Multi-labels Classification for Satellite Images

Project Document

Team-1

Preface

We had been discussing on what should be the right project for this data mining class when our team members have diverse experiences. Some team members proposed a research problem while another member proposed a practical problem. It turns out that we decided to choose a highly impactful problem that may be beneficial to our community. We chose to work on a problem of labelling an Amazon rainforest satellite image since we realize that the technology can be a human greatest tool to solve deforest problem in Amazon basin. We do hope that our contribution in this project will make this world a better place to live.

Acknowledgements

We would like to thank Kaggle and scientist teams for collecting and labelling a high quality satellite images. Without their efforts, this project would not be possible. We appreciate Professor Wang Ming-hwa for his feedback on our project. We would like to thank those who allow us to reproduce images, figures and data from their publications. We indicate their contributions in the figure captions throughout the text.

Table of content

List of Tables

List of Figures

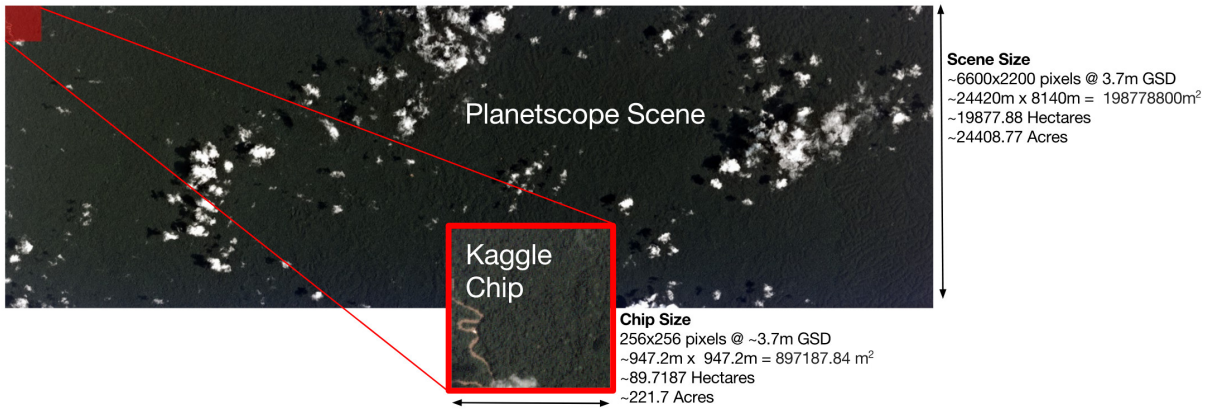
1. Abstract

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. In this paper, we propose a deep learning model to predict a point of interests from the satellite images. Our model can be summarized as 2 stages learning: unsupervised and supervised learning. The first stage, we train an unsupervised model, an autoencoder, on a small patch extracted from the original image. The second stage, we train a supervised model, a CNN network, to learn to predict a set of labels from the feature vectors constructed from the previous step. We evaluate our model using F1-score which is a measure of trade off between precision and recall. We hope that our approach will yield a significant improvement over our baseline methods and state-of-arts methods on fine-grained image classification tasks.

2. Introduction

• Objective

We want to find an efficient solution for labelling a collection of satellite images on Amazon forests in order to detect human activities. The automatic solution is desirable due to its ability to scale for a very large collection. Also, the scientists team needs to update their satellite images every fix intervals in order to analyze the deforestation in Amazon forests. Thus, the scalable and highly accurate tools for labelling these images are very crucial for their team.



• What is the problem

This problem can be viewed as a multi-label classification task on a satellite images. The goal is to predict multiple conceptual labels of a satellite image of Amazon river basin. There are two main challenges of this project.

- The features in these satellite images are not easy to learn through the CNN model, because of the low resolution and noise due to bad weather, ambiguity, and human-errors on labelling an image。That cause this task is more difficult than a vainilla image classification.
- Furthermore, a satellite image poses another challenge in terms of identify the local point of interests. For example, the cloudy region, it will be difficult to detect terrain. Some local point of interests are very similar such as a river and road. Thus, by pixelated information alone may not be a strong indication of a particular local features. Each satellite images contains multiple local features and these set of features could form a hierarchy relationship. For example, a cloudy label is classified further into the type of cloud such as partial cloudy or dense cloud. The hierarchical labelling may pose another challenge when we develop a classification model.

figure 2.1 Satellite image example with labels: agriculture, clear, primary, road, water,

• Why this is a project related the this class

Deep Learning has a connection with Data Mining in terms of extracting an underlying representation and knowledge from the high dimensional data. In data mining, we care about understanding the key insight of the given information which is related to our project in which we are interested in discover a point of interests in the satellite images through deep learning approach.

•Statement of the problem

We are given a collection of satellite images as training samples. Each image is 256x256 color pixels derived from the original Planet's full-frame analytic scene taken by the sun-synchronous orbit (SSO) and International Space Station (ISS) orbit. Each training images are labelled by crowdsourcing to determine the conceptual labels. There are 17 conceptual labels such as cloud cover labels, primary rain forest, water, habitation, agriculture, road, cultivation, and ground. The labels distribution followed a Power's laws where the primary rain forest contributes 80% of the training sample where there many rare-labels that have very few presence in the dataset. The goal is train the classification model to recognize these terrain features from the labels provided collected by the human experts. Furthermore, due to it is a multi-label

classification problem, each image is associated with more than one label, we want to predict all related labels for the given test satellite image.

•Area or scope of investigation

This project will investigate the state-of-the-arts in representation learning using a deep model such as convolutional neural networks and fine-grained detection on images. This project is required an understanding of satellite images and its properties. Our team needs to study the state-of-the-arts approach in multi-labels classification.

3. Theoretical bases and literature review

• theoretical background of the problem

Multi-label classification on satellite images is task of finding multiple key features from a noisy image. Since each image may contain multiple point of interests, fine-grained image classification approach is appropriate.

• related research to solve the problem

1. Multi-label classification

The simplest approach in a multi-label classification is to assume that all labels are uncorrelated. This assumption turns a multi-label classification into a K-way binary classification. The deep convolutional neural networks is commonly used for learning a discriminant features while the prediction layer has K-sigmoid outputs, each predict the present of labels.

Another approach is to design an appropriate loss function. For some tasks, it is more appropriate to use hamming distance as a loss function in order to account for the mis-labelled information. However, using cross-entropy seems to be effective in terms of account for the confidence in predicting a particular label. The rank-loss is another approach that is commonly used in this task since most samples have at most 3-5 labels, rank-loss could be utilized.

2. Fine-grained image classification

Fine-grained image classification has become an important task in computer vision on natural image because each image may contain multiple visual concepts. For example, an image of a beach may contain multiple concepts such as sunset, sand, and water. Many studies have proposed deep learning model to detect such a fine-grained discriminated feature.

• advantage/disadvantage of those research

Fine-grained feature detection is domain specific to natural image. We need to modify our approach in order to utilize their idea on detecting a discriminant feature.

Multi-label classification is a generic solution and has zero knowledge on domain specific. We need to incorporate domain-specific information such as the property of Amazon rainforests or how reliable of satellite images, how much noise are introduced in the training samples.

• your solution to solve this problem

1. Baseline approach

We train the CNN nets and use an appropriate loss function. In this case, we will use log-loss on predicted labels probability generated by the prediction layer.

2. Second approach

Since each image is collection of point of interests, we need to train the model to recognize these local features. We can create a patch with a size of 16x16 pixels as training samples. We train a CNN-based autoencoder to learn a representation of these samples. Then, we pool all local-feature representations to create a feature vector for the original satellite images. Then, we train a multi-label classifier on this feature vector. This process can be trained as an end-to-end system.

• where your solution different from others

Although our approach is similar to [2] in terms of learning a local feature, we use an autoencoder to learn a feature whereas [2] uses features learned by the CNN network. The task in [2] concerns about finding a local feature in which accuracy is very important, in our work, a learned local feature can be noisy as long as the predicted labels are accurate.

• why your solution is better

With our best knowledges, there is no existing work that utilizes an autoencoder to learn a local feature from a satellite image. There are many related works in terms of learning a fine-grained features from the image. We believe that using an autoencoder providing a strong regularizer through a constraint that force the model to minimize an reconstruction error.

4. Hypothesis (or goals)

We assume that the point of interests share similar local features. For example, forest and trees are always a green patch, river and road are always surrounded by trees which can be seen as a brown stripe enclosed by green patches. It is possible that the common point of interests will represent a label.

• Goals

We want to find an efficient solution for labelling a collection of satellite images on Amazon forests in order to detect human activities.

We will use F2 score to evaluate each model. The F score, commonly used in information retrieval, measures accuracy using the precision p and recall r. Precision is the ratio of true positives (tp) to all predicted positives (tp + fp). Recall is the ratio of true positives to all actual positives (tp + fn).

The F2 score is given as follows. Note that the F2 score weights recall higher than precision. The mean F2 score is formed by averaging the individual F2 scores for each row in the test set.

$$(1+\beta^2)\frac{pr}{\beta^2 p + r} \quad \text{where} \quad p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}, \quad \beta = 2.$$

5. Methodology

• how to generate/collect input data

Kaggle provide a modified version of Planet's full-frame analytic scene products using 4 band satellites in sun-synchronous orbit (SSO) and International Space Station (ISS) orbit. The satellites images are converted to a 256x256 jpeg and Tiff formats with the number of training samples of 40,479 and testing samples of 40,669. Each image is labeled by Crowd Flower platform. There are 17 labels.

Sample Labels

- Language: Python

- Libraries: Keras, TensorFlow, Scipy, Numpy, Pandas
- Backend: TensorFlow
- Models: Three

1. Baseline approach

We train the CNN nets and use an appropriate loss function. In this case, we will use log-loss on predicted labels probability generated by the prediction layer.

2. Second approach

Since each image is collection of point of interests, we need to train the model to recognize these local features. We can create a patch with a size of 16x16 pixels as training samples. We train a CNN-based autoencoder to learn a representation of these samples. Then, we pool all local-feature representations to create a feature vector for the original satellite images. Then, we train a multi-label classifier on this feature vector. This process can be trained as an end-to-end system.

3. Third approach

With the same model architecture as approach 1, we train an off-the-shelf L-separated classifier, one for a group of labels.

The output is a set of labels indicating a unique features of the rainforest from the satellite images. These outputs are generated by learning a function that mapping a test image to a set of labels.

6. Implementation

6.1 The First Try

We tried to use transfer learning to classify our satellite images. Transfer learning is a technique that shortcuts a lot of this work by taking a fully-trained model for a set of categories like ImageNet, and retrains from the existing weights for new classes. And we try to retrain Inception v3 model's final layer to classify our satellite images. Furthermore, we changed this Inception model to multi-labels classifier.

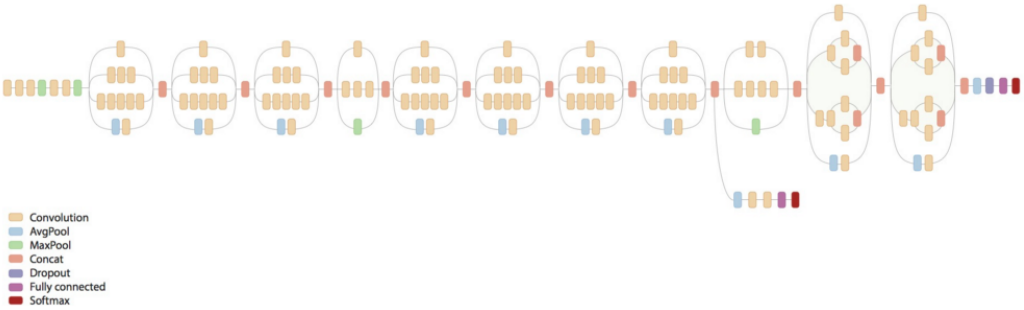


figure 6.1 inception model

As the figure 6.1, Inception model is a deep neural network with million parameters.
We expected it will effective for our project and we can get pretty good result. But unfortunately, it totally doesn't work.
The reason why this method doesn't work is because our satellite is pretty much particular than the normal image sets. So the features of satellite can not be extracted just through transfer learning. We have to train the model from the scratch.

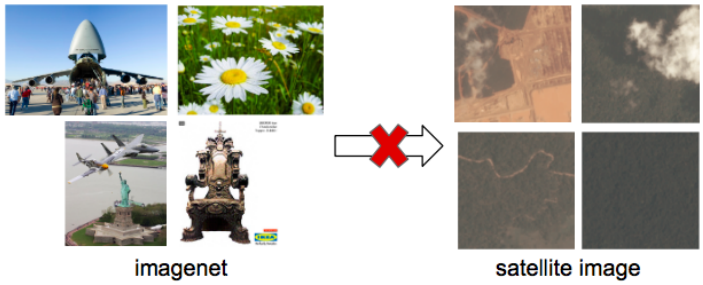
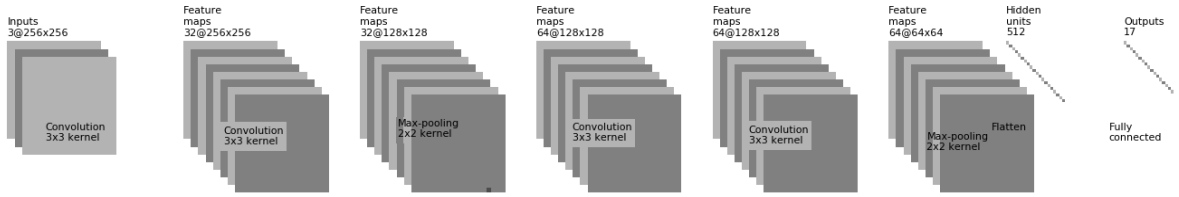


figure 6.2 imagenet data set and satellite data set

6.2 Initial model

We started with a fundamental CNN model. In this model, we used Sigmoid to implement multi-label training. The sequence of this model is as below:
Cov 3*3 kernel layer → Cov 3*3 kernel layer → max pooling 2*2 → Cov 3*3 kernel layer → Cov 3*3 kernel layer → max pooling 2*2 → Fully connected layer
The input is 256*256*3 satellite image and the output is 17 labels.

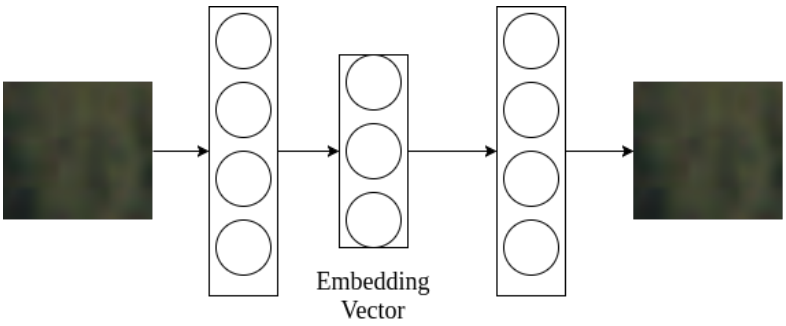


6.3 Model 2

In model 2, we used patch embedding. For each satellite image, we created a patch of 16*16 pixels and convert each patch to an embedding patch using the autoencoder from Step 2. For each embedding patch, we count the number of point of interest (POI) concepts in this image by the following procedure:

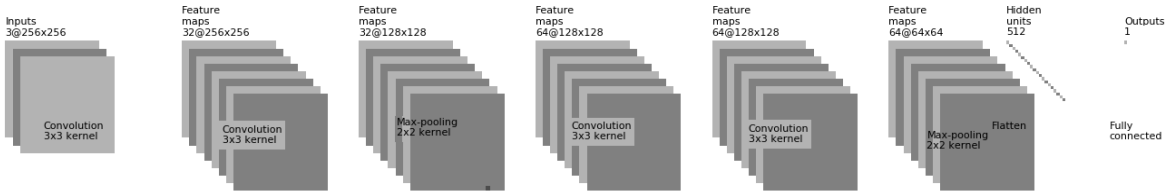
- For each patch, we compute Euclidean distance to each POI concept.
- Let's say the closest POI concept is "River", then we will increment the count of river concept by 1.

The final histogram will be a satellite image representation.



6.4 Model 3

We used the same model architecture as model1. But modified the the input from mutli-label to single label and used softmax instead of sigmoid. Unlike the model 1 that deals with multi-label, we trained an off-the-shelf L-separated classifier, one for a group. The output is only one label. For example, we divide all images into two groups: cloud and non-cloud and then acquire the label regarding to cloud for all images.



7. Data Analysis and Discussion

We will use F score to evaluate each model. The F score, commonly used in information retrieval, measure accuracy using the precision p and recall r. Precision is the ratio of true positives(tp) to all predicted positives (tp+fp). Recall is the ratio of true positives to all actual positives (tp+fn).

The F score is given by:

$$(1+\beta^2)\frac{pr}{\beta^2 p + r} \quad \text{where} \quad p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}, \quad \beta = 2.$$

• Evaluation Matrix

Models	Precision	Recall	F2-Score

Model 1	0.8400	0.7685	0.7741
Model 2	0.2480	**0.8031**	0.5634
Model 3	**0.8591**	0.8029	**0.8234**

• compare output against hypothesis

• discussion

8. Conclusions and recommendations

• Summary and conclusions

Label satellite image chips with atmospheric conditions and various classes of land cover/land use by deep learning method.

We implemented three different models to predict multi-labels for images and did the result analysis for these three models

• Future works

Data augmentation before training

We can try even deeper models

AlexNet: 5 Cov layers

VGG

Inception, ResNet

Build our own models

9. Bibliography

[1] Silla Jr, Carlos N., and Alex A. Freitas. "A survey of hierarchical classification across different application domains." Data Mining and Knowledge Discovery 22.1-2 (2011): 31-72.

[2] Xiao, Tianjun, et al. "The application of two-level attention models in deep convolutional neural network for fine-grained image classification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[3] Krause, Jonathan, et al. "Fine-grained recognition without part annotations." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[4] Lin, Di, et al. "Deep lac: Deep localization, alignment and classification for fine-grained recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

10. Appendices

• program source code with documentation

```
# In[1]:

get_ipython().magic('matplotlib inline')

import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm


# In[5]:

label_db = {}
with open('data/train_v2.csv') as input_labels:
    for line in input_labels:
        tokens = line.strip().split(',')
        label_db[tokens[0]] = tokens[1]


# In[3]:

from scipy.misc import imread

import os
```

```
data_path = 'dataset/train-jpg'

train_img = []
input_labels = []
cnt = 0
all_files = os.listdir(data_path)

max_cnt = len(all_files)

for fn in tqdm(all_files, total=len(all_files)):
    im = imread(os.path.join(data_path, fn))
    rgb_only = im[:,:,:3].astype(np.float32)
    #gray = np.sum(rgb_only * np.array([.2989, .5870, .1140]), axis=2)
    #train_img.append(gray)
    train_img.append(rgb_only)

    filename, file_extension = os.path.splitext(fn)
    input_labels.append(label_db[filename])
    cnt += 1
    if cnt >= max_cnt:
        break


# In[4]:

train_img = np.array(train_img)
n_samples = train_img.shape[0]
train_img.shape


# In[5]:

from sklearn.feature_extraction.text import CountVectorizer
raw_labels = []
cnt = 0

for line in input_labels:
    tokens = line.strip()
    raw_labels.append(tokens)
    cnt += 1
    if cnt >= max_cnt:
        break

label_tf = CountVectorizer(binary=True, max_features=20)
```

```
labels = label_tf.fit_transform(raw_labels)


# In[6]:

indices = np.random.permutation(n_samples)

# split the train and test
n_test = int(n_samples * 0.1)
test = train_img[indices[:n_test]]
train = train_img[indices[n_test:]]
test_labels = labels[indices[:n_test]]
train_labels = labels[indices[n_test:]]

# Compute KL divergence
test_disb = np.sum(test_labels, axis=0)
train_disb = np.sum(train_labels, axis=0)
test_disb = test_disb / float(np.sum(test_disb))
train_disb = train_disb / float(np.sum(train_disb))

kl = np.sum(np.multiply(train_disb, (np.log(train_disb) - np.log(test_disb))))
print(kl)

kl = np.sum(np.multiply(test_disb, (np.log(test_disb) - np.log(train_disb))))
print(kl)


# In[7]:

import os
import tensorflow as tf

os.environ["CUDA_VISIBLE_DEVICES"]="3"

def get_session(gpu_fraction=0.1):
    '''Assume that you have 6GB of GPU memory and want to allocate ~2GB'''
    num_threads = os.environ.get('OMP_NUM_THREADS')
    gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=gpu_fraction)

    if num_threads:
        return tf.Session(config=tf.ConfigProto(
            gpu_options=gpu_options, intra_op_parallelism_threads=num_threads))
    else:
        return tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
```

```
import keras.backend as K
K.set_session(get_session(1.0))


# In[8]:

import os
import tensorflow as tf
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

batch_size = 100
num_classes = 17
epochs = 5
data_augmentation = False

x_train = train
x_test = test
y_train = train_labels.toarray()
y_test = test_labels.toarray()

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')


# In[9]:

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('sigmoid'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

def accuracy_with_threshold(y_true, y_pred):
    threshold = 0.5
    y_pred = K.cast(K.greater(y_pred, threshold), K.floatx())
    return K.mean(K.equal(y_true, y_pred))

def hamming_dist(y_true, y_pred):
    threshold = 0.5
    y_pred = K.cast(K.greater(y_pred, threshold), K.floatx())
    return K.mean(K.sum(K.abs(y_true - y_pred), axis=1))

# def precision_scores(y_true, y_pred):
#     threshold = 0.5
#     y_pred = K.cast(K.greater(y_pred, threshold), K.floatx())
#     tf = y_pred & y_true

# Let's train the model using RMSprop
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=[accuracy_with_threshold, hamming_dist])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255.0
x_test /= 255.0

if not data_augmentation:
    print('Not using data augmentation.')
```

```
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images

    # Compute quantities required for feature-wise normalization
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)

    # Fit the model on the batches generated by datagen.flow().
    model.fit_generator(datagen.flow(x_train, y_train,
                                     batch_size=batch_size),
                        steps_per_epoch=x_train.shape[0] // batch_size,
                        epochs=epochs,
                        validation_data=(x_test, y_test))


# In[10]:

pred = model.predict(x_test, batch_size=batch_size, verbose=0)


# In[11]:

test_img_files = []
for idx in indices[:n_test]:
    test_img_files.append(all_files[idx])
```

```
# In[12]:

with open('label_predict.csv', 'w') as outfn:
    for idx in range(n_test):
        s = test_img_files[idx]
        for j in range(num_classes):
            s += ', {:.3f}'.format(pred[idx,j])
        outfn.write(s)
        outfn.write('\n')


# In[53]:

y_pred = (pred >= 0.5).astype(np.int32)
num_relevants = np.sum(y_test & y_pred, axis=1)
num_retrieves = np.sum(y_pred, axis=1)
total_relevants = np.sum(y_test, axis=1)

# precision
precision = np.divide(num_relevants.astype(np.float), num_retrieves.astype(np.float))

# recall
recall = np.divide(num_relevants.astype(np.float), total_relevants.astype(np.float))

beta = 2.
beta_sqr = beta * beta
f1 = (1 + beta_sqr) * np.divide((precision * recall),(beta_sqr * precision + recall))

print('precision={}'.format(np.nanmean(precision)))
print('recall={}'.format(np.nanmean(recall)))
print('f1={}'.format(np.nanmean(f1)))
```