

第 10 周教案：核心开发与调试

计算机视觉课程组

1 基本信息

周次	第 10 周
主题	核心开发与调试
学时	3 学时（160 分钟）
故事问题	让系统真正跑起来
OBE 目标	A4-系统集成：完成完整阅卷系统开发
项目阶段	集中开发 + 测试验证

2 教学目标

1. 知识目标：

- 理解系统集成的方法
- 掌握调试技巧
- 了解测试策略

2. 能力目标：

- 能够完成模块集成
- 能够进行系统测试
- 能够调试和优化系统

3. 素养目标：

- 培养工程实践能力
- 提升问题解决能力
- 建立质量意识

3 教学重点与难点

教学重点

- 模块集成与接口对接
- 系统测试与验证
- 性能优化

教学难点

- 复杂问题的调试
- 准确率优化

4 教学过程设计

4.1 环节一：集成指导（30 分钟）

4.1.1 1.1 主程序框架（15 分钟）

```
"""
自动阅卷系统主程序
"""

import cv2
import os
import sys
sys.path.append('..')

from modules.preprocess import Preprocessor
from modules.layout import LayoutAnalyzer
from modules.choice_recognizer import ChoiceRecognizer
from modules.judge_recognizer import JudgeRecognizer
from modules.essay_recognizer import EssayRecognizer
from modules.grading import GradingModule
from utils.visualization import visualize_results
from utils.logger import setup_logger

class AutoGradingSystem:
    """自动阅卷系统"""

```

```

def __init__(self, config_path='config.py'):
    """初始化系统"""
    # 加载配置
    self.config = self.load_config(config_path)

    # 初始化日志
    self.logger = setup_logger(self.config.get('log_file', 'app.log'))

    # 初始化各模块
    self.preprocessor = Preprocessor(self.config.get('preprocess', {}))
    self.layout_analyzer = LayoutAnalyzer(self.config.get('layout', {}))
    self.choice_recognizer = ChoiceRecognizer(self.config.get('choice', {}))
    self.judge_recognizer = JudgeRecognizer(self.config.get('judge', {}))
    self.essay_recognizer = EssayRecognizer(self.config.get('essay', {}))
    self.grading_module = GradingModule(self.config.get('grading', {}))

    self.logger.info("Auto Grading System initialized")

def load_config(self, config_path):
    """加载配置文件"""
    # 简化实现，实际可以用 configparser 等
    config = {
        'preprocess': {'denoise_method': 'median'},
        'choice': {'threshold': 0.3},
        'judge': {'method': 'feature'},
        'essay': {'method': 'paddle'}
    }
    return config

def process(self, image_path, output_dir='data/output'):
    """
    处理试卷图像
    """

```

参数：

image_path: 试卷图像路径

output_dir: 输出目录

返回：

处理结果字典

"""

```
self.logger.info(f"Processing image: {image_path}")
```

1. 读取图像

```
image = cv2.imread(image_path)
```

```
if image is None:
```

```
    raise ValueError(f"Cannot load image: {image_path}")
```

2. 预处理

```
self.logger.info("Step 1: Preprocessing")
```

```
preprocessed = self.preprocessor.process(image)
```

3. 版面分析

```
self.logger.info("Step 2: Layout analysis")
```

```
layout_result = self.layout_analyzer.analyze(image)
```

```
areas = layout_result['areas']
```

4. 选择题识别

```
self.logger.info("Step 3: Choice question recognition")
```

```
choice_results = []
```

```
if 'choice' in areas:
```

```
    choice_results = self.choice_recognizer.recognize_all(
```

```
        preprocessed['binary'],
```

```
        areas['choice'])
```

```
)
```

5. 判断题识别

```
self.logger.info("Step 4: Judge question recognition")
```

```
judge_results = []
```

```
if 'judge' in areas:
```

```
    judge_results = self.judge_recognizer.recognize_all(
```

```
        preprocessed['binary'],
```

```
        areas['judge'])
```

```

        )

# 6. 简答题识别
self.logger.info("Step 5: Essay question recognition")
essay_results = []
if 'essay' in areas:
    essay_results = self.essay_recognizer.recognize_all(
        preprocessed['binary'],
        areas['essay']
)

# 7. 评分
self.logger.info("Step 6: Grading")
all_answers = {
    'choice': choice_results,
    'judge': judge_results,
    'essay': essay_results
}
grading_result = self.grading_module.grade(all_answers)

# 8. 整理结果
result = {
    'image_path': image_path,
    'answers': all_answers,
    'grading': grading_result,
    'preprocessed': preprocessed,
    'layout': layout_result
}

# 9. 可视化结果
self.logger.info("Step 7: Visualization")
visualize_results(image, result, output_dir)

self.logger.info("Processing completed")
return result

def main():
    """主函数"""
    import argparse

```

```

parser = argparse.ArgumentParser(description='Auto Grading System')
parser.add_argument('image', help='Path to exam image')
parser.add_argument('--output', default='data/output', help='Output directory')
parser.add_argument('--config', default='config.py', help='Config file')

args = parser.parse_args()

# 创建系统
system = AutoGradingSystem(args.config)

# 处理图像
try:
    result = system.process(args.image, args.output)
    print("\n" + "="*50)
    print("Processing completed successfully!")
    print(f"Score: {result['grading']['total_score']}") 
    print(f"Results saved to: {args.output}")
    print("=="*50 + "\n")

except Exception as e:
    print(f"Error: {e}")
    import traceback
    traceback.print_exc()

if __name__ == '__main__':
    main()

```

4.1.2 1.2 评分模块示例（15 分钟）

```

"""
评分模块
"""

import json

class GradingModule:

```

"""评分模块"""

```
def __init__(self, config=None):
    self.config = config or {}
    self.standard_answers = self.load_standard_answers()

def load_standard_answers(self):
    """
    加载标准答案

    格式: {
        'choice': {1: 'A', 2: 'B', ...},
        'judge': {1: True, 2: False, ...},
        'essay': {1: '答案关键词', ...}
    }
    """

# 从文件加载
try:
    with open('data/templates/standard_answers.json', 'r',
              encoding='utf-8') as f:
        return json.load(f)
except:
    # 默认示例
    return {
        'choice': {1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'A'},
        'judge': {1: True, 2: False, 3: True, 4: False, 5:
                  True},
        'essay': {1: '关键词1 关键词2'}
    }

def grade_choice(self, student_answers):
    """
    评分选择题

    参数:
        student_answers: 学生答案列表

    返回:
        评分结果
    """

```

```

correct = 0
total = len(student_answers)
details = []

for ans in student_answers:
    question_num = ans['question']
    student_answer = ans['answer']
    standard_answer = self.standard_answers['choice'].get(
        question_num)

    is_correct = (student_answer == standard_answer)
    if is_correct:
        correct += 1

    details.append({
        'question': question_num,
        'student': student_answer,
        'standard': standard_answer,
        'correct': is_correct
    })

score_per_question = self.config.get('choice_score', 2)
total_score = correct * score_per_question

return {
    'type': 'choice',
    'correct': correct,
    'total': total,
    'score': total_score,
    'details': details
}

def grade_judge(self, student_answers):
    """评分判断题"""
    correct = 0
    total = len(student_answers)
    details = []

    for ans in student_answers:
        question_num = ans['question']

```

```

        student_answer = ans['answer']
        standard_answer = self.standard_answers['judge'].get(
            question_num)

        is_correct = (student_answer == standard_answer)
        if is_correct:
            correct += 1

    details.append({
        'question': question_num,
        'student': student_answer,
        'standard': standard_answer,
        'correct': is_correct
    })

    score_per_question = self.config.get('judge_score', 2)
    total_score = correct * score_per_question

    return {
        'type': 'judge',
        'correct': correct,
        'total': total,
        'score': total_score,
        'details': details
    }

def grade_essay(self, student_answers):
    """
    评分简答题
    简化实现：基于关键词匹配
    实际应用中可能需要人工复核或更复杂的NLP
    """
    results = []

    for ans in student_answers:
        question_num = ans['question']
        student_text = ans.get('text', '')
        standard_keywords = self.standard_answers['essay'].get(
            question_num, '')

```

```

# 简单的关键词匹配
keywords_found = 0
if standard_keywords:
    keywords = standard_keywords.split()
    for keyword in keywords:
        if keyword in student_text:
            keywords_found += 1

# 计算得分比例
if len(keywords) > 0:
    score_ratio = keywords_found / len(keywords)
else:
    score_ratio = 0

max_score = self.config.get('essay_score', 10)
score = int(max_score * score_ratio)

results.append({
    'question': question_num,
    'text': student_text,
    'keywords_found': keywords_found,
    'score': score
})

total_score = sum(r['score'] for r in results)

return {
    'type': 'essay',
    'total': len(results),
    'score': total_score,
    'details': results
}

def grade(self, all_answers):
    """
    综合评分
    """

    参数：
        all_answers: 所有题型答案

```

```

    返回：
        评分结果
    """
results = {}

# 选择题评分
if 'choice' in all_answers and all_answers['choice']:
    results['choice'] = self.grade_choice(all_answers['choice'])

# 判断题评分
if 'judge' in all_answers and all_answers['judge']:
    results['judge'] = self.grade_judge(all_answers['judge'])

# 简答题评分
if 'essay' in all_answers and all_answers['essay']:
    results['essay'] = self.grade_essay(all_answers['essay'])

# 计算总分
total_score = sum(
    r.get('score', 0) for r in results.values()
)

results['total_score'] = total_score
results['summary'] = self.generate_summary(results)

return results

def generate_summary(self, results):
    """生成评分摘要"""
    summary = []

    for type_name, result in results.items():
        if type_name == 'total_score' or type_name == 'summary':
            continue

        if 'correct' in result and 'total' in result:
            summary.append(
                f"{type_name.capitalize()}: {result['correct']}/{result['total']}"
            )

```

```

        result['total']]}) correct, "
        f"Score: {result['score']}"

    )
    elif 'score' in result:
        summary.append(
            f"{type_name.capitalize()}: Score: {result['score']}
                ]}"
        )

return '\n'.join(summary)

```

4.2 环节二：测试策略（30分钟）

4.2.1 2.1 测试集准备（10分钟）

测试数据准备：

```

"""
测试数据准备
"""

test_cases = [
    {
        'name': 'basic_choice',
        'image': 'data/test/test_choice_basic.jpg',
        'description': '基础选择题测试',
        'expected': {
            'choice': [
                {'question': 1, 'answer': 'A'},
                {'question': 2, 'answer': 'B'},
                {'question': 3, 'answer': 'C'}
            ]
        }
    },
    {
        'name': 'basic_judge',
        'image': 'data/test/test_judge_basic.jpg',
        'description': '基础判断题测试',
        'expected': {
            'judge': [
                {'question': 1, 'answer': True},

```

```

        {'question': 2, 'answer': False}
    ]
}
},
{
    'name': 'full_exam',
    'image': 'data/test/test_exam_full.jpg',
    'description': '完整试卷测试',
    'expected': {
        'choice': 5, # 至少识别出5道选择题
        'judge': 5, # 至少识别出5道判断题
        'essay': 1 # 至少识别出1道简答题
    }
}
]

```

4.2.2 2.2 单元测试（10 分钟）

```

"""
单元测试示例
"""

import unittest
import cv2
import sys
sys.path.append('..')

from modules.choice_recognizer import ChoiceRecognizer

class TestChoiceRecognizer(unittest.TestCase):
    """选择题识别器测试"""

    @classmethod
    def setUpClass(cls):
        """测试前准备"""
        cls.recognizer = ChoiceRecognizer({'threshold': 0.3})

    def test_calculate_density(self):
        """测试密度计算"""

```

```

# 创建测试图像：半黑半白
test_img = np.zeros((50, 50), dtype=np.uint8)
test_img[:, :25] = 255 # 左半边白色

density = self.recognizer.calculate_density(test_img)

# 应该约等于0.5
self.assertAlmostEqual(density, 0.5, places=1)

def test_recognize_single_choice(self):
    """测试单道选择题识别"""
    # 加载测试图像
    test_img = cv2.imread('data/test/single_choice.jpg', cv2.
        IMREAD_GRAYSCALE)

    # 定义选项位置
    option_positions = [(10, 10, 30, 30), (50, 10, 30, 30)]

    # 识别
    answer, densities = self.recognizer.recognize_question(
        test_img,
        option_positions
    )

    # 验证返回值
    self.assertIn(answer, ['A', 'B', 'C', 'D', None])
    self.assertEqual(len(densities), 2)

def test_recognize_all_choices(self):
    """测试多道选择题识别"""
    test_img = cv2.imread('data/test/multi_choice.jpg', cv2.
        IMREAD_GRAYSCALE)

    # 模拟选项组
    choice_groups = [
        [(10, 10, 30, 30), (50, 10, 30, 30), (90, 10, 30, 30),
         (130, 10, 30, 30)],
        [(10, 50, 30, 30), (50, 50, 30, 30), (90, 50, 30, 30),
         (130, 50, 30, 30)]
    ]

```

```

        results = self.recognizer.recognize_all(test_img,
                                              choice_groups)

        # 验证
        self.assertEqual(len(results), 2)
        for r in results:
            self.assertIn('question', r)
            self.assertIn('answer', r)

if __name__ == '__main__':
    unittest.main()

```

4.2.3 2.3 集成测试（10 分钟）

```

"""
集成测试示例
"""

import unittest
import os
from auto_grading_system import AutoGradingSystem

class TestAutoGradingSystem(unittest.TestCase):
    """系统集成测试"""

    def setUp(self):
        """每个测试前执行"""
        self.system = AutoGradingSystem()

    def test_full_pipeline(self):
        """测试完整处理流程"""
        test_image = 'data/test/test_exam_full.jpg'

        # 确保测试图像存在
        self.assertTrue(os.path.exists(test_image))

        # 处理
        result = self.system.process(test_image)

```

```

# 验证结果结构
self.assertIn('answers', result)
self.assertIn('grading', result)
self.assertIn('total_score', result['grading'])

def test_choice_only(self):
    """测试仅选择题"""
    test_image = 'data/test/test_choice_only.jpg'
    result = self.system.process(test_image)

    # 验证选择题结果
    self.assertIn('choice', result['answers'])
    self.assertIsInstance(result['answers']['choice'], list)

def test_error_handling(self):
    """测试错误处理"""
    # 测试不存在的图像
    with self.assertRaises(ValueError):
        self.system.process('nonexistent.jpg')

    # 测试损坏的图像
    # self.system.process('data/test/corrupted.jpg')

```

4.3 环节三：开发实践（90 分钟）

学生分组开发 + 教师一对一指导

4.3.1 3.1 开发检查点（30 分钟）

检查点 1：选择题模块完成

- 能够检测填涂
- 能够识别 A/B/C/D 选项
- 输出格式正确

检查点 2：判断题模块完成

- 能够检测 √/✗ 符号

- 能够区分正误
- 输出格式正确

4.3.2 3.2 集成检查点（30 分钟）

检查点 3：模块集成完成

- 主程序能调用各模块
- 数据能正确流转
- 无致命错误

检查点 4：基本功能可用

- 能处理测试图像
- 能输出评分结果
- 能生成可视化结果

4.3.3 3.3 优化检查点（30 分钟）

检查点 5：准确率优化

- 选择题识别率 >80%
- 判断题识别率 >80%
- 简答题能提取文字

4.4 环节四：常见问题解决（10 分钟）

问题清单与解决方案：

问题	解决方案
填涂识别不准	调整阈值，检查二值化效果
符号识别失败	检查轮廓检测，使用模板匹配作为后备
OCR 结果乱码	检查图像预处理，尝试调整分辨率
模块导入错误	检查 sys.path 设置，确保 __init__.py 存在
内存不足	处理大图像时先 resize，释放不用的变量

5 课后任务

5.1 开发任务

1. 完成所有模块开发
2. 完成模块集成
3. 通过基本功能测试
4. 准备演示材料

5.2 提交内容

- 可运行的系统代码
- 测试结果截图
- 遇到的问题及解决方案
- 下周演示计划

6 教学反思

6.1 进度提醒

- 本周是最后一周完整开发时间
- 下周只有展示和总结
- 务必确保系统能基本运行

6.2 下节预告

下周进行成果展示，每组 5 分钟演示 + 2 分钟答辩。