

验证码识别文档说明

1 生成数据 -----Create_captcha.py

导入 `from captcha.image import ImageCaptcha`, 利用 `ImageCaptcha` 可以将文本变成验证码图片, 并且实例 `PIL` 库可以识别打开和识别输入的图像:

`PIL.Image.Open(fp, mode='r')` ----- 打开和识别输入的图像

`captcha.image.ImageCaptcha(width,height)` ----- 创建 `ImageCaptcha` 实例

`captcha.image.ImageCaptcha.write('1234','1234.png')` ----- 生成验证码图片并保存

`captcha.image.ImageCaptcha.generate('1234')` -----仅生成验证码图片, 不保存

步骤: ①建 `ImageCaptcha` 实例 `captcha`

②生成随机字符串 `text`

③利用 `captcha.write(text,filedir)`将字符文本变成验证码图片, 并保存在 `filedir` 文件中, 注意的是该方法 `overview` 相同的验证码, 所以实际上产生的验证码图片会比预期的要少。



2 输入输出数据的处理----- Captcha_recognize.py

2.1 输入数据处理

2.1.1 图像的处理: RGB 图像→灰度图→规范化数据



①其中 `RGB 图像→灰度图`的方法很多, 这里我用到的是 `wiki` 上的一个方法: $0.299R+0.587G+0.14B$, 实现代码见 `rgb2grey` 方法;

```
代码:
def rgb2gray(image):
    """将RGB验证码图像转为灰度图"""
    # 这里一种方法 0.299R+0.587G+0.14B
    return np.dot(image[..., :3], [0.299, 0.587, 0.114])
```

②灰度图→规范化数据: `x_train/255`

2.1.2 适配 Keras 图像数据格式:

有两种格式: `'channels_first'` 和 `'channels_last'`(默认)

```

代码:
def fit_keras_channels(batch, rows=CAPTCHA_HEIGHT, cols=CAPYCHA_WIDTH):
    """适配Keras 图像数据格式"""
    if K.image_data_format() == 'channels_first':
        batch = batch.reshape(batch.shape[0], 1, rows, cols)
        input_shape = (1, rows, cols)
    else: # keras 默认的是 channels_last
        batch = batch.reshape(batch.shape[0], rows, cols, 1)
        input_shape = (rows, cols, 1)
    return batch, input_shape


```

2.2 输出数据处理

2.2.1 one-hot 编码

验证码→向量, 和我们平常说的 one-hot 可能不太一样, 这里的验证码涉及 4 个字符, 每个字符进行 one-hot 编码后进行拼接, 维度就变成(40,), 训练集的输出也需要这样编码。

Label: 6046



→

```

array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 0., 0.])

```

```

代码:
def text2vec(text, length=CAPTCHA_LEN, charset=CAPTCHA_CHARSET):
    """对验证码中的每个字符进行 one-hot 编码"""
    text_len = len(text)
    if text_len != length:
        raise ValueError('Error:length od captcha should be {}, but got {}'.format(length, text_len))
    vec = np.zeros(length * len(charset))
    for i in range(length):
        # one-hot 编码验证码中的每个数字, 每个字符的编码 = 索引 + 偏移量
        vec[charset.index(text[i]) + i * len(charset)] = 1
    return vec

```

2.2.1 解码


模型输出向量→验证码

```

array([[2.0792404e-10, 4.3756086e-07, 3.1140310e-10, 9.9823320e-01,
        5.1135743e-15, 3.7417038e-05, 1.0556480e-08, 9.0933057e-13,
        2.7572466e-07, 1.7206760e-03, 1.1030550e-07, 1.1852034e-07,
        7.9457263e-10, 3.4533365e-09, 6.6065012e-14, 2.8996323e-05,
        7.6345085e-13, 3.1817032e-16, 3.9540555e-05, 9.9993122e-01,
        5.3814397e-13, 1.2061575e-10, 1.6408040e-03, 9.9833637e-01,
        6.5149628e-08, 5.2246549e-12, 1.1365444e-08, 9.5700288e-12,
        2.2725430e-05, 5.2195204e-10, 3.2457771e-13, 2.1413280e-07,
        7.3547295e-14, 4.4094882e-06, 3.8390007e-07, 9.9230206e-01,
        6.4467136e-03, 3.9224533e-11, 1.2461344e-03, 1.1253484e-07]],
      dtype=np.float32)

```

argmax



Label: 3935

```

代码:
vet2text(vec):
    """将验证码向量解码成对应的字符"""
    if not isinstance(vec, np.ndarray):

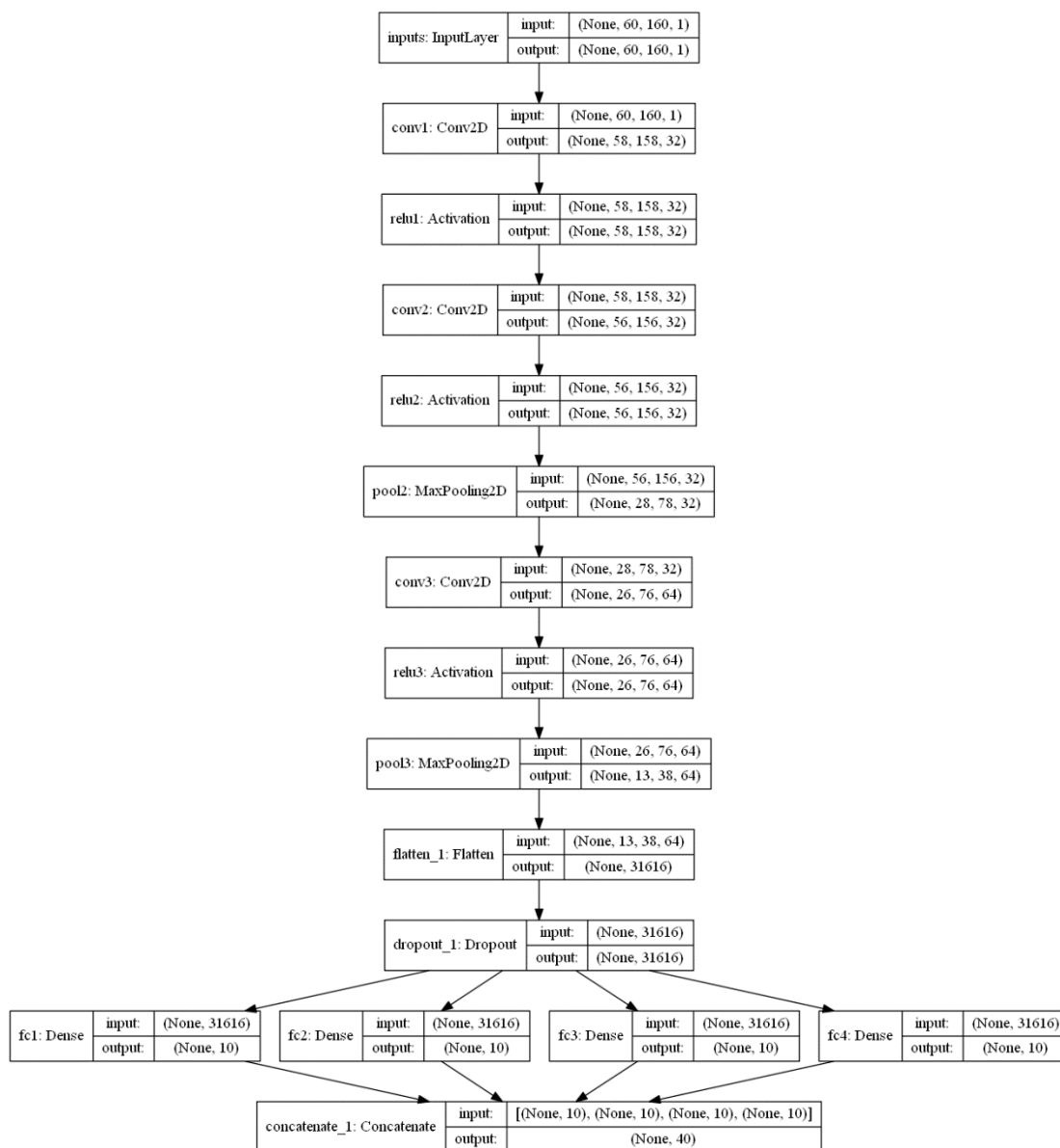
```

```

vec = np.asarray(vec)
vec = np.reshape(vec, [CAPTCHA_LEN, -1])
text = ''
for item in vec:
    text += CAPTCHA_CHARSET[np.argmax(item)]
return text

```

3 分类模型



这里用到了 keras 的可视化工具：

```
from keras.utils import plot_model
```

使用的时候只要指定模型和输出的文件即可

```
plot_model(model, to_file='model.png')
```

Note:但是在 Keras 框架的可视化代码时候，可能遇到下面的问题：

OSError: `pydot` failed to call GraphViz.Please install GraphViz (<https://www.graphviz.org/>) and ensure that its executables are in the \$PATH.

解决方法:

```
pip install pydot-ng
pip install graphviz
```

```
pip install pydot==1.2.3(这个很关键, 指定版本)
```

然后到错误指定的网站(<https://www.graphviz.org/>)下载安装 graphviz-2.3.8.msi 安装方式很简单, 双击后逐步选择 next, 在选择安装路径的时候推荐安装到 D 盘, 我这里安装到(D:\Program Files (x86)\Graphviz2.38\bin)

然后将路径 (D:\Program Files (x86)\Graphviz2.38\bin) 添加环境变量 Path 中即可。

其实模型时借鉴 AlexNet 的最后几个 FC 层, 所以这里设计了 4 个 FC 层, 我们可以理解验证码中的每个字符对应一个 FC 层, 如果是 6 个验证码字符, 那么就会中 6 个 FC 层, 这样应该好理解。

4 模型分析

4.1 损失函数选择

categorical_crossentropy 和 binary_crossentropy 是两种常用的交叉熵损失函数, 下面是这两种损失函数的区别:

- ① categorical_crossentropy: 常用于 One-hot 向量的 **多类别分类模型 (Multi-Class Classification)**;
- ② binary_crossentropy: 对于每个向量分量都是独立的, 也就是说每个向量分量计算的损失不受其他分量的影响, 常用于 **多标签分类模型 (Multi-Label Classification)**。

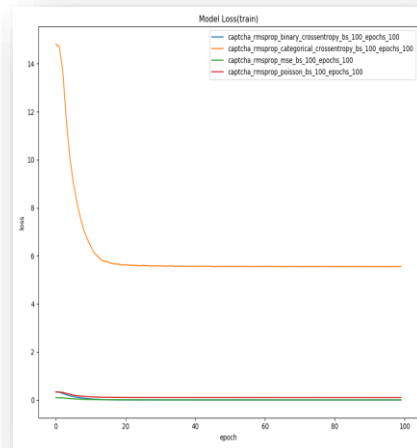
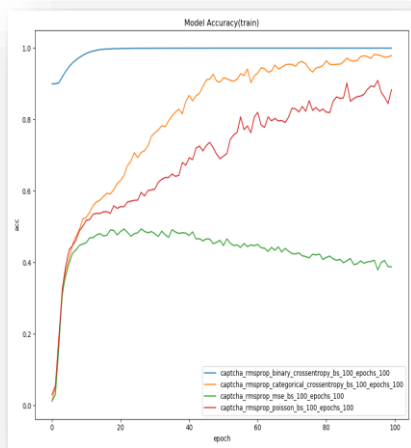
对于验证码识别的模型, 我们可以认为这是个多标签分类的模型, 采用 binary_crossentropy 交叉熵作为损失函数。

4.2 不同损失函数之间的对比

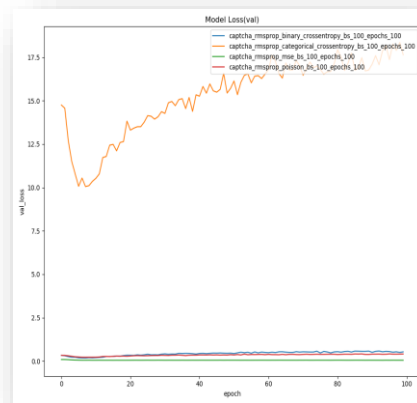
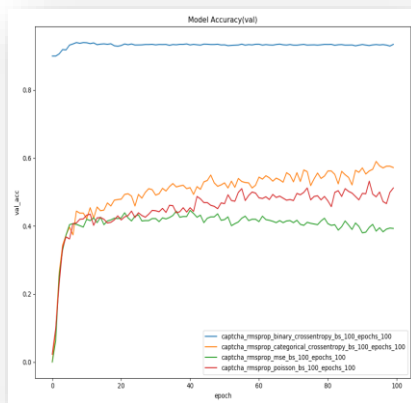
保持优化器不变, 改变损失函数, 选择了 4 个损失函数进行对比:

```
captcha_rmsprop_binary_crossentropy_bs_100_epochs_100
captcha_rmsprop_categorical_crossentropy_bs_100_epochs_100
captcha_rmsprop_mse_bs_100_epochs_100
captcha_rmsprop_poisson_bs_100_epochs_100
```

下面是不同损失函数下**训练集**的准确率和损失值的变化:



下面是不同损失函数下测试集的准确率和损失值的变化：

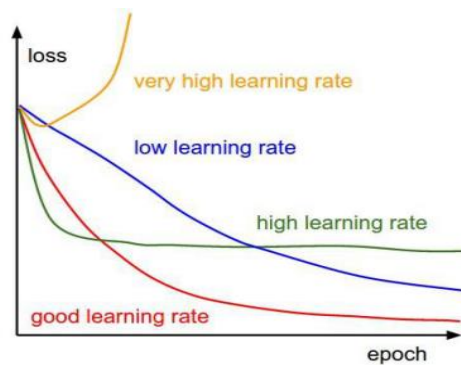


所有综合来看 binary_crossentropy 交叉熵损失函数更适合这里的验证码识别模型

4.2 优化器的选择

4.2.1 Learning Rate

介绍优化器之前，我们先来看一下学习率和损失值变化（模型的收敛速度）的关系：



1、何时加大学习率：

- 训练初期，损失值一直都没什么波动；

2、何时减少学习率：

- 训练初期，损失值爆炸增长
- 损失值先开始速降，后保持平稳状态
- 训练后期，损失值反复上下波动

4.2.2 不同优化器对比

这里选择了 4 种不同的优化器进行对比

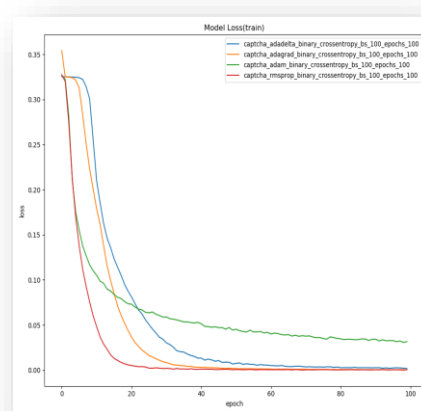
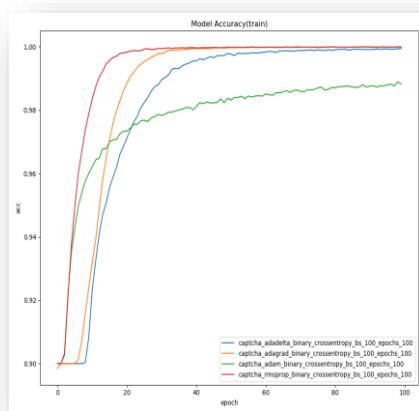
captcha_adadelta_binary_crossentropy_bs_100_epochs_100

captcha_adagrad_binary_crossentropy_bs_100_epochs_100

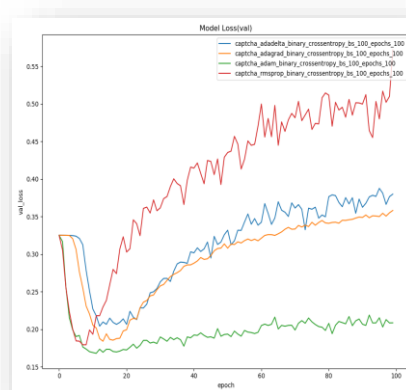
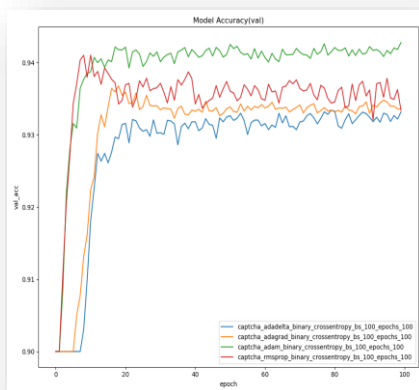
captcha_adam_binary_crossentropy_bs_100_epochs_100

captcha_rmsprop_binary_crossentropy_bs_100_epochs_100

下面是不同优化器下**训练集**的准确率和损失值的变化：



下面那是不同优化器下**测试集**的准确率和损失值的变化：



通过上面的可视化我们发现 Adam 优化更好。