AU3304 数字图像处理基础

手势识别大作业

学　　号: 521021910404

姓　　名: 彭俊杰

学　　院: 电子信息与电气工程学院

专　　业: 自动化

一、数据采集

1、训练集和验证集

将文件'rps.zip'解压后，作为数据集的总和。

```
with zipfile.ZipFile('rps.zip', 'r') as zip_ref:
    zip_ref.extractall('rps')
```

每次运行，数据集随机划分为训练集和验证集，训练集占 80%，验证集占 20%。

```
X_train, X_val, y_train, y_val = train_test_split(train_hog_features, train_labels, test_size=0.2, random_state=None)
```

```
print(f"Training set size: {training_set_size} images")
print(f"Validation set size: {validation_set_size} images")
```

运行后，'rock'、'paper'、'scissors'三种图片训练集和验证集数量总量分别为 2016、504。

2、测试集

将文件' rps-test-set.zip'解压后，作为训练集的总和。

```
with zipfile.ZipFile('rps-test-set.zip', 'r') as zip_ref:
    zip_ref.extractall('rps-test-set')

print(f"Number of images in the test set: {test_set_size}")
```

'rock'、'paper'、'scissors'三种图片测试集数量总量为 372。

二、数据预处理

```
1. def load_dataset(directory):
2.     images = []  # 存储图像数据的列表
3.     labels = []  # 存储标签的列表
4.     classes = {'rock': 0, 'paper': 1, 'scissors': 2}  # 将手势类别映射到标签的字典
5.
6.     for class_name, class_label in classes.items():  # 遍历手势类别及其对应的标签
7.         class_path = os.path.join(directory, class_name)  # 构建手势类别的路径
8.         for filename in os.listdir(class_path):  # 遍历手势类别文件夹下的图像文件
9.             img_path = os.path.join(class_path, filename)  # 构建图像文件的完整路径
10.             img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # 读取图像并转换为灰度图
11.             img = cv2.resize(img, (50, 50))  # 调整图像大小为(50, 50)
12.             images.append(img)  # 将处理后的图像添加到列表中
13.             labels.append(class_label)  # 将对应的标签添加到列表中
14.
15.     return np.array(images), np.array(labels)  # 将图像和标签转换为 NumPy 数组并返回
```

在 `load_dataset(directory)` 中，1-3 行，为了将解压得到的数据集附上标签，用以区分'rock'、'paper'、'scissors'三种图片，准备了两个列表。

将图像转换为灰度图，以减少计算复杂性。

```
1. img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
```

这行代码的执行会读取指定路径的图像，并将其转换为灰度图。在灰度图中，每个像素只有一个灰度值，而不是彩色图像中的红、绿、蓝三个通道的颜色值。灰

度图更加简单，通常用于图像处理和计算机视觉任务中。在一些情况下，使用灰度图可以降低计算成本，同时保留图像的主要特征。

使用 OpenCV 库读取和调整图像大小，归一化为(50, 50)

```
1.  img = cv2.resize(img, (50, 50))
```

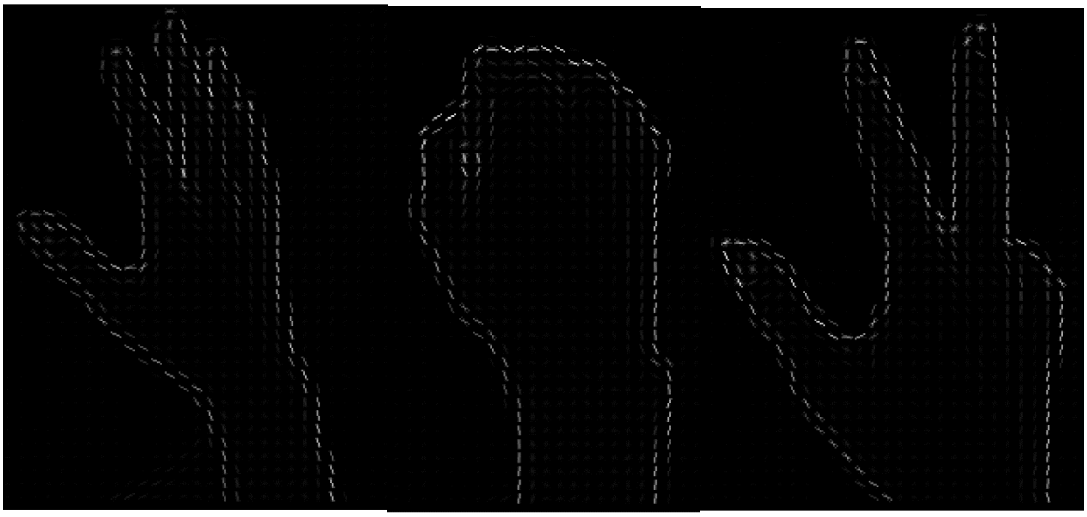
在机器学习中，经常需要将图像调整为统一的大小，以便输入模型进行训练。这有助于确保输入数据的一致性，使模型能够更好地学习和泛化。

三、模型设计——HOG+SVM 算法

在 2005 年 CVPR 上，两位研究人员提出利用 Hog 进行特征提取，利用线性 SVM 作为分类器，从而实现人的检测，此后 HOG 与 SVM 结合的方法开始在行人检测上有了较多的应用。这一方法的基本步骤为：数据标注与预处理、HOG 运算、SVM 训练。本次手势识别也采取 HOG 进行特征提取，利用线性 SVM 作为分类器。

1、经典特征提取

采用了 HOG 特征提取方法，该方法用于捕捉图像中的边缘和纹理信息。



可以看到三种手势的 HOG 特征图有明显的区分，可以进行分类。

```
# 提取 HOG 特征
def extract_hog_features(images):
    hog_features = []
    for img in images:
        features, _ = hog(img, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), block_norm='L2-Hys', visualize=True)
        hog_features.append(features)
    return np.array(hog_features)

train_hog_features = extract_hog_features(train_images)

 test_hog_features = extract_hog_features(test_images)
```

orientation: 指定块的个数。把所有的方向都转换为 0°~180°内，然后按照 orientation 划分块，选定的 orientation= 9，则 bin 一共有 9 个，每 20°有一个。

pixels_per_cell=(8, 8)： 每个细胞单元的像素数，是一个 tuple 类型数据

cell_per_block：每个块内有多少个细胞单元，tuple 类型，例如(2,2)，意思是将 block 均匀划分为 2x2 的块。

visualize=True：返回 HOG 图像以进行可视化。

block_norm='L2-Hys'：块的归一化方法。'L2-Hys'是用于规范化直方图值的常用方法。

2、分类器设计

线性分类器的基本思想是在特征空间中找到一个超平面，将不同类别的样本分隔开。对于二分类问题，这个超平面可以被看作是一个直线，而对于多分类问题，它是一个超平面。

在 SVM 中，通过最大化类别之间的间隔来选择这个超平面。间隔是指离超平面最近的样本点到超平面的距离。通过最大化这个间隔，SVM 试图找到一个决策边界，使得对新的样本点进行分类时具有较好的泛化性能。在线性分类中，决策边界是一个线性函数，其方程可以表示为 $a_1x_1 + a_2x_2 + ... + a_nx_n + b = 0$，其中$a_1, a_2, ..., a_n$是模型的权重参数，$b$是截距。

线性核函数是 SVM 中的一种核函数，它在特征空间中表示为内积。对于给定的两个样本点x和y，线性核函数的形式是$k(x, y) = x \cdot y$，即输入特征的内积。

通过使用线性核函数，SVM 尝试在特征空间中找到一个线性决策边界，使得训练数据能够被有效地分离。

利用 scikit-learn 中用于实现支持向量机（SVM）的部分，搭建线性支持向量机。

```
svm_model = svm.SVC(kernel='linear', C=1) #C 是正则化参数
svm_model.fit(X_train, y_train)
```

在训练集上训练 SVM 模型，其中 X_train 是训练集的特征，y_train 是对应的标签。

四、HOG+SVM 模型测试

1、测试结果

```
# 在验证集上进行预测
val_predictions = svm_model.predict(X_val)

# 计算验证集准确率
val_accuracy = accuracy_score(y_val, val_predictions)
print(f'Validation Accuracy: {val_accuracy}')
```

```
# 在测试集上进行预测
test_predictions = svm_model.predict(test_hog_features)

# 计算测试集准确率
test_accuracy = accuracy_score(test_labels, test_predictions)
print(f'Test Accuracy: {test_accuracy}')
```

运行结果如下：

```
(base) D:\AU3304\hw_final\tmp>python test.py
Validation Accuracy: 1.0
Test Accuracy: 0.7553763440860215
Training set size: 2016 images
Validation set size: 504 images
Number of images in the test set: 372
```

训练集的不同对正确率有些许影响：

测试次数	1	2	3	4	5
验证集正确率	1	1	1	1	1
数据集正确率	0.7634	0.7527	0.7554	0.7573	0.7608

2、结果分析

验证集准确率达到了 100%，说明模型在训练数据的验证子集上表现非常好。可以看到验证集正确率总为 1。也就是说，如果直接从训练数据集中随机选出图像组成测试样本集，用测试集测试分类的准确率，将为 100%。

验证集准确率达到 100% 的原因可能为数据泄露或者过拟合。数据泄露是指在模型训练中，将验证集的信息泄露到训练集中，导致在验证集上的表现过于乐观；模型可能在训练集上过拟合，可以尝试使用更复杂的模型（例如，增加卷积层、神经元等）或者通过正则化来减轻过拟合。

测试集准确率为 75.79%，这表明模型在未见过的数据上的性能相对较好，但仍有改进的空间。

改进策略有以下几点：

特征提取改进：尝试使用其他特征提取方法，例如卷积神经网络（CNN），以更好地捕捉图像的抽象特征。

模型调优：调整 SVM 的超参数，例如尝试不同的核函数、调整正则化参数 C，以优化模型在测试集上的性能。

数据增强：对训练数据进行数据增强，通过随机旋转、平移、缩放等方式生成更多的样本，提升模型的泛化能力。

尝试集成学习：尝试使用集成学习方法，如随机森林或梯度提升树，以进一步提高模型的稳定性和性能。

处理噪声：对输入图像进行预处理，例如使用滤波器或去噪技术，以降低噪声对模型的影响。

本次手势识别分类器选用线性分类器，但灰度图像并不完全符合线性可分，因此导致正确率在 76%左右。最佳改进方法是选择更加合适的分类器。使用支持向量机（SVM）时，选择合适的核函数对模型的性能影响较大。一些常见的核函数如下：

1）线性核函数：

当数据线性可分时，或者特征维度较高时，线性核函数通常是一个不错的选择。它在计算效率上较高，适合处理大规模数据集。

2）多项式核函数：

当数据的决策边界是多项式形状时，可以考虑使用多项式核函数。它可以通过调整多项式的阶数来适应不同复杂度的决策边界。使用多项式函数来映射输入数据，将其转换为高维空间，从而使数据在该空间中更容易分离。公式为 $K(x, y) = (x * y + c)^d$，其中 c 是常数，d 是多项式的次数。

3）高斯核函数（RBF 核函数、径向基核函数）：

通过使用高斯分布来映射数据到高维空间。公式为 $K(x, y) = \exp(-\gamma * ||x - y||^2)$，其中 γ 是一个调整尺度的参数。高斯核函数通常对于非线性可分的

问题表现较好。它可以捕捉到数据之间的复杂关系，但在计算复杂度上较高，可能在大规模数据集上需要更多的计算资源。

4）Sigmoid 核函数：

使用 sigmoid 函数来进行映射。公式为 K(x, y) = tanh(α * x * y + c)，其中 α 和 c 是常数。Sigmoid 核函数在神经网络中常被使用，但在一般的 SVM 问题中使用相对较少。它可以用于处理非线性关系，但对核函数的选择较为敏感。

对于手势识别，笔者初步尝试使用了线性核函数。从结果来看是合理的。如果为了更高的正确率，可以尝试多项式核函数或高斯核函数，根据数据的复杂性来选择合适的核函数和参数。

五、进一步尝试卷积神经网络（CNN）

代码实现了一个简单的卷积神经网络（CNN）来进行手势识别。主要调用了 `torch` 库，训练集验证集分别占' rps.zip'的 80%、20%，' rps-test-set.zip'全部图片构成测试集。其核心内容包括：

1）数据转换为 PyTorch 张量

将 NumPy 数组转换为 PyTorch 张量，并进行必要的维度调整。

```
1. X_train_tensor = torch.from_numpy(X_train).unsqueeze(1).flo
   at()
2. y_train_tensor = torch.from_numpy(y_train).long()
3.
4. X_val_tensor = torch.from_numpy(X_val).unsqueeze(1).float()
5. y_val_tensor = torch.from_numpy(y_val).long()
```

2）构建卷积神经网络（CNN）模型

使用 SimpleCNN 类定义了一个简单的三层卷积神经网络，包括卷积层、池化层和全连接层。

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.dropout1 = nn.Dropout2d(0.1)  # Dropout before the pooling layer
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.dropout2 = nn.Dropout2d(0.1)  # Dropout before the pooling layer
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(128 * 6 * 6, 128)
        self.fc2 = nn.Linear(128, 3)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.dropout1(x)
        x = self.pool(F.relu(self.conv2(x)))
        x = self.dropout2(x)
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128 * 6 * 6)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

3）模型初始化

初始化了模型、损失函数（交叉熵损失）和优化器（Adam 优化器）。

```
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

4）数据加载

使用 TensorDataset 和 DataLoader 将训练数据封装成 PyTorch 数据集和数据加载器。

```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

5）模型训练：

通过多个 epoch 循环来训练模型，使用 Adam 优化器进行反向传播，最小化交叉熵损失。

```
num_epochs = 5
for epoch in range(num_epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

其余部分与传统方法（HOG+SVM）保持一致。

六、卷积神经网络模型测试结果与分析

三次运行结果分别为 86.29%、85.48%、88.44%。相比传统方法，卷积神经网络方法在正确率上有一定提升，但仍有进步空间。

```
(base) D:\AU3304\hw_final\tmp>python code5.py
2024-01-14 20:05:08.747983: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\12345\AppData\Roaming\Python\Python39\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning
: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(
2024-01-14 20:05:22.228366: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, reb
uild TensorFlow with the appropriate compiler flags.
Epoch 1/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.5873 - loss: 10.1852 - val_accuracy: 0.9722 - val_loss: 0.0842
Epoch 2/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9953 - loss: 0.0513 - val_accuracy: 1.0000 - val_loss: 0.0033
Epoch 3/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 4/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 6.4607e-04
Epoch 5/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 1.0000 - loss: 5.6299e-04 - val_accuracy: 1.0000 - val_loss: 4.5338e
-04
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Test Accuracy: 0.717741935483871
Validation Accuracy: 1.0
Final Test Accuracy: 0.8629032258064516
Training set size: 2016 images
Validation set size: 504 images
Number of images in the test set: 372
```

```
(base) D:\AU3304\hw_final\tmp>python code5.py
2024-01-14 20:03:42.722379: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\12345\AppData\Roaming\Python\Python39\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning
: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(
2024-01-14 20:03:56.372610: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, reb
uild TensorFlow with the appropriate compiler flags.
Epoch 1/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.5469 - loss: 12.6427 - val_accuracy: 0.9484 - val_loss: 0.1336
Epoch 2/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9866 - loss: 0.0647 - val_accuracy: 0.9821 - val_loss: 0.0394
Epoch 3/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9977 - loss: 0.0097 - val_accuracy: 1.0000 - val_loss: 8.1119e-04
Epoch 4/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 1.0000 - loss: 7.3924e-04 - val_accuracy: 1.0000 - val_loss: 3.5939e
-04
Epoch 5/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 1.0000 - loss: 3.1628e-04 - val_accuracy: 1.0000 - val_loss: 2.4367e
-04
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Test Accuracy: 0.6639784946236559
Validation Accuracy: 0.996031746031746
Final Test Accuracy: 0.8548387096774194
Training set size: 2016 images
Validation set size: 504 images
Number of images in the test set: 372
```

```
(base) D:\AU3304\hw_final\tmp>python code5.py
2024-01-14 20:02:10.370139: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly d
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, s
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
C:\Users\12345\AppData\Roaming\Python\Python39\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarnin
: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(sh
pe)` object as the first layer in the model instead.
  super().__init__(
2024-01-14 20:02:24.142266: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, re
uild TensorFlow with the appropriate compiler flags.
Epoch 1/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.5815 - loss: 8.8303 - val_accuracy: 0.9802 - val_loss: 0.1105
Epoch 2/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9759 - loss: 0.0702 - val_accuracy: 0.9881 - val_loss: 0.0275
Epoch 3/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9977 - loss: 0.0119 - val_accuracy: 1.0000 - val_loss: 0.0025
Epoch 4/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 0.0016
Epoch 5/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 1.0000 - loss: 6.5803e-04 - val_accuracy: 1.0000 - val_loss: 0.0016
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Test Accuracy: 0.75
Validation Accuracy: 0.9841269841269841
Final Test Accuracy: 0.8844086021505376
Training set size: 2016 images
Validation set size: 504 images
Number of images in the test set: 372
```

具体的提升方法有如下几点：

1）学习率调整：尝试不同的学习率，可能通过学习率调度器（如学习率衰减）来动态调整学习率，以提高训练效果。

2）数据增强：对训练集进行数据增强，包括随机旋转、平移、缩放等操作，以扩充数据集，提高模型的泛化能力。

3）正则化：添加正则化项，如 L1 正则化、L2 正则化，以减小过拟合的可能性。

4）批量归一化：在卷积层之后添加批量归一化层，有助于加速训练过程并提高模型的稳定性。

5）调整数据集划分：尝试不同的训练集和验证集划分方式，有时候不同的划分方式会影响模型的泛化性能。

6）超参数搜索：通过网格搜索或随机搜索的方式，寻找最优的超参数组合，包括学习率、批量大小、卷积核大小等。

7）调整图像尺寸：尝试不同的图像大小，有时调整图像大小可以影响模型的性能。

8）集成学习：尝试使用集成学习方法，如投票、平均等，结合多个模型的预测结果。

七、总结与感悟

HOG 特征对目标的几何和光学变化具有一定的不变性，使其对目标的姿态、形状、光照变化相对稳健。HOG 特征能够捕捉到图像的纹理信息，有助于区分不同类别的物体。但 HOG 特征在图像的局部变化较敏感，可能受到噪声的影响。HOG 这一算法作为经典的行人识别算法，在手势识别中，手势识别与行人具有

一定的相似度，也还有着较大的可取性与很大的优化空间。

线性 SVM 在处理高维数据时具有较强的泛化能力，适用于特征维度较大的问题。在小样本数据情况下，线性 SVM 表现优秀，能够避免过拟合。但在处理非线性可分问题时，线性 SVM 的表现可能不如使用非线性核的 SVM。在噪声较多的情况下，线性 SVM 可能对异常值敏感。

卷积神经网络（CNN）通过卷积、池化操作有效捕捉图像特征，具有参数共享和空间层次结构优势，对大规模数据表现良好，且具备迁移学习潜力。然而，其训练过程需大量计算资源，对标注数据依赖较大，模型解释性差，容易过拟合。

本次大作业的完成仍有着许多不足之处。对于传统方法，本次实验正确率仅有 75%-76%，有巨大的提升空间。由于时间与精力的原因，对于非线性的 SVM 分类器，只是去查找了资料，没有真正将分类器搭建并运行测试正确率。在尝试传统方法后，进行深度的改进，由于没能在 python3.9 环境搭建支持卷积神经网络的常见的 Tensorflow 库，因此使用的是 torch 库，同样正确率也有待提高。

本次实验，我基本熟悉传统目标检测和卷积神经网络的基本思路与代码实现方法，并以此代码为基础进行调整与改进，进行基本库函数的完善与封装，并整合加入了之前历次作用和实验中使用的积累的各种算法，在此基础上完成了此次实验，真正进行了一次数字图像处理的实验，为今后的学习提供了宝贵的经验。

最后，感谢周越老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解数字图像处理开发的诸多知识，将复杂的知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。也感谢助教老师们准备的丰富的实践代码，让我轻松地体会到各种算法的作用效果。