数据生成

为什么需要使用数据生成器？ 答：数据占内存，可能没有足够的内存一下子存储所有数据，生成器可以把小量数据多批次读入内存。虽然花费了时间，但是能够处理很大的数据量。

不使用DA

In [1]:
```
from keras.preprocessing.image import ImageDataGenerator #从kera
s导入需要需要的包

IMSIZE=128
validation_generator = ImageDataGenerator(rescale=1./255).flow_f
rom_directory(
    '/clubear/Lecture 5.1 - Batch Normalization/data/CatDog/vali
dation',
    target_size=(IMSIZE, IMSIZE), #不管数据里的图像大小，在input统一
shape
    batch_size=200, #每一个batch只读取200个数据
    class_mode='categorical') #分类问题
#简单的制作validation data，rescale 是因为TensorFlow需要倒入的数据值在
0-1之间，而我们拥有的图片是0-255

train_generator = ImageDataGenerator(rescale=1./255).flow_from_d
irectory(
    '/clubear/Lecture 5.1 - Batch Normalization/data/CatDog/trai
n',
    target_size=(IMSIZE, IMSIZE),
    batch_size=200,
    class_mode='categorical')
```

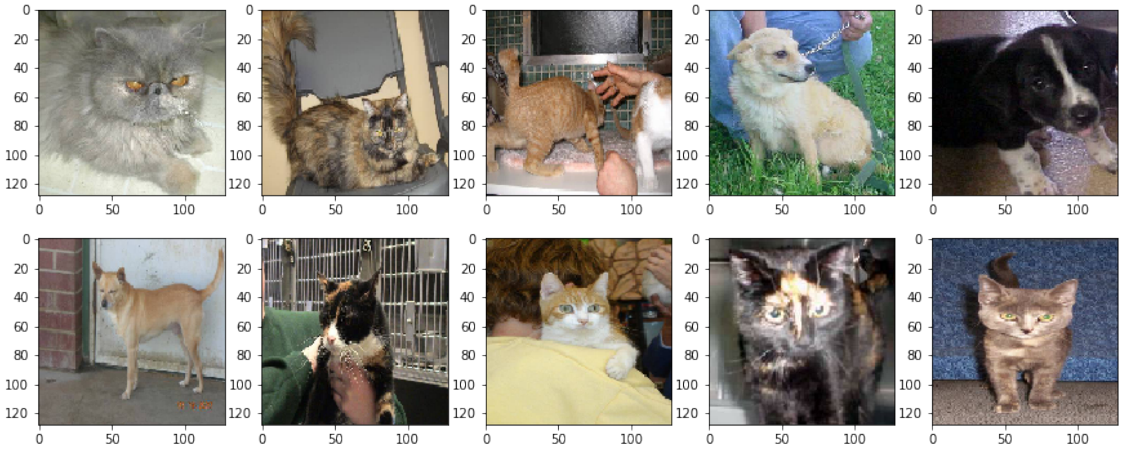
```
Using TensorFlow backend.

Found 10000 images belonging to 2 classes.
Found 15000 images belonging to 2 classes.
```

In [2]:
```
#数据展示
from matplotlib import pyplot as plt

plt.figure()
fig,ax = plt.subplots(2,5)
fig.set_figheight(6)
fig.set_figwidth(15)
ax=ax.flatten()
X,Y=next(train_generator) #next是一个简单的方程将训练数据添加到X, Y上
for i in range(10): ax[i].imshow(X[i,:,:,:])
```

```
<Figure size 432x288 with 0 Axes>
```



使用DA

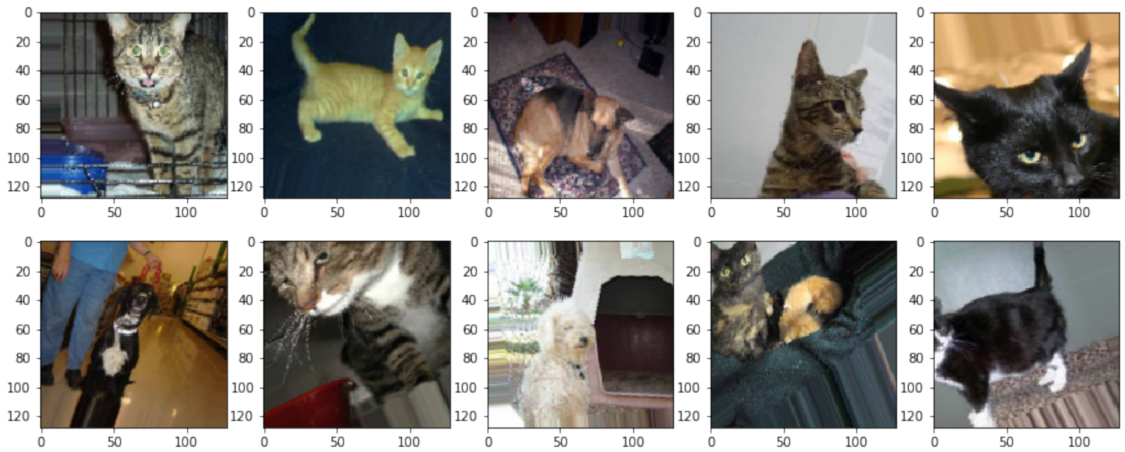
In [3]:
```
train_generator1 = ImageDataGenerator(
    rescale=1./255, #tensorFlow要求
    shear_range=0.5, #扭曲变形
    rotation_range=30, #旋转
    zoom_range=0.2, #放大
    width_shift_range=0.2, #水平平移
    height_shift_range=0.2 , #垂直评议
    horizontal_flip=True ).flow_from_directory(
    '/clubear/Lecture 5.1 - Batch Normalization/data/CatDog/trai
n',
    target_size=(IMSIZE, IMSIZE),
    batch_size=200,
    class_mode='categorical')
#这里使用了Data Augmentation技术, 即将图片变形, 使得1) 数据量更大; 2) 在
现实生活中的图片不可能是全部正面, 通过数据增强可以
#使我们的模型识别到更多变形的图像
#train现在存为了train_generator1
```

```
Found 15000 images belonging to 2 classes.
```

In [4]:
```
#数据展示
from matplotlib import pyplot as plt

plt.figure()
fig,ax = plt.subplots(2,5)
fig.set_figheight(6)
fig.set_figwidth(15)
ax=ax.flatten()
X,Y=next(train_generator1) #X和Y现在从train_generator1中取
for i in range(10): ax[i].imshow(X[i,:,:,:])
```

```
<Figure size 432x288 with 0 Axes>
```



模型搭建

In [5]:
```
from keras.layers import Conv2D,Dense,Flatten,Input,MaxPooling2D
,Dropout
from keras import Model

input_layer = Input([IMSIZE,IMSIZE,3])
x = input_layer
for _ in range(5):
    x = Conv2D(20,[3,3], activation = 'relu')(x)
    x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)

x = Flatten()(x)
x = Dense(84,activation = 'relu')(x)
x = Dense(2,activation = 'softmax')(x)
output_layer=x
model=Model(input_layer,output_layer)
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 128, 128, 3)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 126, 126, 20)      560       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 20)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 20)        3620      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 20)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 20)        3620      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 20)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 20)        3620      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 20)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 20)          3620      
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 20)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 80)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 84)                6804      
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 170       
=================================================================
Total params: 22,014
Trainable params: 22,014
Non-trainable params: 0
_________________________________________________________________
```

参数解释

第一层input layer，无参数。
for loop中含有5个卷积层和5个池化层，除去第一个卷积层，其他卷积层参数为：
conv2D： （3 3 20+1） *20 = 181* 20 = 3620 , 其中3 *3 是kernel size，20是上一层遗留通道数，20是卷积核个数。*
池化使用[2,2]矩形，步长行列都是2，没有学习项，无参数
for loop的第一个卷积层参数计算如下：
(3 3 3 +1） 20 = 28 *20 = 560，其中3* 3 是kernel size，3是上一层遗留通道数，20是卷积核个数。

压扁，无参数

Dense1： (hidden） 80 *84 + 84 = 6720 +84 = 6804*
Dense2： *84* 2 + 2 = 168 + 2 = 170
所以总共有22014个参数。
没有non-trainable parameters因为BN没有被用到

In [6]:
```
from keras.optimizers import Adam
model.compile(loss='categorical_crossentropy',optimizer=Adam(lr=
0.01),metrics=['accuracy'])
model.fit_generator(train_generator,epochs=4,validation_data=val
idation_generator)
```

```
Epoch 1/4
75/75 [==============================] - 145s 2s/step - loss: 0
.6950 - accuracy: 0.5033 - val_loss: 0.6955 - val_accuracy: 0.5
000
Epoch 2/4
75/75 [==============================] - 148s 2s/step - loss: 0
.6933 - accuracy: 0.5015 - val_loss: 0.6926 - val_accuracy: 0.5
000
Epoch 3/4
75/75 [==============================] - 156s 2s/step - loss: 0
.6933 - accuracy: 0.4955 - val_loss: 0.6930 - val_accuracy: 0.5
000
Epoch 4/4
75/75 [==============================] - 165s 2s/step - loss: 0
.6933 - accuracy: 0.4980 - val_loss: 0.6960 - val_accuracy: 0.5
000
```

Out[6]:
```
<keras.callbacks.callbacks.History at 0x7f6caa8003d0>
```

使用BN，无DA

In [7]:
```
from keras.layers import BatchNormalization
input_layer = Input([IMSIZE,IMSIZE,3])
x = input_layer
x=BatchNormalization()(x) #使用BatchNormalisation，对input的一个batch的数据每一层进行normalisation
for _ in range(5):
    x = Conv2D(20,[3,3], activation = 'relu')(x)
    x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)

x = Flatten()(x)
x = Dense(84,activation = 'relu')(x)
x = Dense(2,activation = 'softmax')(x)
output_layer=x
model2=Model(input_layer,output_layer)
model2.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 3)       12
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 126, 126, 20)      560
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 63, 63, 20)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 61, 61, 20)        3620
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 30, 30, 20)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 20)        3620
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 12, 12, 20)        3620
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 20)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 4, 4, 20)          3620
_________________________________________________________________
```

```
__
max_pooling2d_10 (MaxPooling (None, 2, 2, 20)          0
_________________________________________________________________
__
flatten_2 (Flatten)          (None, 80)                0
_________________________________________________________________
__
dense_3 (Dense)              (None, 84)                6804
_________________________________________________________________
__
dense_4 (Dense)              (None, 2)                 170
=================================================================
==
Total params: 22,026
Trainable params: 22,020
Non-trainable params: 6
_________________________________________________________________
__
```

参数解释

第一层input layer，无参数。
第二层BatchNormalisation中，因为每一层/通道都有4个参数，3 *4 = 12； 但是miu 和 sigma不需要训练（因为这两个参数可以从数据总直接计算，和模型学习没有关系），所以 non-trainable params 有 3* 2 = 6个。 for loop中含有5个卷积层和5个池化层，除去第一个卷积层，其他卷积层参数为：
conv2D： （3 *3* 20+1） *20 = 181* 20 = 3620 , 其中3 *3 是kernel size，20是上一层遗留通道数，20是卷积核个数。
池化使用[2,2]矩形，步长行列都是2，没有学习项，无参数
for loop的第一个卷积层参数计算如下：
(3* 3 *3 +1）* 20 = 28 *20 = 560 , 其中3* 3 是kernel size，3是上一层遗留通道数，20是卷积核个数。

压扁，无参数

Dense1：（hidden） 80 *84 + 84 = 6720 +84 = 6804
Dense2： 84* 2 + 2 = 168 + 2 = 170
所以总共有22026个参数，比没有BN的模型多了12个参数，其中包含6个non-trainable参数。
没有non-trainable parameters因为BN没有被用到

In [8]:
```
model2.compile(loss='categorical_crossentropy',optimizer=Adam(lr
=0.01),metrics=['accuracy'])
model2.fit_generator(train_generator,epochs=4,validation_data=va
lidation_generator)
```

```
Epoch 1/4
75/75 [==============================] - 170s 2s/step - loss: 0
.6906 - accuracy: 0.5278 - val_loss: 0.6840 - val_accuracy: 0.5
551
Epoch 2/4
75/75 [==============================] - 154s 2s/step - loss: 0
.6717 - accuracy: 0.5803 - val_loss: 0.6709 - val_accuracy: 0.5
965
Epoch 3/4
75/75 [==============================] - 149s 2s/step - loss: 0
.6330 - accuracy: 0.6447 - val_loss: 0.6960 - val_accuracy: 0.6
154
Epoch 4/4
75/75 [==============================] - 153s 2s/step - loss: 0
.6015 - accuracy: 0.6752 - val_loss: 0.5995 - val_accuracy: 0.6
713
```

Out[8]:
```
<keras.callbacks.callbacks.History at 0x7f6c28101c10>
```

使用DA和BN ¶

In [9]:
```
model2.compile(loss='categorical_crossentropy',optimizer=Adam(lr
=0.01),metrics=['accuracy'])
model2.fit_generator(train_generator1,epochs=4,validation_data=v
alidation_generator)
#模型用的是加上了BN的LeNet，训练数据用的是Data Augmentation 以后的trai
n_generator1
```

```
Epoch 1/4
75/75 [==============================] - 224s 3s/step - loss: 0
.6362 - accuracy: 0.6325 - val_loss: 0.6261 - val_accuracy: 0.6
455
Epoch 2/4
75/75 [==============================] - 196s 3s/step - loss: 0
.6074 - accuracy: 0.6682 - val_loss: 0.5617 - val_accuracy: 0.7
128
Epoch 3/4
75/75 [==============================] - 185s 2s/step - loss: 0
.5991 - accuracy: 0.6787 - val_loss: 0.5329 - val_accuracy: 0.7
185
Epoch 4/4
75/75 [==============================] - 187s 2s/step - loss: 0
.5815 - accuracy: 0.6949 - val_loss: 0.4817 - val_accuracy: 0.7
279
```

Out[9]:
```
<keras.callbacks.callbacks.History at 0x7f6c08713190>
```

总结

普通仿LeNet：accuracy 为 0.50 左右，和一个random guess差不多。
模型with BN：最后一个epoch的accuracy 为 0.67，呈增加趋势，如果epoch再高也许还有更高的精度。
模型with DA & BN：平均accruacy（在4个epoch里）达到约0.67， 比只有BN的时候要更精确。