实验2-基于TensorFlow 的猫-狗分类实验

使用JupyterNotebook来实现模型，并在ModeArts上完成模型的训练

实验目的：掌握如何在JupyterNotebook中编程实现模型，并会使用ModelArts训练JupyerNotebook作业任务。
实验内容：完成猫狗分类的模型的编码实现和模型训练。

实验报告要求：

实验目的

实验内容

实验步骤（需要你自己调试模型并训练，熟悉Tensorflow API接口，使用ModelArts进行训练。相关代码和运行结果截图 贴在pdf文档中）

总结：

1. 用自己的话叙述从逻辑回归到神经元工作原理
2. 给出至少2种常用的激活函数
3. 给出两种池化方式
4. 简述卷积神经网络要素：卷积核，池化，特征图
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）
6. 随堂验收问答

提交方式：研究生信息平台

本次实验时间：周六晚7：00—10：00，地点不变。下次时间依旧周四。

实验大纲

- 回顾MNIST图片分类
  - softmax多分类方法
  - 人工神经网络
- 猫狗分类实践
  1. 上传数据
  2. 生成数据
  3. 建立模型
  4. 训练模型
  5. 可视化

MNIST分类

MNIST:

图片像素： 28*28

softmax多分类方法

softmax 回归(softmax regression)其实是 logistic 回归的一般形式，logistic 回归用于二分类。

对于输入数据$\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$有$k$个类别，即$y_i \in \{1,2,\ldots,k\}$，那么softmax回归主要估算输入数据$x_i$归属每一类的概率，即

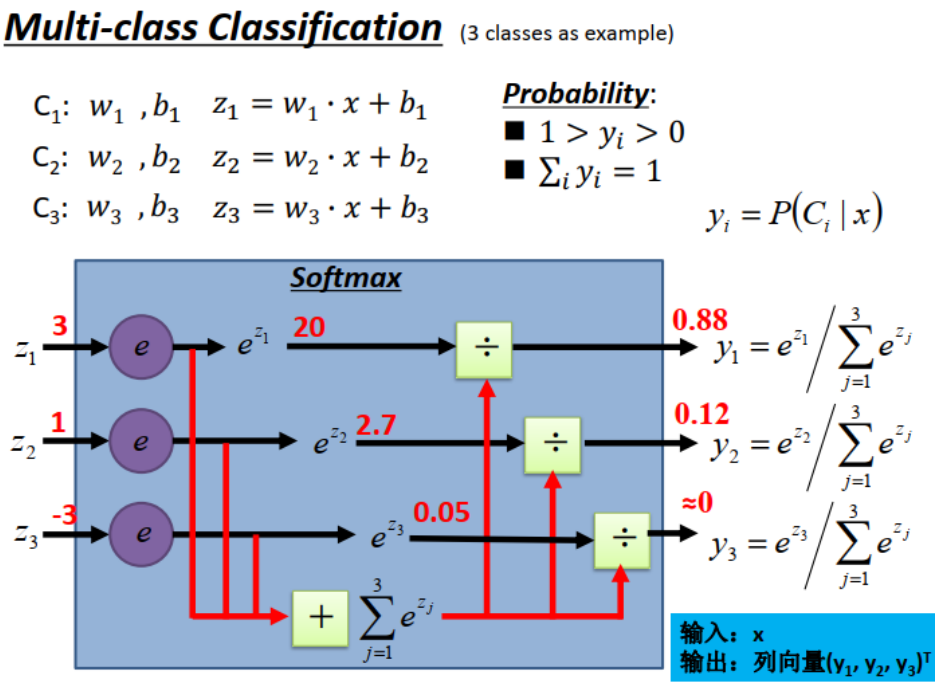
$$h_\theta(x_i)=\begin{bmatrix} p(y_i=1|x_i;\theta) \\ p(y_i=2|x_i;\theta) \\ \vdots \\ p(y_i=k|x_i;\theta) \end{bmatrix}=\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}}\begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \vdots \\ e^{\theta_k^T x_i} \end{bmatrix}$$

其中，$\theta_1,\theta_2,\ldots,\theta_k \in \theta$ 是模型的参数，乘以$\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}}$ 是为了让概率位于[0, 1]并且概率之和为 1，

softmax回归将输入数据x_i归属类别 j 的概率为

$$p(y_i=j|x_i;\theta)=\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}$$

softmax多分类举例：

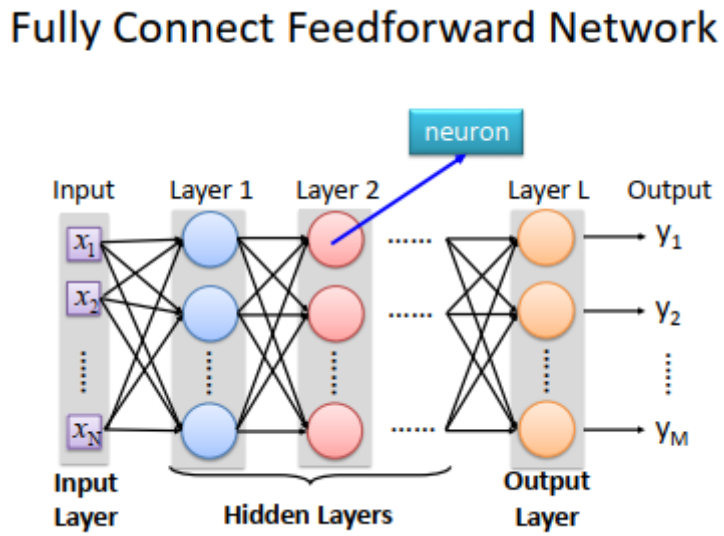


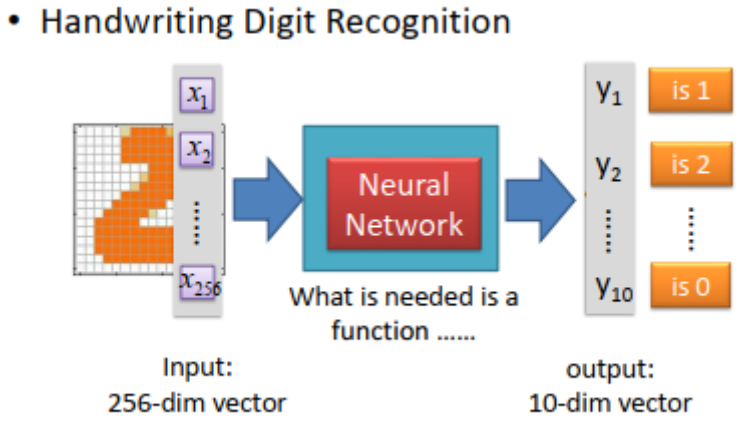
PS: 参考课件Ch3 Classification

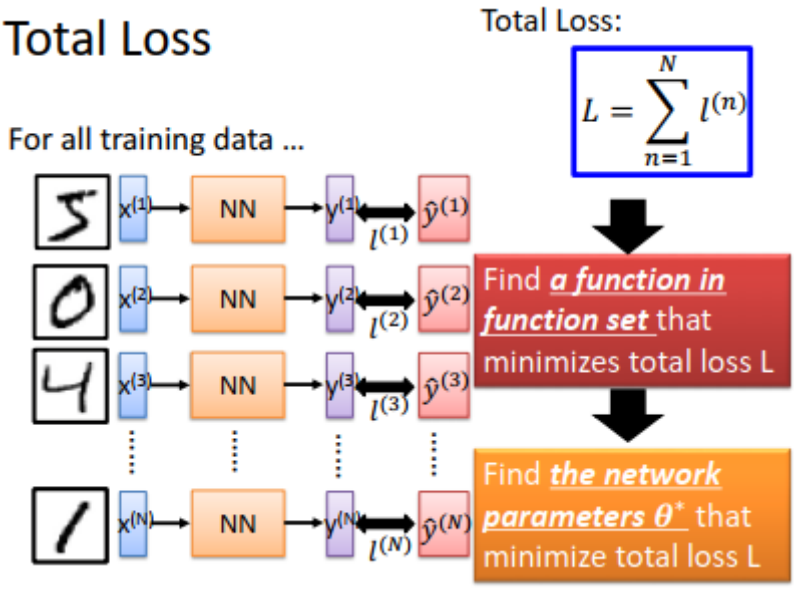
将MNIST每一个像素点都作为输入数据 x_i，28*28像素的MNIST图片意味着x_i是一个28*28大小的向量。经过softmax之后与真实标签进行对比计算代价函数。

人工神经网络方法

神经网络：可通过学习一种多层非线性网络结构，实现复杂函数逼近。







与softmax相同使用神经网络作为模型是输入的x_i同为一个28*28的向量，意味着输入层会有28*28个单元，即x_i^j，其中$j \in (1, 2, \ldots, n)$，此处n即为28*28。

PS:参考课件Ch5_Fundamentas-of-DeepLearning

PS: 上述两种方法本次实验不要求实现，但根据课堂教学和课后作业应当已经掌握。

猫狗分类实践

卷积神经网络方法（Convolutional Neural Network ）

在 CNN 出现之前，图像对于人工智能来说是一个难题，有2个原因：

1. 图像需要处理的数据量太大，导致成本很高，效率很低
2. 图像在数字化的过程中很难保留原有的特征，导致图像处理的准确率不高

现在随随便便一张图片都是 1000×1000 像素以上的， 每个像素都有RGB 3个参数来表示颜色信息。

假如我们处理一张 1000×1000 像素的图片，我们就需要处理3百万个参数!

1000×1000×3=3,000,000

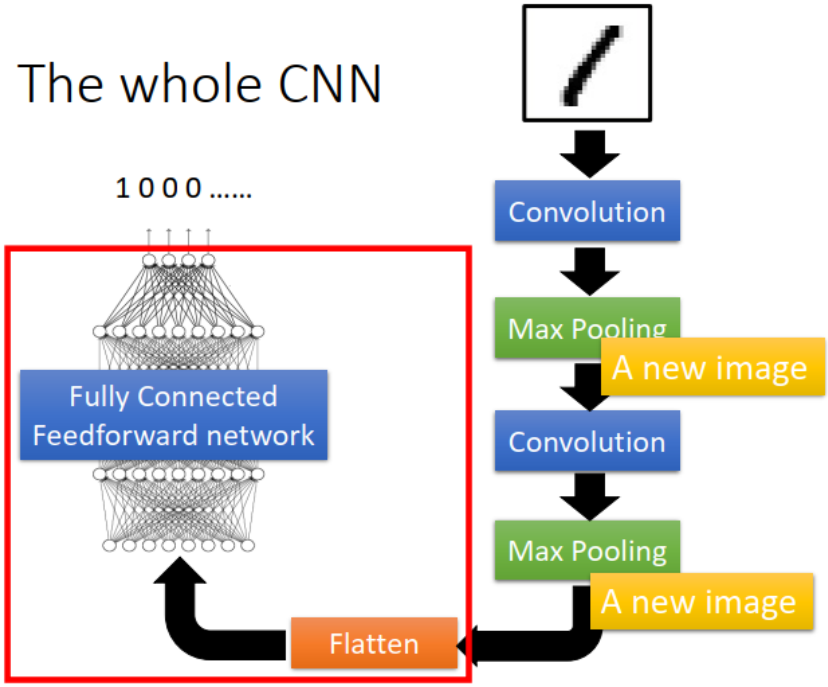
这么大量的数据处理起来是非常消耗资源的，而且这只是一张不算太大的图片!

卷积神经网络 — CNN 解决的第一个问题就是「将复杂问题简化」，把大量参数降维成少量参数，再做处理。

更重要的是：我们在大部分场景下，降维并不会影响结果。比如1000像素的图片缩小成200像素，并不影响肉眼认出来图片中是一只猫还是一只狗，机器也是如此。

典型的 CNN 由3个部分构成：

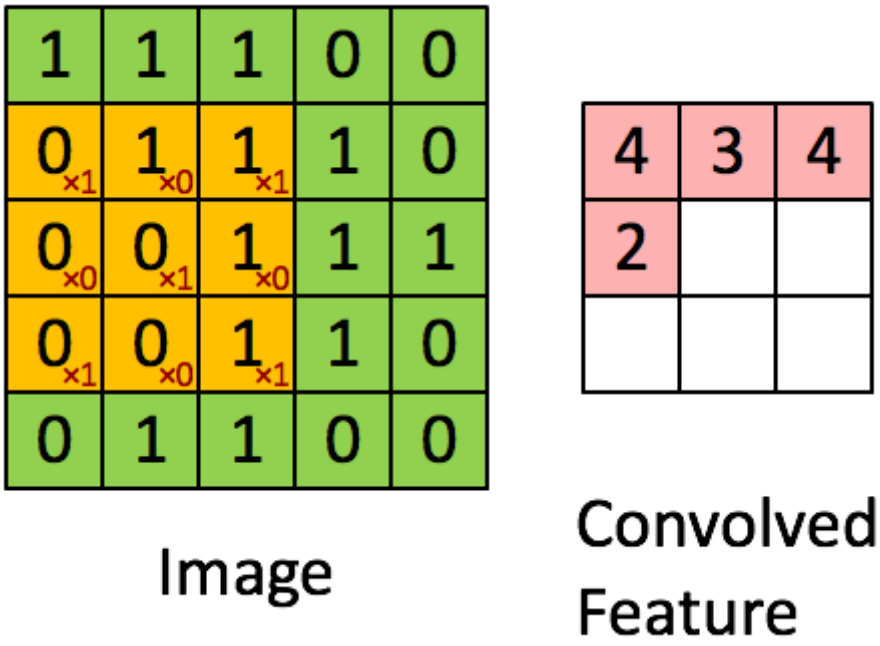
1. 卷积层
2. 池化层
3. 全连接层



卷积 — — 提取特征

这个过程我们可以理解为我们使用一个过滤器（卷积核）来过滤图像的各个小区域，从而得到这些小区域的特征值。

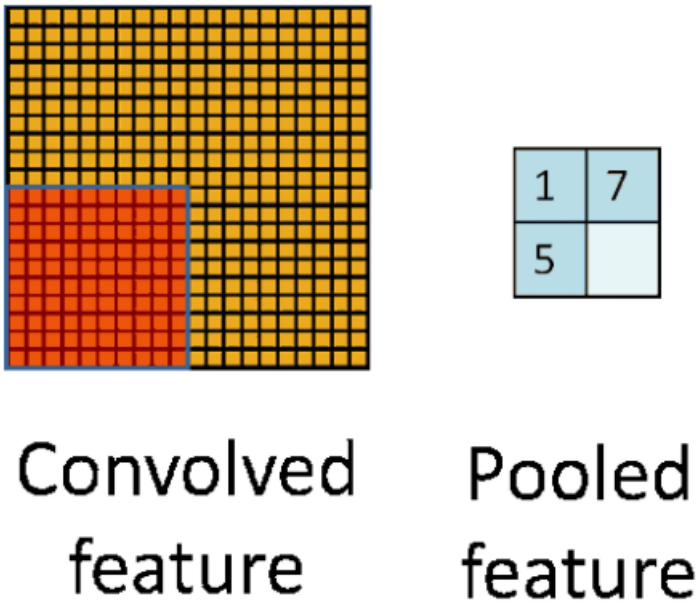
在具体应用中，往往有多个卷积核，可以认为，每个卷积核代表了一种图像模式，如果某个图像块与此卷积核卷积出的值大，则认为此图像块十分接近于此卷积核。如果我们设计了6个卷积核，可以理解：我们认为这个图像上有6种底层纹理模式，也就是我们用6中基础模式就能描绘出一副图像。以下就是25种不同的卷积核的示例：



总结：卷积层的通过卷积核的过滤提取出图片中局部的特征，跟上面提到的人类视觉的特征提取类似。

池化层（下采样） — — 数据降维，避免过拟合

池化层简单说就是下采样，他可以大大降低数据的维度。其过程如下：



上图中，我们可以看到，原始图片是20×20的，我们对其进行下采样，采样窗口为10×10，最终将其下采样成为一个2×2大小的特征图。

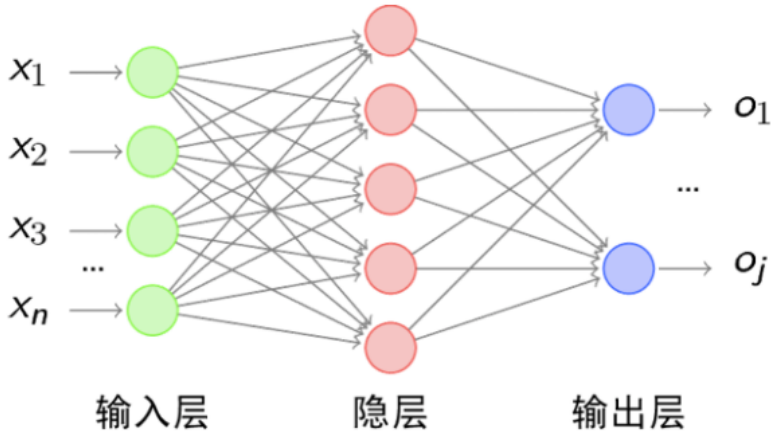
之所以这么做的原因，是因为即使做完了卷积，图像仍然很大（因为卷积核比较小），所以为了降低数据维度，就进行下采样。

总结：池化层相比卷积层可以更有效的降低数据维度，这么做不但可以大大减少运算量，还可以有效的避免过拟合。

全连接层 — — 输出结果

这个部分就是最后一步了，经过卷积层和池化层处理过的数据输入到全连接层，得到最终想要的结果。

经过卷积层和池化层降维过的数据，全连接层才能"跑得动"，不然数据量太大，计算成本高，效率低下。



参考文章：

课件：ch6DL4CV(1)

一文看懂卷积神经网络

下载数据&确定成功指标

- 简介：
  - 猫-狗数据集，训练集包含 25000张猫和狗的图片，测试集为12500张。 (1 = dog, 0 = cat)
  - 构造 小样本数据集：我们只使用4000张猫狗图像（每个类别都有2000张）作为数据集；将数据集分为三个子集：每个类别各1000个样本的训练集、每个类别各500个样本的验证集和每个类别各500个样本的测试集。
  - 这是一个平衡的二分类问题，分类精度可作为衡量成功的指标。

  官网：https://www.kaggle.com/c/dogs-vs-cats

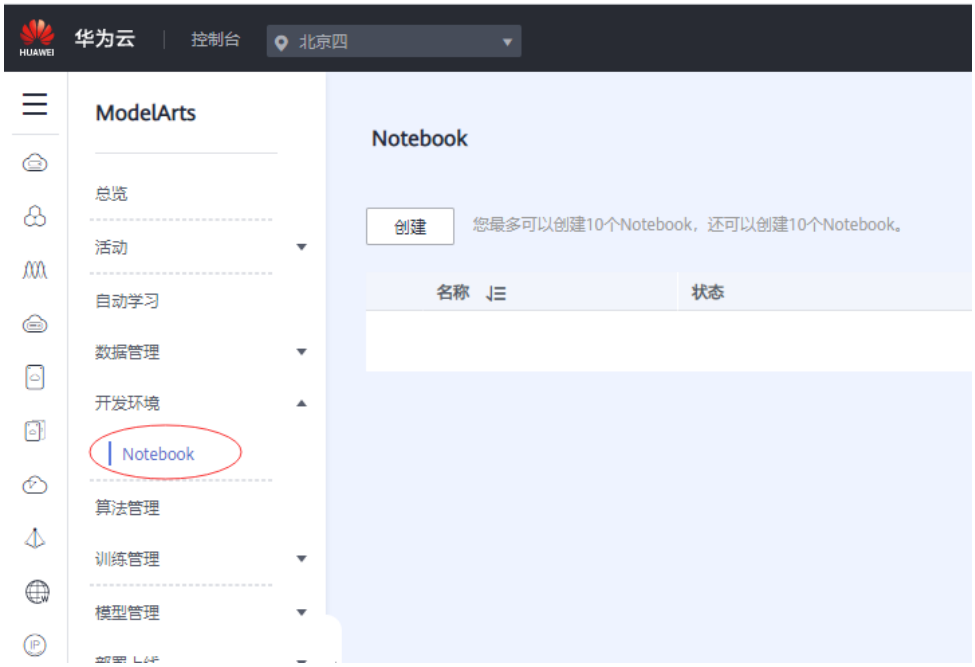
示例：

数据集获取： 在群文件中下载 或者 直接官网[下载]

上传数据到notebook

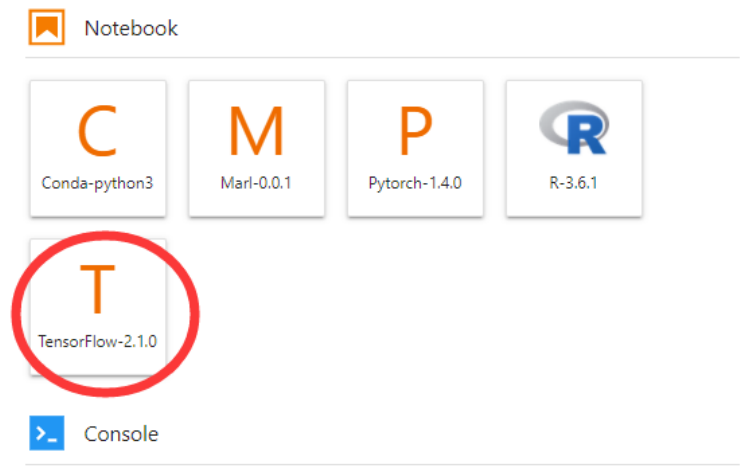
上传数据到OBS

创建notebook

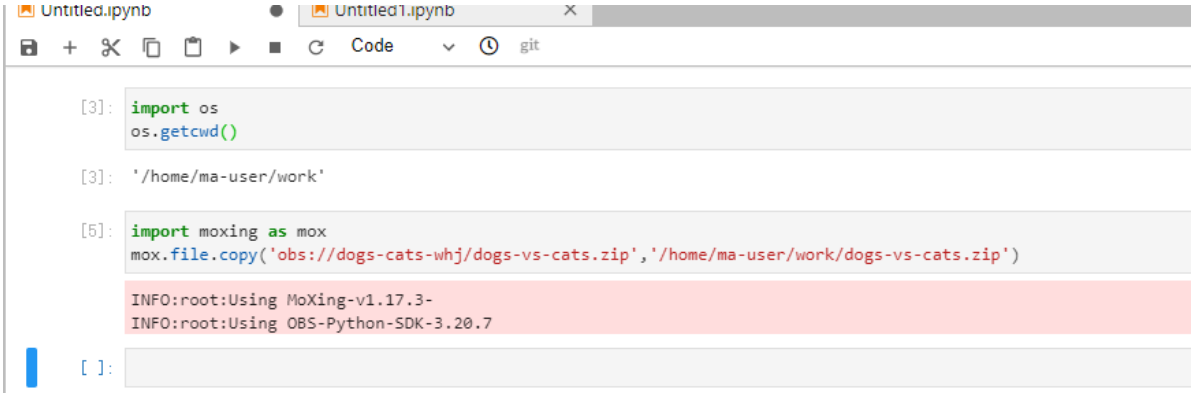


将OBS内的数据复制到notebook中

1. 创建成功后打开JupyterLab，创建一个新notebook



2. 使用moxing方法复制文件



> [moxing文件操作]

```
#下载一个OBS文件，s3 -> 本地
mox.file.copy('s3://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
#下载一个OBS文件夹，s3 -> 本地
mox.file.copy_parallel('s3://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```

3. 解压数据

```
import os
print(os.listdir())
```

```
['Untitled.ipynb', 'Untitled1.ipynb', '.ipynb_checkpoints', 'dogs-vs-cats.zip']
```

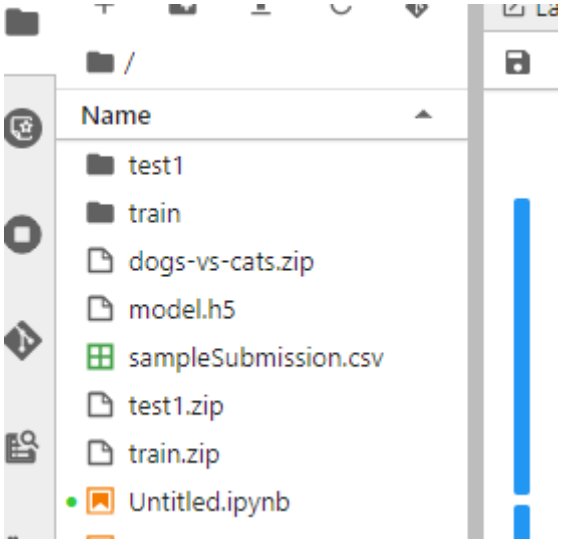
```
import zipfile
with zipfile.ZipFile("dogs-vs-cats.zip","r") as z:
    z.extractall('.')
```

```
print(os.listdir())
```

```
['test1.zip', 'Untitled.ipynb', 'sampleSubmission.csv', 'Untitled1.ipynb', '.ipynb_checkpoints', 'dogs-vs-cats.zip', 'train.zip']
```

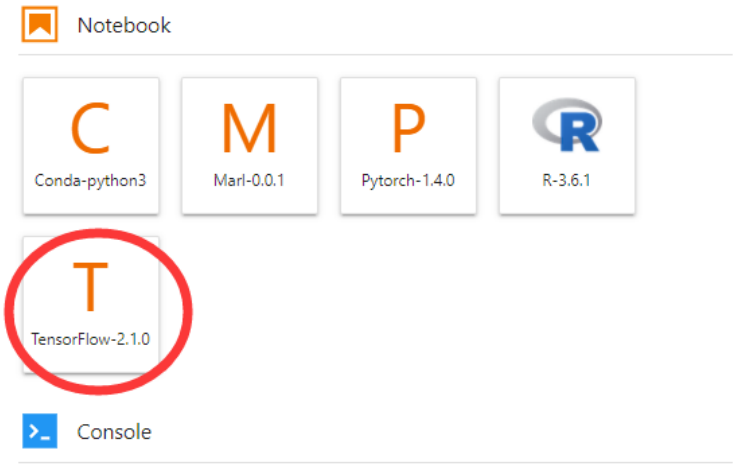
```
with zipfile.ZipFile("train.zip","r") as z:
    z.extractall('.')
with zipfile.ZipFile("test1.zip","r") as z:
    z.extractall('.')
```

左侧文件夹下可能会没有及时更新，点下Refresh File List即可。



猫狗分类

新建一个notebook



导入需要的函数

```
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pylab as plt
import random
import os
```

定义常量

```
]: FAST_RUN = False
   IMAGE_WIDTH=128
   IMAGE_HEIGHT=128
   IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
   IMAGE_CHANNELS=3
```

训练数据

```
filenames = os.listdir('train')
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)
df = pd.DataFrame({
    'filename' : filenames,
    'category' : categories
})
```

[5]:
```
filenames = os.listdir('train')
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)
df = pd.DataFrame({
    'filename' : filenames,
    'category' : categories
})
```

[6]:
```
df.head()
```

[6]:

	category	filename
0	1	dog.9326.jpg
1	1	dog.3026.jpg
2	1	dog.8155.jpg
3	0	cat.6261.jpg
4	1	dog.5129.jpg

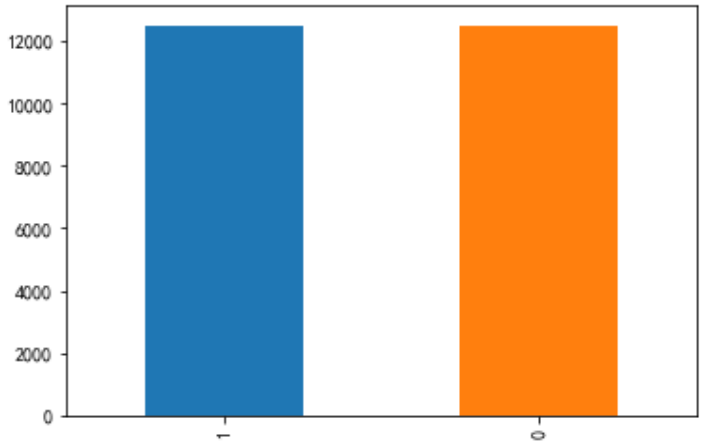
[7]:
```
df.tail()
```

[7]:

	category	filename
24995	1	dog.5098.jpg
24996	0	cat.4210.jpg
24997	1	dog.9898.jpg
24998	1	dog.10311.jpg
24999	1	dog.10466.jpg

[8]:
```
df['category'].value_counts().plot.bar()
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c922ebe48>
```



[9]:
```
sample = random.choice(filenames)
image = load_img('train/'+sample)
plt.imshow(image)
```

[9]:
```
<matplotlib.image.AxesImage at 0x7f6c909104a8>
```

处理数据标签

使用`ImageDataGenerator`生成训练数据，因此需要先把类别列转成`string`类型。

然后`ImageDataGenerator`会把他转成one-hot编码。

因此把 1 转成 dog， 把 0 转成 cat

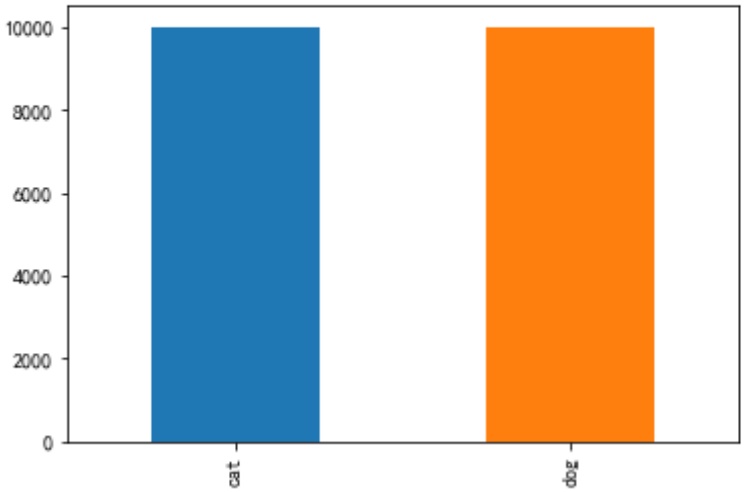
```
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
```

划分训练集、验证集

:
```
train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
```

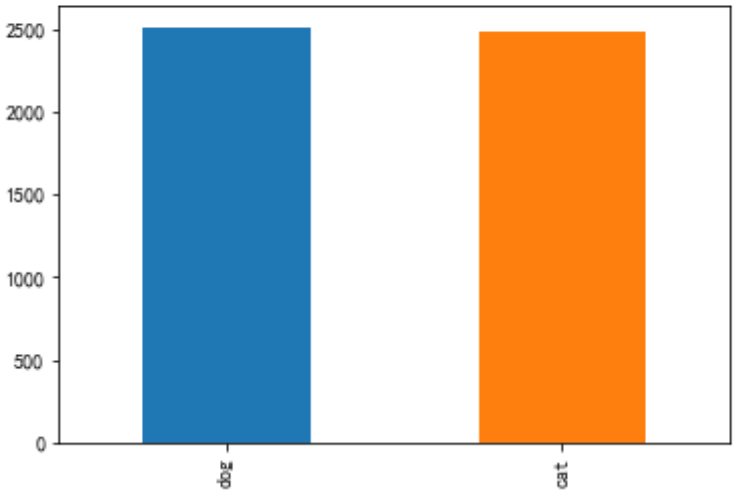
```
train_df['category'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c8859eba8>
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c88572e48>
```



设置Batch Size 与 训练和验证样本数

```
total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
batch_size=128
```

数据生成器

Generate batches of tensor image data with real-time data augmentation.

训练集数据生成

```
train_datagen = ImageDataGenerator(
    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(
    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.
```

验证集数据生成

```
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
)
```

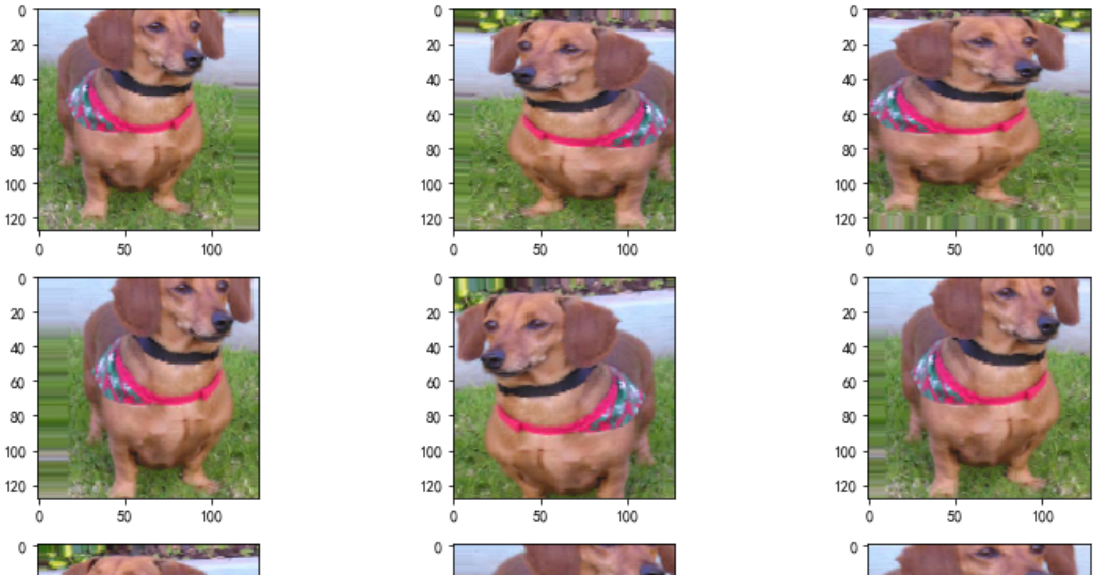
```
Found 5000 validated image filenames belonging to 2 classes.
```

查看数据生成样例

```
example_df = train_df.sample(n=1).reset_index(drop=True)
example_generator = train_datagen.flow_from_dataframe(
    example_df,
    "train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
```

```
Found 1 validated image filenames belonging to 1 classes.
```

```
plt.figure(figsize=(12, 12))
for i in range(0, 15):
    plt.subplot(5, 3, i+1)
    for X_batch, Y_batch in example_generator:
        image = X_batch[0]
        plt.imshow(image)
        break
plt.tight_layout()
plt.show()
```



建立模型

务必了解 构建模型使用的API接口

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout,Flatten,Dense,Activation,BatchNormalization
```

```
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'))
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'))
```

[12]:
```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics = ['accuracy'])
```

[13]:
```
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, 30, 30, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 7200)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3686912
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 3,707,714
Trainable params: 3,707,522
Non-trainable params: 192
_________________________________________________________________
```

回调：在模型训练期间的某些点调用的实用程序

EarlyStoping与学习率下降

:
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

:
```
earlystop = EarlyStopping(patience=10)
```

:
```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=5,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

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

训练模型

```
FAST_RUN = False
epochs=3 if FAST_RUN else 50
# history = model.fit_generator(
#     train_generator,
#     epochs=epochs,
#     validation_data=validation_generator,
#     validation_steps=total_validate//batch_size,
#     steps_per_epoch=total_train//batch_size,
#     callbacks=callbacks
# )
history = model.fit(train_generator,
                    epochs=epochs,
                    validation_data=validation_generator,
                    validation_steps=total_validate//batch_size,
                    steps_per_epoch=total_train//batch_size,
                    callbacks=callbacks,
                    workers=12)
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 156 steps, validate for 39 steps
Epoch 1/50
156/156 [==============================] - 46s 295ms/step - loss: 0.4613 - accuracy: 0.7789 - val_loss: 0.4430 - val_accuracy: 0.7963
Epoch 2/50
156/156 [==============================] - 46s 295ms/step - loss: 0.4480 - accuracy: 0.7873 - val_loss: 0.4467 - val_accuracy: 0.7967
Epoch 3/50
156/156 [==============================] - 46s 297ms/step - loss: 0.4394 - accuracy: 0.7904 - val_loss: 0.4402 - val_accuracy: 0.7993
Epoch 4/50
156/156 [==============================] - 46s 296ms/step - loss: 0.4321 - accuracy: 0.7967 - val_loss: 0.4366 - val_accuracy: 0.7971
Epoch 5/50
155/156 [============================>.] - ETA: 0s - loss: 0.4277 - accuracy: 0.8038
Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
156/156 [==============================] - 46s 295ms/step - loss: 0.4276 - accuracy: 0.8038 - val_loss: 0.4402 - val_accuracy: 0.7971
Epoch 6/50
```

保存模型

请在实验报告中利用 `model.save_weights` 与 `model.load_weights` 先训练10个Epoch，保存模型后，再次加载继续训练20个Epoch。使用方法查看Tensorflow官方文档。

> load
>
> save

[49]:
```
model.save_weights('model.h5')
```

可视化训练结果

]:
```
history.history
```

]:
```
{'accuracy': [0.7559883, 0.7641908, 0.77470815],
 'loss': [0.49938607748579866, 0.4856059509486968, 0.4724945062216545],
 'lr': [0.01, 0.01, 0.01],
 'val_accuracy': [0.76061696, 0.7754407, 0.7734375],
 'val_loss': [0.4991487295199663, 0.48829935529293156, 0.49666367585842425]}
```

]:
```
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))

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

准备测试集&创建测试机Generator

```
test_filenames = os.listdir("test1")
test_df = pd.DataFrame({
    'filename': test_filenames
})
nb_samples = test_df.shape[0]
```

```
test_gen = ImageDataGenerator(rescale=1./255)
test_generator = test_gen.flow_from_dataframe(
    test_df,
    "test1/",
    x_col='filename',
    y_col=None,
    class_mode=None,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    shuffle=False
)
```

```
Found 12500 validated image filenames.
```

预测

取概率最高的作为预测结果

例如predict 中结果为 [[0.1, 0.9], [0.6, 0.4]] 则结果为[1, 0]

并且使用`train_generator.class_indices`映射回原来的标签，

```
predict = model.predict(test_generator,
                        steps=np.ceil(nb_samples/batch_size),
                        workers=12)
```

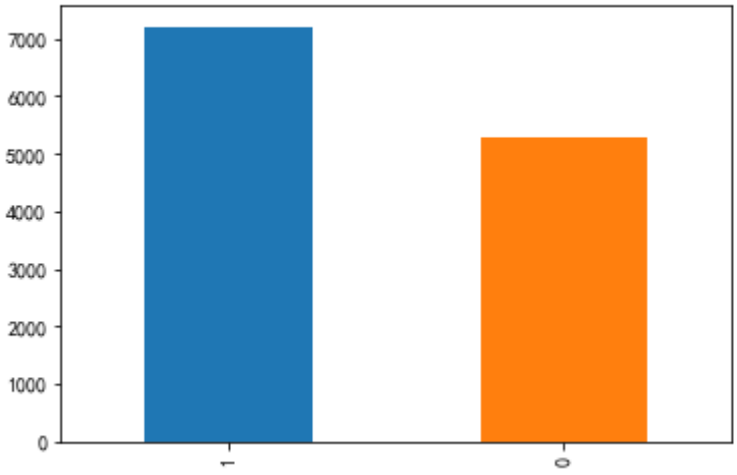
```
test_df['category'] = np.argmax(predict, axis=-1)
```

```
label_map = dict((v,k) for k,v in train_generator.class_indices.items())
test_df['category'] = test_df['category'].replace(label_map)
```

```
test_df['category'] = test_df['category'].replace({ 'dog': 1, 'cat': 0 })
```

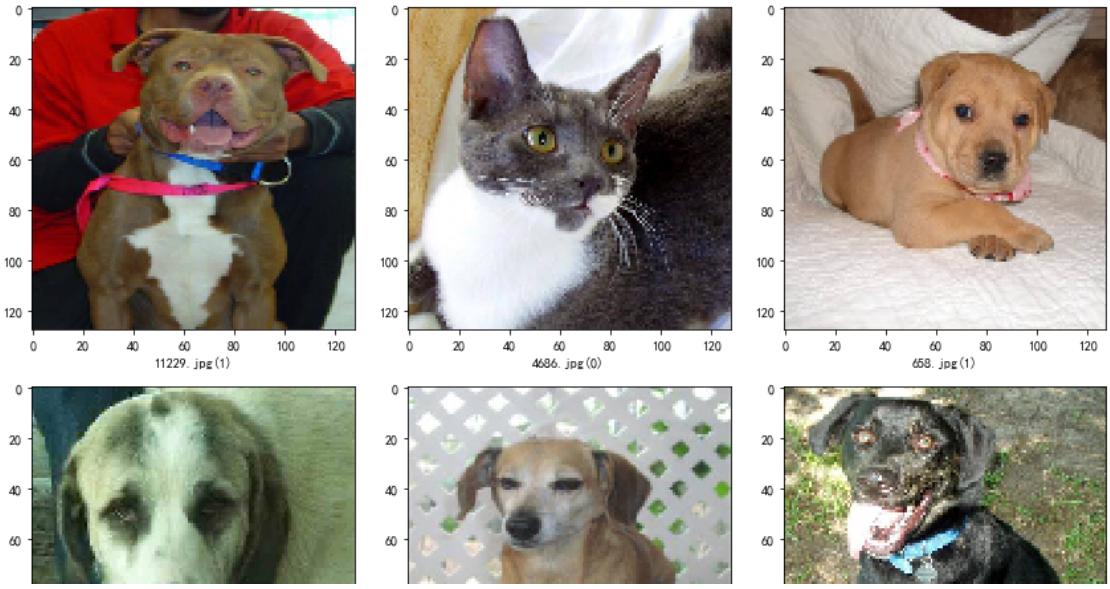
```
test_df['category'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c44294ef0>
```



查看结果

]:
```
sample_test = test_df.head(18)
sample_test.head()
plt.figure(figsize=(12, 24))
for index, row in sample_test.iterrows():
    filename = row['filename']
    category = row['category']
    img = load_img("test1/"+filename, target_size=IMAGE_SIZE)
    plt.subplot(6, 3, index+1)
    plt.imshow(img)
    plt.xlabel(filename + '(' + "{}".format(category) + ')' )
plt.tight_layout()
plt.show()
```



> [参考]