In [1]:
```
import os
import tensorflow as tf
```

```
2023-08-29 03:29:13.847926: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Functions

> *Note that serialize_images() will resize images in the image = tf.image.resize(image, [128,128])

In [2]:
```
# Necessary functions per the TensorFlow Documentation:
# url:  https://www.tensorflow.org/tutorials/load_data/tfrecord#walkthrough_reading_and_writing_image_data

def _int64_feature(value):
  """Returns an int64_list from a bool / enum / int / uint."""
  return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

def _bytes_feature(value):
  """Returns a bytes_list from a string / byte."""
  if isinstance(value, type(tf.constant(0))):
    value = value.numpy() # BytesList won't unpack a string from an EagerTensor.
  return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))


# Create a dictionary with features that may be relevant.
def image_example(image_string, label):
  image_shape = tf.io.decode_jpeg(image_string).shape

  feature = {
      'height': _int64_feature(image_shape[0]),
      'width': _int64_feature(image_shape[1]),
      'depth': _int64_feature(image_shape[2]),
      'label': _int64_feature(label),
      'image_raw': _bytes_feature(image_string),
  }

  return tf.train.Example(features=tf.train.Features(feature=feature))

# Get image file paths and labels
def get_paths_and_labels(data_dir, class_list):
    class_names = class_list
    image_paths = []
    labels = []
    for i, class_name in enumerate(class_names):
        class_path = os.path.join(data_dir, class_name)
        if os.path.isdir(class_path):
            label = i
            for filename in os.listdir(class_path):
                if filename.lower().endswith(('.jpg')):
                    image_path = os.path.join(class_path, filename)
                    image_paths.append(image_path)
                    labels.append(label)
    return image_paths, labels

# Serialize to string per tensorflow documentation
def serialize_images(image_paths, labels, record_file):
    with tf.io.TFRecordWriter(record_file) as writer:
        for image_path, label in zip(image_paths, labels):
            image_string = open(image_path, 'rb').read()
            image = tf.image.decode_jpeg(image_string, channels = 3) #confirm image encoding
            image = tf.image.resize(image, [64,64])  # specify desired height and width
            image = tf.cast(image, tf.uint8)
            image_string = tf.image.encode_jpeg(image).numpy()
            tf_example = image_example(image_string, label)
            writer.write(tf_example.SerializeToString())
```

Encoding issue and solution

> Issue: The dataset provided from the kaggle platform, contains a folder of cloudy images. Initially in the main module these images all displayed as solid black (0,0,0) in rgb after they were decoded from their TFRecord format. I inspected the files compared to the image files in the pother directories using the terminal command 'file' and found that there were 4 components to the cloudy images, compared to 3 in all other files.
>
> Solution: I was able to write a short script first to identify the image mode which was CMYK, then I altered the script to convert all images to rgb.

In [45]:
```
#Convert cloudy images from CMYK to RGB

from PIL import Image


def to_rgb(input_dir):

    for filename in os.listdir(input_dir):
        if filename.endswith('.jpg'):
            img_path = os.path.join(input_dir, filename)
            img = Image.open(img_path)
            img = img.convert('RGB')
            img.save(img_path)

to_rgb('Data/data_dir/test/cloudy')
to_rgb('Data/data_dir/train/cloudy')
to_rgb('Data/data_dir/validation/cloudy')
```

TFRecord writing

In [6]:
```
#Create training set TFRecord


#Define the main data directory and a list of classifications
data_dir = 'Data/data_dir/train' #main data directory path
class_list = ['cloudy', 'desert', 'green_area', 'water'] #list of classifications
record_file = 'train.tfrecords' #Define the filename

# Call the functions to get image file paths and labels
image_paths, labels = get_paths_and_labels(data_dir, class_list)

# Create the TFRecord file
serialize_images(image_paths, labels, record_file)
```

In [7]:
```
#Create test set TFRecord


#Define the main data directory and a list of classifications
data_dir = 'Data/data_dir/test' #main data directory path
class_list = ['cloudy', 'desert', 'green_area', 'water'] #list of classifications
record_file = 'test.tfrecords' #Define the filename

# Call the functions to get image file paths and labels
image_paths, labels = get_paths_and_labels(data_dir, class_list)

# Create the TFRecord file
serialize_images(image_paths, labels, record_file)
```

In [8]:
```
#Create validate set TFRecord


#Define the main data directory and a list of classifications
data_dir = 'Data/data_dir/validation' #main data directory path
class_list = ['cloudy', 'desert', 'green_area', 'water'] #list of classifications
record_file = 'validate.tfrecords' #Define the filename

# Call the functions to get image file paths and labels
image_paths, labels = get_paths_and_labels(data_dir, class_list)

# Create the TFRecord file
serialize_images(image_paths, labels, record_file)
```

In [ ]:
```

```