# 看图说话&微表情识别

作者：雷雅婧 10175501115 熊双宇 10174102103

## 一、设计思想

对于人类来说，描述一张图片的内容是非常重要的。但因这个过程并没有标准答案，因此对于计算机来说这并不是一个简单地过程。我们希望通过本次实验能够设计一个模型完成让计算机给图片设定 caption 的目标。更进一步，如果在图片中检测到人脸，我们希望能识别出人的情绪表情。最终呈现出如图 3.1 的效果：

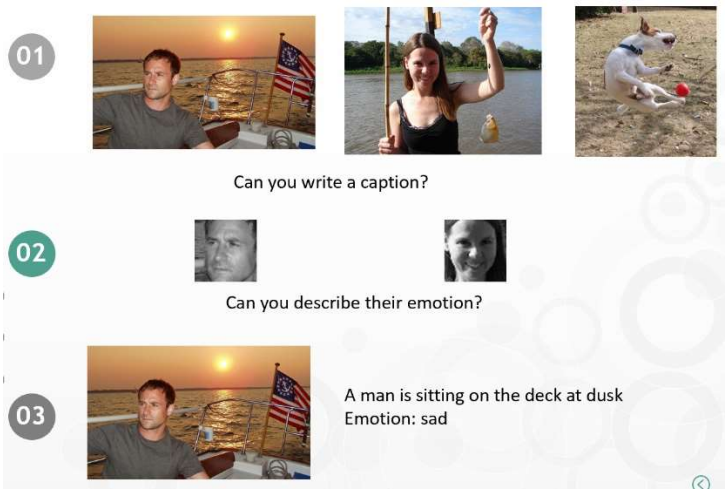



图 3.1 实现目标

## 二、实验环境和工具

Jupyter Notebook：

- Tensorflow
- Keras

## 三、实验过程

### 1. 看图说话

#### 1.1 数据集介绍

Flickr8k Dataset：该数据集已经成为研究基于句子的图片描述的基准，该数据集包括了 8052 张图片，每张图片包括了 5 句相关的描述性句子，示例如下：

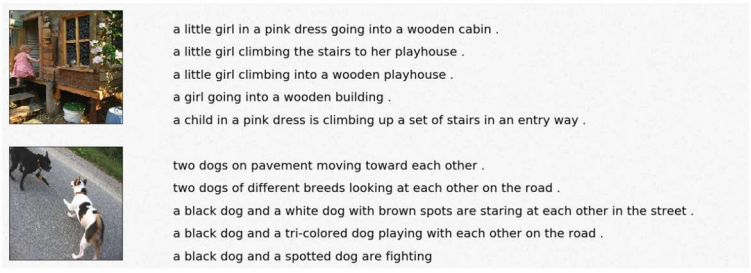



图 1 数据集的示例

### 1.2 实验环境和工具

Jupyter Notebook:

- 。Tensorflow
- 。Keras

### 1.3 数据预处理

#### 1.3.1 基本的数据清理

大写转换为小写，删除标点符号，去除单复数等，实现效果如图 4.1 所示:

```
descriptions['1000268201_693b08cb0e']
```

```
['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

图 4.1 原数据表示

```
descriptions['1000268201_693b08cb0e']
```

```
['child in pink dress is climbing up set of stairs in an entry way',
 'girl going into wooden building',
 'little girl climbing into wooden playhouse',
 'little girl climbing the stairs to her playhouse',
 'little girl in pink dress going into wooden cabin']
```

图 4.2 处理后的数据表示

#### 1.3.2 Unique words 的统计

将所有在描述语言中出现过的单词组成一个 vocabulary，统计在 vocabulary 中出现过的单词。起初计算出约 40000 个语句中总共出现 8763 个单词，但由于许多单词只出现两三次，对于预测性的模型来说，无实质性的帮助，因此接下来我们只考虑在所有语句中出现次数大于十的单词，计算出此时的 vocabulary 中就变为 1651 个单词。更进一步，我们还要多增加一个 0 padding，因此总单词数为 1652。可参考图 4.3 的流程:

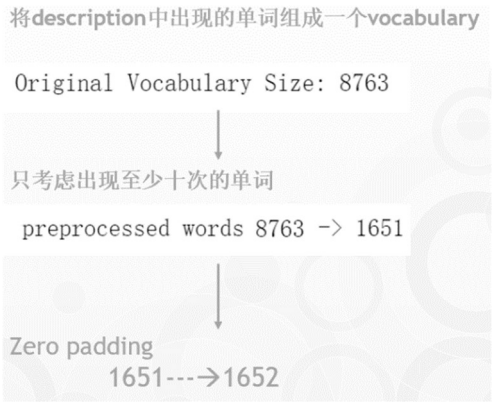



图 4.3 Unique words 统计

#### 1.3.3 特征向量的提取

运用 InceptionV3 模型将图片转换为一个固定长度（length=2018）的向量，使其可以作为输入到神经网路。

InceptionV3 原来是给图片分类的模型，由于我们的目标只是提取图片的特征向量，我们就移去了最后的 softmax 层，从倒数第二层中提取特征向量，如图 4.4 所示:

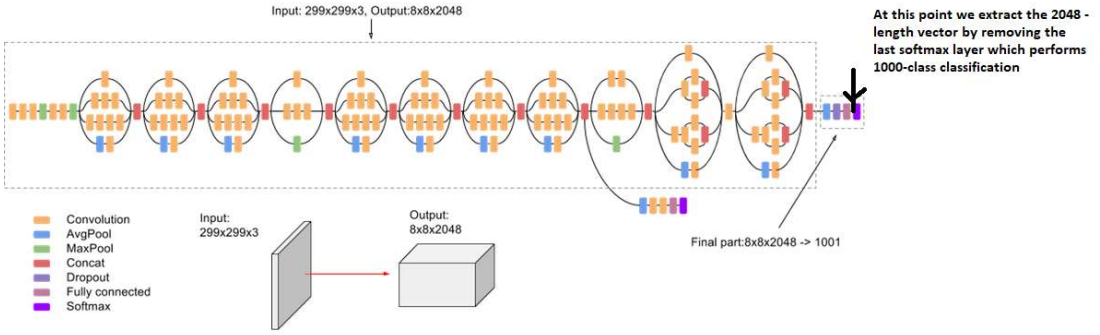



图 4.4 特征向量的提取

### 1.3.4 词编码

将每个 vocabulary 中的词编码为一个固定大小的向量，并创建两个 Pyhon 的 Dictionary，分别为 wordtoix['abc']：返回'abc'的索引；ixtoword[k]：返回索引为"k"的单词，每个单词的索引为 1-1652 的其中一个整数。

### 1.3.5 计算长度

计算 caption 的最大长度：34

### 1.3.6 data matrix 的构建过程

（在此举一个例子以更好地阐释）

eg. 以两张训练图片一张测试图片组成，如图 4.5 所示：


图 4.5 实例

。将 image1、image2 转换为长度为 2048 地特征向量

。给清理过的 caption 加上头尾标志（startseq、endseq）

Caption_1 -> "startseq the black cat sat on grass endseq"

Caption_2 -> "startseq the white cat is walking on road endseq"

vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}

。给 vocabulary 中的单词分配整数索引

black -1, cat -2, endseq -3, grass -4, is -5, on -6, road -7, sat -8, startseq -9, the -10, walking -11, white -12

。为了预测 caption 中的第 t+1 个单词，我们可以通过前 t 个单词组成的部分的 caption 和图片的特征向量来进行。预测 caption 从 startseq 开始直到 endseq 结束，如图 4.6 所示：

| | Xi | | Yi |
|---|---|---|---|
| i | Image feature vector | Partial Caption | Target word |
| 1 | Image_1 | startseq | the |
| 2 | Image_1 | startseq the | black |
| 3 | Image_1 | startseq the black | cat |
| 4 | Image_1 | startseq the black cat | sat |
| 5 | Image_1 | startseq the black cat sat | on |
| 6 | Image_1 | startseq the black cat sat on | grass |
| 7 | Image_1 | startseq the black cat sat on grass | endseq |
| 8 | Image_2 | startseq | the |
| 9 | Image_2 | startseq the | white |
| 10 | Image_2 | startseq the white | cat |
| 11 | Image_2 | startseq the white cat | is |
| 12 | Image_2 | startseq the white cat is | walking |
| 13 | Image_2 | startseq the white cat is walking | on |
| 14 | Image_2 | startseq the white cat is walking on | road |
| 15 | Image_2 | startseq the white cat is walking on road | endseq |

图 4.6 单词依次预测以组成 caption

将每个单词以索引来表示，效果如 4.7 所示：

| | Xi | | Yi |
|---|---|---|---|
| i | Image feature vector | Partial Caption | Target word |
| 1 | Image_1 | [9] | 10 |
| 2 | Image_1 | [9, 10] | 1 |
| 3 | Image_1 | [9, 10, 1] | 2 |
| 4 | Image_1 | [9, 10, 1, 2] | 8 |
| 5 | Image_1 | [9, 10, 1, 2, 8] | 6 |
| 6 | Image_1 | [9, 10, 1, 2, 8, 6] | 4 |
| 7 | Image_1 | [9, 10, 1, 2, 8, 6, 4] | 3 |
| 8 | Image_2 | [9] | 10 |
| 9 | Image_2 | [9, 10] | 12 |
| 10 | Image_2 | [9, 10, 12] | 2 |
| 11 | Image_2 | [9, 10, 12, 2] | 5 |
| 12 | Image_2 | [9, 10, 12, 2, 5] | 11 |
| 13 | Image_2 | [9, 10, 12, 2, 5, 11] | 6 |
| 14 | Image_2 | [9, 10, 12, 2, 5, 11, 6] | 7 |
| 15 | Image_2 | [9, 10, 12, 2, 5, 11, 6, 7] | 3 |

图 4.7 将 partial caption 用索引表示

将 caption 补全为同一长度，统一的长度即为之前计算出的 caption 最大数 34，补全的元素为 0，即所谓的 0 padding，效果如 4.8 所示：

| | Xi | | Yi |
|---|---|---|---|
| i | Image feature vector | Partial Caption | Target word |
| 1 | Image_1 | [9, 0, 0 …., 0] | 10 |
| 2 | Image_1 | [9, 10, 0, 0 …., 0] | 1 |
| 3 | Image_1 | [9, 10, 1, 0, 0 …., 0] | 2 |
| 4 | Image_1 | [9, 10, 1, 2, 0, 0 …., 0] | 8 |
| 5 | Image_1 | [9, 10, 1, 2, 8, 0, 0 …., 0] | 6 |
| 6 | Image_1 | [9, 10, 1, 2, 8, 6, 0, 0 …., 0] | 4 |
| 7 | Image_1 | [9, 10, 1, 2, 8, 6, 4, 0, 0 …., 0] | 3 |
| 8 | Image_2 | [9, 0, 0 …., 0] | 10 |
| 9 | Image_2 | [9, 10, 0, 0 …., 0] | 12 |
| 10 | Image_2 | [9, 10, 12, 0, 0 …., 0] | 2 |
| 11 | Image_2 | [9, 10, 12, 2, 0, 0 …., 0] | 5 |
| 12 | Image_2 | [9, 10, 12, 2, 5, 0, 0 …., 0] | 11 |
| 13 | Image_2 | [9, 10, 12, 2, 5, 11, 0, 0 …., 0] | 6 |
| 14 | Image_2 | [9, 10, 12, 2, 5, 11, 6, 0, 0 …., 0] | 7 |
| 15 | Image_2 | [9, 10, 12, 2, 5, 11, 6, 7, 0, 0 …., 0] | 3 |

图 4.8 zero padding

**1.4 模型搭建**

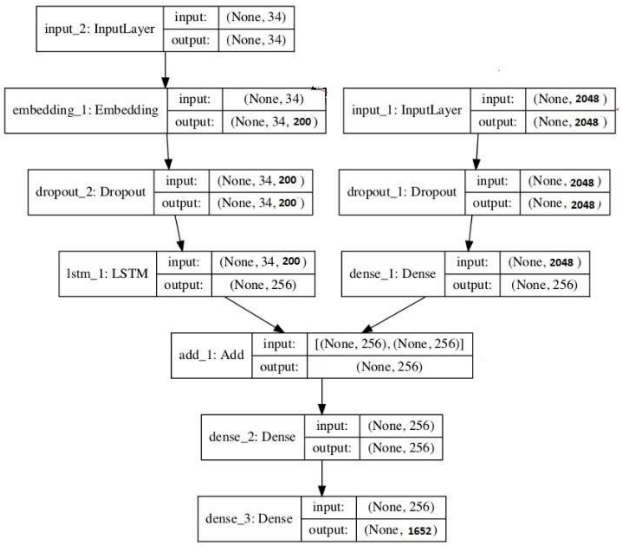


图 5.1 ， 模型的流程思路

如图 5.1 所示，我们希望以 partial caption 和图片的特征向量为输入，因此起初会有两个 tensor。首先 partial caption 经过预处理得到长度为 34 的向量后经过一个 embedding 层，把每个单词都映射到一个长度为 200 的向量，经过一层 Dropout 防止过拟合，之后经过一层 LSTM（选择 LSTM 的原因：LSTM 在自然语言的处理中能发挥不错的作用，并且相比普通的 RNN，LSTM 在更长的序列中有更好的表现）得到一个（batch_size，256）的输出。

同时，图片的特征向量经过一层 Dropout 防止过拟合，之后再经过一层全连接层同样得到一个（batch_size，256）的输出。

我们把两个格式相同的 tensor 合为一个，以便更好的训练得出最终结果，之后再经过一个全连接层后，经最后一层 softmax 层，产生涵盖 1652 个在 vocabulary 出现的单词的概率分布，基于 greedy search，概率分布最大的单词即我们要选择的输出单词。具体实例如图 5.2 所示

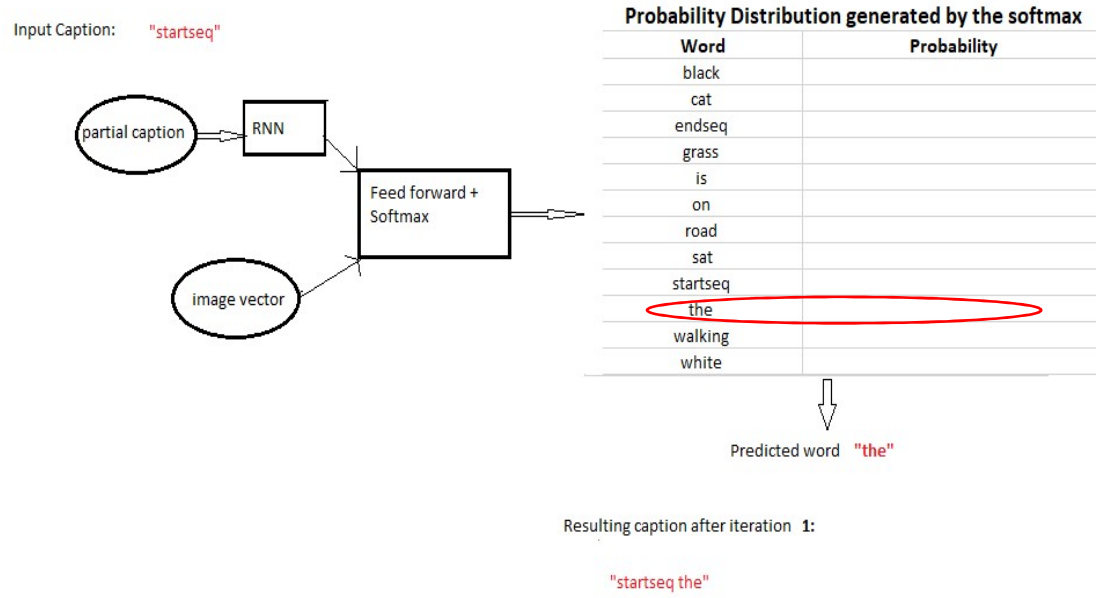



图 5.2 迭代的具体实现实例

迭代循环的终止条件有两个：

（1）以“endseq“结尾，模型认为 caption 已经完成

（2）句长大于 34，为了避免一直迭代下去，强制终止

**1.5 模型的训练**

我们训练这个模型设定了 epoch 为 30，前 20 个 epoch 的学习率设为 0.001，batch size 设为 3。当完成了 20 次迭代后，将学习率降为 0.0001 并且将 batch size 设为 6。用这些超参数的原因是因为当训练到达后半程时，模型逐渐趋向平缓，我们必须减小学习率才能在最低点边缩小步长，以趋近最低点。并且，适当的增加 batch size 使梯度的更新更加有效。

**1.6 模型评估**

本模型的预测结果使用 BLEU 进行预测。

BLEU 能作为机器翻译的一个评估指。它采用了 N-gram 的匹配规则，能够算出比较译文和参考译文之间 n 组词的相似的一个占比。随着 n-gram 的增大，总体的精度得分是呈指数下降的，所以一般 N-gram 最多取到 4-gram。

一般情况 1-gram 可以代表原文有多少词被单独翻译出来，可以反映译文的充分性，2-gram 以上可以反映译文的流畅性，它的值越高说明可读性越好。这两个指标是能够跟人工

评价对标的。

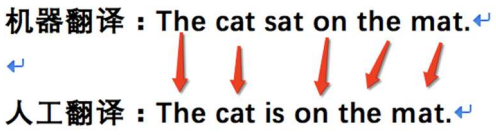



图 5.3.1　1-gram 准确度（该例为 5/6）

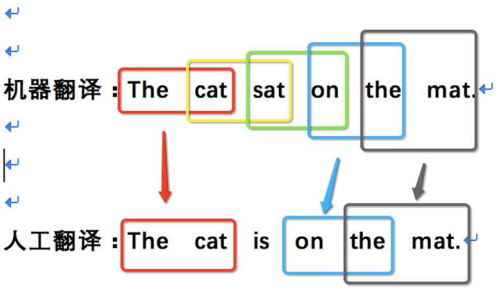



图 5.3.2　2-gram 准确度（该例为 3/5）

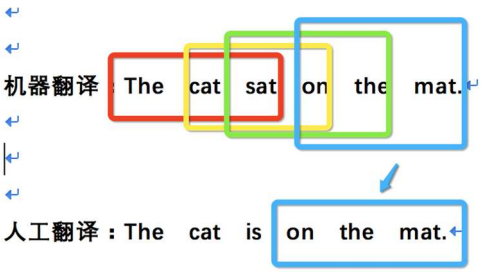



图 5.3.3　3-gram 准确度（该例为 1/4）

N-gram 的一个弊端是其译文准确度的匹配关系不能很好地体现译文长度不准确的问题。因此，针对翻译译文长度比参考译文要短的情况，就需要一个惩罚的机制去控制。在此便引入了惩罚因子的概念。惩罚因子的计算公式如下:

$$\mathrm{BP}=\begin{cases}1 & \text{if } c>r\\ e^{(1-r/c)} & \text{if } c\le r\end{cases}.$$

图 5.4 惩罚因子 BP 的计算

C 是测试译文的词数，r 是参考译文的词数

BLEU 算法就是在这两个概念的基础上整合得到，其计算公式如图 5.5 所示，BLEU 值越大表示测试译文与参考译文越接近，反之则差别越大。

$$\mathrm{BLEU}=\mathrm{BP}\cdot\exp\left(\sum_{n=1}^{N}w_n\log p_n\right).$$

图 5.5 BLEU 算法的计算公式

经过分析，我们发现 BLEU 尽管在一定程度上可以作为测试出的 caption 和原 caption 的评估指标，也比较方便和快捷，但它无法考虑语法上的准确性，测评的精度也会收到常用词的干扰。同时 BLEU 无法考虑同义词或相似表达的情况，因此作为该实验的评估指标还是存在一定的缺陷。

## 1.7 测试结果

**较理想的测试结果：**

Greedy: group of dogs are running through the snow

predict: black dog is running through field of grass

predict: small child in red coat is climbing frozen snowy mountain

predict: girl in pink swimsuit is laying on the beach

图 6.1 比较理想的测试结果

**不太理想的测试结果**

predict: two boys playing soccer on field

predict: woman in red shirt is jogging along path

predict: man in red shirt is kneeling on motocross motorbike

predict: two girls are sitting on the carpet with view of the and the

6.2 不太理想的测试结果

**结果分析：**

针对实验结果，我们发现测试样例中有匹配度高的 caption，也有结果不太理想的测试结果，分析后我们认为：部分测试对图片中的数目、颜色或人物性别的检测出现差错，可能是特征提取的弊端造成的，因此未来优化时可以采取更好的特征提取模型（如 vgg 等）；此外由于该数据集不够大，对模型的训练并不能达到很理想的效果，因此我们需要更大的数据集去训练它（实际操作中，我们有尝试用更大的数据集，但因为 CPU 的限制，最终没有跑出来）；针对优化，我们还提出了可以调整超参数优化模型训练、用交叉验证集避免过拟合、并寻找比 BLEU 更好的检测结果的方法等。

## 2、微表情识别

### 2.1 数据集

#### 1.1.1 来源

– *Human Images Source-CK+: http://www.consortium.ri.cmu.edu/ckagree/*

– *Human Images Source-Kaggle Dataset: https://www.kaggle.com/jonathanoheix/face-expression-recognition-dataset*

– *Human Images Source-Jaffebase Dataset: http://www.kasrl.org/jaffe.html*

– *Animated Images Source-FERG_DB: https://grail.cs.washington.edu/projects/deepexpr/ferg-db.html*

#### 1.1.2 特点展示

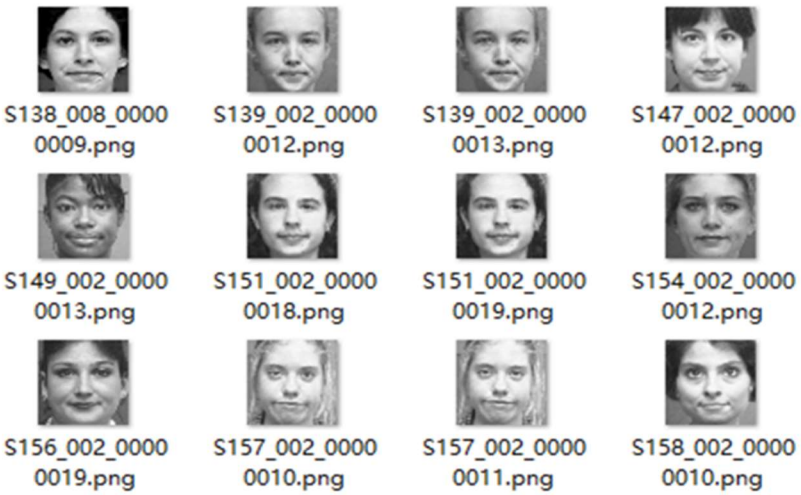
（1）CK+ Dataset:


。最左边的每个文件夹对应一个人；

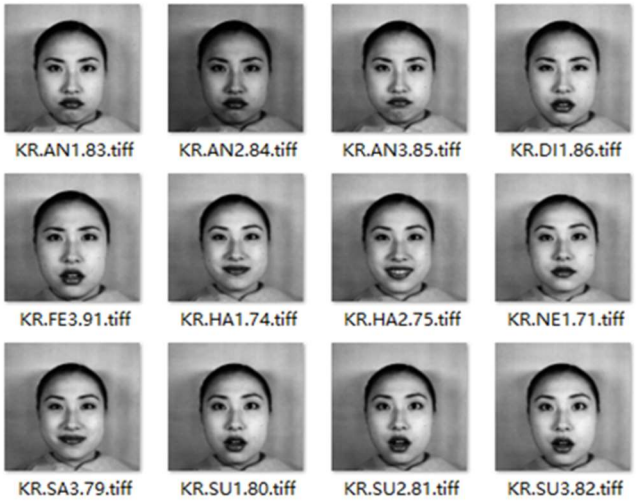
。中间的每个文件夹对应一个人的一组表情序列图片；

。最右边的一组图片记录的是从面无表情到表情饱满的过程：见下图

（2）Kaggle Dataset：已经划分好 train 和 validation 集合，见下图：




（3）Jaffebase Dataset：文件夹中所有的图片都可以通过文件名字划分表情类别，见下图




（4）FERG_DB Dataset：动漫表情，且图片已经按照表情类别分类，见下图：

## 2.2 实验思路

（1）目的：对输入的图像检测人脸，并且基于人类的七种基本表情，对输入图像的人脸进行情绪识别

（2）将人类和动漫表情划分为 7 种不同的类别，划分后每一种表情都有两个文件夹，分别人类和动漫表情，所以总计一共有 14 个文件夹；

（3）将每一个文件夹中的文件读入成为一个 DataFrame，总计 14 个 DataFrame，将所有人类表情的 DataFrame 结合起来，将所有的动漫表情的 DataFrame 结合起来，所有表情总计 1 万多张

（4）最后得到两个 DataFrame，一个是人类表情，一个是动漫表情。

（5）七种表情分别是：

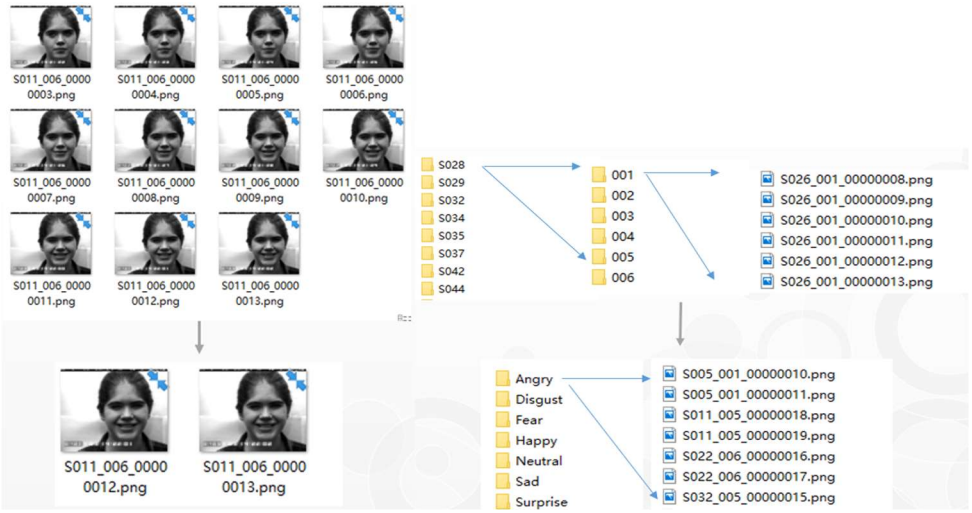
-ANGER -DISGUST -FEAR -HAPPY -NEUTRAL -SAD -SURPRISE

## 2.3 实验过程

### 2.3.1 数据集准备

– CK+ Dataset:

• 处理表情序列，从序列中选择有表情的图片。并运用 linux 脚本将表情按照标签分为 7 类，因为 ck+数据集下载下来每一个文件夹都有对应的表情标签，按照标签给每个表情文件夹分类，见下图



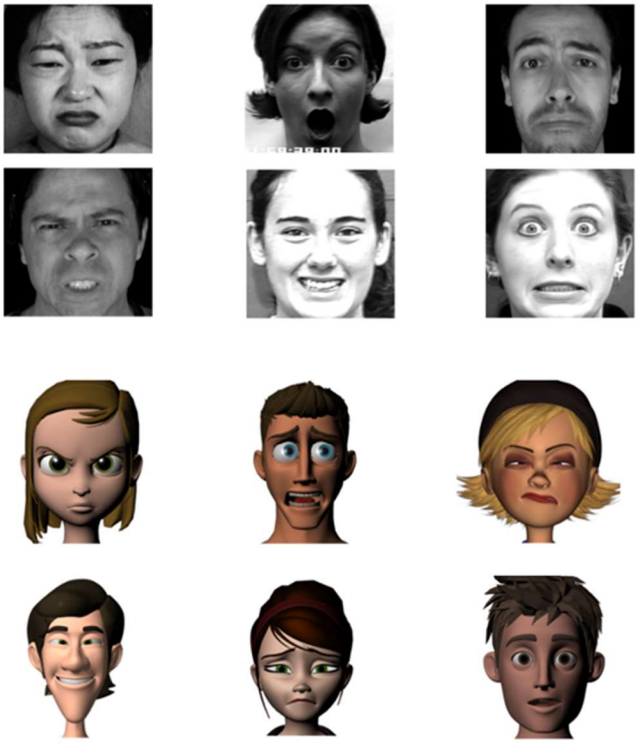

– Jaffebase Dataset

• 从文件名中提取所属类别：


– 准备后的结果：

- 14 folders：7 human image folders and 7 animated image folders

- 1w+ images in total



### 2.3.2 数据预处理

– Crop and Resize

（1）Convert to gray-scale

（2）Detect face： OpenCV HAAR Cascade.

（3）Crop the image to the face.

（4）Resize the image to 350*350.

（5）Save the image.

之所以将图片都转化为灰度是因为我们的图片一部分是灰度，一部分是彩色，而考虑到颜色并不影响表情的识别，所以为了保持一致性，我们将所有图片都转化为灰度图片。

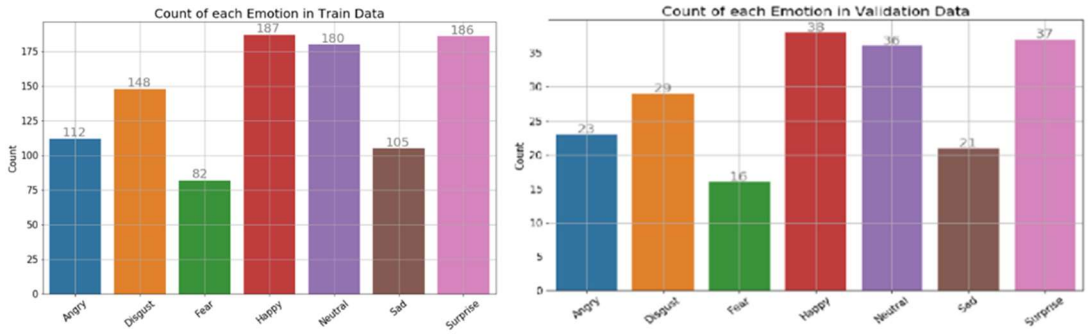
过程见下图：

– Train-Test-Split:

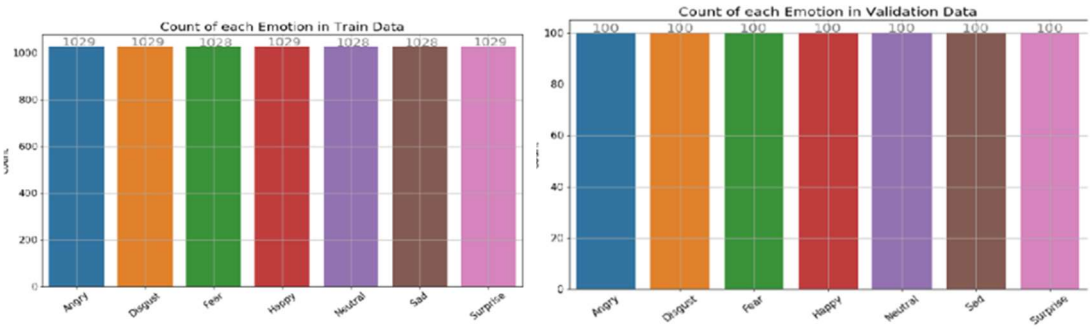
train：validation：test：5：2：3

对于人类和动漫表情图片，我们划分训练集-检验集-测试集

- Human images distribution:




- Animated images distribution:




- 处理好后得到 6 个 set【3 个为 Human images 的 train-validation-test set，3 个为 Animated images 的 train-validation-test set】，将 Human images 的 train set 和 Animated images 的 train set 合并为 Combined train set，结果得到 5 个 DataFrames。

其中 Combined train set 用于训练模型，Human images 的 Validation set 和 Animated images 的 Validation set 用于 cross validation（交叉检验）。

**-第一个模型：借助 VGG16**

**2.3.3-1 模型搭建**

- 使用 pre-trained model 的原因:

（1）对于图像处理我们没有足够的计算能力和足够的图片，卷积本身的运算时间花费不小，所以为了减少时间开销，我们决定 transfer learning。

（2）利用 Transfer learning 的概念，我们将其他 pre-trained model 得到的参数转移到我们的数据中，这样我们将数据传递给这些模型，提取图像的特征（bottleneck_features）。

（3）VGG16 神经网络以数百万幅图像的数据集为基础进行了训练。VGG16 包含 16 层，其中 13 层是卷积层。我们利用 VGG16 作为 pre-trained model 生成 bottleneck_features。

- 从 VGG16 中提取 bottleneck_features

```
model = VGG16(weights='imagenet', include_top=False)
```

| Layer | | Feature Map | Size | Kernel Size | Stride | Activation |
|---|---|---|---|---|---|---|
| Input | Image | 1 | 224 x 224 x 3 | - | - | - |
| 1 | 2 X Convolution | 64 | 224 x 224 x 64 | 3x3 | 1 | relu |
| | Max Pooling | 64 | 112 x 112 x 64 | 3x3 | 2 | relu |
| 3 | 2 X Convolution | 128 | 112 x 112 x 128 | 3x3 | 1 | relu |
| | Max Pooling | 128 | 56 x 56 x 128 | 3x3 | 2 | relu |
| 5 | 2 X Convolution | 256 | 56 x 56 x 256 | 3x3 | 1 | relu |
| | Max Pooling | 256 | 28 x 28 x 256 | 3x3 | 2 | relu |
| 7 | 3 X Convolution | 512 | 28 x 28 x 512 | 3x3 | 1 | relu |
| | Max Pooling | 512 | 14 x 14 x 512 | 3x3 | 2 | relu |
| 10 | 3 X Convolution | 512 | 14 x 14 x 512 | 3x3 | 1 | relu |
| | Max Pooling | 512 | 7 x 7 x 512 | 3x3 | 2 | relu |
| 13 | FC | - | 25088 | - | - | relu |
| 14 | FC | - | 4096 | - | - | relu |
| 15 | FC | - | 4096 | - | - | relu |
| Output | FC | - | 1000 | - | - | Softmax |

利用代码 model.predict()，我们把图片一张一张传入 VGG16 模型，得到 bottleneck_features 并且将其作为 numpy.array 存储，通过这种方式我们实现 transfer learning。

【只采用 VGG16 第 13 层前面的部分，第 12 层的输出作为 bottleneck_features】

- 搭建模型 MLP

在 VGG16 提取图片 bottleneck_features 的基础上，我们将这些特征传递给 MLP（MLP 作为顶级模型），然后利用 MLP 减少损失函数的值，并且更新 MLP 和 CNN 中的权重。

```
#model architecture
def model(input_shape):
  model = Sequential()

    model.add(Dense(512, activation='relu', input_dim
 = input_shape))
    model.add(Dropout(0.1))

    model.add(Dense(256, activation='relu'))

    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(64, activation='relu'))
    model.add(Dense(output_dim = no_of_classes, activ
ation='softmax'))

    return model
```

【借助 MLP，通过 backpropagation 降低 lostfunction 的值】

搭建 5 层全连接层，所有层都是用 relu 作为激活函数，第一层包含 512 个激活单元，第二层包含 256 个激活单元，第三层包含 128 个激活单元，第四层包含 64 个激活单元，第五层包含 7 个 softmax 单元，softmax 是用于多分类的逻辑回归。

该模型会生成 7 个概率值，这些概率值的和为 1，所有结果会传递给 cross-entropy 损失函数。

可以注意到 dropout rate 非常小，起初我们尝试了 0.3，但是在 15 个 epochs 之后训练集和检验集的 loss 值不会减小。在渐渐尝试减小 dropout 的过程中，确定 dropout rate 为 0.1 时，训练集和检验集的 loss 值会降低，同时准确度会提升

- 读取 bottleneck_features，放入 MLP

### 2.3.4-1 模型训练

训练 20 个 eopchs，结果如下

– Multi-Class Log-Loss

•Training Loss: 0.70 to 0.04

•CV Human loss: 2.25 to 0.05

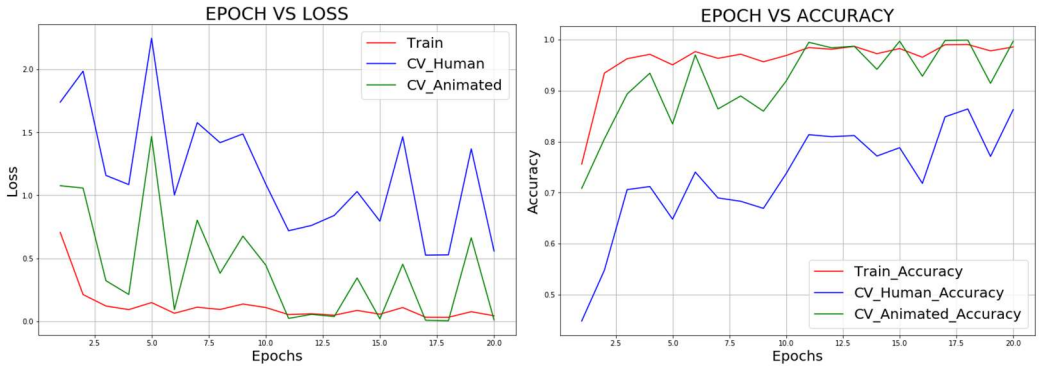
•CV Animated loss: 1.47 to 0.01

•Train Accuracy: 75% to 99%

•CV Human Accuracy: 44% to 86%

•CV Animated Accuracy: 70% to 99%

| | Epoch | Comb_Train_Loss | Comb_Train_Accuracy | CVHuman_Loss | CVHuman_Accuracy | CVAnime_Loss | CVAnime_Accuracy |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.704710 | 0.755769 | 1.739470 | 0.448077 | 1.075818 | 0.708173 |
| 1 | 2 | 0.211229 | 0.934375 | 1.985411 | 0.547837 | 1.056886 | 0.805288 |
| 2 | 3 | 0.120556 | 0.962500 | 1.157691 | 0.705769 | 0.321500 | 0.893269 |
| 3 | 4 | 0.091546 | 0.971154 | 1.084601 | 0.711779 | 0.212145 | 0.934135 |
| 4 | 5 | 0.147811 | 0.950481 | 2.246476 | 0.647837 | 1.466243 | 0.834615 |
| 5 | 6 | 0.062828 | 0.976442 | 1.001590 | 0.740144 | 0.092428 | 0.969952 |
| 6 | 7 | 0.111092 | 0.963221 | 1.576609 | 0.689423 | 0.801960 | 0.863942 |
| 7 | 8 | 0.092735 | 0.971394 | 1.417509 | 0.682933 | 0.380458 | 0.889423 |
| 8 | 9 | 0.136161 | 0.956490 | 1.487133 | 0.668990 | 0.675563 | 0.859615 |
| 9 | 10 | 0.108349 | 0.968750 | 1.089068 | 0.737019 | 0.445498 | 0.918510 |
| 10 | 11 | 0.052084 | 0.984375 | 0.718070 | 0.813462 | 0.022081 | 0.994712 |
| 11 | 12 | 0.058979 | 0.981010 | 0.759884 | 0.809615 | 0.054052 | 0.983894 |
| 12 | 13 | 0.047458 | 0.986779 | 0.839569 | 0.811779 | 0.036898 | 0.987019 |
| 13 | 14 | 0.085495 | 0.972356 | 1.029411 | 0.771635 | 0.343317 | 0.941587 |
| 14 | 15 | 0.055579 | 0.982452 | 0.793958 | 0.787981 | 0.018064 | 0.996875 |
| 15 | 16 | 0.108243 | 0.965385 | 1.464551 | 0.718029 | 0.452601 | 0.928125 |
| 16 | 17 | 0.031406 | 0.989904 | 0.524215 | 0.848558 | 0.007099 | 0.998317 |
| 17 | 18 | 0.030931 | 0.990385 | 0.526826 | 0.863942 | 0.003679 | 0.999038 |
| 18 | 19 | 0.075246 | 0.977885 | 1.368249 | 0.770913 | 0.662559 | 0.914183 |
| 19 | 20 | 0.042613 | 0.985577 | 0.556674 | 0.862740 | 0.009208 | 0.996875 |




### 2.3.5-1 模型评估

我们把人类和动漫表情图片分开，以便能够分别在这两个集合上面进行测验

– Confusion Metric

- Confusion and Recall Matrix Human Images

– In confusion metric:

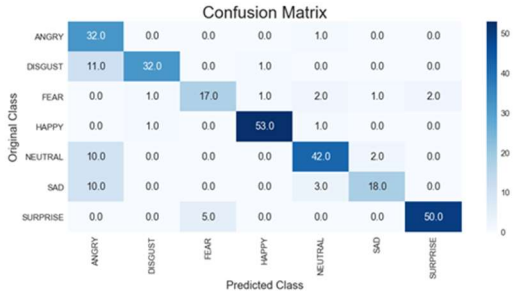
Biased on the “angry” class

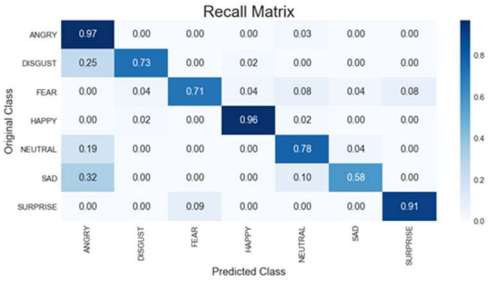
– In recall metric

Many images in “sad” class which are predicted in “angry”.

– Conclusion

Model is not completely tell “angry” and “sad”




• Confusion and Recall Matrix Animated Images

100% accuracy.

Why?

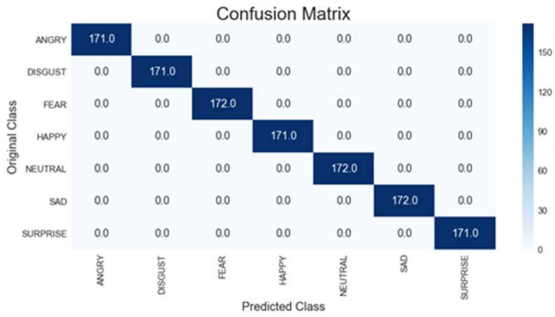
– The ratio of train images from animated to human is approximately 9:1.

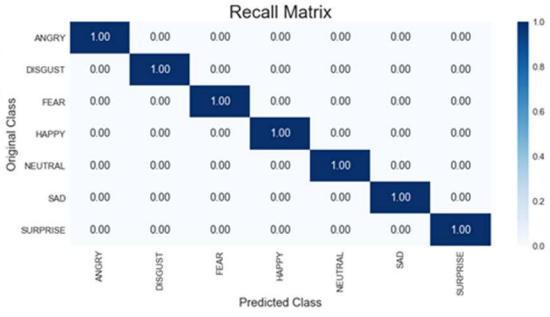
– Learning human features is hard as compared to animated images.

– The size of face

– expression angles

– etc




未来展望

– 微调 VGG16 的最后 2、3 层卷积层

– 寻找有更多 variance 的图片

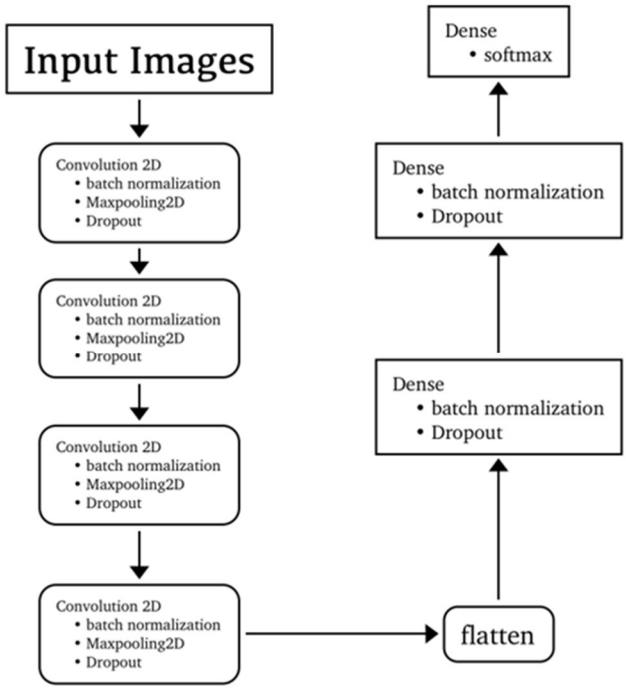
– 图片尺寸大于 400*400

– 一张图片中多个人脸时能够检测大部分人脸并识别情绪

-模型二

和模型一使用相同的经过预处理后划分好 train-validation-test 集合的数据

2.3.3-2 模型搭建




- ```
  from keras.layers import Dense, Input, Dropout, GlobalAvera
  gePooling2D, Flatten, Conv2D, BatchNormalization, Activatio
  n, MaxPooling2D
  from keras.models import Model, Sequential
  from keras.optimizers import Adam

  # number of possible label values
  nb_classes = 7

  # Initialising the CNN
  model = Sequential()

  # 1 - Convolution
  model.add(Conv2D(64,(3,3), padding='same', input_shape=(4
  8, 48,1)))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.25))

  # 2nd Convolution layer
  model.add(Conv2D(128,(5,5), padding='same'))
  model.add(BatchNormalization())
  ```
  ```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flattening
model.add(Flatten())

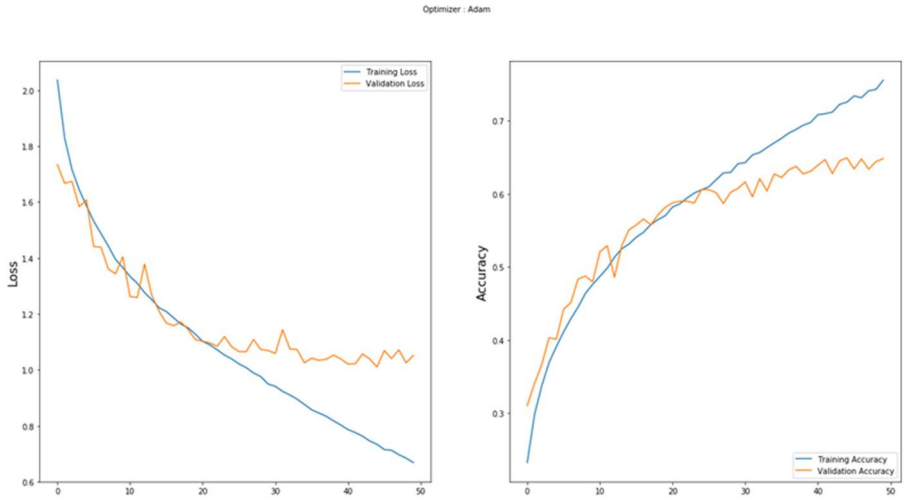
# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(nb_classes, activation='softmax'))

opt = Adam(lr=0.0001)
model.compile(optimizer=opt, loss='categorical_crossentrop
y', metrics=['accuracy'])
```

### 2.3.4-2 模型评估




### 2.3.6 模型测试

测试案例 1:

```
make_prediction("../Data/Test_Images/Surprise_3.jpg")
```

```
Predicted Expression Probabilities
ANGRY: 7.466517854481936e-05
DISGUST: 8.049021005263057e-09
FEAR: 0.0035154588986188173
HAPPY: 1.8760174498311244e-06
NEUTRAL: 0.00012389293988235295
SAD: 1.6487249013152905e-05
SURPRISE: 0.9962676167488098

Dominant Probability = SURPRISE: 0.9962676
```

测试案例 2:

```
Image.open("../Data/Test_Images/Disgust_1.jpg")
```

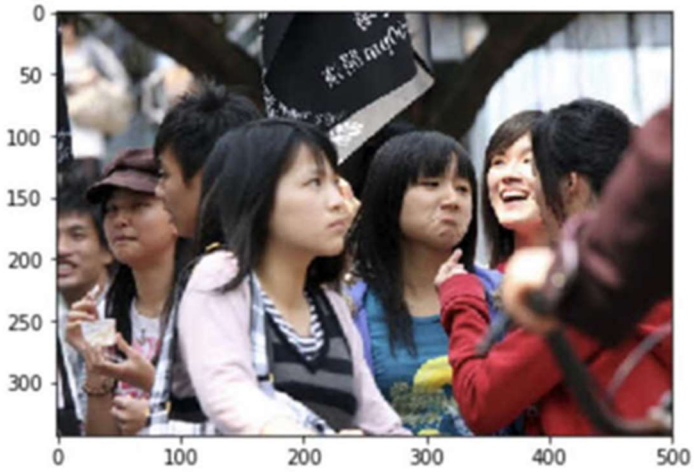
```
make_prediction("../Data/Test_Images/Disgust_1.jpg")
```

```
Predicted Expression Probabilities
ANGRY: 0.6820868849754333
DISGUST: 0.3178218901157379
FEAR: 1.021714297166909e-06
HAPPY: 2.2017935407347977e-05
NEUTRAL: 5.1129391067661345e-06
SAD: 6.297716026892886e-05
SURPRISE: 8.2167117554377e-10

Dominant Probability = ANGRY: 0.6820869
```

## 3、模型整合

我们把每个训练出的模型进行接口的匹配连接。最终呈现出：先将图片输入看图说话的模型分析出图片的 captio，将 caption 存储下来。然后将图片再输入微表情识别的模型，若在模型中检测到人脸，就对人脸的表情进行分析得到情绪，最终将情绪和 caption 进行匹配。最终输出图片和对应的 caption 和 emotion（若检测不到人脸则不输出 emotion）示例如下:




predict: woman in pink costume is talking to woman in crowd
Emotion is Sad


predict: girl in pink swimsuit is laying on the beach
Emotion is Happy