In [26]:
```
from PIL import Image
from glob import glob
import numpy as np
from keras.utils import to_categorical
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from keras.layers import Dense, Flatten,Input,Activation,Conv2D,
MaxPooling2D,BatchNormalization
from keras import Model
import re
```

In [28]:
```
images = glob("/clubear/Lecture 3.3 - Case Study (CIFA10)/cifar1
0/*.*")
N=len(images)
imsize=32
```

In [29]:
```
print(images[0])  #可以看到这个path里有很多个“/”所以不能只用split将图
片的分类取出来
```

```
/clubear/Lecture 3.3 - Case Study (CIFA10)/cifar10/airplane_163
6.png
```

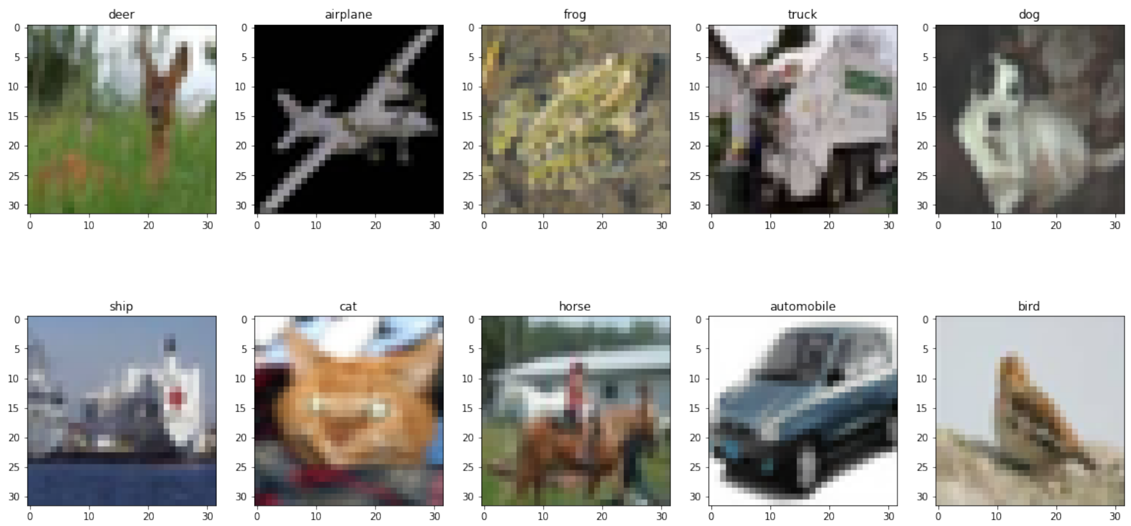
In [44]:
```
Y=[]
X=np.zeros([N,imsize,imsize,3])
for i in range(N):
    Im=Image.open(images[i])
    Im=np.array(Im)/255
    X[i]=Im
    filename = re.split('/|/|.|-|/|/',images[i])[5] #分离多个符
号的function
    cat=filename.split("_")[0] #然后再提取名字
    Y.append(cat)
```

In [45]:
```
unique=list(set(Y))
DICT={}
for i in range(len(unique)): DICT[unique[i]]=i
YY=np.zeros(N)
for i in range(N):
    YY[i]=DICT[Y[i]]
```

In [46]:
```
DICT #10个分类
```

Out[46]:
```
{'deer': 0,
 'airplane': 1,
 'frog': 2,
 'truck': 3,
 'dog': 4,
 'ship': 5,
 'cat': 6,
 'horse': 7,
 'automobile': 8,
 'bird': 9}
```

In [47]:
```
fig,ax=plt.subplots(2,5)
fig.set_figwidth(20)
fig.set_figheight(10)
ax=ax.flatten()
for i in range(len(ax)):
    Im=X[YY==i][0]
    ax[i].imshow(Im)
    ax[i].set_title(unique[i])
```



In [51]:
```
Y=to_categorical(YY) #one-hot变量
X0,X1,Y0,Y1=train_test_split(X,Y,test_size=0.3,random_state=1)
```

In [65]:
```
input_size=[imsize,imsize,3]
input_layer=Input(input_size)
x=input_layer
x=Conv2D(32,[3,3],padding = "same", activation = 'relu')(x)
x=Conv2D(32,[3,3],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(64,[2,2],padding = "same", activation = 'relu')(x)
x=Conv2D(64,[2,2],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(128,[2,2],padding = "same", activation = 'relu')(x)
x=Conv2D(128,[2,2],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Flatten()(x)
x=Dense(10,activation='softmax')(x) #dense只能是10因为有十个分类
output_layer=x
model=Model(input_layer,output_layer)
model.summary()
#感觉一般比较常见的filter个数是32, 64, 128, 有参考VGG模型搭建过程
```

```
Model: "model_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_10 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 16, 16, 64)        8256      
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 16, 16, 64)        16448     
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 8, 8, 128)         32896     
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 8, 8, 128)         65664     
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_11 (Dense)             (None, 10)                20490     
=================================================================
Total params: 153,898
Trainable params: 153,898
Non-trainable params: 0
_________________________________________________________________
```

参数解释

第一层input layer，无参数。
第二层conv2D： （333+1） *32 = 28* 32 = 896, 其中33*是kernel size，3是上一层遗留通道数，32是卷积核个数。*
第三层conv2D： （3332+1） 32 = 289 *32 = 9248, 其中33*是kernel size，32是上一层遗留通道数，32是卷积核个数。
第四层池化没有学习项，无参数
第五层conv2D: （2232+1） *64 = 129* 64 = 8256, 其中22*是kernel size，32是上一层遗留通道数，64是卷积核个数。*
第六层conv2D: （2264+1） 64 = 257 *64 = 16448, 其中22*是kernel size，64是上一层遗留通道数，64是卷积核个数。
第七层池化没有学习项，无参数
第八层conv2D: （2264+1） *128 = 257* 128 =32896, 其中22*是kernel size，64是上一层遗留通道数，128是卷积核个数。*
第九层conv2D: （22128+1） 128 = 513 *128 = 65664, 其中22*是kernel size，128是上一层遗留通道数，128是卷积核个数。
第十层池化没有学习项，无参数
第十一层压扁，无参数
第十二层Dense： 2048 * 10 + 10 = 20480 +10 = 20490
所以总共有153898个参数。

In [66]:
```
from keras.optimizers import Adam
model.compile(optimizer = Adam(0.0001),loss = 'categorical_cross
entropy',metrics = ['accuracy'])
model.fit(X0,Y0,validation_data=(X1,Y1),batch_size=100,epochs=30
)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/30
42000/42000 [==============================] - 4s 85us/step - l
oss: 1.9160 - accuracy: 0.3075 - val_loss: 1.6498 - val_accurac
y: 0.4096
Epoch 2/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.5833 - accuracy: 0.4306 - val_loss: 1.4925 - val_accurac
y: 0.4660
Epoch 3/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.4689 - accuracy: 0.4727 - val_loss: 1.4271 - val_accurac
y: 0.4935
Epoch 4/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.3872 - accuracy: 0.5038 - val_loss: 1.3522 - val_accurac
y: 0.5114
Epoch 5/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.3234 - accuracy: 0.5288 - val_loss: 1.2765 - val_accurac
y: 0.5469
Epoch 6/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.2665 - accuracy: 0.5502 - val_loss: 1.2516 - val_accurac
```

```
y: 0.5573
Epoch 7/30
42000/42000 [==============================] - 3s 76us/step - l
oss: 1.2224 - accuracy: 0.5674 - val_loss: 1.2034 - val_accurac
y: 0.5768
Epoch 8/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.1846 - accuracy: 0.5805 - val_loss: 1.2328 - val_accurac
y: 0.5668
Epoch 9/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.1494 - accuracy: 0.5923 - val_loss: 1.1447 - val_accurac
y: 0.5980
Epoch 10/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.1183 - accuracy: 0.6064 - val_loss: 1.1376 - val_accurac
y: 0.5990
Epoch 11/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.0885 - accuracy: 0.6185 - val_loss: 1.0841 - val_accurac
y: 0.6216
Epoch 12/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 1.0525 - accuracy: 0.6304 - val_loss: 1.1015 - val_accurac
y: 0.6127
Epoch 13/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.0343 - accuracy: 0.6371 - val_loss: 1.0535 - val_accurac
y: 0.6346
Epoch 14/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.0103 - accuracy: 0.6474 - val_loss: 1.0330 - val_accurac
y: 0.6422
Epoch 15/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.9935 - accuracy: 0.6521 - val_loss: 1.0405 - val_accurac
y: 0.6402
Epoch 16/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 0.9731 - accuracy: 0.6625 - val_loss: 1.0419 - val_accurac
y: 0.6402
Epoch 17/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 0.9515 - accuracy: 0.6688 - val_loss: 0.9928 - val_accurac
y: 0.6601
Epoch 18/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 0.9371 - accuracy: 0.6734 - val_loss: 0.9684 - val_accurac
y: 0.6658
Epoch 19/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.9153 - accuracy: 0.6815 - val_loss: 0.9648 - val_accurac
y: 0.6660
Epoch 20/30
42000/42000 [==============================] - 3s 78us/step - l
```

```
oss: 0.9008 - accuracy: 0.6874 - val_loss: 0.9470 - val_accurac
y: 0.6727
Epoch 21/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 0.8848 - accuracy: 0.6947 - val_loss: 0.9389 - val_accurac
y: 0.6772
Epoch 22/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8771 - accuracy: 0.6960 - val_loss: 0.9355 - val_accurac
y: 0.6780
Epoch 23/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8566 - accuracy: 0.7040 - val_loss: 0.9652 - val_accurac
y: 0.6660
Epoch 24/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8449 - accuracy: 0.7087 - val_loss: 0.9600 - val_accurac
y: 0.6691
Epoch 25/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8286 - accuracy: 0.7147 - val_loss: 0.9189 - val_accurac
y: 0.6846
Epoch 26/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8176 - accuracy: 0.7183 - val_loss: 0.9186 - val_accurac
y: 0.6856
Epoch 27/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.8062 - accuracy: 0.7223 - val_loss: 0.8986 - val_accurac
y: 0.6910
Epoch 28/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 0.7931 - accuracy: 0.7284 - val_loss: 0.9013 - val_accurac
y: 0.6919
Epoch 29/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 0.7770 - accuracy: 0.7335 - val_loss: 0.9034 - val_accurac
y: 0.6921
Epoch 30/30
42000/42000 [==============================] - 3s 79us/step - l
oss: 0.7711 - accuracy: 0.7332 - val_loss: 0.8743 - val_accurac
y: 0.6987
```

Out[66]:
```
<keras.callbacks.callbacks.History at 0x7fecea6b7a10>
```

accuracy是0.73， 参数比Lecture 3.3中多但是accuracy低了0.03

与其他模型对比

这是Lecture 3.3中给出的模型

In [75]:
```
input_size=[imsize,imsize,3]
input_layer=Input(input_size)
x=input_layer
x=Conv2D(100,[2,2],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(100,[2,2],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(100,[2,2],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Flatten()(x)

x=Dense(10,activation='softmax')(x)
output_layer=x
model1=Model(input_layer,output_layer)
model1.summary()
```

```
Model: "model_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 32, 32, 100)       1300
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 16, 16, 100)       0
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 16, 16, 100)       40100
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 8, 8, 100)         0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 8, 8, 100)         40100
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 4, 4, 100)         0
_________________________________________________________________
flatten_13 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                16010
=================================================================
Total params: 97,510
Trainable params: 97,510
Non-trainable params: 0
_________________________________________________________________
```

In [68]:

```
from keras.optimizers import Adam
model1.compile(optimizer = Adam(0.00001),loss = 'categorical_crossentropy',metrics = ['accuracy'])
model1.fit(X0,Y0,validation_data=(X1,Y1),batch_size=100,epochs=30)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/30
42000/42000 [==============================] - 3s 79us/step - loss: 2.2895 - accuracy: 0.1458 - val_loss: 2.2734 - val_accuracy: 0.1776
Epoch 2/30
42000/42000 [==============================] - 3s 74us/step - loss: 2.2471 - accuracy: 0.2109 - val_loss: 2.2086 - val_accuracy: 0.2498
```

```
Epoch 3/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 2.1599 - accuracy: 0.2574 - val_loss: 2.1024 - val_accurac
y: 0.2772
Epoch 4/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 2.0626 - accuracy: 0.2898 - val_loss: 2.0186 - val_accurac
y: 0.3149
Epoch 5/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.9943 - accuracy: 0.3203 - val_loss: 1.9580 - val_accurac
y: 0.3410
Epoch 6/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.9394 - accuracy: 0.3419 - val_loss: 1.9058 - val_accurac
y: 0.3567
Epoch 7/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.8900 - accuracy: 0.3564 - val_loss: 1.8571 - val_accurac
y: 0.3701
Epoch 8/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.8456 - accuracy: 0.3707 - val_loss: 1.8158 - val_accurac
y: 0.3810
Epoch 9/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.8093 - accuracy: 0.3788 - val_loss: 1.7829 - val_accurac
y: 0.3878
Epoch 10/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.7790 - accuracy: 0.3875 - val_loss: 1.7560 - val_accurac
y: 0.3958
Epoch 11/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.7538 - accuracy: 0.3934 - val_loss: 1.7309 - val_accurac
y: 0.3997
Epoch 12/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.7306 - accuracy: 0.3991 - val_loss: 1.7105 - val_accurac
y: 0.4037
Epoch 13/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.7108 - accuracy: 0.4025 - val_loss: 1.6900 - val_accurac
y: 0.4087
Epoch 14/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.6929 - accuracy: 0.4088 - val_loss: 1.6743 - val_accurac
y: 0.4154
Epoch 15/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.6765 - accuracy: 0.4119 - val_loss: 1.6568 - val_accurac
y: 0.4192
Epoch 16/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.6603 - accuracy: 0.4166 - val_loss: 1.6421 - val_accurac
```

```
y: 0.4218
Epoch 17/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.6453 - accuracy: 0.4214 - val_loss: 1.6275 - val_accurac
y: 0.4235
Epoch 18/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.6316 - accuracy: 0.4263 - val_loss: 1.6148 - val_accurac
y: 0.4293
Epoch 19/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.6189 - accuracy: 0.4280 - val_loss: 1.6014 - val_accurac
y: 0.4333
Epoch 20/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.6069 - accuracy: 0.4330 - val_loss: 1.5900 - val_accurac
y: 0.4373
Epoch 21/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.5956 - accuracy: 0.4349 - val_loss: 1.5796 - val_accurac
y: 0.4418
Epoch 22/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.5850 - accuracy: 0.4380 - val_loss: 1.5705 - val_accurac
y: 0.4480
Epoch 23/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.5747 - accuracy: 0.4416 - val_loss: 1.5588 - val_accurac
y: 0.4489
Epoch 24/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.5647 - accuracy: 0.4446 - val_loss: 1.5512 - val_accurac
y: 0.4541
Epoch 25/30
42000/42000 [==============================] - 3s 75us/step - l
oss: 1.5554 - accuracy: 0.4462 - val_loss: 1.5407 - val_accurac
y: 0.4577
Epoch 26/30
42000/42000 [==============================] - 3s 74us/step - l
oss: 1.5465 - accuracy: 0.4509 - val_loss: 1.5324 - val_accurac
y: 0.4585
Epoch 27/30
42000/42000 [==============================] - 3s 76us/step - l
oss: 1.5379 - accuracy: 0.4530 - val_loss: 1.5236 - val_accurac
y: 0.4591
Epoch 28/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 1.5294 - accuracy: 0.4563 - val_loss: 1.5162 - val_accurac
y: 0.4630
Epoch 29/30
42000/42000 [==============================] - 3s 78us/step - l
oss: 1.5211 - accuracy: 0.4594 - val_loss: 1.5082 - val_accurac
y: 0.4640
Epoch 30/30
42000/42000 [==============================] - 3s 77us/step - l
```

```
oss: 1.5130 - accuracy: 0.4624 - val_loss: 1.5017 - val_accurac
y: 0.4665
```

Out[68]: <keras.callbacks.callbacks.History at 0x7feceb0153d0>

Learning rate是0.00001，得到的accuracy只有0.46。个人怀疑是learning rate太小，于是尝试了0.001，如下。

In [72]:

```
from keras.optimizers import Adam
model1.compile(optimizer = Adam(0.0001),loss = 'categorical_cros
sentropy',metrics = ['accuracy'])
model1.fit(X0,Y0,validation_data=(X1,Y1),batch_size=100,epochs=3
0)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/30
42000/42000 [==============================] - 3s 77us/step - l
oss: 0.9587 - accuracy: 0.6702 - val_loss: 0.9950 - val_accurac
y: 0.6555
Epoch 2/30
42000/42000 [==============================] - 3s 73us/step - l
oss: 0.9466 - accuracy: 0.6727 - val_loss: 0.9706 - val_accurac
y: 0.6659
Epoch 3/30
42000/42000 [==============================] - 3s 73us/step - l
oss: 0.9381 - accuracy: 0.6769 - val_loss: 0.9845 - val_accurac
y: 0.6570
Epoch 4/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.9293 - accuracy: 0.6802 - val_loss: 1.0067 - val_accurac
y: 0.6478
Epoch 5/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.9252 - accuracy: 0.6810 - val_loss: 0.9705 - val_accurac
y: 0.6656
Epoch 6/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.9160 - accuracy: 0.6827 - val_loss: 0.9600 - val_accurac
y: 0.6699
Epoch 7/30
42000/42000 [==============================] - 3s 73us/step - l
oss: 0.9104 - accuracy: 0.6847 - val_loss: 0.9706 - val_accurac
y: 0.6647
Epoch 8/30
42000/42000 [==============================] - 3s 73us/step - l
oss: 0.9046 - accuracy: 0.6892 - val_loss: 0.9534 - val_accurac
y: 0.6704
Epoch 9/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8929 - accuracy: 0.6923 - val_loss: 0.9510 - val_accurac
y: 0.6762
Epoch 10/30
42000/42000 [==============================] - 3s 71us/step - l
```

```
oss: 0.8880 - accuracy: 0.6939 - val_loss: 0.9376 - val_accurac
y: 0.6779
Epoch 11/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8809 - accuracy: 0.6968 - val_loss: 0.9476 - val_accurac
y: 0.6713
Epoch 12/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8774 - accuracy: 0.6987 - val_loss: 0.9303 - val_accurac
y: 0.6813
Epoch 13/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8708 - accuracy: 0.6994 - val_loss: 0.9291 - val_accurac
y: 0.6855
Epoch 14/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8643 - accuracy: 0.7031 - val_loss: 0.9271 - val_accurac
y: 0.6814
Epoch 15/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8624 - accuracy: 0.7025 - val_loss: 0.9157 - val_accurac
y: 0.6842
Epoch 16/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8503 - accuracy: 0.7080 - val_loss: 0.9220 - val_accurac
y: 0.6841
Epoch 17/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8480 - accuracy: 0.7075 - val_loss: 0.9304 - val_accurac
y: 0.6816
Epoch 18/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8419 - accuracy: 0.7102 - val_loss: 0.9199 - val_accurac
y: 0.6826
Epoch 19/30
42000/42000 [==============================] - 3s 71us/step - l
oss: 0.8361 - accuracy: 0.7128 - val_loss: 0.9074 - val_accurac
y: 0.6912
Epoch 20/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8295 - accuracy: 0.7146 - val_loss: 0.8963 - val_accurac
y: 0.6953
Epoch 21/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8255 - accuracy: 0.7159 - val_loss: 0.9182 - val_accurac
y: 0.6841
Epoch 22/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8221 - accuracy: 0.7174 - val_loss: 0.8929 - val_accurac
y: 0.6953
Epoch 23/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8162 - accuracy: 0.7196 - val_loss: 0.8963 - val_accurac
y: 0.6915
Epoch 24/30
```

```
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8091 - accuracy: 0.7223 - val_loss: 0.9041 - val_accurac
y: 0.6881
Epoch 25/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8078 - accuracy: 0.7220 - val_loss: 0.8992 - val_accurac
y: 0.6924
Epoch 26/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.8019 - accuracy: 0.7262 - val_loss: 0.8908 - val_accurac
y: 0.6967
Epoch 27/30
42000/42000 [==============================] - 3s 73us/step - l
oss: 0.7998 - accuracy: 0.7245 - val_loss: 0.9041 - val_accurac
y: 0.6919
Epoch 28/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.7901 - accuracy: 0.7295 - val_loss: 0.8862 - val_accurac
y: 0.6966
Epoch 29/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.7882 - accuracy: 0.7306 - val_loss: 0.8866 - val_accurac
y: 0.6984
Epoch 30/30
42000/42000 [==============================] - 3s 72us/step - l
oss: 0.7845 - accuracy: 0.7313 - val_loss: 0.8757 - val_accurac
y: 0.7011
```

Out[72]:
```
<keras.callbacks.callbacks.History at 0x7fece856c8d0>
```

同样的model，现在的accuracy是0.7313，比lecture3.3中的0.7662低。 但应该不影响模型搭建。

和自己写的第一个模型strcture差不多，差别在于将卷积核改成了valid

In [73]:
```
input_size=[imsize,imsize,3]
input_layer=Input(input_size)
x=input_layer
x=Conv2D(32,[3,3],padding = "valid", activation = 'relu')(x)
x=Conv2D(32,[3,3],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(64,[2,2],padding = "valid", activation = 'relu')(x)
x=Conv2D(64,[2,2],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Conv2D(128,[2,2],padding = "valid", activation = 'relu')(x)
x=Conv2D(128,[2,2],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2])(x)
x=Flatten()(x)
x=Dense(10,activation='softmax')(x) #dense只能是10因为有十个分类
output_layer=x
model2=Model(input_layer,output_layer)
model2.summary()
```

```
Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_13 (InputLayer)        (None, 32, 32, 3)         0         
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 30, 30, 32)        896       
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 28, 28, 32)        9248      
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 13, 13, 64)        8256      
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 12, 12, 64)        16448     
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 6, 6, 64)          0         
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 5, 5, 128)         32896     
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 4, 4, 128)         65664     
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 2, 2, 128)         0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 512)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                5130      
=================================================================
Total params: 138,538
Trainable params: 138,538
Non-trainable params: 0
_________________________________________________________________
```

参数解释

前面都和第一个模型相同，唯一的差别在于压扁以后的vector只有512 （因为valid比same处理得出的矩阵要小）
第十二层Dense： 512 * 10 + 10 = 5120 +10 = 5130
所以总共有153538个参数。

In [77]:
```
model2.compile(optimizer = Adam(0.0001),loss = 'categorical_crossentropy',metrics = ['accuracy'])
model2.fit(X0,Y0,validation_data=(X1,Y1),batch_size=100,epochs=30)
```

```
Train on 42000 samples, validate on 18000 samples
Epoch 1/30
42000/42000 [==============================] - 3s 74us/step - loss: 0.8760 - accuracy: 0.6997 - val_loss: 0.9806 - val_accuracy: 0.6640
Epoch 2/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.8695 - accuracy: 0.7015 - val_loss: 0.9721 - val_accuracy: 0.6652
Epoch 3/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.8548 - accuracy: 0.7076 - val_loss: 0.9660 - val_accuracy: 0.6689
Epoch 4/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.8436 - accuracy: 0.7105 - val_loss: 0.9782 - val_accuracy: 0.6672
Epoch 5/30
42000/42000 [==============================] - 3s 69us/step - loss: 0.8364 - accuracy: 0.7163 - val_loss: 0.9745 - val_accuracy: 0.6638
Epoch 6/30
42000/42000 [==============================] - 3s 69us/step - loss: 0.8213 - accuracy: 0.7195 - val_loss: 0.9695 - val_accuracy: 0.6693
Epoch 7/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.8115 - accuracy: 0.7227 - val_loss: 0.9680 - val_accuracy: 0.6697
Epoch 8/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.8047 - accuracy: 0.7247 - val_loss: 0.9651 - val_accuracy: 0.6727
Epoch 9/30
42000/42000 [==============================] - 3s 68us/step - loss: 0.7946 - accuracy: 0.7281 - val_loss: 0.9497 - val_accuracy: 0.6762
Epoch 10/30
42000/42000 [==============================] - 3s 69us/step - loss: 0.7875 - accuracy: 0.7313 - val_loss: 0.9523 - val_accuracy: 0.6766
Epoch 11/30
42000/42000 [==============================] - 3s 68us/step - l
```

```
oss: 0.7754 - accuracy: 0.7347 - val_loss: 0.9406 - val_accurac
y: 0.6821
Epoch 12/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7689 - accuracy: 0.7397 - val_loss: 0.9394 - val_accurac
y: 0.6801
Epoch 13/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7593 - accuracy: 0.7403 - val_loss: 0.9706 - val_accurac
y: 0.6722
Epoch 14/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7475 - accuracy: 0.7468 - val_loss: 0.9470 - val_accurac
y: 0.6823
Epoch 15/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7377 - accuracy: 0.7507 - val_loss: 0.9371 - val_accurac
y: 0.6854
Epoch 16/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7313 - accuracy: 0.7521 - val_loss: 0.9483 - val_accurac
y: 0.6779
Epoch 17/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7217 - accuracy: 0.7541 - val_loss: 0.9363 - val_accurac
y: 0.6849
Epoch 18/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7165 - accuracy: 0.7574 - val_loss: 0.9500 - val_accurac
y: 0.6846
Epoch 19/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.7073 - accuracy: 0.7584 - val_loss: 0.9377 - val_accurac
y: 0.6845
Epoch 20/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6955 - accuracy: 0.7654 - val_loss: 0.9469 - val_accurac
y: 0.6829
Epoch 21/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6884 - accuracy: 0.7672 - val_loss: 0.9507 - val_accurac
y: 0.6853
Epoch 22/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6800 - accuracy: 0.7698 - val_loss: 0.9261 - val_accurac
y: 0.6887
Epoch 23/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6709 - accuracy: 0.7722 - val_loss: 0.9346 - val_accurac
y: 0.6888
Epoch 24/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6599 - accuracy: 0.7755 - val_loss: 0.9268 - val_accurac
y: 0.6927
Epoch 25/30
```

```
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6540 - accuracy: 0.7789 - val_loss: 0.9272 - val_accurac
y: 0.6948
Epoch 26/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6451 - accuracy: 0.7809 - val_loss: 0.9197 - val_accurac
y: 0.6896
Epoch 27/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6359 - accuracy: 0.7851 - val_loss: 0.9554 - val_accurac
y: 0.6862
Epoch 28/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6314 - accuracy: 0.7864 - val_loss: 0.9362 - val_accurac
y: 0.6952
Epoch 29/30
42000/42000 [==============================] - 3s 68us/step - l
oss: 0.6227 - accuracy: 0.7883 - val_loss: 0.9202 - val_accurac
y: 0.6967
Epoch 30/30
42000/42000 [==============================] - 3s 69us/step - l
oss: 0.6133 - accuracy: 0.7925 - val_loss: 0.9668 - val_accurac
y: 0.6831
```

Out[77]:
```
<keras.callbacks.callbacks.History at 0x7fece76bbf50>
```

可以看到accuracy是0.79，比相同模型same padding高，但再run一次得到的accruacy不必same padding高，所以应该不会造成太大差别。

In [ ]: