上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

课程报告

REPORT OF CURRICULUM

报告题目： Another Me

小组名称： DDL 工人队

小组成员： 张雅淇 张婷燕 李玮晨

指导教师： 鲍杨

目录

1. 项目介绍

本项目名为“Another Me”，即卡通世界的另一个我，通过运用 OpenCV、TensorFlow、keras 等对手势以及选定的卡通人物表情进行深度的识别、分析于分类，创建相应的学习库，再进一步通过 Python，利用已学习的手势，对应指令进行卡通人物衣服与头饰的切换，同时实时改变游戏者的面部表情特征，实时更换对应卡通人物的面部表情。

此小游戏向人们特别是小朋友们提供了一个十分有趣且操作简单的换装小游戏，在充满童心的同时，还能达到激发孩子们对科学，对编程的兴趣，也不失为我们大朋友的一种消遣、解压和回归童心的方式，兼具科学性与娱乐性。

2. Library 介绍

2.1 OpenCV

主要介绍 Haar 分类器。Haar 分类器由 Haar 特征提取、离散强分类器、强分类级联器组成。核心思想是提取人脸的 Haar 特征，使用积分图对特征进行快速计算，然后挑选出少量关键特征，送入由强分类器组成的级联分类器进行迭代训练[1]。

所谓特征就是像素经过运算之后得到的某种结果，可以是向量也可以是矩阵等，而 Haar 特征便是较为特殊的特征类型，图 1-1 是其在 OpenCV 里的所有类型，特征模板在所选图片上进行遍历，运用公式计算出则会张图片上所有的特征，而由于以往公式所需计算次数太大，因此发明了积分图的方法，大大简便了计算。

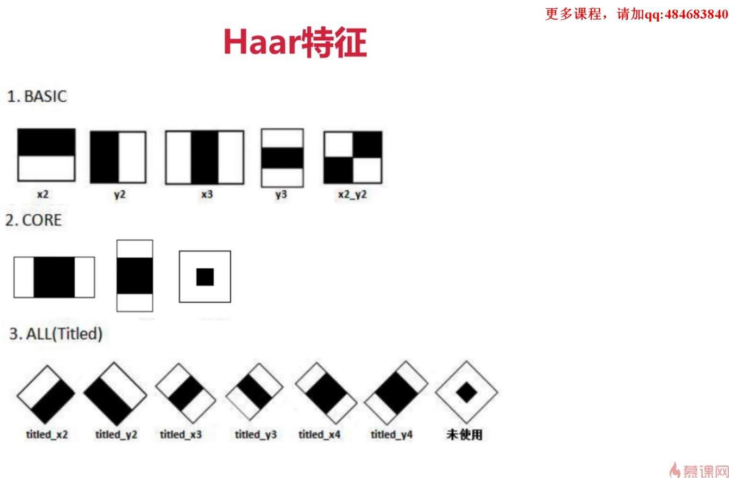


图 1-1

2.2 TensorFlow

TensorFlow 是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库 DistBelief[2]。

简单来说，其本质就是“tensor + 计算图”，而其中“tensor”就是数据（可是维也可以是 n 维），并通过“op（option）”也就是操作（四则运算操作和赋值操作），与数据图相结合（数据操作），而这些数据图等也都放在“Session”里进行处理它可以理解为一种交互式的环境。

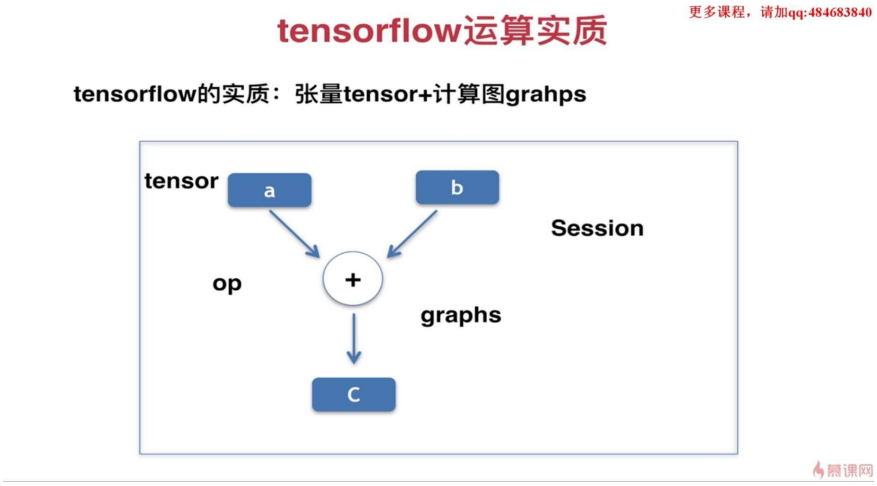


图 1-2

2.3 Keras 神经网络（本项目基于 Tensorflow）

Keras 的神经网络 API 是在封装后与使用者直接进行交互的 API 组件，在使用时可以调用 Keras 的其它组件。除数据预处理外，使用者可以通过神经网络 API 实现机器学习任务中的常见操作，包括人工神经网络的构建、编译、学习、评估、测试等 [2]。

所谓神经网络就是通过信息的正反向传播与回馈，输入大量的数据信息，在中间层进行分析处理，正向输出预测结果，再对结果进行反馈，反向传递改动神经元使其朝着正确的方向靠近，通过无数次信息交互，逐渐搭建起神经网络。可以说是一个人工的数学模型，神经网络搭建，输入的信息被一层层解析优化再识别为计算机能读懂的语言，最后输出想要的结果。有时对于一个有价值的神经网络，我们可以保留其最理解、代表特征转化能力，把他迁移至其他信息的处理过程，即拆除输出层，接上另一类的中间处理层，进行深一步的再次学习。

而本项目主要运用的是卷积神经网络。卷积神经网络基本是通过过滤器收集提取图片长宽高的信息，可以理解生成一个长宽更窄，高度更高的更小的图片，以此类推进行多次卷积，将压缩增高的信息嵌入普通的分类神经层上，就可以进一步处理了。但这样会丧失一部分信息，故再引入池化处理此问题。即压缩时不

压缩长宽。流行的方式是输入图片，再经过一层卷积层，接着通过池化处理卷积的信息，再经过同样的步骤卷积再池化，通过两层全连接神经层，最后接上分类器，形成一个简单的图片处理方式。

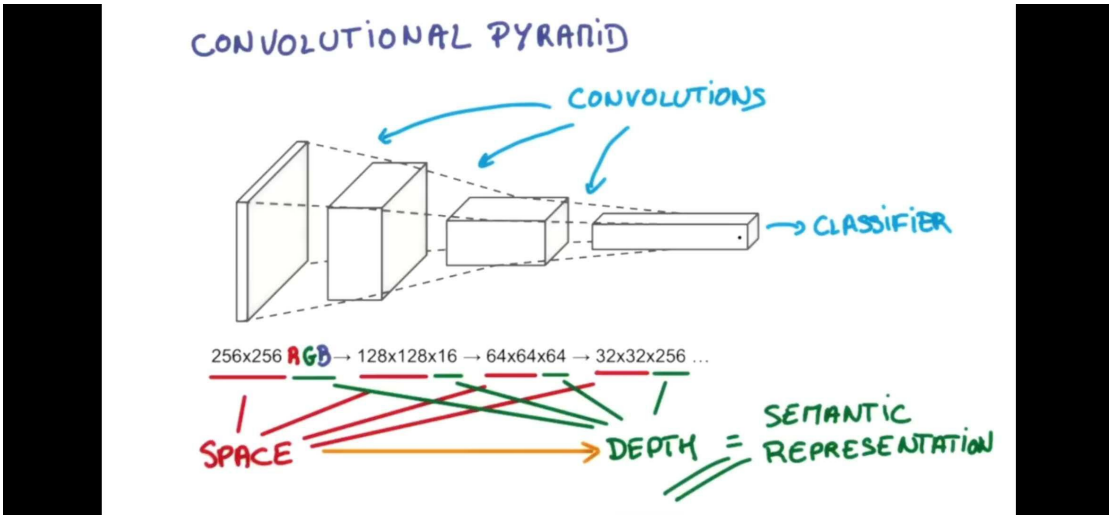


图 1-3

2.4 其他库

2.4.1 Pygame

是一个免费和开源的 python 编程语言库，用于制作基于优秀 SDL 库之上的游戏等多媒体应用程序。与 SDL 一样，pygame 具有高度可移植性，几乎可在所有平台和操作系统上运行。是本项目的一个基本承载模块，采用了其主要的编程思想与重要的指令如 screen.blit（）等。

2.4.2 Numpy

NumPy（Numeric Python）提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。专为进行严格的数字处理而产生。本项目多次运用了 numpy，其强大的运算功能为手势与人脸的机器学习提供了有利的支持。

除此之外，还调用了 sys 模块、time（时间处理）、imutils（提供一些基本的图像处理功能，更易使用 OpenCV 等）。

3. 代码解析

3.1 人脸识别（张婷燕）

1. 导入已经训练好的人脸识别、表情识别数据模型

```
detection_model_path = 'models/haarcascade_frontalface_default.xml'
```

```
emotion_model_path = 'models/_mini_XCEPTION.102-0.66.hdf5'
face_detection = cv2.CascadeClassifier(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
```

2. 创建初始化变量：

其中，np.zero 函数可以返回一个指定形状和类型的用 0 填充的数组.

```
canvas = np.zeros((250, 300, 3), dtype="uint8")
preds = []
label = "neutral"
```

3. 利用 OpenCV 的 VideoCapture 函数捕获导入实时图片

```
cap = cv2.VideoCapture(0)
```

4. 对实时图片进行数据归一化、标准化处理，提取识别脸部识别模型所需的特征量：

a.

图片缩放：由于图片大小比例不一，故而识别所出脸部特征的长度和高度并不可靠。故而，需要利用 OpenCV 的 resize（）函数在保持脸部长宽比例不变的前提下缩放至统一、标准的大小以便分析。代码示例如下：

```
roi = cv2.resize(roi, (64, 64))
```

b.

灰度校正:在图像处理时，部分图像在 RGB 颜色空间的信息特征不如 HSV 夜色空间等颜色空间更清晰，故而需要利用 OpenCV 中的 cvtColor 函数进行色彩空间转化，将图片转换为灰度图。代码示例如下：

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

5. 匹配算法：得到特征量后，将实时图片处理后数据与模型中的特征量进行匹配。

```
faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(30,30),flags=cv2.CASCADE_SCALE_IMAGE)
```

6. 如果检测出人脸，则进一步识别人脸表情：

同理，对脸部数据进行归一化、标准化处理，提取识别表情识别模型所需的特征量：

a.

使用标准库中的 sorted 函数对进行数据处理后的特征量进行排序，sorted 中的参数设置为(迭代对象：faces，reverse=True 降序，key:用于比较的函数,若大于则返回 1，若小于则返回-1，若等于则返回 0)

b.使用 numpy 中的 astype 函数修改数据类型

c.使用 keras 的 img_to_array()函数将 PIL 图片转化成 Numpy 矩阵（二维）

d.使用 numpy 的 expand_dims 函数扩展至三维的

```
if len(faces) > 0: #检测出人脸
                (fX, fY, fW, fH) = sorted(faces, reverse=True,key=lambd
a x: (x[2] - x[0]) * (x[3] - x[1]))[0]
                roi = gray[fY:fY + fH, fX:fX + fW]#灰度校正
                 roi = cv2.resize(roi, (64, 64))#图片缩放
                roi = roi.astype("float") / 255.0#修改数据类型
                roi = img_to_array(roi)#将 PIL 图片转化成矩阵
            roi = np.expand_dims(roi, axis=0)# 扩展数组至三维
```

7. 将经过进一步处理后数据与模型中的特征量进行匹配后，利用模型中的 emotion_classifier.**predict** 得出识别七种表情的实时可能性数组 preds；并使用 numpy 的 max 函数得出实时表情 preds 可能性最大值 emotion_probability；使用 numpy 中的 argmax 函数返回实时表情可能性最大值的索引，从而识别得出实时最大可能性表情 label。

```
                preds = emotion_classifier.predict(roi)[0]
                emotion_probability = np.max(preds)
                label = EMOTIONS[preds.argmax()]#可能性最大值表情
                canvas = np.zeros((250, 300, 3), dtype="uint8")
```

8. 指定与实时最大可能性表情 label 相对应的实时图片路径，进而使用 pygame 的 image.load 函数导入图片，使用 screen.blit 函数在指定位置显示图片

```
                image_path="images//"+label+".png"
                face = pygame.image.load(image_path)
                screen.blit(face,(0,0))
```

9. 创建各种表情可能性值矩形图：

a.使用 zip 将 EMOTIONS 列表（表情名）与各种表情的可能性中对应的元素打包成元组，并返回成列表形式

b.使用 for in enumerate 语句枚举可遍历的数据对象(由数组构成的列表，即：zip(EMOTIONS, preds))，同时列出数据下标（i）和可能性(preds)

c. 使用 OpenCV 中的 rectangle 函数，通过设置对角线端点位置画出实时可能性矩形图并以矩形框框出识别出的实时视频的人脸部分

d.使用 OpenCV 的 putText 函数在图片上添加文字，其中，参数设置为（作用的图片，显示的文本，文本在图片上的位置坐标，字体类型，字体大小，文本颜色，字体粗细）

e. 最终使用 OpenCV 的 imshow 展示图片窗口。

```
for (i, (emotion, prob)) in enumerate(zip(EMOTIONS, preds)):
            text = "{}: {:.2f}%".format(emotion, prob * 100)
             #规定所显示的可能性值保留两位小数
            w = int(prob * 300)
            cv2.rectangle(canvas, (7, (i * 35) + 5),#参数（图像，矩形的一
个顶点，矩形对角线上另一个顶点，线条粗细，取负值时填充了色彩）
            (w, (i * 35) + 35), (0, 0, 255), -1)#实时可能性值矩形图
```

```
            cv2.putText(canvas, text, (10, (i * 35) + 23),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45,
            (255, 255, 255), 2)#在图片中添加文字

             cv2.putText(frame, label, (fX, fY - 10), cv2.FONT_HERSHEY_S
             IMPLEX, 2, (0, 0, 255), 2)
             cv2.rectangle(frame, (fX, fY), (fX + fW, fY + fH),(0, 0, 25
             5), 2)#框出所识别出的人脸部分
             cv2.imshow("Probabilities", canvas)
             cv2.imshow('Original',frame)
```

3.2 手势识别（张雅淇）

1. 手势识别切换物件模块

a. 安装并导入所需库与模块

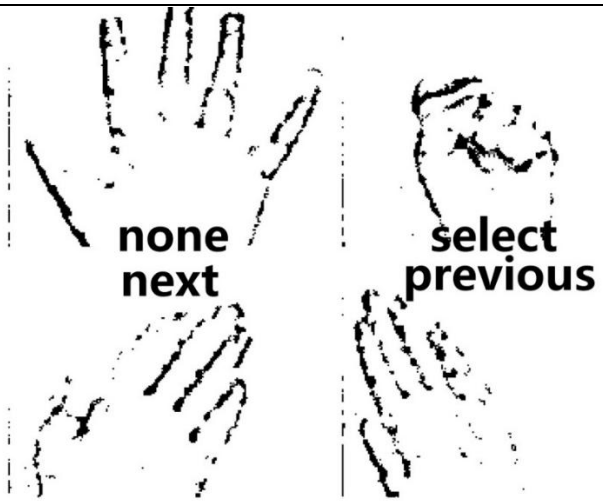
首先导入所需的库与模块，主要有：

cv2（用于实现获取实时拍摄的图像并处理、识别），pygame（用于实现游戏中切换图片素材），time（用于计时以实现每过特定间隔输入一次识别出的手势对应的命令，防止命令输入过快），pickle（用于读取模型中储存的数据），keras（用于基于我们建立的手势分类库建立手势识别及分类模型并后续用于预测输入的图片中的手势）。

```
from PIL import Image
import pygame
from keras.models import load_model
from imutils import paths
import numpy as np
import imutils
import cv2
import sys
import pickle
import time
import os
```

2. 收集手势数据

如下图，为了保证准确率，选择尽可能少的手势个数、尽可能区分度较高的手势；为了保持用户游玩方便，选择可以在较短时间内较容易做出的手势。



随后，定义保存图片函数 saveROIImg，在每保存一张图片后增加 1 计数，从而计算得总共已保存的图片数量，与希望得到的图片个数相比较，使保存图片的程序在达到这一所期望的个数后才终止。

```
#保存图片函数
def saveROIImg(img):
    global counter,saveimg,gestname,sample_nums
    if counter > sample_nums:
        saveimg = False
        counter = 0
        gestname = ""
        return
    counter = counter + 1
    name = gestname + str(counter)
    print("Saving img:",name)
    cv2.imwrite(path+name + ".png", img)
    #防止保存图片过快，造成手势无变化
    time.sleep(0.04 )
```

利用保存图片函数给每种手势拍摄 200 张图片以后，我们就得到了我们的数据库：

同时定义手势处理函数 binaryMask,其中利用 cv2 库中提供的 rectangle，cvtColor，GaussianBlur，adaptiveThreshold 函数分别实现对原图取局部图片，处理为灰度图以便建立模型，对图片高斯滤波去噪，进行自适应阈值处理去噪突出手势线条及轮廓的处理。整体而言，定义这一函数的目的在于把摄像头拍摄得的手部图片处理得相对而言线条边缘清晰，噪点较少，容易“学习”。

```
#手势处理函数(二值掩模)
def binaryMask(frame, x0, y0, width, height):
    #只处理识别框部分
    cv2.rectangle(frame, (x0,y0),(x0+width,y0+height),(0,255,0),1)
    roi = frame[y0:y0+height, x0:x0+width]
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)
    th3 = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
    ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    if saveimg==True:
        saveROIImg(res)
    return res
```

3. 深度学习建立手势模型

本项目通过 keras 库以 tensorflow 为后端使用卷积网络来基于上一步中收集的图片数据集搭建手势图片分类的模型。我们事先设定了操控游戏的几种手势，并建立了相应的数据库，利用这些带有手势名称标签的训练数据对模型进行有监督训练，从而使网络能预测新的图像数据应如何归类。

卷积神经网络（CNN）具有表征学习能力，只要为其提供大量的输入数据，它就能提取其中的潜在特征。本项目进行有监督训练，使卷积网络能隐式地学习出手势图像的标签与图像本身的特征间的关系。由此生成的神经网络非常接近于实际上的生物神经网络。

Structure of CNN

构建模型的过程中，本项目分别使用以下参数：以 32，64，128，256，512 个卷积核数量，3×3 为卷积核大小，每次移动步长为 2×2，采用 same 来为矩阵最后剩余的部分填充 0 以达到一个步长。因此，由于输入图像尺寸为 100×100 当卷积核数量为 32 时，100/2=50，大小为 3×3 的卷积核会移动 50 次，得到一个 50×50 的矩阵。

```
#神经网络构建
model = Sequential()

# 第一层卷积池化
model.add(Conv2D(32, (3, 3), padding="same", input_shape=(100,100, 1), activation="re
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 第二层卷积池化
model.add(Conv2D(64, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 第三层卷积池化
model.add(Conv2D(128, (3, 3), padding="same", activation="relu"))
model.add(Conv2D(256, (3, 3), padding="same", activation="relu"))#提高精度
model.add(Conv2D(512, (3, 3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 全连接层
model.add(Flatten())
model.add(Dense(64, activation="relu"))
```

同时，每次附加纠正线性单元（ReLu）操作，通过向卷积网络中引入非线性计算，使建立的神经网络具备适度的稀疏性、避免过拟合，并加快构建神经网络的速度。

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

relu函数

经过以上过程，便从图像中提取出了对识别手势而言足够多的特征值，从而得到了一个能归类图像的模型。

```
2020-06-08 14:41:40.226495: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x1fc2c20dec0 initialized for
 platform Host (this does not guarantee that XLA will be used). Devices:
2020-06-08 14:41:40.235380: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Defaul
t Version
2020-06-08 14:41:40.258412: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] Device interconnect StreamExecutor
 with strength 1 edge matrix:
2020-06-08 14:41:40.267815: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1108]
构建成功，开始训练
Train on 964 samples, validate on 242 samples
Epoch 1/700
256/964 [======>.....................] - ETA: 1:31 - loss: 1.8196 - accuracy: 0.1641
```

4. 输入命令

为了实现手势实时控制小游戏，先用下图中加载模型与其相应的标签，加载其中建构的神经网络为后续预测实时录入的手势分类做准备。

```
#模型和标签名
MODEL_NAME = "models/ss_model.h5"
LABEL_NAME = "models/ss_labels.dat"
#加载标签
with open(LABEL_NAME, "rb") as f:
    lb = pickle.load(f)
#加载神经网络
model = load_model(MODEL_NAME,compile = False)
```

利用 expand_dims 函数添加摄像头输入的图片的维度后，再使用 predict 函数用先前载入的手势识别模型预测摄像头中实时输入的图片中的手势属于哪一类。

```
# 添加维度
roi1 = np.expand_dims(roi1, axis=2)
roi1 = np.expand_dims(roi1, axis=0)
# 预测手势
prediction = model.predict(roi1)
gesture = lb.inverse_transform(prediction)[0]
```

设置 gesture 在摄像头未输入图像时值为“none”，从而使开头画面直至摄像头打开，开始输入图像前维持“none”值。此时，在每次循环中开始识别手势前利用 time.time 定义 start 计算当前经过时间，再由 time.time 定义 end 计算循环识别完后的当前时间，计算其差值是否大于 1，从而实现一秒内识别摄像头输入的图像中手势一次并输出相应类别标签名，防止小游戏中连续输入标签名，导致难以控制。

```
gesture = "none"
start=time.time()

if ret == True:
    roi_gesture = binaryMask(frame, x0, y0, width, height)

    end  = time.time()
    if end - start > 1:#每过一秒键入手势输入的命令一次
        start=time.time()
```

5. 游戏物件切换

a. 图片素材处理

首先借用 Emselfy 游戏截图得到 22 张不同衣服、饰品、表情、同格式、同尺寸的图片素材。

图片素材处理通过 Pillow 库中的 crop 函数实现对同一尺寸的图片裁剪同一坐标范围内的局部图片，再用 save 函数保存局部图片，从而使最终载入的图片素材做到完全对齐。

```
from PIL import Image
#裁图
body=Image.open("face/6.png")
bodye=body.crop((110,85,385,265))
bodye.save("face/6a.png")
```

b. 手势控制

由于上文中已有代码完成将模型识别出的手势的标签转换为一个字符串量 gesture，在这里只需设置 gesture 的值为“next”、“previous”、“select”时用

pygame 库实现相应发生的事件。

如下图，利用 iconnum 的值为 0 或 1 来区分玩家在选择饰品还是选择衣服，其中 0 代表玩家正在切换衣服，1 代表玩家正在切换饰品；利用 charanum 表示当前衣服图片素材的序号，利用 kazarinum 表示当前饰品图片素材的序号。

然后，为了实现切换饰品和切换衣服之间的转换，定义 gesture 的值为“select”也就是玩家做出 select 的手势时，改变 iconnum 的值。此处，通过 iconnum=1-iconnum 的运算式使得 iconnum 在表示衣服的 0 和表示饰品的 1 间变换。

```
if gesture == "next": #设置输入手势next/previous时切换图片
    if iconnum == 0:
        charanum+=1
        if charanum==len(charalist):
            charanum=0
    else:
        kazarinum+=1
        if kazarinum==len(accessorylist):
            kazarinum=0
elif gesture == "previous":
    if iconnum == 0:
        charanum-=1
        if charanum==-1:
            charanum=len(charalist)-1
    else:
        kazarinum-=1
        if kazarinum==-1:
            kazarinum=len(accessorylist)-1
elif gesture == "select": #设置输入手势select时衣服/饰品间切换
    iconnum = 1 - iconnum
```

随后，通过 pygame 库中提供的 blit 函数根据 charanum、kazarinum 的数值分布在游戏界面中显示出当前的图片素材，最终完成手势操控换人物衣服、饰品的效果。

```
screen.blit(accessorylist[kazarinum],(0,0)) #显示饰品
screen.blit(iconlist[iconnum],(0,0)) #显示分类图标
```

3.3 整合与优化(李玮晨）

1. 删除杂糅，更改形式

```
import win32api
```

```
import win32con
```

```
def keybd_event(VK_CODE):
    #VK_CODE 为键盘编码
    VK_CODE = int(VK_CODE)
    #按键按下
    win32api.keybd_event(VK_CODE, 0, 0, 0)
    #按键弹起
    #win32api.keybd_event(VK_CODE, 0, win32con.KEYEVENTF_KEYUP, 0)
```

```
  else:
        if gesture=='select':
            keybd_event(70) #键盘按下 F(forward)
            win32api.keybd_event(70, 0, win32con.KEYEVENTF_KEYUP, 0)
        elif gesture=='cancel':
            keybd_event(66) #键盘按下 B(back)
            win32api.keybd_event(66, 0, win32con.KEYEVENTF_KEYUP, 0)
        elif gesture=='next':
            keybd_event(65) #按下 A(PREVIOUS)
            win32api.keybd_event(65, 0, win32con.KEYEVENTF_KEYUP, 0)
        elif gesture=='previous':
            keybd_event(68) #键盘按下 D(NEXT)
            win32api.keybd_event(68, 0, win32con.KEYEVENTF_KEYUP, 0)
        elif gesture=='none': #手势输入 NONE 时弹起按键
            win32api.keybd_event(70, 0, win32con.KEYEVENTF_KEYUP, 0)
            win32api.keybd_event(66, 0, win32con.KEYEVENTF_KEYUP, 0)
            win32api.keybd_event(65, 0, win32con.KEYEVENTF_KEYUP, 0)
            win32api.keybd_event(68, 0, win32con.KEYEVENTF_KEYUP, 0)
```

```
    for event in pygame.event.get():
        if event.type==pygame.QUIT: #退出程序
            running=False
        if event.type==pygame.KEYDOWN: #按下键
            if event.key==pygame.K_e:#endgame
                running=False
            elif event.key==pygame.K_d:#下一个
                if layer1==0:
                    iconnum=1
                if layer1==2:
                    charanum+=1
                    if charanum==len(charalist):
                        charanum=0
                elif layer1==3:
```

```
                kazarinum+=1
                if kazarinum==len(accessorylist):
                    kazarinum=0
                    ......
```

原代码运用了 win32api 等命令，这是专门在 32 位平台上的应用程序接口，而首先小组成员计算机皆是 64 位的，因此猜想可以整改一下这一部分代码。原代码设置了一个模拟按键输入的程序而在后面手势代码处理部分，电脑自动对应手势输入相应的字母，这样一定程度加大了代码运行的繁琐度，降低了用户的体验感，且理论上来说这是不必要的，用户完全可以在不模拟按键的情况下进行手势识别操作。

结合小组成员的其他代码与相关资料对这一部分做了一些修改：

```
        if gesture == "next": #设置输入手势 next/previous 时切换图片
            if iconnum == 0:  #判断手势是上一个还是下一个，图标等于一是在换衣服，为零是在换头饰
                charanum+=1
                if charanum==len(charalist): #防止下标超出列表长度
                    charanum=0
            else:
                kazarinum+=1
                if kazarinum==len(accessorylist):
                    kazarinum=0
        elif gesture == "previous":
            if iconnum == 0:
                charanum-=1
                if charanum==-1:
                    charanum=len(charalist)-1
            else:
                kazarinum-=1
                if kazarinum==-1:
                    kazarinum=len(accessorylist)-1
        elif gesture == "select": #设置输入手势 select 时衣服/饰品间切换
            iconnum = 1 - iconnum #因为之前设置了 iconnum 只会是零和一，所以用此便可以做到切换
```

舍弃了之前的命令操作，换之以利用机器学习的手势识别结果来进行图片层级的切换与更改，继承了控制流的方法运用，按照事先划分好的层级，来进行角色，表情，头饰的切分工作。简言之，直接由标签名控制游戏的部分过程。

2. 添加界面，优化效果

a. 添加游戏介绍界面

```
#初始化界面
pygame.init()
screen=pygame.display.set_mode((275,475)) #窗口的大小
```

```
pygame.display.set_caption("Another Me") #类似标题
pygame.mixer.init()
index=pygame.image.load("images/index.png") # 导入第一张图片
screen.blit(index,(0,0)) #画出界面，设置坐标，从左上角开始，铺满整个屏幕
pygame.display.update() #更新界面
time.sleep(3) # 时间间隔
index=pygame.image.load("images/index1.png") # 顺序导入第三张图片，以此类推
screen.blit(index,(0,0))
pygame.display.update()
time.sleep(3)
index=pygame.image.load("images/index3.png")
screen.blit(index,(0,0))
pygame.display.update()
time.sleep(3)
index=pygame.image.load("images/index2.png")
screen.blit(index,(0,0))
pygame.display.update()
time.sleep(3)
pygame.display.update()
```

利用 pygame 的灵活性，事先做好介绍游戏规则等的图片，进行图片的载入，考虑到图片的多样性以及玩家的信息理解接受时间距，设置了时间间隔为 3 秒，同时设置了更新的命令，以保证三张图片能够正常按顺序载入且不被覆盖。

b. 添加游戏音效

```
pygame.mixer.music.load('bgm.mp3') # 加载歌曲
pygame.mixer.music.play() # 播放
            ......
if pygame.mixer.music.get_busy() == False:
        pygame.mixer.music.play() # 播放，循环
```

首先载入游戏音效文件，在游戏主体头部首先利用 if 语句，实现音乐的循环播放。

3. 最终的整合

首先将两个识别主体等需要的库一次性导入，再引入识别的模型数据与图片数据，定义好相关的函数，做好前期预设工作。其次根据组员的代码报告与细节处理文本，在 pygame 主体中设定好手势识别与人脸识别的运行代码，调整好参数，设定角色，表情，头饰的相对位置与顺序，避免被覆盖。

```
 elif gesture == "select": #设置输入手势 select 时衣服/饰品间切换
                iconnum = 1 - iconnum #因为之前设置了 iconnum 只会是零和一，所以用此便可以做到切换
            #为了防止覆盖，先显示角色，因为是最大的图片，然后再显示人脸，头饰，这样图片由大到小就不会被
覆盖了。
            screen.blit(charalist[charanum],(0,0)) #显示角色

            image_path="images//"+label+".png" #实时根据识别出的表情变动的图片路径
```

```
            face = pygame.image.load(image_path) #导入实时图片文件
            screen.blit(face,(0,0)) #制定导入图片位置


            screen.blit(accessorylist[kazarinum],(0,0)) #显示饰品
            screen.blit(iconlist[iconnum],(0,0)) #显示分类图标
            pygame.display.update() #显示完就更新一下
```

最后再做好实时的更新显示环节与游戏结束环节

```
#显示摄像头内容和处理后手势的图像内容
        cv2.imshow('Original',frame)
        cv2.imshow('ROI', roi_gesture)
        cv2.imshow("Probabilities", canvas)#实时表情可能性分析窗口
```

4. 局限与展望

在素材收集与制作过程中出现了切分与选择的错乱，设想可以利用爬虫抓取图片素材，让游戏更加丰富。另外采取卡通形象学习的形式可能与人脸实体有一定差异，有的表情识别可能比较困难，对选取的图片也有了一个更高的要求。

可以朝着利用 OpenCV 等保存游戏中自己喜欢部分，比如保存成一段视频或自己喜欢的图片，存入指定文件夹中。

还可以设置多种的音效，比如换衣服与换头饰的两个不同部分设置不同的音效，增加游戏效果。

5. 小结

我们受到儿时玩过的换装游戏的启发，结合 Python 课上了解与学习到的人脸识别与手势识别，期望可以利用 Python 实现二者的结合给卡通小人物换装的设想，我们深入的查找了相关视频介绍以及 GitHub 上的相关开源项目，积极寻求帮助，小组成员也展开了热烈讨论，最终利用 OpenCV、tensorflow 与 keras 等进行图像的机器学习，形成了适用于本项目的指令类别，再结合 Pygame 等制作了一个操作简单、内涵丰富的智能换装游戏。总而言之，我们小组成员收获颇丰，培养了自己动手解决问题的能力，也期待以后能独立的更好的完成有意义的项目。在此，感谢各位小组成员的付出，感谢鲍老师等的耐心指导！

参考文献

[1] DUffiE D, SCHEICHER M, VUILLEMEY G. Central Clearing and Collateral Demand[J]. : 48.

[2]百度百科》. https://baike.baidu.com/item/Keras/22792516?fr=aladdin