> `ps`：因为时间紧迫，没有系统学习 keras 的所有功能，也没有系统学习 opencv，仅了解了本项目所需要的 keras 和 opencv 功能

代码运行环境

> `python 3.7`
>
> `keras 2.6.0`

代码流程

① 导入并设置训练集和测试集的文件数据

② 搭建神经网络

③ 设置 batch 并训练

④ 使用 opencv 从摄像头读取并进行人脸表情识别

导入并处理训练集和测试集数据

`ImageDataGenerator` 类：

> 通过实时数据增强生成批量图像数据向量。训练时会无限循环生成数据，直到达到规定的 `epoch` 次数为止。

`rescale`：重缩放因子。默认为 `None`。如果是 `None` 或 `0`，不进行缩放，否则将数据乘以所提供的值（在应用任何其他转换**之后**）

`flow_from_directory` 函数：

> 获取目录路径并生成一批增强数据

- `target_size`：整数的元组（高度、宽度）。默认值：（256，256）。将调整找到的所有图像的尺寸。
- `color_mode`：`grayscale`(灰度图)、`rgb`、`rgba` 之一。默认值：`rgb`。是否将图像转换为具有1、3或4个通道。
- `batch_size`：数据 `batch` 的大小（默认值：32）
- `class_mode`：确定返回的标签数组的类型：分类（“categorical”）、二进制（“binary”）、稀疏（“sparse”）、输入（“input”）、无（None）模式之一。默认值：分类。

  > 其中 `“categorial”` 则是 2维 one-hot 编码标签；

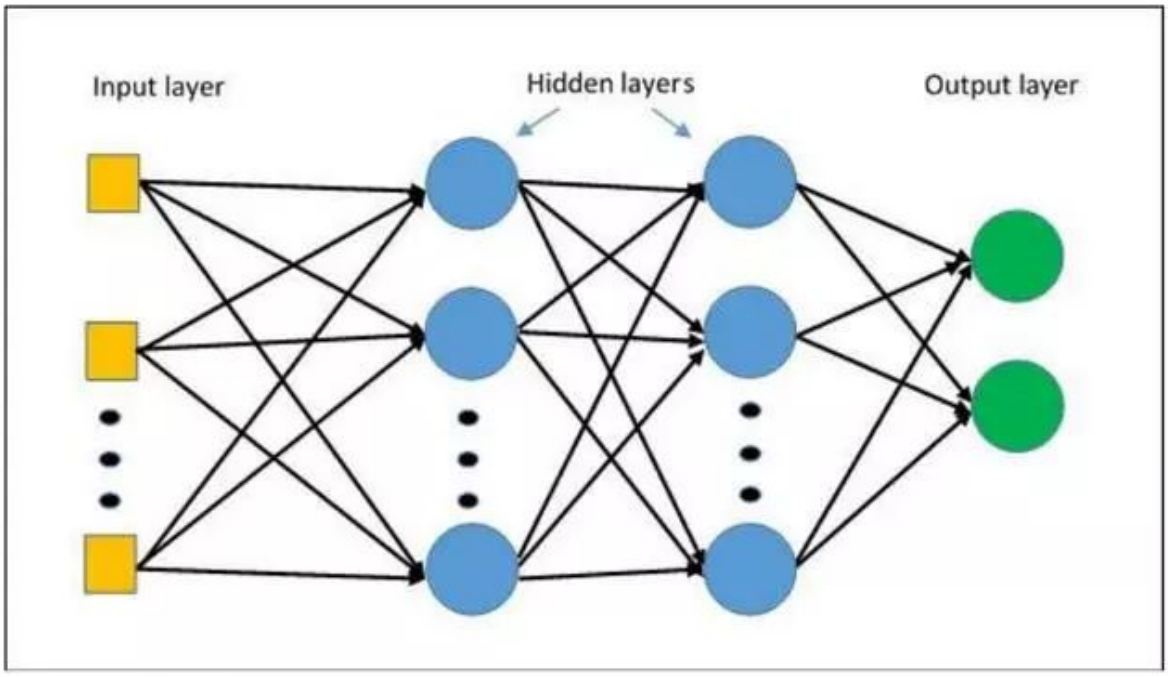
```
train_dir = 'D:\\RecruitNew\\train'
val_dir = 'D:\\RecruitNew\\test'
# 设置数据集路径
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
# 定义类
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(48,48),
        batch_size=64,
```

```
        color_mode="grayscale",
        class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(48,48),
        batch_size=64,
        color_mode="grayscale",
        class_mode='categorical')
# 处理数据
```

搭建神经网络

- `tensorflow.keras.models` 是 `keras` 中的一个模型库默认为顺序模型

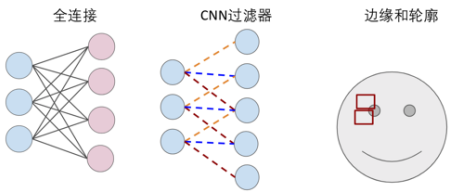


The MLP architecture

卷积层

二、卷积神经网络简介

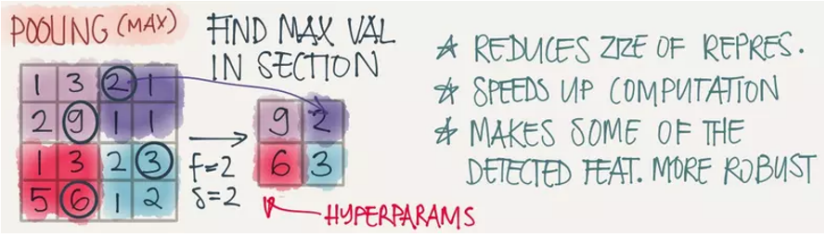
- 在神经网络中，每一层的每个神经元都与下一层的每个神经元相连(如下图)， 这种连接关系叫全连接（Full Connected）。如果以图像识别为例，输入就是是每个像素点，那么每一个像素点两两之间的关系(无论相隔多远)，都被下一层的神经元"计算"了。
- 这种全连接的方法用在图像识别上面就显得太"笨"了，因为图像识别首先得找到图片中各个部分的"边缘"和"轮廓"，而"边缘"和"轮廓"只与相邻近的像素们有关。
- 这个时候卷积神经网络(CNN)就派上用场了，卷积神经网络可以简单地理解为，**用滤波器(Filter)将相邻像素之间的"轮廓"过滤出来。**



二、卷积神经网络简介

■ 池化(Pooling)

- 用滤波器进行窗口滑动过程中，实际上"重叠"计算了很多冗余的信息，而池化操作就是去除这些冗余信息，并加快运动。Pooling的方式其实有多种，用的最多的是max-pooling就是取一个区域中最大的值，如图将一个4x4的图片max-pooling 一个2x2的图片。



`Flatten` 层用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。

`Dense` 层即为 Keras 中的全连接层。

```
emotion_model = Sequential()
#卷积层
emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=
(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))
#全连接层
emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))
```

训练

`compile` 用于配置训练模型

```
emotion_model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001,
decay=1e-6),metrics=['accuracy'])
```

`loss`：为 `Loss function` 也就是损失函数 `categorical_crossentropy` 多类的对数损失

`optimizer`：为优化器

`metrics`：在训练和测试期间的模型评估标准

`fit_generator`：该函数允许生成器在辅助线程中运行，并且运行速度更快。

```
def fit_generator(self, generator,          #生成器，一个yield的函数，迭代返回数据
                  steps_per_epoch,          #一次训练周期(epoch)里面进行多少次batch=(总训练样本数/batchsize)
                  epochs=1,                 #一个epoch代表一次全数据集训练过程，设置进行几次全数据集的训练
                  verbose=1,                #一个开关，打开时，打印清晰的训练数据，即加载ProgbarLogger这个回调函数
                  callbacks=None,           #设置业务需要的回调函数(其实是类)，所有的回调函数都要继承callback类或其子类
                  validation_data=None,     #验证用的数据源设置，evaluate_generator函数要用到这个数据源，也是一个生成器
                  validation_steps=None,    #设置验证多少次数据后取平均值作为此epoch训练后的效果，val_loss,val_acc的值受这个参数直接影响
                  )
```


```
emotion_model_info = emotion_model.fit_generator(
        train_generator,
        steps_per_epoch=28709 // 64,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=7178 // 64)
```

保存模型：

```
emotion_model.save_weights('emotion_model.h5')
```

使用 CV2 捕捉并识别表情