HARISH RAJPUT · 7M AGO · 3 VIEWS · PRIVATE

0 Edit

notebook02a278a6f6

Python · Planet: Understanding the Amazon from Space

Notebook Input Output Logs Comments (0) Settings

Competition Notebook
Planet: Understanding the Amazon from ...

Run
47.6s

Version 2 of 2

Add Tags

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input direct
ory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you cr
eate a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/planet-understanding-the-amazon-from-space/Kaggle-planet-train-tif.torrent
/kaggle/input/planet-understanding-the-amazon-from-space/Kaggle-planet-test-tif.torrent
/kaggle/input/planet-understanding-the-amazon-from-space/test_v2_file_mapping.csv/test_v2_file_mapping.csv
/kaggle/input/planet-understanding-the-amazon-from-space/train_v2.csv/train_v2.csv
/kaggle/input/planet-understanding-the-amazon-from-space/sample_submission_v2.csv/sample_submission_v2.csv
```

In [2]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [3]:
```
from fastai.vision import *
```

In [ ]:
```
```

In [4]:
```
df = pd.read_csv(r'/kaggle/input/planet-understanding-the-amazon-from-space/train_v2.csv/train_v2.csv')
df.head()
```

Out[4]:

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

In [5]:
```
df['tags'].values
```

Out[5]:
```
array(['haze primary', 'agriculture clear primary water', 'clear primary',
       ..., 'agriculture clear primary', 'agriculture clear primary road',
       'agriculture cultivation partly_cloudy primary'], dtype=object)
```

In [6]:
```
df=pd.read_csv(r'/kaggle/input/planet-understanding-the-amazon-from-space/test_v2_file_mapping.csv/test_v2_file_mapping.csv')
```

In [7]:
```
df.tail()
```

Out[7]:

	old	new
20517	file_17774.tif	file_14520.tif
20518	file_3538.tif	file_6633.tif
20519	file_1923.tif	file_4685.tif
20520	file_14047.tif	file_14571.tif
20521	file_8220.tif	file_7483.tif

In [8]:
```
def tag_mapping(df):
    labels=set()
    for i in range(len(df)):
        tags=data['tags'][i].split(' ')
        labels.update(tags)
    labels=list(labels)
    labels.sort()
    labels_dict={labels[i]:i for i in range(len(labels))}
    inv_labels={i:labels[i] for i in range(len(labels))}
    return labels_dict,inv_labels
```

In [9]:
```
label_map,invmap=tag_mapping(df)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_19/577388155.py in <module>
----> 1 label_map,invmap=tag_mapping(df)

/tmp/ipykernel_19/3839504574.py in tag_mapping(df)
      2     labels=set()
      3     for i in range(len(df)):
----> 4         tags=data['tags'][i].split(' ')
      5         labels.update(tags)
      6     labels=list(labels)

NameError: name 'data' is not defined
```

In [ ]:
```
def file_mapping(data):
    mapping={}
    for i in range(len(data)):
        name,tags=train_df['image_name'][i],train_df['tags'][i]
        mapping[name]=tags.split(' ')
    return mapping
```

In [ ]:
```
def one_hot_encode(tags, mapping):
    encoding = np.zeros(len(mapping), dtype='uint8')
    for tag in tags:
        encoding[mapping[tag]] = 1
    return encoding
```

In [ ]:
```
def load_dataset(path,file_mapping,tag_mapping):
```

```
    photos,targets=list(),list()
    for filename in os.listdir(path):
        photo=load_img(path+filename,target_size=(75,75))
        photo=img_to_array(photo,dtype='uint8')
        tags=file_mapping[filename[:-4]]
        target=one_hot_encode(tags,tag_mapping)
        photos.append(photo)
        targets.append(target)
    X=np.asarray(photos,dtype='uint8')
    y=np.asarray(targets,dtype='uint8')
    return X,y
```

In [ ]:

```
tags_mapping,_=tag_mapping(df)
files_mapping=file_mapping(df)
path='/kaggle/input/planet-understanding-the-amazon-from-space/Kaggle-planet-train-tif.torren'
X,y=load_dataset(path,files_mapping,tags_mapping)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import fbeta_score
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import SGD
from keras.optimizers import RMSprop
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [ ]:

```
def fbeta(y_true, y_pred, beta=2):
    y_pred = backend.clip(y_pred, 0, 1)

    tp = backend.sum(backend.round(backend.clip(y_true * y_pred, 0, 1)), axis=1)
    fp = backend.sum(backend.round(backend.clip(y_pred - y_true, 0, 1)), axis=1)
    fn = backend.sum(backend.round(backend.clip(y_true - y_pred, 0, 1)), axis=1)
    p = tp / (tp + fp + backend.epsilon())
    r = tp / (tp + fn + backend.epsilon())
    bb = beta ** 2
    fbeta_score = backend.mean((1 + bb) * (p * r) / (bb * p + r + backend.epsilon()))
    return fbeta_score
```

In [ ]:

```
from keras.applications import InceptionV3
model=InceptionV3(input_shape=(75,75,3),include_top=False)
for layer in model.layers:
    layers.trainable=False
last_layer=model.get_layer('mixed7')
last_output=last_layer.output

x=layers.Flatten()(last_output)
x=layers.Dense(1024,activation='relu')(x)
x=layers.Dense(512,activation='relu')(x)
x=layers.Dropout(0.2)(x)
x=layers.Dense(17,activation='sigmoid')(x)
model=models.Model(model.inputs,x)
model.compile(optimizer=RMSprop(lr=0.0001),loss='binary_crossentropy',metrics=[fbeta])
```

In [ ]:

```
train_datagen=ImageDataGenerator(rescale=1.0/255.0,horizontal_flip=True, vertical_flip=True, rotation_range=9
0)
test_datagen=ImageDataGenerator(rescale=1.0/255.0)
```

In [ ]:

```
history = model.fit(train_gen,steps_per_epoch=506,validation_data=test_gen, validation_steps=127, epochs=250,
verbose=0)
```

In [ ]:

```
loss, fbeta =model.evaluate_generator(test_gen, steps=8, verbose=0)
print('> loss=%.3f, fbeta=%.3f' % (loss, fbeta))
```

In [ ]:

```
history.history['fbeta']
```

In [ ]:
```
test_path_1='/kaggle/input/planets-dataset/planet/planet/test-jpg/'
test_path_2='/kaggle/input/planets-dataset/test-jpg-additional/test-jpg-additional/'
```

In [ ]:
```
submission_df=pd.read_csv('/kaggle/input/planet-understanding-the-amazon-from-space/sample_submission_v2.csv/sample_submission_v2.csv')
```

In [ ]:
```
photo_test=[]
for filename in submission_df['image_name']:
    if filename[:1]=='t':
        img=load_img(test_path_1+filename+'.jpg',target_size=(75,75))
    elif filename[:1]=='f':
        img=load_img(test_path_2+filename+'.jpg',target_size=(75,75))
    ph=img_to_array(img,dtype='uint8')
    photo_test.append(ph)
```

In [ ]:
```
test_x=np.asarray(photo_test,dtype='uint8')
```

In [ ]:
```
image_gen_test=ImageDataGenerator(rescale=1/255.0)
test_data_gen=image_gen_test.flow(test_x,shuffle=False,batch_size=64)
```

In [ ]:
```
result=model.predict(test_data_gen)
```

In [ ]:
```
new_df=pd.DataFrame(result,columns=tags_mapping.keys())
```

In [ ]:
```
tags=new_df.columns
pred_tags=new_df.apply(lambda x: ' '.join(tags[x>0.5]),axis=1)
```

In [ ]:
```
pred_tag=pd.DataFrame(pred_tags,columns=['tags'])
```

In [ ]:
```
submission_df['tags']=pred_tag['tags']
```

In [ ]:
```
submission_df.to_csv('attempt_4.csv',index=False)
```

In [ ]:
```
submission_df.head()
```

Continue exploring

Data
1 input and 0 output →

Logs
47.6 second run - failure →

Comments
0 comments →

Import necessary libraries and load the data

In [ ]:
```
# Importing the necessary libraries
import os
import pandas as pd
import numpy as np
import seaborn as sns
import gc
from matplotlib.image import imread
import matplotlib.pyplot as plt
import gc
import cv2
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras import optimizers
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

In [ ]:
```
#Loading the csv and image datasets
train_path = '/content/planet/planet/train_classes.csv'
test_path = '/content/planet/planet/sample_submission.csv'
train_images = '/content/planet/planet/train-jpg/'
test_images = '/content/planet/planet/test-jpg/'
test_additional = '/content/test-jpg-additional/test-jpg-additional/'
```

In [ ]:
```
# Loading the train and test datasets
train_df = pd.read_csv(train_path)
print(train_df.shape)
train_df.head()
```

```
(40479, 2)
```

Out[9]:

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

In [ ]:
```
test_df = pd.read_csv(test_path)
print(test_df.shape)
test_df.head()
```

```
(61191, 2)
```

Out[10]:

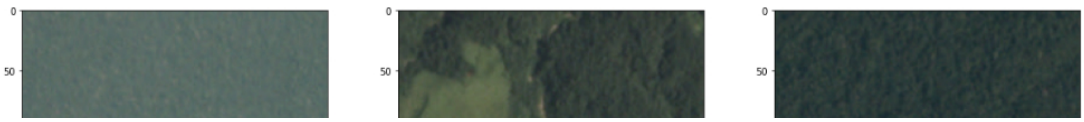
	image_name	tags
0	test_0	primary clear agriculture road water
1	test_1	primary clear agriculture road water
2	test_2	primary clear agriculture road water
3	test_3	primary clear agriculture road water
4	test_4	primary clear agriculture road water

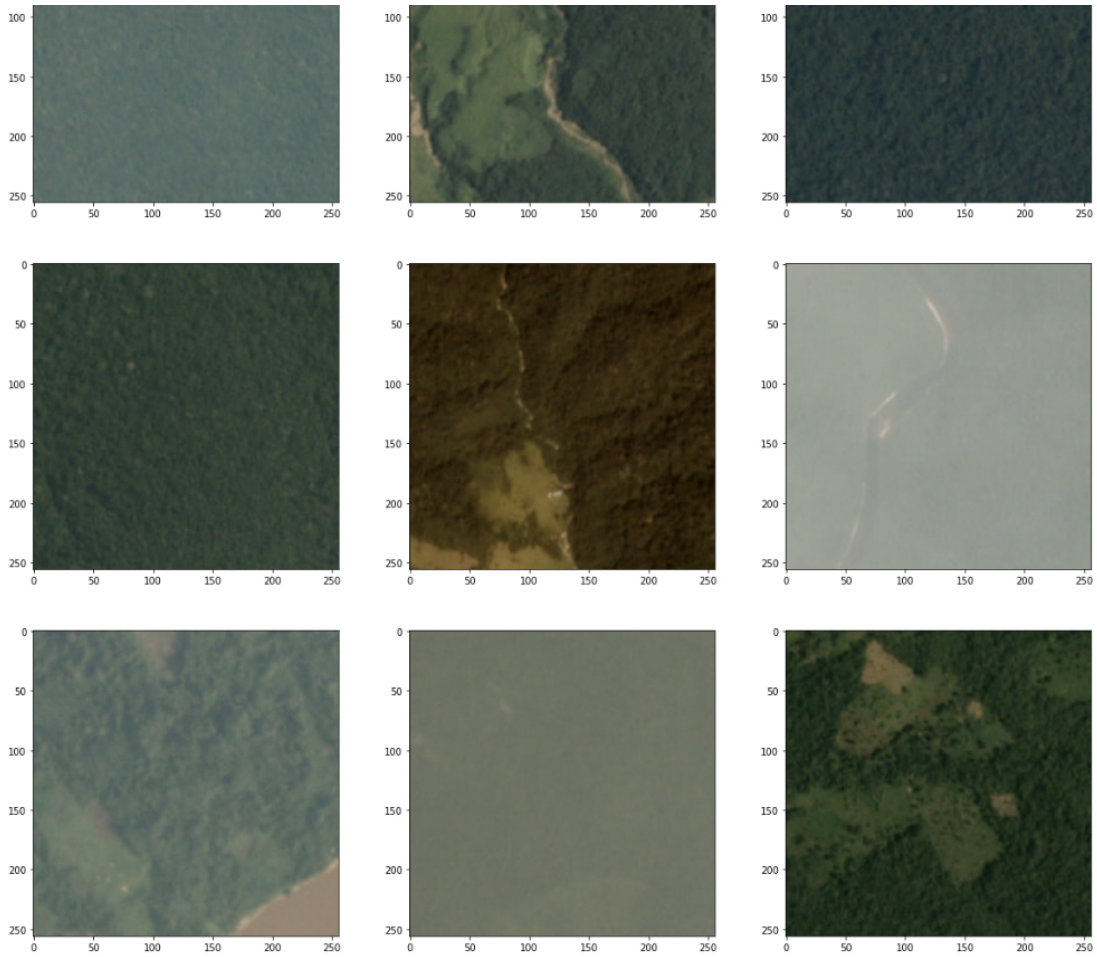
In [ ]:
```
gc.collect()
```

Out[11]:
```
0
```

Exploratory Data Analysis

In [ ]:
```
# Let's view some images
plt.figure(figsize=(20,20))
# define location of dataset
folder = train_images
# plot first few images
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    # define filename
    filename = folder + 'train_' + str(i) + '.jpg'
    # load image pixels
    image = imread(filename)
    # plot raw pixel data
    plt.imshow(image)
# show the figure
plt.show()
```





In [ ]:
```
#get number of unique classes in the train dataset
train_df['tags'].nunique()
```

Out[14]: 449

In [ ]:
```
gc.collect() #Frequently used to avoid session crashing due to memory exhaustion
```

Out[15]: 28331

In [ ]:
```
# Tags present in the dataset
tags = train_df['tags'].apply(lambda x: x.split(' '))
tags = [item for sublist in tags for item in sublist]
tag_counts = pd.Series(tags).value_counts()
```

Preprocessing the data

In [ ]:
```
labels = set()
def splitting_tags(tags):
    '''
    Takes in tags column, splits the tags and store as a set
    '''
    [labels.add(tag) for tag in tags.split()]

# Create a copy of `train_df`
train_df1 = train_df.copy()
train_df1['tags'].apply(splitting_tags)
labels = list(labels)
print(labels)
```

```
['agriculture', 'conventional_mine', 'habitation', 'clear', 'bare_ground', 'blow_down', 'slash_burn', 'haze', 'selective_loggin
g', 'partly_cloudy', 'primary', 'water', 'road', 'artisinal_mine', 'cloudy', 'cultivation', 'blooming']
```

In [ ]:
```
# Onehot encoding is performed on the labels in train classes
for tag in labels:
    train_df1[tag] = train_df1['tags'].apply(lambda x: 1 if tag in x.split() else 0)

# adding .jpg extension to the column image_name so as to have same name format as the image files
train_df1['image_name'] = train_df1['image_name'].apply(lambda x: '{}.jpg'.format(x))
train_df1.head()
```

Out[21]:

	image_name	tags	agriculture	conventional_mine	habitation	clear	bare_ground	blow_down	slash_burn	haze	selective_logging	partly_cloudy	prim
0	train_0.jpg	haze primary	0	0	0	0	0	0	0	1	0	0	
1	train_1.jpg	agriculture clear primary water	1	0	0	1	0	0	0	0	0	0	
2	train_2.jpg	clear primary	0	0	0	1	0	0	0	0	0	0	

3	train_3.jpg	clear primary	0	0	0	1	0	0	0	0	0	0
4	train_4.jpg	agriculture clear habitation primary road	1	0	1	1	0	0	0	0	0	0

In [ ]:
```
# Define the columns
columns = list(train_df1.columns[2:])
columns
```

Out[22]:
```
['agriculture',
 'conventional_mine',
 'habitation',
 'clear',
 'bare_ground',
 'blow_down',
 'slash_burn',
 'haze',
 'selective_logging',
 'partly_cloudy',
 'primary',
 'water',
 'road',
 'artisinal_mine',
 'cloudy',
 'cultivation',
 'blooming']
```

In [ ]:
```
gc.collect()
```

Out[23]:
```
8
```

In [ ]:
```
# Initializing imagedatagenerator
train_datagen = ImageDataGenerator(rescale = 1./255., validation_split = 0.2)

# Generating train data generator
train_generator = train_datagen.flow_from_dataframe(dataframe=train_df1,
                                                    directory=train_images,
                                                    x_col='image_name',
                                                    y_col=columns,
                                                    subset='training',
                                                    batch_size=32,seed=42,
                                                    shuffle=True,
                                                    class_mode='raw',
                                                    target_size=(128,128))

#generating validation data
val_generator = train_datagen.flow_from_dataframe(dataframe=train_df1,
                                                  directory=train_images,
                                                  x_col='image_name',
                                                  y_col=columns, subset='validation',
                                                  batch_size=32,seed=42, shuffle=True,
                                                  class_mode='raw',
                                                  target_size=(128,128))
```

```
Found 32384 validated image filenames.
Found 8095 validated image filenames.
```

In [ ]:
```
#setting up step size for training and validation image data
step_train_size = int(np.ceil(train_generator.samples / train_generator.batch_size))
step_val_size = int(np.ceil(val_generator.samples / val_generator.batch_size))
```

In [ ]:
```
gc.collect()
```

Out[26]:
```
0
```

Modelling

In [ ]:
```
#defining our model
# Define the model
def cnn_model():
    model = Sequential()

    # Convolution layers
    model.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu', input_shape=(128, 128, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten Layer
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(units=512, activation='relu'))
    model.add(Dropout(rate=0.5))
    model.add(Dense(units=17, activation='sigmoid'))

    # Compile the model
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model
```

In [ ]:
```
#initialize the model
```

```
model = cnn_model()

# Preview the model architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, 128, 128, 3)      12
 ormalization)

 conv2d (Conv2D)             (None, 128, 128, 32)      896

 conv2d_1 (Conv2D)           (None, 126, 126, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 dropout (Dropout)           (None, 63, 63, 32)        0

 conv2d_2 (Conv2D)           (None, 63, 63, 64)        18496

 conv2d_3 (Conv2D)           (None, 61, 61, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 30, 30, 64)        0

 conv2d_4 (Conv2D)           (None, 30, 30, 128)       73856

 conv2d_5 (Conv2D)           (None, 28, 28, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 14, 14, 128)       0

 conv2d_6 (Conv2D)           (None, 14, 14, 256)       295168

 conv2d_7 (Conv2D)           (None, 12, 12, 256)       590080

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 dropout_3 (Dropout)         (None, 6, 6, 256)         0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 512)               4719104

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 17)                8721

=================================================================
Total params: 5,900,093
Trainable params: 5,900,087
Non-trainable params: 6
_________________________________________________________________
```

In [ ]:
```
# Define callbacks
callbacks = ModelCheckpoint(filepath = 'best_model.hdf5',
                                        monitor = 'val_accuracy',
                                        mode = 'max',
                                        save_best_only = True,
                                        save_weights_only = True)
```

In [ ]:
```
gc.collect()
```

Out[31]: 174

In [ ]:
```
# Fit the model
history = model.fit(x=train_generator,
                    steps_per_epoch = step_train_size,
                    validation_data = val_generator,
                    validation_steps = step_val_size, epochs = 5,
                    callbacks = [callbacks])
```

```
Epoch 1/5
 387/1012 [==========>...................] - ETA: 42:34 - loss: 0.2404 - accuracy: 0.0134
```

In [ ]:
```
gc.collect()
```

In [ ]:
```
#initializing a second model to make predictions
model1 = cnn_model()
```

In [ ]:
```
##adding .jpg extension to image name in the sample submission file
sample_submission = pd.read_csv(test_path)
#sample_submission1 = sample_submission.copy()
sample_submission['image_name'] = sample_submission['image_name'].apply(lambda x: '{}.jpg'.format(x))
sample_submission.head()
```

In [ ]:
```
# Divide the sample submission file into two splits,
# first test_df contains the first 40669 images
test_df = sample_submission.iloc[:40669]['image_name'].reset_index().drop('index', axis =1)
```

In [ ]:
```
test_df['image_name'] = sample_submission['image_name'].apply(lambda x: '{}.jpg'.format(x))
test_df.head()
```

In [ ]:
```
#initialize imagedatagenerator for the test images and also rescaling
```

```
test_datagen = ImageDataGenerator(rescale = 1./255)

#creating a generator for the images found in the first test image files
test_gen = test_datagen.flow_from_dataframe(dataframe=test_df,
                                            directory=test_images,
                                            x_col="image_name",
                                            y_col=None,
                                            batch_size=32,
                                            shuffle=False,
                                            class_mode=None,
                                            target_size=(128,128))

step_test_size = int(np.ceil(test_gen.samples/test_gen.batch_size))
```

In [ ]:
```
#first, we reset the test generator to avoid shuffling of index
test_gen.reset()
pred = model1.predict(test_gen, steps = step_test_size, verbose = 1)
```

In [ ]:
```
#this is to get the filenames in the generator using the attribute .filenames
file_names = test_gen.filenames

#convert the predicted values to a dataframe and join two labels together if the probability of occurrance
#of the label is greater than 0.5
pred_tags = pd.DataFrame(pred)
pred_tags = pred_tags.apply(lambda x: ' '.join(np.array(labels)[x > 0.5]), axis = 1)

#then the result should look like this
result1 = pd.DataFrame({'image_name': file_names, 'tags': pred_tags})
result1.head()
```

In [ ]:
```
gc.collect()
```

In [ ]:
```
#second batch of the test dataset
test1_df = sample_submission.iloc[40669:]['image_name'].reset_index().drop('index', axis =1)
test1_df.head()
```

In [ ]:
```
#creating a generator for the second batch of test image files
test_gen1 = test_datagen.flow_from_dataframe(dataframe=test1_df,
                                            directory=test_additional,
                                            x_col='image_name',
                                            y_col=None,
                                            batch_size=32,
                                            shuffle=False,
                                            class_mode=None,
                                            target_size=(128,128))

step_test_size1 = int(np.ceil(test_gen1.samples/test_gen1.batch_size))
```

In [ ]:
```
#we reset the generator to avoid shuffling, then make prediction on the generator
test_gen1.reset()
pred1 = model1.predict(test_gen1, steps = step_test_size1, verbose = 1)
```

In [ ]:
```
#this is to get the filenames in the generator using the attribute .filenames
file_names1 = test_gen1.filenames

#convert the predicted values to a dataframe
#join two labels together if the prob(occurrance of the label) > 0.5
pred_tags1 = pd.DataFrame(pred1)
pred_tags1 = pred_tags1.apply(lambda x: ''.join(np.array(labels)[x>0.5]), axis = 1)

result1 = pd.DataFrame({'image_name': file_names1, 'tags': pred_tags1})
result1.head()
```

In [ ]:
```
# Final result of the predicted tags for the test images,
# we need to concat the first and second results in
#that order to avoid shuffling the index
last_result = pd.concat([result, result1])

last_result = last_result.reset_index().drop('index', axis =1)

print(last_result.shape)
last_result.head()
```

In [ ]:
```
# Remove the .jpg extension from the image_name of the last_result
last_result['image_name'] = last_result['image_name'].apply(lambda x: x[:-4])
last_result.head()
```

In [ ]:
```
# Finally, we save the result to a csv file using the .to_csv()
# method and setting the index to false.
last_result.to_csv('submission1.csv', index = False)
```

In [ ]:
```

```