主程序部分：

```
import tkinter as tk
from tkinter import ttk, scrolledtext
import threading
import time
import cv2
import numpy as np
import pyaudio
from vosk import Model, KaldiRecognizer
import json
import requests
from PIL import Image, ImageTk
import os
from datetime import datetime
import queue
from openai import OpenAI

from src.model import CNN2, CNN3

# 全局配置
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"


# 面部识别模块
class FacialExpressionRecognition:
    def __init__(self, video_frame, emotion_queue):
        self.video_frame = video_frame
        self.emotion_queue = emotion_queue
        self.running = False
        self.thread = None
        self.cap = None
        self.model = self.load_model()
        self.border_color = (0, 0, 0)
        self.font_color = (255, 255, 255)
        self.emotion_history = []
        self.current_emotion = "Neutral"

        # 创建视频标签
        self.video_label = tk.Label(self.video_frame)
        self.video_label.pack(fill=tk.BOTH, expand=True)
        self.current_image = None

    def load_model(self):
        # 这里需要加载模型，为简化示例，我们使用占位符
```

```
            model = CNN3()
            model.load_weights(

'E:\Python_Project\PythonProject\FacialExpressionRecognition-master\FacialExpressionRecogniti
on-master\models\cnn3_best_weights.h5')
            return model

        def index2emotion(self, index):
            emotions = ["发怒", "中性", "Fear", "开心", "Sad", "Surprise", "Neutral"]
            return emotions[index] if 0 <= index < len(emotions) else "Unknown"

        def generate_faces(self, face_img, img_size=48):
            # 人脸增广处理
            face_img = face_img / 255.
            face_img = cv2.resize(face_img, (img_size, img_size), interpolation=cv2.INTER_LINEAR)
            resized_images = [face_img, face_img[2:45, :], face_img[1:47, :], cv2.flip(face_img[:, :],
1)]
            for i in range(len(resized_images)):
                resized_images[i] = cv2.resize(resized_images[i], (img_size, img_size))
                resized_images[i] = np.expand_dims(resized_images[i], axis=-1)
            return np.array(resized_images)

        def predict_expression(self):
            self.running = True
            self.cap = cv2.VideoCapture(0)
            self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
            self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
            self.cap.set(cv2.CAP_PROP_FPS, 30)

            while self.running:
                ret, frame = self.cap.read()
                if not ret:
                    continue

                # 简化版处理 - 实际应用中应使用模型
                frame = cv2.resize(frame, (800, 600))
                frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

                # 模拟检测情绪 - 实际应用中应使用真实模型
                emotion_idx = np.random.randint(0, 7)
                self.current_emotion = self.index2emotion(emotion_idx)
                self.emotion_history.append(self.current_emotion)
                self.emotion_queue.put(self.current_emotion)
```

```
                # 在图像上显示情绪
                cv2.putText(frame, self.current_emotion, (50, 50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

                # 转换为 Tkinter 可显示的格式
                img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                img = Image.fromarray(img)
                self.current_image = ImageTk.PhotoImage(image=img)

                # 在主线程中更新视频帧
                self.video_frame.after(0, self.update_video_frame)

                time.sleep(0.033)  # 约 30FPS

        self.cap.release()

    def update_video_frame(self):
        """在主线程中更新视频帧"""
        if self.current_image:
            self.video_label.configure(image=self.current_image)
            self.video_label.image = self.current_image  # 保持引用

    def start(self):
        if self.thread is None or not self.thread.is_alive():
            self.thread = threading.Thread(target=self.predict_expression, daemon=True)
            self.thread.start()

    def stop(self):
        self.running = False
        if self.thread and self.thread.is_alive():
            self.thread.join(timeout=2.0)
            self.thread = None


# 语音识别模块
class SpeechRecognition:
    def __init__(self, text_widget, speech_queue):
        self.text_widget = text_widget
        self.speech_queue = speech_queue
        self.running = False
        self.thread = None
        self.model = Model("vosk-model-small-cn-0.22")
        self.audio = pyaudio.PyAudio()
        self.stream = None
```

```
        self.last_speech = ""

    def recognize_speech(self):
        self.running = True
        FORMAT = pyaudio.paInt16
        RATE = 44100
        CHUNK = 4000
        THRESHOLDNUM = 30
        THRESHOLD = 100

        self.stream = self.audio.open(format=FORMAT, channels=1, rate=RATE,
                                      input=True, frames_per_buffer=CHUNK)
        rec = KaldiRecognizer(self.model, RATE)
        rec.SetWords(True)

        print("语音识别已启动...")

        while self.running:
            data = self.stream.read(CHUNK, exception_on_overflow=False)
            if len(data) == 0:
                continue

            if rec.AcceptWaveform(data):
                result = json.loads(rec.Result())
                if 'text' in result and result['text']:
                    speech_text = result['text'].strip()
                    if speech_text and speech_text != self.last_speech:
                        self.text_widget.insert(tk.END, f"学生: {speech_text}\n")
                        self.text_widget.see(tk.END)
                        self.speech_queue.put(speech_text)
                        self.last_speech = speech_text
            else:
                partial = json.loads(rec.PartialResult())
                if 'partial' in partial and partial['partial']:
                    pass  # 可以在这里显示部分识别结果

            time.sleep(0.1)

    def start(self):
        if self.thread is None or not self.thread.is_alive():
            self.thread = threading.Thread(target=self.recognize_speech, daemon=True)
            self.thread.start()

    def stop(self):
```

```
            self.running = False
            if self.thread and self.thread.is_alive():
                self.thread.join(timeout=2.0)
                self.thread = None
            if self.stream:
                self.stream.stop_stream()
                self.stream.close()
            self.audio.terminate()


# DeepSeek 接口模块
class DeepSeekInterface:
    def __init__(self, chat_widget, api_key):
            self.chat_widget = chat_widget
            self.api_key = api_key
            self.client = OpenAI(
                api_key=self.api_key,
                base_url="https://ark.cn-beijing.volces.com/api/v3"
            )
            self.conversation_history = [
                {"role": "system",
                 "content": "你是一位专业的 AI 助教，需要帮助教师分析学生的学习状态并提供教学建议。请用简洁明了的语言回答。"}
            ]

    def send_query(self, query):
            if not query.strip():
                return

            # 显示用户问题
            self.chat_widget.insert(tk.END, f"教师: {query}\n", "user")
            self.chat_widget.see(tk.END)

            # 添加到历史记录
            self.conversation_history.append({"role": "user", "content": query})

            # 创建并显示"思考中..."消息
            thinking_tag = "thinking"
            self.chat_widget.insert(tk.END, "AI 助手: ", "assistant")
            self.chat_widget.insert(tk.END, "思考中...", thinking_tag)
            self.chat_widget.see(tk.END)
            self.chat_widget.update_idletasks()

            try:
```

```
                # 创建流式响应
                stream = self.client.chat.completions.create(
                    model="deepseek-r1-250528",  # 使用火山引擎上的模型 ID
                    messages=self.conversation_history,
                    stream=True
                )

                # 删除"思考中..."消息
                self.chat_widget.delete("end-2l", "end-1c")

                # 显示 AI 响应
                full_response = ""
                self.chat_widget.insert(tk.END, "AI 助手: ", "assistant")

                # 处理流式响应
                for chunk in stream:
                    if chunk.choices and chunk.choices[0].delta and
chunk.choices[0].delta.content:
                        content = chunk.choices[0].delta.content
                        full_response += content
                        self.chat_widget.insert(tk.END, content, "assistant")
                        self.chat_widget.see(tk.END)
                        self.chat_widget.update_idletasks()

                self.chat_widget.insert(tk.END, "\n\n", "assistant")
                self.conversation_history.append({"role": "assistant", "content": full_response})

            except Exception as e:
                self.chat_widget.delete("end-2l", "end-1c")  # 删除"思考中..."消息
                self.chat_widget.insert(tk.END, f"AI 助手: 请求发生错误: {str(e)}\n\n", "error")

# 学习状态评估模块
# 学习状态评估模块
class LearningAssessment:
    def __init__(self, assessment_frame, emotion_queue, speech_queue):
        self.assessment_frame = assessment_frame
        self.emotion_queue = emotion_queue
        self.speech_queue = speech_queue
        self.running = False
        self.thread = None
        self.emotion_history = []
        self.speech_history = []
        self.current_assessment = "等待数据..."
```

```
# 创建评估显示组件
self.assessment_label = tk.Label(
    assessment_frame,
    text=self.current_assessment,
    font=("Arial", 14, "bold"),
    wraplength=300,
    justify="center",
    bg="#F0F8FF",
    padx=20,
    pady=20
)
self.assessment_label.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# 创建情绪图表标签
self.emotion_chart_label = tk.Label(
    assessment_frame,
    text="情绪变化趋势:",
    font=("Arial", 10),
    anchor="w"
)
self.emotion_chart_label.pack(fill=tk.X, padx=10, pady=(10, 0))

# 创建情绪图表（简化版）
self.emotion_chart = tk.Canvas(assessment_frame, height=150, bg="white")
self.emotion_chart.pack(fill=tk.BOTH, expand=True, padx=10, pady=(0, 10))

# 创建关键词标签
self.keywords_label = tk.Label(
    assessment_frame,
    text="关键词分析:",
    font=("Arial", 10),
    anchor="w"
)
self.keywords_label.pack(fill=tk.X, padx=10, pady=(5, 0))

# 创建关键词显示区域
self.keywords_text = tk.Text(
    assessment_frame,
    height=4,
    font=("Arial", 9),
    wrap=tk.WORD
)
self.keywords_text.pack(fill=tk.X, padx=10, pady=(0, 10))
self.keywords_text.insert(tk.END, "等待语音识别数据...")
```

```
        self.keywords_text.config(state=tk.DISABLED)

        # 初始化模拟数据
        self.emotion_history = ["中性"] * 10  # 初始 10 个中性情绪点
        self.speech_history = ["准备开始学习"]  # 初始语音记录

        # 更新图表和评估
        self.update_emotion_chart()
        self.current_assessment = self.analyze_learning_state()
        self.assessment_label.config(text=self.current_assessment)

        # 更新关键词显示
        keywords = self.extract_keywords()
        self.keywords_text.config(state=tk.NORMAL)
        self.keywords_text.delete(1.0, tk.END)
        self.keywords_text.insert(tk.END, keywords)
        self.keywords_text.config(state=tk.DISABLED)

        # 在初始化后立即更新一次图表（延迟 100ms 确保 UI 渲染）
        self.emotion_chart.after(100, self.update_emotion_chart)

    def analyze_learning_state(self):
        """分析学习状态（简化版实现）"""
        # 情绪分类
        positive_emotions = ["开心", "惊讶"]
        negative_emotions = ["发怒", "悲伤", "恐惧"]

        # 计算情绪分布
        emotion_counts = {emotion: self.emotion_history.count(emotion) for emotion in
set(self.emotion_history)}

        # 计算积极情绪比例
        positive_count = sum(emotion_counts.get(emotion, 0) for emotion in
positive_emotions)
        negative_count = sum(emotion_counts.get(emotion, 0) for emotion in
negative_emotions)
        neutral_count = emotion_counts.get("中性", 0)

        total = len(self.emotion_history)

        # 根据情绪分布生成评估
        if positive_count / total > 0.5:
            return "学习状态：积极专注\n 学生表现出浓厚兴趣"
        elif negative_count / total > 0.4:
```

```
            return "学习状态：困惑分心\n 学生可能需要更多指导"
        elif "不懂" in " ".join(self.speech_history) or "不明白" in " ".join(self.speech_history):
            return "学习状态：遇到困难\n 学生明确表达了困惑"
        elif neutral_count / total > 0.7:
            return "学习状态：一般稳定\n 学生处于正常学习状态"
        else:
            return "学习状态：观察中\n 继续收集更多数据"

    def extract_keywords(self):
        """提取关键词（简化版实现）"""
        if not self.speech_history:
            return "等待语音识别数据..."

        # 常见教育关键词
        edu_keywords = ["老师", "问题", "不懂", "明白", "为什么", "学习", "理解", "例子", "
讲解"]

        # 找出出现频率高的关键词
        word_counts = {}
        for speech in self.speech_history:
            for word in speech.split():
                if word in edu_keywords:
                    word_counts[word] = word_counts.get(word, 0) + 1

        # 按频率排序
        sorted_words = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)

        # 返回前 3 个关键词
        if sorted_words:
            return ", ".join(f"{word}({count})" for word, count in sorted_words[:3])
        return "未检测到明显关键词"

    def update_emotion_chart(self):
        # 获取画布尺寸
        width = self.emotion_chart.winfo_width()
        height = self.emotion_chart.winfo_height()

        # 如果画布太小，使用默认尺寸
        if width < 50 or height < 50:
            width = 300
            height = 150

        # 清空画布
        self.emotion_chart.delete("all")
```

```
# 如果没有情绪数据，显示提示信息
if not self.emotion_history:
    self.emotion_chart.create_text(width / 2, height / 2,
                                   text="等待情绪数据...",
                                   fill="gray",
                                   font=("Arial", 10))
    return

# 添加图表标题和说明
self.emotion_chart.create_text(width / 2, 15,
                               text="情绪变化趋势 (最近 20 个点)",
                               fill="#333333",
                               font=("Arial", 9, "bold"))

# 添加情绪高度说明
self.emotion_chart.create_text(width - 50, height - 15,
                               text="↑ 积极 ↓ 消极",
                               fill="#666666",
                               font=("Arial", 8))

# 情绪高度映射 (修正后的值)
emotion_heights = {
    "开心": 0.1 * height,  # 最上方 - 积极
    "惊讶": 0.2 * height,
    "中性": 0.4 * height,  # 中间
    "厌恶": 0.6 * height,
    "恐惧": 0.7 * height,
    "悲伤": 0.8 * height,
    "发怒": 0.9 * height   # 最下方 - 消极
}

# 情绪颜色映射 (中文键)
emotion_colors = {
    "开心": "#FFD700",  # 金色
    "中性": "#1E90FF",  # 道奇蓝
    "惊讶": "#FF6347",  # 番茄红
    "悲伤": "#4682B4",  # 钢蓝
    "发怒": "#DC143C",  # 猩红
    "恐惧": "#9370DB",  # 中紫
    "厌恶": "#32CD32"   # 酸橙绿
}

# 计算 x 轴步长 (考虑左右边距)
```

```
        margin = 20
        available_width = width - 2 * margin
        x_step = available_width / (len(self.emotion_history) - 1) if len(self.emotion_history) > 1 else available_width

        # 绘制坐标轴
        self.emotion_chart.create_line(margin, margin, margin, height - margin, fill="#888888", width=1)  # y 轴
        self.emotion_chart.create_line(margin, height - margin, width - margin, height - margin, fill="#888888",
                                       width=1)  # x 轴

        # 绘制情绪点并连接
        points = []
        for i, emotion in enumerate(self.emotion_history):
            x = margin + i * x_step
            y = emotion_heights.get(emotion, height / 2)  # 默认在中间
            color = emotion_colors.get(emotion, "#000000")

            points.append((x, y))

            # 绘制点
            point_size = 6
            self.emotion_chart.create_oval(
                x - point_size, y - point_size,
                x + point_size, y + point_size,
                fill=color, outline="black", width=1
            )

            # 添加情绪标签
            if i == 0 or i == len(self.emotion_history) - 1:
                self.emotion_chart.create_text(
                    x, y - 15,
                    text=emotion[:2],  # 只显示前两个字符
                    fill=color,
                    font=("Arial", 8)
                )

        # 连接点形成趋势线
        if len(points) > 1:
            for i in range(1, len(points)):
                x1, y1 = points[i - 1]
                x2, y2 = points[i]
                self.emotion_chart.create_line(x1, y1, x2, y2, fill="#333333", width=2,
```

```
smooth=True)

            # 强制更新画布显示
            self.emotion_chart.update_idletasks()

        def update_assessment(self):
            self.running = True

            # 初始延迟确保 UI 已渲染
            time.sleep(0.5)

            while self.running:
                # 处理情绪队列
                while not self.emotion_queue.empty():
                    emotion = self.emotion_queue.get()
                    self.emotion_history.append(emotion)
                    self.emotion_history = self.emotion_history[-20:]  # 保留最近 20 个

                # 处理语音队列
                while not self.speech_queue.empty():
                    speech = self.speech_queue.get()
                    self.speech_history.append(speech)
                    self.speech_history = self.speech_history[-10:]  # 保留最近 10 条

                    # 更新关键词显示
                    keywords = self.extract_keywords()
                    self.keywords_text.config(state=tk.NORMAL)
                    self.keywords_text.delete(1.0, tk.END)
                    self.keywords_text.insert(tk.END, keywords)
                    self.keywords_text.config(state=tk.DISABLED)

                # 更新评估结果
                self.current_assessment = self.analyze_learning_state()
                self.assessment_label.config(text=self.current_assessment)

                # 更新情绪图表
                self.emotion_chart.update_idletasks()  # 确保 UI 更新
                self.update_emotion_chart()

                time.sleep(1)

# 主应用界面
class AIStudyMonitorApp:
    def __init__(self, root):
```

```
self.root = root
self.root.title("多模态 AI 学习状态监测系统")
self.root.geometry("1200x800")
self.root.configure(bg="#F5F5F5")

# 设置应用图标
try:
    self.root.iconbitmap("brain_icon.ico")  # 如果有图标文件可以使用
except:
    pass

# 创建标题
title_label = tk.Label(
    root,
    text="面向因材施教的 AI 学习状态实时监测系统",
    font=("Arial", 18, "bold"),
    bg="#4B0082",
    fg="white",
    padx=20,
    pady=15
)
title_label.pack(fill=tk.X)

# 创建主框架
main_frame = ttk.Frame(root)
main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# 创建左侧面板（摄像头和语音识别）
left_panel = ttk.Frame(main_frame)
left_panel.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# 创建摄像头面板
camera_frame = ttk.LabelFrame(left_panel, text="实时面部表情识别")
camera_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

# 创建语音识别面板
speech_frame = ttk.LabelFrame(left_panel, text="实时语音识别")
speech_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

# 创建右侧面板（聊天和评估）
right_panel = ttk.Frame(main_frame)
right_panel.pack(side=tk.RIGHT, fill=tk.BOTH, expand=False, padx=5, pady=5)
right_panel.config(width=400)
```

```
# 创建学习状态评估面板
assessment_frame = ttk.LabelFrame(right_panel, text="学习状态评估")
assessment_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

# 创建 DeepSeek 聊天面板
chat_frame = ttk.LabelFrame(right_panel, text="AI 教学助手")
chat_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

# 创建聊天输入框和按钮
input_frame = ttk.Frame(chat_frame)
input_frame.pack(fill=tk.X, padx=5, pady=5)

self.chat_input = ttk.Entry(input_frame)
self.chat_input.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0, 5))

send_button = ttk.Button(input_frame, text="发送", command=self.send_message)
send_button.pack(side=tk.RIGHT)

# 创建聊天显示区域
self.chat_display = scrolledtext.ScrolledText(
    chat_frame,
    wrap=tk.WORD,
    font=("Arial", 10),
    padx=10,
    pady=10
)
self.chat_display.pack(fill=tk.BOTH, expand=True, padx=5, pady=(0, 5))
self.chat_display.tag_config("user", foreground="blue")
self.chat_display.tag_config("assistant", foreground="green")
self.chat_display.tag_config("error", foreground="red")
self.chat_display.insert(tk.END, "系统: AI 教学助手已就绪，请输入您的问题...\n\n")

# 创建语音识别显示区域
self.speech_display = scrolledtext.ScrolledText(
    speech_frame,
    wrap=tk.WORD,
    font=("Arial", 10),
    padx=10,
    pady=10
)
self.speech_display.pack(fill=tk.BOTH, expand=True)
self.speech_display.insert(tk.END, "系统: 语音识别已启动，正在监听...\n\n")

# 创建控制按钮面板
```

```
        control_frame = ttk.Frame(root)
        control_frame.pack(fill=tk.X, padx=10, pady=5)

        start_button = ttk.Button(control_frame, text="开始监测",
command=self.start_monitoring)
        start_button.pack(side=tk.LEFT, padx=5)

        stop_button = ttk.Button(control_frame, text="停止监测",
command=self.stop_monitoring)
        stop_button.pack(side=tk.LEFT, padx=5)

        # 创建状态栏
        self.status_var = tk.StringVar()
        self.status_var.set("系统就绪")
        status_bar = ttk.Label(root, textvariable=self.status_var, relief=tk.SUNKEN,
anchor=tk.W)
        status_bar.pack(side=tk.BOTTOM, fill=tk.X)

        # 创建队列
        self.emotion_queue = queue.Queue()
        self.speech_queue = queue.Queue()

        # 初始化模块
        self.face_recognition = FacialExpressionRecognition(camera_frame,
self.emotion_queue)
        self.speech_recognition = SpeechRecognition(self.speech_display, self.speech_queue)
        # self.deepseek_interface = DeepSeekInterface(self.chat_display,
        #
"sk-fcc33694c5614a6d83c55205f78dd824")
        self.deepseek_interface = DeepSeekInterface(self.chat_display,

"b0557086-5861-4de9-87eb-12f462e073b1")
        self.learning_assessment = LearningAssessment(assessment_frame,
self.emotion_queue, self.speech_queue)

        # 绑定回车键发送消息
        self.root.bind('<Return>', lambda event: self.send_message())

        # 设置窗口关闭事件
        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

    def start_monitoring(self):
        self.status_var.set("开始监测学生学习状态...")
        self.face_recognition.start()
```

```
            self.speech_recognition.start()
            self.learning_assessment.start()
            self.speech_display.insert(tk.END, "系统: 开始监测学生学习状态...\n\n")

        def stop_monitoring(self):
            self.status_var.set("停止监测")
            self.face_recognition.stop()
            self.speech_recognition.stop()
            self.learning_assessment.stop()
            self.speech_display.insert(tk.END, "系统: 监测已停止\n\n")

        def send_message(self):
            message = self.chat_input.get()
            self.chat_input.delete(0, tk.END)
            if message:
                threading.Thread(target=self.deepseek_interface.send_query,      args=(message,),
daemon=True).start()

        def on_closing(self):
            self.stop_monitoring()
            self.root.destroy()


if __name__ == "__main__":
        root = tk.Tk()
        app = AIStudyMonitorApp(root)
        root.mainloop()
```

面部识别部分：

```
"""
author: Zhou Chen
datetime: 2019/6/20 15:44
desc: 利用摄像头实时检测
"""
import os
import argparse
import time    # 添加时间模块用于延时
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import cv2
import numpy as np
```

```
from model import CNN2, CNN3
from utils import index2emotion, cv2_img_add_text
from blazeface import blaze_detect

parser = argparse.ArgumentParser()
parser.add_argument("--source", type=int, default=0, help="data source, 0 for camera 1 for video")
parser.add_argument("--video_path", type=str, default=None)
opt = parser.parse_args()

if opt.source == 1 and opt.video_path is not None:
    filename = opt.video_path
else:
    filename = None


def load_model():
    """
    加载本地模型
    :return:
    """
    model = CNN3()

model.load_weights('E:\Python_Project\PythonProject\FacialExpressionRecognition-master\FacialExpressionRecognition-master\models\cnn3_best_weights.h5')
    return model


def generate_faces(face_img, img_size=48):
    """
    将探测到的人脸进行增广
    :param face_img:  灰度化的单个人脸图
    :param img_size:  目标图片大小
    :return:
    """

    face_img = face_img / 255.
    face_img = cv2.resize(face_img, (img_size, img_size), interpolation=cv2.INTER_LINEAR)
    resized_images = list()
    resized_images.append(face_img)
    resized_images.append(face_img[2:45, :])
    resized_images.append(face_img[1:47, :])
    resized_images.append(cv2.flip(face_img[:, :], 1))
```

```
    for i in range(len(resized_images)):
        resized_images[i] = cv2.resize(resized_images[i], (img_size, img_size))
        resized_images[i] = np.expand_dims(resized_images[i], axis=-1)
    resized_images = np.array(resized_images)
    return resized_images


def predict_expression():
    """
    实时预测
    :return:
    """
    # 参数设置
    model = load_model()

    border_color = (0, 0, 0)  # 黑框框
    font_color = (255, 255, 255)  # 白字字

    # 初始化摄像头/视频
    if filename:
        capture = cv2.VideoCapture(filename)
    else:
        capture = cv2.VideoCapture(0)  # 指定 0 号摄像头
        # 设置摄像头参数降低解码压力 - 关键修复 1
        capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        capture.set(cv2.CAP_PROP_FPS, 30)

    while True:
        # 读取帧并检查有效性 - 关键修复 2
        retry_count = 0
        ret = False
        frame = None

        # 最多重试 5 次获取有效帧
        while retry_count < 5 and not ret:
            ret, frame = capture.read()
            if not ret or frame is None:
                retry_count += 1
                time.sleep(0.05)  # 短暂延迟后重试
            else:
                break

        # 如果连续多次获取失败
```

```
        if not ret or frame is None:
            print("错误：连续多次获取帧失败，退出程序")
            break

        try:
            # 调整大小并转换颜色空间
            frame = cv2.resize(frame, (800, 600))
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)  # 修正：使用 RGB2GRAY

            # 人脸检测
            faces = blaze_detect(frame)

            # 如果检测到人脸
            if faces is not None and len(faces) > 0:
                for (x, y, w, h) in faces:
                    # 确保人脸区域在图像范围内
                    if y >= 0 and x >= 0 and y + h <= frame_gray.shape[0] and x + w <= frame_gray.shape[1]:
                        face = frame_gray[y: y + h, x: x + w]  # 脸部图片

                        # 确保人脸区域有效
                        if face.size > 0:
                            faces_aug = generate_faces(face)
                            results = model.predict(faces_aug)
                            result_sum = np.sum(results, axis=0).reshape(-1)
                            label_index = np.argmax(result_sum, axis=0)
                            emotion = index2emotion(label_index)
                            cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), border_color, thickness=2)
                            frame = cv2_img_add_text(frame, emotion, x + 30, y + 30, font_color, 20)
        except Exception as e:
            print(f"处理过程中发生错误: {e}")
            continue

        # 显示结果
        cv2.imshow("expression recognition(press esc to exit)", cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

        key = cv2.waitKey(30)  # 等待 30ms，返回 ASCII 码

        # 如果输入 esc 则退出循环
```

```
            if key == 27:
                break

        # 释放资源
        capture.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
        predict_expression()


"""
author: Zhou Chen
datetime: 2019/6/18 17:06
desc: 构建 CNN 模型
"""
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout, BatchNormalization, Flatten, Dense, AveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.layers import PReLU


def CNN1(input_shape=(48, 48, 1), n_classes=8):
        """
        参考 VGG 思路设计的第一个模型，主要注意点是感受野不能太大，以免获得很多噪声信息
        :param input_shape: 输入图片的尺寸
        :param n_classes: 目标类别数目
        :return:
        """
        # input
        input_layer = Input(shape=input_shape)
        # block1
        x = Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(input_layer)
        x = Conv2D(32, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(x)
        x = Dropout(0.5)(x)
        # block2
        x = Conv2D(64, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(x)
        x = Conv2D(64, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(x)
        x = Dropout(0.5)(x)
        # block3
```

```
    x = Conv2D(128, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(x)
    x = Conv2D(128, kernel_size=(3, 3), strides=1, padding='same', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(x)
    x = Dropout(0.5)(x)
    # fc
    x = Flatten()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(128, activation='relu')(x)
    output_layer = Dense(n_classes, activation='softmax')(x)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model


def CNN2(input_shape=(48, 48, 1), n_classes=8):
    """
    参考论文 Going deeper with convolutions 在输入层后加一层的 1*1 卷积增加非线性表示

    :param input_shape:
    :param n_classes:
    :return:
    """
    # input
    input_layer = Input(shape=input_shape)
    # block1
    x = Conv2D(32, (1, 1), strides=1, padding='same', activation='relu')(input_layer)
    x = Conv2D(32, (5, 5), strides=1, padding='same', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)
    # block2
    x = Conv2D(32, (3, 3), padding='same', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)
    # block3
    x = Conv2D(64, (5, 5), padding='same', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)
    # fc
    x = Flatten()(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(n_classes, activation='softmax')(x)

    model = Model(inputs=input_layer, outputs=x)
```

```
    return model


def CNN3(input_shape=(48, 48, 1), n_classes=8):
    """
    参考论文实现
    A Compact Deep Learning Model for Robust Facial Expression Recognition
    :param input_shape:
    :param n_classes:
    :return:
    """
    # input
    input_layer = Input(shape=input_shape)
    x = Conv2D(32, (1, 1), strides=1, padding='same', activation='relu')(input_layer)
    # block1
    x = Conv2D(64, (3, 3), strides=1, padding='same')(x)
    x = PReLU()(x)
    x = Conv2D(64, (5, 5), strides=1, padding='same')(x)
    x = PReLU()(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)
    # block2
    x = Conv2D(64, (3, 3), strides=1, padding='same')(x)
    x = PReLU()(x)
    x = Conv2D(64, (5, 5), strides=1, padding='same')(x)
    x = PReLU()(x)
    x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)
    # fc
    x = Flatten()(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(n_classes, activation='softmax')(x)

    model = Model(inputs=input_layer, outputs=x)
    return model


"""
训练脚本，由于数据集不大，这里一次性读入内存
"""
import os
import argparse
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
```

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.model_selection import train_test_split

from data import Fer2013, Jaffe, CK
from model import CNN1, CNN2, CNN3
from visualize import plot_loss, plot_acc

parser = argparse.ArgumentParser()
parser.add_argument("--dataset", type=str, default="fer2013", help="dataset to train, fer2013 or
jaffe or ck+")
parser.add_argument("--epochs", type=int, default=200)
parser.add_argument("--batch_size", type=int, default=32)
parser.add_argument("--plot_history", type=bool, default=True)
opt = parser.parse_args()
his = None
print(opt)

if opt.dataset == "fer2013":
    expressions, x_train, y_train = Fer2013().gen_train()
    _, x_valid, y_valid = Fer2013().gen_valid()
    _, x_test, y_test = Fer2013().gen_test()
    # target 编码
    y_train = to_categorical(y_train).reshape(y_train.shape[0], -1)
    y_valid = to_categorical(y_valid).reshape(y_valid.shape[0], -1)
    # 为了统一几个数据集，必须增加一列为 0 的
    y_train = np.hstack((y_train, np.zeros((y_train.shape[0], 1))))
    y_valid = np.hstack((y_valid, np.zeros((y_valid.shape[0], 1))))
    print("load fer2013 dataset successfully, it has {} train images and {} valid
iamges".format(y_train.shape[0], y_valid.shape[0]))

    model = CNN3(input_shape=(48, 48, 1), n_classes=8)
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
    callback = [
        #       EarlyStopping(monitor='val_loss', patience=50, verbose=True),
        #       ReduceLROnPlateau(monitor='lr', factor=0.1, patience=20, verbose=True),
        ModelCheckpoint('./models/cnn2_best_weights.h5', monitor='val_acc', verbose=True,
save_best_only=True,
                                save_weights_only=True)]
```

```
    train_generator = ImageDataGenerator(rotation_range=10,
                                         width_shift_range=0.05,
                                         height_shift_range=0.05,
                                         horizontal_flip=True,
                                         shear_range=0.2,
                                         zoom_range=0.2).flow(x_train, y_train,
batch_size=opt.batch_size)
    valid_generator = ImageDataGenerator().flow(x_valid, y_valid, batch_size=opt.batch_size)
    history_fer2013 = model.fit_generator(train_generator,
                                          steps_per_epoch=len(y_train) //
opt.batch_size,
                                          epochs=opt.epochs,
                                          validation_data=valid_generator,
                                          validation_steps=len(y_valid) //
opt.batch_size,
                                          callbacks=callback)
    his = history_fer2013

    # test
    pred = model.predict(x_test)
    pred = np.argmax(pred, axis=1)
    print("test accuacy", np.sum(pred.reshape(-1) == y_test.reshape(-1)) / y_test.shape[0])

elif opt.dataset == "jaffe":
    expressions, x, y = Jaffe().gen_train()
    y = to_categorical(y).reshape(y.shape[0], -1)
    # 为了统一几个数据集，必须增加一列为 0 的
    y = np.hstack((y, np.zeros((y.shape[0], 1))))

    # 划分训练集验证集
    x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2, random_state=2019)
    print("load jaffe dataset successfully, it has {} train images and {} valid
iamges".format(y_train.shape[0],

y_valid.shape[0]))
    train_generator = ImageDataGenerator(rotation_range=5,
                                         width_shift_range=0.01,
                                         height_shift_range=0.01,
                                         horizontal_flip=True,
                                         shear_range=0.1,
                                         zoom_range=0.1).flow(x_train, y_train,
batch_size=opt.batch_size)
    valid_generator = ImageDataGenerator().flow(x_valid, y_valid, batch_size=opt.batch_size)
```

```
    model = CNN3()

    sgd = Adam(lr=0.0001)
    model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
    callback = [
        # EarlyStopping(monitor='val_loss', patience=50, verbose=True),
        # ReduceLROnPlateau(monitor='lr', factor=0.1, patience=15, verbose=True),
        ModelCheckpoint('./models/cnn3_best_weights.h5', monitor='val_accuracy',
verbose=True, save_best_only=True,
                        save_weights_only=True)]
    history_jaffe = model.fit(train_generator, steps_per_epoch=len(y_train) // opt.batch_size,
epochs=opt.epochs,
                              validation_data=valid_generator,
validation_steps=len(y_valid) // opt.batch_size,
                              callbacks=callback)
    his = history_jaffe
else:
    expr, x, y = CK().gen_train()
    y = to_categorical(y).reshape(y.shape[0], -1)
    # 划分训练集验证集
    x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2, random_state=2019)
    print("load CK+ dataset successfully, it has {} train images and {} valid
iamges".format(y_train.shape[0],

y_valid.shape[0]))
    train_generator = ImageDataGenerator(rotation_range=10,
                                         width_shift_range=0.05,
                                         height_shift_range=0.05,
                                         horizontal_flip=True,
                                         shear_range=0.2,
                                         zoom_range=0.2).flow(x_train, y_train,
batch_size=opt.batch_size)
    valid_generator = ImageDataGenerator().flow(x_valid, y_valid, batch_size=opt.batch_size)
    model = CNN3()
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
    callback = [
        #       EarlyStopping(monitor='val_loss', patience=50, verbose=True),
        #       ReduceLROnPlateau(monitor='lr', factor=0.1, patience=15, verbose=True),
        ModelCheckpoint('./models/cnn3_best_weights.h5', monitor='val_acc', verbose=True,
save_best_only=True,
                        save_weights_only=True)]
    history_ck = model.fit_generator(train_generator, steps_per_epoch=len(y_train) //
```

```
opt.batch_size, epochs=opt.epochs,
                                                validation_data=valid_generator,
validation_steps=len(y_valid) // opt.batch_size,
                                                callbacks=callback)
    his = history_ck

if opt.plot_history:
    plot_loss(his.history, opt.dataset)
    plot_acc(his.history, opt.dataset)


"""
author: Zhou Chen
datetime: 2019/7/1 17:10
desc: 用于图片预处理，神经网络没有用到这部分
"""
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
# import dlib


def add_noise(input_data):
    """
    增加噪声干扰
    :param input_data:
    :return:
    """
    for i in range(5000):
        x = np.random.randint(0, input_data.shape[0])
        y = np.random.randint(0, input_data.shape[1])
        input_data[x][y][:] = 255
    return input_data


def histogram_equalization(img):
    """
    直方图均衡化
    :param img:
    :return:
    """
    ycrcb = cv.cvtColor(img, cv.COLOR_BGR2YCR_CB)
    channels = cv.split(ycrcb)
    cv.equalizeHist(channels[0], channels[0])
```

```
    cv.merge(channels, ycrcb)
    cv.cvtColor(ycrcb, cv.COLOR_YCR_CB2BGR, img)
    return img


def adaptive_histogram_equalization(img):
    """
    自适应直方图均衡化
    :param img:
    :return:
    """
    ycrcb = img
    channels = cv.split(ycrcb)
    # create a CLAHE object (Arguments are optional).
    clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    channels[0] = clahe.apply(channels[0])
    cv.merge(channels, ycrcb)
    img = ycrcb
    return img


def detection(img):
    """
    人脸检测
    :param img:
    :return:
    """
    detector = dlib.get_frontal_face_detector()
    dets = detector(img, 1)   # 使用 detector 进行人脸检测  dets 为返回的结果
    print("Number of faces detected: {}".format(len(dets)))   # 打印识别到的人脸个数
    for index, face in enumerate(dets):
        # 在图片中标注人脸，并显示
        left = face.left()
        top = face.top()
        right = face.right()
        bottom = face.bottom()
        cv.rectangle(img, (left, top), (right, bottom), (0, 255, 0), 1)
    return dets


def predictor(img, dets):
    """
    特征点标定
    :param img:
```

```
    :param dets:
    :return:
    """
    # shape_predictor_68_face_landmarks.dat 是进行人脸标定的模型，它是基于 HOG 特征的，这里是他所在的路径
    predictor_path = "../model/shape_predictor_68_face_landmarks.dat"
    predictor = dlib.shape_predictor(predictor_path)
    shape_list = []
    for index, face in enumerate(dets):
        shape = predictor(img, face)    # 寻找人脸的 68 个标定点
        # 遍历所有点，打印出其坐标，并用绿色的圈表示出来
        for _, pt in enumerate(shape.parts()):
            pt_pos = (pt.x, pt.y)
            cv.circle(img, pt_pos, 1, (0, 255, 0), 1)
        shape_list.append(shape)
    return shape_list


def gray_norm(img):
    """
    灰度归一化
    :param img:
    :return:
    """
    min_value = np.min(img)
    max_value = np.max(img)
    if max_value == min_value:
        return img
    (n, m) = img.shape
    for i in range(n):
        for j in range(m):
            img[i, j] = int(255 * (img[i][j] - min_value) / (max_value - min_value))
    return img


def normailiztaion(img, dets, shape_list):
    """
    图像尺度灰度归一化
    :param img:
    :param dets:
    :param shape_list:
    :return:
    """
    # 灰度归一化
```

```
    img = gray_norm(img)

    # 尺度归一化
    img_list = []
    pt_pos_list = []
    for index, face in enumerate(dets):
        left = face.left()
        top = face.top()
        right = face.right()
        bottom = face.bottom()
        img1 = img[top:bottom, left:right]
        size = (48, 48)
        img1 = cv.resize(img1, size, interpolation=cv.INTER_LINEAR)

        pos = []
        for _, pt in enumerate(shape_list[index].parts()):
            pt_pos = (int((pt.x - left) / (right - left) * 90), int((pt.y - top) / (bottom - top) * 100))
            pos.append(pt_pos)
            cv.circle(img1, pt_pos, 2, (255, 0, 0), 1)
        pt_pos_list.append(pos)
        img_list.append(img1)
    return img_list, pt_pos_list


# 图片预处理
def deal(img):
    """
    :param img:
    :return:
    img   原图框定后的图片
    dets  人脸框定信息
    shape  特征点在原图的位置
    img_list  框取的图片
    pt_pos_list  每张图片特征点的位置
    """
    # 滤波去噪
    img = cv.blur(img, (5, 5))
    # 人脸框定
    dets = detection(img)
    # 特征点标定
    shape_list = predictor(img, dets)
    # 直方图均衡化
    adaptive_histogram_equalization(img)
    # 尺度灰度归一化
```

```
    img_list, pt_pos_list = normailiztaion(img, dets, shape_list)
    return img, dets, shape_list, img_list, pt_pos_list


def test():
    lena = cv.cvtColor(cv.imread('zhangyu.jpg', flags=1), cv.COLOR_BGR2RGB)
    print(lena.shape)

    lena.flags.writeable = True

    plt.suptitle('preprocess')
    plt.subplots_adjust(wspace=0.3, hspace=0.3)

    # 原图
    plt.subplot(321)
    plt.imshow(lena)
    plt.title('origin_image')
    plt.axis('off')

    # 添加噪声后的图片
    noise_image = add_noise(lena)
    plt.subplot(322)
    plt.imshow(noise_image)
    plt.title('noise_image')
    plt.axis('off')

    # 均值滤波后的图片
    blur_image = cv.blur(noise_image, (5, 5))
    plt.subplot(323)
    plt.imshow(blur_image)
    plt.title('AF_image')
    plt.axis('off')

    # 中值滤波后的图片
    median_blur_image = cv.medianBlur(noise_image, 5)
    plt.subplot(324)
    plt.imshow(median_blur_image)
    plt.title('MF_image')
    plt.axis('off')

    # 自适应中值滤波后的图片
    # adaptive_median_blur_image = cv.ad

    # 直方图均衡化
```

```
    plt.subplot(325)
    plt.imshow(histogram_equalization(median_blur_image))
    plt.title('equalization_image')
    plt.axis('off')

    # 直方图均衡化
    plt.subplot(326)
    plt.imshow(adaptive_histogram_equalization(median_blur_image))
    plt.title('adaptive_equalization_image')
    plt.axis('off')
    plt.show()


if __name__ == '__main__':
    test()

"""
author: Zhou Chen
datetime: 2019/7/1 20:24
desc: 使用 LBP 的实现
"""


import preprocess
import sys
from Gabor import *
import os
import numpy as np
import cv2
from skimage import feature as skif
from sklearn import svm


class LBP(object):
    # 使用 LBP+SVM 实现表情识别

    def load_image(self, foler):
        """
        根据给定的目录，读取当前目录下的所有图片
        :param foler:
        :return:
        """
        categories = os.listdir(foler)    # 得到当前 foler 文件夹下所有的目录
        imags = []
```

```
        labels = []
        for category in categories:
            now_folder = os.path.join(foler, category)
            subsdirectories = os.listdir(now_folder)
            for subsdirectory in subsdirectories:
                now_path = os.path.join(now_folder, subsdirectory)
                image = cv2.imread(now_path)
                grab = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

                imags.append(self.get_lbp(grab))
                labels.append(category)
        images_array = np.array(imags)
        return images_array, labels

    def get_lbp(self, image):
        """
        获取给定图片的 LBP，划分成几个区域后
        :param rgb:
        :return:
        """
        gridx = 6
        gridy = 6
        widx = 8
        widy = 8
        hists = []
        for i in range(gridx):
            for j in range(gridy):
                mat = image[i * widx: (i + 1) * widx, j * widy: (j + 1) * widy]
                lbp = skif.local_binary_pattern(mat, 8, 1, 'uniform')
                max_bins = 10
                hist, _ = np.histogram(lbp, normed=True, bins=max_bins, range=(0,
max_bins))
                hists.append(hist)
        out = np.array(hists).reshape(-1, 1)
        return out

    def label2number(self, label_list):
        label = np.zeros(len(label_list))
        label_unique = np.unique(label_list)
        num = label_unique.shape[0]
        for k in range(num):
            index = [i for i, v in enumerate(label_list) if v == label_unique[k]]
            label[index] = k
        return label, label_unique
```

```
    def train_test(self, train_data, train_label, test_data, test_label):
        train_data = np.squeeze(train_data, axis=-1)
        test_data = np.squeeze(test_data, axis=-1)
        svr_rbf = svm.SVC(kernel='rbf', C=1.0, gamma='auto')
        svr_rbf.fit(train_data, train_label)
        pred = svr_rbf.predict(test_data)
        print(np.sum(pred == test_label))


def evaluate_lbp(data_op=1, op=1, reduction=1, rate=0.2):
    """
    评估函数，在三个数据集上进行评估
    :param op: 1-all 2-part
    :param data_op: 1-CK 2-Fer 3-Jaffe
    :return:
    """
    filters = Gabor().build_filters()
    from tqdm import tqdm
    from data import CK, Fer2013, Jaffe
    _, x, y = CK().gen_train_no()
    if data_op == 2:
        _, x, y = Fer2013().gen_train_no()
    if data_op == 3:
        _, x, y = Jaffe().gen_train_no()
    train = []
    if op == 1:
        for i in tqdm(np.arange(0, x.shape[0], 1)):
            x[i] = preprocess.gray_norm(x[i])
            x[i] = preprocess.adaptive_histogram_equalization(x[i])
            res = Gabor().getGabor(x[i], filters, False, reduction)
            res = np.array(res).reshape(-1)
            res = np.append(res, y[i])
            train.append(res)
        train = np.array(train)
    if op == 2:
        for i in tqdm(np.arange(0, x.shape[0], 1)):
            img, dets, shape_list, img_list, pt_post_list = preprocess.deal(x[i])
            res1 = Gabor().getGabor(img, filters, 0, 1)
            res1 = np.array(res1)
            res = []
            if len(shape_list) == 0:
                continue
            for _, pt in enumerate(shape_list[0].parts()):
```

```
                    px, py = min(max(pt.x, 0), 47), min(max(pt.y, 0), 47)
                    im = res1[0]
                    cv2.circle(im, (px, py), 2, (255, 0, 0), 1)
                    res.append(res1[:, px, py])
                res = np.array(res)
                res = np.append(res, y[i])
                train.append(res)
            train = np.array(train)
        if op == 3:
            for i in tqdm(np.arange(0, x.shape[0], 1)):
                res = LBP().get_lbp(x[i])
                res = np.array(res).reshape(-1)
                res = np.append(res, y[i])
                train.append(res)
            train = np.array(train)
        if data_op != 2:
            from sklearn.model_selection import train_test_split
            x_train, x_test, y_train, y_test = train_test_split(train, train, random_state=2019,
test_size=rate)
            Classifier().SVM(x_train, x_test)
        test1 = []
        test2 = []
        if data_op == 2:
            _, x, y = Fer2013().gen_valid_no(1)
            for i in tqdm(np.arange(0, x.shape[0], 1)):
                res = LBP().get_lbp(x[i])
                res = np.array(res).reshape(-1)
                res = np.append(res, y[i])
                test1.append(res)

            _, x, y = Fer2013().gen_valid_no(2)
            for i in tqdm(np.arange(0, x.shape[0], 1)):
                res = LBP().get_lbp(x[i])
                res = np.array(res).reshape(-1)
                res = np.append(res, y[i])
                test2.append(res)
            test1 = np.array(test1)
            test2 = np.array(test2)
            print("Public")
            Classifier().SVM(train, test1)
            print("Pirvate")
            Classifier().SVM(train, test2)
```

```
# 在未训练的数据集上进行测试
def evaluate1_lbp():
    filters = Gabor().build_filters()
    from tqdm import tqdm
    from data import CK, Fer2013, Jaffe
    _, x, y = Fer2013().gen_train_no()
    train = []
    for i in tqdm(np.arange(0, x.shape[0], 1)):
        res = LBP().get_lbp(x[i])
        res = np.array(res).reshape(-1)
        res = np.append(res, y[i])
        train.append(res)
    train = np.array(train)

    test = []
    _, x, y = Jaffe().gen_train_no()
    for i in tqdm(np.arange(0, x.shape[0], 1)):
        res = LBP().get_lbp(x[i])
        res = np.array(res).reshape(-1)
        res = np.append(res, y[i])
        test.append(res)
    test = np.array(train)
    Classifier().SVM(train, test)

    test = []
    _, x, y = CK().gen_train_no()
    for i in tqdm(np.arange(0, x.shape[0], 1)):
        x[i] = preprocess.gray_norm(x[i])
        x[i] = preprocess.adaptive_histogram_equalization(x[i])
        res = Gabor().getGabor(x[i], filters, False, 6)
        res = np.array(res).reshape(-1)
        res = np.append(res, y[i])
        test.append(res)
    test = np.array(train)
    Classifier().SVM(train, test)


if __name__ == "__main__":
    # 在本数据集上训练并评估
    # 0.9645 0.949 (784, 36865) (197, 36865) re = 6
    print("CK+:")
    evaluate_lbp(1, 3, 3)
    # public: 0.458 private : 0.389
    # print("Fer2013:")
```

```
    # evaluate_lbp(2, 3, 8)
    # 0.4186 0.697
    # print("Jaffe")
    # evaluate_lbp(3, 3, 1, 0.1)
    #  在不同数据集上训练并评估
    # Jaffe 0.705 CK: 0.705
    # evaluate1_lbp()
```

语音识别部分：

```
import json
import pyaudio
import numpy as np
from vosk import Model, KaldiRecognizer, SetLogLevel


def SaveWave(model):
    #  设置音频参数
    FORMAT = pyaudio.paInt16    #  音频流的格式
    RATE = 44100    #  采样率，单位 Hz
    CHUNK = 4000    #  单位帧
    THRESHOLDNUM = 30    #  静默时间，超过这个个数就保存文件
    THRESHOLD = 100    #  设定停止采集阈值

    audio = pyaudio.PyAudio()
    stream = audio.open(format=FORMAT,
                        channels=1,
                        rate=RATE,
                        input=True,
                        frames_per_buffer=CHUNK)
    frames = []
    print("开始录音...")
    count = 0
    while count < THRESHOLDNUM:
        data = stream.read(CHUNK, exception_on_overflow=False)
        np_data = np.frombuffer(data, dtype=np.int16)
        frame_energy = np.mean(np.abs(np_data))
        # print(frame_energy)
        #  如果能量低于阈值持续时间过长，则停止录音
        if frame_energy < THRESHOLD:
            count += 1
        elif count > 0:
```

```
                count -= 1

            frames.append(data)
        print("停止录音!")
        stream.stop_stream()
        stream.close()
        audio.terminate()

        rec = KaldiRecognizer(model, RATE)
        rec.SetWords(True)
        str_ret = ""
        for data in frames:
            if rec.AcceptWaveform(data):
                result = json.loads(rec.Result())
                if 'text' in result:
                    str_ret += result['text']

        result = json.loads(rec.FinalResult())
        if 'text' in result:
            str_ret += result['text']

        str_ret = "".join(str_ret.split())
        return str_ret


if __name__ == "__main__":
        model = Model("vosk-model-small-cn-0.22")
        SetLogLevel(-1)

        while 1:
            res = SaveWave(model)
            if res != "" and res != None:
                print(res)
```