当的预处理，例如标准化、归一化或特征缩放。
* 改变机器学习模型结构，增加神经网络中隐藏层的数量，或引入卷积神经网络模型。

### 附件

#### lr\_utils.py：

import numpy as np  
import h5py  
   
def load\_dataset():  
 train\_dataset = h5py.File('datasets/train\_catvnoncat.h5', "r")  
 train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) *# your train set features* train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) *# your train set labels* test\_dataset = h5py.File('datasets/test\_catvnoncat.h5', "r")  
 test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) *# your test set features* test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) *# your test set labels* classes = np.array(test\_dataset["list\_classes"][:]) *# the list of classes* train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  
 test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  
   
 return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes

#### 模型构建——Deeplearningcat.py

import numpy as np  
import os  
*# import cv2*import matplotlib.pyplot as plt  
from PIL import Image, ImageTk  
import tkinter as tk  
import tkinter.filedialog  
import h5py  
  
""" 0.数据读取与处理  
 1.初始化参数  
 2.前向线性传播  
 3.计算激活值  
 4.计算误差  
 5.反向传播  
 6.更新参数  
 7.预测  
 8.额外功能"""  
  
  
  
def Init\_params(layers): *# 初始化权重矩阵和偏置* np.random.seed(3) *# 保证每次初始化一样* parameters = {} *# 该字典用来储存参数* L = len(layers) *# 神经网络的层数* for l in range(1, L):  
 parameters["W" + str(l)] = np.random.randn(layers[l],  
 layers[l - 1]) / np.sqrt(layers[l - 1]) *# Xaiver初始化方法* parameters['b'+str(l)] = np.zeros((layers\_dims[l], 1)) *# 初始化为0* return parameters  
  
  
def TanH(Z):  
 return (np.exp(2\*Z)-1)/(np.exp(2\*Z)+1)  
  
  
def Sigmoid(Z):  
 return 1/(1+np.exp(-Z))  
  
  
def Linear\_forward(A, W, b): *# 正向线性传播* return np.dot(W, A) + b  
  
  
def Activation\_forward(A\_pre, W, b, Type='Hiden'): *# 计算激活值  
 """  
 Z表示经过线性传播后的矩阵，将输给激活函数  
 A\_pre表示前一层的激活值，将输给线性传播单元  
 b将先广播至与W一样的大小，再进行运算  
 """* Z = Linear\_forward(A\_pre, W, b)  
 cache = (A\_pre, W, b)  
 if Type == "Output":  
 A = Sigmoid(Z)  
 elif Type == "Hiden":  
 A = TanH(Z)  
  
 return A, cache  
  
  
def Forward\_propagation(X, parameters): *# 向前传播  
 """  
 caches用于储存cache  
 每一层的激活值A将输给下一层并作用于线性传播函数  
 输出层的激活值为Yhat，将输给代价函数  
 """* caches = []  
 A = X  
 L = len(parameters) // 2 *# 获得整型* for l in range(1, L): *# （1,3）* A, cache = Activation\_forward(  
 A, parameters['W' + str(l)], parameters['b' + str(l)], "Hiden")  
 caches.append(cache)  
 Yhat, cache = Activation\_forward(  
 A, parameters['W' + str(L)], parameters['b' + str(L)], "Output")  
 caches.append(cache)  
  
 return Yhat, caches  
  
  
def Compute\_cost(Yhat, Y):  
 m = Y.shape[1] *# 图片张数* cost = -np.sum(np.multiply(np.log(Yhat), Y) +  
 np.multiply(np.log(1 - Yhat), 1 - Y)) / m *# 交叉熵误差计算  
 # 计算Yhat的梯度，由此开始反向传播* dYhat = - (np.divide(Y, Yhat) - np.divide(1 - Y, 1 - Yhat))  
 return cost, dYhat  
  
  
def Linear\_backward(dZ, cache):  
 A, W, b = cache *# 拆分cache* m = A.shape[1] *# 获得图片张数  
 # 除以m防止样本过大而导致数据过大* dW = np.dot(dZ, A.T) / m *# dW/dZ=A.T,相乘代表与cost的梯度* db = np.sum(dZ, axis=1, keepdims=True) / m *# db/dZ=I,保持维度不变* dA = np.dot(W.T, dZ)  
  
 return dA, dW, db  
  
  
def Sigmoid\_backward(dA, A):  
 dZ = dA \* A\*(1-A) *# 相对误差的梯度* return dZ  
  
  
def TanH\_backward(dA, A):  
 dZ = dA\*(1-A\*\*2) *# 相对误差的梯度* return dZ  
  
  
def Activation\_backward(dA, cache, A\_next, activation="Hiden"):  
 *"""  
 cache储存A\_pre，W，b  
 A\_next为输给下一层的激活值，即本层的激活值  
 """* if activation == "Hiden":  
 dZ = TanH\_backward(dA, A\_next)  
 elif activation == "Output":  
 dZ = Sigmoid\_backward(dA, A\_next)  
 dA, dW, db = Linear\_backward(dZ, cache)  
  
 return dA, dW, db  
  
  
def Backward\_propagation(dYhat, Yhat, Y, caches):  
 grads = {} *# 用于储存梯度矩阵* L = len(caches) *# 4* m = Y.shape[1] *# 图片个数  
 # 输出层* grads["dA" + str(L)], grads["dW" + str(L)], grads["db" + str(L)  
 ] = Activation\_backward(dYhat, caches[L-1], Yhat, "Output")  
 for l in reversed(range(L-1)): *# (3,0]* grads["dA" + str(l + 1)], grads["dW" + str(l + 1)], grads["db" + str(l + 1)  
 ] = Activation\_backward(grads["dA" + str(l + 2)], caches[l], caches[l+1][0], "Hiden")  
  
 return grads  
  
  
def Update\_params(parameters, grads, learning\_rate):  
 *# 梯度下降更新参数* L = len(parameters) // 2  
 for l in range(L):  
 parameters["W" + str(l + 1)] -= learning\_rate \* \  
 grads["dW" + str(l + 1)]  
 parameters["b" + str(l + 1)] -= learning\_rate \* \  
 grads["db" + str(l + 1)]  
 return parameters  
  
  
def Predict(X, parameters): *# 预测  
 # 将数据和训练好的参数进行一次预测* probs, caches = Forward\_propagation(X, parameters)  
 if X.shape[1] == 1: *# 判断输入的是否是一张图片* print(f'有{probs[0][0]\*100}%概率是猫')  
 return probs  
  
  
def Accuracy(name, probs, y): *# 准确率* p = [] *# 储存预测值* for i in range(0, probs.shape[1]):  
 p.append(int(1) if probs[0, i] > 0.5 else int(0)) *# 分类* print(f"{name}准确度为: {100\*np.mean((p == y))}%")  
  
 return p  
  
  
def Save\_params(parameters, layers, path):  
 *# 储存神经网络各层的信息* np.savetxt(path+'layers.csv', layers, delimiter=',')  
 n = len(parameters)//2  
 *# 将每个参数分开储存，方便读取* for i in range(1, n+1):  
 np.savetxt(path+'W'+str(i)+'.csv',  
 parameters['W'+str(i)], delimiter=',')  
 np.savetxt(path+'b'+str(i)+'.csv',  
 parameters['b'+str(i)], delimiter=',')  
  
  
def Load\_params(path):  
 parameters = {} *# 用于接收参数* layers = list(np.loadtxt(path+'layers.csv', dtype=int, delimiter=','))  
 n = len(layers)  
 for i in range(1, n):  
 parameters['W'+str(i)] = np.loadtxt(path+'W'+str(i) +  
 '.csv', delimiter=",").reshape(layers[i], -1)  
 parameters['b'+str(i)] = np.loadtxt(path+'b'+str(i) +  
 '.csv', delimiter=",").reshape(layers[i], 1)  
 return layers, parameters  
  
def Plot(costs, layers, learning\_rate):  
 *"""  
 costs储存每100次迭代后的误差值  
 layers是神经网络的信息  
 """* plt.plot(np.squeeze(costs))  
 plt.ylabel('cost')  
 plt.xlabel('iterations')  
 plt.title("Learning rate =" +  
 str(learning\_rate) + f",layers={layers}")  
 save\_path = 'datasets\plot.png' *# 指定保存的路径和文件名* plt.savefig(save\_path)  
 plt.show()  
  
def Find\_wrong(p, label, data):  
 *"""  
 p为预测的标签，是一个列表  
 label为真实的标签，是一个1行n列的矩阵  
 data为经过处理后的图片矩阵  
 """* m = len(p)  
 diff = [] *# 储存预测错误图片的索引值* for i in range(m):  
 if label[0, i] != p[i]:  
 diff.append(i)  
  
def Train\_model(X, Y, parameters, learning\_rate, iterations, threshold): *# 训练用模块* costs = [] *# 储存每100次迭代的损失值，用于绘制折线图* for i in range(iterations):  
 Yhat, caches = Forward\_propagation(X, parameters) *# 正向传播* cost, dYhat = Compute\_cost(Yhat, Y) *# 计算误差* grads = Backward\_propagation(dYhat, Yhat, Y, caches) *# 计算梯度* parameters = Update\_params(  
 parameters, grads, learning\_rate) *# 更新参数* if i % 100 == 0:  
 costs.append(cost)  
 print(f"迭代次数：{i}，误差值：{cost}")  
 if cost < threshold:  
 costs.append(cost)  
 print(f"迭代次数：{i}，误差值：{cost}")  
 break  
 return parameters, costs, i  
  
  
def BP(X, Y, test\_data, test\_label, path, layers, iterations, learning\_rate, threshold=0.06,  
 find\_wrong=False, save\_params=False, load\_params=False, continue\_train=False, plot=True):  
 parameters = Init\_params(layers) *# 接收初始化的参数* if load\_params or continue\_train: *# 继续训练也需要读取储存的参数* layers, parameters = Load\_params(path)  
 *# 当不继续训练或不加载数据时* if continue\_train or not load\_params:  
 parameters, costs, times = Train\_model(  
 X, Y, parameters, learning\_rate, iterations, threshold)  
 if save\_params:  
 Save\_params(parameters, layers, path)  
 train\_probs = Predict(X, parameters)  
 train\_p = Accuracy('训练集', train\_probs, Y)  
 test\_probs = Predict(test\_data, parameters)  
 test\_p = Accuracy('测试集', test\_probs, test\_label)  
 if find\_wrong:  
 Find\_wrong(train\_p, Y, X)  
 Find\_wrong(test\_p, test\_label, test\_data)  
 if plot:  
 Plot(costs, layers, learning\_rate)  
 return parameters  
  
  
def load\_dataset():  
 train\_dataset = h5py.File('datasets/train\_catvnoncat.h5', "r")  
 train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) *# your train set features* train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) *# your train set labels* test\_dataset = h5py.File('datasets/test\_catvnoncat.h5', "r")  
 test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) *# your test set features* test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) *# your test set labels* classes = np.array(test\_dataset["list\_classes"][:]) *# the list of classes* train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  
 test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  
  
 return train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes  
  
def switch(s, addr, parameters, iterations, learning\_rate):  
 if s==1:  
 *# 文件的路径* Path\_params = 'E:/'  
  
 train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y, classes = load\_dataset()  
 *# 测试集和训练集和图片矩阵纵向维度保持一致  
 # 将训练集的维度降低并转置。* train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0], -1).T  
 *# 将测试集的维度降低并转置。* test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0], -1).T  
  
 train\_set\_x = train\_set\_x\_flatten / 255  
 test\_set\_x = test\_set\_x\_flatten / 255  
  
 *# 各层的节点数* global layers\_dims  
 layers\_dims = [train\_set\_x.shape[0], 20, 8, 6, 1]  
 *# global learning\_rate  
 # learning\_rate = 0.0075  
  
 # global parameters* parameters=BP(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, Path\_params, layers\_dims,  
 iterations, learning\_rate, threshold=0.006, find\_wrong=False, save\_params=False,  
 load\_params=False, continue\_train=False, plot=True)  
 """  
 train\_data，train\_label为训练集的图片数据和标签，test\_data，test\_label为测试集数据和标签  
 path为保存和读取参数的绝对路径n  
 layers储存了每层神经元个数  
 interactions为迭代次数  
 threshold为阈值，误差小于该值可结束程序  
  
 find\_wrong：是否显示识别错误的图像  
 save\_params：是否储存训练好的参数  
 load\_params：是否读取参数  
 continue\_train：是否读取参数继续训练  
 plot：是否显示误差与迭代次数关系的图像  
 test：是否自己选取测试用图像  
 """  
 return parameters  
 if s==0:  
 *# 图片路径* image\_path = addr  
  
 image = Image.open(image\_path)  
  
 *# 将图像调整为 64x64 大小* resized\_image = image.resize((64, 64))  
  
 *# 将图像数据转换为 NumPy 数组* image\_array = np.array(resized\_image)  
  
 *# 创建 HDF5 文件并保存图像数据、list\_classes 和 train\_set\_y* with h5py.File('datasets/output\_image.h5', 'w') as h5\_file:  
 h5\_file.create\_dataset('output\_x', data=image\_array.reshape(1, 64, 64, 3))  
  
 output\_dataset = h5py.File('datasets/output\_image.h5', "r")  
 output\_x\_orig = np.array(output\_dataset["output\_x"][:]) *# your train set features  
 # 将图像的维度降低并转置。* output\_x\_flatten = output\_x\_orig.reshape(output\_x\_orig.shape[0], -1).T  
 output\_x = output\_x\_flatten / 255  
 probs = Predict(output\_x,parameters)  
 if probs[0][0] >= 0.5:  
 print('是猫')  
 return 1  
 else:  
 print("不是猫")  
 return 0

#### UI界面开发——UI,py

from PyQt5 import QtCore, QtGui, QtWidgets  
from PyQt5.Qt import \*  
from Deeplearningcat import switch  
import sys  
  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 *# MainWindow.setWindowIcon(QIcon('..\icons\zhenyanglog.png'))* MainWindow.resize(1025, 600)  
 MainWindow.setFixedSize(1025, 600)  
 self.outputTextEdit = QtWidgets.QTextEdit(MainWindow)  
 self.outputTextEdit.setGeometry(QtCore.QRect(620, 120, 300, 390))  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(10, 0, 1001, 31))  
 self.label.setObjectName("label")  
 self.line = QtWidgets.QFrame(self.centralwidget)  
 self.line.setGeometry(QtCore.QRect(10, 25, 1011, 21))  
 self.line.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line.setObjectName("line")  
 self.pushButton = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton.setGeometry(QtCore.QRect(10, 40, 91, 61))  
 self.pushButton.setObjectName("pushButton")  
  
 self.pushButton\_2 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_2.setGeometry(QtCore.QRect(100, 40, 91, 61))  
 self.pushButton\_2.setObjectName("pushButton\_2")  
  
 self.pushButton\_4 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_4.setGeometry(QtCore.QRect(440, 54, 91, 31))  
 self.pushButton\_4.setObjectName("pushButton\_4")  
  
 self.pushButton\_5 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_5.setGeometry(QtCore.QRect(320, 30, 110, 25))  
 self.pushButton\_5.setObjectName("pushButton\_5")  
  
 self.pushButton\_6 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_6.setGeometry(QtCore.QRect(320, 55, 110, 25))  
 self.pushButton\_6.setObjectName("pushButton\_6")  
  
 self.pushButton\_7 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_7.setGeometry(QtCore.QRect(320, 80, 110, 25))  
 self.pushButton\_7.setObjectName("pushButton\_7")  
  
 font3 = QFont()  
 font3.setPointSize(17)  
  
  
 self.pushButton.setEnabled(False)  
 self.pushButton\_2.setEnabled(False)  
 self.pushButton\_5.setEnabled(False)  
 self.pushButton\_6.setEnabled(False)  
 self.pushButton\_7.setEnabled(False)  
 *# self.pushButton\_3.setEnabled(False)  
  
 # 按钮关联函数* self.pushButton.clicked.connect(self.ClickButton1)  
 self.pushButton\_2.clicked.connect(self.ClickButton2)  
 self.pushButton\_4.clicked.connect(self.openImage)  
 self.pushButton\_5.clicked.connect(self.ClickButton5)  
 self.pushButton\_6.clicked.connect(self.ClickButton6)  
 self.pushButton\_7.clicked.connect(self.ClickButton7)  
  
 self.lineEdit\_5 = QtWidgets.QLineEdit(self.centralwidget)  
 self.lineEdit\_5.setGeometry(QtCore.QRect(200, 30, 130, 25))  
 self.lineEdit\_5.setObjectName("lineEdit\_5")  
 self.lineEdit\_6 = QtWidgets.QLineEdit(self.centralwidget)  
 self.lineEdit\_6.setGeometry(QtCore.QRect(200, 55, 130, 25))  
 self.lineEdit\_6.setObjectName("lineEdit\_6")  
 self.lineEdit\_7 = QtWidgets.QLineEdit(self.centralwidget)  
 self.lineEdit\_7.setGeometry(QtCore.QRect(200, 80, 130, 25))  
 self.lineEdit\_7.setObjectName("lineEdit\_7")  
  
 self.lineEdit\_4 = QtWidgets.QLineEdit(self.centralwidget)  
 self.lineEdit\_4.setGeometry(QtCore.QRect(530, 54, 241, 31))  
 self.lineEdit\_4.setObjectName("lineEdit\_4")  
 self.line\_2 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_2.setGeometry(QtCore.QRect(10, 95, 1011, 21))  
 self.line\_2.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line\_2.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_2.setObjectName("line\_2")  
  
 self.line\_5 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_5.setGeometry(QtCore.QRect(-20, 109, 60, 486))  
 self.line\_5.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_5.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_5.setObjectName("line\_5")  
 self.line\_6 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_6.setGeometry(QtCore.QRect(-10, 34, 38, 71))  
 self.line\_6.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_6.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_6.setObjectName("line\_6")  
 self.line\_7 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_7.setGeometry(QtCore.QRect(1000, 34, 38, 71))  
 self.line\_7.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_7.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_7.setObjectName("line\_7")  
 self.line\_8 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_8.setGeometry(QtCore.QRect(970, 0, 98, 37))  
 self.line\_8.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_8.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_8.setObjectName("line\_8")  
 self.line\_9 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_9.setGeometry(QtCore.QRect(-10, 0, 38, 33))  
 self.line\_9.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_9.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_9.setObjectName("line\_9")  
 self.line\_10 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_10.setGeometry(QtCore.QRect(10, 10, 1011, 46))  
 self.line\_10.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line\_10.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_10.setObjectName("line\_10")  
 self.line\_11 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_11.setGeometry(QtCore.QRect(10, -9, 1011, 22))  
 self.line\_11.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line\_11.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_11.setObjectName("line\_11")  
  
 self.line\_13 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_13.setGeometry(QtCore.QRect(10, 95, 801, 31))  
 self.line\_13.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line\_13.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_13.setObjectName("line\_13")  
  
 self.line\_15 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_15.setGeometry(QtCore.QRect(10, 566, 801, 61))  
 self.line\_15.setFrameShape(QtWidgets.QFrame.HLine)  
 self.line\_15.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_15.setObjectName("line\_15")  
 self.line\_16 = QtWidgets.QFrame(self.centralwidget)  
 self.line\_16.setGeometry(QtCore.QRect(920, 110, 41, 486))  
 self.line\_16.setFrameShape(QtWidgets.QFrame.VLine)  
 self.line\_16.setFrameShadow(QtWidgets.QFrame.Sunken)  
 self.line\_16.setObjectName("line\_16")  
 self.label\_3 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_3.setGeometry(QtCore.QRect(20, 120, 300, 390))  
 self.label\_3.setStyleSheet("font:28px;\n"  
 "border-style:solid;\n"  
 "border-width:1px;\n"  
 "border-color:rgb(45, 45, 45);\n"  
 "\n"  
 "")  
 self.label\_3.setText("")  
 self.label\_3.setObjectName("label\_3")  
  
 self.label\_30 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_30.setGeometry(QtCore.QRect(320, 120, 300, 390))  
 self.label\_30.setStyleSheet("font:28px;\n"  
 "border-style:solid;\n"  
 "border-width:1px;\n"  
 "border-color:rgb(45, 45, 45);\n"  
 "\n"  
 "")  
 self.label\_30.setText("")  
 self.label\_30.setObjectName("label\_30")  
  
 self.label\_5 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_5.setGeometry(QtCore.QRect(780, 54, 191, 41))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 self.label\_5.setFont(font)  
 self.label\_5.setLayoutDirection(QtCore.Qt.LeftToRight)  
 self.label\_5.setAlignment(QtCore.Qt.AlignLeading | QtCore.Qt.AlignLeft | QtCore.Qt.AlignVCenter)  
 self.label\_5.setWordWrap(True)  
 self.label\_5.setObjectName("label\_5")  
 self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)  
 self.lineEdit.setGeometry(QtCore.QRect(850, 54, 100, 41))  
 font = QtGui.QFont()  
 font.setPointSize(10)  
 self.lineEdit.setFont(font)  
 self.lineEdit.setObjectName("lineEdit")  
 self.label\_6 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_6.setGeometry(QtCore.QRect(820, 280, 191, 41))  
  
 MainWindow.setCentralWidget(self.centralwidget)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
  
 def ClickButton1(self):  
 self.lineEdit.clear()  
 res = switch(0, imgNamepath,parameters,iterations, learning\_rate)  
 if res == 1:  
 self.lineEdit.setText("是猫")  
 else:  
 self.lineEdit.setText("不是猫")  
 *# self.distance()* def ClickButton2(self):  
 global parameters  
 parameters=switch(1, imgNamepath,0,iterations, learning\_rate)  
 img\_path = 'datasets\plot.png' *# 图像路径* pixmap = QPixmap(img\_path)  
 self.label\_30.setPixmap(pixmap)  
 self.label\_30.setScaledContents(True)  
 self.pushButton.setEnabled(True)  
  
 def ClickButton5(self):  
 global batch\_size  
 input\_value1 = self.lineEdit\_5.text()  
 batch\_size = int(input\_value1)  
 print(f'更新batch\_size为: {batch\_size}')  
 def ClickButton6(self):  
 global iterations  
 input\_value2 = self.lineEdit\_6.text()  
 iterations = int(input\_value2)  
 print(f'更新迭代次数为: {iterations}')  
  
 def ClickButton7(self):  
 global learning\_rate  
 input\_value3 = self.lineEdit\_7.text()  
 learning\_rate = float(input\_value3)  
 print(f'更新学习率为: {learning\_rate}')  
  
 *# 系统目录方法* def initUI(self, Qmodelidx):  
 self.label\_3.clear()  
 self.label\_30.clear()  
  
 *# 选择本地图片上传* def openImage(self):  
 global imgNamepath *# 这里为了方便别的地方引用图片路径，将其设置为全局变量* imgNamepath, imgType = QFileDialog.getOpenFileName(self.centralwidget, "选择图片",  
 ".",  
 "All Files(\*);;\*.jpg;;\*.png")  
 *# 通过文件路径获取图片文件，并设置图片长宽为label控件的长、宽  
 # print(imgNamepath)* img = QtGui.QPixmap(imgNamepath) *# .scaled(self.label\_3.width(), self.label\_3.height())  
 # 在label控件上显示选择的图片* self.label\_3.setPixmap(img)  
 self.label\_3.setScaledContents(True)  
 *# 显示所选图片的路径* self.lineEdit\_4.setText(imgNamepath)  
  
 self.pushButton\_2.setEnabled(True)  
 self.pushButton\_5.setEnabled(True)  
 self.pushButton\_6.setEnabled(True)  
 self.pushButton\_7.setEnabled(True)  
  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "图像识别"))  
 self.label.setText(\_translate("MainWindow", "识别UI界面"))  
 self.pushButton.setText(\_translate("MainWindow", "识别图像"))  
 self.pushButton\_2.setText(\_translate("MainWindow", "训练与测试"))  
 self.label\_5.setText(\_translate("MainWindow", "识别结果:"))  
 self.pushButton\_4.setText(\_translate("MainWindow", "选择图片"))  
 self.pushButton\_5.setText(\_translate("MainWindow", "batch\_size"))  
 self.pushButton\_6.setText(\_translate("MainWindow", "iterations"))  
 self.pushButton\_7.setText(\_translate("MainWindow", "learning\_rate"))  
 def redirect\_output(self):  
 sys.stdout = EmittingStream(textWritten=self.normalOutputWritten)  
  
 def normalOutputWritten(self, text):  
 cursor = self.outputTextEdit.textCursor()  
 cursor.movePosition(QtGui.QTextCursor.End)  
 cursor.insertText(text)  
 self.outputTextEdit.setTextCursor(cursor) *# 设置完文本后，重新设置光标位置* self.outputTextEdit.ensureCursorVisible()  
  
class EmittingStream(QtCore.QObject):  
 textWritten = QtCore.pyqtSignal(str)  
  
 def write(self, text):  
 self.textWritten.emit(str(text))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# global iterations, learning\_rate, batch\_size* iterations=1600  
 learning\_rate=0.02  
 batch\_size=10  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow()  
 ui = Ui\_MainWindow()  
 ui.setupUi(MainWindow)  
 ui.redirect\_output()  
 MainWindow.show()  
 sys.exit(app.exec\_())