zhejiang-yige

智能软件综合研究

实训报告



学期 2024-2025-1

班级 22计科（3）

姓名 张雅瑞

学号 2022334323029

教师 马明泽

|  |  |  |
| --- | --- | --- |
| 学号： 2022334323029 |  | 姓名： 张雅瑞 |

1. **实验目的和要求**
   1. 掌握数据预处理相关操作；
   2. 掌握深度学习模型构建、训练、评估、优化等过程；
   3. 掌握智能软件全周期开发、部署过程。
2. **实验内容**

街景字符识别数据集 (The Street View House Numbers Dataset, SVHN)为Google街景图像中的门牌号数据集。该数据集包含10个数字类别，0~9，73257个训练样本，26032个测试样本。通过数据预处理、通过数据预处理、模型构建、模型训练、模型评估等步骤，完成SVHN训练、优化、分析任务。

1. **实验任务**

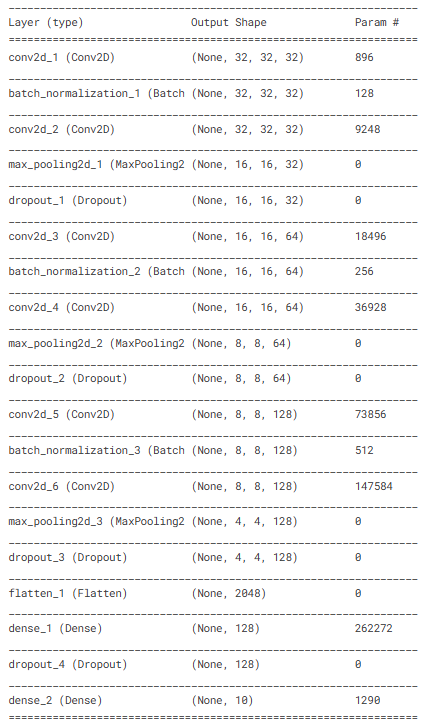
## 12周任务（街景字符识别，11.28~12.11）

### 要求

12周任务要求按以下步骤完成SVHN识别问题（问题2）。

1. **数据加载及预处理。**
2. **模型构建、训练、评估**

* 搭建以下CNN网络，不可使用预训练的模型。



* 模型可视化。利用TensorBoard生成该卷积网络的结构。
* 模型训练。
* 模型评估。包括accuracy、loss、confusion matrix等评估指标。

1. **实验过程和结果分析**

数据读取和预处理

|  |
| --- |
| import numpy as np  import seaborn as sns  import scipy.io  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import confusion\_matrix  from tensorflow.keras import Sequential  from tensorflow.keras.utils import to\_categorical  from tensorflow.keras.layers import  Conv2D, MaxPooling2D, Flatten, Dense, Dropout,BatchNormalization  from tensorflow.keras.layers import  Conv2D, MaxPooling2D, Flatten, Dense, Dropout,BatchNormalization  from tensorflow.keras.callbacks import TensorBoard  log\_dir = "board/model"  tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=0)  # 设置TensorBoard回调  train\_data = scipy.io.loadmat('data/train/train\_32x32.mat')  test\_data=scipy.io.loadmat('data/test/test\_32x32.mat')  train\_images = train\_data['X'].transpose(3, 0, 1, 2).astype('float32') / 255.0  # 归一化图像数据  train\_labels = train\_data['y'].flatten() - 1  # 假设标签是从1开始的，转换为从0开始  test\_images = test\_data['X'].transpose(3, 0, 1, 2).astype('float32') / 255.0  test\_labels = test\_data['y'].flatten() - 1  # 将标签转换为分类格式  num\_classes = 10  # SVHN 数据集有10个类（0-9的数字）  train\_labels = to\_categorical(train\_labels, num\_classes=num\_classes)  test\_labels = to\_categorical(test\_labels, num\_classes=num\_classes)    # 划分训练集和验证集（如果需要的话，可以从训练集中划分出一部分作为验证集）  X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_images, train\_labels, test\_size=0.2, random\_state=42) |

模型设计

|  |
| --- |
| CNN = Sequential([      Conv2D(32, (3, 3), activation='relu',  padding='same',input\_shape=(32, 32, 3)),      BatchNormalization(),      Conv2D(32, (3, 3), activation='relu',padding='same'),      MaxPooling2D((2, 2)),      Dropout(0.25),      Conv2D(64, (3, 3), activation='relu', padding='same'),      BatchNormalization(),  # batch\_normalization\_2      Conv2D(64, (3, 3), activation='relu', padding='same'),  # conv2d\_4      MaxPooling2D((2, 2)),  # max\_pooling2d\_2      Dropout(0.25),  # dropout\_2      Conv2D(128, (3, 3), activation='relu', padding='same'),  # conv2d\_5      BatchNormalization(),  # batch\_normalization\_3      Conv2D(128, (3, 3), activation='relu', padding='same'),  # conv2d\_6      MaxPooling2D((2, 2)),  # max\_pooling2d\_3      Dropout(0.25),  # dropout\_3      Flatten(),  # flatten\_1      Dense(128, activation='relu'),  # dense\_1      Dropout(0.5),  # dropout\_4      Dense(10, activation='softmax')  # dense\_2  ])  CNN.summary() |

模型训练

|  |
| --- |
| CNN.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  history = CNN.fit(X\_train,y\_train,batch\_size=64,epochs=8,shuffle=True,validation\_data=(X\_val,y\_val),callbacks=[tensorboard\_callback])  loss,accuracy = CNN.evaluate(test\_images,test\_labels)  print("loss:",loss)  print("Accuracy:",accuracy)  CNN.save('model/CNN\_model.keras') |

训练结果

|  |
| --- |
| plt.style.use("ggplot")  fig = plt.figure(figsize=(12,6))  epochs = range(1,9)  plt.subplot(1,2,1)  plt.plot(epochs,history.history["accuracy"],"go-")  plt.plot(epochs,history.history["val\_accuracy"],"ro-")  plt.title("Model Accuracy")  plt.xlabel("Epochs")  plt.ylabel("Accuracy")  plt.legend(["Train","val"],loc = "upper left")  plt.subplot(1,2,2)  plt.plot(epochs,history.history["loss"],"go-")  plt.plot(epochs,history.history["val\_loss"],"ro-")  plt.title("Model Loss")  plt.xlabel("Epochs")  plt.ylabel("Loss")  plt.legend(["Train","val"],loc = "upper left")  plt.show()  y\_pred = CNN.predict(test\_images).argmax(axis=1)  y\_true = test\_labels.argmax(axis=1)  conf\_matrix = confusion\_matrix(y\_true, y\_pred)    # 绘制混淆矩阵  plt.figure(figsize=(10, 8))  sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.arange(num\_classes), yticklabels=np.arange(num\_classes))  plt.xlabel('Predicted Label')  plt.ylabel('True Label')  plt.title('Confusion Matrix')  plt.show() |

1. **实验总结**
   1. **总结**

详细记录实验过程中使用的环境配置、版本信息、代码修改和结果，对于后续的调试和优化非常重要。

通过对比不同配置下的实验结果，可以更好地理解各个组件对模型性能的影响，并为未来的实验提供宝贵的参考。

* 1. **心得体会**

在这次实验中，我深刻体会到了不同Python版本对训练结果的显著影响。第一次实验使用的是Python 3.8和TensorFlow 2.14，在相同代码下，训练出来的模型准确度只有0.18，这让我感到非常困惑和挫败。经过多次尝试和检查代码后，我意识到可能是环境配置的问题。

于是，我决定重新配置实验环境。经过无数次尝试和调整，终于成功搭建了Python 3.11和TensorFlow 2.18的环境。这一次，训练结果令人惊喜，模型准确度达到了0.94，不仅如此，运行界面也发生了明显的变化，更加友好和直观。

需要特别注意的是，在使用Python 3.11时，TensorFlow版本必须是2.18才能正常运行。如果使用TensorFlow 2.17或更低版本，会导致无法运行tensorflow.keras库，这完全是版本兼容性的问题。

这一经历让我深刻认识到，环境配置在机器学习实验中的重要性。即使是代码完全相同，不同的Python和TensorFlow版本也会导致截然不同的结果。在遇到类似问题时，不能仅仅依赖工具或助手（例如GPT），因为它们可能会简单地认为是环境未正确安装，而忽略了版本兼容性的问题。因此，在进行实验和调试时，务必仔细检查环境配置和版本兼容性，以确保实验结果的可靠性和准确性。