Satellite Image Classification using ConvNet, Keras and TensorFlow

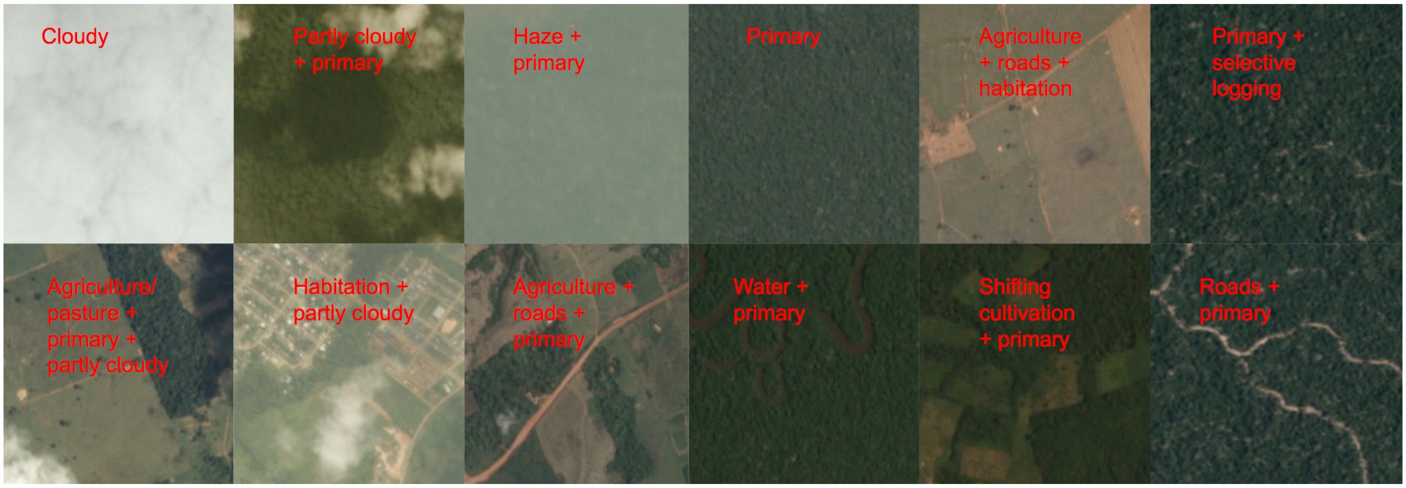
Introduction

Problem

> Every minute, the world loses an area of forest the size of 48 football fields. And deforestation in the Amazon Basin accounts for the largest share, contributing to reduced biodiversity, habitat loss, climate change, and other devastating effects. But better data about the location of deforestation and human encroachment on forests can help governments and local stakeholders respond more quickly and effectively.

Class Labels

Datasets comprise of 17 labels.
haze, primary, agriculture, clear, water, habitation, road, cultivation, slash_burn, cloudy, partly_cloudy, conventional_mine, bare_ground, artisinal_mine, blooming, selective_logging, blow_down


Cloudy
Partly cloudy + primary
Haze + primary
Primary
Agriculture + roads + habitation
Primary + selective logging
Agriculture/ pasture + primary + partly cloudy
Habitation + partly cloudy
Agriculture + roads + primary
Water + primary
Shifting cultivation + primary
Roads + primary

Neural Networks

```
In [ ]:
```

Convolutional Neural Networks (ConvNet/CNN)

```
In [ ]:
```

Deep Learning

```
In [ ]:
```

TensorFlow

```
In [ ]:
```

Keras

```
In [ ]:
```

```
In [ ]:
```

Enviornment Setup

(Verify these, probably will need to add remove some packages)

Install

```
conda create -n keras python=3.5 jupyter
activate keras
conda install theano
conda install mingw libpython
pip install tensorflow
pip install keras
pip install scikit-learn
pip install pillow
pip install h5py
pip install tensorflow-gpu
pip install imagenet_utils
activate keras
```

Open CV

```
conda install -c menpo opencv3
```

Verify

```
python -c "from keras import backend; print(backend._BACKEND)"
```

Config

```
python -c "import os; print(os.path.expanduser('~') + '.keras\keras.json')"
```

Verify GPU

```
python -c "import tensorflow as tf; sess = tf.Session(); hello = tf.constant('Hello, TensorFlow!');
print(sess.run('hello'))"
```

Run

```
activate keras
```

Data Preparation

Download and unzip following datasets from Kaggle. Note for this project we'll be using jpg instead of tif. As processing high resolution tif is computationally expensive. With jpg datasets we can achieve satisfactory results, 96% accuracy and top 15% on Kaggle Leader board.

test-jpg.tar.7z
train-jpg.tar.7z

URL: https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/data (https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/data)

In [4]:

```
import numpy as np
import pandas as pd
import h5py
import cv2
import scipy.io as sio
import os
import time # Timing
```

Serialize train and test datasets in hdf5.

Create onehot encoding for train labels

In [5]:

```python
test_output_file = 'test-dataset-128.h5'
train_output_file = "train-dataset-128.h5"

test_image_path = 'c:/data/amazon/test-jpg'
train_image_path = 'c:/data/amazon/train-jpg'

train_max_image_idx = 40478
test_max_image_idx = 40668

train_csv = 'c:/data/amazon/train.csv'

image_resize = (128,128) # Resize images

x = []

start_time_data_prep = time.time()

for i in range(0, test_max_image_idx + 1):
    img = test_image_path + "/test_" + str(i) + ".jpg"
    if i % 5000 == 0:
        print("reading image: {}".format(img))
    img = cv2.imread(img)
    img = cv2.resize(img,image_resize)
    # img = img.transpose((2,0,1))
    x.append(img)

print('Saving file: {}'.format(test_output_file))
x = np.array(x)
f = h5py.File(test_output_file)
f['x'] = x
f.close()

print('Time elapsed: {} seconds'.format(time.time()-start_time_data_prep))
```

```
reading image: c:/data/amazon/test-jpg/test_0.jpg
reading image: c:/data/amazon/test-jpg/test_5000.jpg
reading image: c:/data/amazon/test-jpg/test_10000.jpg
reading image: c:/data/amazon/test-jpg/test_15000.jpg
reading image: c:/data/amazon/test-jpg/test_20000.jpg
reading image: c:/data/amazon/test-jpg/test_25000.jpg
reading image: c:/data/amazon/test-jpg/test_30000.jpg
reading image: c:/data/amazon/test-jpg/test_35000.jpg
reading image: c:/data/amazon/test-jpg/test_40000.jpg
Saving file: test-dataset-128.h5
Time elapsed: 85.76890587806702 seconds
```

In [6]:

```
## Train dataset
train_output_file = "train-dataset-128.h5"
train_image_path = 'c:/data/amazon/train-jpg'
train_max_image_idx = 40478
train_csv = 'c:/data/amazon/train.csv'

image_resize = (128,128) # Resize images

start_time_data_prep = time.time()

df = pd.read_csv(train_csv)
print('Training dataset shape: {}'.format(df.shape))
df.head()

# Build list with unique labels
label_list = []
for tag_str in df.tags.values:
    labels = tag_str.split(' ')
    for label in labels:
        if label not in label_list:
            label_list.append(label)

print('Labels: {}'.format(label_list))

# Add onehot features for every label
for label in label_list:
    df[label] = df['tags'].apply(lambda x: 1 if label in x.split(' ') else 0)

# Display head
df.head()

y = np.array(df.ix[:,2:])
#print(y.shape)

x = []

for i in range(0, train_max_image_idx + 1):
    img = train_image_path + "/train_" + str(i) + ".jpg"
    if i % 5000 == 0:
        print("reading image: {}".format(img))
    img = cv2.imread(img)
    img = cv2.resize(img,image_resize)
    # img = img.transpose((2,0,1))
    x.append(img)

print('Saving file: {}'.format(train_output_file))
x = np.array(x)
f = h5py.File(train_output_file)
f['x'] = x
f['y'] = y
f.close()

print('Time elapsed: {} seconds'.format(time.time()-start_time_data_prep))
```

```
Training dataset shape: (40479, 2)
Labels: ['haze', 'primary', 'agriculture', 'clear', 'water', 'habitation', 'r
oad', 'cultivation', 'slash_burn', 'cloudy', 'partly_cloudy', 'conventional_m
ine', 'bare_ground', 'artisinal_mine', 'blooming', 'selective_logging', 'blow
_down']
reading image: c:/data/amazon/train-jpg/train_0.jpg
reading image: c:/data/amazon/train-jpg/train_5000.jpg
reading image: c:/data/amazon/train-jpg/train_10000.jpg
reading image: c:/data/amazon/train-jpg/train_15000.jpg
reading image: c:/data/amazon/train-jpg/train_20000.jpg
reading image: c:/data/amazon/train-jpg/train_25000.jpg
reading image: c:/data/amazon/train-jpg/train_30000.jpg
reading image: c:/data/amazon/train-jpg/train_35000.jpg
reading image: c:/data/amazon/train-jpg/train_40000.jpg
Saving file: train-dataset-128.h5
Time elapsed: 102.22684693336487 seconds
```

One hot encoding for the labels

```
In [8]: df.head()
```

Out[8]:

	image_name	tags	haze	primary	agriculture	clear	water	habitation	road	cul
0	train_0	haze primary	1	1	0	0	0	0	0	0
1	train_1	agriculture clear primary water	0	1	1	1	1	0	0	0
2	train_2	clear primary	0	1	0	1	0	0	0	0
3	train_3	clear primary	0	1	0	1	0	0	0	0
4	train_4	agriculture clear habitation primary road	0	1	1	1	0	1	1	0

Model

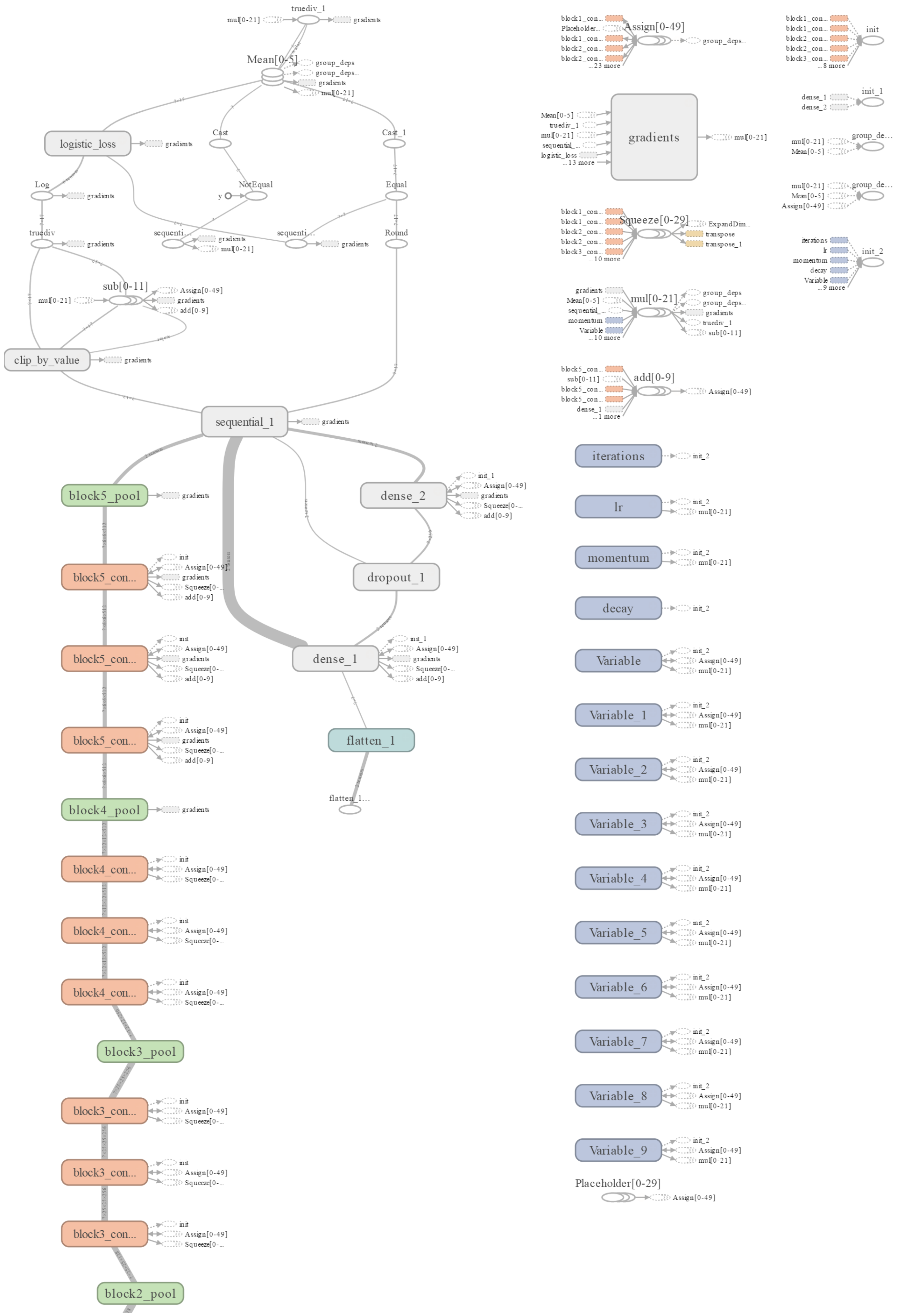
Description

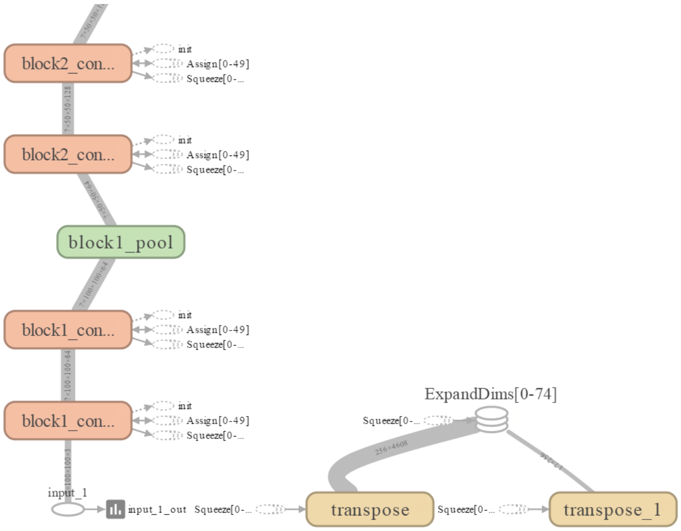
In [ ]:

```
```

TensorBoard Graph

VGG16 pre-trained model frozen upto conv4. Set the first 15 layers (up to the conv4) to non-trainable (weights will not be updated)


truediv_1
mul[0-21]
gradients
Mean[0-5]
group_deps
group_deps...
gradients
mul[0-21]
logistic_loss
gradients
Cast
Cast_1
Log
gradients
NotEqual
y
Equal
truediv
gradients
sequenti...
gradients
mul[0-21]
sequenti...
gradients
Round
sub[0-11]
mul[0-21]
Assign[0-49]
gradients
add[0-9]
clip_by_value
gradients
sequential_1
gradients
block5_pool
gradients
dense_2
init_1
Assign[0-49]
gradients
Squeeze[0-...
add[0-9]
block5_con...
init
Assign[0-49]
gradients
Squeeze[0-...
add[0-9]
block5_con...
block5_con...
dropout_1
dense_1
init_1
Assign[0-49]
gradients
Squeeze[0-...
add[0-9]
flatten_1
flatten_1...
block4_pool
block4_con...
block4_con...
block4_con...
block3_pool
block3_con...
block3_con...
block3_con...
block2_pool
block1_con...
Placeholder...
block1_con...
block2_con...
block2_con...
... 23 more
Assign[0-49]
group_deps...
init
block1_con...
block1_con...
block2_con...
block2_con...
block3_con...
... 8 more
init_1
dense_1
dense_2
Mean[0-5]
truediv_1
mul[0-21]
sequential_...
logistic_loss
... 13 more
gradients
mul[0-21]
group_de...
mul[0-21]
Mean[0-5]
group_de...
mul[0-21]
Mean[0-5]
Assign[0-49]
block1_con...
block1_con...
block2_con...
block2_con...
block3_con...
... 10 more
Squeeze[0-29]
ExpandDim...
transpose
transpose_1
init_2
iterations
lr
momentum
decay
Variable
... 9 more
gradients
Mean[0-5]
sequential_...
momentum
Variable
... 10 more
mul[0-21]
group_deps
group_deps...
gradients
truediv_1
sub[0-11]
block5_con...
sub[0-11]
block5_con...
block5_con...
dense_1
... 1 more
add[0-9]
Assign[0-49]
iterations
init_2
lr
init_2
mul[0-21]
momentum
init_2
mul[0-21]
decay
init_2
Variable
init_2
Assign[0-49]
mul[0-21]
Variable_1
Variable_2
Variable_3
Variable_4
Variable_5
Variable_6
Variable_7
Variable_8
Variable_9
Placeholder[0-29]
Assign[0-49]



In [ ]:

In [1]:
```
import numpy as np
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Flatten, Reshape
from keras.layers import Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import applications
from keras.optimizers import adam
import h5py
from sklearn.model_selection import train_test_split
import time
```

```
Using TensorFlow backend.
```

Function to load traing dataset for a given batch size and train/validation split

In [2]:
```
def load_train_dataset(dataset, random_state, batch_size=-1, test_size=0.2):
    '''
    batch_size=-1 returns all
    '''
    f = h5py.File(dataset)
    if batch_size == -1:
        x = f['x'].value
        y = f['y'].value
    else:
        x = f['x'][:batch_size,]
        y = f['y'][:batch_size,]

    f.close()
    x_train , x_test, y_train, y_test = train_test_split(x,y,test_size=test_si
ze, random_state=random_state)

    return x_train, x_test, y_train, y_test
```

Function to load test dataset

In [3]:

```
def load_test_dataset(dataset):
    f = h5py.File(dataset)
    x = f['x'].value   #[0:100,]
    f.close()

    return x
```

In [ ]:

Save Bottleneck Features from VGG16 Model

Global settings

In [4]:

```
epochs = 3
batch_size = 8
bottleneck_features_train_path = 'bottleneck_features_train_128_vgg.npy'
bottleneck_features_validation_path = 'bottleneck_features_validation_128_vgg.
npy'
# path to the model weights files.
top_model_weights_path = 'bottleneck_fc_model_128_vgg.h5'
# dimensions of our images.
input_shape = (128,128,3)
```

Save Bottleneck Features

In [5]:

```
def save_bottleneck_features():
    # build the VGG16 network
    # First time it will take longer, as it downloads the weights.
    model = applications.VGG16(include_top=False, weights='imagenet')
    model.summary()
    start_time = time.time()

    bottleneck_features_train = model.predict(
        x_train, batch_size = batch_size, verbose=1)

    np.save(open(bottleneck_features_train_path, 'wb'),
            bottleneck_features_train)

    bottleneck_features_validation = model.predict(
        x_test, batch_size = batch_size, verbose=1)
    np.save(open(bottleneck_features_validation_path, 'wb'),
            bottleneck_features_validation)
    print('save_bottleneck_features(): Time elapsed: {} seconds'.format(time.t
ime()-start_time))
```

Train Top Model

Train fully conected model.

In [19]:

```
def train_top_model():
    start_time = time.time()
    train_data = np.load(open(bottleneck_features_train_path, 'rb'))
    train_labels = y_train
    validation_data = np.load(open(bottleneck_features_validation_path, 'rb'))
    validation_labels = y_test

    print('train_data.shape[1:]: {}'.format(train_data.shape[1:]))
    model = Sequential()
    model.add(Flatten(input_shape=train_data.shape[1:]))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(17, activation='sigmoid'))

    model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy', metrics=['accuracy'])

    #### Update this if using weights from previous run ####
    # model.load_weights(top_model_weights_path)
    model.fit(train_data, train_labels,
              epochs=epochs,
              batch_size=batch_size,
              validation_data=(validation_data, validation_labels))
    model.save_weights(top_model_weights_path)
    print('train_top_model(): Time elapsed: {} seconds'.format(time.time()-sta
rt_time_data_prep))
```

Putting them together and train full model

In [12]:

```
def train_full_model():
    start_time = time.time()
    # build the VGG16 network
    base_model = applications.VGG16(weights='imagenet', include_top=False, inp
ut_shape=input_shape)
    print('Model loaded.')
    # base_model.summary()
    # build a classifier model to put on top of the convolutional model
    top_model = Sequential()
    top_model.add(Flatten(input_shape=(4,4,512)))
    top_model.add(Dense(256, activation='relu'))
    top_model.add(Dropout(0.5))
    top_model.add(Dense(17, activation='sigmoid'))

    # note that it is necessary to start with a fully-trained
    # classifier, including the top classifier,
    # in order to successfully do fine-tuning
    top_model.load_weights(top_model_weights_path)
    # top_model.summary()

    # add the model on top of the convolutional base
    model = Model(inputs= base_model.input, outputs= top_model(base_model.outp
ut))

    # set the first 15 layers (up to the last conv block)
    # to non-trainable (weights will not be updated)
    for layer in model.layers[:15]:
        # print(layer.name)
        layer.trainable = False

    model.summary()

    #### Update following if using weights from previous run ####
    #model.load_weights('weights-model-07.01-0.95972.hdf5')

    # compile the model with adam optimizer
    # and a very slow learning rate.
    model.compile(loss='binary_crossentropy',
                optimizer=adam(lr=1e-4),
                metrics=['accuracy'])

    x_train, x_test, y_train, y_test = load_train_dataset(random_state=random_
state, dataset=train_dataset, test_size=test_size)

    x_train = x_train.astype('float32')
    x_test  = x_test.astype('float32')

    x_train /= 255
    x_test /= 255

    tbCallBack = TensorBoard(log_dir='graph', histogram_freq=0, write_graph=Tr
ue, write_images=False, embeddings_freq=0)
    check = ModelCheckpoint("weights-model-07.{epoch:02d}-{val_acc:.5f}.hdf5",
 monitor='val_acc', verbose=1,
                        save_best_only=True, save_weights_only=True, mode='aut
o')
    model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs,callbacks=[check],validation_data=(x_test,y_test))


    ## Predict
    print('Generate Predictions...')
    x_test =  load_test_dataset(test_dataset)
    x_test = x_test.astype('float32')
    x_test /= 255.

    best_threshold = [0.2] * 17

    # print("best_threshold: {}".format(best_threshold))

    pred = model.predict(x_test, verbose=1, batch_size=8)
    # print(pred)
    print(pred.shape)

    classes = ['haze',
            'primary',
            'agriculture',
            'clear',
            'water',
            'habitation',
            'road',
            'cultivation',
            'slash_burn',
            'cloudy',
            'partly_cloudy',
            'conventional_mine',
            'bare_ground',
            'artisinal_mine',
            'blooming',
            'selective_logging',
            'blow_down']

    y_pred = []

    ##text=List of strings to be written to file
    with open(submission_file,'w') as file:
        file.write("image_name,tags")
        file.write('\n')

        for i in range(pred.shape[0]):
            y_pred = np.array([1 if pred[i, j] >= best_threshold[j] else 0 for j in range(pred.shape[1])])
            # print(y_pred)

            # extracting actual class name
            y_pred = [classes[i] for i in range(17) if y_pred[i] == 1]
            y_pred = " ".join([str(item) for item in y_pred])
            # print(y_pred)
            line = "test_{},{}".format(i, y_pred)
            file.write(line)
            file.write('\n')

    print('train_full_model(): Time elapsed: {} seconds'.format(time.time()-start_time))
```

Execution

```
In [8]: random_state = 101
        train_dataset = 'train-dataset-128.h5'
        test_size = 0.3 # for the train/test split
        submission_file = 'model-07-submission.csv'

        x_train, x_test, y_train, y_test = load_train_dataset(dataset=train_dataset,
                                                              random_state=random_stat
        e,
                                                              test_size=test_size)

        x_train = x_train.astype('float32')
        x_test  = x_test.astype('float32')

        x_train /= 255
        x_test /= 255
```

Save bottleneck features to npy arrays

```
In [9]: save_bottleneck_features()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
28335/28335 [==============================] - 1269s
12144/12144 [==============================] - 540s
save_bottleneck_features(): Time elapsed: 1816.2711477279663 seconds
```

Train Top Model with the Bottleneck Features

In [18]:

```
# Ideally you should run 10 epochs
# If you are re-running, make sure to load the
# weights from last run
epochs = 1
# Adjust this per your HW (from 8-32)
batch_size = 8

train_top_model()
```

```
train_data.shape[1:]: (4, 4, 512)
Train on 28335 samples, validate on 12144 samples
Epoch 1/1
28335/28335 [==============================] - 65s - loss: 0.2107 - acc: 0.92
12 - val_loss: 0.1816 - val_acc: 0.9319
train_top_model(): Time elapsed: 77.15817737579346 seconds
```

Train Full Model

In [21]:

```
# Ideally you should run 10 epochs
# If you are re-running, make sure to load the
# weights from last run
epochs = 1
# Adjust this per your HW (from 8-32)
batch_size = 8

random_state = 55
train_dataset = 'train-dataset-128.h5'
test_dataset = 'test-dataset-128.h5'
test_size = 0.3 # for the train/test split
submission_file = 'model-07-submission.csv'

train_full_model()
```

```
Model loaded.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (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         
_________________________________________________________________
sequential_5 (Sequential)    (None, 17)                2101777   
=================================================================
Total params: 16,816,465
Trainable params: 9,181,201
Non-trainable params: 7,635,264
_________________________________________________________________
Train on 28335 samples, validate on 12144 samples
Epoch 1/1
28328/28335 [============================>.] - ETA: 0s - loss: 0.2339 - acc:
 0.9060Epoch 00000: val_acc improved from -inf to 0.90660, saving model to we
ights-model-07.00-0.90660.hdf5
28335/28335 [==============================] - 2296s - loss: 0.2339 - acc: 0.
9060 - val_loss: 0.2071 - val_acc: 0.9066
Predictions...
40669/40669 [==============================] - 1836s
```