计算机视觉作业一

题目：基于 Google Teachable Machine 实现一个计算机视觉应用

学 生 姓 名 ： 李闯

学 号 ： 2253214

专 业 班 级 ： 计算机科学与技术 2 班

指 导 教 师 ： 赵才荣

2025 年 03 月 10 日

一、实验目标

（1） 发挥个人创作力和想象力，独立完成一个计算机视觉应用小创意。

（2） 了解计算机视觉的基本原理，并对计算机视觉有初步了解。

（3） 实践计算机视觉技术，通过实现手写数字识别任务，初步了解深度学习在计算机视觉中的应用。

二、实验说明

任务一、计算机视觉应用小创意

创意名称：基于手势识别的音乐控制系统

实验环境：

- 编程语言：Python
- 主要库：TensorFlow/Keras、OpenCV、Numpy、PIL、threading、pygame
- 开发工具：VSCode

模型信息：

使用 Google 的 Teachable Machine 训练了一个 7 分类的图片识别模型。

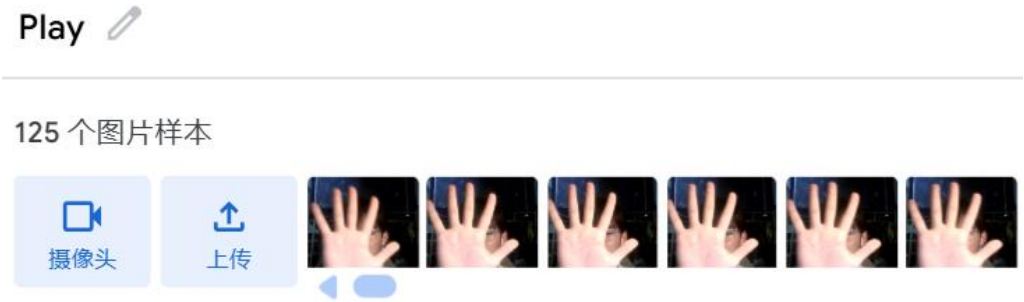


表 1 分类 0 播放音乐

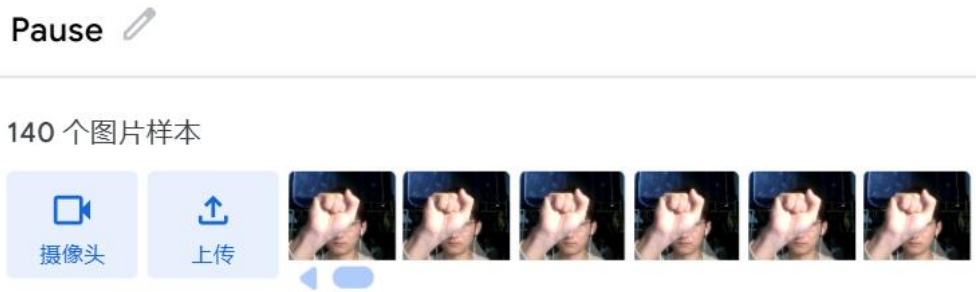


表 2 分类 1 暂停音乐

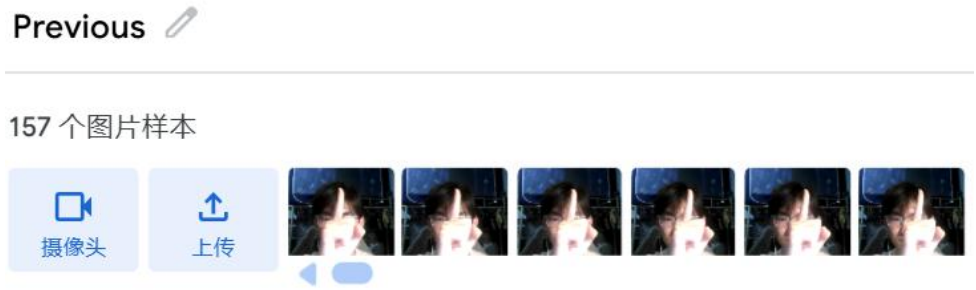


表 3 分类 2 切换音乐至上一首

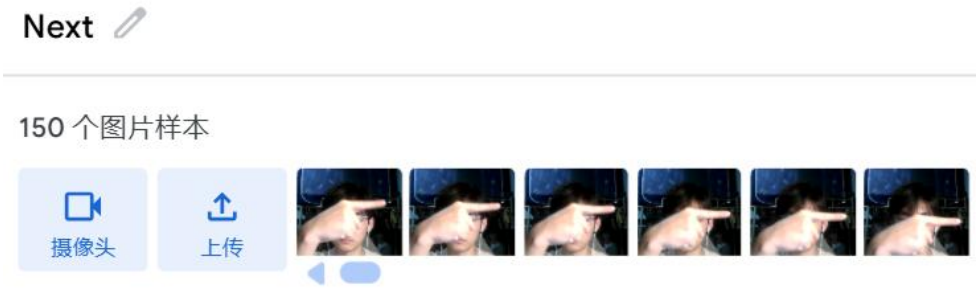


表 4 分类 3 切换音乐至下一首

表 5 分类 4 增大播放音量

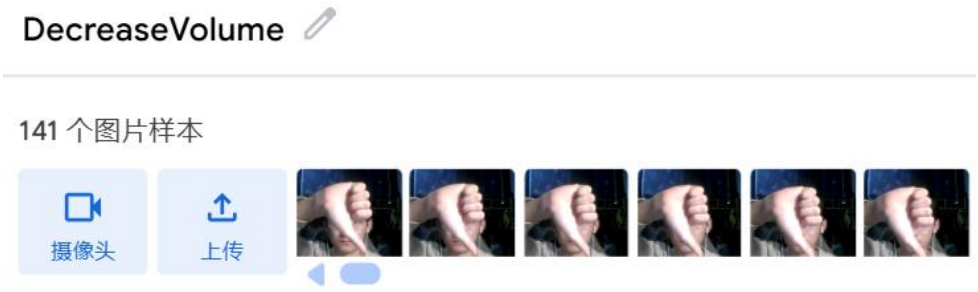


表 6 分类 5 降低播放音量

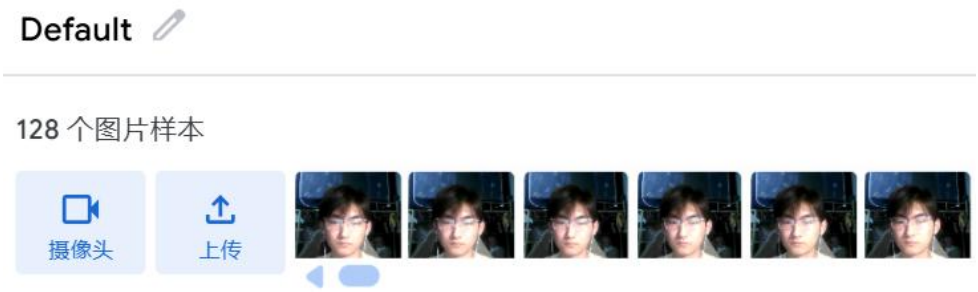
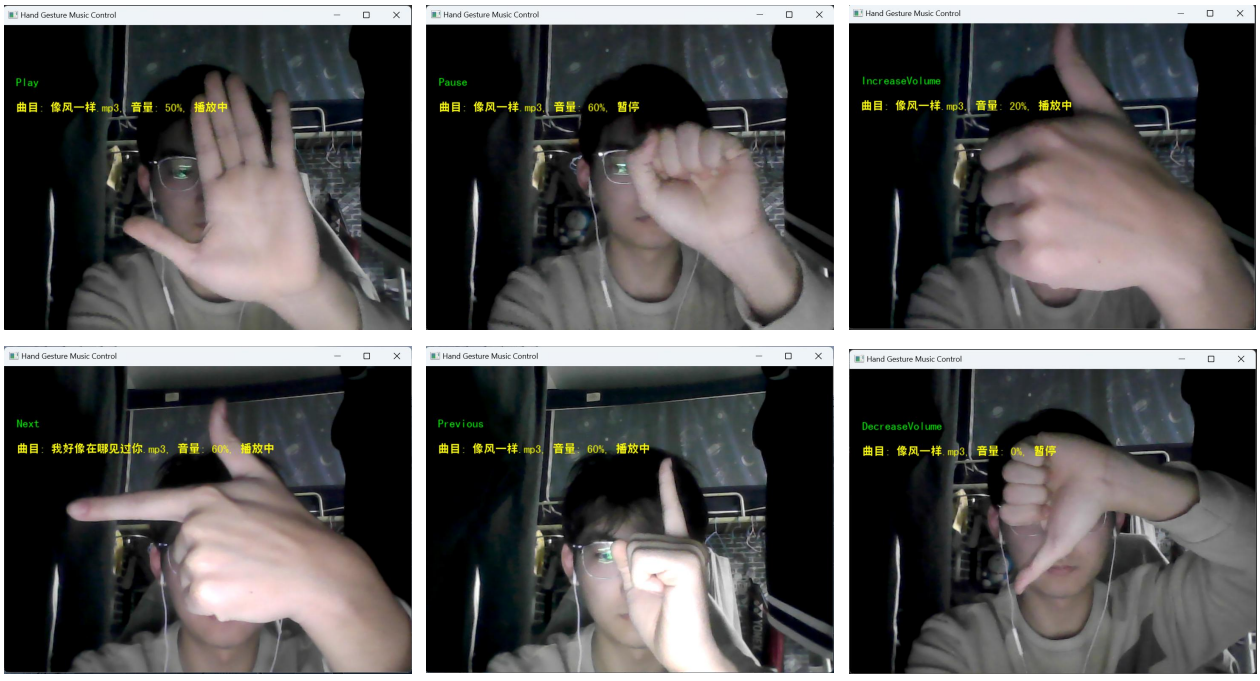


表 7 分类 6 默认状态无操作

实现步骤：

1. 初始化：读取标签、初始化语音合成与音乐播放模块。
2. 视频采集：使用 OpenCV 读取摄像头画面并存入队列。
3. 手势识别：通过 TensorFlow 进行分类预测，识别用户手势。
4. 音乐控制：根据识别出的手势执行相应的音乐控制操作（播放、暂停、音量调整等）。
5. 语音播报：通过 pyttsx3 反馈当前音乐状态。

实验结果：



核心实验代码：

视频捕捉线程代码

```
def video_capture_thread():
    """视频捕获线程（带异常处理）"""
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
```

```
        print("摄像头打开失败")
        return

    while True:
        with camera_lock: # 确保在同一时刻只有一个线程能够访问摄像头
            try:
                ret, frame = cap.read()
                if not ret:
                    print("读取帧失败")
            except cv2.error as e:
                print(f"OpenCV 错误: {e}")
            except Exception as e:
                print(f"发生了其他错误: {e}")

            # 锁住对队列的访问
            with frame_queue_lock:
                if frame_queue.full():
                    try:
                        frame_queue.get_nowait()
                    except queue.Empty:
                        pass
                frame_queue.put(frame)

    # 清理资源
    cap.release()
    cv2.destroyAllWindows()
    print("视频线程退出")
```

视频帧处理代码

```
def process_frame(frame):
    """处理视频帧并返回带标注的帧"""
    # 预处理图像
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # 转换为 RGB
    image_pil = Image.fromarray(image)  # 转换为 PIL 格式
    image_resized = ImageOps.fit(image_pil, (224, 224), Image.Resampling.LANCZOS)
    image_array = np.asarray(image_resized).astype(np.float32)  # 转换为数组
    normalized_image_array = (image_array / 127.5) - 1  # 归一化

    # 预测手势
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
    data[0] = normalized_image_array
    input_tensor = tf.convert_to_tensor(data)
    output = infer(input_tensor)

    output_keys = list(output.keys())
    prediction = output[output_keys[0]].numpy()
    index = np.argmax(prediction)
    label = class_names[index].strip()  # 获取类别标签

    # 使用 Pillow 来绘制中文文本
    music_status = f"曲目: {os.path.basename(music_files[current_track])}, 音量: {int(volume * 100)}%, "
    music_status += "暂停" if is_paused else "播放中"

    # 将 OpenCV 图像转换为 PIL 图像
    draw = ImageDraw.Draw(image_pil)
    draw.text((20, 80), label, font=font, fill=(0, 255, 0))  # 绘制手势标签
    draw.text((20, 120), music_status, font=font, fill=(255, 255, 0))  # 绘制音乐信息

    # 将 PIL 图像转换回 OpenCV 图像
    frame = np.array(image_pil)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)  # 转回 BGR 格式用于 OpenCV 显示

    # 执行控制命令（非阻塞方式）
    control_mapping = {
        "Play": lambda: process_music_action("Play"),
```

```
        "Pause": lambda: process_music_action("Pause"),
        "Nest": lambda: process_music_action("Next"),
        "Previous": lambda: process_music_action("Previous"),
        "IncreaseVolume": lambda: process_music_action("IncreaseVolume"),
        "DecreaseVolume": lambda: process_music_action("DecreaseVolume")
    }

    if label in control_mapping:
        control_thread = threading.Thread(target=control_mapping[label])
        control_thread.daemon = True
        control_thread.start()

    return frame

def process_frame_thread():
    """视频帧处理线程"""
    while True:
        with frame_queue_lock:
            if not frame_queue.empty():
                frame = frame_queue.get(timeout=0.5)
                if frame is not None:  # 确保 frame 有效
                    processed_frame = process_frame(frame)
                    cv2.imshow("Hand Gesture Music Control", processed_frame)

                    # 加入 waitKey，控制显示
                    if cv2.waitKey(1) & 0xFF == ord('q'):  # 按'q'键退出
                        break
            else:
                pass
```

音乐控制代码

```
def process_music_action(action):
    """处理音乐控制动作"""
    global current_track, volume, is_paused, last_action_time

    current_time = time.time()  # 获取当前时间

    if current_time - last_action_time < 1:
        return  # 如果没有持续 1 秒，则跳过，不执行

    with music_control_lock:
        current_time = pygame.mixer.music.get_pos() / 1000.0  # 获取当前歌曲播放时间，单位秒

        if action == "Play":
            # 检查是否正在播放，如果已播放，则无需再次加载
            if not pygame.mixer.music.get_busy():
                tts_say(f"现在开始播放 {os.path.basename(music_files[current_track])}")
                pygame.mixer.music.load(music_files[current_track])
                pygame.mixer.music.play()
                is_paused = False
            else:
                pygame.mixer.music.unpause()
                tts_say("音乐继续播放")
                is_paused = False
        elif action == "Pause":
            if not is_paused:
                pygame.mixer.music.pause()
                tts_say("音乐暂停")
                is_paused = True  # 设置为暂停状态
            else:
                tts_say("音乐已经暂停")  # 提示用户音乐已经暂停
        elif action == "Next":
            current_track = (current_track + 1) % len(music_files)
            tts_say("播放下一首")
            pygame.mixer.music.load(music_files[current_track])
            pygame.mixer.music.play()
            is_paused = False
        elif action == "Previous":
            current_track = (current_track - 1) % len(music_files)
```

```
            tts_say("播放上一首")
            pygame.mixer.music.load(music_files[current_track])
            pygame.mixer.music.play()
            is_paused = False
        elif action == "IncreaseVolume":
            volume = min(1.0, volume + 0.1)
            pygame.mixer.music.set_volume(volume)
            tts_say(f"音量上升 {int(volume*100)}%")
        elif action == "DecreaseVolume":
            print("增加音量")
            volume = max(0.0, volume - 0.1)
            print(f"volume:{volume}")
            pygame.mixer.music.set_volume(volume)
            tts_say(f"音量下降 {int(volume*100)}%")
        else:
            pass

        # 更新操作时间
        print(action)
        last_action_time = time.time()
```

任务二、简单识别分类网络--手写数字识别

任务目标：以 MNIST 手写数字作为数据集训练一个简单的分类网络。

实验环境：

- 编程语言：Python
- 主要库：PyTorch、Torchvision、Matplotlib

数据集：

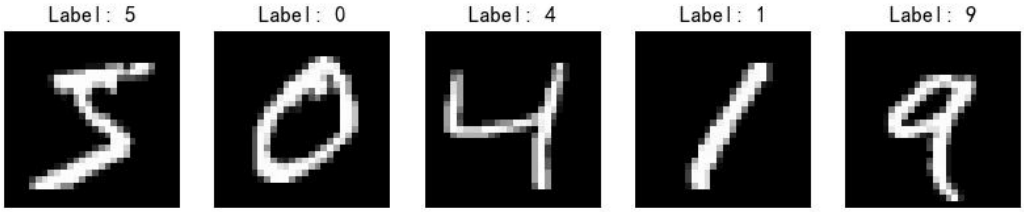


图 1 训练集样本图片

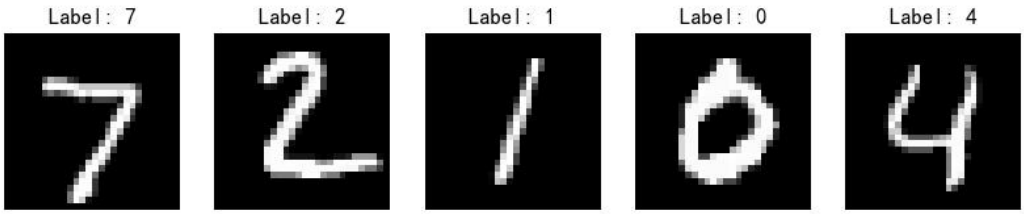


图 2 测试集样本图片

实验步骤：

1. 数据加载：使用 Pytorch 提供的 torchvision.datasets.MNIST 加载 MNIST 数据集。
2. 数据预处理：对图像进行标准化处理，并转换为张量。

3. 模型构建：构建一个简单的卷积神经网络(CNN)，包括卷积层、池化层和全连接层。
4. 模型训练：使用交叉熵损失函数和 Adam 优化器，在 GPU 上训练模型。
5. 模型评估：在训练集上评估模型准确率，并绘制混淆矩阵。
6. 可视化结果：展示部分测试样本机器预测结果。

模型信息：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 28, 28]             320
              ReLU-2          [-1, 32, 28, 28]               0
         MaxPool2d-3          [-1, 32, 14, 14]               0
            Conv2d-4          [-1, 64, 14, 14]          18,496
              ReLU-5          [-1, 64, 14, 14]               0
         MaxPool2d-6            [-1, 64, 7, 7]               0
            Linear-7                 [-1, 128]         401,536
              ReLU-8                 [-1, 128]               0
            Linear-9                  [-1, 10]           1,290
================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.65
Params size (MB): 1.61
Estimated Total Size (MB): 2.26
----------------------------------------------------------------
```

图 3 训练网络信息

实验结果：

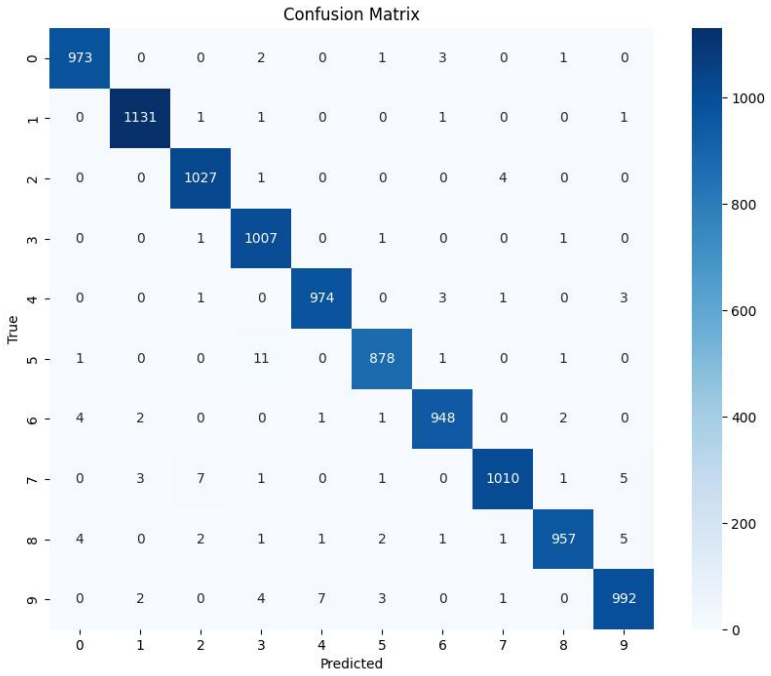


图 4 训练结果混淆矩阵

实验代码：

卷积网络模型

```
class SimpleCNN(nn.Module):
    """
    SimpleCNN: A Simple Convolutional Neural Network for MNIST digit classification

    Architecture:
    - Conv1   : 1  input channel,  32 output channels, 3x3 kernel, stride=1, padding=1
    - Conv2   : 32 input channels, 64 output channels, 3x3 kernel, stride=1, padding=1
    - MaxPool : 2x2 kernel, stride=2
    - FC1     : Fully connected layer  64*7*7 -> 128
    - FC2     : Fully connected layer  128    -> 10

    Forward Pass:
    - Conv1 -> ReLu -> MaxPool
    - Conv2 -> ReLu -> MaxPool
    - Flatten -> FC1 -> ReLu -> FC2
    """
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU() # ReLU 激活函数

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

模型训练

```
def train(model, train_loader, criterion, optimizer, epochs=5):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.train()  # 将模式设置为训练模式
    start_time = time.time()
    for epoch in range(epochs):
        running_loss = 0.0
        total_batches = len(train_loader)
        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()  # 清空梯度
            outputs = model(inputs)  # 前向传播
            loss = criterion(outputs, labels)  # 计算损失
            loss.backward()  # 反向传播
            optimizer.step()  # 更新权重
            running_loss += loss.item()

            # 每 100 个 batch 打印一次损失
            if i % 100 == 99:
                avg_loss = running_loss / 100
                print(f"Epoch [{epoch+1}/{epochs}], Batch [{i+1}/{total_batches}], Loss:
{avg_loss:.4f}")
                running_loss = 0.0

        # 每个 epoch 结束时打印平均损失
```

```
        epoch_loss = running_loss / total_batches
        print(f"Epoch [{epoch+1}/{epochs}], Average Loss: {epoch_loss:.4f}")

    end_time = time.time()
    training_time = end_time - start_time  # 计算训练时间
    print(f"Training completed in {training_time:.2f} seconds.")
```

模型测试

```
def test(model, test_loader):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval() # 将模型设置为评估模式
    correct = 0
    total = 0
    all_labels = []
    all_predicted = []

    with torch.no_grad(): # 禁用梯度测试
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predicted.extend(predicted.cpu().numpy())

    # 计算准确率
    accuracy = 100 * correct / total
    print(f'Accuracy on test set: {accuracy:.2f}%')

    # 绘制混淆矩阵
    cm = confusion_matrix(all_labels, all_predicted)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

保存与加载模型

```
def save_model(model, path="model/model.pth"):
    """保存模型到指定路径"""
    model_dir = os.path.dirname(path)
    if not os.path.exists(model_dir):
        os.makedirs(model_dir)

    torch.save(model.state_dict(), path)
    print(f"Model saved to {path}")

def load_model(model, path='model.pth'):
    """从指定路径加载模型"""
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.load_state_dict(torch.load(path))
    model.to(device)
    print(f"Model loaded from {path}")
```

主函数

```
def main():
    train_dataset, test_dataset = load_mnist_data()

    # 创建数据加载器
```

```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

model = SimpleCNN()
criterion = nn.CrossEntropyLoss() # 交叉熵损失函数
optimizer = optim.Adam(model.parameters(), lr=0.001) # Adam 优化器，学习率为 0.001

train(model, train_loader, criterion, optimizer, epochs=5)
test(model, test_loader)

# 保存模型
save_model(model, "model/mnist_cnn.pth")
```