实验 2 基于 TensorFlow 的猫-狗分类实验

一、实验目的

掌握如何在 Jupyter Notebook 中编程实现模型，并会使用 ModelArts 训练 Jupyter Notebook 作业任务。

二、实验内容

万恒猫狗分类的模型的编码实现和模型训练。

三、实验步骤

1、准备数据

① 在 OBS 中新建一个桶，将数据压缩包上传到桶内。在华为云 ModelArts 中选择开发环境 Notebook。

② 创建 Notebook，选择工作环境为 TF2.1.0&Python3.6，GPU，存储配置为云硬盘。创建成功后打开 JupyterLab，创建一个新的 notebook，类型为 TensorFlow-2.1.0。

③ 获取 notebook 工作区

```
[1]: import os
     os.getcwd()
```

```
[1]: '/home/ma-user/work'
```

④ 将 OBS 桶内的数据压缩包复制到 notebook 工作区，并解压

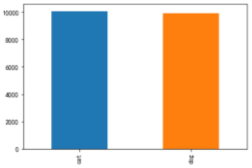
```
[*]: import zipfile
     with zipfile.ZipFile("dogs-vs-cats.zip","r") as z:
         z.extractall('.')
     with zipfile.ZipFile("train.zip","r") as z:
         z.extractall('.')
     with zipfile.ZipFile("test1.zip","r") as z:
         z.extractall('.')
```

⑤ 划分训练集、验证集，两个集合中猫狗比例都是 1：1

```
[8]: train_df, validate_df = train_test_split(df, test_size = 0.2, random_state=42)
     train_df = train_df.reset_index(drop=True)
     validate_df = validate_df.reset_index(drop=True)
```

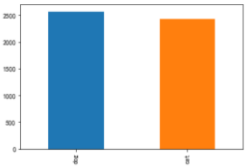
```
[9]: train_df['category'].value_counts().plot.bar()
```

```
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6a4cfbbda0>
```



```
[10]: validate_df['category'].value_counts().plot.bar()
```

```
[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6a4cfac748>
```



⑥ 创建一个 ImageDataGenerator 类，这个类可以批量生成图像，可以通过设置旋转、缩放、剪切、反转等参数来进行增强图像数据。ImageDataGenerator 类中有一个 flow_from_dataframe 方法，这个方法返回一个（x，y）元组的迭代器，该迭代器按照方法参数批量返回图像数据，其中 x 是一个包含一批尺寸为 (batch_size, *target_size, channels) 的图像样本的 numpy 数组，y 是对应的标签的 numpy 数组。

```
[12]: train_datagen = ImageDataGenerator(#含参初始化一个ImageDataGenerator，这是一个关于批量生成数据的类
          rotation_range = 5,
          rescale = 1./255,
          shear_range = 0.1,
          zoom_range = 0.2,
          horizontal_flip = True,
          width_shift_range = 0.1,
          height_shift_range = 0.1
      )
      train_generator = train_datagen.flow_from_dataframe(#在ImageDataGenerator类中有一个flow_from_dataframe的方法，
          train_df,
          "train/",
          x_col = 'filename',
          y_col = 'category',
          target_size = IMAGE_SIZE,
          class_mode = 'categorical',
          batch_size = batch_size
      )
      Found 20000 validated image filenames belonging to 2 classes.
```

参数有：

train_df 是 Pandas dataframe，一列为图像的文件名，另一列为图像的类别

“train/”是数据集的位置，

x_col 是字符串，dataframe 中包含目标图像文件夹的目录的列

y_col 是字符串或字符串列表，图像的标签值 dataframe 中将作为目标数据的列

target_size 是整数元组（h，w），所有找到的图像都会调整到这个维度

class_mode 决定返回标签值数据的类型，“categorical”2D 的 one-hot 编码，“binary” 1D 的二进制标签

color_mode 是“grayscale”，“rgb”之一

batch_size 批量数据的数量

shuffle 是否混洗数据

来源：https://blog.csdn.net/qq_27825451/article/details/90056896

⑦ 按照同样的方法创建一个批量生成 test 数据集的类，区别是不对测试集进行图像增强。

```
[13]: validation_datagen = ImageDataGenerator(rescale = 1./255)
      validation_generator = validation_datagen.flow_from_dataframe(
          validate_df,
          "train/",
          x_col = 'filename',
          y_col = 'category',
          target_size = IMAGE_SIZE,
          class_mode = 'categorical',
          batch_size = batch_size
      )
      Found 5000 validated image filenames belonging to 2 classes.
```

2、建立模型

利用 Sequential（）来定义网络，损失函数使用交叉熵，优化器 sgd，评价标准准确率。Batch_Normalization 层的作用是，在深度神经网络训练过程中，通常以送入网络的每一个 batch 训练，这样每一个 batch 具有不同的分布，所以通过 Batch_Normalization 将数据的均值拉回 0，方差为 1 的正态分布上，这样不仅数据分布一致，还避免发生梯度消失。归一化的方向是沿通道方向，归一化之后还加入缩放和平移变量。

[17]:
```
model = Sequential()
model.add(Conv2D(32,(3,3),activation = 'relu',input_shape = (IMAGE_WIDTH,IMAGE_HEIGHT,IMAGE_CHANNELS),padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),activation = 'relu',padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),activation = 'relu',padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(2,activation = 'softmax'))
```

[18]:
```
model.compile(loss = 'categorical_crossentropy',optimizer = 'sgd',metrics = ['accuracy'])
```

生成的神经网络结构。

[19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 4,215,618
Trainable params: 4,215,426
Non-trainable params: 192
_________________________________________________________________
```

在模型训练期间调用的回调函数。

[20]:
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

[21]:
```
earlystop = EarlyStopping(patience=10)
```

[22]:
```
learning_rate_reduction = ReduceLROnPlateau(
    monitor = 'val_accuracy',
    patience = 5,
    verbose = 1,
    factor = 0.5,
    min_lr = 0.0001
)
```

[23]:
```
callbacks = [earlystop, learning_rate_reduction]
```

3、训练模型

训练模型，train_generator 为训练集生成迭代器，每步生成 batch_size 个图像，一个 epoch 有 step_per_epoch 步，一共要迭代 epochs 次，validation_data 是验证集生成器。

[24]:
```
FAST_RUN = False
epochs = 3 if FAST_RUN else 20
history = model.fit(train_generator,
                    steps_per_epoch = total_train//batch_size,
                    epochs = epochs,
                    validation_data = validation_generator,
                    validation_steps = total_validate//batch_size,
                    callbacks = callbacks,
                    workers = 12
                   )
```

利用 model.save("xxx.h5")可以将模型的结构保存下来，允许重新实例化模型，保存模型当前参数，保存优化器状态，允许下一次可以紧接上次中止处开始（model.save_weights 只能保存参数）。

[26]:
```
model.save('model_total.h5')
```

将之前保存下来的模型重载，可以继续训练，亦可以用于迁移学习。

[38]:
```
model_total = load_model("model_total.h5")
```

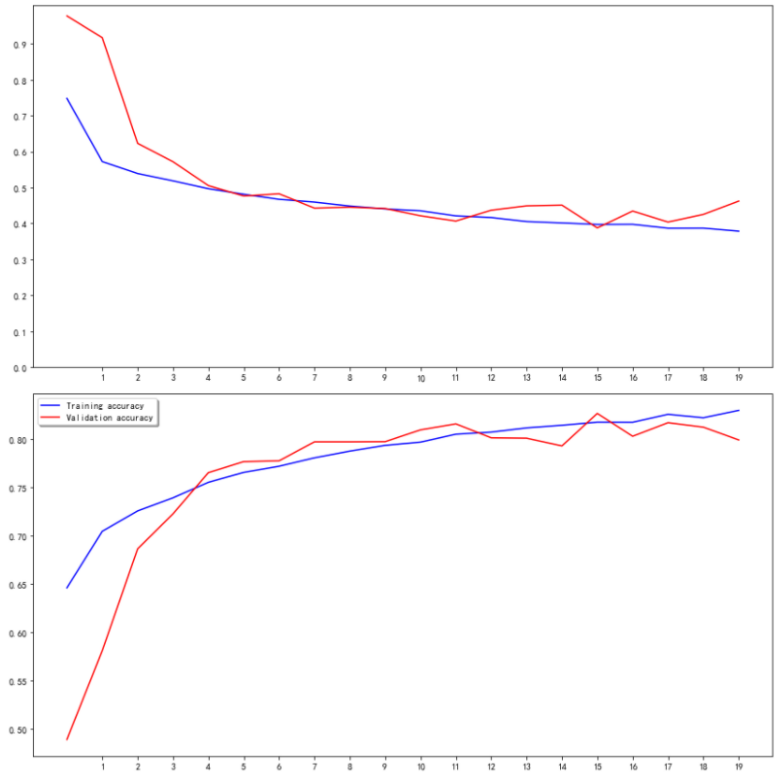
[39]:
```
history = model_total.fit(train_generator,
                          steps_per_epoch = total_train//batch_size,
                          epochs = 10,
                          validation_data = validation_generator,
                          validation_steps = total_validate//batch_size,
                          callbacks = callbacks,
                          workers = 12
                          )
```

4、可视化

```
[28]: fig,(ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
ax1.plot(history.history['loss'],color = 'b',label = "Training loss")
ax1.plot(history.history['val_loss'],color = 'r',label = "validation loss")
ax1.set_xticks(np.arange(1,epochs,1))
ax1.set_yticks(np.arange(0,1,0.1))

ax2.plot(history.history['accuracy'],color = 'b',label = "Training accuracy")
ax2.plot(history.history['val_accuracy'],color = 'r',label = "Validation accuracy")
ax2.set_xticks(np.arange(1,epochs,1))

lengend = plt.legend(loc = 'best',shadow = True)
plt.tight_layout()
plt.show()
```



模型训练在 17 个 epoch 时发生了过拟合，训练集损失值降低，验证集损失值增加。

四、总结

1. 用自己的话叙述从逻辑回归到神经元工作原理

输入神经元的数据经过线性加权求和之后，通过非线性的激活函数决定输出。Logistic 回归可以看作是两层神经元，激活函数是 Sigmoid 函数的神经网络。

2. 给出至少 2 种常用的激活函数

sigmoid 函数：$$S(x)=\frac{1}{1+e^{-x}}$$

ReLU 函数：$$f(x)=max(0,x)$$

Tanh 函数：$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

3. 给出两种池化方式

一般池化（最大、最小、平均），叠层池化，空间金字塔池化（SSP）

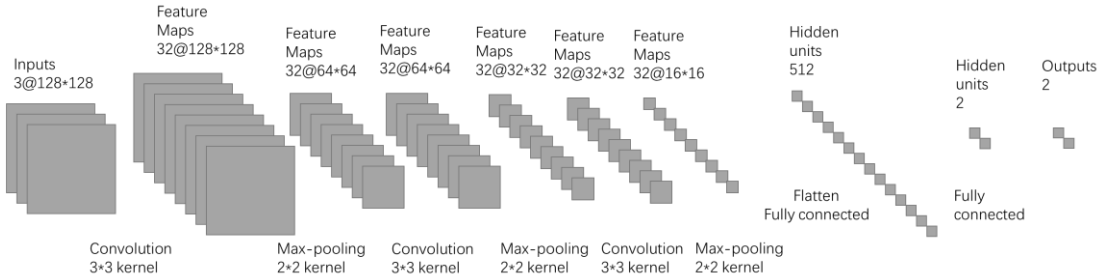
4. 简述卷积神经网络要素：卷积核，池化，特征图

卷积核：它是对物体的局部的感知，根据自身的不同提取物体的不同特征。卷积核的个数对应输出的通道数，一个输出通道是由所有输入通道对于某一个卷积核的卷积操作之后求和得到的。可以通过控制卷积核个数控制输出通道数。卷积层可以在通道与通道之间进行交互。

池化：卷积核对应的是某个局部的特征，池化层时将更大尺寸上的卷积特征进行池化可以得到全局性的特征。池化操作只在通道内部进行操作，通道间无数据分享。可以通过控制池化层来控制某通道的图像尺寸。

特征图：又可以称为通道，一个通道是对某个特征的检测，通道中某一处数值的强弱就是对当前特征反应的强弱。

5. 给出实验用到的模型图示，给出实验中的关键代码并解释 其实现原理 (数据预处理，模型建立，模型训练)



数据预处理

创建一个 ImageDataGenerator 类，这个类可以批量生成图像，可以通过设置旋转、缩放、剪切、反转等参数来进行增强图像数据。ImageDataGenerator 类中有一个 flow_from_dataframe 方法，这个方法返回一个（x，y）元组的迭代器，该迭代器按照方法参数批量返回图像数据，其中 x 是一个包含一批尺寸为 (batch_size, *target_size, channels) 的图像样本的 numpy 数组，y 是对应的标签的 numpy 数组。

[12]:
```
train_datagen = ImageDataGenerator(#含参初始化一个ImageDataGenerator，这是一个关于批量生成数据的类
    rotation_range = 5,
    rescale = 1./255,
    shear_range = 0.1,
    zoom_range = 0.2,
    horizontal_flip = True,
    width_shift_range = 0.1,
    height_shift_range = 0.1
)
train_generator = train_datagen.flow_from_dataframe(#在ImageDataGenerator类中有一个flow_from_dataframe的方法，
    train_df,
    "train/",
    x_col = 'filename',
    y_col = 'category',
    target_size = IMAGE_SIZE,
    class_mode = 'categorical',
    batch_size = batch_size
)
```

```
Found 20000 validated image filenames belonging to 2 classes.
```

参数有：

train_df 是 Pandas dataframe，一列为图像的文件名，另一列为图像的类别

“train/”是数据集的位置，

x_col 是字符串，dataframe 中包含目标图像文件夹的目录的列
y_col 是字符串或字符串列表，图像的标签值 dataframe 中将作为目标数据的列
target_size 是整数元组（h，w），所有找到的图像都会调整到这个维度
class_mode 决定返回标签值数据的类型，“categorical”2D 的 one-hot 编码，“binary”1D 的二进制标签
color_mode 是“grayscale”，“rgb”之一
batch_size 批量数据的数量
shuffle 是否混洗数据

模型建立

利用 Sequential（）来定义网络，损失函数使用交叉熵，优化器 sgd，评价标准准确率。Batch_Normalization 层的作用是，在深度神经网络训练过程中，通常以送入网络的每一个 batch 训练，这样每一个 batch 具有不同的分布，所以通过 Batch_Normalization 将数据的均值拉回 0，方差为 1 的正态分布上，这样不仅数据分布一致，还避免发生梯度消失。归一化的方向是沿通道方向，归一化之后还加入缩放和平移变量。

[17]:
```
model = Sequential()
model.add(Conv2D(32,(3,3),activation = 'relu',input_shape = (IMAGE_WIDTH,IMAGE_HEIGHT,IMAGE_CHANNELS),padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),activation = 'relu',padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),activation = 'relu',padding = 'same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(2,activation = 'softmax'))
```

[18]:
```
model.compile(loss = 'categorical_crossentropy',optimizer = 'sgd',metrics = ['accuracy'])
```

在模型训练期间调用的回调函数。

[20]:
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

[21]:
```
earlystop = EarlyStopping(patience=10)
```

[22]:
```
learning_rate_reduction = ReduceLROnPlateau(
    monitor = 'val_accuracy',
    patience = 5,
    verbose = 1,
    factor = 0.5,
    min_lr = 0.0001
)
```

[23]:
```
callbacks = [earlystop, learning_rate_reduction]
```

训练模型

训练模型，train_generator 为训练集生成迭代器，每步生成 batch_size 个图像，一个 epoch 有 step_per_epoch 步，一共要迭代 epochs 次，validation_data 是验证集生成器。

[24]:
```
FAST_RUN = False
epochs = 3 if FAST_RUN else 20
history = model.fit(train_generator,
                    steps_per_epoch = total_train//batch_size,
                    epochs = epochs,
                    validation_data = validation_generator,
                    validation_steps = total_validate//batch_size,
                    callbacks = callbacks,
                    workers = 12
                   )
```

利用 model.save("xxx.h5")可以将模型的结构保存下来，允许重新实例化模型，保存模型当前参数，保存优化器状态，允许下一次可以紧接上次中止处开始（model.save_weights 只能保存参数）。

[26]:
```
model.save('model_total.h5')
```

将之前保存下来的模型重载，可以继续训练，亦可以用于迁移学习。

[38]:
```
model_total = load_model("model_total.h5")
```

[39]:
```
history = model_total.fit(train_generator,
                          steps_per_epoch = total_train//batch_size,
                          epochs = 10,
                          validation_data = validation_generator,
                          validation_steps = total_validate//batch_size,
                          callbacks = callbacks,
                          workers = 12
                          )
```

6. 随堂验收问答

sigmoid 激活函数与 softmax 激活函数之间的区别?

sigmoid 函数:

$$s(x) = \frac{1}{1+e^{-x}}$$

softmax 函数:

$$s(x) = \frac{e^{x_i}}{e^{x_1}+e^{x_2}} = \frac{1}{1+e^{x_j-x_i}}$$

可以看出，当 softmax 函数中的 i 等于 1 的时候，函数形式与 sigmoid 等价。