# 一、需求分析

在这个手写数字识别web应用中，使用了Python来编写整个项目，使用tensorflow和keras来建立模型，将建立的模型保存到根目录下，利用base64接收web发送来的图像数据，利用opencv2来处理图像数据，再用建立的模型读取处理完的数据，发送会web应用，已完成对数字的预测。

Web端利用canvas画出图像，利用Jquery发送canvas的图像和接收Flask返回的数据，并将呈现在页面上。

# 二、详细设计

在本阶段中，确定应该如何具体地实现所要求，从而在编码阶段可以把这个描述直接翻译成用具体的程序语言。主要的工作有：

1.接受：接收图像数据；

2.处理：读取数字模型，解析收到的图像；

3.输出：将处理完的数据返回；

# 三、代码实现

## 在Python中利用Numpy和keras实现MNIST数字分类

MNIST数据集(Mixed National Institute of Standards and Technology database)是美国国家标准与技术研究院收集整理的大型手写数字数据库,包含60,000个示例的训练集以及10,000个示例的测试集。

### 1、在Python中导入要使用的库

|  |
| --- |
| import numpy as np from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense, Dropout, Flatten from keras.layers.convolutional import Conv2D, MaxPooling2D from keras.utils import np\_utils from keras.preprocessing.image import ImageDataGenerator |

### 2、测试的数据集

|  |
| --- |
| # 加载数据 (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() print(X\_train, y\_train) # 重塑为 [samples][width][height][channels] //[样本] [宽度] [高度] [通道] X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1)).astype('float32') X\_test = X\_test.reshape((X\_test.shape[0], 28, 28, 1)).astype('float32') # 将输入从 0-255 标准化为 0-1 X\_train, X\_test = normalize\_x(X\_train, X\_test) # 一个热编码输出 y\_train = np\_utils.to\_categorical(y\_train) y\_test = np\_utils.to\_categorical(y\_test) num\_classes = y\_test.shape[1] |

### 3、超参数的设置

|  |
| --- |
| # 超参数的常数配置 BATCH\_SIZE = 100 EPOCHS = 10 VALIDATION\_FRACTION\_SIZE = 0.10 ROTATION\_RANGE = 10 ZOOM\_RANGE = 0.10 SHIFT\_RANGE = 0.10 SHEAR\_RANGE = 0.10 |

### 4、定义模型

|  |
| --- |
| def larger\_model():  # 创建模型  model = Sequential()  model.add(Conv2D(30, (5, 5), input\_shape=(28, 28, 1), activation='relu'))  model.add(MaxPooling2D())  model.add(Conv2D(15, (3, 3), activation='relu'))  model.add(MaxPooling2D())  model.add(Dropout(0.2))  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(num\_classes, activation='softmax'))  # 编译模型  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  return model |

### 5、构造并保存该模型

|  |
| --- |
| # 构造模型 model = larger\_model() # 拟合模型 # model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200) model.fit(  datagen.flow(  X\_train, y\_train,  batch\_size=BATCH\_SIZE,  subset='training'  ),  validation\_data=datagen.flow(  X\_train, y\_train,  batch\_size=BATCH\_SIZE,  subset='validation'  ),  steps\_per\_epoch=(X\_train.shape[0] \* (1 - VALIDATION\_FRACTION\_SIZE)) // BATCH\_SIZE,  epochs=EPOCHS, ) # 保存模型 model.save('classifier.h5') |

### 6、输出返回模型的最终评估

|  |
| --- |
| # 模型的最终评估 scores = model.evaluate(X\_test, y\_test, verbose=0) print("Large CNN Error: %.2f%%" % (100 - scores[1] \* 100)) |

## 在Python中利用OpenCV和Numpy处理收到的图片数据

### 1、导入要使用的库

|  |
| --- |
| import numpy as np import base64 import cv2 import math from scipy import ndimage |

### 2、处理收到的图像模块

|  |
| --- |
| def preprocess(img):  """  : param img : 输入黑白图像  : returns img : 处理过的图像  """  return cv2.resize(img, (28, 28))  # 裁剪出完全白色的边缘  while int(np.mean(img[0])) == 255:  img = img[1:]  while np.mean(img[:, 0]) == 255:  img = np.delete(img, 0, 1)  while np.mean(img[-1]) == 255:  img = img[:-1]  while np.mean(img[:, -1]) == 255:  img = np.delete(img, -1, 1)  # 调整到合适的形状  rows, cols = img.shape  if rows > cols:  factor = 20.0 / rows  rows = 20  cols = int(round(cols \* factor))  img = cv2.resize(img, (cols, rows))  else:  factor = 20.0 / cols  cols = 20  rows = int(round(rows \* factor))  img = cv2.resize(img, (cols, rows))   # 添加预定义的填充  colsPadding = (int(math.ceil((28 - cols) / 2.0)),  int(math.floor((28 - cols) / 2.0)))  rowsPadding = (int(math.ceil((28 - rows) / 2.0)),  int(math.floor((28 - rows) / 2.0)))  img = np.lib.pad(img, (rowsPadding, colsPadding),  'constant', constant\_values=255)  # 移动图像,使数字的质心,很好地居中  shiftx, shifty = getBestShift(img)  shifted = shift(img, shiftx, shifty)  img = shifted  return img |

### 3、偏移图像模块

|  |
| --- |
| def shift(img, sx, sy):  """  将图像移动一些偏移量  : param img : 黑白图像  : param sx : x 方向位移  : param sy : y 方向位移  : returns : 位移后的图像  """  # 生成仅表示平移（旋转）的变形矩阵,部分是单位矩阵  M = np.float32([[1, 0, sx], [0, 1, sy]])  # 应用变形矩阵  rows, cols = img.shape  shifted = cv2.warpAffine(img, M, (cols, rows),  borderMode=cv2.BORDER\_CONSTANT, borderValue=255)  return shifted |

### 4、得到最佳的移位模块

|  |
| --- |
| def getBestShift(img):  """  : param img : 一个数字的黑白图像  : returns img : 最佳移位 (x, y)  """  # 计算质量中心  cy, cx = ndimage.measurements.center\_of\_mass(img)  # 计算质心与实际的差异,图像中心以获得转变  rows, cols = img.shape  shiftx = np.round(cols / 2.0 - cx).astype(int)  shifty = np.round(rows / 2.0 - cy).astype(int)  return shiftx, shifty |

### 5、接收图像模块

|  |
| --- |
| def data\_uri\_to\_cv2\_img(uri):  """  将数据 URL 转换为 OpenCV 图像  : param uri : 表示 BW 图像的数据 URI  : returns : OpenCV图像  """  encoded\_data = uri.split(',')[1]  nparr = np.fromstring(base64.b64decode(encoded\_data), np.uint8)  img = cv2.imdecode(nparr, cv2.IMREAD\_GRAYSCALE)  return img |

### 6、矩阵转置模块

|  |
| --- |
| def value\_invert(array):  """  接受一个数组，假设包含 0 到 1 之间的值，然后反转那些具有转换 x -> 1 - x 的值。  """  # 将数组展平以进行循环  flatarray = array.flatten()  # 将变换应用于展平数组  for i in range(flatarray.size):  flatarray[i] = 1 - flatarray[i]  # 返回具有原始形状的转换后的数组  return flatarray.reshape(array.shape) |

## 在Python中利用Flask实现API的创建

### 1、导入要使用到的库

|  |
| --- |
| import numpy as np from keras.models import load\_model from flask import Flask, request, render\_template from utils import preprocess, data\_uri\_to\_cv2\_img |

### 2、定义路由

主页

|  |
| --- |
| # 定义根节点的路由项  @app.route('/') def home():  return render\_template('index.html') |

预测

|  |
| --- |
| @app.route('/predict', methods=['POST']) def predict():  # 响应web的请求,从 base64 数据 URL 读取图像数据  imgstring = request.form.get('data')  # 转换为 OpenCV 图像  img = preprocess(data\_uri\_to\_cv2\_img(imgstring))  # cv2.imwrite('static/user\_drawn/temp.png', img)  data = (img / 255).reshape((1, 28, 28, 1))  prediction = model.predict(data)  classes\_x = np.argmax(prediction, axis=1)  predicted\_class = classes\_x  # 返回请求  print("识别为:", predicted\_class)  return f"这个数看起来像: {predicted\_class}" |

### 三、创建并启动服务器

|  |
| --- |
| app = Flask(\_\_name\_\_) model = load\_model("classifier.h5")  if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='127.0.0.1', port=5000, debug=False) |

# 四、项目运行结果

## MNIST模型建立结果

|  |
| --- |
| Epoch 1/10  540/540 [==============================] - 26s 45ms/step - loss: 0.5170 - accuracy: 0.8330 - val\_loss: 0.1559 - val\_accuracy: 0.9565  Epoch 2/10  540/540 [==============================] - 25s 46ms/step - loss: 0.1696 - accuracy: 0.9479 - val\_loss: 0.1080 - val\_accuracy: 0.9703  Epoch 3/10  540/540 [==============================] - 25s 46ms/step - loss: 0.1256 - accuracy: 0.9615 - val\_loss: 0.0867 - val\_accuracy: 0.9745  Epoch 4/10  540/540 [==============================] - 25s 46ms/step - loss: 0.1040 - accuracy: 0.9680 - val\_loss: 0.0700 - val\_accuracy: 0.9778  Epoch 5/10  540/540 [==============================] - 24s 45ms/step - loss: 0.0924 - accuracy: 0.9706 - val\_loss: 0.0659 - val\_accuracy: 0.9803  Epoch 6/10  540/540 [==============================] - 24s 45ms/step - loss: 0.0837 - accuracy: 0.9739 - val\_loss: 0.0637 - val\_accuracy: 0.9813  Epoch 7/10  540/540 [==============================] - 25s 46ms/step - loss: 0.0781 - accuracy: 0.9754 - val\_loss: 0.0623 - val\_accuracy: 0.9805  Epoch 8/10  540/540 [==============================] - 24s 45ms/step - loss: 0.0713 - accuracy: 0.9773 - val\_loss: 0.0637 - val\_accuracy: 0.9807  Epoch 9/10  540/540 [==============================] - 24s 45ms/step - loss: 0.0686 - accuracy: 0.9789 - val\_loss: 0.0543 - val\_accuracy: 0.9848  Epoch 10/10  540/540 [==============================] - 35s 65ms/step - loss: 0.0635 - accuracy: 0.9806 - val\_loss: 0.0647 - val\_accuracy: 0.9803  Large CNN Error: 1.32% |