《机器学习基础理论及其在工程科学中的应用》

课程实践手册（一）

案例 1：人脸图像性别识别

一、问题描述

用人脸图像作为数据集，基于机器学习方法进行性别分类，请按步骤完成以下问题：

（1）下载图片集 http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz，其中包含有 5000 多人的名字和图片，按文件夹进行了分类。

（2）根据.../example1 路径下的 lfw-deepfunneled-gender.txt 文件中提供的人名和性别信息，从下载的图片中任意挑出 4000 张不同男性图片、1200 张女性图片。

（3）用挑出的 3500 张男性图片+1000 张女性图片当作训练集，剩余 500 张男性图片+200 张女性图片当作测试集。为了减小内存消耗，每张图片先截取中心200×200的像素，再缩小成100×100的图片用于学习。

（4）基于 Keras，用这些图片训练出一个能识别图片性别的模型。本例对模型形式和精度没有要求，大家自由发挥。

（5）基于 matplotlib，对预测图片结果进行显示，显示 1 张或几张都可以。

二、学习方法

本例将利用卷积神经网络（CNN）进行分类任务的学习，图 1 给出了一种可能的卷积神经网络模型，利用该模型可解决本例中的分类问题。

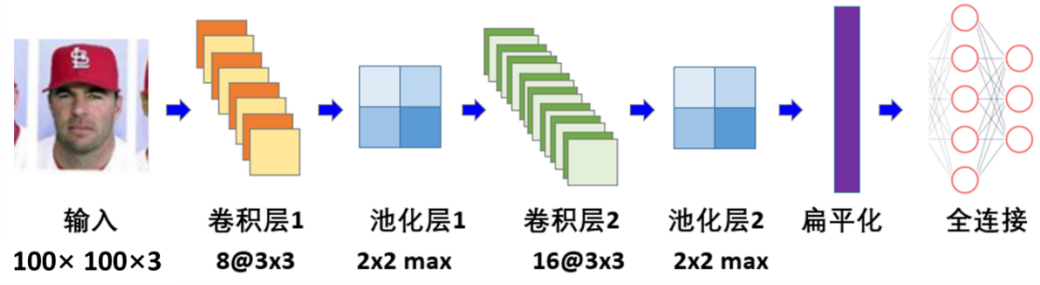


图 1 卷积神经网络模型结构

三、数据准备

图片数据准备：将含有图片数据的 lfw-deepfunneled 文件夹拷贝到…/example1 路径下；

标签数据准备：将含有“人名-性别”信息的 lfw-deepfunneled-gender.txt 文件拷贝到…/example1 路径下。

四、编程过程

打开 python 开发工具（IDE），在…/example1 路径下新建 genderTest.py 文件。实现本例中的分类任务的参考代码如下（[·] 为行号标记）：

（1）导入相关库

```
[1] # 1. 导入相关库
[2] from tensorflow import keras
[3] from tensorflow.keras import layers
[4] from PIL import Image
[5] import numpy as np
[6] import matplotlib.pyplot as plt
[7] import matplotlib.image as mpimg
```

（2）准备数据，用于构建神经网络模型

1. 选取需要的"人名-性别"信息，划分训练集、测试集

- 从"人名-性别"文件中读取全部"人名-性别"信息
- 选取前 4000 名男性和前 1200 名女性，作为需要利用的"人名-性别"信息
- 再选取前 3500 名男性和前 1000 名女性作为训练集，其余作为测试集
- 随机打乱训练集和测试集

2. 读取、处理图片，设置标签（先处理训练集，测试集同理）

- 创建训练集图片数组、训练集标签数组

- 读取"人名-性别"信息，存入标签数组
- 读取、处理图片信息、存入图片数组

3. 进一步处理成机器学习模型需要的数据

- 归一化训练集、测试集图片数组中的颜色值至[0,1]
- 二进制化训练集、测试集标签数组中的类别标签
- 输出数据信息

```
[8] # 2.1 选取需要的"人名-性别"信息，划分训练集、测试集
[9] # 设置图片数目，注意不要超过图片总数
[10]    num_male_use = 4000
[11]    num_female_use = 1200
[12]    num_male_train = 3500
[13]    num_female_train = 1000
[14]    # 从下载的图片中任意挑选4000张不同男性图片，1200张女性图片
[15]    # 其中，3500张男性图片+1000张女性图片 用于训练集training & validation
[16]    num_male_test = num_male_use - num_male_train
[17]    num_female_test = num_female_use - num_female_train
[18]    num_train = num_male_train + num_female_train  # num_train = 3500 + 1000
[19]    num_test = num_male_test + num_female_test  # num_test = 500 + 200
[20]    # 剩余图片，500张男性图片+200张女性图片 用于测试集testing
[21]    # 读入 "人名-性别" 文件信息
[22]    info_file = './lfw-deepfunneled-gender.txt'
[23]    # 注意将lfw-deepfunneled-gender.txt放到genderTest.py所在的文件夹中
[24]    people_info = np.genfromtxt(info_file, dtype='str')
[25]    # 将txt文件中全部数据 转为数据类型为str的数组 people_info(5750, 2)，首行为列标
题(Name, Gender)
[26]    # 挑出4000男性，1200女性
[27]    count_female = 0
[28]    count_male = 0
[29]    people_male_use = []
[30]    people_female_use = []
[31]    for person in people_info:
```

```
[32]        if (person[1] == 'male') and (count_male < num_male_use):
[33]    # num_male_use = 4000
[34]            people_male_use.append(person)
[35]            count_male += 1
[36]        elif (person[1] == 'female') and (count_female < num_female_use):
[37]    # num_female_use = 1200
[38]            people_female_use.append(person)
[39]            count_female += 1
[40]    # 空列表 people_male_use & people_female_use 用于存放 从 people_info 中读取的每一行 (people, gender)
[41]    # 注意，通过这样的循环取出来的，是“人名-性别”文件中，前面 4000 个男性和 1200 个女性，这里从前到后选取，再进行打乱；你也可以用其他方法实现"任意挑选"用于训练和预测的图片
[42]    # 选取 前 3500 男性+1000 女性用于训练
[43]    # num_male_train = 3500, num_female_train = 1000
[44]    # 其余 后 500 男性+200 女性用于测试
[45]    train_people = people_male_use[0:num_male_train] + people_female_use[0:num_female_train]  #  这是同一行代码
[46]    test_people = people_male_use[num_male_train:] + people_female_use[num_female_train:]  #  这是同一行代码
[47]    # 列表 train_people 存放了 4500 个训练集人名-性别信息
[48]    # 列表 test_people 存放了 700 个测试集人名-性别信息
[49]    # 随机打乱，不然前面的都是男性，后面的都是女性
[50]    np.random.shuffle(train_people)
[51]    np.random.shuffle(test_people)
```

执行 2.1 部分代码，得到需要用于训练的 4000 张不同男性图片、1200 张女性图片对应的“人名-性别”信息，并分成了训练集和测试集。

```
[52]    # 2.2 读取、处理图片，设置标签（先处理训练集，测试集同理）
[53]    # 从图片文件夹中，根据挑出的人名-性别信息，处理图片
[54]    # 设置图片信息
[55]    img_width_use = 100
```

```
[56]    img_height_use = 100
[57]    image_dir = './lfw-deepfunneled/'
[58]    # 注意将 lfw-deepfunneled 图片文件夹放到 genderTest.py 所在的文件夹中
[59]    # 读入训练集图片，设置标签
[60]    print('Import image data for training ...')
[61]    # 预分配内存，提高效率
[62]    train_images = np.zeros((num_train, img_width_use, img_height_use, 3))
[63]    # 训练集图片数组 零元数组(4500, 100, 100, 3)
[64]    train_labels = np.zeros(num_train)
[65]    # 训练集标签数组，0 = 男， 1 = 女 ，零元数组(4500,1) 4500 列 1 行
[66]    person_index = 0
[67]    for person in train_people:
[68]        # 人员信息
[69]        name = person[0]  # 注意从这一行开始 for 的缩进
[70]        gender = person[1]
[71]        # 设置标签
[72]        train_labels[person_index] = 0
[73]        if gender == 'female':
[74]            train_labels[person_index] = 1
[75]        # 给训练集图片数组对应的标签数组打标签，0 = 男， 1 = 女
[76]        # 读取图片
[77]        # 图片文件路径
[78]        image_path = image_dir + name + '/' + name + '_0001.jpg'
[79]        # 打开图片
[80]        image = Image.open(image_path)
[81]        # 截取图片局部，已知图片大小都为 250x250，为了简单，这里不判断图片大小和区域
[82]        image = image.crop((25, 25, 225, 225))
[83]        # 缩小图片至 100x100
[84]        image = image.resize((img_width_use, img_height_use), Image.ANTIALIAS)
[85]        # 将数据存储到 numpy 数组中
[86]        train_images[person_index] = np.asarray(image)
```

```
[87]        person_index += 1    # 注意从这一行起 for 的缩进结束
[88]    # 读入测试集图片，设置标签
[89]    print('Import image data for testing ...')
[90]    test_images = np.zeros((num_test, img_width_use, img_height_use, 3))
[91]    # 测试集图片数组 零元数组(700, 100, 100, 3)
[92]    test_labels = np.zeros(num_test)
[93]    # 测试集标签数组，0 = 男， 1 = 女 ，零元数组(700,1) 700 列 1 行
[94]    person_index = 0
[95]    for person in test_people:
[96]        # 人员信息
[97]        name = person[0]  # 注意从这一行开始 for 的缩进
[98]        gender = person[1]
[99]        # 设置标签
[100]       test_labels[person_index] = 0
[101]       if gender == 'female':
[102]           test_labels[person_index] = 1
[103]       # 给测试集图片数组对应的标签数组打标签，0 = 男， 1 = 女
[104]       # 图片文件路径
[105]       image_path = image_dir + name + '/' + name + '_0001.jpg'
[106]       # 打开图片
[107]       image = Image.open(image_path)
[108]       # 截取图片局部，已知图片大小都为 250x250，为了简单，这里不判断图片大小和区域
[109]       image = image.crop((25, 25, 225, 225))
[110]       # 缩小图片至 100x100
[111]       image = image.resize((img_width_use, img_height_use), Image.ANTIALIAS)
[112]       # 将数据存储到 numpy 数组中
[113]       test_images[person_index] = np.asarray(image)
[114]       person_index += 1   # 注意从这一行起 for 的缩进结束
```

执行 2.2 部分代码，得到需要用于训练的 4000 张不同男性图片、1200 张女性图片，并分成了训练集 train_images + train_labels 和测试集 test_images + test_labels。

```
[115]  # 2.3 进一步处理成机器学习模型需要的数据
[116]  # 归一化图片数组颜色值
[117]  # 修改图片数据用于机器学习，图片颜色值是 0 到 255 之间，
[118]  # 若用于神经网络模型，需要将这些值缩小至 0 到 1 之间，所以，将这些值除以 255.0
[119]  train_images = train_images / 255.0
[120]  test_images = test_images / 255.0
[121]  # 二进制化标签数组 类别标签值: train_labels, test_labels
[122]  # 性别只有男/女，所以相当于有 2 个类
[123]  num_classes = 2
[124]  train_labels = keras.utils.to_categorical(train_labels, num_classes)
[125]  test_labels = keras.utils.to_categorical(test_labels, num_classes)
[126]  # 男性标签原本为 0，经过二进制化，变成[1, 0];
[127]  # 女性标签原本为 1，经过二进制化，变成[0, 1]
[128]  # 输出数据信息
[129]  print('\n----------------------\nImage data are prepared.')
[130]  print('Number of training images: ', num_train)
[131]  print('Number of testing images:  ', num_test)
[132]  print('Shape of train_images: ', train_images.shape)
[133]  print('Shape of test_images:  ', test_images.shape)
```

执行 2.3 部分代码，得到需要用于 keras 建模的训练集和测试集数据。其中，训练集图片数组的颜色值值域为[0,1]，类别标签数组为 4500×2 的 0、1 二元数组。

（3）构建神经网络

1. 设置层

- 输入图片的维度、通道数
- 卷积层：卷积核的维度、数量、激活函数
- 池化层：池化方式
- 扁平层、Droupout、全连接层
- 输出模型信息

2. 编译模型

- 损失函数

- 优化算法
- 评价指标

```
[134]  # 3.1 设置层
[135]  model = keras.Sequential(
[136]      [
[137]          keras.Input(shape=(img_width_use, img_height_use, 3)),
[138]  # 每个输入图片的维度、通道数 (100,100,3)
[139]          xxxx,
[140]          # 神经网络各层自行搭建
[141]          xxxx,
[142]      ]
[143]  )
[144]  # 输出模型信息
[145]  model.summary()
[146]  # 3.2 编译模型
[147]  model.compile(
[148]                  xxxx
[149]                  # 神经网络编译部分自行搭建
[150]                  xxxx
[151]                  )
```

执行本段代码将完成神经网络模型的定义，并显示如图 2 所示的模型参数信息（不同的模型结构设置会得到不同的模型参数信息）。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 8)         224
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 16)        1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 8464)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 16930
=================================================================
Total params: 18,322
Trainable params: 18,322
Non-trainable params: 0
```

图 2 模型参数信息

（4）训练模型

1. 向模型送入数据，设置训练超参数

- 训练集 图片数组、标签数组
- 超参数：批大小、轮数、验证集的比例

2. 用测试集评估训练好的模型的准确率

- 输出测试集的准确率信息

3. 保存模型

```
[152]  # 4.1 向模型送入数据，设置训练超参数
[153]  model.fit(train_images,
[154]           train_labels,
[155]           xxxx
[156]           # 超参数部分自行设置
[157]           xxxx
[158]           )
[159]  # 4.2 评估训练好的模型的准确率
[160]  score = model.evaluate(test_images, test_labels, verbose=2)
[161]  # 输出测试集的准确率信息
[162]  print('----------------------\nEvaluating the trained model.')
[163]  print('Test loss:', score[0])
[164]  print('Test accuracy:', score[1])
[165]  # 4.3 保存模型
[166]  model.save('genderModel.h5')
```

执行本段代码将完成神经网络模型的训练，每一步训练后将显示模型在训练集和验证集上的交叉熵损失和准确率，在完成规定的训练步数后，将显示训练耗时。

```
3808/4050 [===========================>..] - ETA: 0s - loss: 0.0491 - accuracy: 0.9858
3840/4050 [===========================>..] - ETA: 0s - loss: 0.0488 - accuracy: 0.9859
3872/4050 [===========================>..] - ETA: 0s - loss: 0.0488 - accuracy: 0.9861
3904/4050 [===========================>..] - ETA: 0s - loss: 0.0487 - accuracy: 0.9859
3936/4050 [============================>.] - ETA: 0s - loss: 0.0490 - accuracy: 0.9855
3968/4050 [============================>.] - ETA: 0s - loss: 0.0491 - accuracy: 0.9856
4000/4050 [============================>.] - ETA: 0s - loss: 0.0490 - accuracy: 0.9858
4032/4050 [============================>.] - ETA: 0s - loss: 0.0493 - accuracy: 0.9856
4050/4050 [==============================] - 7s 2ms/sample - loss: 0.0492 - accuracy: 0.9857 - val_loss: 0.4073 - val_accuracy: 0.8778
Epoch 14/15
```

图 3 每步训练结果

所有训练步完成后，输出在测试集上的模型评估结果。

```
----------------------
Evaluating the trained model.
Test loss: 0.4497767537406513
Test accuracy: 0.88285714
```

图 4 模型测试集评估结果

（5）模型预测

1. 向模型送入测试数据

- 送入部分测试集的图片数组
- 输出预测类别标签数组

2. 显示预测结果

- 显示送入的每张测试图片
- 显示预测为男性或女性的概率

```
[167]  # 5.1 模型预测
[168]  # 现在可以使用训练好的模型来预测图片中人物的性别
[169]  # 预测测试集中的前 5 张图片
[170]  prediction_results = model.predict(test_images[0:5])
[171]  # prediction_results(5,2)，每行是预测的类别标签数组[a, b]，第 1、2 个元素分别代表预测为男性、女性的概率，越接近[1, 0]表示预测为男性的可能性越大
[172]  # 5.2 显示结果
[173]  # 定义预测结果显示函数
[174]  def show_prediction(person, predict_array):
[175]  # 输入：（“人名-性别”信息，预测出来的类别标签数组）
[176]      name = person[0]  # 注意从这一行开始 def 的缩进
[177]      gender = person[1]
[178]      image_dir = './lfw-deepfunneled/'
[179]  # 注意，将 lfw-deepfunneled 图片文件夹放到 genderTest.py 所在的文件夹中
[180]      image_path = image_dir + name + '/' + name + '_0001.jpg'
[181]      image = mpimg.imread(image_path)  # 读取选取的人物图片
[182]      plt.imshow(image)  # 画出 image 中的颜色值数组
[183]      plt.axis('off')  # 关闭坐标轴的值
[184]      ax = plt.gca()  # 获取图片对象
```

```
[185]       # 设置标题 e.g.
[186]       # Jack_Ma (male)
[187]       # 80% male, 20% female
[188]       ax.set_title("{} ({}) {:2.0f}% male, {:2.0f}% female".format(name,gender,
100 * predict_array[0], 100 * predict_array[1]), color='red')  # 注意从这一行起 for 的
缩进结束
[189]   # 选取前 5 张测试集图片，显示预测结果
[190]   figure, axis = plt.subplots()
[191]   for i in range(5):
[192]       plt.cla()  # 动态绘图、清除前面的图
[193]       show_prediction(test_people[i], prediction_results[i])
[194]       plt.pause(5.0)  # 显示 5.0 秒
```

执行本段代码后将分别得到 5 张测试集图片的预测结果，图 5 给出了其中两张人像的预测结果。

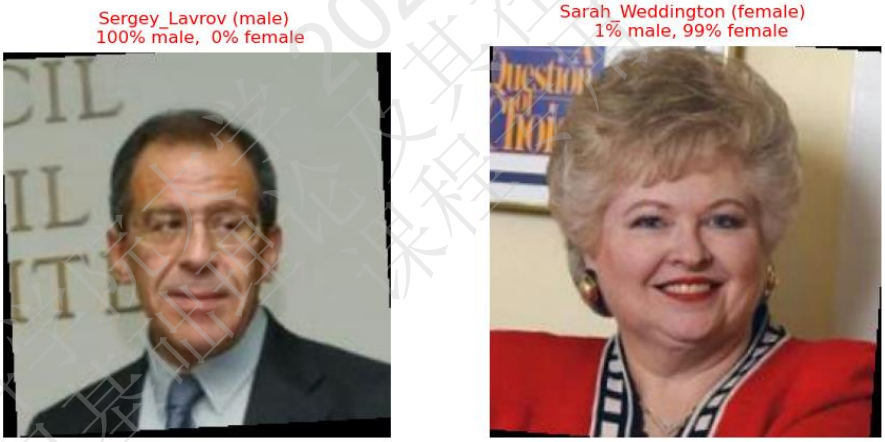


图 5 性别分类预测结果

五、结果

本例的结果详见编程过程中各个步骤的结果。

六、拓展

（1）尝试调整卷积神经网络结构以及超参数，寻找训练效率更高、精度更好的模型。

（2）尝试采用其他方法，从给定的照片集中任意选取待训练的“人名-性别”图片、标签集，研究数据集的随机性对模型训练的影响。

（3）在训练中，偶尔能发现有一些人物照片显示的性别和“人名-性别”文件给出的性别有矛盾，也就是说，初始标签可能有错误。尝试修正这些错误，或添加新的照片数据，增大训练集的规模，看看能否提升训练模型的预测准确率。

案例 2：肺炎影像学诊断

一、问题描述

肺炎是指由细菌、病毒等病原体引起的，在肺泡、远端气道和肺间质的感染性炎症，常伴有咳嗽、咳痰、呼吸困难等症状。由于肺炎的轻微体征和症状通常与感冒或流感相似，且上呼吸道感染容易转化成肺炎，因此，及时发现肺炎对于病情的治疗、病理研究乃至大规模公共卫生事件的预防都具有重要的意义。目前，很多肺炎病情可以通过肺部计算机断层成像（CT）或胸部 X 光影像发现，也可借助这一手段明确肺炎病变位置、严重程度，观察治疗是否有效，监测病情恢复程度等。正常胸部 X 光图像中没有任何异常的不透明区域。细菌性肺炎通常表现出焦点叶巩固，而病毒性肺炎表现为在两个肺中具有更弥漫的“间膜”模式。因此，通过机器学习方法帮助医生快速检查大量 X 光图像，提高影像学检查准确率，减少医生负担，不仅是切实可行的技术手段，更是迫切的医学需求。本例中，我们将利用卷积神经网络(CNN)方法对 5000 多张带标签的肺部影像图片进行学习，建立肺炎影像学诊断智能模型，为新病例的快速筛查和临床诊断提供辅助手段。

二、学习方法

本例介绍的肺炎影响诊断问题是一个典型的图像二分类问题，我们以标准化后的肺炎 X 光图像为输入，以图像的分类标签 $(1,0)$（正常）和$(0,1)$（肺炎）为输出，通过卷积神经网络（CNN）建立肺炎诊断模型，并利用学得的模型对新的病例进行判断，检测是否被感染。

三、数据准备

本例使用的数据集来自 Kaggle（https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia），其中收集的胸部 X 光图像是从广州市妇女儿童医疗中心一至五岁儿科患者的回顾组中挑选的。数据集被分成 3 个文件夹（train、val、test），每个文件夹内的图像被分为肺炎和正常两类并存储于相应的子文件夹中。本例仅使用 train 和 val 两个文件夹中的 X 光图像进行模型训练和测试。其中，train 文件夹内共有 5210 张图像，包含正常图像 1340 个，肺炎图像 3870 个，用作训练集和验证集；val 文件夹内共有 521 张图像，包含正常图像 134 个，肺炎图像 387 个，用作测试集。图 6 给出了训练和验证集中的部分胸部肺炎 X 光图像样本示意图。

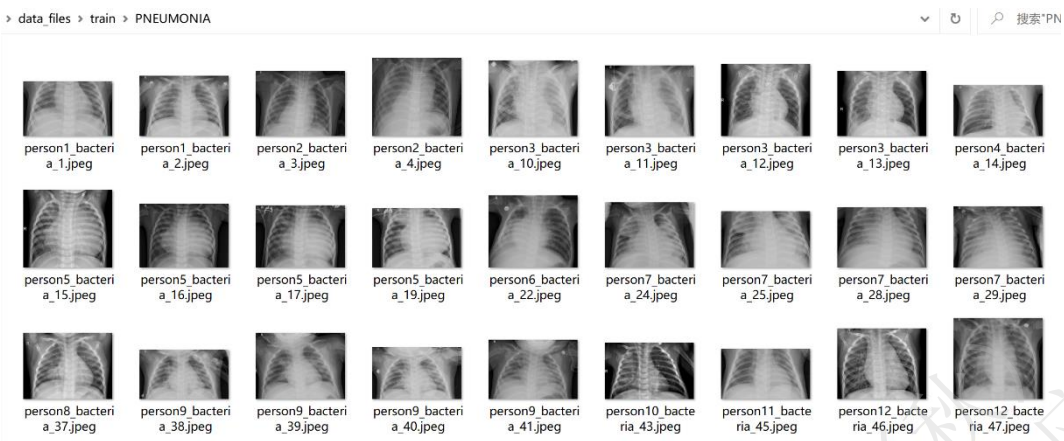


图 6 训练和验证集中的部分胸部肺炎 X 光图像样本

四、编程过程

（1）导入图像处理、卷积神经网络、矩阵运算等相关函数库

```
[1] # 1. 导入函数库
[2] import tensorflow as tf
[3] from tensorflow import keras
[4] from tensorflow.keras import layers
[5] from tensorflow.keras.preprocessing.image import ImageDataGenerator,
    array_to_img, img_to_array
[6] import os
[7] from PIL import Image
[8] import random
[9] import matplotlib.pyplot as plt
[10]  import numpy as np
```

（2）数据读取和预处理

① 将 train 和 val 文件夹中的图像标准化为 300 × 300 的灰度图像，并保存到 datafiles_modify 文件夹中。注意，train 文件夹中的数据用于训练和验证，val 文件夹中的数据用作测试，每个文件夹均包含正常和肺炎两个子文件夹。

```
[11]  # 添加图像数据路径
[12]  trainPath_nomal = 'data_files/train/NORMAL'  # 用于训练和验证
[13]  trainPath_pneumonia = 'data_files/train/PNEUMONIA'
```

```
[14]  valPath_nomal = 'data_files/val/NORMAL'  # 用于测试
[15]  valPath_pneumonia = 'data_files/val/PNEUMONIA'

[16]  # 将图像标准化为 300 × 300 的灰度图像，并保存到 datafiles_modify 文件夹中
[17]  def read_pic(path_file):
[18]      listing_pic = os.listdir(path_file)
[19]      return listing_pic

[20]  def change_pic(path_file):
[21]      listing_pic = os.listdir(path_file)
[22]      for pic in listing_pic:
[23]          if pic != "":
[24]              img = Image.open(path_file + '/' + pic)
[25]              img = img.convert("L")  # 转换成灰度图像
[26]              resizeImg = img.resize((300, 300))  #图片像素标准化
[27]              SavePath = path_file.replace('data_files', 'datafiles_modify')
[28]              if os.path.exists(SavePath) == False:
[29]                  os.makedirs(SavePath)
[30]              resizeImg.save(SavePath + '/' + pic)

[31]  def save_path(path_file):
[32]      SavePath = path_file.replace('data_files', 'datafiles_modify')
[33]      return SavePath

[34]  def b_or_v(picture_list):
[35]      virus = []
[36]      bacteria = []
[37]      for i in picture_list:
[38]          if 'virus' in i:
[39]              virus.append(i)
[40]          elif 'bacteria' in i:
```

```
[41]                bacteria.append(i)
[42]        return [virus, bacteria]

[43]    change_pic(trainPath_nomal)
[44]    change_pic(trainPath_pneumonia)
[45]    change_pic(valPath_nomal)
[46]    change_pic(valPath_pneumonia)
```

② 读取标准化后的图像数据，并将其转换为矩阵形式。

```
[47]    # 读取标准化后的图片数据
[48]    train_nomal = read_pic(save_path(trainPath_nomal))
[49]    train_pneumonia = read_pic(save_path(trainPath_pneumonia))
[50]    val_nomal = read_pic(save_path(valPath_nomal))
[51]    val_pneumonia = read_pic(save_path(valPath_pneumonia))

[52]    # 将诊断为肺炎的图片分为病毒（virus）和细菌（bacteria）感染两类
[53]    [train_virus, train_bacteria] = b_or_v(train_pneumonia)
[54]    [val_virus, val_bacteria] = b_or_v(val_pneumonia)

[55]    # 加载图片并将其转换为矩阵形式
[56]    def load_pic(path_file, piclist):
[57]        list1 = []
[58]        for pic in piclist:
[59]            if pic != "":
[60]                img = Image.open(path_file + '/' + pic)
[61]                x = img_to_array(img)
[62]                list1.append(x)
[63]        return list1

[64]    pic_train_nomal = (load_pic(save_path(trainPath_nomal), train_nomal))
[65]    pic_train_virus = (load_pic(save_path(trainPath_pneumonia), train_virus))
```

```
[66]    pic_train_bacteria = (load_pic(save_path(trainPath_pneumonia),
    train_bacteria))
[67]    pic_val_nomal = (load_pic(save_path(valPath_nomal), val_nomal))
[68]    pic_val_virus = (load_pic(save_path(valPath_pneumonia), val_virus))
[69]    pic_val_bacteria = (load_pic(save_path(valPath_pneumonia), val_bacteria))
[70]    print("==>>==>>==>>==>>数据加载完成")
```

③ 定义训练和验证集、测试集上的输入和输出，并对输入数据进行随机化、归一化处理，对输出分类标签进行二进制化处理。

```
[71]    # 定义训练和验证集上的输入和输出
[72]    X_train = pic_train_nomal + pic_train_virus + pic_train_bacteria
[73]    Y_train = [0] * len(pic_train_nomal) + [1] * len(pic_train_virus) + [1] *
    len(pic_train_bacteria) # 定义输出标签：正常：0，肺炎（病毒或细菌感染）：1
[74]    c = list(zip(X_train, Y_train))
[75]    random.shuffle(c) # 数据排列随机化
[76]    X_train, Y_train = zip(*c)

[77]    # 定义测试集上的输入和输出
[78]    X_val = pic_val_nomal + pic_val_virus + pic_val_bacteria
[79]    Y_val = [0] * len(pic_val_nomal) + [1] * len(pic_val_virus) + [1] *
    len(pic_val_bacteria)
[80]    c2 = list(zip(X_val, Y_val))
[81]    random.shuffle(c2)
[82]    X_val, Y_val = zip(*c2)
[83]    # 复制测试集的输出标签
[84]    Y_val1 = Y_val

[85]    # 计算并显示训练和验证集样本数量，计算测试集样本数量
[86]    num_classes = 2
[87]    train_num = len(pic_train_nomal) + len(pic_train_virus) +
    len(pic_train_bacteria)
```

```
[88]   val_num = len(pic_val_nomal) + len(pic_val_virus) + len(pic_val_bacteria)
[89]   total_input = train_num
[90]   print("Total Train Data : %d" % total_input)

[91]   # 输入数据 3D 化
[92]   X_val = np.array(X_val)
[93]   X_val = np.reshape(X_val, (val_num, 300, 300))
[94]   X_val = X_val.astype('float32')
[95]   X_train = np.array(X_train)
[96]   X_train = np.reshape(X_train, (train_num, 300, 300))
[97]   X_train = X_train.astype('float32')
[98]   # 输入数据矩阵归一化
[99]   X_train /= 255
[100]  X_val /= 255

[101]  # 显示训练和验证集中的图片
[102]  show_single_image = True
[103]  if show_single_image:
[104]      image_index = 0 #图片编号
[105]      plt.figure()
[106]      plt.imshow(X_train[image_index])
[107]      plt.colorbar()
[108]      plt.grid(False)
[109]      plt.show()

[110]  # 输入数据扩维
[111]  X_train = np.expand_dims(X_train, -1)
[112]  X_val = np.expand_dims(X_val, -1)

[113]  # 将输出转化为二进制格式
[114]  Y_train = np.array(Y_train)
```

```
[115]  Y_train = keras.utils.to_categorical(Y_train, num_classes)
[116]  Y_val = np.array(Y_val)
[117]  Y_val = keras.utils.to_categorical(Y_val, num_classes)
[118]  # 显示训练和验证集、测试集上的图片数量
[119]  print('Number of training images: ', total_input)
[120]  print('Number of validation images: ', val_num)
[121]  # 显示输入数据维数
[122]  print('Shape of train_images: ', X_train.shape)
[123]  print('Shape of val_images:  ', X_val.shape)
```

（3）构建卷积神经网络模型

```
[124]  # 3. 构建神经网络
[125]  # 设置卷积层、池化层、全链接层
[126]  model = keras.Sequential(
[127]      [
[128]          keras.Input(shape=(300, 300, 1)), # 输入图片的维度300×300，通道数为1
[129]          layers.Conv2D(8, kernel_size=(3, 3), activation="relu"),
[130]          layers.MaxPooling2D(pool_size=(2, 2)),
[131]          layers.Dropout(0.25), # 防止过拟合
[132]          layers.Conv2D(16, kernel_size=(3, 3), activation="relu"),
[133]          layers.MaxPooling2D(pool_size=(2, 2)),
[134]          layers.Dropout(0.25),
[135]          layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
[136]          layers.Flatten(),# 将上一层数据压缩成一维数据
[137]          layers.Dropout(0.25),
[138]          layers.Dense(num_classes, activation="softmax"),
[139]      ]
[140]  )
[141]  # 输出模型信息
[142]  model.summary()
```

【结果】运行本部分代码后将得到如图 7 所示的模型参数信息。

```
In [33]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 298, 298, 8)       80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 149, 149, 8)       0
_________________________________________________________________
dropout (Dropout)            (None, 149, 149, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 147, 147, 16)      1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 73, 73, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 73, 73, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 32)        4640
_________________________________________________________________
flatten (Flatten)            (None, 161312)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 161312)            0
_________________________________________________________________
dense (Dense)                (None, 2)                 322626
=================================================================
Total params: 328,514
Trainable params: 328,514
Non-trainable params: 0
_________________________________________________________________
```

图 7 模型参数信息

（4）模型训练

① 模型编译

```
[143]  # 编译模型
[144]  model.compile(loss="categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"]) # 使用多分类交叉熵作为损失函数
```

② 模型训练

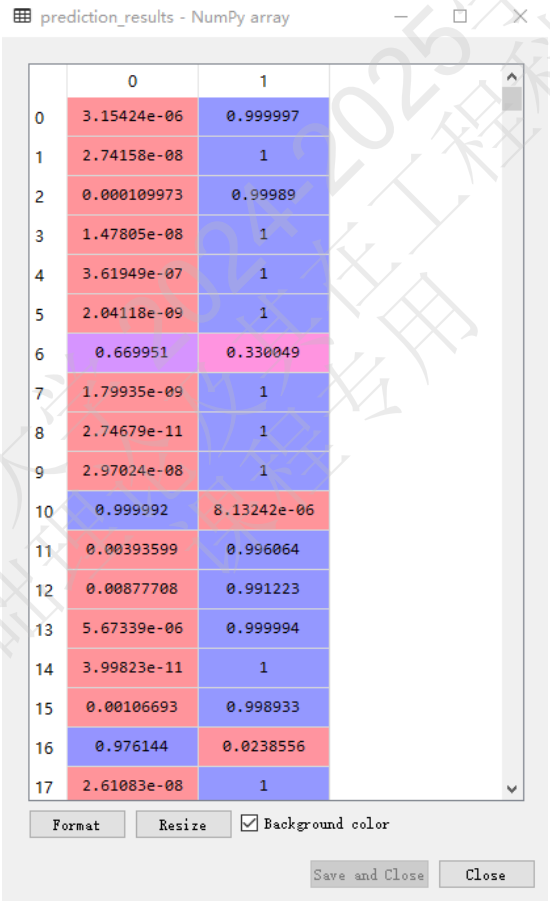
```
[145]  #模型训练
[146]  # 设置 checkpoint
[147]  from keras.callbacks import ModelCheckpoint
[148]  filepath = 'chest_xray_model.h5'
[149]  checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=2,
    save_best_only=True, mode='min') # 保存在验证集上损失函数最小的模型
[150]  callbacks_list = [checkpoint]

[151]  history = model.fit(X_train, Y_train, validation_split=0.1, batch_size=128,
    epochs=20, callbacks=callbacks_list) # 将 10%的数据（即 521 个样本）作为验证集
```

（5）模型预测

① 以测试集输入数据为输入，利用训练好的模型预测输出，即正常和肺炎的概率。

```
[152]  # 预测测试集输出（即正常和肺炎的概率）
[153]  prediction_results = model.predict(X_val)
[154]  # 输出分类值，正常为 0，肺炎为 1
[155]  predicted_index = np.argmax(prediction_results, axis=1)
```

【结果】运行本部分代码后将得到如图 8 所示的预测结果（prediction_results），它的每一行对应测试集上的一张胸部 X 光图像，第 1 列表示正常的概率，第 2 列表示病人患有肺炎的概率。



	0	1
0	3.15424e-06	0.999997
1	2.74158e-08	1
2	0.000109973	0.99989
3	1.47805e-08	1
4	3.61949e-07	1
5	2.04118e-09	1
6	0.669951	0.330049
7	1.79935e-09	1
8	2.74679e-11	1
9	2.97024e-08	1
10	0.999992	8.13242e-06
11	0.00393599	0.996064
12	0.00877708	0.991223
13	5.67339e-06	0.999994
14	3.99823e-11	1
15	0.00106693	0.998933
16	0.976144	0.0238556
17	2.61083e-08	1

图 8 训练模型在测试集上的预测结果

② 评估训练模型在测试集上的准确率

```
[156]  score = model.evaluate(X_val, Y_val, verbose=2)
[157]  # 输出训练模型在测试集上的准确率信息
[158]  print('----------------------\nEvaluating the trained model.')
[159]  print('Test accuracy:', score[1])
```

【结果】运行本部分代码后将显示如图 9 所示的训练模型在测试集上的预测准确率信息。

```
----------------------
Evaluating the trained model.
Test accuracy: 0.85988486
```

图 9 训练模型在测试集上的预测准确率信息

五、结果

本例的结果详见编程过程中相关步骤的结果。

六、拓展

（1）尝试通过调整卷积神经网络结构、超参数等，寻找训练效率更高、精度更好的模型；

（2）重点关注测试集上预测错误的样本及对应的概率结果，探究提高诊断准确率的方法；

（3）尝试更精细化的肺炎诊断模型，比如使模型能够判断引发肺炎的原因（病毒感染或细菌感染）。

案例 3：二维翼型气动力预测

一、问题描述

二维翼型是机翼上垂直于机翼前缘的二维剖面形状，是影响机翼气动性能的关键因素。翼型在不同飞行工况下的气动力通常通过风洞试验或计算流体力学（CFD）方法来确定。试验和计算虽然是获得翼型气动力的有效方法，但当需要考虑的工况很多时面临试验成本高和计算量大的问题。随着机器学习方法的发展，以神经网络模型为基础，利用有限试验或计算数据建立翼型气动力预测模型的方法逐渐进入人们的视野。本例中，我们将以来流速度 V_∞ 和飞行攻角 α 为输入，以翼型升力 L、阻力 D 和绕距前缘 1/4 弦长点的力矩 M（以下简称力矩）为输出（如图 10 所示），利用有限的 CFD 计算数据来建立输入与输出之间的映射关系，从而获得二维翼型气动力预测模型。

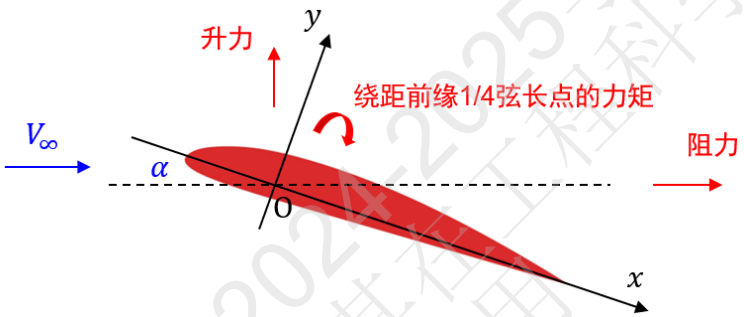


图 10 飞行工况（来流速度和攻角）与翼型气动力示意图

二、学习方法

本例中，我们使用多层前馈神经网络来建立输入与输出之间的映射关系。其中，输入为来流速度和飞行攻角，对应输入层的维数为 2；输出为翼型升力、阻力和力矩，对应输出层的维数为 3；隐藏层的数量为 1，神经元个数为 5。我们通过 CFD 方法计算不同来流速度和飞行攻角下的翼型气动力值，以构成样本数据。其中，样本数据分为训练集、验证集和测试集，仅训练集中的样本用于模型参数的更新，即基于模型在训练集上的误差，利用误差反向传播（BP）算法来更新神经元的权重和偏置值。当经过足够多步的训练后，模型在训练集和验证集上的误差趋于收敛，我们将此时的模型作为最终学得的预测模型，并用它来代替 CFD 求解器，以测试集上样本的来流速度和攻角为输入，预测二维翼型对应的升力、阻力和力矩。

三、数据准备

（1）选取低速翼型 NACA 3412 为研究对象，利用 Pointwise 18.0 软件建立流场网格模型，如图 11 所示。网格文件 naca3412.pw 存放在…/example2 路径下。

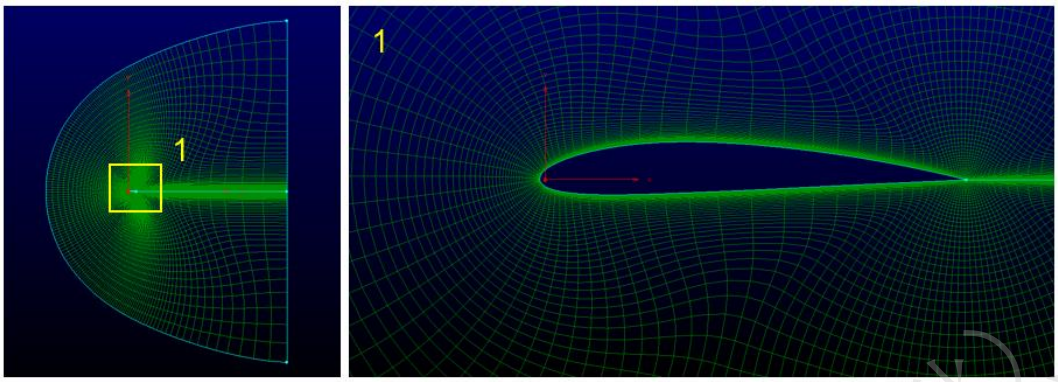


图 11 NACA 3412 翼型流场网格

（2）将网格文件导入 ANSYS 19.0 软件中的 Fluent 模块，建立 CFD 分析模型，流场边界条件如图 12 所示。采用不可压流计算，湍流模型选用 SST $k-\omega$ 模型。来流速度的变化范围取为 $50\ \mathrm{m/s} < V_\infty < 100\ \mathrm{m/s}$，飞行攻角的变化范围取为 $0° < \alpha < 10°$。图 13 给出了 $V_\infty = 50\ \mathrm{m/s}$，$\alpha = 10°$ 条件下翼型周围的静压、 x 向速度和 y 向速度云图

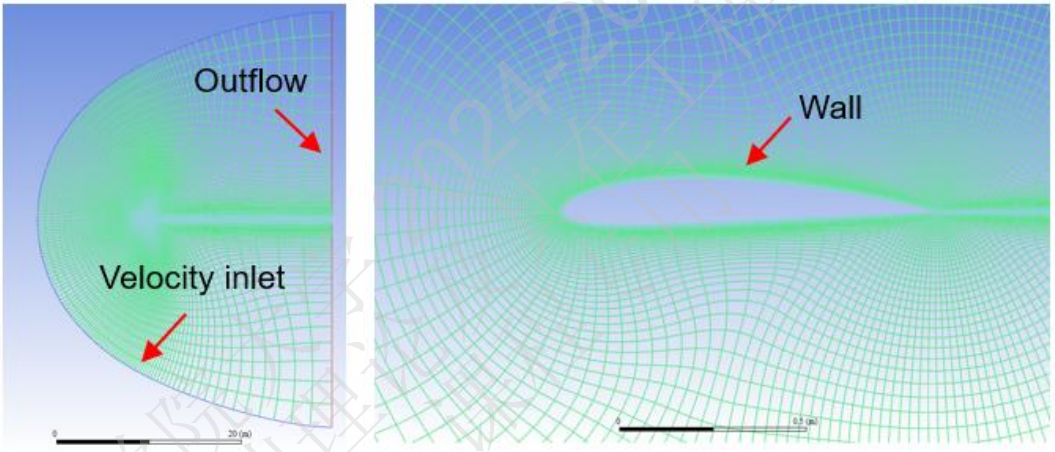
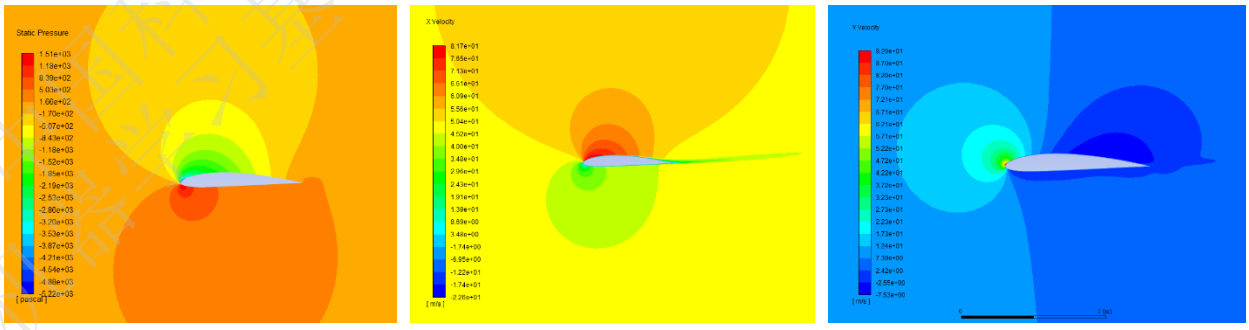


图 12 NACA 3412 翼型 CFD 模型边界条件示意图

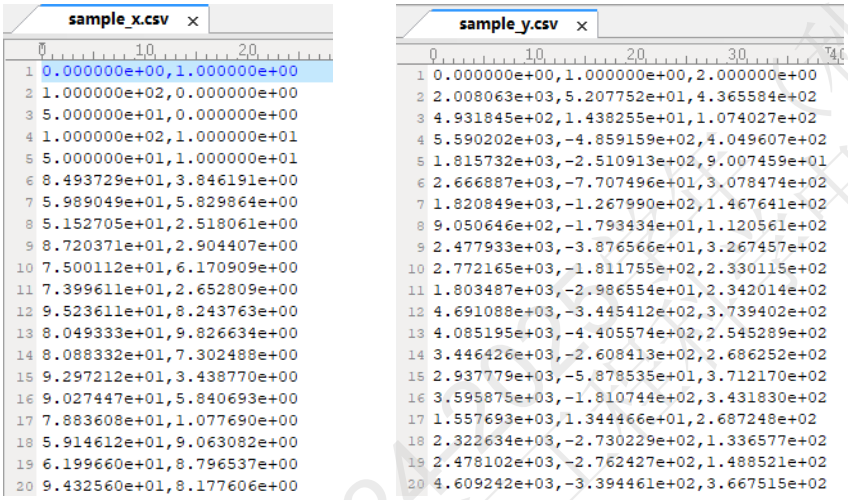


(a) (b) (c)

图 13 $V_\infty = 50\ \mathrm{m/s}$，$\alpha = 10°$ 时翼型的（a）静压、（b）x 向速度和（c）y 向速度云图

（3）在上述来流速度和攻角的取值范围内随机生成 100 个飞行工况，利用建立的 CFD 分析模型分别计算每一个工况下翼型的升力、阻力和力矩，从而构建样本输入数据文件

sample_x.csv 和样本输出数据文件 sample_y.csv。其中，sample_x.csv 的第 1 行为列标签，第 2~101 行的每一行对应一个样本，第 1 和 2 列分别为对应样本的来流速度和攻角（如图 14（a）所示，仅截取前 20 行）；sample_y.csv 的第 1 行为列标签，第 2~101 行的每一行对应一个样本，第 1 至 3 列分别为与来流速度和攻角对应的升力、阻力和力矩（如图 14（b）所示，仅截取前 20 行）。将上述样本输入和输出数据文件存放在…/example2 路径下供后续模型训练使用。



(a) (b)

图 14 （a）样本输入数据文件和（b）样本输出数据文件

四、编程过程

（1）打开 python 开发工具（IDE），在…/example2 路径下新建 airfoil.py 文件。

（2）读取样本数据

读取输入样本数据 sample_x.csv 和输出样本数据 sample_y.csv。参考代码如下（[·] 为行号标记）：

```
[160]  # 1. 读取样本数据
[161]  import pandas as pd
[162]  sample_x1 = pd.read_csv('sample_x.csv',error_bad_lines=False)  # 读取样本输入
100*2
[163]  sample_y1 = pd.read_csv('sample_y.csv',error_bad_lines=False)  # 读取样本输出
100*3
[164]  sample_x = sample_x1.values
[165]  sample_y = sample_y1.values
```

执行本段代码后将分别得到输入矩阵 sample_x 和输出矩阵 sample_y。其中，sample_x 是

100×2 的矩阵，每一行代表一个样本，第 1 和 2 列分别为来流速度 V_∞ 和攻角 α 的取值；sample_y 是 100×3 的矩阵，每一行代表一个样本，第 1 至 3 列分别为与输入来流速度和攻角对应的 CFD 计算得到的升力、阻力和力矩。

（3）训练集、验证集、测试集划分

将输入和输出样本数据划分为训练集、验证集和测试集。前 80 个样本用于训练和验证（其中，60 个样本用于训练，20 个样本用于验证，训练集和验证集的划分将在步骤（7）中进行），后 20 个样本用于验证。参考代码如下：

```
[166]  # 2. 训练集、验证集、测试集划分
[167]  n_tr_val = 80 #用于训练和验证的样本数量
[168]  train_x = sample_x[0:n_tr_val,:] # 训练集和验证集输入
[169]  train_y = sample_y[0:n_tr_val,:] # 训练集和验证集输出
[170]  test_x = sample_x[n_tr_val:,:] # 测试集输入
[171]  test_y = sample_y[n_tr_val:,:] # 测试集输出
[172]  feature = 3 # 输出特征数量
```

执行本段代码后将分别得到训练集和验证集输入矩阵 train_x（80×2），训练集和验证集输出矩阵 train_y（80×3），测试集输入矩阵 test_x（20×2），测试集输出矩阵 test_y（20×3）。

（4）输入和输出数据的标准化处理

将输入矩阵 sample_x 中的每一个数据按列线性标准化至$[-1,1]$，将输出矩阵 sample_y 中的每一个数据按列线性标准化至$[0,1]$，并将标准化后的输入和输出划分为训练和验证集、测试集。注意：本例中选用 sigmoid 函数作为隐藏层的激活函数，其定义域为 $(-\infty,+\infty)$，值域为 $[0,1]$，当输入的范围处于$[-1,1]$时，sigmoid 函数的输出值变化较为明显，因此将输入数据标准化至$[-1,1]$。同时，选用 relu 函数作为输出层的激活函数，其值域为 $[0,1]$，因此需将输出数据标准化至$[0,1]$。相关激活函数的定义见步骤（5）。参考代码如下：

```
[173]  # 3. 输入和输出数据的标准化处理
[174]  from sklearn.preprocessing import MinMaxScaler
[175]  scalerX=MinMaxScaler(feature_range=(-1,1))
[176]  sample_x_scaler = scalerX.fit_transform(sample_x)
[177]  scalerY=MinMaxScaler(feature_range=(0,1))
[178]  sample_y_scaler = scalerY.fit_transform(sample_y)
```

```
[179]  train_x_scaler = sample_x_scaler[0:n_tr_val,:]
[180]  train_y_scaler = sample_y_scaler[0:n_tr_val,:]
[181]  test_x_scaler = sample_x_scaler[n_tr_val:,:]
[182]  test_y_scaler = sample_y_scaler[n_tr_val:,:]
```

执行本段代码后将分别得到训练集和验证集标准化输入矩阵 train_x_scaler（80×2），训练集和验证集标准化输出矩阵 train_y_scaler（80×3），测试集标准化输入矩阵 test_x_scaler（20×2），测试集标准化输出矩阵 test_y_scaler（20×3）。

（5）神经网络模型定义

定义神经网络模型，采用含有一个隐藏层的多层感知机网络来构造学习模型。其中，隐藏层的神经元数目为 5，输入层的维数为 2，输出层的维数为 3。隐藏层和输出层中的神经元分别使用 sigmoid 函数和 relu 函数作为激活函数。每个神经元的权重采用 normal 准则进行初始化，即初始化为一组满足均值为 0，标准差为 0.05 的高斯分布的随机数。参考代码如下：

```
[183]  # 4. 神经网络模型定义
[184]  from keras.models import Sequential
[185]  from keras.layers import Dense
[186]  model = Sequential()
[187]  model.add(Dense(units=5,input_dim=2,kernel_initializer='normal',activation='
   sigmoid'))
[188]  model.add(Dense(units=feature,kernel_initializer='normal',activation='relu')
   )
[189]  model.summary() # 输出模型参数信息
```

执行本段代码将完成神经网络模型的定义，并显示如图 15 所示的模型参数信息。其中，输入层与隐藏层之间有 10 个权重，隐藏层神经元本身有 5 个偏置；隐藏层与输出层之间有 15 个权重，输出层神经元本身有 3 个偏置，共计 33 个参数。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 5)                 15
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 18
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

图 15 模型参数信息

（6）模型编译

定义学习率为 0.005，将模型预测值与真实值之间的均方误差（MSE）作为损失函数，选用 adam 优化算法进行模型参数更新，对建立的神经网络模型进行编译。参考代码如下：

```
[190]  # 5. 模型编译
[191]  from keras import optimizers
[192]  adam=optimizers.Adam(lr=0.005, beta_1=0.9, beta_2=0.999, epsilon=1e-7,
    decay=0, amsgrad=False)
[193]  model.compile(loss='MSE',optimizer='adam') # 选择误差评价准则、参数优化方法
```

执行本段代码将完成神经网络模型的编译。

（7）模型训练

分别将训练集和验证集上的标准化输入矩阵 train_x_scaler（80×2）、标准化输出矩阵 train_y_scaler（80×3）作为模型训练的输入和输出。其中，前 75%的样本（即 60 个）用作训练，其余 25%的样本（即 20 个）用作验证。设置批处理的大小 batch size=8，训练步数 epochs=50000，并记录模型完成训练所需的时间，以及模型在训练集和验证集上的均方误差。参考代码如下：

```
[194]  # 6. 模型训练
[195]  import time
[196]  time_start = time.time()
[197]  history = model.fit(train_x_scaler, train_y_scaler, epochs = 50000,
    batch_size = 8, validation_split = 0.25, verbose = 2, shuffle = False)
[198]  time_end=time.time()
[199]  print('totally cost',time_end-time_start)
[200]  loss_y=history.history['loss'] # 模型在训练集上的均方误差历程
[201]  val_loss_y=history.history['val_loss'] # 模型在验证集上的均方误差历程
```

执行本段代码将完成神经网络模型的训练，每一步训练后将显示模型在训练集和验证集上的均方误差，在完成规定的训练步数后，将显示训练耗时。

（8）模型预测

以训练集和验证集上的标准化输入矩阵 train_x_scaler（80×2）、测试集上的标准化输入矩阵 test_x_scaler（20×2）为输入，利用训练得到的模型分别预测训练集和验证集、测试集上的

标准化预测输出矩阵，并通过线性标准化的反变换过程获得训练集和验证集的预测输出矩阵 train_y_pre（80×3），以及测试集上的预测输出矩阵 test_y_pre（20×3）。预测输出矩阵的每一行代表一个样本，第 1 至 3 列分别为每个样本对应的升力、阻力和力矩的预测值。参考代码如下：

```
[202]  # 7. 模型预测
[203]  train_y_scaler_pre = model.predict(train_x_scaler) # 模型在训练和验证集上的标准化预测值
[204]  train_y_pre = scalerY.inverse_transform(train_y_scaler_pre) # 模型在训练和验证集上的预测值
[205]  test_y_scaler_pre = model.predict(test_x_scaler) # 模型在测试集上的标准化预测值
[206]  test_y_pre = scalerY.inverse_transform(test_y_scaler_pre) # 模型在测试集上的预测值
```

执行本段代码后将分别得到模型在训练集和验证集上的预测输出矩阵 train_y_pre（80×3），以及在测试集上的预测输出矩阵 test_y_pre（20×3）。

（9）气动力预测值和模型训练误差历程输出

输出模型在训练集和验证集、测试集上的输出（即气动力）预测值，以及模型在训练集和验证集上的均方误差随训练步数的变化历程。参考代码如下：

```
[207]  # 8. 气动力预测值和模型训练误差历程输出
[208]  name1=['lift','drag','moment']
[209]  ex1=pd.DataFrame(columns=name1,data=train_y_pre)
[210]  ex1.to_csv('…/example2/train_y_pre.csv') # 输出模型在训练集和验证集上的预测值
[211]  name2=['lift','drag','moment']
[212]  ex2=pd.DataFrame(columns=name2,data=test_y_pre)
[213]  ex2.to_csv('…/example2/test_y_pre.csv') # 输出模型在测试集上的预测值
[214]  name3=['loss']
[215]  ex3=pd.DataFrame(columns=name3,data=loss_y)
[216]  ex3.to_csv('…/example2/loss.csv') # 输出模型在训练集上的均方误差历程
[217]  name4=['val_loss']
[218]  ex4=pd.DataFrame(columns=name4,data=val_loss_y)
```

```
[219]  ex4.to_csv('…/example2/val_loss.csv') # 输出模型在验证集上的均方误差历程
```

执行本段代码后将分别得到模型在训练集和验证集上的预测值 train_y_pre.csv，模型在测试集上的预测值 test_y_pre.csv，以及模型训练过程中训练集和验证集上的预测均方误差历程 loss.csv 和 val_loss.csv。上述文件为 csv 格式，存储在…/example2 路径下。

五、结果

执行二维翼型气动力预测程序 airfoil.py 后，结果文件将被输出到…/example2 路径下，包括（1）模型在训练集和验证集上的预测值 train_y_pre.csv，如图 16（a）所示（仅截取前 25 行），第 1 行为列标签，第 2~81 行的每一行对应一个样本，第 1 列为样本序号，第 2~4 列分别为升力、阻力、力矩的预测值；（2）模型在测试集上的预测值 test_y_pre.csv，如图 16（b）所示，第 1 行为列标签，第 2~21 行的每一行对应一个样本，第 1 列为样本序号，第 2~4 列分别为升力、阻力、力矩的预测值；（3）模型训练过程中训练集和验证集上的预测均方误差历程 loss.csv（如图 17（a）所示，仅截取前 20 行）和 val_loss.csv（如图 17（b）所示，仅截取前 20 行），文件 loss.csv 和 val_loss.csv 的第 1 行为列标签，第 2~50001 行的每一行代表一个训练步，第 1 列为训练步序号，前者的第 2 列为完成对应训练步后模型在训练集上的预测均方误差，后者的第 2 列为完成对应训练步后模型在验证集上的预测均方误差。

train_y_pre.csv

```
,lift,drag,moment
0,2427.2107,38.92672,405.94492
1,655.3237,37.57539,129.16397
2,5275.4717,-430.90256,400.77048
3,2312.5574,-291.3731,101.224785
4,2578.8992,-83.93901,309.9767
5,1699.9006,-131.37196,134.58305
6,885.7781,-13.546602,115.19874
7,2447.314,-44.043003,328.9728
8,2646.8625,-182.92903,228.55005
9,1653.6543,-16.710035,236.451
10,4665.1987,-351.77194,378.11038
11,4200.3994,-394.85757,267.24866
12,3433.5002,-264.01483,271.7608
13,2984.8914,-83.85191,368.85187
14,3602.7578,-207.78275,347.09274
15,1584.8579,33.80249,279.66815
16,2550.3823,-283.1367,134.70126
17,2641.4863,-280.62012,148.05305
18,4595.7783,-346.7176,371.91098
19,893.6446,-15.85728,114.20367
20,2541.5774,-169.2566,225.27306
21,825.71185,42.442215,164.61694
22,3653.6428,-145.29593,409.1833
23,2412.6218,-51.005947,317.1229
```

(a)

test_y_pre.csv

```
,lift,drag,moment
0,2815.393,-123.36867,307.83267
1,3739.177,-293.79956,290.20023
2,1162.8104,-32.268448,142.29703
3,2046.2834,-214.73222,117.75552
4,3873.516,-173.93869,416.0823
5,838.80676,40.055027,164.19698
6,811.66455,4.7299304,121.28964
7,3204.2295,-250.52232,250.25037
8,3491.1604,-156.42064,376.42654
9,1891.0673,16.517342,307.8686
10,1210.8441,-38.3501,144.07248
11,2205.6772,-160.98366,183.34117
12,1474.9176,8.888079,235.21524
13,4646.625,-316.92703,407.30502
14,1023.13635,-8.911812,142.89543
15,4764.465,-390.57812,357.01068
16,2093.3513,-26.192572,294.22037
17,2714.2173,-245.10373,185.20502
18,1005.1442,6.28742,155.72643
19,1577.921,-23.477274,217.57858
```

(b)

图 16 模型在（a）训练集和验证集、（b）测试集上的输出预测文件

loss.csv

```
,loss
0,0.26287792523701986
1,0.2398307204246521
2,0.2185190002123515
3,0.19903370539347331
4,0.181377050280571
5,0.16547893087069193
6,0.1512859026590983
7,0.13867374410231909
8,0.12756568143765132
9,0.11782181163628896
10,0.10932619522015254
11,0.10195176402727762
12,0.09557947317759195
13,0.09009292324384054
14,0.08537999043862025
15,0.0813532218337059
16,0.07792743941148123
17,0.07500880807638169
18,0.07252117097377778
```

val_loss.csv

```
,val_loss
0,0.24581972658634185
1,0.22335796058177948
2,0.20267406702041627
3,0.1838408887386322
4,0.16680842638015747
5,0.15148967504501343
6,0.13780274093151093
7,0.1256658434867859
8,0.11494720727205276
9,0.1055181235074997
10,0.09727129340171814
11,0.09007394909858704
12,0.08381606638431549
13,0.07837098836898804
14,0.0736580416560173
15,0.06958921998739243
16,0.06607508212327957
17,0.06303155124187469
18,0.060391998291015624
```

图 17 模型在（a）训练集和（b）验证集上的预测均方误差历程文件

将训练模型预测的气动力值与 CFD 计算值进行比较，可得到模型在每一个样本上的预测值的相对误差情况。以升力为例，图 18 给出了模型在训练集、验证集和测试集上的升力预测值的相对误差分布情况。此外，将模型在训练集和验证集上预测均方误差历程绘制成误差随训练步数的变化曲线，可得到如图 19 所示的预测均方误差收敛曲线。

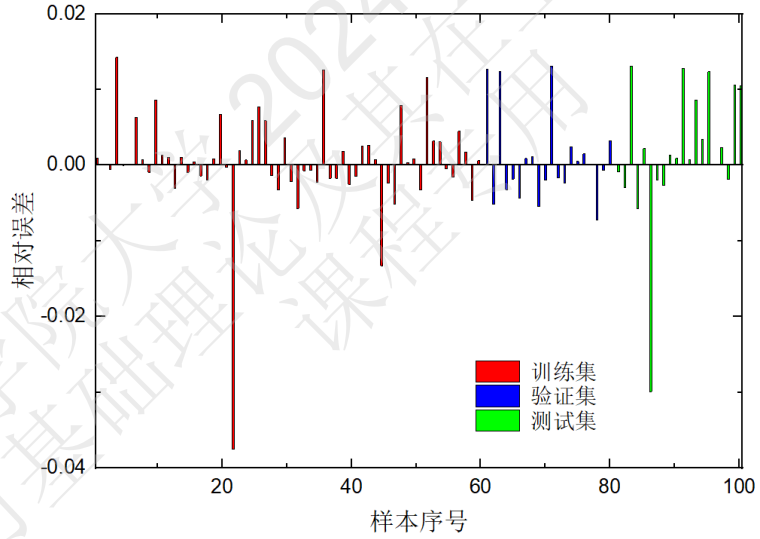


图 18 模型在训练集、验证集和测试集上的升力预测值的相对误差情况

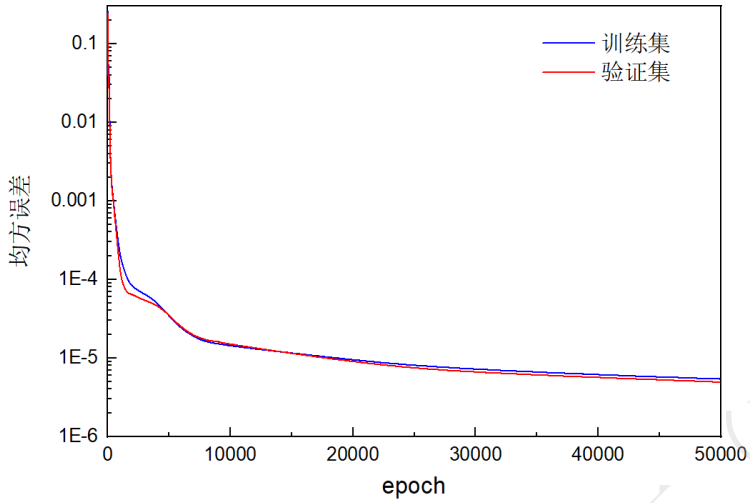


图 19 模型在训练集和验证集上的预测均方误差历程曲线

六、拓展

（1）尝试通过调整神经元数目、学习率、批处理大小等模型参数，寻找训练效率更高、精度更好的翼型气动力预测模型；

（2）尝试通过增加样本数量或优化神经网络结构等方法，改善学习模型对于阻力的预测能力。

案例 3：二维圆柱绕流场预测

一、问题描述

卡门涡街是流体力学中重要的现象，在自然界中经常会遇到，在一定条件下的定常来流绕过某些物体时，物体两侧会周期性地脱落出旋转方向相反、排列规则的双列线涡，经过非线性作用后，形成卡门涡街。如水流过桥墩，风吹过高塔、烟囱、电线等都会形成卡门涡街。二维圆柱绕流问题是研究卡门涡街现象的一个经典案例，二维圆柱低速定常绕流的流型仅与雷诺数 $Re = \rho v d/\mu$ 有关。其中，ρ 为来流的密度，v 为来流的速度，d 为特征长度，即圆柱的直径，μ 为流体的粘性系数。本例中，我们将以如图 10 所示的二维圆柱绕流问题为研究对象，在计算稳定并产生周期性的卡门涡街时，利用机器学习中的循环神经网络方法建立前若干时间步流场与后若干时间步流场之间的映射关系，从而获得二维圆柱稳定绕流流场的预测模型。

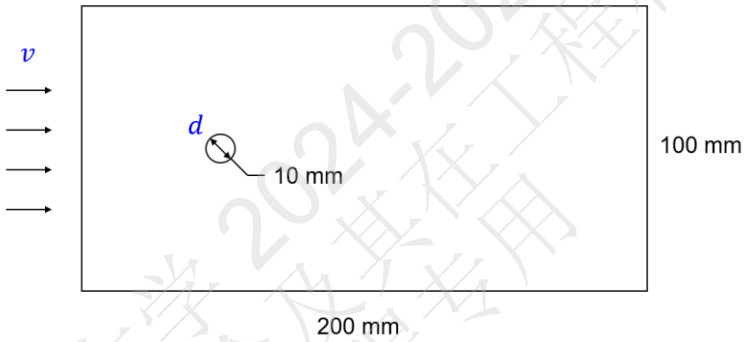


图 20 二维圆柱绕流问题的物理模型

二、学习方法

二维圆柱流场绕流问题是一个典型的时间序列问题，同时流场预测和重构涉及每一个网格节点的物理量信息，属于高维问题，需要通过降维方法减少变量的数量。本例中，我们使用主成分分析方法（PCA）来对流场进行降维，使用循环神经网络（RNN）的一种变体——门控循环单元（GRU）网络来建立降维后的输入与输出之间的时间依赖关系。基本思路如下：（1）调试雷诺数，使非定常计算稳定，产生周期性卡门涡街；（2）连续输出若干步的流场信息（如压强场、速度场等），需要包含多个周期；（3）利用主 PCA 方法将流场从高维空间转换到低维空间（在低维空间中的维数为主要成分的个数），获得每一步流场信息对应的系数 $\boldsymbol{\alpha}^t = (\alpha_1^t, \alpha_1^t, \cdots, \alpha_r^t)$，其中 $t = 1,2,\cdots,N$，r 为主要成分的个数；（4）构建输入和输出样本数据，如取 K 为每一个输入和输出样本的序列长度，构建输入样本 $\{\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \cdots, \boldsymbol{\alpha}^K\}$，$\{\boldsymbol{\alpha}^2, \boldsymbol{\alpha}^3, \cdots, \boldsymbol{\alpha}^{K+1}\}$，$\cdots$，$\{\boldsymbol{\alpha}^{N-K}, \boldsymbol{\alpha}^{N-K+1}, \cdots, \boldsymbol{\alpha}^{N-1}\}$，以及对应的输出样本 $\{\boldsymbol{\alpha}^2, \boldsymbol{\alpha}^3, \cdots, \boldsymbol{\alpha}^{K+1}\}$，

$\{\boldsymbol{\alpha}^3, \boldsymbol{\alpha}^4, \cdots, \boldsymbol{\alpha}^{K+2}\}$，…，$\{\boldsymbol{\alpha}^{N-K+1}, \boldsymbol{\alpha}^{N-K+2}, \cdots, \boldsymbol{\alpha}^N\}$；（5）利用 GRU 网络模型学习输入与输出之间的关系，最终获得二维圆柱稳定绕流流场的预测模型。

三、数据准备

（1）选取如图 10 所示的二维圆柱绕流问题为研究对象，利用网格前处理软件建立流场网格模型，网格文件 naca3412.pw 存放在…/example3 路径下。

（2）将网格文件导入 ANSYS 19.0 软件中的 Fluent 模块，建立 CFD 分析模型，流场边界条件如图 12 所示。采用非定常不可压流计算，湍流模型选用 SST $k-\omega$ 模型，并使用低雷诺数修正，计算 case 文件 cylinder.cas 存放在…/example3 路径下。本例中，来流的雷诺数 Re 取为 100。图 22 给出了非定常计算稳定时绕流流场的静压云图、迹线图和 x 向速度云图。

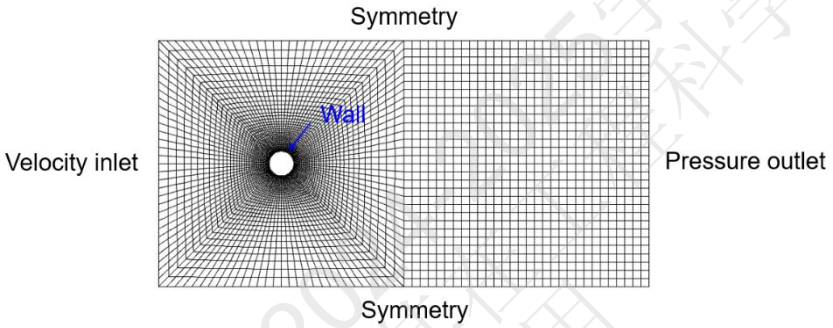
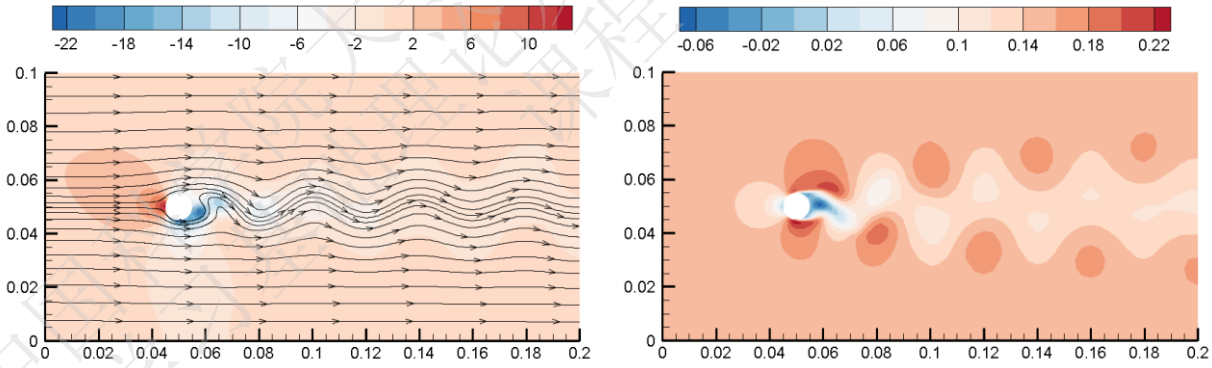


图 21 二维圆柱绕流模型边界条件示意图



（a） （b）

图 22 $Re = 100$ 时绕流流场的（a）静压云图、迹线图和（b）x 向速度云图

（3）非定常计算稳定后，连续输出 600 个时间步流场节点的静压（pre.csv）、x 向速度（vx.csv），y 向速度（vy.csv）和涡量（vo.csv）信息，均存放在…/example3 路径下。每个文件均为 5251×600 的数据矩阵，其中，第 1 行为列标签，第 2~5251 行分别为每个节点上的流场物理信息（共 5250 个网格节点）；第 1~600 列分别对应每一个时间步上的流场物理信息。若需要流场中的其他物理信息作为预测的目标，可通过 CFD 求解获得。本例中，将以 x 向速

度场的预测为例给出完整的学习程序编写过程及相关说明。

四、编程过程

（1）打开 python 开发工具（IDE），在…/example3 路径下新建 cylinder.py 文件。

（2）计算数据读取

读取 x 方向速度场的 CFD 计算结果数据 vx.csv。参考代码如下（[·] 为行号标记）：

```
[220]  # 1.计算数据读取
[221]  import pandas as pd
[222]  vx_1 = pd.read_csv('vx.csv',error_bad_lines=False)  # 读取 CFD 计算结果数据，
    5250 × 600，行：网格节点；列：时间步
[223]  vx = vx_1.values
[224]  vx = vx.T # 转置，600 × 5250，行：时间步；列：网格节点
```

执行本段代码后将得到 x 方向的节点速度矩阵 vx（600×5250），每一行代表一个时间步，每一列对应一个流场节点。

（3）主成分分析

利用主成分分析方法对速度矩阵 vx 进行降维。参考代码如下：

```
[225]  # 2. 主成分分析
[226]  from sklearn.decomposition import PCA
[227]  import numpy as np
[228]  from sklearn.preprocessing import StandardScaler
[229]  vx_scaler = StandardScaler()
[230]  vx0 = vx_scaler.fit_transform(vx) # 将矩阵 vx 的均值规范化为 0，方差规范化为 1
[231]  pca = PCA (n_components=0.99) # 保证降维后的数据保持 99% 的信息
[232]  pca.fit(vx0) # 主成分分析
[233]  vx_pca = pca.transform(vx0) # 主成分系数矩阵，600 × 4，行：时间步；列：第一至四
    主成分系数
```

执行本段代码后将得到降维后的系数矩阵 vx_pca，该矩阵的大小为 600×4，每一行代表一个时间步，第 1 至 4 列分别为第一至四主成分系数，即速度场由 5250 维降至 4 维，且保留了原速度场 99% 的信息。

（4）输入和输出样本数据构造

构造输入样本数据矩阵 A_x 和输出样本数据矩阵 A_y。参考代码如下：

```
[234]  # 3. 输入和输出样本数据构造
[235]  order_pca = vx_pca.shape[1] # 选取的主成分个数
[236]  num_step = vx.shape[0] # 读入数据的时间步数
[237]  len_sample = 5 # 每一个样本包含的时间步数
[238]  # 构造输入样本，2975 × 4，第 1~5 行：时间步 1~5，第 6~10 行：时间步 2~6，……，第
       2971~2975 行：时间步 595~599；列：每个时间步对应的第一至四主成分系数
[239]  A_x = np.zeros(((num_step-len_sample)*len_sample, order_pca))
[240]  A_y = np.zeros(((num_step-len_sample)*len_sample, order_pca))
[241]  for i in range(0,num_step-len_sample):
[242]      A_x[i*5,:] = vx_pca[i,:]
[243]      A_x[i*5+1,:] = vx_pca[i+1,:]
[244]      A_x[i*5+2,:] = vx_pca[i+2,:]
[245]      A_x[i*5+3,:] = vx_pca[i+3,:]
[246]      A_x[i*5+4,:] = vx_pca[i+4,:]
[247]  # 构造输出样本，2975 × 4，第 1~5 行：时间步 2~6，第 6~10 行：时间步 3~7，……，第
       2971~2975 行：时间步 596~600；列：每个时间步对应的第一至四主成分系数
[248]  for i in range(0,num_step-len_sample):
[249]      A_y[i*5,:] = vx_pca[i+1,:]
[250]      A_y[i*5+1,:] = vx_pca[i+2,:]
[251]      A_y[i*5+2,:] = vx_pca[i+3,:]
[252]      A_y[i*5+3,:] = vx_pca[i+4,:]
[253]      A_y[i*5+4,:] = vx_pca[i+5,:]
```

执行本段代码后将分别得到输入样本数据矩阵 A_x（2975×4）和输出样本数据矩阵 A_y（2975×4）。

（5）输入和输出数据的标准化处理

将输入矩阵 A_x 中的每一个元素按列线性标准化至 $[-1,1]$，将输出矩阵 A_y 中的每一个元素按列线性标准化至 $[0,1]$。参考代码如下：

```
[254]  # 4. 输入和输出数据的标准化处理
```

```
[255]  from sklearn.preprocessing import MinMaxScaler
[256]  scalerX=MinMaxScaler(feature_range=(-1,1))
[257]  XX = scalerX.fit_transform(A_x) # 输入矩阵按列线性标准化至 [−1,1]
[258]  scalerY=MinMaxScaler(feature_range=(0,1))
[259]  YY = scalerY.fit_transform(A_y) # 输出矩阵按列线性标准化至 [0,1]
```

执行本段代码后将分别得到标准化输入矩阵 XX （2975×4）和标准化输出矩阵 YY（2975×4）。

（6）训练集和验证集划分

将输入和输出样本数据（共 595 个样本，每个样本包含 5 个时间步）划分为训练集和验证集。其中，前 400 个样本用于训练，后 195 个样本用于验证。参考代码如下：

```
[260]  # 5. 训练集和验证集划分
[261]  sample = num_step-len_sample # 样本总数
[262]  nn = 400 # 训练集样本数量
[263]  mm = sample-nn # 验证集样本数量
[264]  deltlength_x = len_sample # 每个输入样本的序列长度
[265]  deltlength_y = len_sample # 每个输出样本的序列长度
[266]  bs = 20 # 批处理大小
[267]  X = XX[:sample*deltlength_x,:] # 样本输入
[268]  Y = YY[:sample*deltlength_y] # 样本输出
[269]  n_train_x = deltlength_x*nn # 训练集输入样本的末行行数
[270]  n_validation_x = deltlength_x*sample # 验证集输入样本的末行行数
[271]  n_train_y = deltlength_y*nn # 训练集输出样本的末行行数
[272]  n_validation_y = deltlength_y*sample # 验证集输出样本的末行行数
[273]  trainX = X[:n_train_x, :] # 训练集输入样本矩阵（2000 × 4）
[274]  trainY = Y[:n_train_y, :] # 训练集输出样本矩阵（2000 × 4）
[275]  validationX = X[n_train_x:n_validation_x, :] # 验证集输入样本矩阵（975 × 4）
[276]  validationY = Y[n_train_y:n_validation_y, :] # 验证集输入样本矩阵（975 × 4）
[277]  train3DX = trainX.reshape((nn,deltlength_x,trainX.shape[1])) # 训练集输入样本
       矩阵的 3D 化（400 × 5 × 4，即样本数 × 时间步数 × 主成分系数，下同）
[278]  validation3DX = validationX.reshape((mm,deltlength_x,validationX.shape[1]))
```

```
    # 验证集输入样本矩阵的 3D 化（400 × 5 × 4）
[279]  train3DY = trainY.reshape((nn,deltlength_y,trainY.shape[1])) # 训练集输出样本
    矩阵的 3D 化（195 × 5 × 4）
[280]  validation3DY = validationY.reshape((mm,deltlength_y,validationY.shape[1]))
    # 验证集输出样本矩阵的 3D 化（195 × 5 × 4）
```

执行本段代码后将分别得到训练集上的样本输入矩阵 train3DX（400×5×4）和样本输出矩阵 train3DY（400×5×4），验证集上的样本输入矩阵 validation3DX（195×5×4）和样本输出矩阵 validation3DY（195×5×4）。

（7）神经网络模型定义

定义神经网络模型，采用循环神经网络的一种变体—— GRU 网络来构造学习模型。网络包含一个输入层、两个 GRU 隐藏层和一个输出层，两个隐藏层的神经元数目均为 10，输入的维数为 5×4，输出的维数为 5×4。第二层 GRU 隐藏层与输出层之间采用全链接，并使用 sigmoid 函数作为激活函数。参考代码如下：

```
[281]  # 6. 神经网络模型定义
[282]  from keras.models import Sequential
[283]  from keras.layers import Dense
[284]  from keras.layers.recurrent import GRU
[285]  model = Sequential()
[286]  model.add(GRU(units=10,input_shape=(train3DX.shape[1],train3DX.shape[2]),ret
    urn_sequences=True)) # 定义第一层 GRU 隐藏层
[287]  model.add(GRU(units=10,return_sequences=True)) # 定义第二层 GRU 隐藏层
[288]  model.add(Dense(units=order_pca,kernel_initializer='normal',activation='sigm
    oid')) # 定义第二层 GRU 隐藏层与输出层之间的全链接层
[289]  model.summary() # 输出模型参数信息
```

执行本段代码将完成神经网络模型的定义，并显示如图 15 所示的模型参数信息，共计 1124 个参数。

```
Layer (type)                 Output Shape              Param #
=================================================================
gru_3 (GRU)                  (None, 5, 10)             450
_________________________________________________________________
gru_4 (GRU)                  (None, 5, 10)             630
_________________________________________________________________
dense_2 (Dense)              (None, 5, 4)              44
=================================================================
Total params: 1,124
Trainable params: 1,124
Non-trainable params: 0
```

图 23 模型参数信息

（8）第一阶段模型编译

设置第一阶段训练的学习率为 0.008，自定义模型在每个输出样本最后一个时间步上的预测值与真实值之间的均方误差为损失函数，选用 adam 优化算法进行模型参数更新，并保存截至每一个训练步下使得验证集上的损失函数最小的模型为最优模型，对建立的神经网络模型进行编译。参考代码如下：

```
[290]  # 7. 第一阶段模型编译
[291]  from keras import optimizers
[292]  adma=optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7,
    decay=0, amsgrad=False) # 定义 Adam 优化器参数
[293]  # 自定义损失函数
[294]  import keras.backend as K
[295]  def myloss(y_true, y_pred):
[296]      return K.mean(K.square(y_pred[:,4:]-y_true[:,4:]),axis=-1) # 将模型在每个
    输出样本最后一个时间步上的预测值与真实值之间的均方误差定义为损失函数
[297]  model.compile(loss=myloss,optimizer='adam') # 选择损失函数、参数优化方法
[298]  # 调整学习率
[299]  lr_new=0.008
[300]  K.set_value(model.optimizer.lr,lr_new)
[301]  model.optimizer.get_config()
[302]  # 设置检查点
[303]  from keras.callbacks import ModelCheckpoint
[304]  filepath = 'model_n10n10_size5_lr0.008_epoch800_vx_best.h5' # 定义第一阶段训
    练的最优模型的文件名称
[305]  checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
    save_best_only=True, mode='min') # 保存截至每一个训练步下使得验证集上的损失函数最小
```

```
    的模型为最优模型
[306]  callbacks_list = [checkpoint]
```

执行本段代码将完成神经网络模型的编译。

（9）第一阶段模型训练

将输入矩阵 train3DX（400×5×4）和输出矩阵 train3DY（400×5×4）作为训练集的输入和输出，将输入矩阵 validation3DX（195×5×4）和输出矩阵 validation3DY（195×5×4）作为验证集的输入和输出用于模型训练。设置批处理的大小为 20，第一阶段的训练步数为 800，保存截至每一个训练步下的最优模型，并记录完成第一阶段模型训练所需的时间，以及模型在训练集和验证集上的均方误差历程。参考代码如下：

```
[307]  # 8. 第一阶段模型训练
[308]  import time
[309]  time_start = time.time()
[310]  history = model.fit(train3DX,train3DY, epochs=800, batch_size =
    bs,validation_data = (validation3DX,validation3DY), verbose = 2, shuffle =
    False, callbacks = callbacks_list)
[311]  time_end = time.time()
[312]  print('totally cost',time_end-time_start) # 显示第一阶段模型训练耗时
[313]  time_record = time_end-time_start
[314]  aa1 = history.history['loss'] # 记录训练集上的均方误差历程（第一阶段训练）
[315]  bb1 = history.history['val_loss'] # 记录验证集上的均方误差历程（第二阶段训练）
```

执行本段代码将得到第一阶段训练的最优模型，每一步训练后将显示模型在训练集和验证集上的均方误差，在完成规定的训练步数后，将显示第一阶段训练耗时。

（10）第二阶段模型编译和训练

调整第二阶段训练的学习率至 0.005，设置第二阶段的训练步数为 600，在第一阶段得到的最优模型的基础上继续训练，保存截至每一个训练步下的最优模型，并记录完成第二阶段模型训练所需的时间，以及模型在训练集和验证集上的均方误差历程。参考代码如下：

```
[316]  # 9. 第二阶段模型编译和训练
[317]  # 第二阶段模型编译
[318]  from keras.models import load_model
```

```
[319]  model=load_model('model_n10n10_size5_lr0.008_epoch800_vx_best.h5',custom_objects={'myloss':myloss}) # 加载第一阶段训练后使得验证集上的损失函数最小的最优模型
[320]  # 调整学习率
[321]  lr_new=0.005
[322]  K.set_value(model.optimizer.lr,lr_new)
[323]  # 设置检查点
[324]  from keras.callbacks import ModelCheckpoint
[325]  filepath = 'model_n10n10_size5_lr0.008+lr0.005_epoch1400_vx_best.h5' # 定义第二阶段训练的最优模型的文件名称
[326]  checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')  # 保存截至每一个训练步下使得验证集上的损失函数最小的模型为最优模型
[327]  callbacks_list = [checkpoint]
[328]  # 第二阶段模型训练
[329]  import time
[330]  time_start=time.time()
[331]  history =model.fit(train3DX,train3DY, epochs=600, batch_size=bs,validation_data=(validation3DX,validation3DY), verbose=2, shuffle=False, callbacks=callbacks_list)
[332]  time_end=time.time()
[333]  print('totally cost',time_end-time_start) # 显示第二阶段模型训练耗时
[334]  aa2=history.history['loss'] # 记录训练集上的均方误差历程（第二阶段训练）
[335]  bb2=history.history['val_loss'] # 记录验证集上的均方误差历程（第二阶段训练）
[336]  # 输出两个阶段训练的均方根误差历程
[337]  import numpy as np
[338]  aa=np.hstack((aa1,aa2))
[339]  bb=np.hstack((bb1,bb2))
[340]  name1=['loss']
[341]  ex1=pd.DataFrame(columns=name1,data=aa)
[342]  ex1.to_csv('…/example3/vx_loss.csv') # 输出训练集上的均方误差历程
[343]  name2=['val_loss']
```

```
[344]  ex2=pd.DataFrame(columns=name2,data=bb)
[345]  ex2.to_csv('…/example3/vx_val_loss.csv') # 输出验证集上的均方误差历程
```

执行本段代码将得到第二阶段训练的最优模型，每一步训练后将显示模型在训练集和验证集上的均方误差，在完成规定的训练步数后，将显示第二阶段训练耗时，并将两个阶段训练中模型在训练集和验证集上的预测均方误差历程 vx_loss.csv 和 vx_val_loss.csv 输出至路径…/example3 下。

（11）模型预测

将验证集上的 195 个输入和输出样本作为测试集，以测试集上的标准化输入矩阵 testX_nor（975×4）为输入，利用训练得到的模型计算测试集上的标准化预测输出值，通过线性标准化处理的反变换过程获得第 406~600 个时间步上 x 向速度的主成分系数的预测值 Y_pca_pre（195×4），并通过主成分分析的逆过程求得第 406~600 个时间步上流场中每个节点的规范化 x 向速度预测值 Y_pre（195 × 5250），最终通过样本数据规范化处理的反变换过程得到第 406~600 个时间步上流场中每个节点的 x 向速度预测值 vx_pre（195 × 5250），计算每个时间步上所有节点 x 向速度的预测均方根误差 vx_rmse.csv，并输出至路径…/example3 下。参考代码如下：

```
[346]  # 10. 模型预测
[347]  testX_nor = XX[nn*deltlength_x:,:] # 标准化后的测试集样本输入，975 × 4
[348]  testY_nor = YY[nn*deltlength_y:,:] # 标准化后的测试集样本输出，975 × 4
[349]  import numpy as np
[350]  Y_pre_nor = np.zeros((mm,4)) # 195 × 4，其中，第 1~195 行对应第 406~600 个时间步
[351]  X0 = np.zeros((3,4))
[352]  X0[0,0] = min(A_x[:,0])
[353]  X0[0,1] = min(A_x[:,1])
[354]  X0[0,2] = min(A_x[:,2])
[355]  X0[0,3] = min(A_x[:,3])
[356]  X0[1,0] = max(A_x[:,0])
[357]  X0[1,1] = max(A_x[:,1])
[358]  X0[1,2] = max(A_x[:,2])
[359]  X0[1,3] = max(A_x[:,3])
[360]  X = testX_nor[:5,:] # 测试集上的第 1 个输入样本
```

```
[361]  # 加载最优模型（该最优模型为两个阶段训练中使得验证集上的均方误差最小的模型）
[362]  from keras.models import load_model
[363]  model=load_model('....h5',custom_objects={'myloss':myloss})
[364]  # 模型在测试集上的预测
[365]  for i in range(0,mm): # 这里，mm = 195
[366]      test3DX = X.reshape((1,deltlength_x,X.shape[1])) # 单个输入样本的 3D 化输入矩阵（1 × 5 × 4）
[367]      test3DY = model.predict(test3DX) # 单个输入样本的 3D 标准化预测值（1 × 5 × 4）
[368]      test2DY = test3DY.reshape((deltlength_y,4)) # 将测试集上的 3D 输出矩阵（1 × 5 × 4）转换为 2D 输出矩阵（5 × 4），即单个输入样本的 2D 标准化预测值
[369]      test2DY_pre = scalerY.inverse_transform(test2DY) # 单个输入样本的预测值（5 × 4）
[370]      X0[2,:] = test2DY_pre[4,:]
[371]      X_nor = scalerX.fit_transform(X0) # 将样本输入标准化为 [−1,1]
[372]      Y_pre_nor[i,:] = test2DY[4,:] # 测试集上第 i + 1 个样本的输出中最后一个时间步上的标准化预测值，即第 i + 406 个时间步上的预测值
[373]      # 将第 i + 1 个输入样本的后四个时间步上的值赋给第 i + 2 个输入样本的前四个时间步，将第 i + 1 个输出样本的最后一个时间步上的预测值赋给第 i + 2 个输入样本的最后一个时间步
[374]      X[0:4,:] = X[1:,:]
[375]      X[4,:] = X_nor[2,:]
[376]  Y_pca_pre = scalerY.inverse_transform(Y_pre_nor) # 第 406~600 个时间步上 x 向速度的主成分系数预测值（195 × 4）
[377]  Y_pre=pca.inverse_transform(Y_pca_pre) # 第 406~600 个时间步上流场中每个节点的规范化 x 向速度预测值（195 × 5250）
[378]  vx_pre = vx_scaler.inverse_transform(Y_pre) # 第 406~600 个时间步上流场中每个节点的 x 向速度预测值（195 × 5250）
[379]  # 计算第 406~600 个时间步上所有节点的 x 向速度预测值的均方根误差
[380]  from sklearn.metrics import mean_squared_error
[381]  from math import sqrt
```

```
[382]  rmse = np.zeros((195,2))
[383]  for i in range(0,195):
[384]      rmse[i,0]=i+406 # 第 1 列：时间步数
[385]      rmse[i,1] = sqrt(mean_squared_error(vx_pre[i,:], vx[i+405])) # 第 2 列：
    对应时间步上流场中所有节点的 x 向速度预测均方根误差
[386]  name3=['step','rmse']
[387]  ex3=pd.DataFrame(columns=name3,data=rmse)
[388]  ex3.to_csv('.../example3/vx_rmse.csv') # 输出每个时间步上所有节点 x 向速度的预测
    均方根误差
```

执行本段代码后将得到模型在测试集上的预测值 vx_pre（195 × 5250），即第 406~600 个时间步上流场中每个节点的 x 向速度预测值，并将每个时间步上所有节点 x 向速度的预测均方根误差输出至路径…/example3 下。

五、结果

执行二维圆柱绕流流场预测程序 cylinder.py 后，以下结果文件将被输出到…/example3 路径下：（1）第一阶段训练得到的最优模型“model_n10n10_size5_lr0.008_epoch800_vx_best.h5”，第二阶段训练得到的最优模型“model_n10n10_size5_lr0.008+lr0.005_epoch1400_vx_best.h5”；（2）模型训练过程中训练集和验证集上的预测均方误差历程 vx_loss.csv 和 vx_val_loss.csv；（3）第 406~600 个时间步上流场中所有节点 x 向速度的预测值的均方根误差 vx_rmse.csv。

将模型在训练集和验证集上预测均方误差历程绘制成误差随训练步数的变化曲线，可得到如图 24 所示的预测均方误差收敛曲线。

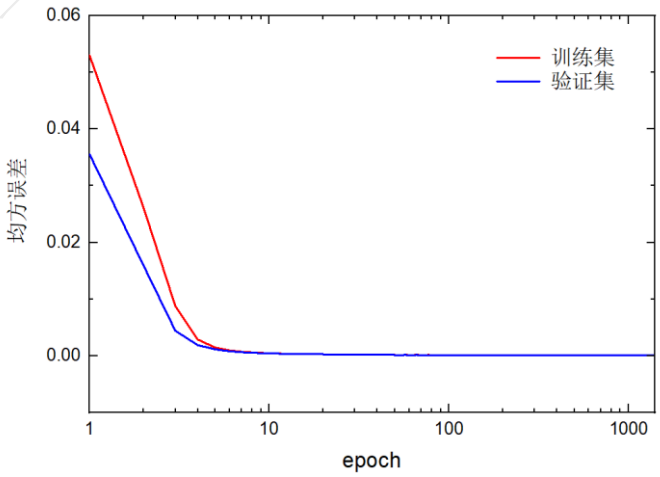


图 24 模型在训练集和验证集上的预测均方误差历程曲线

图 18 给出了第 406~600 个时间步上（即 4.06 ~ 6.0 s 时）流场中所有节点 x 向速度的预测均方根误差的分布情况。

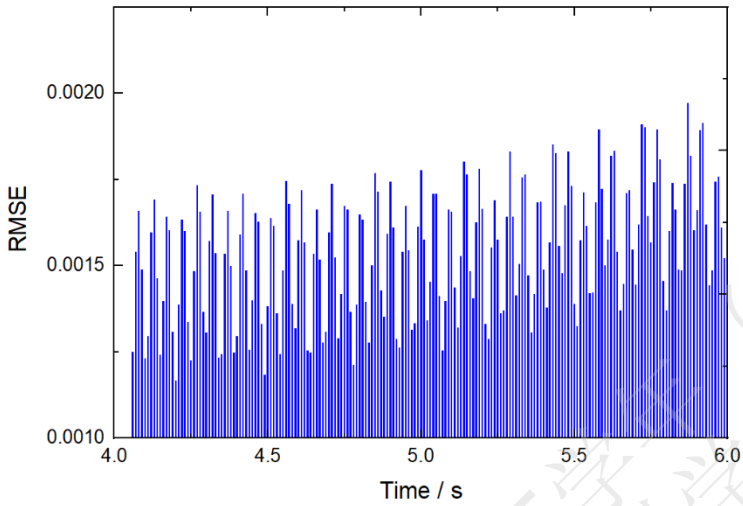


图 25 流场中所有节点 x 向速度预测值的均方根误差分布情况

此外，借助 tecplot 等后处理软件，可以将流场云图重构出来。图 19 给出了 t = 6.0 s 时 CFD 计算得到的 x 向速度场与学习模型预测值的对比，以及两者的相对误差情况。可见，学习模型具有较高的预测精度。

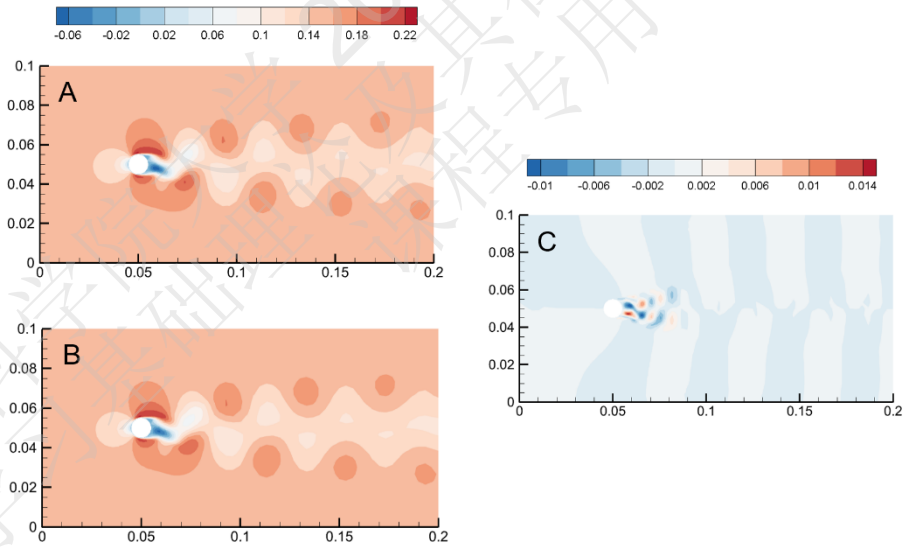


图 26 学习模型预测 x 向速度场的能力：（A）CFD 计算值，（B）预测值，（C）相对误差

六、拓展

（1）尝试通过调整网络层数、神经元数目、学习率、批处理大小等超参数，变换每个样本包含的序列长度，采用其他循环神经网络模型（如长短期记忆网络等）等方式，寻找训练效率更高、精度更好的预测模型；

（2）尝试预测二维圆柱绕流的压力场、涡量场等；

（3）探究克服时序学习模型预测误差随时间逐渐累积增大的方法。