Satellite Image Analysis with CNNs for Environmental Monitoring

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import gc
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
import keras as k
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D, BatchNormalization
import cv2
from tqdm import tqdm
from collections import Counter
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import fbeta_score
import plotly.express as px
```

Loading The Dataset

```
In [2]: path = "../input/planets-dataset/planet/planet/"
path_train = os.path.join(path, "train-jpg")
path_test = os.path.join(path, "test-jpg")
print( f"train files: {len(os.listdir(path_train))}, "
   f"test files: {len(os.listdir(path_test))}")
```

```
train files: 40479, test files: 40669
```

In [3]:

```
path_train_class = os.path.join(path, "train_classes.csv")

df_train = pd.read_csv(path_train_class)

print(df_train.shape)

df_train.head()
```

```
(40479, 2)
```

Out[3]:

	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

Exploring and Understanding the Labels in the dataset

```
In [4]: all_tags = [item for sublist in list(df_train['tags'].apply(lambda row:
row.split(" ")).values) for item in sublist]

print('total of {} non-unique tags in all training images'.format(len(all_tags)))
print('average number of labels per image
{}'.format(1.0*len(all_tags)/df_train.shape[0]))
```

```
total of 116278 non-unique tags in all training images
average number of labels per image 2.8725511993873365
```

```
df_train["list_tags"] = df_train.tags.str.split(" ")
row_tags = df_train.list_tags.values
tags = [tag for row in row_tags for tag in row]
df_tags = pd.DataFrame(
    {"tag": counter_tags.keys(), "total": counter_tags.values()}
).sort_values("total")
fig = px.bar(df_tags, x="total", y="tag", orientation="h",color="total",)
fig.update_layout(title="Tags distribution")
fig.show()
```

Machine Learning
Preparing the Data

```
df_train = df_train.drop("list_tags", axis='columns')
df_train.head()
```

Out[6]:

	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

```
x_train = []
y_train = []
flatten = lambda l: [item for sublist in l for item in sublist]
labels = list(set(flatten([l.split(' ') for l in df_train['tags'].values])))
label_map = {l: i for i, l in enumerate(labels)}
inv_label_map = {i: l for l, i in label_map.items()}
for f, tags in tqdm(df_train.values, miniters=1000):
img = cv2.imread('../input/planets-dataset/planet/planet/train-
jpg/{}.jpg'.format(f))
targets = np.zeros(17)
for t in tags.split(' '):
targets[label_map[t]] = 1
x_train.append(cv2.resize(img, (64, 64)))    # Indicate the IMG Size
y_train.append(targets)
x_train = np.array(x_train, np.float16) / 255.
y_train = np.array(y_train, np.uint8)
```

```
100%|██████████| 40479/40479 [07:41<00:00, 87.67it/s]
```

In [8]:

```
y_train = np.array(y_train, np.uint8)
x_train = np.array(x_train, np.float16) / 255.0
```

```
# Splitting the data into train and validation sets.
```

```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size =
0.2, shuffle = True, random_state = 1)

# Prints the shape of the training and validation data.
print("Train data shape:",x_train.shape)
print("Train label shape:",y_train.shape)

print("Validation data shape:",x_val.shape)
print("Validation label shape:",y_val.shape)
```

```
Train data shape: (32383, 64, 64, 3)
Train label shape: (32383, 17)
Validation data shape: (8096, 64, 64, 3)
Validation label shape: (8096, 17)
```

```
gc.collect()
```

```
Out[9]:
        96
```

Establishing Evaluation Metrics for the Model

In [10]:

```
def fbeta(y_true, y_pred, threshold_shift=0):
    beta = 2

    # Clipping y_pred between 0 and 1
    y_pred = K.clip(y_pred, 0, 1)

    # Rounding y_pred to binary values
    y_pred_bin = K.round(y_pred + threshold_shift)

    # Counting true positives, false positives, and false negatives
    tp = K.sum(K.round(y_true * y_pred_bin)) + K.epsilon()
    fp = K.sum(K.round(K.clip(y_pred_bin - y_true, 0, 1)))
    fn = K.sum(K.round(K.clip(y_true - y_pred, 0, 1)))

    # Calculating precision and recall
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
```

```
def accuracy_score(y_true, y_pred, epsilon = 1e-4):

y_true = tf.cast(y_true, tf.float32)

y_pred = tf.cast(tf.greater(tf.cast(y_pred, tf.float32), tf.constant(0.5)), tf.float32)

tp = tf.reduce_sum(y_true * y_pred, axis = 1)

fp = tf.reduce_sum(y_pred, axis = 1) - tp

fn = tf.reduce_sum(y_true, axis = 1) - tp

y_true = tf.cast(y_true, tf.bool)

y_pred = tf.cast(y_pred, tf.bool)

tn = tf.reduce_sum(tf.cast(tf.logical_not(y_true), tf.float32) * tf.cast(tf.logical_not(y_pred), tf.float32),

axis = 1)

return (tp + tn)/(tp + tn + fp + fn + epsilon)
```

Constructing the Neural Network Architecture

```
optimizer_Adam = Adam()
optimizer_Adagrad = Adagrad()
optimizer_RMSprop = RMSprop()
model = keras.Sequential()
model.add(BatchNormalization(input_shape=(64, 64, 3)))
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(17, activation='sigmoid'))
model.compile(optimizer=optimizer_Adam,
        loss='binary_crossentropy',
       metrics=[fbeta, accuracy_sco
      history = model.fit(x_train, y_train,
                   batch_size=128,
                   epochs=10,
                   verbose=1,
                   validation_data=(x_val, y_val))
```

```
2023-01-19 05:34:51.633669: I tensorflow/core/common_runtime/process_ut
il.cc:146] Creating new thread pool with default inter op setting: 2. T
une using inter_op_parallelism_threads for best performance.
```

```
2023-01-19 05:34:55.028825: I tensorflow/compiler/mlir/mlir_graph_optim
ization_pass.cc:185] None of the MLIR Optimization Passes are enabled (
registered 2)
Epoch 1/10
253/253 [==============================] - 231s 907ms/step - loss: 0.23
69 - fbeta: 0.6281 - accuracy_score: 0.9103 - val_loss: 0.2538 - val_fb
eta: 0.6118 - val_accuracy_score: 0.9069
Epoch 2/10
253/253 [==============================] - 227s 896ms/step - loss: 0.19
70 - fbeta: 0.6946 - accuracy_score: 0.9231 - val_loss: 0.2217 - val_fb
eta: 0.6582 - val_accuracy_score: 0.9170
Epoch 3/10
253/253 [==============================] - 230s 910ms/step - loss: 0.17
28 - fbeta: 0.7379 - accuracy_score: 0.9323 - val_loss: 0.1644 - val_fb
eta: 0.7372 - val_accuracy_score: 0.9352
Epoch 4/10
253/253 [==============================] - 230s 908ms/step - loss: 0.16
17 - fbeta: 0.7569 - accuracy_score: 0.9370 - val_loss: 0.1504 - val_fb
eta: 0.7691 - val_accuracy_score: 0.9412
Epoch 5/10
253/253 [==============================] - 231s 914ms/step - loss: 0.15
26 - fbeta: 0.7725 - accuracy_score: 0.9408 - val_loss: 0.1439 - val_fb
eta: 0.7831 - val_accuracy_score: 0.9431
Epoch 6/10
253/253 [==============================] - 229s 907ms/step - loss: 0.14
56 - fbeta: 0.7820 - accuracy_score: 0.9429 - val_loss: 0.1397 - val_fb
eta: 0.7882 - val_accuracy_score: 0.9448
Epoch 7/10
253/253 [==============================] - 227s 898ms/step - loss: 0.14
13 - fbeta: 0.7901 - accuracy_score: 0.9447 - val_loss: 0.1371 - val_fb
eta: 0.7848 - val_accuracy_score: 0.9454
Epoch 8/10
253/253 [==============================] - 228s 901ms/step - loss: 0.13
72 - fbeta: 0.7963 - accuracy_score: 0.9462 - val_loss: 0.1354 - val_fb
eta: 0.7900 - val_accuracy_score: 0.9457
Epoch 9/10
253/253 [==============================] - 231s 911ms/step - loss: 0.13
21 - fbeta: 0.8030 - accuracy_score: 0.9476 - val_loss: 0.1337 - val_fb
eta: 0.8061 - val_accuracy_score: 0.9470
Epoch 10/10
253/253 [==============================] - 228s 901ms/step - loss: 0.12
75 - fbeta: 0.8117 - accuracy_score: 0.9493 - val_loss: 0.1335 - val_fb
eta: 0.7949 - val_accuracy_score: 0.9474
```

```
plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])
```

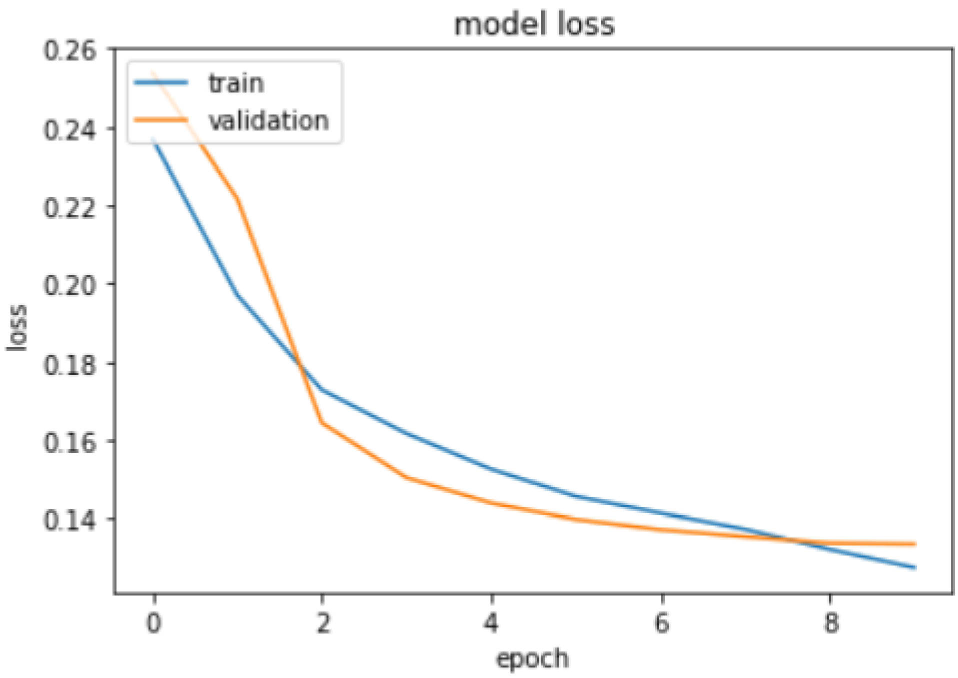
```
plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()
```




```
model.evaluate(x_val, y_val)

train_fscore = fbeta_score(y_train, np.round(model.predict(x_train)), beta=2,average = 'weighted')

print("train fscore: ", train_fscore)

val_fscore = fbeta_score(y_val, np.round(model.predict(x_val)), beta=2, average = 'weighted')

print("val fscore: ", val_fscore)
```

```
253/253 [==============================] - 15s 59ms/step - loss: 0.1335
- fbeta: 0.7952 - accuracy_score: 0.9474
train fscore:  0.7965989848478681
val fscore:  0.7771786500700735
```

```
df_samplesub = pd.read_csv('../input/planets-dataset/planet/planet/sample_submission.csv')

test = df_samplesub[0 : 40669]

files = df_samplesub[40669 : ]

test_img = []

for image_name, tags in tqdm(test.values, miniters=1000):

arr = cv2.imread('../input/planets-dataset/planet/planet/test-jpg/{}.jpg'.format(image_name))

test_img.append(cv2.resize(arr, (64, 64)))

for image_name, tags in tqdm(files.values, miniters=1000):

arr = cv2.imread('../input/planets-dataset/test-jpg-additional/test-jpg-additional/{}.jpg'.format(image_name))

test_img.append(cv2.resize(arr, (64, 64)))

test_img = np.array(test_img, np.float16)/255.0
```

```
100%|██████████| 40669/40669 [08:17<00:00, 81.73it/s]
100%|██████████| 20522/20522 [03:52<00:00, 88.44it/s]
```

```
gc.collect()
```

Out[19]:

6626

```
yres = []

predictions = model.predict(test_img, batch_size = 64, verbose = 2)

yres.append(predictions)
```

957/957 - 102s

```
gc.collect()
```

Out[21]:

788

```
sub = np.array(yres[0])

for i in range (1, len(yres)):
```

```
  sub += np.array(yres[i])
sub = pd.DataFrame(sub, columns = label_map)
preds = []

# Loop through the sample submission DataFrame
for i in tqdm(range(sub.shape[0]), miniters=1000):
    # Get the i-th row of the DataFrame
    a = sub.loc[[i]]
    # Apply a lambda function to get a Boolean array indicating which columns have values greater than 0.2
    a = a.apply(lambda x: x > 0.2, axis=1)
    # Transpose the DataFrame
    a = a.transpose()
    # Get the rows where the Boolean array is True
    a= a.loc[a[i] == True]
    # Join the index of the DataFrame (which contains the tags) into a single string
    ' '.join(list(a.index))
    # Append the string of tags to the preds list
    preds.append(' '.join(list(a.index)))

# Assign the preds list as the 'tags' column of the sample submission DataFrame
df_samplesub['tags'] = preds

# Save the sample submission DataFrame to a CSV file
df_samplesub.to_csv('CMT_submission.csv', index=False)
100%|██████████| 61191/61191 [01:51<00:00, 550.94it/s]
```

image_nar	tags
test_0	cloudy
test_1	cloudy
test_2	
test_3	cloudy
test_4	
test_5	cloudy
test_6	cloudy
test_7	primary water clear artisinal_mine
test_8	cloudy
test_9	cloudy