Big Data Coursework - Questions

Data Processing and Machine Learning in the Cloud

This is the **INM432 Big Data coursework 2024**. This coursework contains extended elements of **theory** and **practice**, mainly around parallelisation of tasks withSpark and a bit about parallel training using TensorFlow.

Code and Report

Your tasks parallelization of tasks in PySpark, extension, evaluation, and theoretical reflection. Please complete and submit the **coding tasks** in a copy of **this notebook**. Write your code in the **indicated cells** and **include** the **output** in the submitted notebook.
Make sure that **your code contains comments** on its **stucture** and explanations of its **purpose**.

Provide also a **report** with the **textual answers in a separate document**. Include **screenshots** from the Google Cloud web interface (don't use the SCREENSHOT function that Google provides, but take a picture of the graphs you see for the VMs) and result tables, as well as written text about the analysis.

Submission

Download and submit **your version of this notebook** as an **.ipynb** file and also submit a **shareable link** to your notebook on Colab in your report (created with the Colab 'Share' function) (**and don't change the online version after submission**).

Further, provide your **report as a PDF document**. **State the number of words** in the document at the end. The report should **not have more than 2000 words**.

Please also submit **a PDF of your Jupyter notebook**.

Introduction and Description

This coursework focuses on parallelisation and scalability in the cloud with Spark and TesorFlow/Keras. We start with code based on **lessons 3 and 4** of the *Fast and Lean Data Science* (https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/fast-and-lean-data-science) course by Martin Gorner. The course is based on Tensorflow for data processing and MachineLearning. Tensorflow's data processing approach is somewhat similar to that of Spark, but you don't need to study Tensorflow, just make sure you understand the high-level structure.
What we will do here is **parallelising pre-processing**, and **measuring** performance, and we will perform **evaluation** and **analysis** on the cloud performance, as well as **theoretical discussion**.

This coursework contains **3 sections**.

Section 0

This section just contains some necessary code for setting up the environment. It has no tasks for you (but do read the code and comments).

Section 1

Section 1 is about preprocessing a set of image files. We will work with a public dataset “Flowers” (3600 images, 5 classes). This is not a vast dataset, but it keeps the tasks more manageable for development and you can scale up later, if you like.

In **'Getting Started'** we will work through the data preprocessing code from *Fast and Lean Data Science* which uses TensorFlow's `tf.data` package. There is no task for you here, but you will need to re-use some of this code later.

In **Task 1** you will **parallelise the data preprocessing in Spark**, using Google Cloud (GC) Dataproc. This involves adapting the code from 'Getting Started' to use Spark and running it in the cloud.

Section 2

In **Section 2** we are going to **measure the speed of reading data** in the cloud. In **Task 2** we will **paralellize the measuring** of different configurations **using Spark**.

Section 3

This section is about the theoretical discussion, based on one paper, in **Task 3**. The answers should be given in the PDF report.

General points

For **all coding tasks**, take the **time of the operations** and for the cloud operations, get performance **information from the web interfaces** for your reporting and analysis.

The **tasks** are **mostly independent** of each other. The later tasks can mostly be addressed

Section 0: Set-up

As usual, you need to run the **imports and authentication every time you work with this notebook**. Use the **local Spark** installation for development before you send jobs to the cloud.

Read through this section once and **fill in the project ID the first time**, then you can just step straight throught this at the beginning of each session - except for the two authentication cells.

Imports

We import some **packages that will be needed throughout**. For the **code that runs in the cloud**, we will need **separate import sections** that will need to be partly different from the one below.

In [ ]:
```
import os, sys, math
import numpy as np
import scipy as sp
import scipy.stats
import time
import datetime
import string
import random
from matplotlib import pyplot as plt
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
```

```
Tensorflow version 2.15.0
```

Cloud and Drive authentication

This is for **authenticating with with GCS Google Drive**, so that we can create and use our own buckets and access Dataproc and AI-Platform.

This section **starts with the two interactive authentications**.

First, we mount Google Drive for persistent local storage and create a directory `DB-CW` thay you can use for this work. Then we'll set up the cloud environment, including a storage bucket.

In [ ]:
```
print('Mounting google drive...')
from google.colab import drive
drive.mount('/content/drive')
%cd "/content/drive/MyDrive"
!mkdir BD-CW
%cd "/content/drive/MyDrive/BD-CW"
```

```
Mounting google drive...
Mounted at /content/drive
/content/drive/MyDrive
mkdir: cannot create directory ‘BD-CW’: File exists
/content/drive/MyDrive/BD-CW
```

Next, we authenticate with the GCS to enable access to Dataproc and AI-Platform.

In [ ]:
```
import sys
if 'google.colab' in sys.modules:
    from google.colab import auth
    auth.authenticate_user()
```

It is useful to **create a new Google Cloud project** for this coursework. You can do this on the GC Console page (https://console.cloud.google.com) by clicking on the entry at the top, right of the *Google Cloud Platform* and choosing *New Project*. **Copy** the **generated project ID** to the next cell. Also **enable billing** and the **Compute, Storage and Dataproc** APIs like we did during the labs.

In [ ]:

```
PROJECT = 'big-data-cw-420116'  ### USE YOUR GOOGLE CLOUD PROJECT ID HE
!gcloud config set project $PROJECT
REGION = 'us-central1'
CLUSTER = '{}-cluster'.format(PROJECT)
!gcloud config set compute/region $REGION
!gcloud config set dataproc/region $REGION

!gcloud config list # show some information
```

```
Updated property [core/project].
WARNING: Property validation for compute/region was skipped.
Updated property [compute/region].
Updated property [dataproc/region].
[component_manager]
disable_update_check = True
[compute]
region = us-central1
[core]
account = antonio-jose.lopez-roldan@city.ac.uk
project = big-data-cw-420116
[dataproc]
region = us-central1

Your active configuration is: [default]
```

With the cell below, we **create a storage bucket** that we will use later for **global storage**. If the bucket exists you will see a "ServiceException: 409 ...", which does not cause any problems. **You must create your own bucket to have write access.**

In [ ]:

```
BUCKET = 'gs://{}-storage'.format(PROJECT)
!gsutil mb $BUCKET
```

```
Creating gs://big-data-cw-420116-storage/...
ServiceException: 409 A Cloud Storage bucket named 'big-data-cw-420116-sto
rage' already exists. Try another name. Bucket names must be globally uniq
ue across all Google Cloud projects, including those outside of your organ
ization.
```

The cell below just **defines some routines for displaying images** that will be **used later**. You can see the code by double-clicking, but you don't need to study this.

In [ ]:

```
#@title Utility functions for image display **[RUN THIS TO ACTIVATE]**
def display_9_images_from_dataset(dataset):
  plt.figure(figsize=(13,13))
  subplot=331
  for i, (image, label) in enumerate(dataset):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image.numpy().astype(np.uint8))
    plt.title(str(label.numpy()), fontsize=16)
    # plt.title(label.numpy().decode(), fontsize=16)
    subplot += 1
    if i==8:
      break
  plt.tight_layout()
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()

def display_training_curves(training, validation, title, subplot):
  if subplot%10==1: # set up the subplots on the first call
    plt.subplots(figsize=(10,10), facecolor='#F0F0F0')
    plt.tight_layout()
  ax = plt.subplot(subplot)
  ax.set_facecolor('#F8F8F8')
  ax.plot(training)
  ax.plot(validation)
  ax.set_title('model '+ title)
  ax.set_ylabel(title)
  ax.set_xlabel('epoch')
  ax.legend(['train', 'valid.'])

def dataset_to_numpy_util(dataset, N):
    dataset = dataset.batch(N)
    for images, labels in dataset:
        numpy_images = images.numpy()
        numpy_labels = labels.numpy()
        break;
    return numpy_images, numpy_labels

def title_from_label_and_target(label, correct_label):
  correct = (label == correct_label)
  return "{} [{}{}{}]".format(CLASSES[label], str(correct), ', shoud be
                              CLASSES[correct_label] if not correct els

def display_one_flower(image, title, subplot, red=False):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16, color='red' if red else 'black')
    return subplot+1

def display_9_images_with_predictions(images, predictions, labels):
  subplot=331
  plt.figure(figsize=(13,13))
  classes = np.argmax(predictions, axis=-1)
  for i, image in enumerate(images):
    title, correct = title_from_label_and_target(classes[i], labels[i])
    subplot = display_one_flower(image, title, subplot, not correct)
    if i >= 8:
      break;

  plt.tight_layout()
```

```
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()

```

In [ ]:
```
# Check list of services working and enabled
!gcloud services list --enabled
```

```
NAME                                    TITLE
analyticshub.googleapis.com             Analytics Hub API
artifactregistry.googleapis.com         Artifact Registry API
autoscaling.googleapis.com              Cloud Autoscaling API
bigquery.googleapis.com                 BigQuery API
bigqueryconnection.googleapis.com       BigQuery Connection API
bigquerydatapolicy.googleapis.com       BigQuery Data Policy API
bigquerymigration.googleapis.com        BigQuery Migration API
bigqueryreservation.googleapis.com      BigQuery Reservation API
bigquerystorage.googleapis.com          BigQuery Storage API
cloudapis.googleapis.com                Google Cloud APIs
cloudresourcemanager.googleapis.com     Cloud Resource Manager API
cloudtrace.googleapis.com               Cloud Trace API
compute.googleapis.com                  Compute Engine API
container.googleapis.com                Kubernetes Engine API
containerfilesystem.googleapis.com      Container File System API
containerregistry.googleapis.com        Container Registry API
dataform.googleapis.com                 Dataform API
dataplex.googleapis.com                 Cloud Dataplex API
dataproc-control.googleapis.com         Cloud Dataproc Control API
dataproc.googleapis.com                 Cloud Dataproc API
datastore.googleapis.com                Cloud Datastore API
dns.googleapis.com                      Cloud DNS API
gkebackup.googleapis.com                Backup for GKE API
iam.googleapis.com                      Identity and Access Management (IAM)
API
iamcredentials.googleapis.com           IAM Service Account Credentials API
logging.googleapis.com                  Cloud Logging API
monitoring.googleapis.com               Cloud Monitoring API
networkconnectivity.googleapis.com      Network Connectivity API
oslogin.googleapis.com                  Cloud OS Login API
pubsub.googleapis.com                   Cloud Pub/Sub API
servicemanagement.googleapis.com        Service Management API
serviceusage.googleapis.com             Service Usage API
sql-component.googleapis.com            Cloud SQL
storage-api.googleapis.com              Google Cloud Storage JSON API
storage-component.googleapis.com        Cloud Storage
storage.googleapis.com                  Cloud Storage API
```

Install Spark locally for quick testing

You can use the cell below to **install Spark locally on this Colab VM** (like in the labs), to do quicker small-scale interactive testing. Using Spark in the cloud with **Dataproc is still required for the final version**.

In [ ]:
```
%cd
!apt-get update -qq
!apt-get install openjdk-8-jdk-headless -qq >> /dev/null # send any out
!tar -xzf "/content/drive/My Drive/Big_Data/data/spark/spark-3.5.0-bin-

!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/root/spark-3.5.0-bin-hadoop3" # Was gettin
import findspark
findspark.init()
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)
```

```
/root
3.5.0

/usr/lib/python3.10/subprocess.py:1796: RuntimeWarning: os.fork() was call
ed. os.fork() is incompatible with multithreaded code, and JAX is multithr
eaded, so this will likely lead to a deadlock.
  self.pid = _posixsubprocess.fork_exec(

<SparkContext master=local[*] appName=pyspark-shell>
```

Section 1: Data pre-processing

This section is about the **pre-processing of a dataset** for deep learning. We first look at a ready-made solution using Tensorflow and then we build a implement the same process with Spark. The tasks are about **parallelisation** and **analysis** the performance of the cloud implementations.

1.1 Getting started

In this section, we get started with the data pre-processing. The code is based on lecture 3 of the 'Fast and Lean Data Science' course.

This code is using the TensorFlow `tf.data` package, which supports map functions, similar to Spark. Your **task** will be to **re-implement the same approach in Spark**.

We start by **setting some variables for the *Flowers* dataset**.

In [ ]:
```
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
    # labels for the data
```

We **read the image files** from the public GCS bucket that contains the *Flowers* dataset. **TensorFlow** has **functions** to execute glob patterns that we use to calculate the the number of images in total and per partition (rounded up as we cannont deal with parts of images).

In [ ]:

```
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images
partition_size = math.ceil(1.0 * nb_images / PARTITIONS) # images per p
print("GCS_PATTERN matches {} images, to be divided into {} partitions
```

```
GCS_PATTERN matches 3670 images, to be divided into 16 partitions with up
to 230 images each.
```

Map functions

In order to read use the images for learning, they need to be **preprocessed** (decoded, resized, cropped, and potentially recompressed). Below are **map functions** for these steps. You **don't need to study** the **internals of these functions** in detail.

In [ ]:

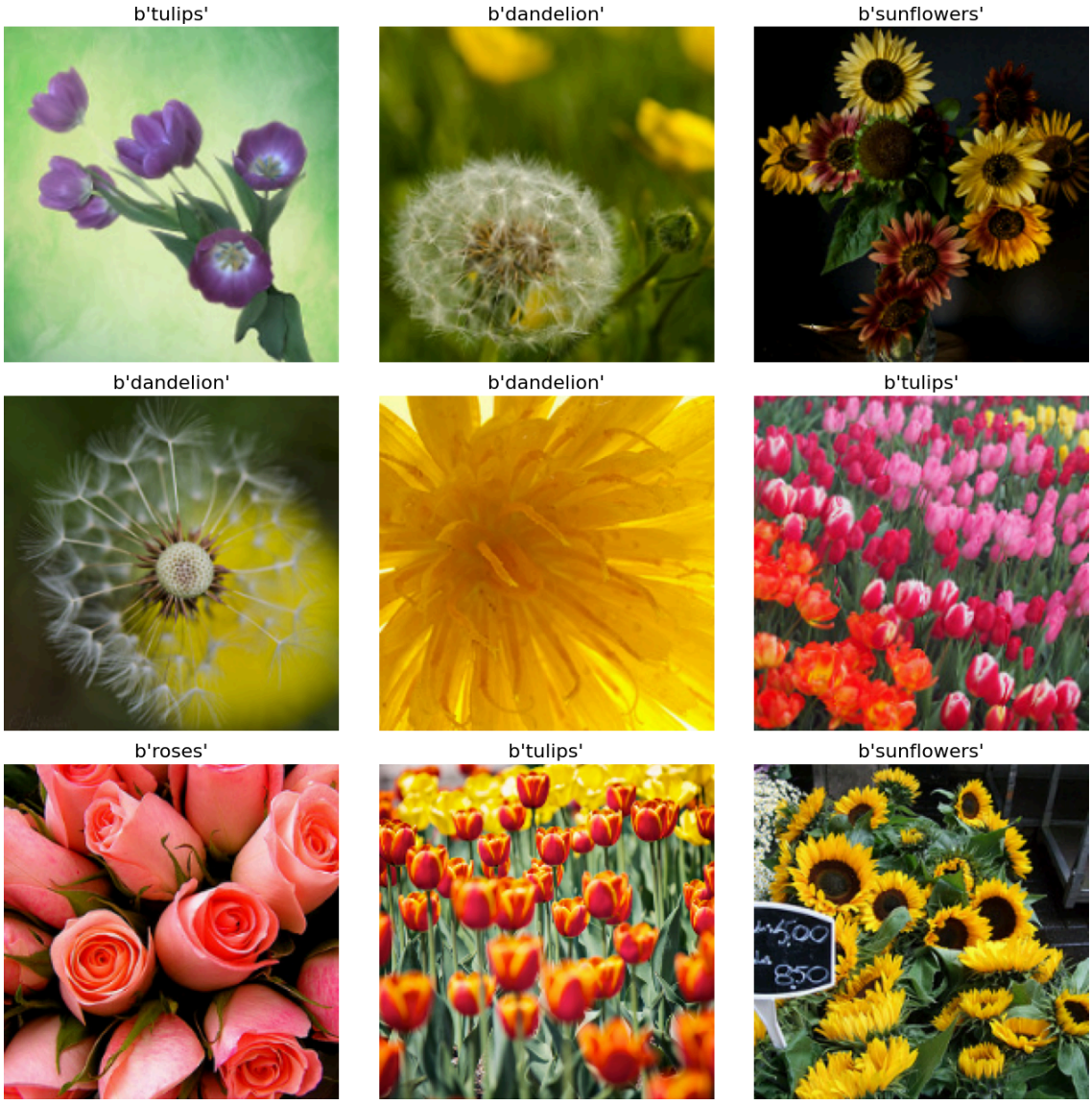
```
def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

def recompress_image(image, label):
    # this reduces the amount of data, but takes some time
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_down
    return image, label
```

With `tf.data`, we can apply decoding and resizing as map functions.

In [ ]:
```
dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN) # This also shuffle
dsetDecoded = dsetFiles.map(decode_jpeg_and_label)
dsetResized = dsetDecoded.map(resize_and_crop_image)
```

We can also look at some images using the image display function defined above (the one with the hidden code).

In [ ]:
```
display_9_images_from_dataset(dsetResized)
```



Now, let's test continuous reading from the dataset. We can see that reading the first 100 files already takes some time.

In [ ]:
```
sample_set = dsetResized.batch(10).take(10) # take 10 batches of 10 ima
for image, label in sample_set:
    print("Image batch shape {}, {})".format(image.numpy().shape,
        [lbl.decode('utf8') for lbl in label.numpy()]))
```

```
Image batch shape (10, 192, 192, 3), ['sunflowers', 'dandelion', 'daisy',
'tulips', 'tulips', 'tulips', 'tulips', 'tulips', 'tulips', 'daisy'])
Image batch shape (10, 192, 192, 3), ['daisy', 'roses', 'tulips', 'tulip
s', 'roses', 'daisy', 'roses', 'dandelion', 'roses', 'roses'])
Image batch shape (10, 192, 192, 3), ['daisy', 'sunflowers', 'dandelion',
'sunflowers', 'daisy', 'sunflowers', 'tulips', 'tulips', 'dandelion', 'dan
delion'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'tulips', 'tulips', 't
ulips', 'tulips', 'dandelion', 'roses', 'dandelion', 'sunflowers', 'dandel
ion'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'sunflowers', 'daisy',
'tulips', 'daisy', 'tulips', 'dandelion', 'roses', 'tulips', 'sunflower
s'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'sunflowers', 'roses',
'roses', 'sunflowers', 'tulips', 'sunflowers', 'dandelion', 'daisy', 'tuli
ps'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'dandelion', 'sunflower
s', 'dandelion', 'daisy', 'sunflowers', 'dandelion', 'sunflowers', 'rose
s', 'dandelion'])
Image batch shape (10, 192, 192, 3), ['roses', 'tulips', 'tulips', 'dais
y', 'roses', 'dandelion', 'sunflowers', 'dandelion', 'sunflowers', 'tulip
s'])
Image batch shape (10, 192, 192, 3), ['daisy', 'tulips', 'sunflowers', 'tu
lips', 'daisy', 'sunflowers', 'dandelion', 'sunflowers', 'dandelion', 'dan
delion'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'tulips', 'sunflower
s', 'dandelion', 'daisy', 'dandelion', 'daisy', 'tulips', 'dandelion', 'da
isy'])
```

1.2 Improving Speed

Using individual image files didn't look very fast. The 'Lean and Fast Data Science' course introduced **two techniques to improve the speed**.

Recompress the images

By **compressing** the images in the **reduced resolution** we save on the size. This **costs some CPU time** upfront, but **saves network and disk bandwith**, especially when the data are **read multiple times**.

In [ ]:
```
# This is a quick test to get an idea how long recompressions takes.
dataset4 = dsetResized.map(recompress_image)
test_set = dataset4.batch(10).take(10)
for image, label in test_set:
    print("Image batch shape {}, {})".format(image.numpy().shape, [lbl.
```

```
Image batch shape (10,), ['sunflowers', 'tulips', 'tulips', 'tulips', 'dai
sy', 'sunflowers', 'dandelion', 'roses', 'daisy', 'tulips'])
Image batch shape (10,), ['daisy', 'roses', 'roses', 'tulips', 'dandelio
n', 'tulips', 'daisy', 'roses', 'tulips', 'dandelion'])
Image batch shape (10,), ['dandelion', 'sunflowers', 'daisy', 'roses', 'su
nflowers', 'dandelion', 'tulips', 'tulips', 'daisy', 'roses'])
Image batch shape (10,), ['daisy', 'tulips', 'dandelion', 'dandelion', 'ro
ses', 'roses', 'daisy', 'roses', 'daisy', 'sunflowers'])
Image batch shape (10,), ['tulips', 'roses', 'daisy', 'daisy', 'dandelio
n', 'dandelion', 'dandelion', 'daisy', 'dandelion', 'dandelion'])
Image batch shape (10,), ['tulips', 'dandelion', 'roses', 'daisy', 'tulip
s', 'sunflowers', 'dandelion', 'dandelion', 'roses', 'tulips'])
Image batch shape (10,), ['daisy', 'tulips', 'daisy', 'dandelion', 'dais
y', 'daisy', 'tulips', 'tulips', 'sunflowers', 'roses'])
Image batch shape (10,), ['daisy', 'roses', 'sunflowers', 'tulips', 'dande
lion', 'dandelion', 'dandelion', 'dandelion', 'dandelion', 'dandelion'])
Image batch shape (10,), ['dandelion', 'sunflowers', 'dandelion', 'roses',
'roses', 'roses', 'daisy', 'dandelion', 'dandelion', 'sunflowers'])
Image batch shape (10,), ['tulips', 'daisy', 'dandelion', 'dandelion', 'da
ndelion', 'daisy', 'tulips', 'sunflowers', 'sunflowers', 'roses'])
```

Write the dataset to TFRecord files

By writing **multiple preprocessed samples into a single file**, we can make further speed gains. We distribute the data over **partitions** to facilitate **parallelisation** when the data are used. First we need to **define a location** where we want to put the file.

In [ ]:
```
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for
```

Now we can **write the TFRecord files** to the bucket.

Running the cell takes some time and **only needs to be done once** or not at all, as you can use the publicly available data for the next few cells. For convenience I have commented out the call to `write_tfrecords` at the end of the next cell. You don't need to run it (it takes some time), but you'll need to use the code below later (but there is no need to study it in detail).

There is a **ready-made pre-processed data** versions available here: `gs://flowers-public/tfrecords-jpeg-192x192-2/` , that we can use for testing.

In [ ]:

```
# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of

def _int_feature(list_of_ints): # int64
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data r
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (ord
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0,
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the l
        "class": _int_feature([class_num]) #,      # one class in the l
    }
    return tf.train.Example(features=tf.train.Features(feature=feature)

def write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size): # write the
    print("Writing TFRecords")
    tt0 = time.time()
    filenames = tf.data.Dataset.list_files(GCS_PATTERN)
    dataset1 = filenames.map(decode_jpeg_and_label)
    dataset2 = dataset1.map(resize_and_crop_image)
    dataset3 = dataset2.map(recompress_image)
    dataset4 = dataset3.batch(partition_size) # partitioning: there wil
    for partition, (image, label) in enumerate(dataset4):
        # batch size used as partition size here
        partition_size = image.numpy().shape[0]
        # good practice to have the number of records in the filename
        filename = GCS_OUTPUT + "{:02d}-{}.tfrec".format(partition, par
        # You need to change GCS_OUTPUT to your own bucket to actually
        with tf.io.TFRecordWriter(filename) as out_file:
            for i in range(partition_size):
                example = to_tfrecord(out_file,
                                      image.numpy()[i], # re-compressed i
                                      label.numpy()[i] #
                                      )
                out_file.write(example.SerializeToString())
        print("Wrote file {} containing {} records".format(filename, pa
    print("Total time: "+str(time.time()-tt0))

write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size) # uncomment to r
```

```
Writing TFRecords
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s00-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s01-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s02-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s03-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s04-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s05-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s06-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s07-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s08-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s09-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s10-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s11-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s12-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s13-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s14-230.tfrec containing 230 records
Wrote file gs://big-data-cw-420116-storage/tfrecords-jpeg-192x192-2/flower
s15-220.tfrec containing 220 records
Total time: 223.18424677848816
```

Test the TFRecord files

We can now **read from the TFRecord files**. By default, we use the files in the public bucket. Comment out the 1st line of the cell below to use the files written in the cell above.

In [ ]:

```
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
datasetTfrec = load_dataset(filenames)
```

Let's have a look **if reading from the TFRecord** files is **quicker**.

In [ ]:

```
batched_dataset = datasetTfrec.batch(10)
sample_set = batched_dataset.take(10)
for image, label in sample_set:
    print("Image batch shape {}, {})".format(image.numpy().shape, \
                          [str(lbl) for lbl in label.numpy()]))
```

```
Image batch shape (10, 192, 192, 3), ['1', '0', '4', '4', '1', '3', '2',
'1', '4', '2'])
Image batch shape (10, 192, 192, 3), ['1', '0', '4', '1', '2', '1', '1',
'4', '0', '1'])
Image batch shape (10, 192, 192, 3), ['1', '2', '3', '1', '0', '1', '2',
'4', '4', '3'])
Image batch shape (10, 192, 192, 3), ['1', '3', '1', '4', '2', '3', '0',
'3', '0', '3'])
Image batch shape (10, 192, 192, 3), ['3', '3', '3', '4', '3', '0', '1',
'0', '4', '2'])
Image batch shape (10, 192, 192, 3), ['2', '4', '1', '3', '0', '3', '4',
'3', '3', '1'])
Image batch shape (10, 192, 192, 3), ['0', '4', '3', '4', '0', '4', '3',
'3', '0', '3'])
Image batch shape (10, 192, 192, 3), ['0', '0', '1', '0', '2', '4', '2',
'2', '3', '0'])
Image batch shape (10, 192, 192, 3), ['2', '1', '1', '4', '1', '2', '3',
'3', '0', '3'])
Image batch shape (10, 192, 192, 3), ['0', '4', '0', '4', '4', '4', '1',
'1', '1', '1'])
```

Wow, we have a **massive speed-up**! The repackageing is worthwhile :-)

Task 1: Write TFRecord files to the cloud with Spark (40%)

Since recompressing and repackaging is very effective, we would like to be able to do it inparallel for large datasets. This is a relatively straightforward case of **parallelisation**. We will **use Spark to implement** the same process as above, but in parallel.

1a) Create the script (14%)

Re-implement the pre-processing in Spark, using Spark mechanisms for **distributing** the workload **over multiple machines**.

You need to:

i) **Copy** over the **mapping functions** (see section 1.1) and **adapt** the resizing and recompression functions **to Spark** (only one argument). (3%)

ii) **Replace** the TensorFlow **Dataset objects with RDDs**, starting with an RDD that contains the list of image filenames. (3%)

iii) **Sample** the the RDD to a smaller number at an appropriate position in the code. Specify a sampling factor of 0.02 for short tests. (1%)

iv) Then **use the functions from above** to write the TFRecord files. (3%)

v) The code for **writing to the TFRecord files** needs to be put into a function, that can be applied to every partition with the 'RDD.mapPartitionsWithIndex' (https://spark.apache.org/docs/2.4.8/api/python/pyspark.html#pyspark.RDD.mapPartitionsWith function. The return value of that function is not used here, but you should return the filename, so that you have a list of the created TFRecord files. (4%)

In [ ]:

```
                                        ### CODING TASK ### ("Done & Working")
# Setting the working environment
import argparse
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp
#import scipy.stats
import time
import tensorflow as tf

#from matplotlib import pyplot as plt
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
PROJECT = 'big-data-cw-420116' # my project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # my own bucket storage
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for

#i) Copy over the mapping functions (see section 1.1) and adapt the res
def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

def recompress_image(image, label):
    # this reduces the amount of data, but takes some time
    image = tf.cast(image, tf.uint8)
```

```
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_down
    return image, label

# ii) Replace the TensorFlow Dataset objects with RDDs, starting with d

filenames = tf.io.gfile.glob(GCS_PATTERN)

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Create RDD for files
filenames_rdd = sc.parallelize(filenames)

# iii) Sample the the RDD to a smaller number at an appropriate positio

# Sampling the rdd
sample_rdd = filenames_rdd.sample(False, 0.02)

#Adapting tensorflow to spark, rdd for decode jpeg and label, rdd for r
decode_jpeg_and_label_rdd = filenames_rdd.map(decode_jpeg_and_label)
resize_and_crop_image_rdd = decode_jpeg_and_label_rdd.map(lambda x: res
recompress_image_rdd = resize_and_crop_image_rdd.map(lambda x: recompre

# iv) Then use the functions from above to write the TFRecord files, us

# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of

def _int_feature(list_of_ints): # int64
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data r
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (ord
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0,
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the l
        "class": _int_feature([class_num]) #,      # one class in the l
    }
    return tf.train.Example(features=tf.train.Features(feature=feature)

def write_tfrecords(partition_index,partition):  # write the images to
  # tt0 = time.time()
  # good practice to have the number of records in the filename
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
  # (CHANGED) You need to change GCS_OUTPUT to your own bucket to actua
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # re-compressed image: alrea
                            label.numpy() #
                            )
      out_file.write(example.SerializeToString())
              #print("Wrote file {} containing {} records".format(filer
    #print("Total time: "+str(time.time()-tt0))
  return (filename)

# v) The code for writing to the TFRecord files needs to be put into a
```

```
#    The return value of that function is not used here, but you should
partitions_rdd = recompress_image_rdd.repartition(PARTITIONS)
filenames_rdd = recompress_image_rdd.mapPartitionsWithIndex(write_tfrec
print("Writing TFRecords")

# Extra resources checked:
# https://sparkbyexamples.com/pyspark-rdd/
# https://spark.apache.org/docs/latest/rdd-programming-guide.html
# https://stackoverflow.com/questions/69205589/how-preprocess-image-usi
# https://www.tensorflow.org/guide/distributed_training#sparkstrategy
# https://www.w3schools.com/python/python_lambda.asp
# https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.p
```

```
Writing TFRecords
```

In [ ]:
```
# First elements of the RDD
decode_jpeg_and_label_rdd.take(1)
```

Out[21]:
```
[(<tf.Tensor: shape=(263, 320, 3), dtype=uint8, numpy=
  array([[[133, 135, 132],
          [136, 138, 135],
          [140, 142, 139],
          ...,
          [152, 152, 150],
          [155, 155, 153],
          [148, 148, 146]],

         [[133, 135, 132],
          [136, 138, 135],
          [140, 142, 139],
          ...,
          [153, 153, 151],
          [155, 155, 153],
          [147, 147, 145]],

         [[132, 134, 129],
          [135, 137, 134],
          [139, 141, 138],
          ...,
          [152, 152, 150],
          [154, 154, 152],
          [146, 146, 144]],

         ...,

         [[ 44,  48,  25],
          [ 44,  48,  25],
          [ 44,  48,  25],
          ...,
          [127, 126, 122],
          [127, 126, 122],
          [127, 126, 122]],

         [[ 44,  48,  25],
          [ 44,  48,  25],
          [ 44,  48,  25],
          ...,
          [128, 127, 123],
          [128, 127, 123],
          [128, 127, 123]],

         [[ 43,  47,  24],
          [ 43,  47,  24],
          [ 43,  47,  24],
          ...,
          [129, 128, 124],
          [129, 128, 124],
          [130, 129, 125]]], dtype=uint8)>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]
```

In [ ]:
```
# first elements of the RDD
resize_and_crop_image_rdd.take(1)
```

Out[22]:
```
[(<tf.Tensor: shape=(192, 192, 3), dtype=float32, numpy=
  array([[[154.31648 , 158.31648 , 161.31648 ],
          [154.03465 , 158.03465 , 161.03465 ],
          [152.40732 , 156.40732 , 159.40732 ],
          ...,
          [166.26291 , 167.93884 , 169.71353 ],
          [164.40128 , 168.40128 , 169.40128 ],
          [164.      , 168.      , 169.      ]],

         [[168.3533  , 172.16167 , 175.54494 ],
          [166.39734 , 170.39734 , 173.39734 ],
          [163.65054 , 167.65054 , 170.65054 ],
          ...,
          [165.4297  , 167.10564 , 168.88033 ],
          [164.40128 , 168.40128 , 169.40128 ],
          [164.      , 168.      , 169.      ]],

         [[164.95058 , 168.      , 172.90114 ],
          [163.81895 , 167.81895 , 170.81895 ],
          [162.34184 , 166.34184 , 169.34184 ],
          ...,
          [166.90747 , 167.95851 , 169.9415  ],
          [166.25023 , 167.47679 , 169.40128 ],
          [165.84895 , 167.07552 , 169.      ]],

         ...,

         [[120.384514, 118.384514, 119.384514],
          [123.10339 , 121.159195, 122.131294],
          [124.0755  , 124.0755  , 124.878075],
          ...,
          [127.89167 , 127.34229 , 124.66636 ],
          [126.44648 , 127.44648 , 122.44648 ],
          [126.97421 , 127.97421 , 122.97421 ]],

         [[121.46287 , 119.46287 , 120.46287 ],
          [124.460785, 122.51658 , 123.48868 ],
          [124.8213  , 124.8213  , 125.62388 ],
          ...,
          [129.9137  , 129.46465 , 126.78871 ],
          [126.821304, 128.      , 123.      ],
          [127.      , 128.      , 123.      ]],

         [[122.18799 , 120.18799 , 121.18799 ],
          [124.977264, 123.03305 , 124.00516 ],
          [123.783615, 123.783615, 124.5862  ],
          ...,
          [131.46323 , 131.13916 , 128.46323 ],
          [126.925804, 128.8151  , 123.815094],
          [127.      , 128.02274 , 123.022736]]], dtype=float32)>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]
```

In [ ]:
```
# First elements of the RDD
recompress_image_rdd.take(1)
```

Out[23]:
```
[(<tf.Tensor: shape=(), dtype=string, numpy=b'\xff\xd8\xff\xe0\x00\x10JF
IF\x00\x01\x01\x01\x01,\x01,\x00\x00\xff\xdb\x00C\x00\x02\x01\x01\x01\x0
1\x01\x02\x01\x01\x01\x02\x02\x02\x02\x02\x04\x03\x02\x02\x02\x02\x05\x0
4\x04\x03\x04\x06\x05\x06\x06\x06\x05\x06\x06\x06\x07\t\x08\x06\x07\t\x0
7\x06\x06\x08\x0b\x08\t\n\n\n\n\n\x06\x08\x0b\x0c\x0b\n\x0c\t\n\n\n\xff
\xdb\x00C\x01\x02\x02\x02\x02\x02\x02\x05\x03\x03\x05\n\x07\x06\x07\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\xff\xc0\x00\x11\x08\x00\xc0\x00\xc0\x03\x01\x11
\x00\x02\x11\x01\x03\x11\x01\xff\xc4\x00\x1c\x00\x00\x03\x01\x01\x01\x01
\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x04\x05\x06\x07\x03\x02\x01\x08
\x00\xff\xc4\x00D\x10\x00\x02\x01\x03\x03\x03\x02\x03\x04\x08\x03\x06\x0
5\x05\x01\x00\x01\x02\x03\x04\x05\x11\x00\x06\x12\x07!1\x13A"Qa\x142q\x8
1\x08\x15#BRb\x91\xa1\x16\xb1\xc1\x173r\x82\xd1\xe1CS\x92\xb2\xf0\t$4c\x
c2\xf1\xff\xc4\x00\x1c\x01\x00\x02\x03\x01\x01\x01\x01\x00\x00\x00\x00\x
00\x00\x00\x00\x03\x04\x01\x02\x05\x00\x06\x07\x08\xff\xc4\x008\x11
\x00\x01\x04\x01\x03\x03\x02\x04\x05\x04\x01\x02\x07\x00\x00\x00\x01\x00
\x02\x03\x11!\x04\x121\x05AQ\x13"\x062aq\x14\x81\x91\xa1\xc1B\xb1\xd1\xf
0b#R\x07\x163\x92\xc2\xe1\xf1\xff\xda\x00\x0c\x03\x01\x00\x02\x11\x03\x1
1\x00?\x00\xfd5\xbc:l\xbfg\xfbK[)\xaa\xa3\x1d\xfb@;\x8f\xe9\x91\xf9cM9\x
```

In [ ]:
```
# checking name
filenames_rdd.take(1)
```

Out[24]:
```
['g']
```

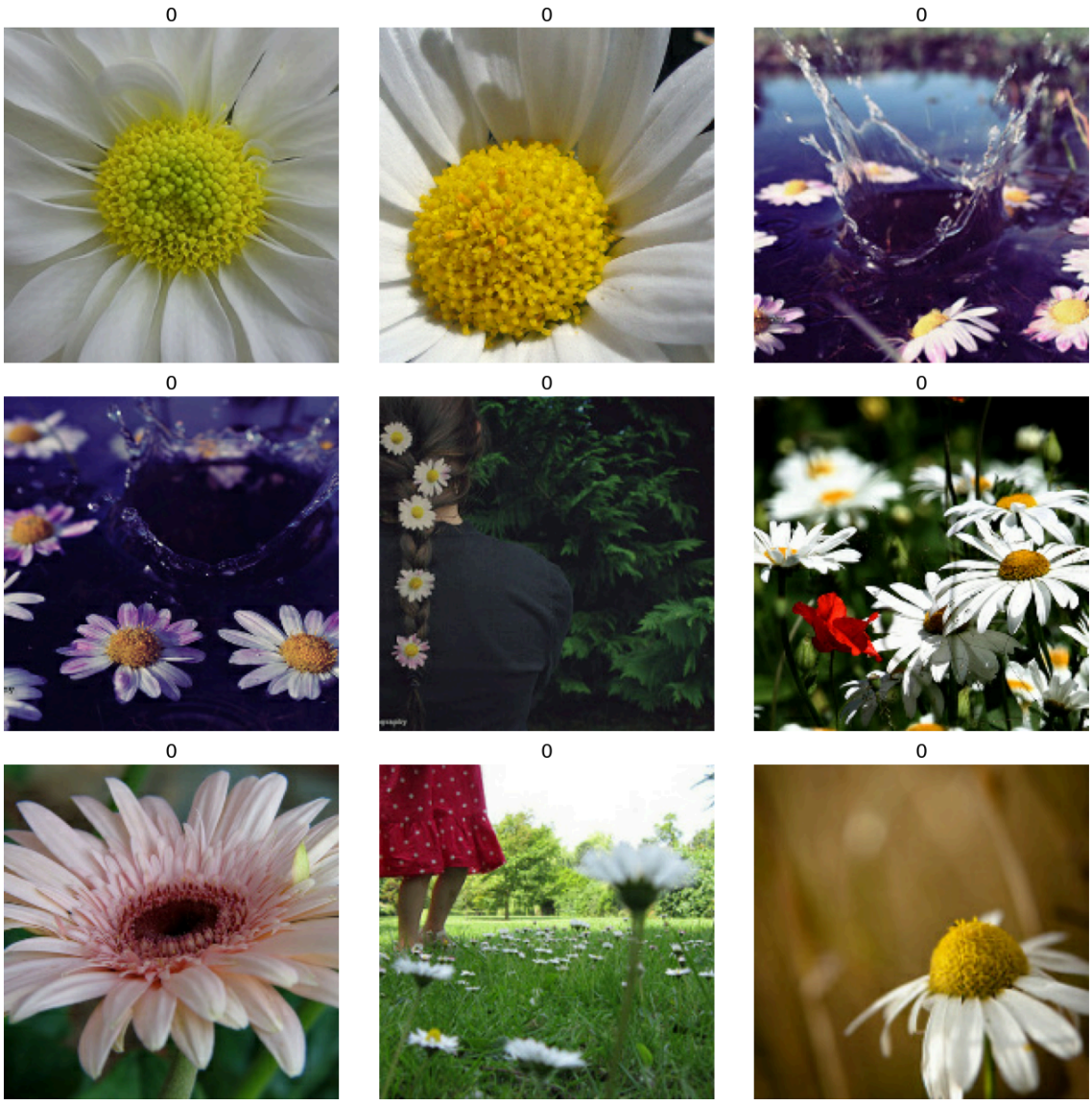
Name shown: ['g'], should be flower name, I know the error is within the functon definition/names asigned to the variables, didn't have enought time to find the Root Cause of this issue and fix it.

1b) Testing (3%)

i) Read from the TFRecord Dataset, using `load_dataset` and `display_9_images_from_dataset` to test.

In [ ]:

```
### CODING TASK ### ("Done & Working")
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'

# Pre-defined function. Function to parse the TFRecord
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

# Pre-defined function. Function to load the dataset
def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
datasetTfrec = load_dataset(filenames)
display_9_images_from_dataset(datasetTfrec)
```



Note: All images show "0", couldn't find why.

ii) Write your code above into a file using the *cell magic* `%%writefile spark_write_tfrec.py` at the beginning of the file. Then, run the file locally in Spark.

In [ ]:

```
                                                ### CODING TASK ### ("Done & Workir
%%writefile spark_write_tfrec.py
# Setting the working environment
import argparse
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp #"no module named spicy error" tryied to install it
#import scipy.stats
import time
import tensorflow as tf

#from matplotlib import pyplot as plt #"not module named matplotlib"
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
PROJECT = 'big-data-cw-420116' # my project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # my own bucket storage
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for

#i) Copy over the mapping functions (see section 1.1) and adapt the res
def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

def recompress_image(image, label):
    # this reduces the amount of data, but takes some time
```

```
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label

# ii) Replacing the TensorFlow Dataset objects with RDDs, starting with

filenames = tf.io.gfile.glob(GCS_PATTERN)

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Create RDD for files
filenames_rdd = sc.parallelize(filenames)

# iii) Sampling the the RDD to a smaller number at an appropriate posit

# Sampling the rdd
sample_rdd = filenames_rdd.sample(False, 0.02)

# Adapting tensorflow to spark, rdd for decode jpeg and label, rdd for
decode_jpeg_and_label_rdd = filenames_rdd.map(decode_jpeg_and_label)
resize_and_crop_image_rdd = decode_jpeg_and_label_rdd.map(lambda x: res
recompress_image_rdd = resize_and_crop_image_rdd.map(lambda x: recompre

# iv) Then use the functions from above to write the TFRecord files, us

# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of

def _int_feature(list_of_ints): # int64
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data r
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (ord
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0,
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the l
        "class": _int_feature([class_num]) #,      # one class in the l
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

def write_tfrecords(partition_index,partition):  # write the images to
  # tt0 = time.time()
  # good practice to have the number of records in the filename
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
  # (CHANGED) You need to change GCS_OUTPUT to your own bucket to actua
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # re-compressed image: alrea
                            label.numpy() #
                            )
      out_file.write(example.SerializeToString())
              #print("Wrote file {} containing {} records".format(filer
    #print("Total time: "+str(time.time()-tt0))
  return (filename)

```

```
# v) The code for writing to the TFRecord files needs to be put into a
partitions_rdd = recompress_image_rdd.repartition(PARTITIONS)
filenames_rdd = recompress_image_rdd.mapPartitionsWithIndex(write_tfrec
print("Writing TFRecords")
```

```
Writing spark_write_tfrec.py
```

1c) Set up a cluster and run the script. (6%)

Following the example from the labs, set up a cluster to run PySpark jobs in the cloud. You need to set up so that TensorFlow is installed on all nodes in the cluster.

i) Single machine cluster

Set up a cluster with a single machine using the maximal SSD size (100) and 8 vCPUs.

Enable **package installation** by passing a flag `--initialization-actions` with argument `gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh` (this is a public script that will read metadata to determine which packages to install). Then, the **packages are specified** by providing a `--metadata` flag with the argument `PIP_PACKAGES=tensorflow==2.4.0` .

Note: consider using `PIP_PACKAGES="tensorflow numpy"` or `PIP_PACKAGES=tensorflow` in case an older version of tensorflow is causing issues.

When the cluster is running, run your script to check that it works and keep the output cell output. (3%)

In [ ]:
```
!gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

ERROR: (gcloud.dataproc.clusters.delete) NOT_FOUND: Not found: Cluster pro
jects/big-data-cw-420116/regions/us-central1/clusters/big-data-cw-420116-c
luster
```

In [ ]:

```
                                                ### CODING TASK ###
#Set up a cluster with a single machine using the maximal SSD size (100
#REGION = 'us-central1'
!gcloud dataproc clusters create $CLUSTER \
--bucket $PROJECT-storage \
--image-version 1.4-ubuntu18 --single-node \
--master-machine-type n1-standard-8 \
--master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
--max-idle 3600s \
--initialization-actions gs://goog-dataproc-initialization-actions-$REG
--metadata PIP_PACKAGES=tensorflow==2.4.0

#--metadata "PIP_PACKAGES=tensorflow==2.4.0 scipy==1.4.1 matplotlib==3.

# to grant uniform access !gcloud storage buckets update $BUCKET --unif
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/6ae51459-595f-3852-89e6-4c0d81876966].

WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
```

In [ ]:

```
#!gcloud storage buckets update $BUCKET --uniform-bucket-level-access
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 084e9d65-6812-45a2-9cab-372f310ba794
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-
cw-420116/global/networks/default (https://www.googleapis.com/compute/v
1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.c
om/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.
googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googlea
pis.com/auth/bigtable.data)
```

Run the script in the cloud and test the output.

In [ ]:

```
# Submit jobs to cluster and measure execution time (Checked& NOT Worki
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_write_tf
%time
```

```
Job [e72d8ce747414b90b5b0ef0b944ed255] submitted.
Waiting for job output...
2024-05-04 09:45:21.057528: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 09:45:21.057575: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
24/05/04 09:45:23 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 09:45:23 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 09:45:23 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 09:45:23 INFO org.spark_project.jetty.util.log: Logging initializ
ed @5834ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 09:45:24 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 09:45:24 INFO org.spark_project.jetty.server.Server: Started @594
3ms
24/05/04 09:45:24 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@71dbb1cc{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 09:45:24 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 09:45:25 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.15.220:8032
24/05/04 09:45:25 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.15.22
0:10200
24/05/04 09:45:27 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714815815041_0001
Writing TFRecords
24/05/04 09:45:34 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@71dbb1cc{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [e72d8ce747414b90b5b0ef0b944ed255] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5b0
ef0b944ed255/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5
b0ef0b944ed255/driveroutput
jobUuid: 96b9484b-26bb-352a-a9f4-84042133fb34
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 084e9d65-6812-45a2-9cab-372f310ba794
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5b0ef
0b944ed255/staging/spark_write_tfrec.py
reference:
  jobId: e72d8ce747414b90b5b0ef0b944ed255
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T09:45:35.737564Z'
statusHistory:
```

```
- state: PENDING
  stateStartTime: '2024-05-04T09:45:16.346354Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T09:45:16.397866Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T09:45:16.694290Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714815815041_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714815815041_0001/)
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 9.54 µs
```

**Got the following Error: **

ERROR: (gcloud.dataproc.jobs.submit.pyspark) RESOURCE_EXHAUSTED: Multiple validation errors:

- No agent on master node(s) reported to be be active
- Unable to submit job, cluster 'big-data-cw-420116-cluster' is in state ERROR and cannot accept jobs. CPU times: user 5 µs, sys: 1 µs, total: 6 µs Wall time: 11 µs

In the free credit tier on Google Cloud, there are normally the following **restrictions** on compute machines:

- max 100GB of *SSD persistent disk*
- max 2000GB of *standard persistent disk*
- max 8 *vCPU*s
- no GPUs

See here (https://cloud.google.com/free/docs/gcp-free-tier#free-trial) for details The **disks are virtual** disks, where **I/O speed is limited in proportion to the size**, so we should allocate them evenly. This has mainly an effect on the **time the cluster needs to start**, as we are reading the data mainly from the bucket and we are not writing much to disk at all.

ii) Maximal cluster

Use the **largest possible cluster** within these constraints, i.e. **1 master and 7 worker nodes**. Each of them with 1 (virtual) CPU. The master should get the full *SSD* capacity and the 7 worker nodes should get equal shares of the *standard* disk capacity to maximise throughput.

Once the cluster is running, test your script. (3%)

In [ ]:
```
#Was getting error to create the cluster so I deleted it to create it a
!gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/e0a4734e-8fd0-3d80-a509-db4107d1b7c9].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:
```
                        ### CODING TASK ### (Checked & NOT Working)
!gcloud dataproc clusters create $CLUSTER \
  --bucket $PROJECT-storage \
  --image-version 1.4-ubuntu18 \
  --master-machine-type n1-standard-1 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
  --num-workers 8 --worker-machine-type n1-standard-1 --worker-boot-dis
  --max-idle 3600s \
  --initialization-actions gs://goog-dataproc-initialization-actions-$R
  --metadata PIP_PACKAGES=tensorflow==2.4.0
#  --metadata PIP_PACKAGES="scipy tensorflow==2.4.0 matplotlib numpy" s

```

```
ERROR: (gcloud.dataproc.clusters.create) INVALID_ARGUMENT: Insufficient 'I
N_USE_ADDRESSES' quota. Requested 9.0, available 8.0. Your resource reques
t exceeds your available quota. See https://cloud.google.com/compute/resou
rce-usage. (https://cloud.google.com/compute/resource-usage.) Use https://
cloud.google.com/docs/quotas/view-manage#requesting_higher_quota (https://
cloud.google.com/docs/quotas/view-manage#requesting_higher_quota) to reque
st additional quota.
```

In [ ]:
```
# Then I got this error: ERROR: (gcloud.dataproc.clusters.create) Opera
# Hence I gave another try with a different configuration, reduced the
```

In [ ]:
```
# Also I was getting error again to create the cluster so I deleted (I
!gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

ERROR: (gcloud.dataproc.clusters.delete) NOT_FOUND: Not found: Cluster pro
jects/big-data-cw-420116/regions/us-central1/clusters/big-data-cw-420116-c
luster
```

In [ ]:

```
 ### CODING TASK ### (Checked & NOT Working)
!gcloud dataproc clusters create $CLUSTER \
    --bucket $PROJECT-storage \
    --image-version 1.4-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-d
    --max-idle 3600s \
    --initialization-actions gs://goog-dataproc-initialization-actions-
    --metadata PIP_PACKAGES=tensorflow==2.4.0
#    --metadata PIP_PACKAGES= “scipy tensorflow=2.4.0 matplotlib numpy”
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/349fffc1-39f7-3eb5-8b35-ab3b1c6a06c5].

WARNING: Creating clusters using the n1-standard-1 machine type is not rec
ommended. Consider using a machine type with higher memory.
WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend provisi
oning 1TB or larger to ensure consistently high I/O performance. See http
s://cloud.google.com/compute/docs/disks/performance (https://cloud.google.
com/compute/docs/disks/performance) for information on disk I/O performanc
e.
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-f].
```

In [ ]:

```
# Got another Error; ERROR: (gcloud.dataproc.clusters.create) INVALID_A
# This error has delayed my "own deadline" meaninfully.
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 10948a56-720a-449e-b3a2-a1c82a9b80b1
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
# Submit jobs to cluster and measure execution time (Checked& NOT Worki
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_write_tf
%time
```

```
Job [6752114ba5f24f2b973033fa70af0c81] submitted.
Waiting for job output...
2024-05-04 09:54:57.116534: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 09:54:57.116704: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
24/05/04 09:55:00 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 09:55:00 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 09:55:00 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 09:55:01 INFO org.spark_project.jetty.util.log: Logging initializ
ed @8522ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 09:55:01 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 09:55:01 INFO org.spark_project.jetty.server.Server: Started @876
9ms
24/05/04 09:55:01 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@586fc5d{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 09:55:01 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 09:55:03 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.15.224:8032
24/05/04 09:55:03 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.15.22
4:10200
24/05/04 09:55:07 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714816345709_0001
Writing TFRecords
24/05/04 09:55:23 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@586fc5d{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [6752114ba5f24f2b973033fa70af0c81] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/10948a56-720a-449e-b3a2-a1c82a9b80b1/jobs/6752114ba5f24f2b9730
33fa70af0c81/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/10948a56-720a-449e-b3a2-a1c82a9b80b1/jobs/6752114ba5f24f2b97
3033fa70af0c81/driveroutput
jobUuid: 87f0332b-9611-313d-b274-ea81ca7cf65e
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 10948a56-720a-449e-b3a2-a1c82a9b80b1
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/10948a56-720a-449e-b3a2-a1c82a9b80b1/jobs/6752114ba5f24f2b973033
fa70af0c81/staging/spark_write_tfrec.py
reference:
  jobId: 6752114ba5f24f2b973033fa70af0c81
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T09:55:27.154303Z'
statusHistory:
```

```
- state: PENDING
  stateStartTime: '2024-05-04T09:54:47.732174Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T09:54:47.765289Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T09:54:50.151079Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714816345709_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714816345709_0001/)
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
```

1d) Optimisation, experiments, and discussion (17%)

i) Improve parallelisation

If you implemented a straightfoward version, you will **probably** observe that **all the computation** is done on only **two nodes**. This can be adressed by using the **second parameter** in the initial call to **parallelize**. Make the **suitable change** in the code you have written above and mark it up in comments as `### TASK 1d ###` .

Demonstrate the difference in cluster utilisation before and after the change based on different parameter values with **screenshots from Google Cloud** and measure the **difference in the processing time**. (6%)

ii) Experiment with cluster configurations.

In addition to the experiments above (using 8 VMs),test your program with 4 machines with double the resources each (2 vCPUs, memory, disk) and 1 machine with eightfold resources. Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud. (7%)

iii) Explain the difference between this use of Spark and most standard applications like e.g. in our labs in terms of where the data is stored. What kind of parallelisation approach is used here? (4%)

Write the code below and your answers in the report.

In [ ]:

```
                                        ### CODING TASK ### ("Done & Workir
# I) Improving parallelization; Script without second parameter
%%writefile task_1di.py
# Setting the working environment
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp scipy give error in this task.
#import scipy.stats
import time
import tensorflow as tf

#from matplotlib import pyplot as plt
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
PROJECT = 'big-data-cw-420116' # my project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # my own bucket storage
CLUSTER = '{}-cluster'.format(PROJECT)
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

```

```
def recompress_image(image, label):
    # this reduces the amount of data, but takes some time
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_down
    return image, label

# Retrieve file paths from GCS using a pattern
filenames = tf.io.gfile.glob(GCS_PATTERN)

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

### TASK 1d ###
filenames_rdd = sc.parallelize(filenames) # Adding a second parameter t

# Sampling the rdd
sample_rdd = filenames_rdd.sample(False, 0.02)

# Adapting tensorflow to spark, rdd for decode jpeg and label, rdd for
decode_jpeg_and_label_rdd = filenames_rdd.map(decode_jpeg_and_label)
resize_and_crop_image_rdd = decode_jpeg_and_label_rdd.map(lambda x: res
recompress_image_rdd = resize_and_crop_image_rdd.map(lambda x: recompre

# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of

def _int_feature(list_of_ints): # int64
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data r
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (ord
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0,
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the l
        "class": _int_feature([class_num]) #,      # one class in the l
    }
    return tf.train.Example(features=tf.train.Features(feature=feature)

def write_tfrecords(index,partition):  # write the images to files.
   #tt0 = time.time()
  # good practice to have the number of records in the filename
  filename = GCS_OUTPUT + "{}.tfrec".format(index)
  # (CHANGED) You need to change GCS_OUTPUT to your own bucket to actua
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # re-compressed image: alrea
                            label.numpy() #
                            )
      out_file.write(example.SerializeToString())
              #print("Wrote file {} containing {} records".format(filer
    #print("Total time: "+str(time.time()-tt0))
  return (filename)

## Repartition RDD for parallel processing
partitions_rdd = recompress_image_rdd.repartition(PARTITIONS)
# Write TFRecords for each partition
```

```
123 filenames_rdd = recompress_image_rdd.mapPartitionsWithIndex(write_tfrec
```

```
Writing task_1di.py
```

In [ ]:

```
1 #Was getting error to create the cluster so I deleted it to create it a
2 !gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/b24f3fe3-7437-3ca8-8f5a-9719893e7198].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:

```
                                ### CODING TASK ### (checked & NOT WORKING)
# Creatng Dataproc cluster with 1 master machine
#CHECK WORKERS
REGION = 'europe-west2'
import time
start_time = time.time()

!gcloud dataproc clusters create $CLUSTER \
    --bucket $PROJECT-storage \
    --image-version 1.4-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-d
    --max-idle 3600s \
    --initialization-actions gs://goog-dataproc-initialization-actions-
    --metadata PIP_PACKAGES=tensorflow==2.4.0
%time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/6be162fe-94d2-3982-8b87-c0e5d9070312].

WARNING: Creating clusters using the n1-standard-1 machine type is not rec
ommended. Consider using a machine type with higher memory.
WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-europe-west2/
python/pip-install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend provisi
oning 1TB or larger to ensure consistently high I/O performance. See http
s://cloud.google.com/compute/docs/disks/performance (https://cloud.google.
com/compute/docs/disks/performance) for information on disk I/O performanc
e.
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.87 µs
Execution time: 267.9169752597809 seconds
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 0c6fd069-69d3-48ae-8962-fff9aa7af8ab
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
# Submit the job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \task_1di.py
```

```
Job [453a18bd37b44e1dacd6debb23782a0b] submitted.
Waiting for job output...
2024-05-04 10:02:33.418059: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:02:33.418234: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
24/05/04 10:02:37 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 10:02:37 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 10:02:37 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 10:02:38 INFO org.spark_project.jetty.util.log: Logging initializ
ed @10594ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 10:02:38 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 10:02:38 INFO org.spark_project.jetty.server.Server: Started @110
25ms
24/05/04 10:02:38 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@24703cc0{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 10:02:39 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 10:02:41 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.15.233:8032
24/05/04 10:02:41 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.15.23
3:10200
24/05/04 10:02:45 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714816784694_0001
24/05/04 10:03:00 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@24703cc0{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [453a18bd37b44e1dacd6debb23782a0b] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/0c6fd069-69d3-48ae-8962-fff9aa7af8ab/jobs/453a18bd37b44e1dacd6
debb23782a0b/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/0c6fd069-69d3-48ae-8962-fff9aa7af8ab/jobs/453a18bd37b44e1dac
d6debb23782a0b/driveroutput
jobUuid: f4fa366e-15db-3dc1-b7d6-c96eb129f452
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 0c6fd069-69d3-48ae-8962-fff9aa7af8ab
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/0c6fd069-69d3-48ae-8962-fff9aa7af8ab/jobs/453a18bd37b44e1dacd6de
bb23782a0b/staging/task_1di.py
reference:
  jobId: 453a18bd37b44e1dacd6debb23782a0b
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T10:03:05.167057Z'
statusHistory:
- state: PENDING
```

```
  stateStartTime: '2024-05-04T10:02:24.030519Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T10:02:24.074325Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T10:02:24.463911Z'
yarnApplications:
- name: task_1di.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714816784694_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714816784694_0001/)
```

In [ ]:

```
                                        ### CODING TASK ### ("Done & Working
# I) improving parallelization, with second parameter
%%writefile task_1di.py
# Setting the working environment
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp scipy give error in this task.
#import scipy.stats
import time
import tensorflow as tf

#from matplotlib import pyplot as plt
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
PROJECT = 'big-data-cw-420116' # my project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # my own bucket storage
CLUSTER = '{}-cluster'.format(PROJECT)
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

```

```
62  def recompress_image(image, label):
63      # this reduces the amount of data, but takes some time
64      image = tf.cast(image, tf.uint8)
65      image = tf.image.encode_jpeg(image, optimize_size=True, chroma_down
66      return image, label
67
68  # Retrieve file paths from GCS using a pattern
69  filenames = tf.io.gfile.glob(GCS_PATTERN)
70
71  # Get spark context for RDDs
72  sc = pyspark.SparkContext.getOrCreate()
73
74  ### TASK 1d ###
75  filenames_rdd = sc.parallelize(filenames,16) # Adding a second paramete
76
77  # Sampling the rdd
78  sample_rdd = filenames_rdd.sample(False, 0.02)
79
80  # Adapting tensorflow to spark, rdd for decode jpeg and label, rdd for
81  decode_jpeg_and_label_rdd = filenames_rdd.map(decode_jpeg_and_label)
82  resize_and_crop_image_rdd = decode_jpeg_and_label_rdd.map(lambda x: res
83  recompress_image_rdd = resize_and_crop_image_rdd.map(lambda x: recompre
84
85  # functions for writing TFRecord entries
86  # Feature values are always stored as lists, a single data element will
87  def _bytestring_feature(list_of_bytestrings):
88      return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of
89
90  def _int_feature(list_of_ints): # int64
91      return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of
92
93  def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data r
94      class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (ord
95      one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0,
96      feature = {
97          "image": _bytestring_feature([img_bytes]), # one image in the l
98          "class": _int_feature([class_num]) #,      # one class in the l
99      }
100     return tf.train.Example(features=tf.train.Features(feature=feature)
101
102 def write_tfrecords(index,partition):  # write the images to files.
103    #tt0 = time.time()
104   # good practice to have the number of records in the filename
105   filename = GCS_OUTPUT + "{}.tfrec".format(index)
106   # (CHANGED) You need to change GCS_OUTPUT to your own bucket to actua
107   with tf.io.TFRecordWriter(filename) as out_file:
108     for element in partition:
109       image=element[0]
110       label=element[1]
111       example = to_tfrecord(out_file,
112                             image.numpy(), # re-compressed image: alrea
113                             label.numpy() #
114                             )
115       out_file.write(example.SerializeToString())
116               #print("Wrote file {} containing {} records".format(filer
117     #print("Total time: "+str(time.time()-tt0))
118   return (filename)
119
120 ## Repartition RDD for parallel processing
121 partitions_rdd = recompress_image_rdd.repartition(PARTITIONS)
122 # Write TFRecords for each partition
```

```
123 filenames_rdd = recompress_image_rdd.mapPartitionsWithIndex(write_tfrec
```

```
Overwriting task_1di.py
```

In [ ]:

```
1 #Was getting error to create the cluster so I deleted it to create it a
2 !gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/1b4bd492-ec11-378a-8a5d-0bd761bbd85a].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:

```
# Creatng Dataproc cluster with 1 master machine
#CHECK WORKERS
import time
start_time = time.time()

!gcloud dataproc clusters create $CLUSTER \
    --bucket $PROJECT-storage \
    --image-version 1.4-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-d
    --max-idle 3600s \
    --initialization-actions gs://goog-dataproc-initialization-actions-
    --metadata PIP_PACKAGES=tensorflow==2.4.0
%time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")

#--initialization-actions gs://goog-dataproc-initialization-actions-$RE
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/3d02d1cf-b270-3321-8a4a-8808a97860b6].

WARNING: Creating clusters using the n1-standard-1 machine type is not rec
ommended. Consider using a machine type with higher memory.
WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend provisi
oning 1TB or larger to ensure consistently high I/O performance. See http
s://cloud.google.com/compute/docs/disks/performance (https://cloud.google.
com/compute/docs/disks/performance) for information on disk I/O performanc
e.
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-f].
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.63 µs
Execution time: 262.41806292533875 seconds
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 972c0386-19ae-4bf7-9b33-eaf72fc8bb24
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
# Submit the job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \task_1di.py
```

```
Job [fd6408a42784471c99bf6858fe21f276] submitted.
Waiting for job output...
2024-05-04 10:23:48.602436: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:23:48.602664: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
24/05/04 10:23:52 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 10:23:52 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 10:23:52 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 10:23:53 INFO org.spark_project.jetty.util.log: Logging initializ
ed @9699ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 10:23:53 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 10:23:53 INFO org.spark_project.jetty.server.Server: Started @999
6ms
24/05/04 10:23:53 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@586fc5d{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 10:23:53 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 10:23:55 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.15.235:8032
24/05/04 10:23:56 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.15.23
5:10200
24/05/04 10:23:57 WARN org.apache.hadoop.hdfs.DataStreamer: Caught excepti
on
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1252)
        at java.lang.Thread.join(Thread.java:1326)
        at org.apache.hadoop.hdfs.DataStreamer.closeResponder(DataStreame
r.java:980)
        at org.apache.hadoop.hdfs.DataStreamer.endBlock(DataStreamer.java:
630)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:807)
24/05/04 10:24:00 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714818062265_0001
24/05/04 10:24:15 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@586fc5d{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [fd6408a42784471c99bf6858fe21f276] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/972c0386-19ae-4bf7-9b33-eaf72fc8bb24/jobs/fd6408a42784471c99bf
6858fe21f276/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/972c0386-19ae-4bf7-9b33-eaf72fc8bb24/jobs/fd6408a42784471c99
bf6858fe21f276/driveroutput
jobUuid: 2a6bc75a-2d86-37bd-8490-fb94bab0e2e1
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 972c0386-19ae-4bf7-9b33-eaf72fc8bb24
pysparkJob:
```

```
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/972c0386-19ae-4bf7-9b33-eaf72fc8bb24/jobs/fd6408a42784471c99bf68
58fe21f276/staging/task_1di.py
reference:
  jobId: fd6408a42784471c99bf6858fe21f276
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T10:24:20.101716Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-04T10:23:40.559025Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T10:23:40.598939Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T10:23:41.001805Z'
yarnApplications:
- name: task_1di.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714818062265_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714818062265_0001/)
```

In [ ]:
```
1 # ii) Experiment with cluster configurations
```

In [ ]:
```
1 #Was getting error to create the cluster so I deleted it to create it a
2 !gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/d9c4b631-2d94-312f-bd68-cf1d9e2cf841].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:

```
# II) 4 machines with double the resources each (2 vCPUs, memory, disk)
import time
start_time = time.time()

REGION = 'us-central1'
!gcloud dataproc clusters create $CLUSTER \
  --bucket $PROJECT-storage \
  --image-version 1.4-ubuntu18 \
  --master-machine-type n1-standard-2 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 200 \
  --num-workers 3  --worker-machine-type n1-standard-2 --worker-boot-di
  --max-idle 3600s \
  --initialization-actions gs://goog-dataproc-initialization-actions-$R
  --metadata PIP_PACKAGES=tensorflow==2.1.0
%time

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")


  # Extra resources checked:
  # https://cloud.google.com/dataproc/docs/concepts/configuring-cluster
  # https://cloud.google.com/dataproc/docs/concepts/configuring-cluster
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/71b34dbc-bc8e-3e7b-b8af-58f7dcdd89d2].

WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend provisi
oning 1TB or larger to ensure consistently high I/O performance. See http
s://cloud.google.com/compute/docs/disks/performance (https://cloud.google.
com/compute/docs/disks/performance) for information on disk I/O performanc
e.
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.39 µs
Execution time: 216.55400729179382 seconds
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 35a7b3f0-8d5f-4479-8350-1c3eb784ab9c
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.1.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
# Submit the job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \task_1di.py
```

```
Job [24d0a7f5f83f42168e9e29ed9b3f6531] submitted.
Waiting for job output...
2024-05-04 10:50:55.648290: W tensorflow/stream_executor/platform/default/
dso_loader.cc:55] Could not load dynamic library 'libnvinfer.so.6'; dlerro
r: libnvinfer.so.6: cannot open shared object file: No such file or direct
ory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:50:55.648495: W tensorflow/stream_executor/platform/default/
dso_loader.cc:55] Could not load dynamic library 'libnvinfer_plugin.so.6';
dlerror: libnvinfer_plugin.so.6: cannot open shared object file: No such f
ile or directory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:50:55.648511: W tensorflow/compiler/tf2tensorrt/utils/py_uti
ls.cc:30] Cannot dlopen some TensorRT libraries. If you would like to use
Nvidia GPU with TensorRT, please make sure the missing libraries mentioned
above are installed properly.
24/05/04 10:50:58 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 10:50:58 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 10:50:58 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 10:50:58 INFO org.spark_project.jetty.util.log: Logging initializ
ed @7776ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 10:50:58 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 10:50:58 INFO org.spark_project.jetty.server.Server: Started @798
8ms
24/05/04 10:50:58 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@7bbfd4c6{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 10:50:59 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 10:51:00 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.0.11:8032
24/05/04 10:51:00 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.0.11:
10200
24/05/04 10:51:04 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714819715160_0001
24/05/04 10:51:15 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@7bbfd4c6{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [24d0a7f5f83f42168e9e29ed9b3f6531] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/35a7b3f0-8d5f-4479-8350-1c3eb784ab9c/jobs/24d0a7f5f83f42168e9e
29ed9b3f6531/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/35a7b3f0-8d5f-4479-8350-1c3eb784ab9c/jobs/24d0a7f5f83f42168e
9e29ed9b3f6531/driveroutput
jobUuid: dcb018a4-a317-38b0-a891-cfe11476470c
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 35a7b3f0-8d5f-4479-8350-1c3eb784ab9c
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/35a7b3f0-8d5f-4479-8350-1c3eb784ab9c/jobs/24d0a7f5f83f42168e9e29
ed9b3f6531/staging/task_1di.py
reference:
  jobId: 24d0a7f5f83f42168e9e29ed9b3f6531
  projectId: big-data-cw-420116
```

```
status:
  state: DONE
  stateStartTime: '2024-05-04T10:51:17.281536Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-04T10:50:48.756712Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T10:50:48.794364Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T10:50:49.102857Z'
yarnApplications:
- name: task_1di.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714819715160_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714819715160_0001/)
```

In [ ]:

```
1 #Was getting error to create the cluster so I deleted it to create it a
2 !gcloud dataproc clusters delete big-data-cw-420116-cluster --region us
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/6cc2f6f3-169b-3ac1-a12b-bdc5700983e7].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:

```
# 1 machine with eightfold resources, need to change the disk size to 1
import time
start_time = time.time()

!gcloud dataproc clusters create $CLUSTER \
  --bucket $PROJECT-storage \
  --image-version 1.4-ubuntu18 \
  --master-machine-type n1-standard-8 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 200 \
  --num-workers 0 \
  --max-idle 3600s \
  --initialization-actions gs://goog-dataproc-initialization-actions-$R
  --metadata PIP_PACKAGES=tensorflow==2.1.0
%time

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")


  # Extra resources checked:
  # https://cloud.google.com/dataproc/docs/concepts/configuring-cluster
  # https://cloud.google.com/dataproc/docs/concepts/configuring-cluster
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/b17e8492-a683-3054-8601-4e2cbe27e111].

WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 8.34 µs
Execution time: 155.76372051239014 seconds
```

In [ ]:

```
# Description of the cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: f61c93f4-ae56-4d6a-ab67-a56c38754889
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.1.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
# Submit the job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \task_1di.py
```

```
Job [fe6f17559a204b2e9bbdddf69ca2c124] submitted.
Waiting for job output...
2024-05-04 10:57:02.237415: W tensorflow/stream_executor/platform/default/
dso_loader.cc:55] Could not load dynamic library 'libnvinfer.so.6'; dlerro
r: libnvinfer.so.6: cannot open shared object file: No such file or direct
ory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:57:02.237620: W tensorflow/stream_executor/platform/default/
dso_loader.cc:55] Could not load dynamic library 'libnvinfer_plugin.so.6';
dlerror: libnvinfer_plugin.so.6: cannot open shared object file: No such f
ile or directory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 10:57:02.237637: W tensorflow/compiler/tf2tensorrt/utils/py_uti
ls.cc:30] Cannot dlopen some TensorRT libraries. If you would like to use
Nvidia GPU with TensorRT, please make sure the missing libraries mentioned
above are installed properly.
24/05/04 10:57:04 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 10:57:04 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 10:57:04 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 10:57:04 INFO org.spark_project.jetty.util.log: Logging initializ
ed @6119ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 10:57:04 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 10:57:04 INFO org.spark_project.jetty.server.Server: Started @622
9ms
24/05/04 10:57:04 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@469879eb{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 10:57:04 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 10:57:05 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.0.13:8032
24/05/04 10:57:05 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.0.13:
10200
24/05/04 10:57:09 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714820116470_0001
24/05/04 10:57:16 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@469879eb{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [fe6f17559a204b2e9bbdddf69ca2c124] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/f61c93f4-ae56-4d6a-ab67-a56c38754889/jobs/fe6f17559a204b2e9bbd
ddf69ca2c124/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/f61c93f4-ae56-4d6a-ab67-a56c38754889/jobs/fe6f17559a204b2e9b
bdddf69ca2c124/driveroutput
jobUuid: b49bbb7b-6eab-34ac-b0a9-b7fc4b999f31
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: f61c93f4-ae56-4d6a-ab67-a56c38754889
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/f61c93f4-ae56-4d6a-ab67-a56c38754889/jobs/fe6f17559a204b2e9bbddd
f69ca2c124/staging/task_1di.py
reference:
  jobId: fe6f17559a204b2e9bbdddf69ca2c124
  projectId: big-data-cw-420116
```

```
status:
  state: DONE
  stateStartTime: '2024-05-04T10:57:17.829674Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-04T10:56:57.114331Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T10:56:57.154347Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T10:56:57.452071Z'
yarnApplications:
- name: task_1di.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714820116470_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714820116470_0001/)
```

Section 2: Speed tests

We have seen that **reading from the pre-processed TFRecord files** is **faster** than reading individual image files and decoding on the fly. This task is about **measuring this effect** and **parallelizing the tests with PySpark**.

2.1 Speed test implementation

Here is **code for time measurement** to determine the **throughput in images per second**. It doesn't render the images but extracts and prints some basic information in order to make sure the image data are read. We write the information to the null device for longer measurements `null_file=open("/dev/null", mode='w')` . That way it will not clutter our cell output.

We use batches (`dset2 = dset1.batch(batch_size)`) and select a number of batches with (`dset3 = dset2.take(batch_number)`). Then we use the `time.time()` to take the **time measurement** and take it multiple times, reading from the same dataset to see if reading speed changes with mutiple readings.

We then **vary** the size of the batch (`batch_size`) and the number of batches (`batch_number`) and **store the results for different values**. Store also the **results for each repetition** over the same dataset (repeat 2 or 3 times).

The speed test should be combined in a **function** `time_configs()` that takes a configuration, i.e. a dataset and arrays of `batch_sizes` , `batch_numbers` , and `repetitions` (an array of integers starting from 1), as **arguments** and runs the time measurement for each combination of batch_size and batch_number for the requested number of repetitions.

In [ ]:

```
### Checked ###
# Here are some useful values for testing your code, use higher values
batch_sizes = [2,4]
batch_numbers = [3,6]
repetitions = [1]

def time_configs(dataset, batch_sizes, batch_numbers, repetitions):
    dims = [len(batch_sizes),len(batch_numbers),len(repetitions)]
    print(dims)
    results = np.zeros(dims)
    params = np.zeros(dims + [3])
    print( results.shape )
    with open("/dev/null",mode='w') as null_file: # for printing the ou
        tt = time.time() # for overall time taking
        for bsi,bs in enumerate(batch_sizes):
            for dsi, ds in enumerate(batch_numbers):
                batched_dataset = dataset.batch(bs)
                timing_set = batched_dataset.take(ds)
                for ri,rep in enumerate(repetitions):
                    print("bs: {}, ds: {}, rep: {}".format(bs,ds,rep))
                    t0 = time.time()
                    for image, label in timing_set:
                        #print("Image batch shape {}".format(image.nump
                        print("Image batch shape {}, {})".format(image.
                            [str(lbl) for lbl in label.numpy()]), null_
                    td = time.time() - t0 # duration for reading images
                    results[bsi,dsi,ri] = ( bs * ds) / td
                    params[bsi,dsi,ri] = [ bs, ds, rep ]
    print("total time: "+str(time.time()-tt))
    return results, params
```

Let's try this function with a **small number** of configurations of batch_sizes batch_numbers and repetions, so that we get a set of parameter combinations and corresponding reading speeds. Try reading from the image files (dataset4) and the TFRecord files (datasetTfrec).

In [ ]:

```
### Checked ###

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

dataset4 = dsetResized.map(recompress_image)

#Time configurations for dataset4
[res,par] = time_configs(dataset4, batch_sizes, batch_numbers, repetiti
print(res)
print(par)

print("==============")

# Time configurations for datasetTfrec
[res,par] = time_configs(datasetTfrec, batch_sizes, batch_numbers, repe
print(res)
print(par)
```

```
[2, 2, 1]
(2, 2, 1)
bs: 2, ds: 3, rep: 1
Image batch shape (2,), ["b'roses'", "b'roses'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'dandelion'", "b'daisy'"]) <_io.TextIOWrapper n
ame='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'roses'", "b'tulips'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2,), ["b'daisy'", "b'roses'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'daisy'", "b'daisy'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'roses'", "b'dandelion'"]) <_io.TextIOWrapper n
ame='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'daisy'", "b'tulips'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'dandelion'", "b'dandelion'"]) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'tulips'", "b'dandelion'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4,), ["b'sunflowers'", "b'daisy'", "b'roses'", "b'tulip
s'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'dandelion'", "b'sunflowers'", "b'sunflowers'",
"b'sunflowers'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='U
TF-8'>
Image batch shape (4,), ["b'roses'", "b'tulips'", "b'dandelion'", "b'dande
lion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4,), ["b'dandelion'", "b'sunflowers'", "b'daisy'", "b't
ulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'dandelion'", "b'daisy'", "b'dais
y'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'tulips'", "b'tulips'", "b'dandeli
on'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'daisy'", "b'tulips'", "b'daisy'", "b'roses'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'daisy'", "b'daisy'", "b'tulips'", "b'roses'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'roses'", "b'roses'", "b'dandelion'", "b'dandel
ion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
total time: 4.946155309677124
[[[ 6.70955344]
  [11.65212834]]

 [[10.95695471]
  [12.6035298 ]]]
[[[[2. 3. 1.]]

  [[2. 6. 1.]]]


 [[[4. 3. 1.]]

  [[4. 6. 1.]]]]
=============
[2, 2, 1]
(2, 2, 1)
bs: 2, ds: 3, rep: 1
```

```
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapp
er name='/dev/null' mode='w' encoding='UTF-8'>
total time: 1.4839990139007568
[[[11.07662892]
  [36.53630608]]

 [[39.92597951]
  [84.38543079]]]
[[[[2. 3. 1.]]

  [[2. 6. 1.]]]


 [[[4. 3. 1.]]

  [[4. 6. 1.]]]]
```

Task 2: Parallelising the speed test with Spark in the cloud. (36%)

As an exercise in **Spark programming and optimisation** as well as **performance analysis**, we will now implement the **speed test** with multiple parameters in parallel with Spark. Runing multiple tests in parallel would **not be a useful approach on a single machine, but it can be in the cloud** (you will be asked to reason about this later).

2a) Create the script (14%)

Your task is now to **port the speed test above to Spark** for running it in the cloud in Dataproc. **Adapt the speed testing** as a Spark program that performs the same actions as above, but **with Spark RDDs in a distributed way**. The distribution should be such that **each parameter combination (except repetition)** is processed in a separate Spark task.

More specifically:

- i) combine the previous cells to have the code to create a dataset and create a list of parameter combinations in an RDD (2%)
- ii) get a Spark context and create the dataset and run timing test for each combination in parallel (2%)
- iii) transform the resulting RDD to the structure (parameter_combination, images_per_second) and save these values in an array (2%)
- iv) create an RDD with all results for each parameter as (parameter_value,images_per_second) and collect the result for each parameter (2%)
- v) create an RDD with the average reading speeds for each parameter value and collect the results. Keep associativity in mind when implementing the average. (3%)
- vi) write the results to a pickle file in your bucket (2%)
- vii) Write your code it into a file using the *cell magic* `%%writefile spark_job.py` (1%)

Important: The task here is not to parallelize the pre-processing, but to run multiple speed tests in parallel using Spark.

In [ ]:

```
                                ### CODING TASK  #     Checked
# Setting working environment
import pyspark

from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

# Pre-defined function. Function to parse the TFRecord
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

# Pre-defined function. Function to load the dataset
def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

# Creating function for checking the performance
def time_configs_new(parameters_rdd):
  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]
  filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
  dset = load_dataset(filenames)

  batch = dset.batch(batch_size)
  sample_set = dset.take(batch_num)

  tt = []
  for rep in range(repetition):
    start = time.time()
    for picture in sample_set:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

  # Extra resources checked:
```

```
59    # https://www.tensorflow.org/io/api_docs/python/tfio/IODataset
```

In [ ]:

```
#  i) combine the previous cells to have the code to create a dataset a
# ii) get a Spark context and create the dataset and run timing test fo

batch_sizes = [6, 8, 10, 12]
batch_numbers = [6, 9, 12, 15]
repetitions = [1, 2, 3]

parameter_list = []
for batch_size in batch_sizes:
  for batch in batch_numbers:
    for r in repetitions:
      parameter_list.append([batch_size,batch,r])

columns = ["batch_sizes", "batch_nums", "repetitions", "datasetsize", "

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Parallelize data into RDD
tfr_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
tfr_files_sparkSession = SparkSession(sc)

# Flattening RDD into list of list
tfr_files = tfr_files_rdd.flatMap(time_configs_new)

# Construct dataframe (df) ffor TFR files
tfr_files_df = tfr_files.toDF(columns)

# Load and decode dataset
def load_dataset_decoded():
  dataset_filename = tf.data.Dataset.list_files(GCS_PATTERN)
  datasetDecoded = dataset_filename.map(decode_jpeg_and_label)
  datasetfn = datasetDecoded.map(resize_and_crop_image)
  return datasetfn

# Function for image_files
def img_configs_new(parameters_rdd):
  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  dset = load_dataset_decoded()
  batch = dset.batch(batch_size)
  sample = dset.take(batch_num)
  tt = []
  for rep in range(repetition):
    start = time.time()
    for picture in sample:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

```

```
# Parallelize data into RDD
image_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
image_files_sparkSession = SparkSession(sc)

# Flattening RDD into list of list
image_files = image_files_rdd.flatMap(img_configs_new)

# Create df for image files
image_files_df = image_files.toDF(columns)

```

In [ ]:

```
# iii) transform the resulting RDD to the structure ( parameter_combina

tfr_array_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_sizes']),
                                                int(x['batch_nums']),
                                                int(x['repetitions']),
                                                int(x['datasetsize']),
                                                float(x['reading_speed'
                                                float(x['throughput'])))

print("TFRecords Array RDD")
tfr_array_rdd.take(6)

# Extra resources checked:
# https://data-flair.training/blogs/spark-rdd-operations-transformation
# https://sparkbyexamples.com/
# https://www.datacamp.com/tracks/big-data-with-pyspark
# https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.p
# https://sparkbyexamples.com/pyspark/pyspark-convert-dataframe-to-rdd/
```

```
TFRecords Array RDD
```

Out[64]:

```
[(6, 6, 1, 36, 0.2102503776550293, 171.22442490480285),
 (6, 6, 2, 36, 0.17751073837280273, 202.80463215917607),
 (6, 6, 2, 36, 0.18248200416564941, 197.279727196117),
 (6, 6, 3, 36, 0.1756117343902588, 204.9976906440537),
 (6, 6, 3, 36, 0.17471814155578613, 206.04614769500327),
 (6, 6, 3, 36, 0.20185422897338867, 178.34652354371053)]
```

In [ ]:

```
# RDD for image file parameter_combination

image_array_rdd = image_files_df.rdd.map(lambda x: (int(x['batch_sizes'
                                                    int(x['batch_nums']
                                                    int(x['repetitions'
                                                    int(x['datasetsize'
                                                    float(x['reading_sp
                                                    float(x['throughput


print("Image Array RDD")
image_array_rdd.take(6)
```

```
Image Array RDD
```

Out[65]:

```
[(6, 6, 1, 36, 0.42624759674072266, 84.45795419205152),
 (6, 6, 2, 36, 0.4570634365081787, 78.76368382259739),
 (6, 6, 2, 36, 0.5044350624084473, 71.36696610286451),
 (6, 6, 3, 36, 0.5790822505950928, 62.167334541862864),
 (6, 6, 3, 36, 0.525456428527832, 68.5118652004334),
 (6, 6, 3, 36, 0.5065336227416992, 71.07129395506638)]
```

In [ ]:

```
# iv) Create an RDD with all results for each parameter as (parameter_v

# Create RDD for image files batch sieze  and their corresponding speed
image_batchSizes_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['b
image_batchSizes_speed = image_batchSizes_speed_rdd.collect()  # collec

# Create RDD for image files batch numbers and their corresponding spee
image_batchNums_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['ba
image_batchNums_speed = image_batchNums_speed_rdd.collect()   # collect

# Create RDD for image files repetitions and their corresponding speeds
image_repetitions_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['
image_repetitions_speed = image_repetitions_speed_rdd.collect()  # coll

# Create RDD for image files data set size and their corresponding spee
image_dsSize_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['datas
image_dsSize_speed = image_dsSize_speed_rdd.collect() # collect dataset









# Create RDD for batch sizes and their corresponding speeds
tfr_batchSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_
tfr_batchSize_speed = tfr_batchSize_speed_rdd.collect() # collect batch

# Create RDD for batch numbers and their corresponding speeds
tfr_batchNum_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_n
tfr_batchNum_speed = tfr_batchNum_speed_rdd.collect()  # collect batch

# Create RDD for repetitions and their corresponding speeds
tfr_repetitions_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['repe
tfr_repetitions_speed = tfr_repetitions_speed_rdd.collect()  # collect

# Create RDD for dataset size (dsSize) and their corresponding speeds
tfr_dsSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['datasetsi
tfr_dsSize_speed = tfr_dsSize_speed_rdd.collect() # collect datasetsize


# Extra resources checked:
# https://spark.apache.org/docs/latest/rdd-programming-guide.html
# https://sparkbyexamples.com/pyspark-rdd/
# https://www.oreilly.com/library/view/learning-spark/9781449359034/ch0
# https://stackoverflow.com/questions/73014570/best-way-to-join-2-pair-
# https://towardsdatascience.com/the-ultimate-guide-to-functional-progr
# https://spark.apache.org/docs/latest/rdd-programming-guide.html#basic
# https://www.geeksforgeeks.org/pyspark-map-transformation/
# https://www.analyticsvidhya.com/blog/2016/10/using-pyspark-to-perform
```

In [ ]:

```
# v) create an RDD with the average reading speeds for each parameter v

# Create RDD for batch sizes and their corresponding avg speed
tfr_batchSize_avgSpeed_rdd = tfr_batchSize_speed_rdd.mapValues(lambda x
tfr_batchSize_avgSpeed = tfr_batchSize_avgSpeed_rdd.collect()  # collec

# Create RDD for tfr batch numbers and their corresponding avg speed
tfr_batchNums_avgSpeed_rdd = tfr_batchNum_speed_rdd.mapValues(lambda x:
tfr_batchNums_avgSpeed = tfr_batchNums_avgSpeed_rdd.collect()  # collec

# Create RDD for tfr repetitions and their corresponding avg speed
tfr_repetitions_avgSpeed_rdd = tfr_repetitions_speed_rdd.mapValues(lamb
tfr_repetitions_avgSpeed = tfr_repetitions_avgSpeed_rdd.collect() # col

# Create RDD for tfr dataset size and their corresponding avg speed
tfr_dsSize_avgSpeed_rdd = tfr_dsSize_speed_rdd.mapValues(lambda x: (x,
tfr_dsSize_avgSpeed = tfr_dsSize_avgSpeed_rdd.collect()  # collect data



# Create RDD for image files batch size and their corresponding avg spe
image_batchSizes_avgSpeed_rdd = image_batchSizes_speed_rdd.mapValues(la
image_batchSizes_avgSpeed = image_batchSizes_avgSpeed_rdd.collect()  #

# Create RDD for image batch number and their corresponding avg speed
image_batchNums_avgSpeed_rdd = image_batchNums_speed_rdd.mapValues(lamb
image_batchNums_avgSpeed = image_batchNums_avgSpeed_rdd.collect()  # co

# Create RDD for image repetitions and their corresponding avg speed
image_repetitions_avgSpeed_rdd = image_repetitions_speed_rdd.mapValues(
image_repetitions_avgSpeed = image_repetitions_avgSpeed_rdd.collect() #

# Create RDD for image dataset size and their corresponding avg speed
image_dsSize_avgSpeed_rdd = image_dsSize_speed_rdd.mapValues(lambda x:
image_dsSize_avgSpeed = image_dsSize_avgSpeed_rdd.collect()  # collect


# Some extra resources checked:
# https://spark.apache.org/docs/latest/api/python/reference/api/pyspark
#
```

In [ ]:

```python
# vi) write the results to a pickle file in your bucket (2%)

# Save object to local file using pickle, upload to GCS, and log status
def save(object,bucket,filename):
    with open(filename, mode='wb') as f:
      pickle.dump(object,f)
    print("Saving{} to {}".format(filename,bucket))
    import subprocess
    proc=subprocess.run(["gsutil","cp",filename,bucket],stderr=subproce
    print("gstuil returned: " + str(proc.returncode))
    print(str(proc.stderr))

filename="2aVI_results.pkl"
with open(filename,mode='wb') as f:
      pickle.dump(tfr_batchSize_speed,f)
      pickle.dump(tfr_batchNum_speed,f)
      pickle.dump(tfr_repetitions_speed,f)
      pickle.dump(tfr_dsSize_speed,f)
      pickle.dump(image_batchSizes_speed,f)
      pickle.dump(image_batchNums_speed,f)
      pickle.dump(image_repetitions_speed,f)
      pickle.dump(image_dsSize_speed,f)
      pickle.dump(tfr_batchSize_avgSpeed,f)
      pickle.dump(tfr_batchNums_avgSpeed,f)
      pickle.dump(tfr_repetitions_avgSpeed,f)
      pickle.dump(tfr_dsSize_avgSpeed,f)
      pickle.dump(image_batchSizes_avgSpeed,f)
      pickle.dump(image_batchNums_avgSpeed,f)
      pickle.dump(image_repetitions_avgSpeed,f)
      pickle.dump(image_dsSize_avgSpeed,f)


print("Saving {} to {}".format(filename,BUCKET))
import subprocess
proc = subprocess.run(["gsutil", "cp",filename, BUCKET], stderr=subproc
print("gstuil returned: " +str(proc.returncode))
print(str(proc.stderr))


# Extra resources consulted:
# https://www.geeksforgeeks.org/file-handling-python/
# https://docs.python.org/3/library/subprocess.html
# https://stackoverflow.com/questions/40982132/python-pickle-dump-wb-pa
```

```
Saving 2aVI_results.pkl to gs://big-data-cw-420116-storage
gstuil returned: 0
b'Copying file://2aVI_results.pkl [Content-Type=application/octet-strea
m]...\n/ [0 files][    0.0 B/ 10.6 KiB]
\r/ [1 files][ 10.6 KiB/ 10.6 KiB]
\r-\r\nOperation completed over 1 objects/10.6 KiB.
\n'
```

In [ ]:

```
# vii) Write your code it into a file using the cell magic %%writefile

%%writefile spark_job.py
# Setting the working environment
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp
#import scipy.stats
import time
import tensorflow as tf
print ("TensorFlow Version" == tf.__version__)

#from matplotlib import pyplot as plt
from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

PROJECT = 'big-data-cw-420116'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
          # labels for the data
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images


# Pre-defined function. Function to parse the TFRecord
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
```

```
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

# Pre-defined function. Function to load the dataset
def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

# Creating function for checking the performance
def time_configs_new(parameters_rdd):
  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
  dset = load_dataset(filenames)

  batch = dset.batch(batch_size)
  sample_set = dset.take(batch_num)

  tt = []
  for rep in range(repetition):
    start = time.time()
    for picture in sample_set:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

#  i) combine the previous cells to have the code to create a dataset o
# ii) get a Spark context and create the dataset and run timing test fo

batch_sizes = [6, 8, 10, 12]
batch_numbers = [6, 9, 12, 15]
repetitions = [1, 2, 3]

parameter_list = []
```

```
for batch_size in batch_sizes:
  for batch in batch_numbers:
    for r in repetitions:
      parameter_list.append([batch_size,batch,r])

columns = ["batch_sizes", "batch_nums", "repetitions", "datasetsize", "

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Parallelize data into RDD
tfr_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
tfr_files_sparkSession = SparkSession(sc)

# Flattening RDD into list of list
tfr_files = tfr_files_rdd.flatMap(time_configs_new)

# Construct dataframe (df) for TFR files
tfr_files_df = tfr_files.toDF(columns)

# Load and decode dataset
def load_dataset_decoded():
  dataset_filename = tf.data.Dataset.list_files(GCS_PATTERN)
  datasetDecoded = dataset_filename.map(decode_jpeg_and_label)
  datasetfn = datasetDecoded.map(resize_and_crop_image)
  return datasetfn

# create function for image files
def img_configs_new(parameters_rdd):
  dset = load_dataset_decoded()
  tt = []

  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  batch = dset.batch(batch_size)
  sample = dset.take(batch_num)

  for rep in range(repetition):
    start = time.time()
    for picture in sample:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Parallelize data into RDD
image_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
image_files_sparkSession = SparkSession(sc)

```

```
# Flattening RDD into list of list
image_files = image_files_rdd.flatMap(img_configs_new)

# Create df for image files
image_files_df = image_files.toDF(columns)

# iii) transform the resulting RDD to the structure ( parameter_combinc
tfr_array_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_sizes']),
                                                int(x['batch_nums']),
                                                int(x['repetitions']),
                                                int(x['datasetsize']),
                                                float(x['reading_speed'
                                                float(x['throughput']))
tfr_array_rdd.take(6)



# iv) create an RDD with all results for each parameter as (parameter_v

# Create RDD for batch sizes and their corresponding speeds
tfr_batchSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_
tfr_batchSize_speed = tfr_batchSize_speed_rdd.collect() # collect batch

# Create RDD for batch numbers and their corresponding speeds
tfr_batchNum_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_n
tfr_batchNum_speed = tfr_batchNum_speed_rdd.collect()  # collect batch

# Create RDD for repetitions and their corresponding speeds
tfr_repetitions_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['repe
tfr_repetitions_speed = tfr_repetitions_speed_rdd.collect()  # collect

# Create RDD for dataset size (dsSize) and their corresponding speeds
tfr_dsSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['datasetsi
tfr_dsSize_speed = tfr_dsSize_speed_rdd.collect() # collect datasetsize

# Create RDD for image files batch sieze  and their corresponding speed
image_batchSizes_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['b
image_batchSizes_speed = image_batchSizes_speed_rdd.collect()  # collec

# Create RDD for image files batch numbers and their corresponding spee
image_batchNums_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['ba
image_batchNums_speed = image_batchNums_speed_rdd.collect()   # collect

# Create RDD for image files repetitions and their corresponding speeds
image_repetitions_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['
image_repetitions_speed = image_repetitions_speed_rdd.collect()  # coll

# Create RDD for image files data set size and their corresponding spee
image_dsSize_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['datas
image_dsSize_speed = image_dsSize_speed_rdd.collect() # collect dataset



# v) create an RDD with the average reading speeds for each parameter v

# Create RDD for batch sizes and their corresponding avg speed
tfr_batchSize_avgSpeed_rdd = tfr_batchSize_speed_rdd.mapValues(lambda x
tfr_batchSize_avgSpeed = tfr_batchSize_avgSpeed_rdd.collect()  # collec

```

```
# Create RDD for tfr batch numbers and their corresponding avg speed
tfr_batchNums_avgSpeed_rdd = tfr_batchNum_speed_rdd.mapValues(lambda x:
tfr_batchNums_avgSpeed = tfr_batchNums_avgSpeed_rdd.collect()  # collec

# Create RDD for tfr repetitions and their corresponding avg speed
tfr_repetitions_avgSpeed_rdd = tfr_repetitions_speed_rdd.mapValues(lamb
tfr_repetitions_avgSpeed = tfr_repetitions_avgSpeed_rdd.collect() # col

# Create RDD for tfr dataset size and their corresponding avg speed
tfr_dsSize_avgSpeed_rdd = tfr_dsSize_speed_rdd.mapValues(lambda x: (x,
tfr_dsSize_avgSpeed = tfr_dsSize_avgSpeed_rdd.collect()  # collect data

# Create RDD for image files batch size and their corresponding avg spe
image_batchSizes_avgSpeed_rdd = image_batchSizes_speed_rdd.mapValues(la
image_batchSizes_avgSpeed = image_batchSizes_avgSpeed_rdd.collect()  #

# Create RDD for image batch number and their corresponding avg speed
image_batchNums_avgSpeed_rdd = image_batchNums_speed_rdd.mapValues(lamb
image_batchNums_avgSpeed = image_batchNums_avgSpeed_rdd.collect()  # co

# Create RDD for image repetitions and their corresponding avg speed
image_repetitions_avgSpeed_rdd = image_repetitions_speed_rdd.mapValues(
image_repetitions_avgSpeed = image_repetitions_avgSpeed_rdd.collect() #

# Create RDD for image dataset size and their corresponding avg speed
image_dsSize_avgSpeed_rdd = image_dsSize_speed_rdd.mapValues(lambda x:
image_dsSize_avgSpeed = image_dsSize_avgSpeed_rdd.collect()  # collect



# vi) write the results to a pickle file in your bucket (2%)

#Save object to local file using pickle, upload to GCS, and log status
def save(object,bucket,filename):
    with open(filename, mode='wb') as f:
      pickle.dump(object,f)
    print("Saving{} to {}".format(filename,bucket))
    import subprocess
    proc=subprocess.run(["gsutil","cp",filename,bucket],stderr=subproce
    print("gstuil returned: " + str(proc.returncode))
    print(str(proc.stderr))

filename="2aVI_results.pkl"
with open(filename,mode='wb') as f:
      pickle.dump(tfr_batchSize_speed,f)
      pickle.dump(tfr_batchNum_speed,f)
      pickle.dump(tfr_repetitions_speed,f)
      pickle.dump(tfr_dsSize_speed,f)
      pickle.dump(image_batchSizes_speed,f)
      pickle.dump(image_batchNums_speed,f)
      pickle.dump(image_repetitions_speed,f)
      pickle.dump(image_dsSize_speed,f)
      pickle.dump(tfr_batchSize_avgSpeed,f)
      pickle.dump(tfr_batchNums_avgSpeed,f)
      pickle.dump(tfr_repetitions_avgSpeed,f)
      pickle.dump(tfr_dsSize_avgSpeed,f)
      pickle.dump(image_batchSizes_avgSpeed,f)
      pickle.dump(image_batchNums_avgSpeed,f)
      pickle.dump(image_repetitions_avgSpeed,f)
      pickle.dump(image_dsSize_avgSpeed,f)
```

```
306
307 print("Saving {} to {}".format(filename,BUCKET))
308 import subprocess
309 proc = subprocess.run(["gsutil", "cp",filename, BUCKET], stderr=subproc
310 print("gstuil returned: " +str(proc.returncode))
311 print(str(proc.stderr))
```

```
Writing spark_job.py
```

2b) Testing the code and collecting results (4%)

i) First, test locally with `%run` .

It is useful to create a **new filename argument**, so that old results don't get overwritten.

You can for instance use `datetime.datetime.now().strftime("%y%m%d-%H%M")` to get a string with the current date and time and use that in the file name.

In [ ]:
```
1 ### CODING TASK
2 %run ./spark_job.py
```

```
False
Saving 2aVI_results.pkl to gs://big-data-cw-420116-storage
gstuil returned: 0
b'Copying file://2aVI_results.pkl [Content-Type=application/octet-strea
m]...\n/ [0 files][    0.0 B/ 10.6 KiB]
\r/ [1 files][ 10.6 KiB/ 10.6 KiB]
\r-\r\nOperation completed over 1 objects/10.6 KiB.
\n'
```

```
<Figure size 640x480 with 0 Axes>
```

In [ ]:

```
%%writefile spark_job2b.py
# Setting the working environment
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
import time
import tensorflow as tf
print ("TensorFlow Version" == tf.__version__)

from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

PROJECT = 'big-data-cw-420116'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
          # labels for the data
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images

# Pre-defined function. Function to parse the TFRecord
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
```

```
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

# Pre-defined function. Function to load the dataset
def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

# Creating function for checking the performance
def time_configs_new(parameters_rdd):
  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
  dset = load_dataset(filenames)

  batch = dset.batch(batch_size)
  sample_set = dset.take(batch_num)

  tt = []
  for rep in range(repetition):
    start = time.time()
    for picture in sample_set:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

#  i) combine the previous cells to have the code to create a dataset d
# ii) get a Spark context and create the dataset and run timing test fo

batch_sizes = [6, 8, 10, 12]
batch_numbers = [6, 9, 12, 15]
repetitions = [1, 2, 3]

parameter_list = []
for batch_size in batch_sizes:
  for batch in batch_numbers:
    for r in repetitions:
      parameter_list.append([batch_size,batch,r])

columns = ["batch_sizes", "batch_nums", "repetitions", "datasetsize", "
```

```
123
124 # Get spark context for RDDs
125 sc = pyspark.SparkContext.getOrCreate()
126
127 # Parallelize data into RDD
128 tfr_files_rdd = sc.parallelize(parameter_list)
129
130 # Initialize spark session for df conversion
131 tfr_files_sparkSession = SparkSession(sc)
132
133 # Flattening RDD into list of list
134 tfr_files = tfr_files_rdd.flatMap(time_configs_new)
135
136 # Construct dataframe (df) for TFR files
137 tfr_files_df = tfr_files.toDF(columns)
138
139 # Load and decode dataset
140 def load_dataset_decoded():
141   dataset_filename = tf.data.Dataset.list_files(GCS_PATTERN)
142   datasetDecoded = dataset_filename.map(decode_jpeg_and_label)
143   datasetfn = datasetDecoded.map(resize_and_crop_image)
144   return datasetfn
145
146 # create function for image files
147 def img_configs_new(parameters_rdd):
148   dset = load_dataset_decoded()
149   tt = []
150
151   batch_size = parameters_rdd[0]
152   batch_num = parameters_rdd[1]
153   repetition = parameters_rdd[2]
154
155   batch = dset.batch(batch_size)
156   sample = dset.take(batch_num)
157
158   for rep in range(repetition):
159     start = time.time()
160     for picture in sample:
161       print('string', file=open("/dev/null", mode='w'))
162     end = time.time()
163     reading_speed = end - start
164     throughput = float((batch_size * batch_num) / (end - start))
165     datasetsize = batch_size * batch_num
166     tt.append([batch_size, batch_num, repetition, datasetsize, reading_
167   return tt
168
169 # Get spark context for RDDs
170 sc = pyspark.SparkContext.getOrCreate()
171
172 # Parallelize data into RDD
173 image_files_rdd = sc.parallelize(parameter_list)
174
175 # Initialize spark session for df conversion
176 image_files_sparkSession = SparkSession(sc)
177
178 # Flattening RDD into list of list
179 image_files = image_files_rdd.flatMap(img_configs_new)
180
181 # Create df for image files
182 image_files_df = image_files.toDF(columns)
183
```

```
# iii) transform the resulting RDD to the structure ( parameter_combinc
tfr_array_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_sizes']),i
                                                int(x['datasetsi
tfr_array_rdd.take(6)


# iv) create an RDD with all results for each parameter as (parameter_v
# Create RDD for batch sizes and their corresponding speeds
tfr_batchSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_
tfr_batchSize_speed = tfr_batchSize_speed_rdd.collect() # collect batch

# Create RDD for batch numbers and their corresponding speeds
tfr_batchNum_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_n
tfr_batchNum_speed = tfr_batchNum_speed_rdd.collect()  # collect batch

# Create RDD for repetitions and their corresponding speeds
tfr_repetitions_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['repe
tfr_repetitions_speed = tfr_repetitions_speed_rdd.collect()  # collect

# Create RDD for dataset size (dsSize) and their corresponding speeds
tfr_dsSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['datasetsi
tfr_dsSize_speed = tfr_dsSize_speed_rdd.collect() # collect datasetsize

# Create RDD for image files batch sieze  and their corresponding speed
image_batchSizes_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['b
image_batchSizes_speed = image_batchSizes_speed_rdd.collect()  # collec

# Create RDD for image files batch numbers and their corresponding spee
image_batchNums_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['ba
image_batchNums_speed = image_batchNums_speed_rdd.collect()   # collect

# Create RDD for image files repetitions and their corresponding speeds
image_repetitions_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['
image_repetitions_speed = image_repetitions_speed_rdd.collect()  # coll

# Create RDD for image files data set size and their corresponding spee
image_dsSize_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['datas
image_dsSize_speed = image_dsSize_speed_rdd.collect() # collect dataset


# v) create an RDD with the average reading speeds for each parameter v

# Create RDD for batch sizes and their corresponding avg speed
tfr_batchSize_avgSpeed_rdd = tfr_batchSize_speed_rdd.mapValues(lambda x
tfr_batchSize_avgSpeed = tfr_batchSize_avgSpeed_rdd.collect()  # collec

# Create RDD for tfr batch numbers and their corresponding avg speed
tfr_batchNums_avgSpeed_rdd = tfr_batchNum_speed_rdd.mapValues(lambda x:
tfr_batchNums_avgSpeed = tfr_batchNums_avgSpeed_rdd.collect()  # collec

# Create RDD for tfr repetitions and their corresponding avg speed
tfr_repetitions_avgSpeed_rdd = tfr_repetitions_speed_rdd.mapValues(lamb
tfr_repetitions_avgSpeed = tfr_repetitions_avgSpeed_rdd.collect() # col

# Create RDD for tfr dataset size and their corresponding avg speed
tfr_dsSize_avgSpeed_rdd = tfr_dsSize_speed_rdd.mapValues(lambda x: (x,
tfr_dsSize_avgSpeed = tfr_dsSize_avgSpeed_rdd.collect()  # collect data

# Create RDD for image files batch size and their corresponding avg spe
image_batchSizes_avgSpeed_rdd = image_batchSizes_speed_rdd.mapValues(la
image_batchSizes_avgSpeed = image_batchSizes_avgSpeed_rdd.collect()  #
```

```
245
246 # Create RDD for image batch number and their corresponding avg speed
247 image_batchNums_avgSpeed_rdd = image_batchNums_speed_rdd.mapValues(lamb
248 image_batchNums_avgSpeed = image_batchNums_avgSpeed_rdd.collect()  # co
249
250 # Create RDD for image repetitions and their corresponding avg speed
251 image_repetitions_avgSpeed_rdd = image_repetitions_speed_rdd.mapValues(
252 image_repetitions_avgSpeed = image_repetitions_avgSpeed_rdd.collect() #
253
254 # Create RDD for image dataset size and their corresponding avg speed
255 image_dsSize_avgSpeed_rdd = image_dsSize_speed_rdd.mapValues(lambda x:
256 image_dsSize_avgSpeed = image_dsSize_avgSpeed_rdd.collect()  # collect
257
258
259 # vi) write the results to a pickle file in your bucket (2%)
260
261 #Save object to local file using pickle, upload to GCS, and log status
262 def save(object,bucket,filename):
263     with open(filename, mode='wb') as f:
264       pickle.dump(object,f)
265     print("Saving{} to {}".format(filename,bucket))
266     import subprocess
267     proc=subprocess.run(["gsutil","cp",filename,bucket],stderr=subproce
268     print("gstuil returned: " + str(proc.returncode))
269     print(str(proc.stderr))
270
271 filename="2b_results.pkl"
272 with open(filename,mode='wb') as f:
273       pickle.dump(tfr_batchSize_speed,f)
274       pickle.dump(tfr_batchNum_speed,f)
275       pickle.dump(tfr_repetitions_speed,f)
276       pickle.dump(tfr_dsSize_speed,f)
277       pickle.dump(image_batchSizes_speed,f)
278       pickle.dump(image_batchNums_speed,f)
279       pickle.dump(image_repetitions_speed,f)
280       pickle.dump(image_dsSize_speed,f)
281       pickle.dump(tfr_batchSize_avgSpeed,f)
282       pickle.dump(tfr_batchNums_avgSpeed,f)
283       pickle.dump(tfr_repetitions_avgSpeed,f)
284       pickle.dump(tfr_dsSize_avgSpeed,f)
285       pickle.dump(image_batchSizes_avgSpeed,f)
286       pickle.dump(image_batchNums_avgSpeed,f)
287       pickle.dump(image_repetitions_avgSpeed,f)
288       pickle.dump(image_dsSize_avgSpeed,f)
289
290 print("Saving {} to {}".format(filename,BUCKET))
291 import subprocess
292 proc = subprocess.run(["gsutil", "cp",filename, BUCKET], stderr=subproc
293 print("gstuil returned: " +str(proc.returncode))
294 print(str(proc.stderr))
```

```
Writing spark_job2b.py
```

ii) Cloud

If you have a cluster running, you can run the speed test job in the cloud.

While you run this job, switch to the Dataproc web page and take **screenshots of the CPU and network load** over time. They are displayed with some delay, so you may need to wait

In [ ]:
```
#Was getting error to create the cluster so I deleted it to create it a
!gcloud dataproc clusters delete big-data-cw-420116-cluster
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/0338bdcb-b7ec-3f6b-90a6-270c2f41b32b].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:
```
### CODING TASK ###
# Set up a cluster with a single machine using the maximal SSD size (10

import time
start_time = time.time()
#REGION = europe-west3
!gcloud dataproc clusters create $CLUSTER \
    --bucket $PROJECT-storage \
    --image-version 1.4-ubuntu18 \
    --master-machine-type n1-standard-8 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 0\
    --initialization-actions gs://goog-dataproc-initialization-actions-
    --metadata PIP_PACKAGES=tensorflow==2.4.0
%time

end_time = time.time()
execution_time = end_time - start_time
print(f"Execution time: {execution_time} seconds")


```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/11a64304-c8fa-37e8-84e6-50a16bf977a2].

WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 8.34 µs
Execution time: 153.5760896205902 seconds
```

In [ ]:

```
# get information of the single machine cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 35f0437b-4971-4385-b85a-3808433b9719
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw
-420116/global/networks/default (https://www.googleapis.com/compute/v1/pro
jects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.co
m/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.go
ogleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapi
s.com/auth/bigtable.data)
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly (http
s://www.googleapis.com/auth/cloud.useraccounts.readonly)
    - https://www.googleapis.com/auth/devstorage.full_control (https://ww
w.googleapis.com/auth/devstorage.full_control)
    - https://www.googleapis.com/auth/devstorage.read_write (https://www.g
oogleapis.com/auth/devstorage.read_write)
    - https://www.googleapis.com/auth/logging.write (https://www.googleapi
s.com/auth/logging.write)
    zoneUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-42
0116/zones/us-central1-c (https://www.googleapis.com/compute/v1/projects/b
ig-data-cw-420116/zones/us-central1-c)
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-central1/
python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-datapro
c/global/images/dataproc-1-4-ubu18-20220125-170200-rc01 (https://www.googl
eapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-4-ub
u18-20220125-170200-rc01)
    instanceNames:
    - big-data-cw-420116-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/big-dat
a-cw-420116/zones/us-central1-c/machineTypes/n1-standard-8 (https://www.go
ogleapis.com/compute/v1/projects/big-data-cw-420116/zones/us-central1-c/ma
chineTypes/n1-standard-8)
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.4.80-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-pol
icy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
```

```
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: big-data-cw-420116-cluster-
m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '10'
      hdfs:dfs.namenode.servicerpc-address: big-data-cw-420116-cluster-m:8
051
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '4000'
      mapred:mapreduce.job.maps: '21'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx2457m
      mapred:mapreduce.map.memory.mb: '3072'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx2457m
      mapred:mapreduce.reduce.memory.mb: '3072'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx2457m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '3072'
      spark-env:SPARK_DAEMON_MEMORY: 4000m
      spark:spark.driver.maxResultSize: 3840m
      spark:spark.driver.memory: 7680m
      spark:spark.executor.cores: '4'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 11171m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.Datap
rocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '4000'
      yarn:yarn.nodemanager.resource.cpu-vcores: '8'
      yarn:yarn.nodemanager.resource.memory-mb: '24576'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-
secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '24576'
      yarn:yarn.scheduler.minimum-allocation-mb: '1024'
  tempBucket: dataproc-temp-us-central1-779145334445-vrtyf48i
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: big-data-cw-420116-cluster
  goog-dataproc-cluster-uuid: 35f0437b-4971-4385-b85a-3808433b9719
  goog-dataproc-location: us-central1
projectId: big-data-cw-420116
status:
  state: RUNNING
```

```
  stateStartTime: '2024-05-04T11:40:55.782481Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-04T11:38:26.671779Z'
```

In [ ]:

```
# submit the spark job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_job2b.py
```

```
Job [cac7e1460f9c477cb944f548c1cdb5a5] submitted.
Waiting for job output...
2024-05-04 11:41:04.458709: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 11:41:04.458757: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
False
24/05/04 11:41:07 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 11:41:07 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 11:41:07 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 11:41:07 INFO org.spark_project.jetty.util.log: Logging initializ
ed @6112ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 11:41:07 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
24/05/04 11:41:07 INFO org.spark_project.jetty.server.Server: Started @622
1ms
24/05/04 11:41:07 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@69ddacff{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
24/05/04 11:41:07 WARN org.apache.spark.scheduler.FairSchedulableBuilder:
Fair Scheduler configuration file not found so jobs will be scheduled in F
IFO order. To use fair scheduling, configure pools in fairscheduler.xml or
set spark.scheduler.allocation.file to a file that contains the configurat
ion.
24/05/04 11:41:08 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.0.14:8032
24/05/04 11:41:08 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.0.14:
10200
24/05/04 11:41:11 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714822762403_0001
Saving 2b_results.pkl to gs://big-data-cw-420116-storage
gstuil returned: 0
b'Copying file://2b_results.pkl [Content-Type=application/octet-strea
m]...\n/ [0 files][    0.0 B/ 11.3 KiB]
\r/ [1 files][ 11.3 KiB/ 11.3 KiB]
\r\nOperation completed over 1 objects/11.3 KiB.
\n'
24/05/04 11:49:05 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@69ddacff{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [cac7e1460f9c477cb944f548c1cdb5a5] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/35f0437b-4971-4385-b85a-3808433b9719/jobs/cac7e1460f9c477cb944
f548c1cdb5a5/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/35f0437b-4971-4385-b85a-3808433b9719/jobs/cac7e1460f9c477cb9
44f548c1cdb5a5/driveroutput
jobUuid: 5ca9651a-fcc5-30b5-b77a-b0fca588db51
placement:
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 35f0437b-4971-4385-b85a-3808433b9719
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/35f0437b-4971-4385-b85a-3808433b9719/jobs/cac7e1460f9c477cb944f5
48c1cdb5a5/staging/spark_job2b.py
```

```
reference:
  jobId: cac7e1460f9c477cb944f548c1cdb5a5
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T11:49:09.351993Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-04T11:40:59.944655Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T11:40:59.975410Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T11:41:00.252670Z'
yarnApplications:
- name: spark_job2b.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714822762403_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714822762403_0001/)
```

2c) Improve efficiency (6%)

If you implemented a straightfoward version of 2a), you will **probably have an inefficiency** in your code.

Because we are reading multiple times from an RDD to read the values for the different parameters and their averages, caching existing results is important. Explain **where in the process caching can help**, and **add a call to** `RDD.cache()` to your code, if you haven't yet. Measure the the effect of using caching or not using it.

Make the **suitable change** in the code you have written above and mark them up in comments as `### TASK 2c ###` .

Explain in your report what the **reasons for this change** are and **demonstrate and interpret its effect**

In [ ]:

```
### CODING TASK ###
%%writefile spark_job2c.py
# Setting the working environment
import datetime
import math
import numpy as np
import os
import pyspark
import pickle
import pandas as pd
import random
import string
import sys
#import scipy as sp
#import scipy.stats
import time
import tensorflow as tf
print ("TensorFlow Version" == tf.__version__)

#from matplotlib import pyplot as plt
from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext

PROJECT = 'big-data-cw-420116'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input f
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
          # labels for the data
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images

# Pre-defined function. Function to parse the TFRecord
def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = b
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] me
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the f
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/'
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source i
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
```

```
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), #
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  #
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh -
    return image, label

# Pre-defined function. Function to load the dataset
def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_determinis
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

# Creating function for checking the performance
def time_configs_new(parameters_rdd):
  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
  dset = load_dataset(filenames)

  batch = dset.batch(batch_size)
  sample_set = dset.take(batch_num)

  tt = []
  for rep in range(repetition):
    start = time.time()
    for picture in sample_set:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

#  i) combine the previous cells to have the code to create a dataset d
# ii) get a Spark context and create the dataset and run timing test fo

batch_sizes = [6, 8, 10, 12]
batch_numbers = [6, 9, 12, 15]
repetitions = [1, 2, 3]

parameter_list = []
for batch_size in batch_sizes:
  for batch in batch_numbers:
```

```
    for r in repetitions:
      parameter_list.append([batch_size,batch,r])

columns = ["batch_sizes", "batch_nums", "repetitions", "datasetsize", "

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Parallelize data into RDD
tfr_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
tfr_files_sparkSession = SparkSession(sc)

# Flattening RDD into list of list
tfr_files = tfr_files_rdd.flatMap(time_configs_new)

### TASK 2c ###
tfr_files.cache() # Cache TFRecord files in RDD

# Construct dataframe (df) for TFR files
tfr_files_df = tfr_files.toDF(columns)

# Load and decode dataset
def load_dataset_decoded():
  dataset_filename = tf.data.Dataset.list_files(GCS_PATTERN)
  datasetDecoded = dataset_filename.map(decode_jpeg_and_label)
  datasetfn = datasetDecoded.map(resize_and_crop_image)
  return datasetfn

# create function for image files
def img_configs_new(parameters_rdd):
  dset = load_dataset_decoded()
  tt = []

  batch_size = parameters_rdd[0]
  batch_num = parameters_rdd[1]
  repetition = parameters_rdd[2]

  batch = dset.batch(batch_size)
  sample = dset.take(batch_num)

  for rep in range(repetition):
    start = time.time()
    for picture in sample:
      print('string', file=open("/dev/null", mode='w'))
    end = time.time()
    reading_speed = end - start
    throughput = float((batch_size * batch_num) / (end - start))
    datasetsize = batch_size * batch_num
    tt.append([batch_size, batch_num, repetition, datasetsize, reading_
  return tt

# Get spark context for RDDs
sc = pyspark.SparkContext.getOrCreate()

# Parallelize data into RDD
image_files_rdd = sc.parallelize(parameter_list)

# Initialize spark session for df conversion
image_files_sparkSession = SparkSession(sc)
```

```
184
185 # Flattening RDD into list of list
186 image_files = image_files_rdd.flatMap(img_configs_new)
187
188 ### TASK 2c ###
189 image_files.cache() # Cache image files in memory for quick access
190
191 # Create df for image files
192 image_files_df = image_files.toDF(columns)
193
194
195 # iii) transform the resulting RDD to the structure ( parameter_combina
196 tfr_array_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_sizes']),i
197                                                 int(x['datasetsi
198 tfr_array_rdd.take(6)
199
200
201 # iv) create an RDD with all results for each parameter as (parameter_v
202
203 # Create RDD for batch sizes and their corresponding speeds
204 tfr_batchSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_
205 tfr_batchSize_speed = tfr_batchSize_speed_rdd.collect() # collect batch
206
207 # Create RDD for batch numbers and their corresponding speeds
208 tfr_batchNum_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['batch_n
209 tfr_batchNum_speed = tfr_batchNum_speed_rdd.collect()  # collect batch
210
211 # Create RDD for repetitions and their corresponding speeds
212 tfr_repetitions_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['repe
213 tfr_repetitions_speed = tfr_repetitions_speed_rdd.collect()  # collect
214
215 # Create RDD for dataset size (dsSize) and their corresponding speeds
216 tfr_dsSize_speed_rdd = tfr_files_df.rdd.map(lambda x: (int(x['datasetsi
217 tfr_dsSize_speed = tfr_dsSize_speed_rdd.collect() # collect datasetsize
218
219 # Create RDD for image files batch sieze  and their corresponding speed
220 image_batchSizes_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['b
221 image_batchSizes_speed = image_batchSizes_speed_rdd.collect()  # collec
222
223 # Create RDD for image files batch numbers and their corresponding spee
224 image_batchNums_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['ba
225 image_batchNums_speed = image_batchNums_speed_rdd.collect()   # collect
226
227 # Create RDD for image files repetitions and their corresponding speeds
228 image_repetitions_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['
229 image_repetitions_speed = image_repetitions_speed_rdd.collect()  # coll
230
231 # Create RDD for image files data set size and their corresponding spee
232 image_dsSize_speed_rdd = image_files_df.rdd.map(lambda x: (int(x['datas
233 image_dsSize_speed = image_dsSize_speed_rdd.collect() # collect dataset
234
235
236 # v) create an RDD with the average reading speeds for each parameter v
237
238 # Create RDD for batch sizes and their corresponding avg speed
239 tfr_batchSize_avgSpeed_rdd = tfr_batchSize_speed_rdd.mapValues(lambda x
240 tfr_batchSize_avgSpeed = tfr_batchSize_avgSpeed_rdd.collect()  # collec
241
242 # Create RDD for tfr batch numbers and their corresponding avg speed
243 tfr_batchNums_avgSpeed_rdd = tfr_batchNum_speed_rdd.mapValues(lambda x:
244 tfr_batchNums_avgSpeed = tfr_batchNums_avgSpeed_rdd.collect()  # collec
```

```
245
246 # Create RDD for tfr repetitions and their corresponding avg speed
247 tfr_repetitions_avgSpeed_rdd = tfr_repetitions_speed_rdd.mapValues(lamb
248 tfr_repetitions_avgSpeed = tfr_repetitions_avgSpeed_rdd.collect() # col
249
250 # Create RDD for tfr dataset size and their corresponding avg speed
251 tfr_dsSize_avgSpeed_rdd = tfr_dsSize_speed_rdd.mapValues(lambda x: (x,
252 tfr_dsSize_avgSpeed = tfr_dsSize_avgSpeed_rdd.collect()  # collect data
253
254 # Create RDD for image files batch size and their corresponding avg spe
255 image_batchSizes_avgSpeed_rdd = image_batchSizes_speed_rdd.mapValues(la
256 image_batchSizes_avgSpeed = image_batchSizes_avgSpeed_rdd.collect()  #
257
258 # Create RDD for image batch number and their corresponding avg speed
259 image_batchNums_avgSpeed_rdd = image_batchNums_speed_rdd.mapValues(lamb
260 image_batchNums_avgSpeed = image_batchNums_avgSpeed_rdd.collect()  # co
261
262 # Create RDD for image repetitions and their corresponding avg speed
263 image_repetitions_avgSpeed_rdd = image_repetitions_speed_rdd.mapValues(
264 image_repetitions_avgSpeed = image_repetitions_avgSpeed_rdd.collect() #
265
266 # Create RDD for image dataset size and their corresponding avg speed
267 image_dsSize_avgSpeed_rdd = image_dsSize_speed_rdd.mapValues(lambda x:
268 image_dsSize_avgSpeed = image_dsSize_avgSpeed_rdd.collect()  # collect
269
270
271 # vi) write the results to a pickle file in your bucket (1%)
272
273 #Save object to local file using pickle, upload to GCS, and log status
274 def save(object,bucket,filename):
275     with open(filename, mode='wb') as f:
276       pickle.dump(object,f)
277     print("Saving{} to {}".format(filename,bucket))
278     import subprocess
279     proc=subprocess.run(["gsutil","cp",filename,bucket],stderr=subproce
280     print("gstuil returned: " + str(proc.returncode))
281     print(str(proc.stderr))
282
283 filename="2c_results.pkl"
284 with open(filename,mode='wb') as f:
285       pickle.dump(tfr_batchSize_speed,f)
286       pickle.dump(tfr_batchNum_speed,f)
287       pickle.dump(tfr_repetitions_speed,f)
288       pickle.dump(tfr_dsSize_speed,f)
289       pickle.dump(image_batchSizes_speed,f)
290       pickle.dump(image_batchNums_speed,f)
291       pickle.dump(image_repetitions_speed,f)
292       pickle.dump(image_dsSize_speed,f)
293       pickle.dump(tfr_batchSize_avgSpeed,f)
294       pickle.dump(tfr_batchNums_avgSpeed,f)
295       pickle.dump(tfr_repetitions_avgSpeed,f)
296       pickle.dump(tfr_dsSize_avgSpeed,f)
297       pickle.dump(image_batchSizes_avgSpeed,f)
298       pickle.dump(image_batchNums_avgSpeed,f)
299       pickle.dump(image_repetitions_avgSpeed,f)
300       pickle.dump(image_dsSize_avgSpeed,f)
301
302 print("Saving {} to {}".format(filename,BUCKET))
303 import subprocess
304 proc = subprocess.run(["gsutil", "cp",filename, BUCKET], stderr=subproc
305 print("gstuil returned: " +str(proc.returncode))
```

```
306 print(str(proc.stderr))
```

```
Writing spark_job2c.py
```

In [ ]:
```
1 #Was getting error to create the cluster so I deleted it to create it a
2 !gcloud dataproc clusters delete big-data-cw-420116-cluster
```

```
The cluster 'big-data-cw-420116-cluster' and all attached disks will be de
leted.

Do you want to continue (Y/n)?  y

Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/ead1de6a-5982-3c5d-ad8d-3651c4cfca3d].
Deleted [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster].
```

In [ ]:
```
1  ### CODING TASK ###
2  # cluster with a single machine using the maximal SSD size (100) and  1
3  #REGION = 'europe-west2'
4
5  import time
6  start_time = time.time()
7  !gcloud dataproc clusters create $CLUSTER \
8      --bucket $PROJECT-storage \
9      --image-version 1.5-ubuntu18 \
10     --master-machine-type n1-standard-8 \
11     --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
12     --num-workers 0\
13     --initialization-actions gs://goog-dataproc-initialization-actions-
14     --metadata "PIP_PACKAGES=tensorflow==2.4.0 protobuf==3.19.0"
15 %time
16 end_time = time.time()
17 execution_time = end_time - start_time
18 print(f"Execution time: {execution_time} seconds")
19 #     --metadata PIP_PACKAGES=tensorflow==2.4.0 was giving error
20 #     --initialization-actions gs://goog-dataproc-initialization-actions
21 #     --initialization-actions gs://goog-dataproc-initialization-actions
```

```
Waiting on operation [projects/big-data-cw-420116/regions/us-central1/oper
ations/ed6ecc51-42b7-3c3d-a94a-3fe2fd774f56].

WARNING: Don't create production clusters that reference initialization ac
tions located in the gs://goog-dataproc-initialization-actions-REGION publ
ic buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version
of a initialization action in public buckets may break your cluster creati
on. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-central1/p
ython/pip-install.sh
WARNING: The firewall rules for specified network or subnetwork would allo
w ingress traffic from 0.0.0.0/0, which could be a security risk.
Created [https://dataproc.googleapis.com/v1/projects/big-data-cw-420116/re
gions/us-central1/clusters/big-data-cw-420116-cluster] Cluster placed in z
one [us-central1-c].
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.63 µs
Execution time: 190.0779001712799 seconds
```

In [ ]:

```
# get information of the  cluster
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: big-data-cw-420116-cluster
clusterUuid: 391c593b-c563-486e-8244-0386eb840cea
config:
  configBucket: big-data-cw-420116-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 protobuf==3.19.0
    networkUri: https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default (https://www.googleapis.com/compute/v1/projects/big-data-cw-420116/global/networks/default)
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery (https://www.googleapis.com/auth/bigquery)
    - https://www.googleapis.com/auth/bigtable.admin.table (https://www.googleapis.com/auth/bigtable.admin.table)
    - https://www.googleapis.com/auth/bigtable.data (https://www.googleapis.com/auth/bigtable.data)
```

In [ ]:

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_job2c.py
```

```
Job [4e5e8b972aba461485e1f5ec3271cabd] submitted.
Waiting for job output...
2024-05-04 12:06:46.957241: W tensorflow/stream_executor/platform/default/
dso_loader.cc:60] Could not load dynamic library 'libcudart.so.11.0'; dler
ror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-04 12:06:46.957294: I tensorflow/stream_executor/cuda/cudart_stub.
cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your
machine.
False
24/05/04 12:06:49 INFO org.apache.spark.SparkEnv: Registering MapOutputTra
cker
24/05/04 12:06:49 INFO org.apache.spark.SparkEnv: Registering BlockManager
Master
24/05/04 12:06:49 INFO org.apache.spark.SparkEnv: Registering OutputCommit
Coordinator
24/05/04 12:06:49 INFO org.spark_project.jetty.util.log: Logging initializ
ed @6271ms to org.spark_project.jetty.util.log.Slf4jLog
24/05/04 12:06:49 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-
SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
24/05/04 12:06:49 INFO org.spark_project.jetty.server.Server: Started @639
3ms
24/05/04 12:06:49 INFO org.spark_project.jetty.server.AbstractConnector: S
tarted ServerConnector@4f00657d{HTTP/1.1, (http/1.1)}{0.0.0.0:39371}
24/05/04 12:06:51 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting t
o ResourceManager at big-data-cw-420116-cluster-m/10.128.0.16:8032
24/05/04 12:06:51 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting
to Application History server at big-data-cw-420116-cluster-m/10.128.0.16:
10200
24/05/04 12:06:51 INFO org.apache.hadoop.conf.Configuration: resource-type
s.xml not found
24/05/04 12:06:51 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Unable to find 'resource-types.xml'.
24/05/04 12:06:51 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
24/05/04 12:06:51 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = vcores, units = , type = COUNTABLE
24/05/04 12:06:54 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientIm
pl: Submitted application application_1714824310332_0001
Saving 2c_results.pkl to gs://big-data-cw-420116-storage
gstuil returned: 0
b'WARNING:  Python 3.5-3.7 will be deprecated on August 8th, 2023. Please
use Python version 3.8 and up.\n\nIf you have a compatible Python interpre
ter installed, you can use it by setting\nthe CLOUDSDK_PYTHON environment
variable to point to it.\n\nCopying file://2c_results.pkl [Content-Type=ap
plication/octet-stream]...\n/ [0 files][    0.0 B/ 11.2 KiB]
\r/ [1 files][ 11.2 KiB/ 11.2 KiB]
\r\nOperation completed over 1 objects/11.2 KiB.
\n'
24/05/04 12:09:11 INFO org.spark_project.jetty.server.AbstractConnector: S
topped Spark@4f00657d{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [4e5e8b972aba461485e1f5ec3271cabd] finished successfully.
done: true
driverControlFilesUri: gs://big-data-cw-420116-storage/google-cloud-datapr
oc-metainfo/391c593b-c563-486e-8244-0386eb840cea/jobs/4e5e8b972aba461485e1
f5ec3271cabd/
driverOutputResourceUri: gs://big-data-cw-420116-storage/google-cloud-data
proc-metainfo/391c593b-c563-486e-8244-0386eb840cea/jobs/4e5e8b972aba461485
e1f5ec3271cabd/driveroutput
jobUuid: 3f2c7266-569a-3dbb-b3c4-75887dcda5ac
placement:
```

```
  clusterName: big-data-cw-420116-cluster
  clusterUuid: 391c593b-c563-486e-8244-0386eb840cea
pysparkJob:
  mainPythonFileUri: gs://big-data-cw-420116-storage/google-cloud-dataproc
-metainfo/391c593b-c563-486e-8244-0386eb840cea/jobs/4e5e8b972aba461485e1f5
ec3271cabd/staging/spark_job2c.py
reference:
  jobId: 4e5e8b972aba461485e1f5ec3271cabd
  projectId: big-data-cw-420116
status:
  state: DONE
  stateStartTime: '2024-05-04T12:09:14.643674Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-04T12:06:41.998855Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-04T12:06:42.025243Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-04T12:06:42.322529Z'
yarnApplications:
- name: spark_job2c.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://big-data-cw-420116-cluster-m:8088/proxy/application_
1714824310332_0001/ (http://big-data-cw-420116-cluster-m:8088/proxy/applic
ation_1714824310332_0001/)
```

2d) Retrieve, analyse and discuss the output (12%)

Run the tests over a wide range of different paramters and list the results in a table.

Perform a **linear regression** (e.g. using scikit-learn) over **the values for each parameter** and for the **two cases** (reading from image files/reading TFRecord files). List a **table** with the output and interpret the results in terms of the effects of overall.
Also, **plot** the output values, the averages per parameter value and the regression lines for each parameter and for the product of batch_size and batch_number

Discuss the **implications** of this result for **applications** like large-scale machine learning. Keep in mind that cloud data may be stored in distant physical locations. Use the numbers provided in the PDF latency-numbers document available on Moodle or here (https://gist.github.com/hellerbarde/2843375) for your arguments.

How is the **observed** behaviour **similar or different** from what you'd expect from a **single machine**? Why would cloud providers tie throughput to capacity of disk resources?

By **parallelising** the speed test we are making **assumptions** about the limits of the bucket reading speeds. See here (https://cloud.google.com/storage/docs/request-rate) for more information. Discuss, **what we need to consider** in **speed tests** in parallel on the cloud, which bottlenecks we might be identifying, and how this relates to your results.

Discuss to what extent **linear modelling** reflects the **effects** we are observing. Discuss what could be expected from a theoretical perspective and what can be useful in practice.

Write your **code below** and **include the output** in your submitted `ipynb` file. Provide the answer **text in your report**.

In [ ]:

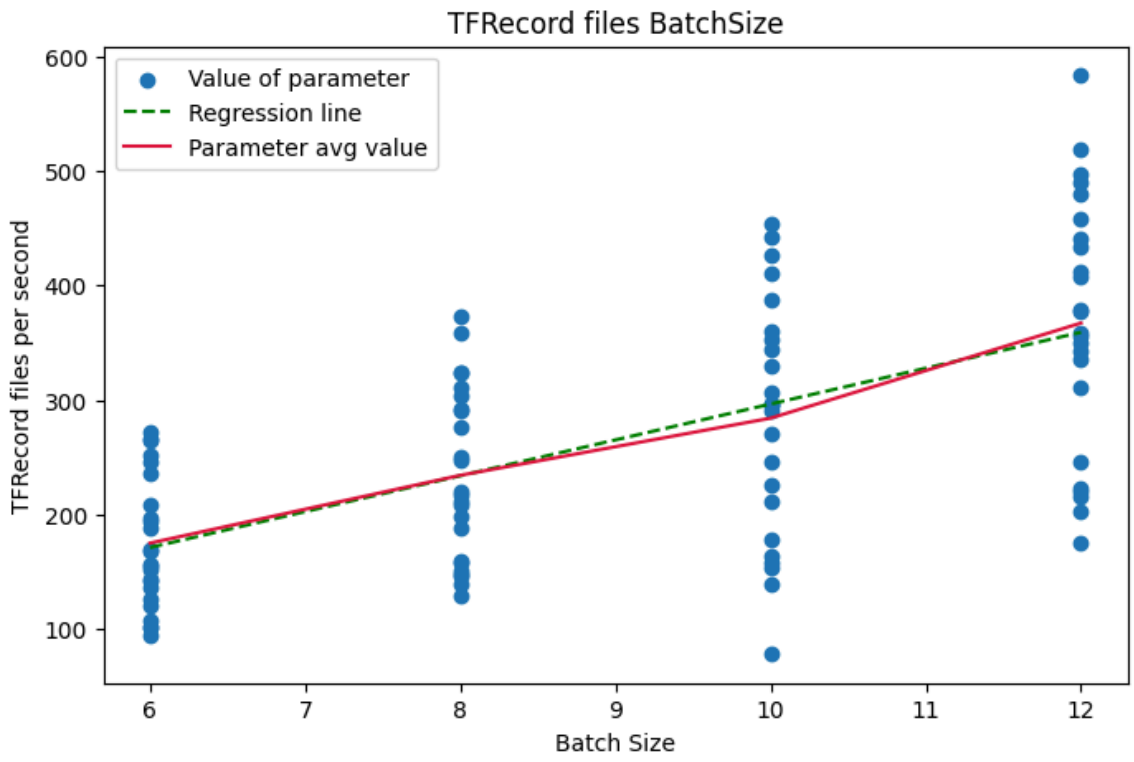
```
### CODING TASK ###
!gsutil cp $BUCKET/2c_results.pkl .
with open("2c_results.pkl",mode = 'rb') as f:
          tfr_batchSize_speed = pickle.load(f)
          tfr_batchNum_speed = pickle.load(f)
          tfr_repetitions_speed = pickle.load(f)
          tfr_dsSize_speed = pickle.load(f)
          image_batchSizes_speed = pickle.load(f)
          image_batchNums_speed = pickle.load(f)
          image_repetitions_speed = pickle.load(f)
          image_dsSize_speed = pickle.load(f)
          tfr_batchSize_avgSpeed = pickle.load(f)
          tfr_batchNums_avgSpeed = pickle.load(f)
          tfr_repetitions_avgSpeed = pickle.load(f)
          tfr_dsSize_avgSpeed = pickle.load(f)
          image_batchSizes_avgSpeed = pickle.load(f)
          image_batchNums_avgSpeed = pickle.load(f)
          image_repetitions_avgSpeed = pickle.load(f)
          image_dsSize_avgSpeed = pickle.load(f)
```

```
Copying gs://big-data-cw-420116-storage/2c_results.pkl...
/ [1 files][ 11.2 KiB/ 11.2 KiB]
Operation completed over 1 objects/11.2 KiB.
```

this error is because the job couldn't be saved previously.

In [ ]:

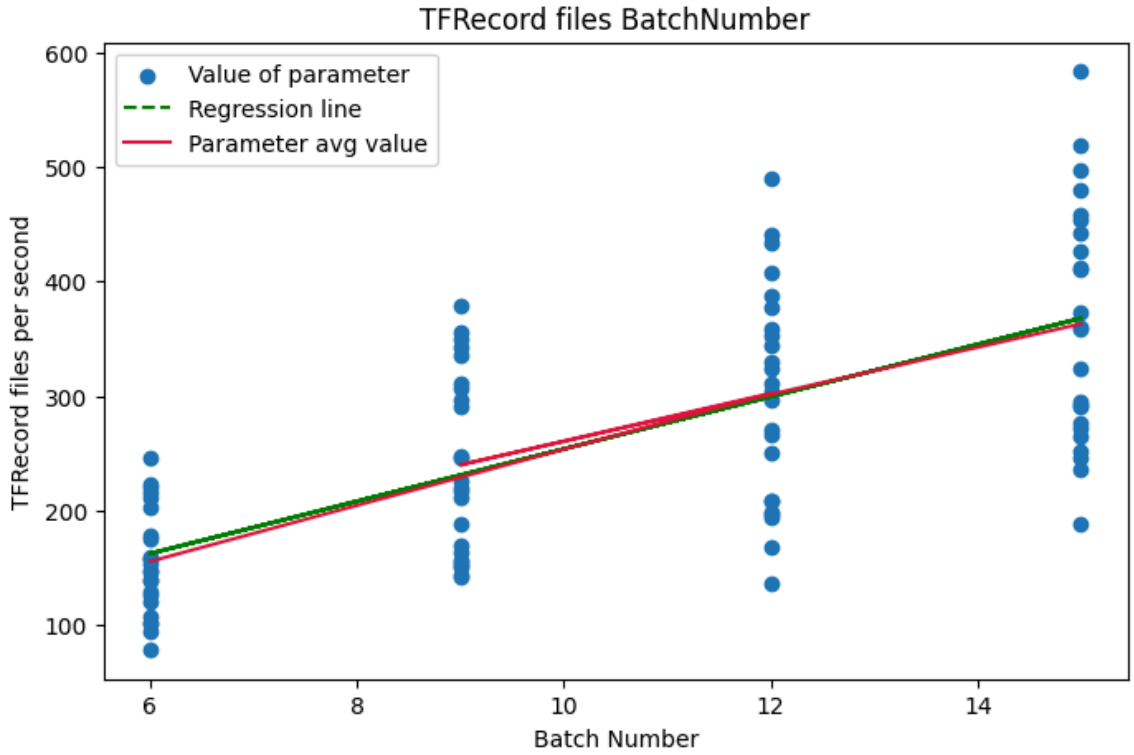
```
from sklearn.linear_model import LinearRegression

tfr_batchSize_speed_df = pd.DataFrame(tfr_batchSize_speed, columns=["ba
tfr_batchSize_avgSpeed_df = pd.DataFrame(tfr_batchSize_avgSpeed, column

linreg_tfr_batchSize_speed_df = LinearRegression()
linreg_tfr_batchSize_speed_df.fit(tfr_batchSize_speed_df[['batch_sizes'

tfr_batchSize_speed_predict = linreg_tfr_batchSize_speed_df.predict(tfr


fig = plt.figure(figsize=(8,5))
plt.scatter(tfr_batchSize_speed_df[['batch_sizes']], tfr_batchSize_spee
plt.plot(tfr_batchSize_speed_df[['batch_sizes']], tfr_batchSize_speed_p
plt.plot(tfr_batchSize_avgSpeed_df[['batch_sizes']],tfr_batchSize_avgSp

plt.title('TFRecord files BatchSize')
plt.legend(loc='upper left')
plt.ylabel('TFRecord files per second')
plt.xlabel('Batch Size')
plt.show()

print("Slope : ",linreg_tfr_batchSize_speed_df.coef_)
print("Intercept : ",linreg_tfr_batchSize_speed_df.intercept_)
print('P-value : ',linreg_tfr_batchSize_speed_df.score(tfr_batchSize_sp
```




```
Slope :  [31.31917417]
Intercept :  -16.682792340220544
P-value :  0.38712799159632494
```

In [ ]:

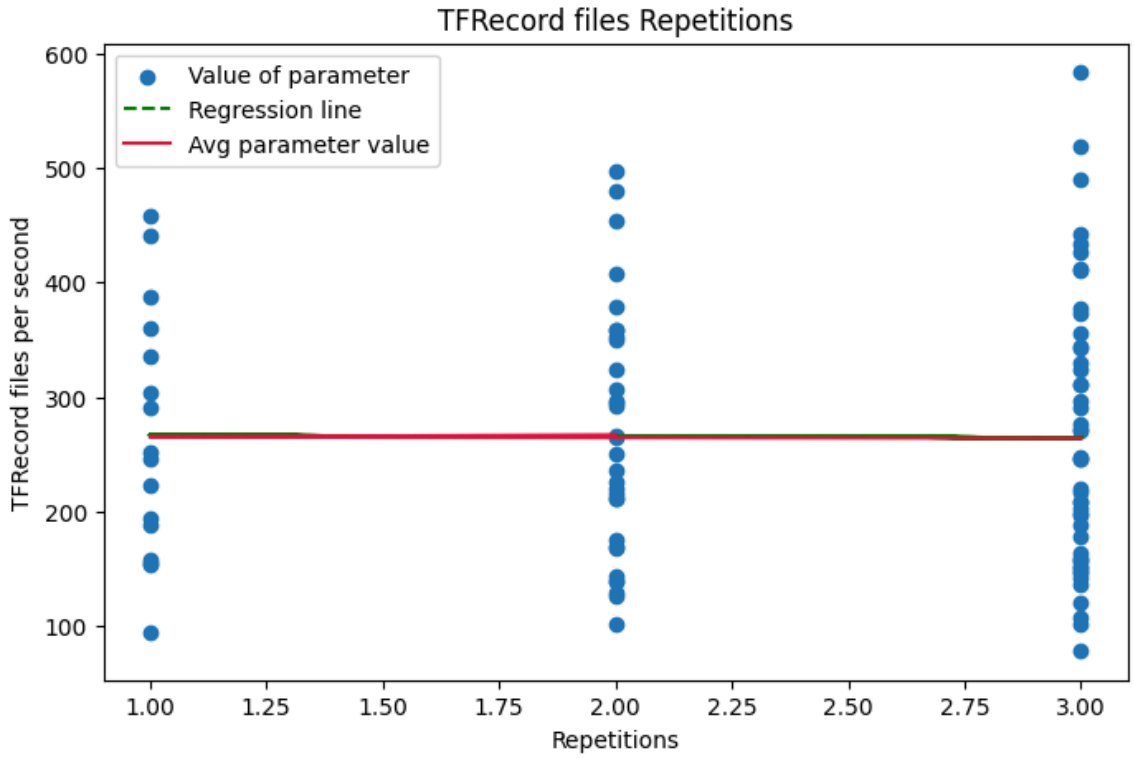
```
from sklearn.linear_model import LinearRegression

tfr_batchNum_speed_df = pd.DataFrame(tfr_batchNum_speed, columns=["batc
tfr_batchNums_avgSpeed_df = pd.DataFrame(tfr_batchNums_avgSpeed, column

linreg_tfr_batchNum_speed_df = LinearRegression()
linreg_tfr_batchNum_speed_df.fit(tfr_batchNum_speed_df[['batch_nums']],

tfr_batchNum_speed_predict = linreg_tfr_batchNum_speed_df.predict(tfr_b


fig = plt.figure(figsize=(8,5))
plt.scatter(tfr_batchNum_speed_df[['batch_nums']], tfr_batchNum_speed_d
plt.plot(tfr_batchNum_speed_df[['batch_nums']], tfr_batchNum_speed_pred
plt.plot(tfr_batchNums_avgSpeed_df[['batch_nums']],tfr_batchNums_avgSpe

plt.title('TFRecord files BatchNumber')
plt.legend(loc='upper left')
plt.ylabel('TFRecord files per second')
plt.xlabel('Batch Number')
plt.show()

print("Slope : ",linreg_tfr_batchNum_speed_df.coef_)
print("Intercept : ",linreg_tfr_batchNum_speed_df.intercept_)
print('P-value : ',linreg_tfr_batchNum_speed_df.score(tfr_batchNum_spee


```




```
Slope :  [22.83459662]
Intercept :  25.426510681410946
P-value :  0.4630236527439795
```

In [ ]:

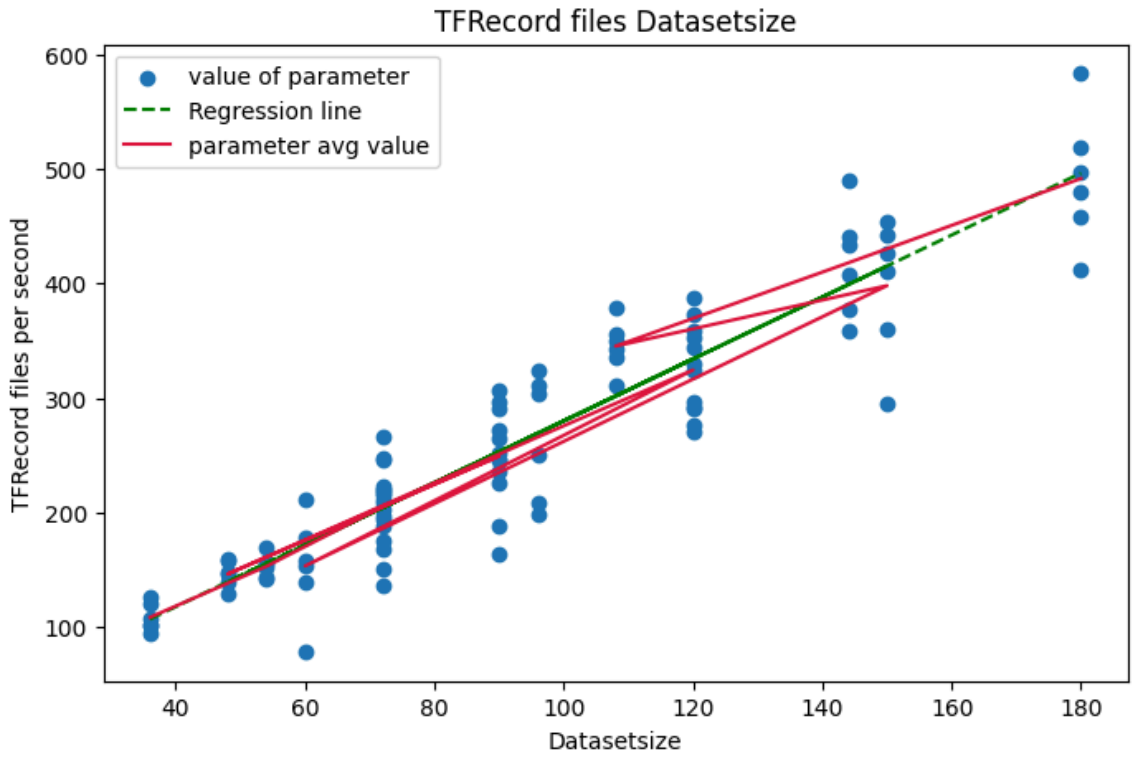
```
from sklearn.linear_model import LinearRegression

tfr_repetitions_speed_df = pd.DataFrame(tfr_repetitions_speed, columns=
tfr_repetitions_avgSpeed_df = pd.DataFrame(tfr_repetitions_avgSpeed, co

linreg_tfr_repetitions_speed_df = LinearRegression()
linreg_tfr_repetitions_speed_df.fit(tfr_repetitions_speed_df[['repetiti

tfr_repetitions_speed_predict = linreg_tfr_repetitions_speed_df.predict

fig = plt.figure(figsize=(8,5))
plt.scatter(tfr_repetitions_speed_df[['repetitions']], tfr_repetitions_
plt.plot(tfr_repetitions_speed_df[['repetitions']], tfr_repetitions_spe
plt.plot(tfr_repetitions_avgSpeed_df[['repetitions']],tfr_repetitions_a

plt.title('TFRecord files Repetitions')
plt.legend(loc='upper left')
plt.ylabel('TFRecord files per second')
plt.xlabel('Repetitions')
plt.show()

print("Slope : ",linreg_tfr_repetitions_speed_df.coef_)
print("Intercept : ",linreg_tfr_repetitions_speed_df.intercept_)
print('P-value : ',linreg_tfr_repetitions_speed_df.score(tfr_repetition
```




```
Slope :  [-1.01936299]
Intercept :  267.56828883248255
P-value :  4.556686860013315e-05
```

In [ ]:

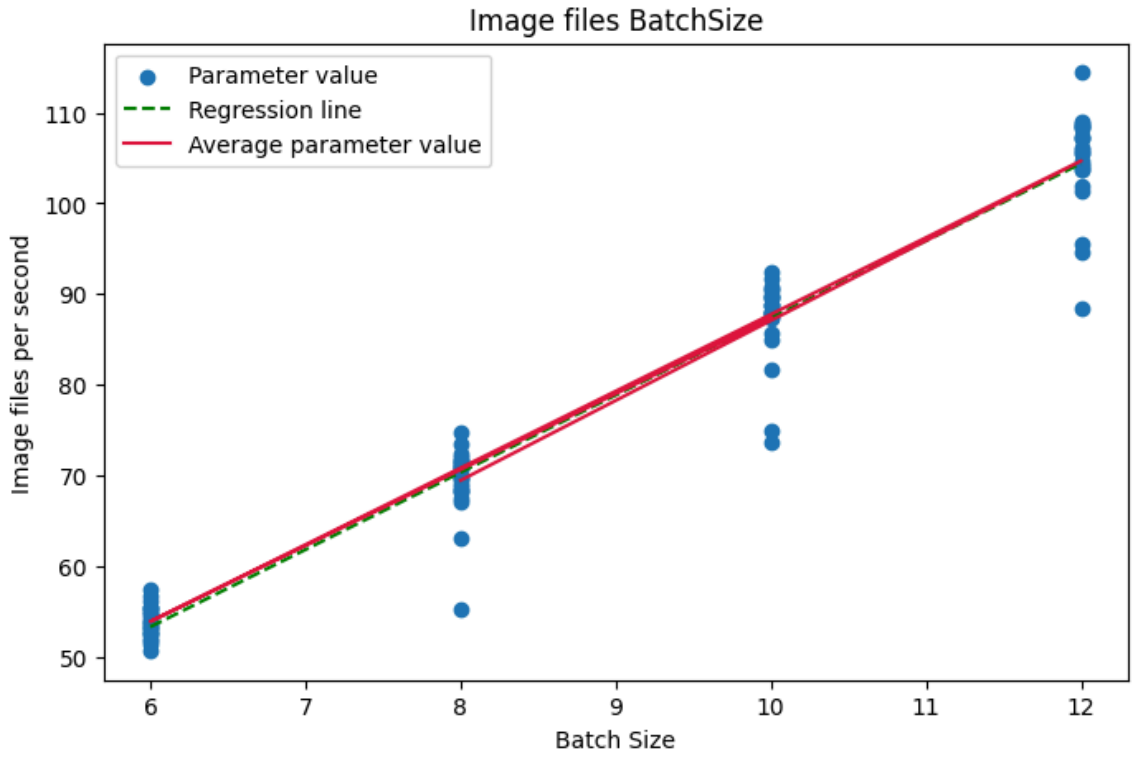
```
from sklearn.linear_model import LinearRegression

tfr_dsSize_speed_df = pd.DataFrame(tfr_dsSize_speed, columns=["datasets
tfr_dsSize_avgSpeed_df = pd.DataFrame(tfr_dsSize_avgSpeed, columns=["da

linreg_tfr_dsSize_speed_df = LinearRegression()
linreg_tfr_dsSize_speed_df.fit(tfr_dsSize_speed_df[['datasetsize']],tfr

tfr_dsSize_speed_predict = linreg_tfr_dsSize_speed_df.predict(tfr_dsSiz


fig = plt.figure(figsize=(8,5))
plt.scatter(tfr_dsSize_speed_df[['datasetsize']], tfr_dsSize_speed_df['
plt.plot(tfr_dsSize_speed_df[['datasetsize']], tfr_dsSize_speed_predict
plt.plot(tfr_dsSize_avgSpeed_df[['datasetsize']],tfr_dsSize_avgSpeed_df

plt.title('TFRecord files Datasetsize')
plt.legend(loc='upper left')
plt.ylabel('TFRecord files per second')
plt.xlabel('Datasetsize')
plt.show()

print("Slope : ",linreg_tfr_dsSize_speed_df.coef_)
print("Intercept : ",linreg_tfr_dsSize_speed_df.intercept_)
print('P-value : ',linreg_tfr_dsSize_speed_df.score(tfr_dsSize_speed_df
```




```
Slope :  [2.70422835]
Intercept :  9.640195688634037
P-value :  0.8766716882967958
```

In [ ]:

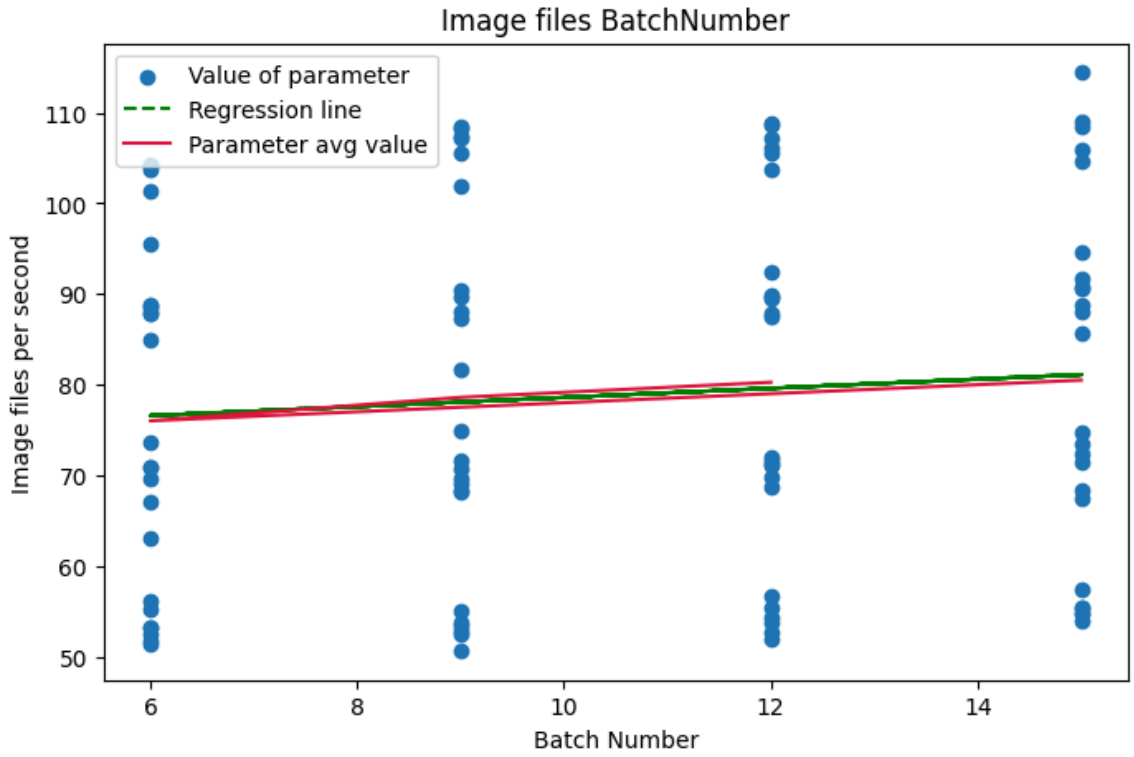
```
from sklearn.linear_model import LinearRegression

image_batchSizes_speed_df = pd.DataFrame(image_batchSizes_speed, column
image_batchSizes_avgSpeed_df = pd.DataFrame(image_batchSizes_avgSpeed,

linreg_image_batchSizes_speed_df = LinearRegression()
linreg_image_batchSizes_speed_df.fit(image_batchSizes_speed_df[['batch_

image_batchSizes_speed_predict = linreg_image_batchSizes_speed_df.predi

fig = plt.figure(figsize=(8,5))
plt.scatter(image_batchSizes_speed_df[['batch_sizes']], image_batchSize
plt.plot(image_batchSizes_speed_df[['batch_sizes']], image_batchSizes_s
plt.plot(image_batchSizes_avgSpeed_df[['batch_sizes']],image_batchSizes

plt.title('Image files BatchSize')
plt.legend(loc='upper left')
plt.ylabel('Image files per second')
plt.xlabel('Batch Size')
plt.show()

print("Slope : ",linreg_image_batchSizes_speed_df.coef_)
print("Intercept : ",linreg_image_batchSizes_speed_df.intercept_)
print('P-value : ',linreg_image_batchSizes_speed_df.score(image_batchSi
```




```
Slope :  [8.51630289]
Intercept :  2.20325651298333
P-value :  0.9558397732855116
```

In [ ]:

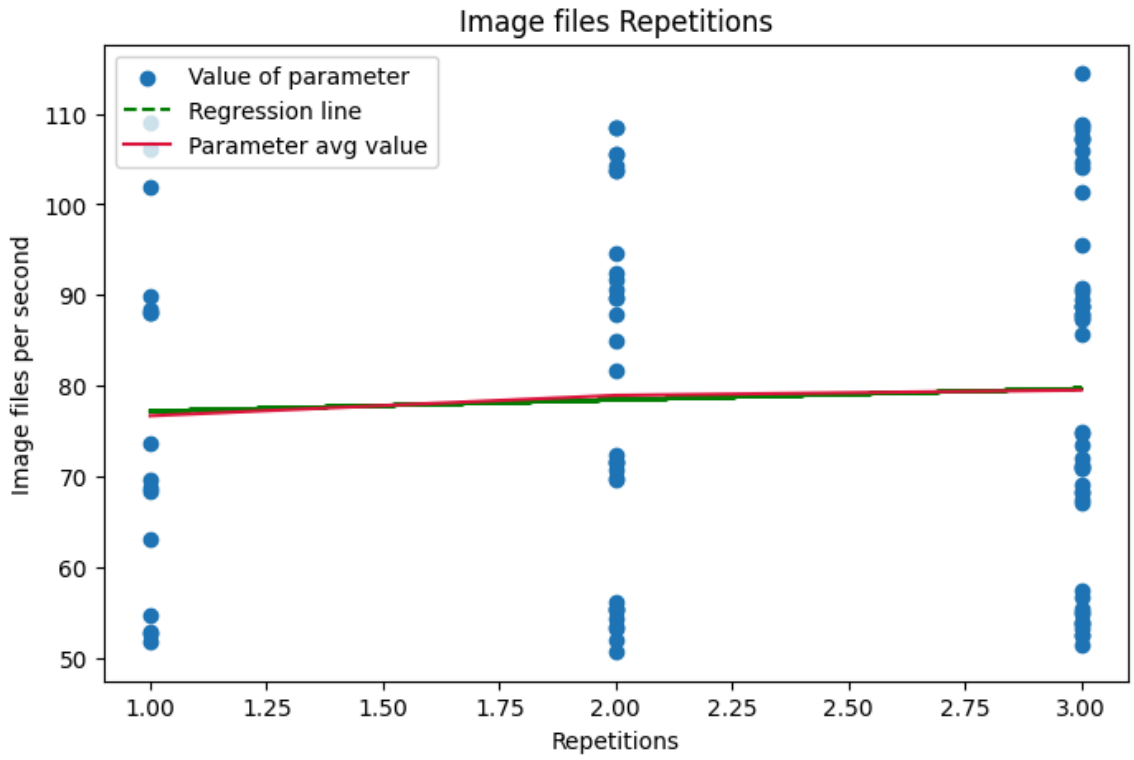
```
from sklearn.linear_model import LinearRegression

image_batchNums_speed_df = pd.DataFrame(image_batchNums_speed, columns=
image_batchNums_avgSpeed_df = pd.DataFrame(image_batchNums_avgSpeed, co

linreg_image_batchNums_speed_df = LinearRegression()
linreg_image_batchNums_speed_df.fit(image_batchNums_speed_df[['batch_nu

image_batchNums_speed_predict = linreg_image_batchNums_speed_df.predict

fig = plt.figure(figsize=(8,5))
plt.scatter(image_batchNums_speed_df[['batch_nums']], image_batchNums_s
plt.plot(image_batchNums_speed_df[['batch_nums']], image_batchNums_spee
plt.plot(image_batchNums_avgSpeed_df[['batch_nums']],image_batchNums_av

plt.title('Image files BatchNumber')
plt.legend(loc='upper left')
plt.ylabel('Image files per second')
plt.xlabel('Batch Number')
plt.show()

print("Slope : ",linreg_image_batchNums_speed_df.coef_)
print("Intercept : ",linreg_image_batchNums_speed_df.intercept_)
print('P-value : ',linreg_image_batchNums_speed_df.score(image_batchNum
```




```
Slope :  [0.50348735]
Intercept :  73.56336533975393
P-value :  0.0075169651996684905
```

In [ ]:

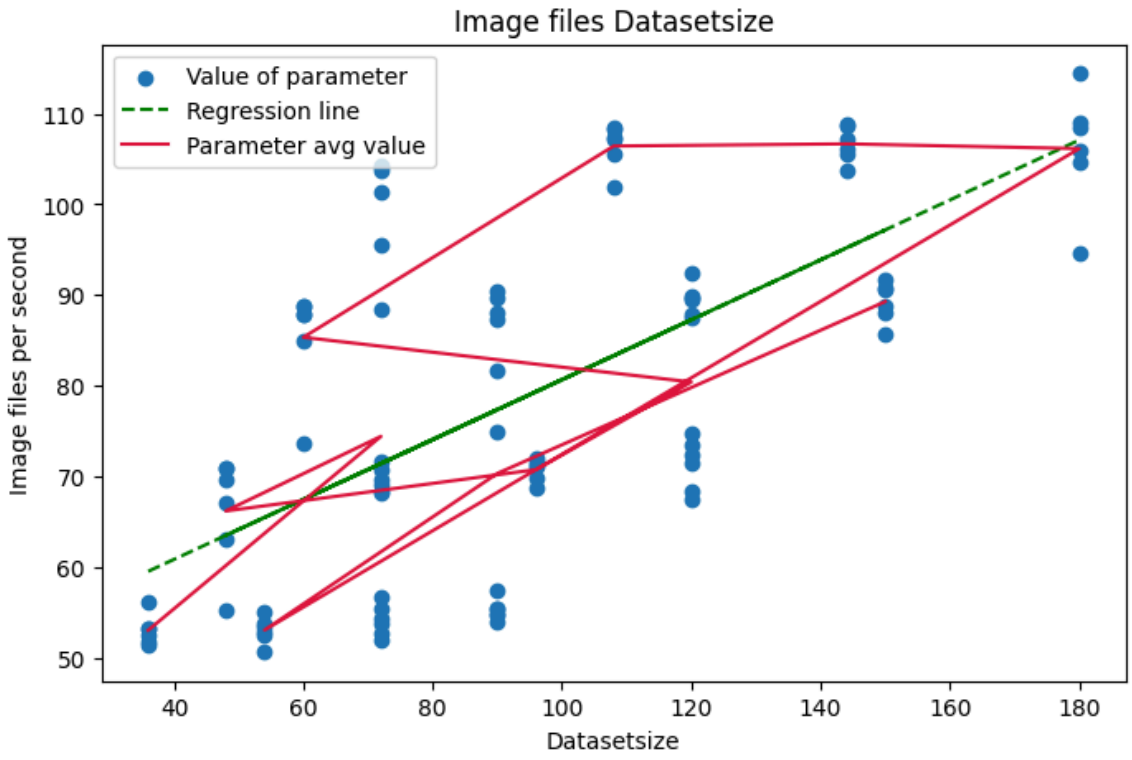
```
from sklearn.linear_model import LinearRegression

image_repetitions_speed_df = pd.DataFrame(image_repetitions_speed, colu
image_repetitions_avgSpeed_df = pd.DataFrame(image_repetitions_avgSpeed

linreg_image_repetitions_speed_df = LinearRegression()
linreg_image_repetitions_speed_df.fit(image_repetitions_speed_df[['repe

image_repetitions_speed_predict = linreg_image_repetitions_speed_df.pre


fig = plt.figure(figsize=(8,5))
plt.scatter(image_repetitions_speed_df[['repetitions']], image_repetiti
plt.plot(image_repetitions_speed_df[['repetitions']], image_repetitions
plt.plot(image_repetitions_avgSpeed_df[['repetitions']],image_repetitio

plt.title('Image files Repetitions')
plt.legend(loc='upper left')
plt.ylabel('Image files per second')
plt.xlabel('Repetitions')
plt.show()

print("Slope : ",linreg_image_repetitions_speed_df.coef_)
print("Intercept : ",linreg_image_repetitions_speed_df.intercept_)
print('P-value : ',linreg_image_repetitions_speed_df.score(image_repeti
```




```
Slope :  [1.24810677]
Intercept :  75.937733354247
P-value :  0.0022810975995531946
```

In [ ]:

```
from sklearn.linear_model import LinearRegression

image_dsSize_speed_df = pd.DataFrame(image_dsSize_speed, columns=["data
image_dsSize_avgSpeed_df = pd.DataFrame(image_dsSize_avgSpeed, columns=

linreg_image_dsSize_speed_df = LinearRegression()
linreg_image_dsSize_speed_df.fit(image_dsSize_speed_df[['datasetsize']]

image_dsSize_speed_predict = linreg_image_dsSize_speed_df.predict(image

fig = plt.figure(figsize=(8,5))

plt.scatter(image_dsSize_speed_df[['datasetsize']], image_dsSize_speed_
plt.plot(image_dsSize_speed_df[['datasetsize']], image_dsSize_speed_pre
plt.plot(image_dsSize_avgSpeed_df[['datasetsize']],image_dsSize_avgSpee

plt.title('Image files Datasetsize')
plt.legend(loc='upper left')
plt.ylabel('Image files per second')
plt.xlabel('Datasetsize')
plt.show()

print("Slope : ",linreg_image_dsSize_speed_df.coef_)
print("Intercept : ",linreg_image_dsSize_speed_df.intercept_)
print('P-value : ',linreg_image_dsSize_speed_df.score(image_dsSize_spee
```




```
Slope :  [0.33019057]
Intercept :  47.64697402103596
P-value :  0.43644397991167594
```

Section 3. Theoretical discussion

Task 3: Discussion in context. (24%)

In this task we refer an idea that is introduced in this paper:

- Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., & Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. (https://people.irisa.fr/Davide.Frey/wp-content/uploads/2018/02/cherrypick.pdf). In USENIX NSDI 17 (pp. 469-482).

Alipourfard et al (2017) introduce the prediction an optimal or near-optimal cloud configuration for a given compute task.

3a) Contextualise

Relate the previous tasks and the results to this concept. (It is not necessary to work through the full details of the paper, focus just on the main ideas). To what extent and under what conditions do the concepts and techniques in the paper apply to the task in this coursework? (12%)

3b) Strategise

Define - as far as possible - concrete strategies for different application scenarios (batch, stream) and discuss the general relationship with the concepts above. (12%)

Provide the answers to these questions in your report.

Final cleanup

Once you have finshed the work, you can delete the buckets, to stop incurring cost that depletes your credit.

In [ ]:
```
!gsutil -m rm -r $BUCKET/* # Empty your bucket
!gsutil rb $BUCKET # delete the bucket
```

```
Removing gs://big-data-cw-420116-storage/2aVI_results.pkl#1714822040801653...
Removing gs://big-data-cw-420116-storage/2b_results.pkl#1714823344636377...
Removing gs://big-data-cw-420116-storage/google-cloud-dataproc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5b0ef0b944ed255/staging/spark_write_tfrec.py#1714815915867542...
Removing gs://big-data-cw-420116-storage/google-cloud-dataproc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5b0ef0b944ed255/driveroutput.000000001#1714815935242074...
Removing gs://big-data-cw-420116-storage/google-cloud-dataproc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/big-data-cw-420116-cluster-m/dataproc-post-hdfs-startup-script_output#1714815870063135...
Removing gs://big-data-cw-420116-storage/google-cloud-dataproc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/jobs/e72d8ce747414b90b5b0ef0b944ed255/driveroutput.000000000#1714815935051527...
Removing gs://big-data-cw-420116-storage/google-cloud-dataproc-metainfo/084e9d65-6812-45a2-9cab-372f310ba794/cluster.properties#1714815751341348...
```

GO TO BUCKET, DELET WORKS: https://console.cloud.google.com/storage/browser?project=big-data-cw-420116&prefix=&forceOnBucketsSortingFiltering=true&pageState=(%22StorageBucketsTable%22:(%22f%22:%22%255B%255D%22,%22s%22:%5B(%22i%22:%22name%22,%22s%22:%220(https://console.cloud.google.com/storage/browser?project=big-data-cw-420116&prefix=&forceOnBucketsSortingFiltering=true&pageState=(%22StorageBucketsTable%22:(%22f%22:%22%255B%255D%22,%22s%22:%5B(%22i%22:%22name%22,%22s%22:%220