In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

This is an interesting task, I have experienced a lot, including writing code with colab, because the caltech256 data set is particularly large, split it into four 4GB files by pickle, and then to shorten the time of each raw data read, first time When reading, the data is saved to a CSV file, so that when the program is tested later, it can be read directly from the CSV file, but the CSV file is particularly large, and the x_train.csv file size is 16 GB, which directly causes the RAM to explode, and then more I tried to directly read the data in the pickle file, split it into the train and test parts, and finally run the program. The RAM usage is 25.33GB, and the total RAM is only 25.51GB. And because of the excessive use of colab GPU resources, Google has blocked the account. But in the end, after asking classmates and Dr.yang, as well as online to find a solution, finally successfully used the VGG16 to train the caltech256 data set.

There are two parts : the first one is training from scratch the accuracy is 24% the second one is training from pre-trained the accuracy is 55.7%

In [6]:
```
!/opt/bin/nvidia-smi
```

```
Sun Nov 10 22:24:40 2019
+-----------------------------------------------------------------------------
-----+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 10.1
|
|-------------------------------+----------------------+-----------------
-----+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr.
ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Comput
e M. |
|===============================+======================+=================
=====|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |
0 |
| N/A   34C    P0    25W / 250W |      0MiB / 16280MiB |      0%      Def
ault |
+-------------------------------+----------------------+-----------------
-----+

+-----------------------------------------------------------------------------
-----+
| Processes:                                                       GPU Me
mory |
|  GPU       PID   Type   Process name                             Usage
|
|=============================================================================
=====|
|  No running processes found
|
+-----------------------------------------------------------------------------
-----+
```

Load the dataset from pickle file

By http://places2.csail.mit.edu/PAMI_places (http://places2.csail.mit.edu/PAMI_places) (given by Dr.yang)and reference code ,learning the method to generate the training and testing dataset.

In [0]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tarfile
import cv2
import pickle
%matplotlib inline
import tarfile
%matplotlib inline
```

load the picklepath from google drive

In [0]:
```
import os
picklepath ='/content/drive/My Drive/Colab Notebooks/VGG16_Practice/pickle'
os.chdir(picklepath)
```

Using data_process.ipnyb divide data to four pickle file , so I should load those file

In [0]:
```
import pickle
pickle_in = open("pickle_all_images_df1.pickle","rb")
all_images_df1 = pickle.load(pickle_in)
```

In [0]:
```
pickle_in = open("pickle_all_images_df2.pickle","rb")
all_images_df2 = pickle.load(pickle_in)
```

In [0]:
```
pickle_in = open("pickle_all_images_df3.pickle","rb")
all_images_df3 = pickle.load(pickle_in)
```

In [0]:
```
pickle_in = open("pickle_all_classes.pickle","rb")
all_classes = pickle.load(pickle_in)
```

set the img size is 128

In [0]:
```
img_size = 128
```

concatenate those four array together

In [0]:
```
all_images = np.concatenate((all_images_df1, all_images_df2,all_images_df3)
```

In [10]:
```
all_images.shape
```

Out[10]:
```
(30607, 49152)
```

del those pre-data, in order to release RAM

In [0]:
```
del all_images_df1
del all_images_df2
del all_images_df3
```

In [12]:
```
all_images.shape
```

Out[12]:
```
(30607, 49152)
```

In [0]:
```
all_images.shape
```

In [0]:
```
# train_total = np.concatenate((train_x, train_y), axis=1)
# test_total = np.concatenate((test_x, test_y), axis=1)
# df_train = pd.DataFrame(train_total)
# df_test = pd.DataFrame(test_total)
```

using pandas to implement one hot encode

In [0]:
```
all_classes = pd.get_dummies(all_classes)
```

In [14]:
```
all_classes.shape
```

Out[14]: (30607, 257)

In [0]:
```
all_images = np.array(all_images)
```

In [16]:
```
all_images.shape
```

Out[16]: (30607, 49152)

split all_images to X_train part and X_test part. split all_classes to y_train part and y_test part

In [0]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(all_images, all_classes
```

In [19]:
```
X_train.shape
```

Out[19]: (27546, 49152)

In [15]:
```
X_test.shape
```

Out[15]: (3061, 49152)

In [23]:
```
y_train.shape
```

Out[23]: (27546, 257)

In [24]:
```
y_test.shape
```

Out[24]: (3061, 257)

In [0]:
```
X_train[0].shape
```

Because using VGG16 network,so I should fit the dimension,so using reshape method to change dimension

In [0]:
```
X_train = X_train.reshape(-1,img_size,img_size,3)
```

In [21]:
```
X_train.shape
```

Out[21]: (27546, 128, 128, 3)

In [0]:
```
X_test = X_test.reshape(-1,img_size,img_size,3)
```

In [21]:
```
X_test.shape
```
Out[21]:
```
(3061, 128, 128, 3)
```

Save X_train, X_test, y_train, y_test to CSV file

In [0]:
```
df_x_train = pd.DataFrame(X_train)
df_x_test = pd.DataFrame(X_test)
df_y_train = pd.DataFrame(y_train)
df_y_test = pd.DataFrame(y_test)
```

In [0]:
```
df_x_train.to_csv('/content/drive/My Drive/Colab Notebooks/VGG16_Practice/x
```

In [0]:
```
df_x_test.to_csv('/content/drive/My Drive/Colab Notebooks/VGG16_Practice/x_
```

In [0]:
```
df_y_train.to_csv('/content/drive/My Drive/Colab Notebooks/VGG16_Practice/y
```

In [0]:
```
df_y_test.to_csv('/content/drive/My Drive/Colab Notebooks/VGG16_Practice/y_
```

In [0]:
```
del df_x_test
del df_y_train
del df_y_test
```

In [0]:
```

```

Type *Markdown* and LaTeX: α^2

Load train_x test_x train_y test_y from CSV

In [0]:
```
import numpy as np
import pandas as pd
```

In [0]:
```
train_x = pd.read_csv('/Users/wangxiang/Code/Jupyter/VGG16_Practice/pickle/
```

In [0]:
```
test_x = pd.read_csv('/Users/wangxiang/Code/Jupyter/VGG16_Practice/pickle/x
```

In [0]:
```
train_y = pd.read_csv('/Users/wangxiang/Code/Jupyter/VGG16_Practice/pickle/
```

In [0]:
```
test_y = pd.read_csv('/Users/wangxiang/Code/Jupyter/VGG16_Practice/pickle/y
```

In [0]:
```
train_x = train_x.values
(train_x.shape)
```

In [0]:
```
# X_train = [cv2.cvtColor(cv2.resize(i,(224,224)), cv2.COLOR_GRAY2BGR) for
# X_train = np.concatenate([arr[np.newaxis] for arr in X_train]).astype('fl

# X_test  = [cv2.cvtColor(cv2.resize(i,(224,224)), cv2.COLOR_GRAY2BGR) for
# X_test  = np.concatenate([arr[np.newaxis] for arr in X_test] ).astype('fl

# print("X_train shape :",X_train.shape)
# print("X_test shape:",X_test.shape)
```

Normalize the images Section

If you execute the following code, the accuracy will be reduced to 10%

In [22]:
```
from tensorflow.keras.applications.vgg16 import preprocess_input

X_test = preprocess_input(X_test)
X_train = preprocess_input(X_train)
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you [upgrade (https://www.tensorflow.org/guide/migrate)](https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: [more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb)](https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

VGG16 from Scratch

this part using VGG16 model without weights

In [0]:
```
import os
import keras
from keras.models import Model
from keras.layers import Dense,Flatten,Dropout
from keras import datasets
from keras.applications.vgg16 import VGG16
```

Setting weights=None and layer.trainable=True,which is important

In [42]:
```
model = VGG16(weights=None, include_top=False, input_shape=(128,128,3), cla
for layer in model.layers:
  layer.trainable=True
print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
None
```

Setting top-layer:add three Dense layer,especially the last Dense layer is softmax layer with 257 neruons to classify label

In [0]:
```
from keras.layers import Flatten, Dense
from keras.models import Model

x = model.get_layer('block5_pool').output
x = Flatten(name='flatten')(x)
x = Dense(512, activation='relu', name='fc1')(x)
x = Dense(4096, activation='relu', name='fc2')(x)
x = Dense(257, activation='softmax', name='predictions')(x)

model_caltech_vgg = Model(inputs=model.input, outputs=x)
```

In [44]:
```
model_caltech_vgg.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
fc1 (Dense)                  (None, 512)               4194816
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              2101248
_________________________________________________________________
predictions (Dense)          (None, 257)               1052929
=================================================================
Total params: 22,063,681
Trainable params: 22,063,681
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
def learning_rate_schedule(epoch):
    if epoch <= 10:
        return 1e-4 # 0.00001
    elif epoch <= 20:
        return 1e-5
    elif epoch <= 30:
        return 1e-6
    else:
        return 1e-7
    return LR
```

In [0]:
```
from keras import optimizers
from keras.callbacks import EarlyStopping
```

setting epochs=30 and each batch_size=32

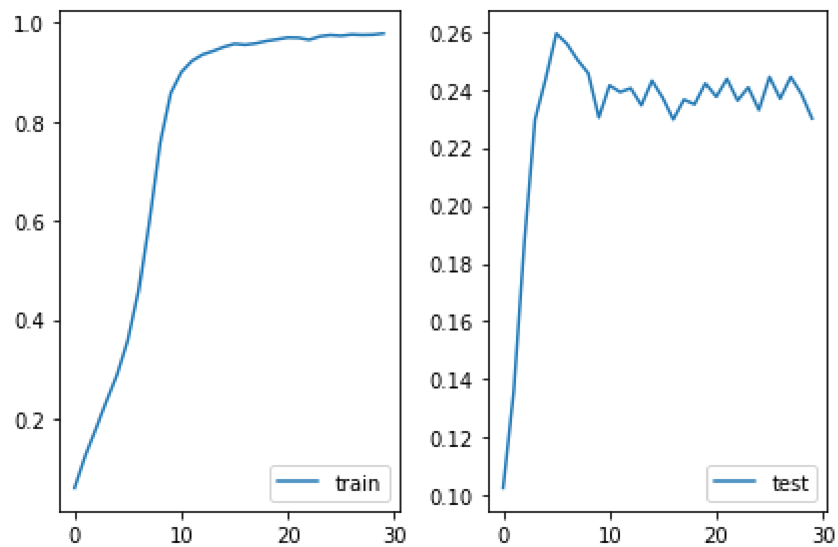
In [47]:
```
model_caltech_vgg.compile(
    loss='categorical_crossentropy',
    optimizer=optimizers.adam(lr=0.0001), metrics=['accuracy']
)
keras.callbacks.LearningRateScheduler(learning_rate_schedule)
history = model_caltech_vgg.fit(
    X_train, y_train,
    batch_size=32, shuffle=True, epochs=30,
    validation_data=(X_test, y_test)
)
```

```
Train on 27546 samples, validate on 3061 samples
Epoch 1/30
27546/27546 [==============================] - 88s 3ms/step - loss: 5.224
0 - acc: 0.0565 - val_loss: 4.9047 - val_acc: 0.0856
Epoch 2/30
27546/27546 [==============================] - 83s 3ms/step - loss: 4.614
6 - acc: 0.1212 - val_loss: 4.4271 - val_acc: 0.1297
Epoch 3/30
27546/27546 [==============================] - 82s 3ms/step - loss: 4.149
2 - acc: 0.1747 - val_loss: 4.0834 - val_acc: 0.1875
Epoch 4/30
27546/27546 [==============================] - 82s 3ms/step - loss: 3.757
0 - acc: 0.2279 - val_loss: 3.9624 - val_acc: 0.2130
Epoch 5/30
27546/27546 [==============================] - 82s 3ms/step - loss: 3.378
5 - acc: 0.2803 - val_loss: 3.8363 - val_acc: 0.2411
Epoch 6/30
27546/27546 [==============================] - 82s 3ms/step - loss: 2.928
6 - acc: 0.3484 - val_loss: 3.7786 - val_acc: 0.2509
Epoch 7/30
27546/27546 [==============================] - 83s 3ms/step - loss: 2.388
8 - acc: 0.4405 - val_loss: 3.9854 - val_acc: 0.2594
Epoch 8/30
27546/27546 [==============================] - 82s 3ms/step - loss: 1.720
1 - acc: 0.5657 - val_loss: 4.5628 - val_acc: 0.2496
Epoch 9/30
27546/27546 [==============================] - 82s 3ms/step - loss: 1.020
9 - acc: 0.7243 - val_loss: 5.6658 - val_acc: 0.2440
Epoch 10/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.580
4 - acc: 0.8357 - val_loss: 6.6711 - val_acc: 0.2346
Epoch 11/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.360
9 - acc: 0.8962 - val_loss: 7.1479 - val_acc: 0.2375
Epoch 12/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.268
9 - acc: 0.9194 - val_loss: 7.7502 - val_acc: 0.2418
Epoch 13/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.218
6 - acc: 0.9343 - val_loss: 8.2997 - val_acc: 0.2372
Epoch 14/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.197
2 - acc: 0.9431 - val_loss: 7.3544 - val_acc: 0.2395
Epoch 15/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.168
```

```
1 - acc: 0.9511 - val_loss: 8.3142 - val_acc: 0.2414
Epoch 16/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.160
8 - acc: 0.9526 - val_loss: 8.1826 - val_acc: 0.2401
Epoch 17/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.142
2 - acc: 0.9602 - val_loss: 8.4789 - val_acc: 0.2372
Epoch 18/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.144
1 - acc: 0.9593 - val_loss: 7.8706 - val_acc: 0.2421
Epoch 19/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.116
9 - acc: 0.9653 - val_loss: 8.1067 - val_acc: 0.2382
Epoch 20/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.136
1 - acc: 0.9603 - val_loss: 8.0828 - val_acc: 0.2398
Epoch 21/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.103
1 - acc: 0.9703 - val_loss: 8.2957 - val_acc: 0.2336
Epoch 22/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.112
4 - acc: 0.9674 - val_loss: 8.2640 - val_acc: 0.2293
Epoch 23/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.102
9 - acc: 0.9713 - val_loss: 8.2929 - val_acc: 0.2418
Epoch 24/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.091
9 - acc: 0.9726 - val_loss: 8.0287 - val_acc: 0.2421
Epoch 25/30
27546/27546 [==============================] - 82s 3ms/step - loss: 0.090
8 - acc: 0.9731 - val_loss: 8.0642 - val_acc: 0.2418
Epoch 26/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.084
3 - acc: 0.9757 - val_loss: 8.3474 - val_acc: 0.2437
Epoch 27/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.089
7 - acc: 0.9738 - val_loss: 8.8847 - val_acc: 0.2470
Epoch 28/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.084
9 - acc: 0.9763 - val_loss: 8.5041 - val_acc: 0.2385
Epoch 29/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.077
2 - acc: 0.9787 - val_loss: 8.0128 - val_acc: 0.2440
Epoch 30/30
27546/27546 [==============================] - 83s 3ms/step - loss: 0.075
0 - acc: 0.9785 - val_loss: 8.0370 - val_acc: 0.2404
```

By train 30 epochs the VGG16 from scratch model give me 28.29% testing accuracy

In [31]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.subplot(1, 2, 1)
plt.plot(history.history['acc'])
plt.legend(['train'], loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['val_acc'])
plt.legend(['test'], loc='lower right')

plt.tight_layout()

plt.show()
```



VGG16 Pre-trained

this part using VGG16 pre-trained model and run 3 times

In [32]:
```
model = VGG16(weights='imagenet', include_top=False, input_shape=(128,128,3

for layer in model.layers:
    layer.trainable = False
print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
None
```

Setting top layer : three Dense layer and last layer is softmax layer

In [0]:
```
from keras.layers import Flatten, Dense
from keras.models import Model

x = model.get_layer('block5_pool').output
x = Flatten(name='flatten')(x)
x = Dense(512, activation='relu', name='fc1')(x)
x = Dense(4096, activation='relu', name='fc2')(x)
x = Dense(257, activation='softmax', name='predictions')(x)
model_updated = Model(inputs=model.input, outputs=x)
#model_caltech_vgg_pre = Model(inputs=model.input, outputs=x)
```

In [0]:
```
def learning_rate_schedule(epoch):
    if epoch <= 10:
        return 1e-4 # 0.00001
    elif epoch <= 20:
        return 1e-5
    elif epoch <= 30:
        return 1e-6
    else:
        return 1e-7
    return LR
```

save weights for each time

In [0]:
```
model_updated.save_weights('model_caltech_initial.h5')
```

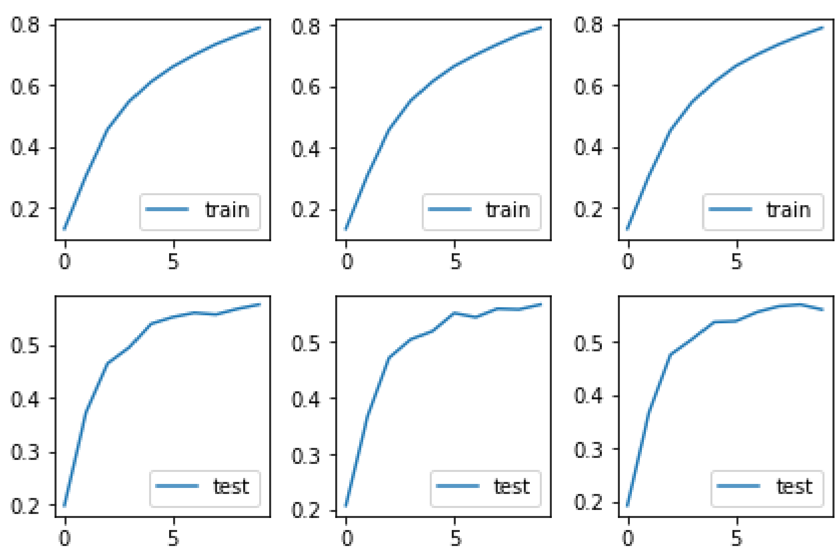
In [0]:
```
from keras import optimizers
from keras.callbacks import EarlyStopping
```

In [39]:
```
._updated.save_weights('model_caltech_initial.h5')
ıing_runs = []
. in range(3):
#model_updated.compile(loss='categorical_crossentropy', optimizer='sgd', met
ıodel_updated.compile(loss='mean_squared_error', optimizer=optimizers.adam(l
ceras.callbacks.LearningRateScheduler(learning_rate_schedule)
ıistory = model_updated.fit(X_train, y_train, batch_size=32,shuffle=True, ve
training_runs.append(history)
ıodel_updated.get_weights()
.f i == 2:
    model_updated.save_weights('model1.h5')
else:
    model_updated.load_weights('model_caltech_initial.h5')
print()
```

```
Train on 27546 samples, validate on 3061 samples
Epoch 1/10
27546/27546 [==============================] - 38s 1ms/step - loss: 0.003
6 - acc: 0.1293 - val_loss: 0.0034 - val_acc: 0.1973
Epoch 2/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.003
1 - acc: 0.3033 - val_loss: 0.0029 - val_acc: 0.3728
Epoch 3/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.002
6 - acc: 0.4542 - val_loss: 0.0025 - val_acc: 0.4655
Epoch 4/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.002
2 - acc: 0.5458 - val_loss: 0.0024 - val_acc: 0.4959
Epoch 5/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
9 - acc: 0.6105 - val_loss: 0.0022 - val_acc: 0.5407
Epoch 6/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
7 - acc: 0.6597 - val_loss: 0.0022 - val_acc: 0.5534
Epoch 7/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
5 - acc: 0.6986 - val_loss: 0.0022 - val_acc: 0.5616
Epoch 8/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
3 - acc: 0.7338 - val_loss: 0.0022 - val_acc: 0.5583
Epoch 9/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
2 - acc: 0.7620 - val_loss: 0.0022 - val_acc: 0.5691
Epoch 10/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
1 - acc: 0.7874 - val_loss: 0.0021 - val_acc: 0.5773

Train on 27546 samples, validate on 3061 samples
Epoch 1/10
27546/27546 [==============================] - 37s 1ms/step - loss: 0.003
6 - acc: 0.1350 - val_loss: 0.0034 - val_acc: 0.2078
Epoch 2/10
27546/27546 [==============================] - 34s 1ms/step - loss: 0.003
1 - acc: 0.3099 - val_loss: 0.0029 - val_acc: 0.3665
Epoch 3/10
27546/27546 [==============================] - 33s 1ms/step - loss: 0.002
5 - acc: 0.4573 - val_loss: 0.0026 - val_acc: 0.4721
```

```
Epoch 4/10
27546/27546 [==============================] - 33s 1ms/step - loss: 0.002
2 - acc: 0.5517 - val_loss: 0.0024 - val_acc: 0.5051
Epoch 5/10
27546/27546 [==============================] - 33s 1ms/step - loss: 0.001
9 - acc: 0.6150 - val_loss: 0.0023 - val_acc: 0.5201
Epoch 6/10
27546/27546 [==============================] - 33s 1ms/step - loss: 0.001
7 - acc: 0.6646 - val_loss: 0.0022 - val_acc: 0.5524
Epoch 7/10
27546/27546 [==============================] - 34s 1ms/step - loss: 0.001
5 - acc: 0.7020 - val_loss: 0.0022 - val_acc: 0.5449
Epoch 8/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
3 - acc: 0.7356 - val_loss: 0.0022 - val_acc: 0.5603
Epoch 9/10
27546/27546 [==============================] - 34s 1ms/step - loss: 0.001
2 - acc: 0.7667 - val_loss: 0.0022 - val_acc: 0.5590
Epoch 10/10
27546/27546 [==============================] - 33s 1ms/step - loss: 0.001
1 - acc: 0.7900 - val_loss: 0.0022 - val_acc: 0.5678

Train on 27546 samples, validate on 3061 samples
Epoch 1/10
27546/27546 [==============================] - 38s 1ms/step - loss: 0.003
6 - acc: 0.1298 - val_loss: 0.0034 - val_acc: 0.1931
Epoch 2/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.003
1 - acc: 0.3035 - val_loss: 0.0029 - val_acc: 0.3662
Epoch 3/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.002
6 - acc: 0.4511 - val_loss: 0.0025 - val_acc: 0.4750
Epoch 4/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.002
2 - acc: 0.5441 - val_loss: 0.0024 - val_acc: 0.5041
Epoch 5/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
9 - acc: 0.6093 - val_loss: 0.0023 - val_acc: 0.5361
Epoch 6/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
7 - acc: 0.6622 - val_loss: 0.0022 - val_acc: 0.5377
Epoch 7/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
5 - acc: 0.7003 - val_loss: 0.0022 - val_acc: 0.5550
Epoch 8/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
3 - acc: 0.7335 - val_loss: 0.0022 - val_acc: 0.5658
Epoch 9/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
2 - acc: 0.7620 - val_loss: 0.0022 - val_acc: 0.5691
Epoch 10/10
27546/27546 [==============================] - 35s 1ms/step - loss: 0.001
1 - acc: 0.7877 - val_loss: 0.0022 - val_acc: 0.5596
```

In [40]:
```
port matplotlib.pyplot as plt
port numpy as np

t.subplot(2, 3, 1)
t.plot(training_runs[0].history['acc'])
t.legend(['train'], loc='lower right')

t.subplot(2, 3, 2)
t.plot(training_runs[1].history['acc'])
t.legend(['train'], loc='lower right')

t.subplot(2, 3, 3)
t.plot(training_runs[2].history['acc'])
t.legend(['train'], loc='lower right')

t.subplot(2, 3, 4)
t.plot(training_runs[0].history['val_acc'])
t.legend(['test'], loc='lower right')

t.subplot(2, 3, 5)
t.plot(training_runs[1].history['val_acc'])
t.legend(['test'], loc='lower right')

t.subplot(2, 3, 6)
t.plot(training_runs[2].history['val_acc'])
t.legend(['test'], loc='lower right')

t.tight_layout()

t.show()

int("Average training accuracy: {}".format(np.mean([training_runs[0].history
                                                     training_runs[1].history[
int("Average testing accuracy: {}".format(np.mean([training_runs[0].history[
                                                     training_runs[1].history[
```




```
Average training accuracy: 0.7883661269576123
Average testing accuracy: 0.5682238921106063
```

By Using pre trained model run three times ,The best tesing accuracy performace for each run time is : 56.39% 55.90% 55.05% and average testing accuracy is 55.77%