

# 实战三：Quick, Draw! Google涂鸦识别挑战项目(下)

导师：GAUSS

---



# 目录

1/ 数据处理

2/ 建模方法

3/ baseline构建

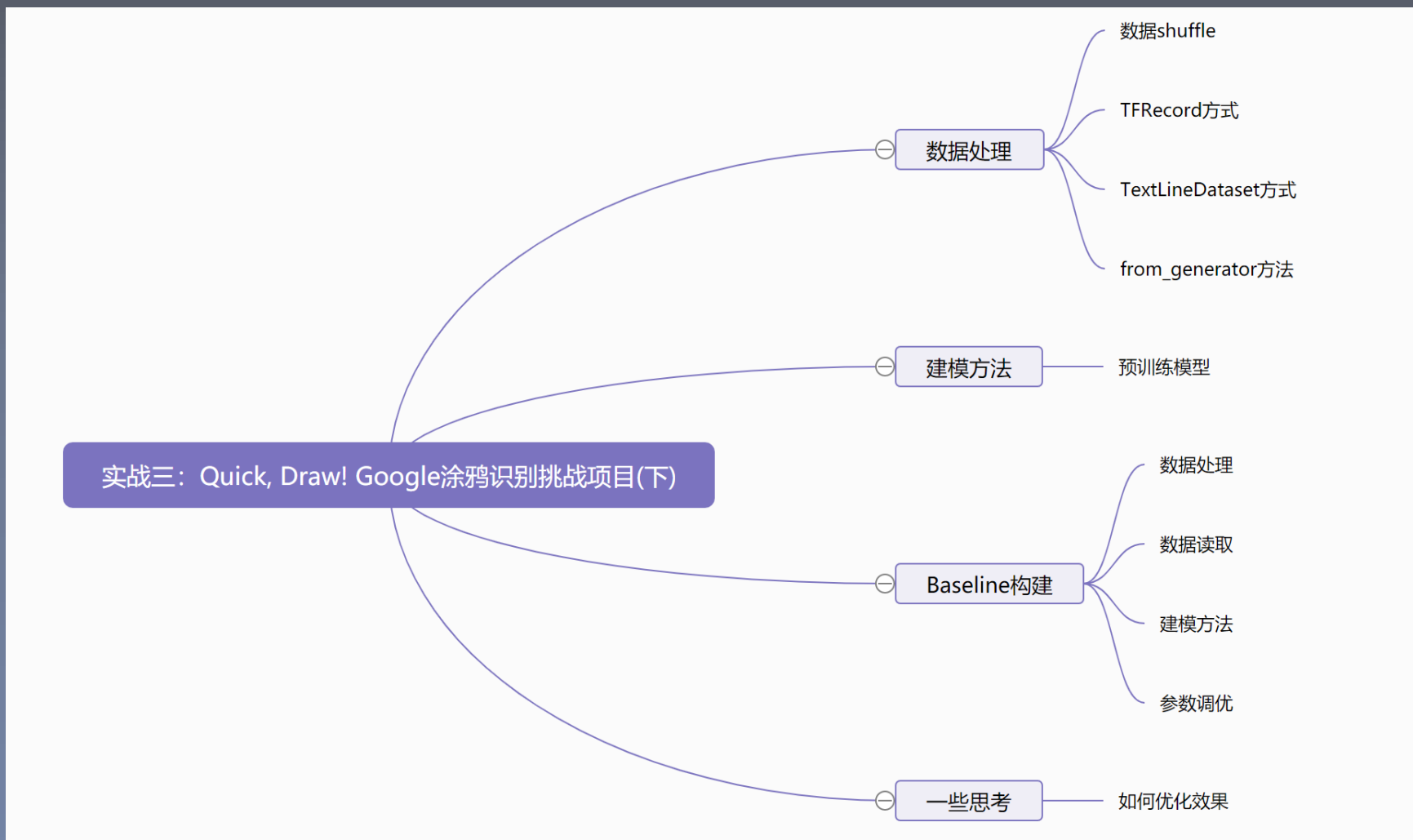


# 知识树

## Knowledge tree



深度之眼  
deepshare.net





# 数据处理

---





















# 数据shuffle

340个csv文件，需要将数据打乱，以便小批量数据读取！

如果电脑内存不够，可以考虑分批写入数据。

```
s = Simplified('./data/')
NCSVS = 100
categories = s.list_all_categories()
print(len(categories))

for y, cat in tqdm(enumerate(categories)):
    df = s.read_training_csv(cat)
    df['y'] = y
    df['cv'] = (df.key_id // 10 ** 7) % NCSVS
    for k in range(NCSVS):
        filename = './shuffle_data/train_k{}.csv'.format(k)
        chunk = df[df.cv == k]
        chunk = chunk.drop(['key_id'], axis=1)
        if y == 0:
            chunk.to_csv(filename, index=False)
        else:
            chunk.to_csv(filename, mode='a', header=False, index=False)
```

Name	Last Modified
 train_k0.csv.gz	5 days ago
 train_k1.csv.gz	5 days ago
 train_k10.csv.gz	5 days ago
 train_k11.csv.gz	5 days ago
 train_k12.csv.gz	5 days ago
 train_k13.csv.gz	5 days ago
 train_k14.csv.gz	5 days ago
 train_k15.csv.gz	5 days ago
 train_k16.csv.gz	5 days ago
 train_k17.csv.gz	5 days ago
 train_k18.csv.gz	5 days ago
 train_k19.csv.gz	5 days ago
 train_k2.csv.gz	5 days ago
 train_k20.csv.gz	5 days ago
 train_k21.csv.gz	5 days ago
 train_k22.csv.gz	5 days ago
 train_k23.csv.gz	5 days ago
 train_k24.csv.gz	5 days ago



# TFRecord方式

对于train\_k0.csv.gz文件，生成tfrecord的文件大约7.7G。

对于所有的train\_k{}csv.gz文件，则 $7.7\text{G} \times 100 = 770\text{G}$ （需要消耗巨大的存储空间）

```
with tf.io.TFRecordWriter(tfrecored_file) as writer:
    for filename in fileList[:1]:
        df = pd.read_csv(filename)
        df['drawing'] = df['drawing'].apply(json.loads)
        for row in range(df.shape[0]):
            drawing = df.loc[row, 'drawing']
            img = draw_cv2(drawing, BASE_SIZE=128, size=128, lw=6)
            img = img.tostring()
            label = df.loc[row, 'y']
            # 建立 tf.train.Feature 字典
            feature = {
                'image': tf.train.Feature(bytes_list=tf.train.BytesList(value=[img])), # 图片是一个 Bytes 对象
                'label': tf.train.Feature(int64_list=tf.train.Int64List(value=[label])) # 标签是一个 Int 对象
            }
            example = tf.train.Example(features=tf.train.Features(feature=feature)) # 通过字典建立 Example
            writer.write(example.SerializeToString()) # 将Example序列化并写入 TFRecord 文件
```



# TextLineDataset方式

考虑TextLineDataset类读取csv数据，难点：**需要对drawing数据进行解码，变成image像素数据。**

```
def draw_cv2(raw_strokes, size=64, lw=6):  
    raw_strokes = eval(raw_strokes.numpy())  
    img = np.zeros((256, 256), np.uint8)  
    for stroke in raw_strokes:  
        for i in range(len(stroke[0]) - 1):  
            _ = cv2.line(img, (stroke[0][i], stroke[1][i]), (stroke[0][i + 1], stroke[1][i + 1]), 255, lw)  
    return cv2.resize(img, (size, size))
```

```
def tf_draw_cv2(image, label):  
    [image] = tf.py_function(draw_cv2, [image], [tf.float32])  
    image = tf.reshape(image, (64, 64, 1))  
    label = tf.one_hot(label, depth=NCATS)  
    image.set_shape((64, 64, 1))  
    label.set_shape((340,))  
    return image, label
```

```
train_ds = tf.data.TextLineDataset(fileList[2], compression_type='GZIP').skip(1).map(parse_csv, num_parallel_calls=tf.data.experimental.AUTOTUNE)  
train_ds = train_ds.map(tf_draw_cv2, num_parallel_calls=tf.data.experimental.AUTOTUNE)  
train_ds = train_ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE).shuffle(3000).batch(1024)
```

# TextLineDataset方式

---

**py\_function**的作用：

这是因为目前tf.data.Dataset.map函式里头的计算是在计算图模式（Graph mode）下执行，所以里头的Tensors并不会有Eager Execution下才有的numpy属性。

解法是使用tf.py\_function将我们刚刚定义的encode函式包成一个以eager模式执行的TensorFlow Operation。

参考：[https://www.tensorflow.org/api\\_docs/python/tf/py\\_function](https://www.tensorflow.org/api_docs/python/tf/py_function)



# py\_function解释

```
tf.py_function(  
    func, inp, Tout, name=None  
)
```

func	一个Python函数，该函数接受具有与inp中对应的tf.Tensor对象匹配的元素类型的Tensor对象的列表，并返回具有与Tout中的对应值匹配的元素类型的Tensor对象的列表（或单个Tensor或None）。
inp	Tensor对象列表。
Tout	张量流数据类型的列表或元组，或者只有一个张量流数据类型的单元组，指示func返回什么；如果没有返回值（即返回值为None），则为空列表。
name	操作的名称（可选）。

# from\_generator方法

tf.data.Dataset类提供from\_generator方法，针对大数据量比较友好。难点`gen`必须是一个可调用的对象，返回支持iter()对象的协议。

```
>>> import itertools
>>>
>>> def gen():
...     for i in itertools.count(1):
...         yield (i, [1] * i)
>>>
>>> dataset = tf.data.Dataset.from_generator(
...     gen,
...     (tf.int64, tf.int64),
...     (tf.TensorShape([]), tf.TensorShape([None])))
>>>
>>> list(dataset.take(3).as_numpy_iterator())
[(1, array([1])), (2, array([1, 1])), (3, array([1, 1, 1]))]
```



# from\_generator方法

```
from_generator(  
    generator, output_types, output_shapes=None, args=None  
)
```

generator	可调用对象，该对象返回支持iter()协议的对象。如果args未指定，则generator必须不带参数；否则，它必须接受与中的值一样多的参数args。
output_types	<a href="#">tf.dtype</a> 对象的嵌套结构，对应于产生的元素的每个组成部分generator。
output_shapes	(可选) <a href="#">tf.TensorShape</a> 对象的嵌套结构，对应于由产生的元素的每个组件generator。
args	(可选) <a href="#">tf.Tensor</a> 将被评估并generator作为NumPy-array参数传递给对象的元组。

```
class DataLoader(object):
    def __init__(self, resize_height=64, resize_width=64, batch_size=512, fileList=None, size=256, lw=6):
        self.resize_height = resize_height #图片高
        self.resize_width = resize_width #图片宽
        self.batch_size = batch_size #batch
        self.fileList = fileList #文件数据
        self.size = size #画图时图片大小
        self.lw = lw

    def __call__(self):
        def _generator(size, lw):
            while True:
                for filename in np.random.permutation(self.fileList):
                    df = pd.read_csv(filename)
                    df['drawing'] = df['drawing'].apply(json.loads)
                    x = np.zeros((len(df), size, size))
                    for i, raw_strokes in enumerate(df.drawing.values):
                        x[i] = draw_cv2(raw_strokes, size=size, lw=lw)
                    x = x / 255.
                    x = x.reshape((len(df), size, size, 1)).astype(np.float32)
                    y = tf.keras.utils.to_categorical(df.y, num_classes=n_labels)
                    for x_i, y_i in zip(x, y):
                        yield (x_i, y_i)

        dataset = tf.data.Dataset.from_generator(generator=_generator,
                                                output_types=(tf.dtypes.float32, tf.dtypes.int32),
                                                output_shapes=((self.resize_height, self.resize_width, 1), (340, )),
                                                args=(self.size, self.lw))

        dataset = dataset.prefetch(buffer_size=10240)
        dataset = dataset.shuffle(buffer_size=10240).batch(self.batch_size)
        return dataset
```



# 建模方法

---



# 预训练模型

**tf.keras.applications** 中有一些预定义好的经典卷积神经网络结构，如 VGG16、VGG19、ResNet、MobileNet 等。我们可以直接调用这些经典的卷积神经网络结构（甚至载入预训练的参数），而无需手动定义网络结构。

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-



# 预训练模型

---

DenseNet121(...)

DenseNet169(...)

DenseNet201(...)

InceptionResNetV2(...)

InceptionV3(...)

MobileNet(...)

MobileNetV2(...)

NASNetLarge(...)

NASNetMobile(...)

ResNet101(...)

ResNet101V2(...)

ResNet152(...)

ResNet152V2(...)

ResNet50(...)

ResNet50V2(...)

VGG16(...)

VGG19(...)

Xception(...)

# 举个例子

---

例如，我们可以使用以下代码来实例化一个 **MobileNetV2** 网络结构：

```
model = tf.keras.applications.mobilenet.MobileNet(input_shape=None,  
alpha=1.0, depth_multiplier=1, dropout=1e-3, include_top=True,  
weights='imagenet', input_tensor=None, pooling=None, classes=1000)
```

在 ImageNet 上预训练的 MobileNet 模型。

注意，该模型目前只支持 `channels_last` 的维度顺序（高度、宽度、通道）。

模型默认输入尺寸是 224x224。



# 参数介绍

- `input_shape`: 输入尺寸元组, 仅当 `include_top=False` 时有效, 否则输入形状必须是 (224, 224, 3) (`channels_last` 格式) 或 (3, 224, 224) (`channels_first` 格式)。它必须为 3 个输入通道, 且宽高必须不小于 32, 比如 (200, 200, 3) 是一个合法的输入尺寸。
- `alpha`: 控制网络的宽度:
  - 如果  $\alpha < 1.0$ , 则同比例减少每层的滤波器个数。
  - 如果  $\alpha > 1.0$ , 则同比例增加每层的滤波器个数。
  - 如果  $\alpha = 1$ , 使用论文默认的滤波器个数
- `depth_multiplier`: depthwise卷积的深度乘子, 也称为 (分辨率乘子)
- `dropout`: dropout 概率
- `include_top`: 是否包括顶层的全连接层。

备注: Depthwise卷积的一个卷积核负责一个通道, 一个通道只被一个卷积核卷积。上面所提到的常规卷积每个卷积核是同时操作输入图片的每个通道。

# 参数介绍

---

- `weights`: `None` 代表随机初始化, `'imagenet'` 代表加载在 ImageNet 上预训练的权值。
- `input_tensor`: 可选, Keras tensor 作为模型的输入 (比如 `layers.Input()` 输出的 tensor)
- `pooling`: 可选, 当 `include_top` 为 `False` 时, 该参数指定了特征提取时的池化方式。  
`None` 代表不池化, 直接输出最后一层卷积层的输出, 该输出是一个四维张量。  
`'avg'` 代表全局平均池化 (`GlobalAveragePooling2D`), 相当于在最后一层卷积层后面再加一层全局平均池化层, 输出是一个二维张量。  
`'max'` 代表全局最大池化
- `classes`: 可选, 图片分类的类别数, 仅当 `include_top` 为 `True` 并且不加载预训练权值时可用。



# Baseline构建

---



# Baseline构建

---

数据处理：shuffle数据

数据读取：from\_generator方法读取

建模方法：MobileNetV2构建

参数调优：图片大小、batch\_size等



# 一些思考

---



# 如何优化效果

---

- 不同图片大小（图片越大，效果越好）
- 图片黑白 -> RGB图片
- RNN方法建模
- 指标优化
- 多模型融合



# 总结

---



# 本节小结

## Summary

<b>实战三：</b> <b>Quick, Draw!</b> <b>Google涂鸦</b> <b>识别挑战项目</b>	数据处理	
	建模方法	
	baseline构建	
	一些思考	模型stacking、模型调参、指标优化



结语

——我 说——

**看过千万代码，不如实践一把！**





深度之眼  
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

QQ：2677693114



公众号



客服微信

