BD-CW

May 6, 2023

1 Student name, Student ID : Morteza Layegh Mirhosseini (220003166) - PG

2 Big Data Coursework - Questions

2.1 Data Processing and Machine Learning in the Cloud

This is the **INM432 Big Data coursework 2023**. This coursework contains extended elements of **theory** and **practice**, mainly around parallelisation of tasks withSpark and a bit about parallel training using TensorFlow.

2.2 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.

2.3 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**.

2.4 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* 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**.

2.4.1 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).

2.4.2 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.

2.4.3 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**.

2.4.4 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.

2.4.5 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 without needing the solution to the earlier ones.

3 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.

3.0.1 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.

[ ]:
```
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 2.12.0
```

3.0.2 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.

[ ]:
```
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.

[ ]:
```
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 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.

We also specify the **default project and region**. The REGION should be `us-central1` as that seems to be the only one that reliably works with the free credit. This way we don't have to specify this information every time we access the cloud.

[ ]:
```
PROJECT = 'my-project-220003166'  ### USE YOUR GOOGLE CLOUD PROJECT ID HERE.␣
 ↪###
!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].
Updated property [compute/region].
Updated property [dataproc/region].
[component_manager]
disable_update_check = True
[compute]
region = us-central1
[core]
account = Morteza.Layegh-Mirhosseini@city.ac.uk
project = my-project-220003166
[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.**

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

```
Creating gs://my-project-220003166-storage/…
ServiceException: 409 A Cloud Storage bucket named 'my-
project-220003166-storage' already exists. Try another name. Bucket names must
be globally unique across all Google Cloud projects, including those outside of
your organization.
```

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.

```
[ ]: #@title Utility functions for image display **[RUN THIS TO ACTIVATE]** {␣
     ↪display-mode: "form" }
     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 ' if␣
     ↪not correct else '',
                                   CLASSES[correct_label] if not correct else ''),␣
     ↪correct

     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()
```

3.0.3 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**.

[ ]:
```
%cd
!apt-get update -qq
!apt-get install openjdk-8-jdk-headless -qq >> /dev/null # send any output to␣
↪null device
!tar -xzf "/content/drive/My Drive/Big_Data/data/spark/spark-3.2.0-bin-hadoop2.
↪7.tgz" # unpack

!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/root/spark-3.2.0-bin-hadoop2.7"
import findspark
findspark.init()
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)
```

```
/root
3.2.0
<SparkContext master=local[*] appName=pyspark-shell>
```

4 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.

4.1 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**.

[ ]:
```
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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).

[ ]:
```
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images
partition_size = math.ceil(1.0 * nb_images / PARTITIONS) # images per partition␣
 ↪(float)
print("GCS_PATTERN matches {} images, to be divided into {} partitions with up␣
 ↪to {} images each.".format(nb_images, PARTITIONS, partition_size))
```

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

4.1.1 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.

[ ]:
```
def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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
```

```python
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 image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //␣
 ↪2, tw, th)
    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
```

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

[ ]:
```python
dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN) # This also shuffles the␣
 ↪images
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).

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

```
Output hidden; open in https://colab.research.google.com to view.
```

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

[ ]:
```python
sample_set = dsetResized.batch(10).take(10) # take 10 batches of 10 images for␣
 ↪testing
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', 'sunflowers', 'sunflowers',
'dandelion', 'dandelion', 'roses', 'roses', 'dandelion', 'roses', 'sunflowers'])
Image batch shape (10, 192, 192, 3), ['roses', 'dandelion', 'roses', 'daisy',
'daisy', 'dandelion', 'tulips', 'dandelion', 'dandelion', 'tulips'])
Image batch shape (10, 192, 192, 3), ['roses', 'tulips', 'dandelion', 'tulips',
'tulips', 'tulips', 'daisy', 'dandelion', 'daisy', 'dandelion'])
Image batch shape (10, 192, 192, 3), ['daisy', 'tulips', 'daisy', 'tulips',
'roses', 'dandelion', 'dandelion', 'daisy', 'tulips', 'tulips'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'dandelion', 'tulips',
'daisy', 'tulips', 'roses', 'roses', 'dandelion', 'tulips', 'tulips'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'tulips', 'daisy', 'daisy',
'dandelion', 'sunflowers', 'dandelion', 'tulips', 'tulips', 'tulips'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'tulips', 'tulips',
'dandelion', 'sunflowers', 'dandelion', 'roses', 'dandelion', 'roses',
'dandelion'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'dandelion', 'tulips',
'tulips', 'roses', 'dandelion', 'dandelion', 'sunflowers', 'tulips',
'dandelion'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'tulips', 'dandelion',
'roses', 'dandelion', 'sunflowers', 'dandelion', 'dandelion', 'dandelion',
'dandelion'])
Image batch shape (10, 192, 192, 3), ['daisy', 'tulips', 'tulips', 'dandelion',
'tulips', 'roses', 'daisy', 'sunflowers', 'roses', 'sunflowers'])
```

4.2 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**.

4.2.1 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**.

[ ]:
```
# 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.
 ↪decode('utf8') for lbl in label.numpy()]))
```

```
Image batch shape (10,), ['roses', 'dandelion', 'dandelion', 'daisy',
'sunflowers', 'dandelion', 'dandelion', 'tulips', 'daisy', 'roses'])
Image batch shape (10,), ['roses', 'dandelion', 'daisy', 'sunflowers',
'dandelion', 'sunflowers', 'tulips', 'dandelion', 'tulips', 'tulips'])
```

```
Image batch shape (10,), ['sunflowers', 'daisy', 'roses', 'dandelion',
'dandelion', 'sunflowers', 'dandelion', 'sunflowers', 'tulips', 'dandelion'])
Image batch shape (10,), ['tulips', 'tulips', 'daisy', 'dandelion', 'daisy',
'tulips', 'sunflowers', 'sunflowers', 'tulips', 'tulips'])
Image batch shape (10,), ['daisy', 'tulips', 'tulips', 'dandelion', 'tulips',
'roses', 'dandelion', 'dandelion', 'roses', 'daisy'])
Image batch shape (10,), ['dandelion', 'tulips', 'dandelion', 'daisy', 'tulips',
'daisy', 'roses', 'tulips', 'dandelion', 'sunflowers'])
Image batch shape (10,), ['roses', 'tulips', 'sunflowers', 'daisy',
'sunflowers', 'tulips', 'roses', 'sunflowers', 'tulips', 'dandelion'])
Image batch shape (10,), ['tulips', 'dandelion', 'tulips', 'sunflowers',
'daisy', 'daisy', 'dandelion', 'dandelion', 'roses', 'daisy'])
Image batch shape (10,), ['dandelion', 'roses', 'sunflowers', 'sunflowers',
'tulips', 'sunflowers', 'roses', 'tulips', 'sunflowers', 'dandelion'])
Image batch shape (10,), ['tulips', 'daisy', 'daisy', 'roses', 'sunflowers',
'roses', 'daisy', 'dandelion', 'sunflowers', 'dandelion'])
```

4.2.2 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.

[ ]:
```
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
↪file names
```

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.

[ ]:
```
# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will be a␣
↪list of size 1
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.
↪BytesList(value=list_of_bytestrings))

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

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order␣
↪defined in CLASSES)
```

```
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0, 0] for␣
↪class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
        "class": _int_feature([class_num]) #,      # one class in the list
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

def write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size): # write the images␣
↪to files.
    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 will be one␣
↪"batch" of images per file
    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,␣
↪partition_size)
        # You need to change GCS_OUTPUT to your own bucket to actually create␣
↪new files
        with tf.io.TFRecordWriter(filename) as out_file:
            for i in range(partition_size):
                example = to_tfrecord(out_file,
                                    image.numpy()[i], # re-compressed image:␣
↪already a byte string
                                    label.numpy()[i] #
                                    )
                out_file.write(example.SerializeToString())
        print("Wrote file {} containing {} records".format(filename,␣
↪partition_size))
    print("Total time: "+str(time.time()-tt0))

#  write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size) # uncomment to run␣
↪this cell
```

4.2.3 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.

[ ]:
```python
GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/'
# remove the line above to use your own files that you generated above

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # 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_deterministic = False
    # 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**.

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

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

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

4.3 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.

4.3.1 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' 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%)

[ ]:
```
### CODING TASK ###
%%time
#libraries
import os, sys, math
import numpy as np
```

```python
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

# initilizing SparkContext
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)


#Global Variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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']
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
↪file names



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

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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(row):
    image, label = row
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //␣
↪2, tw, th)
    return image, label

def recompress_image(row):
    image, label = row
    # 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) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD␣
↪that contains the list of image filenames. (3%)
file_paths = tf.io.gfile.glob(GCS_PATTERN)
### TASK 1d ###
#file_paths_rdd = sc.parallelize(file_paths,16)
file_paths_rdd = sc.parallelize(file_paths)

# apply the mapping functions to the file paths RDD
decoded_rdd = file_paths_rdd.map(decode_jpeg_and_label)
resized_rdd = decoded_rdd.map(resize_and_crop_image)
compressed_rdd = resized_rdd.map(recompress_image)

#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%)
compressed_rdd_sampled = compressed_rdd.sample(withReplacement=False,␣
↪fraction=0.02, seed=42)


# functions for writing TFRecord entries
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.
↪BytesList(value=list_of_bytestrings))
```

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

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


# #iv)Then use the functions from above to write the TFRecord files. (3%)
def write_tfrecord(partition_index, partition_iterator):
    filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
    with tf.io.TFRecordWriter(filename) as out_file:
        for x in partition_iterator:
            image = x[0]
            label = x[1]
            example = to_tfrecord(out_file,
                                  image.numpy(), # re-compressed image: already␣
↪a byte string
                                  label.numpy() #, height.numpy()[i], width.
↪numpy()[i]
                                  )
            out_file.write(example.SerializeToString())

    return [filename]

#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'␣
↪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%)

list_tfrecord_paths = compressed_rdd.mapPartitionsWithIndex(write_tfrecord).
↪collect()
```

```
Tensorflow version 2.12.0
3.2.0
<SparkContext master=local[*] appName=pyspark-shell>
CPU times: user 1.34 s, sys: 138 ms, total: 1.48 s
Wall time: 3min 21s
```

4.3.2 1b) Testing (3%)

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

[ ]:
```
### CODING TASK ###
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
↪file names

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string =␣
↪bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means␣
↪scalar
    }
    # 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_deterministic =␣
↪False
    # 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)
```

Output hidden; open in https://colab.research.google.com to view.

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.

[ ]:
```
### CODING TASK ###
%%writefile spark_write_tfrec.py
```

```
### CODING TASK ###
#uncomment the last line



#libraries
import tensorflow as tf
import math
import numpy as np
import time


#import findspark
#findspark.init()
# initilizing SparkContext
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)


#Global Variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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']
PROJECT = 'my-project-220003166'
REGION = 'us-central1'
CLUSTER = '{}-cluster'.format(PROJECT)
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
↪file names



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

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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
```

```python
def resize_and_crop_image(row):
    image, label = row
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //␣
 ↪2, tw, th)
    return image, label

def recompress_image(row):
    image, label = row
    # 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) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD␣
 ↪that contains the list of image filenames. (3%)
file_paths = tf.io.gfile.glob(GCS_PATTERN)
file_paths_rdd = sc.parallelize(file_paths)

# apply the mapping functions to the file paths RDD
decoded_rdd = file_paths_rdd.map(decode_jpeg_and_label)
resized_rdd = decoded_rdd.map(resize_and_crop_image)
compressed_rdd = resized_rdd.map(recompress_image)

#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%)
compressed_rdd_sampled = compressed_rdd.sample(withReplacement=False,␣
 ↪fraction=0.02, seed=42)
```

```python
#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'␣
↪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%)

# functions for writing TFRecord entries
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.
↪BytesList(value=list_of_bytestrings))

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

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


#iv)Then use the functions from above to write the TFRecord files. (3%)
def write_tfrecord(partition_index, partition_iterator):
    filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
    with tf.io.TFRecordWriter(filename) as out_file:
        for x in partition_iterator:
            image = x[0]
            label = x[1]
            example = to_tfrecord(out_file,
                                  image.numpy(), # re-compressed image: already␣
↪a byte string
                                  label.numpy() #, height.numpy()[i], width.
↪numpy()[i]
                                  )
            out_file.write(example.SerializeToString())

    return [filename]

#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'␣
↪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%)
list_tfrecord_paths = compressed_rdd.mapPartitionsWithIndex(write_tfrecord).
↪collect()
```

```
Writing spark_write_tfrec.py
```

[ ]:
```
### CODING TASK ###
%%time
%run ./spark_write_tfrec.py
```

```
3.2.0
<SparkContext master=local[*] appName=pyspark-shell>
CPU times: user 1.1 s, sys: 145 ms, total: 1.25 s
Wall time: 2min 38s
```

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

4.3.3 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%)

[ ]:
```
### CODING TASK ###
CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-central1'

#i) settting up a cluster with a single machine using the maximal SSD size␣
↪(100) and 8 vCPUs.
!gcloud dataproc clusters create $CLUSTER --region $REGION \
  --bucket $PROJECT-storage \
  --region $REGION \
  --image-version 1.5-ubuntu18 --single-node \
  --master-machine-type n1-standard-8 \
  --master-boot-disk-type pd-ssd \
```

```
  --master-boot-disk-size 100 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/
↪python/pip-install.sh \
  --metadata PIP_PACKAGES='tensorflow numpy' \
  --max-idle 3600s
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/7d6b1e6e-21ce-3198-bec9-0e9632fd3255].

WARNING: Don't create production clusters that reference
initialization actions located in the gs://goog-dataproc-initialization-actions-
REGION public buckets. These scripts are provided as reference implementations,
and they are synchronized with ongoing GitHub repository changes-a new version
of a initialization action in public buckets may break your cluster creation.
Instead, copy the following initialization actions from public buckets into your
bucket : gs://goog-dataproc-initialization-actions-us-central1/python/pip-
install.sh
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the staging_bucket
'projects/_/buckets/my-project-220003166-storage'. This usually happens when a
custom resource (ex: custom staging bucket) or a user-managed VM Service account
has been provided and the default/user-managed service account hasn't been
granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the temp_bucket
'projects/_/buckets/dataproc-temp-us-central1-812583826827-nqseqgr3'. This
usually happens when a custom resource (ex: custom staging bucket) or a user-
managed VM Service account has been provided and the default/user-managed
service account hasn't been granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
Created [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster]
Cluster placed in zone [us-central1-b].
```

Run the script in the cloud and test the output.

[ ]:
```
### CODING TASK ###
%%time
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
./spark_write_tfrec.py

#test the output
```

```
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
 ↪file names
filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
datasetTfrec = load_dataset(filenames)
display_9_images_from_dataset(datasetTfrec)
```

Output hidden; open in https://colab.research.google.com to view.

[ ]:
```
#delete the cluster
!gcloud dataproc clusters delete $CLUSTER --region=us-central1 -q
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/576a5324-1667-3b7d-86e9-17af36d19fc2].
Deleted [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster].
```

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 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%)

[ ]:
```
### CODING TASK ###
CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-central1'
#1 master and 7 worker nodes
!gcloud dataproc clusters create $CLUSTER --region $REGION \
  --bucket $PROJECT-storage \
  --region $REGION \
  --num-workers 7 \
  --worker-machine-type n1-standard-1 \
  --worker-boot-disk-size 285 \
  --worker-boot-disk-type pd-standard \
  --master-machine-type n1-standard-1 \
  --master-boot-disk-size 100 \
  --master-boot-disk-type pd-ssd \
  --image-version 1.4-ubuntu18 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/
 ↪python/pip-install.sh \
```

```
  --metadata 'PIP_PACKAGES=tensorflow' \
  --max-idle 3600s
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/d966cfcb-3565-3723-9e51-8601689c198c].

WARNING: Creating clusters using the n1-standard-1 machine type is
not recommended. Consider using a machine type with higher memory.
WARNING: Don't create production clusters that reference
initialization actions located in the gs://goog-dataproc-initialization-actions-
REGION public buckets. These scripts are provided as reference implementations,
and they are synchronized with ongoing GitHub repository changes-a new version
of a initialization action in public buckets may break your cluster creation.
Instead, copy the following initialization actions from public buckets into your
bucket : gs://goog-dataproc-initialization-actions-us-central1/python/pip-
install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend
provisioning 1TB or larger to ensure consistently high I/O performance. See
https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the staging_bucket
'projects/_/buckets/my-project-220003166-storage'. This usually happens when a
custom resource (ex: custom staging bucket) or a user-managed VM Service account
has been provided and the default/user-managed service account hasn't been
granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the temp_bucket
'projects/_/buckets/dataproc-temp-us-central1-812583826827-nqseqgr3'. This
usually happens when a custom resource (ex: custom staging bucket) or a user-
managed VM Service account has been provided and the default/user-managed
service account hasn't been granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
Created [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster]
Cluster placed in zone [us-central1-a].
```

[ ]:
```
%%time
# ruuning the job on the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
./spark_write_tfrec.py
```

```
Job [c0f702d48c614ab68dc004122a2e155a] submitted.
```

```
Waiting for job output…
2023-04-28 10:18:44.575216: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-28 10:18:44.575394: 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.
2.4.8
23/04/28 10:18:48 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/28 10:18:48 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/28 10:18:48 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/28 10:18:48 INFO org.spark_project.jetty.util.log: Logging initialized
@8856ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/28 10:18:48 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
23/04/28 10:18:48 INFO org.spark_project.jetty.server.Server: Started @9142ms
23/04/28 10:18:48 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@24703cc0{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
23/04/28 10:18:49 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair
Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set
spark.scheduler.allocation.file to a file that contains the configuration.
23/04/28 10:18:51 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.0.58:8032
23/04/28 10:18:51 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.0.58:10200
23/04/28 10:18:55 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682676719987_0001
<SparkContext master=yarn appName=spark_write_tfrec.py>
23/04/28 10:19:15 WARN org.apache.spark.scheduler.TaskSetManager: Stage 0
contains a task of very large size (134 KB). The maximum recommended task size
is 100 KB.
23/04/28 10:22:51 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@24703cc0{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [c0f702d48c614ab68dc004122a2e155a] finished successfully.
done: true
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/c0f702d48c614ab68dc004122a2e155a/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/c0f702d48c614ab68dc004122a2e155a/driveroutput
jobUuid: 9e00bc2f-744a-3e09-8b7b-c7a2bf730bc4
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: a86eda35-baae-4d34-9435-8eafa92799db
```

```
pysparkJob:
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/c0f702d48c614ab68dc004122a2e155a/staging/spark_write_tfrec.py
reference:
  jobId: c0f702d48c614ab68dc004122a2e155a
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-28T10:22:55.490545Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-28T10:18:36.355462Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-28T10:18:36.386030Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-28T10:18:36.790446Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682676719987_0001/
CPU times: user 2 s, sys: 287 ms, total: 2.29 s
Wall time: 4min 26s
```

[ ]:
```
#test the output
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
 ↪file names
filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
datasetTfrec = load_dataset(filenames)
display_9_images_from_dataset(datasetTfrec)
```

```
Output hidden; open in https://colab.research.google.com to view.
```

[ ]:
```
#delete the cluster
!gcloud dataproc clusters delete $CLUSTER --region=us-central1 -q
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/e6e3c26a-940a-3963-a27a-e50e8a5302d8].
Deleted [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster].
```

[ ]:
```
#!gcloud compute regions list
```

4.3.4 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.

i)Improve parallelisation

[ ]:
```
%%writefile spark_write_tfrec_Improve_parallelisation.py
### CODING TASK ###
#uncomment the last line




#libraries
import tensorflow as tf
import math
import numpy as np
import time


#import findspark
#findspark.init()
# initilizing SparkContext
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)


#Global Variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
```

```
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']
PROJECT = 'my-project-220003166'
REGION = 'us-central1'
CLUSTER = '{}-cluster'.format(PROJECT)
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output␣
↪file names



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

def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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(row):
    image, label = row
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //␣
↪2, tw, th)
    return image, label

def recompress_image(row):
    image, label = row
```

```python
    # 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) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD␣
 ↪that contains the list of image filenames. (3%)
file_paths = tf.io.gfile.glob(GCS_PATTERN)
### TASK 1d ###
file_paths_rdd = sc.parallelize(file_paths,16)

# apply the mapping functions to the file paths RDD
decoded_rdd = file_paths_rdd.map(decode_jpeg_and_label)
resized_rdd = decoded_rdd.map(resize_and_crop_image)
compressed_rdd = resized_rdd.map(recompress_image)

#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%)
compressed_rdd_sampled = compressed_rdd.sample(withReplacement=False,␣
 ↪fraction=0.02, seed=42)

#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'␣
 ↪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%)

# functions for writing TFRecord entries
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.
 ↪BytesList(value=list_of_bytestrings))

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

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order␣
 ↪defined in CLASSES)
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0, 0] for␣
 ↪class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
        "class": _int_feature([class_num]) #,      # one class in the list
```

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


#iv)Then use the functions from above to write the TFRecord files. (3%)
def write_tfrecord(partition_index, partition_iterator):
    filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
    with tf.io.TFRecordWriter(filename) as out_file:
        for x in partition_iterator:
            image = x[0]
            label = x[1]
            example = to_tfrecord(out_file,
                                  image.numpy(), # re-compressed image: already␣
 ↪a byte string
                                  label.numpy() #, height.numpy()[i], width.
 ↪numpy()[i]
                                  )
            out_file.write(example.SerializeToString())

    return [filename]

#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'␣
 ↪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%)
list_tfrecord_paths = compressed_rdd.mapPartitionsWithIndex(write_tfrecord).
 ↪collect()
```

```
Overwriting spark_write_tfrec_Improve_parallelisation.py
```

[ ]:
```
#  cluster information
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: my-project-220003166-cluster
clusterUuid: a86eda35-baae-4d34-9435-8eafa92799db
config:
  configBucket: my-project-220003166-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow
    networkUri: https://www.googleapis.com/compute/v1/projects/my-
project-220003166/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
```

```
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/my-
project-220003166/zones/us-central1-a
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-
central1/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2023-04-28T10:22:55.490545Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-
dataproc/global/images/dataproc-1-4-ubu18-20220125-170200-rc01
    instanceNames:
    - my-project-220003166-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/my-
project-220003166/zones/us-central1-a/machineTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.4.80-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy:
fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      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: '60'
      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: my-
```

```
project-220003166-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: '30'
      hdfs:dfs.namenode.servicerpc-address: my-project-220003166-cluster-m:8051
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '1000'
      mapred:mapreduce.job.maps: '60'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx819m
      mapred:mapreduce.map.memory.mb: '1024'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx819m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '1024'
      spark-env:SPARK_DAEMON_MEMORY: 1000m
      spark:spark.driver.maxResultSize: 480m
      spark:spark.driver.memory: 960m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners:
com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '1000'
      yarn:yarn.nodemanager.resource.cpu-vcores: '1'
      yarn:yarn.nodemanager.resource.memory-mb: '3072'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs:
'86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '3072'
      yarn:yarn.scheduler.minimum-allocation-mb: '256'
  tempBucket: dataproc-temp-us-central1-812583826827-nqseqgr3
  workerConfig:
    diskConfig:
      bootDiskSizeGb: 285
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-
dataproc/global/images/dataproc-1-4-ubu18-20220125-170200-rc01
    instanceNames:
```

```
    - my-project-220003166-cluster-w-0
    - my-project-220003166-cluster-w-1
    - my-project-220003166-cluster-w-2
    - my-project-220003166-cluster-w-3
    - my-project-220003166-cluster-w-4
    - my-project-220003166-cluster-w-5
    - my-project-220003166-cluster-w-6
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/my-
project-220003166/zones/us-central1-a/machineTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 7
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: my-project-220003166-cluster
  goog-dataproc-cluster-uuid: a86eda35-baae-4d34-9435-8eafa92799db
  goog-dataproc-location: us-central1
metrics:
  hdfsMetrics:
    dfs-blocks-corrupt: '0'
    dfs-blocks-missing: '0'
    dfs-blocks-missing-repl-one: '0'
    dfs-blocks-pending-deletion: '0'
    dfs-blocks-under-replication: '0'
    dfs-capacity-present: '2008414868959'
    dfs-capacity-remaining: '2008414461952'
    dfs-capacity-total: '2074787434496'
    dfs-capacity-used: '407007'
    dfs-nodes-decommissioned: '0'
    dfs-nodes-decommissioning: '0'
    dfs-nodes-running: '7'
  yarnMetrics:
    yarn-apps-completed: '1'
    yarn-apps-failed: '0'
    yarn-apps-killed: '0'
    yarn-apps-pending: '0'
    yarn-apps-running: '0'
    yarn-apps-submitted: '1'
    yarn-containers-allocated: '0'
    yarn-containers-pending: '0'
    yarn-containers-reserved: '0'
    yarn-memory-mb-allocated: '0'
    yarn-memory-mb-available: '21504'
    yarn-memory-mb-pending: '0'
    yarn-memory-mb-reserved: '0'
    yarn-memory-mb-total: '21504'
    yarn-nodes-active: '7'
    yarn-nodes-decommissioned: '0'
```

```
    yarn-nodes-lost: '0'
    yarn-nodes-rebooted: '0'
    yarn-nodes-unhealthy: '0'
    yarn-vcores-allocated: '0'
    yarn-vcores-available: '7'
    yarn-vcores-pending: '0'
    yarn-vcores-reserved: '0'
    yarn-vcores-total: '7'
projectId: my-project-220003166
status:
  state: RUNNING
  stateStartTime: '2023-04-28T10:14:44.234434Z'
statusHistory:
- state: CREATING
  stateStartTime: '2023-04-28T10:10:26.911659Z'
```

[ ]:

```
%%time
# ruuning the job on the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
./spark_write_tfrec_Improve_parallelisation.py
```

```
Job [2619d8a5d3944605920fd7f7a00874bc] submitted.
Waiting for job output…
2023-04-28 10:24:00.354429: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-28 10:24:00.354605: 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.
2.4.8
23/04/28 10:24:03 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/28 10:24:03 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/28 10:24:03 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/28 10:24:03 INFO org.spark_project.jetty.util.log: Logging initialized
@6731ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/28 10:24:03 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
23/04/28 10:24:03 INFO org.spark_project.jetty.server.Server: Started @6971ms
23/04/28 10:24:04 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@21d967bc{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
23/04/28 10:24:04 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair
Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set
spark.scheduler.allocation.file to a file that contains the configuration.
23/04/28 10:24:06 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.0.58:8032
```

```
23/04/28 10:24:06 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.0.58:10200
23/04/28 10:24:09 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682676719987_0002
<SparkContext master=yarn appName=spark_write_tfrec_Improve_parallelisation.py>
23/04/28 10:25:44 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@21d967bc{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [2619d8a5d3944605920fd7f7a00874bc] finished successfully.
done: true
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/2619d8a5d3944605920fd7f7a00874bc/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/2619d8a5d3944605920fd7f7a00874bc/driveroutput
jobUuid: 16604ad9-0514-3d8c-be07-28ecf089b375
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: a86eda35-baae-4d34-9435-8eafa92799db
pysparkJob:
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/a86eda35-baae-4d34-9435-
8eafa92799db/jobs/2619d8a5d3944605920fd7f7a00874bc/staging/spark_write_tfrec_Imp
rove_parallelisation.py
reference:
  jobId: 2619d8a5d3944605920fd7f7a00874bc
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-28T10:25:45.659707Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-28T10:23:55.655594Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-28T10:23:55.690145Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-28T10:23:55.902539Z'
yarnApplications:
- name: spark_write_tfrec_Improve_parallelisation.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682676719987_0002/
CPU times: user 884 ms, sys: 124 ms, total: 1.01 s
Wall time: 1min 57s
```

ii) Experiment with cluster configurations.

```
[ ]: #4 machines with double the resources each (2 vCPUs, memory, disk), (one␣
     ↪master, three workers)
     CLUSTER = '{}-cluster'.format(PROJECT)
     REGION = 'us-central1'

     !gcloud dataproc clusters create $CLUSTER --region $REGION \
       --bucket $PROJECT-storage \
       --region $REGION \
       --num-workers 3 \
       --worker-machine-type n1-standard-2 \
       --worker-boot-disk-size 666 \
       --worker-boot-disk-type pd-standard \
       --master-machine-type n1-standard-2 \
       --master-boot-disk-size 100 \
       --master-boot-disk-type pd-ssd \
       --image-version 1.4-ubuntu18 \
       --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/
     ↪python/pip-install.sh \
       --metadata 'PIP_PACKAGES=tensorflow' \
       --max-idle 3600s
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/c983603d-078a-3cdb-a887-c914d6821b73].

WARNING: Don't create production clusters that reference
initialization actions located in the gs://goog-dataproc-initialization-actions-
REGION public buckets. These scripts are provided as reference implementations,
and they are synchronized with ongoing GitHub repository changes-a new version
of a initialization action in public buckets may break your cluster creation.
Instead, copy the following initialization actions from public buckets into your
bucket : gs://goog-dataproc-initialization-actions-us-central1/python/pip-
install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend
provisioning 1TB or larger to ensure consistently high I/O performance. See
https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the staging_bucket
'projects/_/buckets/my-project-220003166-storage'. This usually happens when a
custom resource (ex: custom staging bucket) or a user-managed VM Service account
has been provided and the default/user-managed service account hasn't been
granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
WARNING: Permissions are missing for the default service account
```

```
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the temp_bucket
'projects/_/buckets/dataproc-temp-us-central1-812583826827-nqseqgr3'. This
usually happens when a custom resource (ex: custom staging bucket) or a user-
managed VM Service account has been provided and the default/user-managed
service account hasn't been granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
Created [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster]
Cluster placed in zone [us-central1-a].
```

[ ]:
```
%%time
# ruuning the job on the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
./spark_write_tfrec_Improve_parallelisation.py
```

```
Job [d27577fe62894ca2ba1e8855eaaa7597] submitted.
Waiting for job output…
2023-04-28 10:49:27.917623: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-28 10:49:27.917678: 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.
2.4.8
23/04/28 10:49:31 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/28 10:49:31 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/28 10:49:31 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/28 10:49:31 INFO org.spark_project.jetty.util.log: Logging initialized
@7501ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/28 10:49:31 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
23/04/28 10:49:31 INFO org.spark_project.jetty.server.Server: Started @7718ms
23/04/28 10:49:32 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@5210a8df{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
23/04/28 10:49:32 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair
Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set
spark.scheduler.allocation.file to a file that contains the configuration.
23/04/28 10:49:33 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.15.196:8032
23/04/28 10:49:33 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.15.196:10200
23/04/28 10:49:37 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682678653359_0001
```

```
<SparkContext master=yarn appName=spark_write_tfrec_Improve_parallelisation.py>
23/04/28 10:51:33 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@5210a8df{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [d27577fe62894ca2ba1e8855eaaa7597] finished successfully.
done: true
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/26e681c6-653f-4660-b39f-
f9f01f6ee152/jobs/d27577fe62894ca2ba1e8855eaaa7597/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/26e681c6-653f-4660-b39f-
f9f01f6ee152/jobs/d27577fe62894ca2ba1e8855eaaa7597/driveroutput
jobUuid: d4900bb1-5cf6-316c-b710-c7fa9c9cb53e
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: 26e681c6-653f-4660-b39f-f9f01f6ee152
pysparkJob:
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/26e681c6-653f-4660-b39f-
f9f01f6ee152/jobs/d27577fe62894ca2ba1e8855eaaa7597/staging/spark_write_tfrec_Imp
rove_parallelisation.py
reference:
  jobId: d27577fe62894ca2ba1e8855eaaa7597
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-28T10:51:36.178094Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-28T10:49:22.151166Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-28T10:49:22.188854Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-28T10:49:22.529416Z'
yarnApplications:
- name: spark_write_tfrec_Improve_parallelisation.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682678653359_0001/
CPU times: user 1.12 s, sys: 148 ms, total: 1.26 s
Wall time: 2min 21s
```

[ ]:

```
#delete the cluster
!gcloud dataproc clusters delete $CLUSTER --region=us-central1 -q
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/b84c3831-2410-39e8-b7c0-c6812847ba55].
```

```
Deleted [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster].
```

[ ]:
```
# one machine with eightfold resources.

CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-central1'

!gcloud dataproc clusters create $CLUSTER --region $REGION \
  --bucket $PROJECT-storage \
  --region $REGION \
  --master-machine-type n1-standard-8 --single-node \
  --master-boot-disk-type pd-ssd \
  --master-boot-disk-size 100 \
  --image-version 1.4-ubuntu18 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/
↪python/pip-install.sh \
  --metadata PIP_PACKAGES='tensorflow' \
  --max-idle 3600s
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/c1e57501-ac38-3f85-8808-80ecaff66212].

WARNING: Don't create production clusters that reference
initialization actions located in the gs://goog-dataproc-initialization-actions-
REGION public buckets. These scripts are provided as reference implementations,
and they are synchronized with ongoing GitHub repository changes-a new version
of a initialization action in public buckets may break your cluster creation.
Instead, copy the following initialization actions from public buckets into your
bucket : gs://goog-dataproc-initialization-actions-us-central1/python/pip-
install.sh
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the staging_bucket
'projects/_/buckets/my-project-220003166-storage'. This usually happens when a
custom resource (ex: custom staging bucket) or a user-managed VM Service account
has been provided and the default/user-managed service account hasn't been
granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the temp_bucket
'projects/_/buckets/dataproc-temp-us-central1-812583826827-nqseqgr3'. This
usually happens when a custom resource (ex: custom staging bucket) or a user-
managed VM Service account has been provided and the default/user-managed
service account hasn't been granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
```

```
accounts#VM_service_account.
Created [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster]
Cluster placed in zone [us-central1-b].
```

[ ]:
```
%%time
# ruuning the job on the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
./spark_write_tfrec_Improve_parallelisation.py
```

```
Job [c4c0aac796254d9abf5c09e35a468dba] submitted.
Waiting for job output…
2023-04-28 11:13:20.204930: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-28 11:13:20.204970: 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.
2.4.8
23/04/28 11:13:22 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/28 11:13:22 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/28 11:13:22 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/28 11:13:23 INFO org.spark_project.jetty.util.log: Logging initialized
@5290ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/28 11:13:23 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_312-b07
23/04/28 11:13:23 INFO org.spark_project.jetty.server.Server: Started @5407ms
23/04/28 11:13:23 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@204d89e4{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
23/04/28 11:13:23 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair
Scheduler configuration file not found so jobs will be scheduled in FIFO order.
To use fair scheduling, configure pools in fairscheduler.xml or set
spark.scheduler.allocation.file to a file that contains the configuration.
23/04/28 11:13:24 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.15.198:8032
23/04/28 11:13:24 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.15.198:10200
23/04/28 11:13:27 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682679715924_0001
<SparkContext master=yarn appName=spark_write_tfrec_Improve_parallelisation.py>
23/04/28 11:15:12 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@204d89e4{HTTP/1.1, (http/1.1)}{0.0.0.0:4040}
Job [c4c0aac796254d9abf5c09e35a468dba] finished successfully.
done: true
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/0fe866c3-9ece-49d1-8e51-
```

5150d9836405/jobs/c4c0aac796254d9abf5c09e35a468dba/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/0fe866c3-9ece-49d1-8e51-
5150d9836405/jobs/c4c0aac796254d9abf5c09e35a468dba/driveroutput
jobUuid: 658077f6-bb95-3aa7-b45a-37e50a473a3d
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: 0fe866c3-9ece-49d1-8e51-5150d9836405
pysparkJob:
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/0fe866c3-9ece-49d1-8e51-
5150d9836405/jobs/c4c0aac796254d9abf5c09e35a468dba/staging/spark_write_tfrec_Imp
rove_parallelisation.py
reference:
  jobId: c4c0aac796254d9abf5c09e35a468dba
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-28T11:15:17.536871Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-28T11:13:15.754728Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-28T11:13:15.780790Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-28T11:13:16.085569Z'
yarnApplications:
- name: spark_write_tfrec_Improve_parallelisation.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682679715924_0001/
```
CPU times: user 1 s, sys: 140 ms, total: 1.14 s
Wall time: 2min 10s
```

5 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**.

5.1 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.

[ ]:
```
# Here are some useful values for testing your code, use higher values later␣
 ↪for actually testing throughput
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 output␣
 ↪without showing it
        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.numpy().
 ↪shape),
                        print("Image batch shape {}, {})".format(image.numpy().
 ↪shape,
                            [str(lbl) for lbl in label.numpy()]), null_file)
                    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).

[ ]:
```
[res,par] = time_configs(dataset4, batch_sizes, batch_numbers, repetitions)
print(res)
print(par)

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

[res,par] = time_configs(datasetTfrec, batch_sizes, batch_numbers, repetitions)
print(res)
print(par)
```

```
[2, 2, 1]
(2, 2, 1)
bs: 2, ds: 3, rep: 1
Image batch shape (2,), ["b'tulips'", "b'daisy'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'daisy'", "b'dandelion'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'roses'", "b'dandelion'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'dandelion'", "b'roses'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'daisy'", "b'sunflowers'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'tulips'", "b'daisy'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'sunflowers'", "b'dandelion'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2,), ["b'tulips'", "b'sunflowers'"]) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4,), ["b'sunflowers'", "b'dandelion'", "b'dandelion'",
"b'sunflowers'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'daisy'", "b'daisy'", "b'tulips'", "b'sunflowers'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'sunflowers'", "b'sunflowers'", "b'dandelion'",
"b'roses'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4,), ["b'daisy'", "b'tulips'", "b'daisy'", "b'dandelion'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'sunflowers'", "b'dandelion'",
"b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'dandelion'", "b'sunflowers'", "b'dandelion'",
```

```
"b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'roses'", "b'tulips'", "b'tulips'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'daisy'", "b'dandelion'", "b'tulips'",
"b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'tulips'", "b'roses'", "b'dandelion'"])
<_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
total time: 12.384223937988281
[[[2.13512339]
  [5.1338729 ]]

 [[4.34217945]
  [5.37657555]]]
[[[[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), ['1', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['3', '1']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['1', '2']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2, 192, 192, 3), ['1', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['3', '1']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['1', '2']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['4', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['4', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['3', '0']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4, 192, 192, 3), ['1', '3', '3', '1']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['1', '2', '4', '3']) <_io.TextIOWrapper
```

```
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['4', '3', '3', '0']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4, 192, 192, 3), ['1', '3', '3', '1']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['1', '2', '4', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['4', '3', '3', '0']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['3', '4', '2', '2']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['3', '2', '0', '3']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4, 192, 192, 3), ['4', '4', '4', '1']) <_io.TextIOWrapper
name='/dev/null' mode='w' encoding='UTF-8'>
total time: 5.790587425231934
[[[ 2.32612875]
  [ 9.26244735]]

 [[ 9.97444897]
  [34.19571565]]]
[[[[2. 3. 1.]]

  [[2. 6. 1.]]]


 [[[4. 3. 1.]]

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

5.2 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).

5.2.1 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.

[ ]:
```
### CODING TASK
# import required libraries
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row
from pyspark.sql import SparkSession
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


# # import required libraries for clouad
# import pyspark
# from pyspark.sql import SQLContext
# from pyspark.sql import Row
# from pyspark.sql import SparkSession
# import os, sys, math
# import numpy as np
# import time
# import datetime
# import string
# import random
# import tensorflow as tf
# print("Tensorflow version " + tf.__version__)
# import pickle
# import argparse



#parameters
```

```python
PROJECT = 'my-project-220003166'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/'
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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']
nb_images = len(tf.io.gfile.glob(GCS_PATTERN))


def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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 image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //
 ↪2, tw, th)
    return image, label


def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string =
 ↪bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means
 ↪scalar
    }
```

```python
    # 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_deterministic =␣
 ↪False
    # 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

def load_dataset_images():
  dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN)
  dsetDecoded = dsetFiles.map(decode_jpeg_and_label)
  dsetResized = dsetDecoded.map(resize_and_crop_image)
  return  dsetResized

# Adapted function for tf record files
def time_configs_TFRecord(parameters_rdd):

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

    filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
    dataset = load_dataset(filenames)
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
```

```python
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
 ↪reading_speed, throughput])

    return measure


def time_configs_images(parameters_rdd):

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

    dataset = load_dataset_images()
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
 ↪reading_speed, throughput])

    return measure


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=subprocess.
 ↪PIPE)
    print("gstuil returned: " + str(proc.returncode))
    print(str(proc.stderr))

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

```
def speed_test(argv):        #function is Adapted from notebook Dr Galkin␣
↪provided about how to save output on the bucket
  # Parse the provided arguments
  print(argv)
  parser = argparse.ArgumentParser() # get a parser object
  parser.add_argument('--out_bucket', metavar='out_bucket', required=True,
                      help='The bucket URL for the result.') # add a required␣
↪argument
  parser.add_argument('--out_file', metavar='out_file', required=True,
                      help='The filename for the result.') # add a required␣
↪argument
  args = parser.parse_args(argv) # read the value

  # 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%)


  batch_sizes = [2,4] # toy parameters for testing
  batch_numbers = [3,6]
  repetitions = [1]

  # batch_sizes = [8,16,32,64,128,256] # real parameters used in final script␣
↪and retrieve the results
  # batch_numbers = [10,20,30,40]
  # repetitions = [1, 2, 3]


  params = []
  for batch_size in batch_sizes:
      for batch_number in batch_numbers:
          for repetition in repetitions:
              params.append([batch_size, batch_number, repetition])

  column_names = ["batch_sizes", "batch_numbers", "repetitions",␣
↪"dataset_size", "reading_speed", "throughput"]



  # Creating a Spark context
  sc = pyspark.SparkContext.getOrCreate()
  # Creating an RDD for parameter combinations
  partition_num = len(params)
  param_combinations_rdd = sc.parallelize(params,partition_num)
  # Creating a Spark session for converting results into dataframes
```

```
ss_TF_params_time = SparkSession(sc)
# Applying each parameter combination to a time measurement function to␣
↪determine reading speed and throughput in images per second
TF_params_time = param_combinations_rdd.flatMap(time_configs_TFRecord)
### TASK 2c ###
#TF_params_time.cache()
# Creating dataframes
df_TF_params_time = TF_params_time.toDF(column_names)


# Creating a Spark context
sc = pyspark.SparkContext.getOrCreate()
# Creating an RDD for parameter combinations
partition_num = len(params)
param_combinations_rdd = sc.parallelize(params,partition_num)
# Creating a Spark session for converting results into dataframes
ss_img_files = SparkSession(sc)
# Applying each parameter combination to a time measurement function to␣
↪determine reading speed and throughput in images per second
images_params_time = param_combinations_rdd.flatMap(time_configs_images)
### TASK 2c ###
#images_params_time.cache()
# Creating dataframes
df_images_params_time = images_params_time.toDF(column_names)


# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
TF_params_time_array = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                                    ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
images_params_time_array = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                                    ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iv) create an RDD with all results for each parameter as␣
↪(parameter_value,images_per_second) and collect the result for each␣
↪parameter (1%)

# RDD for batch sizes and images_per_second (TFrecords files)
```

```
TFrecord_batchSizes_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
TFrecord_batchSizes_throughput = TFrecord_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (TFrecords files)
TFrecord_batchNumbers_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
TFrecord_batchNumbers_throughput = TFrecord_batchNumbers_throughput_rdd.
↪collect()

# RDD for repetitions and images_per_second (TFrecords files)
TFrecord_repetitions_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
TFrecord_repetitions_throughput = TFrecord_repetitions_throughput_rdd.
↪collect()

# RDD for dataset size and images_per_second (TFrecords files)
TFrecord_datasetSize_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
TFrecord_datasetSize_throughput = TFrecord_datasetSize_throughput_rdd.
↪collect()

# RDD for batch sizes and images_per_second (Images)
images_batchSizes_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
images_batchSizes_throughput = images_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (Images)
images_batchNumbers_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
images_batchNumbers_throughput = images_batchNumbers_throughput_rdd.collect()

# RDD for repetitions and images_per_second (Images)
images_repetitions_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
images_repetitions_throughput = images_repetitions_throughput_rdd.collect()

# RDD for dataset size and images_per_second (Images)
images_datasetSize_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
images_datasetSize_throughput = images_datasetSize_throughput_rdd.collect()

␣
↪############################################################################################
```

```python
# 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%)

# RDD for batch size and average images_per_second (TFrecord files)
TFrecord_batchSizes_avg_throughput_rdd = TFrecord_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_batchSizes_avg_throughput = TFrecord_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (TFrecord files)
TFrecord_batchNumbers_avg_throughput_rdd =␣
↪TFrecord_batchNumbers_throughput_rdd.mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_batchNumbers_avg_throughput =␣
↪TFrecord_batchNumbers_avg_throughput_rdd.collect()


# RDD for repetitions and average images_per_second (TFrecord files)
TFrecord_repetitions_avg_throughput_rdd = TFrecord_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_repetitions_avg_throughput = TFrecord_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (TFrecord files)
TFrecord_datasetSize_avg_throughput_rdd = TFrecord_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_datasetSize_avg_throughput = TFrecord_datasetSize_avg_throughput_rdd.
↪collect()
```

```
# RDD for batch size and average images_per_second (images)
images_batchSizes_avg_throughput_rdd = images_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                        .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                        .mapValues(lambda z: z[0]/
↪z[1])
images_batchSizes_avg_throughput = images_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (images)
images_batchNumbers_avg_throughput_rdd = images_batchNumbers_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                        .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                        .mapValues(lambda z: z[0]/
↪z[1])
images_batchNumbers_avg_throughput = images_batchNumbers_avg_throughput_rdd.
↪collect()


# RDD for repetitions and average images_per_second (images)
images_repetitions_avg_throughput_rdd = images_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                        .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                        .mapValues(lambda z: z[0]/
↪z[1])
images_repetitions_avg_throughput = images_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (images)
images_datasetSize_avg_throughput_rdd = images_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                        .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                        .mapValues(lambda z: z[0]/
↪z[1])
images_datasetSize_avg_throughput = images_datasetSize_avg_throughput_rdd.
↪collect()

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

```
  save_object = (TFrecord_batchSizes_throughput,
  TFrecord_batchNumbers_throughput,
  TFrecord_repetitions_throughput,
  TFrecord_datasetSize_throughput,
  images_batchSizes_throughput,
  images_batchNumbers_throughput,
  images_repetitions_throughput,
  images_datasetSize_throughput,
  TFrecord_batchSizes_avg_throughput,
  TFrecord_batchNumbers_avg_throughput,
  TFrecord_repetitions_avg_throughput,
  TFrecord_datasetSize_avg_throughput,
  images_batchSizes_avg_throughput,
  images_batchNumbers_avg_throughput,
  images_repetitions_avg_throughput,
  images_datasetSize_avg_throughput,)


  # save tuple of all parameter results
  save(save_object, args.out_bucket, args.out_file)



# Create a filename with the current date and time
now = datetime.datetime.now().strftime("%y%m%d-%H%M")
FILENAME = f'task_2b_results_{now}.pkl'

if  'google.colab' not in sys.modules: # Don't use system arguments run in␣
 ↪Colab
    speed_test(sys.argv[1:])
elif __name__ == "__main__" : # but define them manually
    speed_test(["--out_bucket", BUCKET, "--out_file", FILENAME])
```

```
Tensorflow version 2.12.0
Tensorflow version 2.12.0
['--out_bucket', 'gs://my-project-220003166-storage', '--out_file',
'task_2b_results_230501-1728.pkl']
Saving task_2b_results_230501-1728.pkl to gs://my-project-220003166-storage
gstuil returned: 0
b'Copying file://task_2b_results_230501-1728.pkl [Content-
Type=application/octet-stream]…\n/ [0 files][    0.0 B/  701.0 B]
\r-\r- [0 files][  701.0 B/  701.0 B]
\r- [1 files][  701.0 B/  701.0 B]
\r\\\r\nOperation completed over 1 objects/701.0 B.
\n'
```

[ ]:
```
%%writefile spark_job.py

# vi) write the results to a pickle file in your bucket (1%)
#vii) Write your code it into a file using the cell magic %%writefile spark_job.
 ↪py (1%)

### CODING TASK ###


# import required libraries for clouad
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row
from pyspark.sql import SparkSession
import os, sys, math
import numpy as np
import time
import datetime
import string
import random
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
import argparse



#parameters
PROJECT = 'my-project-220003166'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/'
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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']
nb_images = len(tf.io.gfile.glob(GCS_PATTERN))


def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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
```

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


def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # 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_deterministic = False
    # 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
```

```python
def load_dataset_images():
  dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN)
  dsetDecoded = dsetFiles.map(decode_jpeg_and_label)
  dsetResized = dsetDecoded.map(resize_and_crop_image)
  return  dsetResized

# Adapted function for tf record files
def time_configs_TFRecord(parameters_rdd):

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

    filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
    dataset = load_dataset(filenames)
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
 ↪reading_speed, throughput])

    return measure


def time_configs_images(parameters_rdd):

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

    dataset = load_dataset_images()
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
```

```
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
↪reading_speed, throughput])

    return measure


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=subprocess.
↪PIPE)
    print("gstuil returned: " + str(proc.returncode))
    print(str(proc.stderr))

  # vi) write the results to a pickle file in your bucket (2%)
def speed_test(argv):
  # Parse the provided arguments
  print(argv)
  parser = argparse.ArgumentParser() # get a parser object
  parser.add_argument('--out_bucket', metavar='out_bucket', required=True,
                      help='The bucket URL for the result.') # add a required␣
↪argument
  parser.add_argument('--out_file', metavar='out_file', required=True,
                      help='The filename for the result.') # add a required␣
↪argument
  args = parser.parse_args(argv) # read the value

  # 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%)


  #batch_sizes = [2,4] # toy parameters for testing
```

```python
#batch_numbers = [3,6]
#repetitions = [1]

batch_sizes = [8,16,32,64,128,256] # real parameters used in final script and
↪retrieve the results
batch_numbers = [10,20,30,40]
repetitions = [1, 2, 3]


params = []
for batch_size in batch_sizes:
    for batch_number in batch_numbers:
        for repetition in repetitions:
            params.append([batch_size, batch_number, repetition])

column_names = ["batch_sizes", "batch_numbers", "repetitions",
↪"dataset_size", "reading_speed", "throughput"]



# Creating a Spark context
sc = pyspark.SparkContext.getOrCreate()
# Creating an RDD for parameter combinations
partition_num = len(params)
param_combinations_rdd = sc.parallelize(params,partition_num)
# Creating a Spark session for converting results into dataframes
ss_TF_params_time = SparkSession(sc)
# Applying each parameter combination to a time measurement function to
↪determine reading speed and throughput in images per second
TF_params_time = param_combinations_rdd.flatMap(time_configs_TFRecord)
### TASK 2c ###
#TF_params_time.cache()
# Creating dataframes
df_TF_params_time = TF_params_time.toDF(column_names)


# Creating a Spark context
sc = pyspark.SparkContext.getOrCreate()
# Creating an RDD for parameter combinations
partition_num = len(params)
param_combinations_rdd = sc.parallelize(params,partition_num)
# Creating a Spark session for converting results into dataframes
ss_img_files = SparkSession(sc)
# Applying each parameter combination to a time measurement function to
↪determine reading speed and throughput in images per second
images_params_time = param_combinations_rdd.flatMap(time_configs_images)
### TASK 2c ###
```

```
#images_params_time.cache()
# Creating dataframes
df_images_params_time = images_params_time.toDF(column_names)


# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
TF_params_time_array = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                                  ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
images_params_time_array = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                                  ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iv) create an RDD with all results for each parameter as␣
↪(parameter_value,images_per_second) and collect the result for each␣
↪parameter (1%)

# RDD for batch sizes and images_per_second (TFrecords files)
TFrecord_batchSizes_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
TFrecord_batchSizes_throughput = TFrecord_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (TFrecords files)
TFrecord_batchNumbers_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
TFrecord_batchNumbers_throughput = TFrecord_batchNumbers_throughput_rdd.
↪collect()

# RDD for repetitions and images_per_second (TFrecords files)
TFrecord_repetitions_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
TFrecord_repetitions_throughput = TFrecord_repetitions_throughput_rdd.
↪collect()

# RDD for dataset size and images_per_second (TFrecords files)
TFrecord_datasetSize_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
TFrecord_datasetSize_throughput = TFrecord_datasetSize_throughput_rdd.
↪collect()
```

```
# RDD for batch sizes and images_per_second (Images)
images_batchSizes_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
images_batchSizes_throughput = images_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (Images)
images_batchNumbers_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
images_batchNumbers_throughput = images_batchNumbers_throughput_rdd.collect()

# RDD for repetitions and images_per_second (Images)
images_repetitions_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
images_repetitions_throughput = images_repetitions_throughput_rdd.collect()

# RDD for dataset size and images_per_second (Images)
images_datasetSize_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
images_datasetSize_throughput = images_datasetSize_throughput_rdd.collect()

␣
↪##########################################################################################

# 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%)

# RDD for batch size and average images_per_second (TFrecord files)
TFrecord_batchSizes_avg_throughput_rdd = TFrecord_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_batchSizes_avg_throughput = TFrecord_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (TFrecord files)
TFrecord_batchNumbers_avg_throughput_rdd =␣
↪TFrecord_batchNumbers_throughput_rdd.mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
```

```
TFrecord_batchNumbers_avg_throughput =␣
↪TFrecord_batchNumbers_avg_throughput_rdd.collect()


# RDD for repetitions and average images_per_second (TFrecord files)
TFrecord_repetitions_avg_throughput_rdd = TFrecord_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_repetitions_avg_throughput = TFrecord_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (TFrecord files)
TFrecord_datasetSize_avg_throughput_rdd = TFrecord_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_datasetSize_avg_throughput = TFrecord_datasetSize_avg_throughput_rdd.
↪collect()


# RDD for batch size and average images_per_second (images)
images_batchSizes_avg_throughput_rdd = images_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
images_batchSizes_avg_throughput = images_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (images)
images_batchNumbers_avg_throughput_rdd = images_batchNumbers_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                     .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                     .mapValues(lambda z: z[0]/
↪z[1])
images_batchNumbers_avg_throughput = images_batchNumbers_avg_throughput_rdd.
↪collect()
```

```
# RDD for repetitions and average images_per_second (images)
images_repetitions_avg_throughput_rdd = images_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
images_repetitions_avg_throughput = images_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (images)
images_datasetSize_avg_throughput_rdd = images_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
images_datasetSize_avg_throughput = images_datasetSize_avg_throughput_rdd.
↪collect()

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

save_object = (TFrecord_batchSizes_throughput,
TFrecord_batchNumbers_throughput,
TFrecord_repetitions_throughput,
TFrecord_datasetSize_throughput,
images_batchSizes_throughput,
images_batchNumbers_throughput,
images_repetitions_throughput,
images_datasetSize_throughput,
TFrecord_batchSizes_avg_throughput,
TFrecord_batchNumbers_avg_throughput,
TFrecord_repetitions_avg_throughput,
TFrecord_datasetSize_avg_throughput,
images_batchSizes_avg_throughput,
images_batchNumbers_avg_throughput,
images_repetitions_avg_throughput,
images_datasetSize_avg_throughput,)


# save tuple of all parameter results
save(save_object, args.out_bucket, args.out_file)
```

```
# Create a filename with the current date and time
now = datetime.datetime.now().strftime("%y%m%d-%H%M")
FILENAME = f'task_2b_results_{now}.pkl'

if  'google.colab' not in sys.modules: # Don't use system arguments run in␣
 ↪Colab
    speed_test(sys.argv[1:])
elif __name__ == "__main__" : # but define them manually
    speed_test(["--out_bucket", BUCKET, "--out_file", FILENAME])
```

```
Writing spark_job.py
```

[ ]:

```
```

5.2.2 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.

[ ]:

```
### CODING TASK

%run ./spark_job.py
```

```
Tensorflow version 2.12.0
['--out_bucket', 'gs://my-project-220003166-storage', '--out_file',
'task_2b_results_230427-1600.pkl']
Saving task_2b_results_230427-1600.pkl to gs://my-project-220003166-storage
gstuil returned: 0
b'Copying file://task_2b_results_230427-1600.pkl [Content-
Type=application/octet-stream]…\n/ [0 files][    0.0 B/  701.0 B]
\r/ [1 files][  701.0 B/  701.0 B]
\r\nOperation completed over 1 objects/701.0 B.
\n'
```

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 a little. These images will be useful in the next task. Again, don't use the SCREENSHOT function that Google provides, but just take a picture of the graphs you see for the VMs.

```
[ ]: ### CODING TASK

#4 machines with double the resources each (2 vCPUs, memory, disk), (one␣
↪master, three workers)
CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-central1'

!gcloud dataproc clusters create $CLUSTER --region $REGION \
  --bucket $PROJECT-storage \
  --region $REGION \
  --num-workers 3 \
  --worker-machine-type n1-standard-2 \
  --worker-boot-disk-size 666 \
  --worker-boot-disk-type pd-standard \
  --master-machine-type n1-standard-2 \
  --master-boot-disk-size 100 \
  --master-boot-disk-type pd-ssd \
  --image-version 1.5-ubuntu18 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/
↪python/pip-install.sh \
  --metadata 'PIP_PACKAGES=tensorflow' \
  --max-idle 3600s
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/22b3075a-c06a-3ac7-8392-a974837dbf1f].

WARNING: Don't create production clusters that reference
initialization actions located in the gs://goog-dataproc-initialization-actions-
REGION public buckets. These scripts are provided as reference implementations,
and they are synchronized with ongoing GitHub repository changes-a new version
of a initialization action in public buckets may break your cluster creation.
Instead, copy the following initialization actions from public buckets into your
bucket : gs://goog-dataproc-initialization-actions-us-central1/python/pip-
install.sh
WARNING: For PD-Standard without local SSDs, we strongly recommend
provisioning 1TB or larger to ensure consistently high I/O performance. See
https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the staging_bucket
'projects/_/buckets/my-project-220003166-storage'. This usually happens when a
custom resource (ex: custom staging bucket) or a user-managed VM Service account
has been provided and the default/user-managed service account hasn't been
granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
```

```
WARNING: Permissions are missing for the default service account
'812583826827-compute@developer.gserviceaccount.com', missing permissions:
[storage.objects.get, storage.objects.update] on the temp_bucket
'projects/_/buckets/dataproc-temp-us-central1-812583826827-nqseqgr3'. This
usually happens when a custom resource (ex: custom staging bucket) or a user-
managed VM Service account has been provided and the default/user-managed
service account hasn't been granted enough permissions on the resource. See
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-
accounts#VM_service_account.
Created [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster]
Cluster placed in zone [us-central1-f].
```

[ ]:
```
# submit the  job

# Create a filename with the current date and time
now = datetime.datetime.now().strftime("%y%m%d-%H%M")
FILENAME = f'task_2b_results_{now}.pkl'

#FILENAME = 'task_2b_results.pkl'
PROJECT = 'my-project-220003166'
BUCKET = 'gs://{}-storage'.format(PROJECT)
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
    ./spark_job.py \
    -- --out_bucket $BUCKET --out_file $FILENAME
```

```
Job [22b8f458f88d47eab3e5c45f9583b3b1] submitted.
Waiting for job output…
2023-04-29 11:02:37.917502: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-04-29 11:02:38.097125: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot
open shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-29 11:02:38.097170: I
tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
2023-04-29 11:02:39.016291: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot
open shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-29 11:02:39.016465: W
```

```
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer_plugin.so.7'; dlerror:
libnvinfer_plugin.so.7: cannot open shared object file: No such file or
directory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2023-04-29 11:02:39.016492: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: 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.
Tensorflow version 2.11.0
['--out_bucket', 'gs://my-project-220003166-storage', '--out_file',
'task_2b_results_230429-1102.pkl']
23/04/29 11:02:42 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/29 11:02:42 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/29 11:02:42 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/29 11:02:43 INFO org.spark_project.jetty.util.log: Logging initialized
@8380ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/29 11:02:43 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_362-b09
23/04/29 11:02:43 INFO org.spark_project.jetty.server.Server: Started @8562ms
23/04/29 11:02:43 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@59e779be{HTTP/1.1, (http/1.1)}{0.0.0.0:46705}
23/04/29 11:02:45 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.15.219:8032
23/04/29 11:02:45 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.15.219:10200
23/04/29 11:02:45 INFO org.apache.hadoop.conf.Configuration: resource-types.xml
not found
23/04/29 11:02:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Unable to find 'resource-types.xml'.
23/04/29 11:02:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
23/04/29 11:02:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = vcores, units = , type = COUNTABLE
23/04/29 11:02:49 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682766019186_0001
Saving task_2b_results_230429-1102.pkl to gs://my-project-220003166-storage
gstuil returned: 0
b'Copying file://task_2b_results_230429-1102.pkl [Content-
Type=application/octet-stream]…\n/ [0 files][    0.0 B/  9.6 KiB]
\r/ [1 files][  9.6 KiB/  9.6 KiB]
\r\nOperation completed over 1 objects/9.6 KiB.
\n'
23/04/29 13:21:24 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@59e779be{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [22b8f458f88d47eab3e5c45f9583b3b1] finished successfully.
done: true
```

```
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/28f955fd-e5b1-49ca-9885-
1e784a2f50bf/jobs/22b8f458f88d47eab3e5c45f9583b3b1/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/28f955fd-e5b1-49ca-9885-
1e784a2f50bf/jobs/22b8f458f88d47eab3e5c45f9583b3b1/driveroutput
jobUuid: 74ae0ad8-5af1-352b-901c-8e43a5841138
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: 28f955fd-e5b1-49ca-9885-1e784a2f50bf
pysparkJob:
  args:
  - --out_bucket
  - gs://my-project-220003166-storage
  - --out_file
  - task_2b_results_230429-1102.pkl
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/28f955fd-e5b1-49ca-9885-
1e784a2f50bf/jobs/22b8f458f88d47eab3e5c45f9583b3b1/staging/spark_job.py
reference:
  jobId: 22b8f458f88d47eab3e5c45f9583b3b1
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-29T13:21:26.054723Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-29T11:02:32.436242Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-29T11:02:32.464631Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-29T11:02:32.823616Z'
yarnApplications:
- name: spark_job.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682766019186_0001/
```


[ ]:
```
!gsutil cp $BUCKET/$FILENAME .
with open(FILENAME,mode='rb') as f:
    results_2b = pickle.load(f)
```

```
Copying gs://my-project-220003166-storage/task_2b_results_230429-1102.pkl…
/ [1 files][  9.6 KiB/  9.6 KiB]
Operation completed over 1 objects/9.6 KiB.
```

[ ]:
```
#delete the cluster
!gcloud dataproc clusters delete $CLUSTER --region=us-central1 -q
```

```
Waiting on operation [projects/my-project-220003166/regions/us-
central1/operations/9f1936d8-f110-307f-89f7-39df145493a7].
Deleted [https://dataproc.googleapis.com/v1/projects/my-
project-220003166/regions/us-central1/clusters/my-project-220003166-cluster].
```

5.2.3 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**

[ ]:
```
%%writefile spark_job_2c.py

### CODING TASK
# import required libraries
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row
from pyspark.sql import SparkSession
import os, sys, math
import numpy as np
import time
import datetime
import string
import random
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
import argparse



#parameters
PROJECT = 'my-project-220003166'
BUCKET = 'gs://{}-storage'.format(PROJECT)
GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/'
```

```
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
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']
nb_images = len(tf.io.gfile.glob(GCS_PATTERN))


def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    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 image
    # 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]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) //␣
↪2, tw, th)
    return image, label


def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string =␣
↪bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means␣
↪scalar
    }
    # 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_deterministic =␣
 ↪False
    # 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

def load_dataset_images():
  dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN)
  dsetDecoded = dsetFiles.map(decode_jpeg_and_label)
  dsetResized = dsetDecoded.map(resize_and_crop_image)
  return  dsetResized

# Adapted function for tf record files
def time_configs_TFRecord(parameters_rdd):

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

    filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
    dataset = load_dataset(filenames)
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
```

```
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
 ↪reading_speed, throughput])

    return measure


def time_configs_images(parameters_rdd):

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

    dataset = load_dataset_images()
    measure = []

    with open("/dev/null", mode='w') as null_file:
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_num )
        for rep in range(repetitions):
            s_time = time.time()
            for image, label in timing_set:
                print("Image batch shape {}, {})".format(image.numpy().shape,
                    [str(lbl) for lbl in label.numpy()]), null_file)
            e_time = time.time()
            reading_speed = e_time - s_time
            throughput = float((batch_size * batch_num ) / (e_time - s_time))
            datasetsize = batch_size * batch_num
            measure.append([batch_size, batch_num, repetitions, datasetsize,␣
 ↪reading_speed, throughput])

    return measure


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=subprocess.
 ↪PIPE)
    print("gstuil returned: " + str(proc.returncode))
    print(str(proc.stderr))

def speed_test(argv):
  # Parse the provided arguments
  print(argv)
  parser = argparse.ArgumentParser() # get a parser object
```

```python
parser.add_argument('--out_bucket', metavar='out_bucket', required=True,
                    help='The bucket URL for the result.') # add a required␣
↪argument
parser.add_argument('--out_file', metavar='out_file', required=True,
                    help='The filename for the result.') # add a required␣
↪argument
args = parser.parse_args(argv) # read the value

# 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%)

batch_sizes = [8,16,32,64,128,256]
batch_numbers = [10,20,30,40]
repetitions = [1, 2, 3]

#batch_sizes = [4,8,16,32]
#batch_numbers = [10,20,30]
#repetitions = [1, 2, 3]

params = []
for batch_size in batch_sizes:
    for batch_number in batch_numbers:
        for repetition in repetitions:
            params.append([batch_size, batch_number, repetition])

column_names = ["batch_sizes", "batch_numbers", "repetitions",␣
↪"dataset_size", "reading_speed", "throughput"]



# Creating a Spark context
sc = pyspark.SparkContext.getOrCreate()
# Creating an RDD for parameter combinations
partition_num = len(params)
param_combinations_rdd = sc.parallelize(params,partition_num)
# Creating a Spark session for converting results into dataframes
ss_TF_params_time = SparkSession(sc)
# Applying each parameter combination to a time measurement function to␣
↪determine reading speed and throughput in images per second
TF_params_time = param_combinations_rdd.flatMap(time_configs_TFRecord)
### TASK 2c ###
TF_params_time.cache()
# Creating dataframes
df_TF_params_time = TF_params_time.toDF(column_names)
```

```
# Creating a Spark context
sc = pyspark.SparkContext.getOrCreate()
# Creating an RDD for parameter combinations
partition_num = len(params)
param_combinations_rdd = sc.parallelize(params,partition_num)
# Creating a Spark session for converting results into dataframes
ss_img_files = SparkSession(sc)
# Applying each parameter combination to a time measurement function to␣
↪determine reading speed and throughput in images per second
images_params_time = param_combinations_rdd.flatMap(time_configs_images)
### TASK 2c ###
images_params_time.cache()
# Creating dataframes
df_images_params_time = images_params_time.toDF(column_names)


# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
TF_params_time_array = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                        ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iii) transform the resulting RDD to the structure ( parameter_combination,␣
↪images_per_second ) and save these values in an array (2%)
images_params_time_array = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']),(x['batch_numbers']),(x['repetitions']),
                                                        ␣
↪(x['dataset_size']),(x['reading_speed']), (x['throughput']))).collect()

# iv) create an RDD with all results for each parameter as␣
↪(parameter_value,images_per_second) and collect the result for each␣
↪parameter (1%)

# RDD for batch sizes and images_per_second (TFrecords files)
TFrecord_batchSizes_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
TFrecord_batchSizes_throughput = TFrecord_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (TFrecords files)
TFrecord_batchNumbers_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
TFrecord_batchNumbers_throughput = TFrecord_batchNumbers_throughput_rdd.
↪collect()
```

```
# RDD for repetitions and images_per_second (TFrecords files)
TFrecord_repetitions_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
TFrecord_repetitions_throughput = TFrecord_repetitions_throughput_rdd.
↪collect()

# RDD for dataset size and images_per_second (TFrecords files)
TFrecord_datasetSize_throughput_rdd = df_TF_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
TFrecord_datasetSize_throughput = TFrecord_datasetSize_throughput_rdd.
↪collect()

# RDD for batch sizes and images_per_second (Images)
images_batchSizes_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_sizes']), (x['throughput'])))
images_batchSizes_throughput = images_batchSizes_throughput_rdd.collect()

# RDD for batch numbers and images_per_second (Images)
images_batchNumbers_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['batch_numbers']), (x['throughput'])))
images_batchNumbers_throughput = images_batchNumbers_throughput_rdd.collect()

# RDD for repetitions and images_per_second (Images)
images_repetitions_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['repetitions']), (x['throughput'])))
images_repetitions_throughput = images_repetitions_throughput_rdd.collect()

# RDD for dataset size and images_per_second (Images)
images_datasetSize_throughput_rdd = df_images_params_time.rdd.map(lambda x:␣
↪((x['dataset_size']), (x['throughput'])))
images_datasetSize_throughput = images_datasetSize_throughput_rdd.collect()

␣
↪##################################################################################

# 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%)

# RDD for batch size and average images_per_second (TFrecord files)
TFrecord_batchSizes_avg_throughput_rdd = TFrecord_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
```

```
                                                    .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_batchSizes_avg_throughput = TFrecord_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (TFrecord files)
TFrecord_batchNumbers_avg_throughput_rdd =␣
↪TFrecord_batchNumbers_throughput_rdd.mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_batchNumbers_avg_throughput =␣
↪TFrecord_batchNumbers_avg_throughput_rdd.collect()


# RDD for repetitions and average images_per_second (TFrecord files)
TFrecord_repetitions_avg_throughput_rdd = TFrecord_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_repetitions_avg_throughput = TFrecord_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (TFrecord files)
TFrecord_datasetSize_avg_throughput_rdd = TFrecord_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
TFrecord_datasetSize_avg_throughput = TFrecord_datasetSize_avg_throughput_rdd.
↪collect()


# RDD for batch size and average images_per_second (images)
images_batchSizes_avg_throughput_rdd = images_batchSizes_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
```

```
images_batchSizes_avg_throughput = images_batchSizes_avg_throughput_rdd.
↪collect()


# RDD for batch numbers and average images_per_second (images)
images_batchNumbers_avg_throughput_rdd = images_batchNumbers_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
images_batchNumbers_avg_throughput = images_batchNumbers_avg_throughput_rdd.
↪collect()


# RDD for repetitions and average images_per_second (images)
images_repetitions_avg_throughput_rdd = images_repetitions_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
images_repetitions_avg_throughput = images_repetitions_avg_throughput_rdd.
↪collect()


# RDD for data size and average images_per_second (images)
images_datasetSize_avg_throughput_rdd = images_datasetSize_throughput_rdd.
↪mapValues(lambda z: (z, 1)) \
                                                    .reduceByKey(lambda x,y:␣
↪(x[0]+y[0], x[1]+y[1])) \
                                                    .mapValues(lambda z: z[0]/
↪z[1])
images_datasetSize_avg_throughput = images_datasetSize_avg_throughput_rdd.
↪collect()


save_object = (TFrecord_batchSizes_throughput,
TFrecord_batchNumbers_throughput,
TFrecord_repetitions_throughput,
TFrecord_datasetSize_throughput,
images_batchSizes_throughput,
images_batchNumbers_throughput,
images_repetitions_throughput,
images_datasetSize_throughput,
TFrecord_batchSizes_avg_throughput,
```

```
  TFrecord_batchNumbers_avg_throughput,
  TFrecord_repetitions_avg_throughput,
  TFrecord_datasetSize_avg_throughput,
  images_batchSizes_avg_throughput,
  images_batchNumbers_avg_throughput,
  images_repetitions_avg_throughput,
  images_datasetSize_avg_throughput,)


  # save tuple of all parameter results
  save(save_object, args.out_bucket, args.out_file)



if  'google.colab' not in sys.modules: # Don't use system arguments run in␣
 ↪Colab
    speed_test(sys.argv[1:])
elif __name__ == "__main__" : # but define them manually
    speed_test(["--out_bucket", BUCKET, "--out_file", FILENAME])
```

```
Overwriting spark_job_2c.py
```

[ ]:
```
# submit the  job
# Create a filename with the current date and time
now = datetime.datetime.now().strftime("%y%m%d-%H%M")
FILENAME = f'task_2c_results_{now}.pkl'

PROJECT = 'my-project-220003166'
BUCKET = 'gs://{}-storage'.format(PROJECT)
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION \
    ./spark_job_2c.py \
    -- --out_bucket $BUCKET --out_file $FILENAME
```

```
Job [8e4128a3a4fc409ba352f393d1dfbd0e] submitted.
Waiting for job output…
2023-04-30 15:52:49.448723: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-04-30 15:52:49.616640: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot
open shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-30 15:52:49.616687: I
```

```
tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
2023-04-30 15:52:50.562176: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot
open shared object file: No such file or directory; LD_LIBRARY_PATH:
:/usr/lib/hadoop/lib/native
2023-04-30 15:52:50.562345: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer_plugin.so.7'; dlerror:
libnvinfer_plugin.so.7: cannot open shared object file: No such file or
directory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2023-04-30 15:52:50.562369: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: 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.
Tensorflow version 2.11.0
['--out_bucket', 'gs://my-project-220003166-storage', '--out_file',
'task_2c_results_230430-1552.pkl']
23/04/30 15:52:52 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/30 15:52:53 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/30 15:52:53 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
23/04/30 15:52:53 INFO org.spark_project.jetty.util.log: Logging initialized
@6780ms to org.spark_project.jetty.util.log.Slf4jLog
23/04/30 15:52:53 INFO org.spark_project.jetty.server.Server:
jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_362-b09
23/04/30 15:52:53 INFO org.spark_project.jetty.server.Server: Started @6949ms
23/04/30 15:52:53 INFO org.spark_project.jetty.server.AbstractConnector: Started
ServerConnector@25771daa{HTTP/1.1, (http/1.1)}{0.0.0.0:36545}
23/04/30 15:52:55 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to
ResourceManager at my-project-220003166-cluster-m/10.128.0.7:8032
23/04/30 15:52:55 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to
Application History server at my-project-220003166-cluster-m/10.128.0.7:10200
23/04/30 15:52:55 INFO org.apache.hadoop.conf.Configuration: resource-types.xml
not found
23/04/30 15:52:55 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Unable to find 'resource-types.xml'.
23/04/30 15:52:55 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
23/04/30 15:52:55 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils:
Adding resource type - name = vcores, units = , type = COUNTABLE
23/04/30 15:52:58 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl:
Submitted application application_1682864881095_0003
Saving task_2c_results_230430-1552.pkl to gs://my-project-220003166-storage
gstuil returned: 0
b'Copying file://task_2c_results_230430-1552.pkl [Content-
```

```
Type=application/octet-stream]…\n/ [0 files][    0.0 B/ 19.7 KiB]
\r/ [1 files][ 19.7 KiB/ 19.7 KiB]
\r\nOperation completed over 1 objects/19.7 KiB.
\n'
23/04/30 16:59:28 INFO org.spark_project.jetty.server.AbstractConnector: Stopped
Spark@25771daa{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [8e4128a3a4fc409ba352f393d1dfbd0e] finished successfully.
done: true
driverControlFilesUri: gs://my-project-220003166-storage/google-cloud-dataproc-m
etainfo/2c072c5b-9cef-4b36-95b8-
2d8fbf8379b5/jobs/8e4128a3a4fc409ba352f393d1dfbd0e/
driverOutputResourceUri: gs://my-project-220003166-storage/google-cloud-
dataproc-metainfo/2c072c5b-9cef-4b36-95b8-
2d8fbf8379b5/jobs/8e4128a3a4fc409ba352f393d1dfbd0e/driveroutput
jobUuid: dffaf97a-27ed-3ca5-bf88-838cccefbdcc
placement:
  clusterName: my-project-220003166-cluster
  clusterUuid: 2c072c5b-9cef-4b36-95b8-2d8fbf8379b5
pysparkJob:
  args:
  - --out_bucket
  - gs://my-project-220003166-storage
  - --out_file
  - task_2c_results_230430-1552.pkl
  mainPythonFileUri: gs://my-project-220003166-storage/google-cloud-dataproc-met
ainfo/2c072c5b-9cef-4b36-95b8-
2d8fbf8379b5/jobs/8e4128a3a4fc409ba352f393d1dfbd0e/staging/spark_job_2c.py
reference:
  jobId: 8e4128a3a4fc409ba352f393d1dfbd0e
  projectId: my-project-220003166
status:
  state: DONE
  stateStartTime: '2023-04-30T16:59:33.055776Z'
statusHistory:
- state: PENDING
  stateStartTime: '2023-04-30T15:52:45.714157Z'
- state: SETUP_DONE
  stateStartTime: '2023-04-30T15:52:45.742008Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2023-04-30T15:52:45.969156Z'
yarnApplications:
- name: spark_job_2c.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://my-
project-220003166-cluster-m:8088/proxy/application_1682864881095_0003/
```

```
[ ]: #FILENAME = 'task_2c_results_230429-1403.pkl'
     #FILENAME = 'task_2c_results_230430-1437.pkl'
     #FILENAME = 'task_2c_results_230430-1552.pkl'
     !gsutil cp $BUCKET/$FILENAME .
     with open(FILENAME,mode='rb') as f:
         results_2c = pickle.load(f)
```

```
Copying gs://my-project-220003166-storage/task_2c_results_230430-1552.pkl…
/ [1 files][ 19.7 KiB/ 19.7 KiB]
Operation completed over 1 objects/19.7 KiB.
```

```
[ ]: !gcloud dataproc clusters delete $CLUSTER --region=us-central1 -q
```

5.2.4 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 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 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**.

```
[ ]: ###Tables###
     import pandas as pd
     TFrecord_batchSizes_avg_throughput_df = pd.
      ↪DataFrame(results_2c[8],columns=['batchSizes','avg_throughput'])
     images_batchSizes_avg_throughput_df = pd.
      ↪DataFrame(results_2c[12],columns=['batchSizes','avg_throughput'])
```

```
batchSizes_avg_throughput_bothdatasets_df = pd.
 ↪merge(TFrecord_batchSizes_avg_throughput_df,␣
 ↪images_batchSizes_avg_throughput_df, on='batchSizes' ,␣
 ↪suffixes=('_Tfrecords','_images'))
print(batchSizes_avg_throughput_bothdatasets_df)
print('=================================================================================


TFrecord_batchNumbers_avg_throughput_df = pd.
 ↪DataFrame(results_2c[9],columns=['batchNumbers','avg_throughput'])
images_batchNumbers_avg_throughput_df = pd.
 ↪DataFrame(results_2c[13],columns=['batchNumbers','avg_throughput'])
batchNumbers_avg_throughput_bothdatasets_df = pd.
 ↪merge(TFrecord_batchNumbers_avg_throughput_df,␣
 ↪images_batchNumbers_avg_throughput_df, on='batchNumbers' ,␣
 ↪suffixes=('_Tfrecords','_images'))
print(batchNumbers_avg_throughput_bothdatasets_df)
print('=================================================================================

TFrecord_repetitions_avg_throughput_df = pd.
 ↪DataFrame(results_2c[10],columns=['repetitions','avg_throughput'])
images_repetitions_avg_throughput_df = pd.
 ↪DataFrame(results_2c[14],columns=['repetitions','avg_throughput'])
repetitions_avg_throughput_bothdatasets_df = pd.
 ↪merge(TFrecord_repetitions_avg_throughput_df,␣
 ↪images_repetitions_avg_throughput_df, on='repetitions' ,␣
 ↪suffixes=('_Tfrecords','_images'))
print(repetitions_avg_throughput_bothdatasets_df)
print('=================================================================================


TFrecord_datasetSize_avg_throughput_df = pd.
 ↪DataFrame(results_2c[11],columns=['datasetSize','avg_throughput'])
images_datasetSize_avg_throughput_df = pd.
 ↪DataFrame(results_2c[15],columns=['datasetSize','avg_throughput'])
datasetSize_avg_throughput_bothdatasets_df = pd.
 ↪merge(TFrecord_datasetSize_avg_throughput_df,␣
 ↪images_datasetSize_avg_throughput_df, on='datasetSize' ,␣
 ↪suffixes=('_Tfrecords','_images'))
print(datasetSize_avg_throughput_bothdatasets_df)
print('=================================================================================
```

```
   batchSizes  avg_throughput_Tfrecords  avg_throughput_images
0           8                510.058045              12.864221
1          16                562.774530              13.443209
2          32                636.170541              13.467651
3         256               1248.954330              26.616013
```

```
4         128               735.153808               14.617623
5          64               611.730077               13.709902
================================================================================
==================================================
   batchNumbers  avg_throughput_Tfrecords  avg_throughput_images
0            10                551.726569              13.768251
1            20                613.878616              14.224091
2            30                812.371808              18.129316
3            40                891.917228              17.024089
================================================================================
==================================================
   repetitions  avg_throughput_Tfrecords  avg_throughput_images
0            1                739.490162              14.516821
1            2                695.785998              14.754438
2            3                724.593058              16.897641
================================================================================
==================================================
    datasetSize  avg_throughput_Tfrecords  avg_throughput_images
0          5120                850.598178              15.337438
1            80                369.789021              11.775537
2           160                435.829375              13.526268
3         10240               1741.564284              42.419460
4          3840                814.809392              15.236694
5           960                590.652939              15.039817
6           240                540.754950              15.086032
7           320                639.178633              12.451534
8          2560                655.279640              13.204747
9           480                538.580782              15.159796
10         1920                714.787893              14.837961
11         7680               1674.644891              33.415595
12         1280                681.876653              13.216743
13          640                577.307096              13.852368
================================================================================
==================================================
```

[ ]:
```python
### CODING TASK ###

import seaborn as sns
import pandas as pd
import scipy as sp

def regression_analysis(listOfTuples1, listOfTuples2, title, X_axis_label):
    df1 = pd.DataFrame(listOfTuples1, columns=['x1', 'y1'])
    slope, intercept, r_value, p_value, std_err = sp.stats.
 ↪linregress(df1['x1'], df1['y1'])
    print(f'Slope: {slope}')
    print(f'Intercept: {intercept}')
```

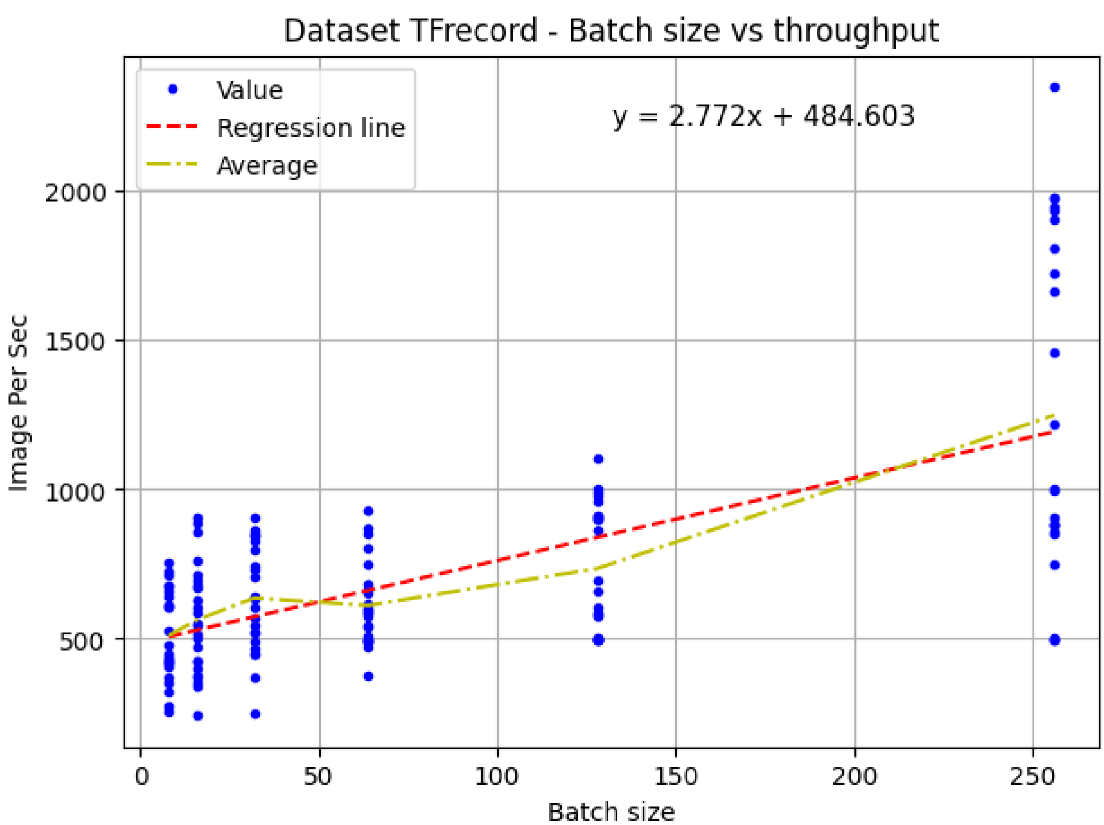
```
    print(f'r-value: {r_value}')
    print(f'p-value: {p_value}')
    print(f'std-err: {std_err}')
    regLine = intercept + slope * df1['x1']
    listOfTuples2_sorted = sorted(listOfTuples2, key=lambda x: x[0])
    df2 = pd.DataFrame(listOfTuples2_sorted, columns=['x2', 'y2'])
    plt.figure(figsize=(7, 5))
    plt.plot(df1['x1'], df1['y1'], 'b.')
    plt.plot(df1['x1'], regLine, 'r--')
    plt.plot(df2['x2'], df2['y2'], 'y-.')
    plt.grid(True)
    plt.title(title)
    plt.xlabel(X_axis_label)
    plt.ylabel('Image Per Sec')
    equation = 'y = {:.3f}x + {:.3f}'.format(slope, intercept)
    plt.annotate(equation, xy=(0.5, 0.9), xycoords='axes fraction', fontsize=11)
    plt.legend(['Value','Regression line', 'Average'], loc='best')
    plt.show()
   ␣
  ↪print('-----------------------------------------------------------------------------------
   ␣
  ↪print('-----------------------------------------------------------------------------------


# TFrecords
# Batch size vs throughput
regression_analysis(results_2c[0], results_2c[8], 'Dataset TFrecord - Batch␣
 ↪size vs throughput','Batch size')
# Batch numbers vs throughput
regression_analysis(results_2c[1], results_2c[9], 'Dataset TFrecord - Batch␣
 ↪numbers vs throughput','Batch number')
# Repititions vs throughput
regression_analysis(results_2c[2], results_2c[10], 'Dataset TFrecord -␣
 ↪Repitition vs throughput','Repititions')
# Datasize vs throughput
regression_analysis(results_2c[3], results_2c[11], 'Dataset TFrecord - Dataset␣
 ↪size vs throughput','Dataset size')


# images
# Batch size vs throughput
regression_analysis(results_2c[4], results_2c[12], 'Dataset images - Batch size␣
 ↪vs throughput','Batch size')
# Batch numbers vs throughput
regression_analysis(results_2c[5], results_2c[13], 'Dataset images - Batch␣
 ↪numbers vs throughput','Batch number')
```

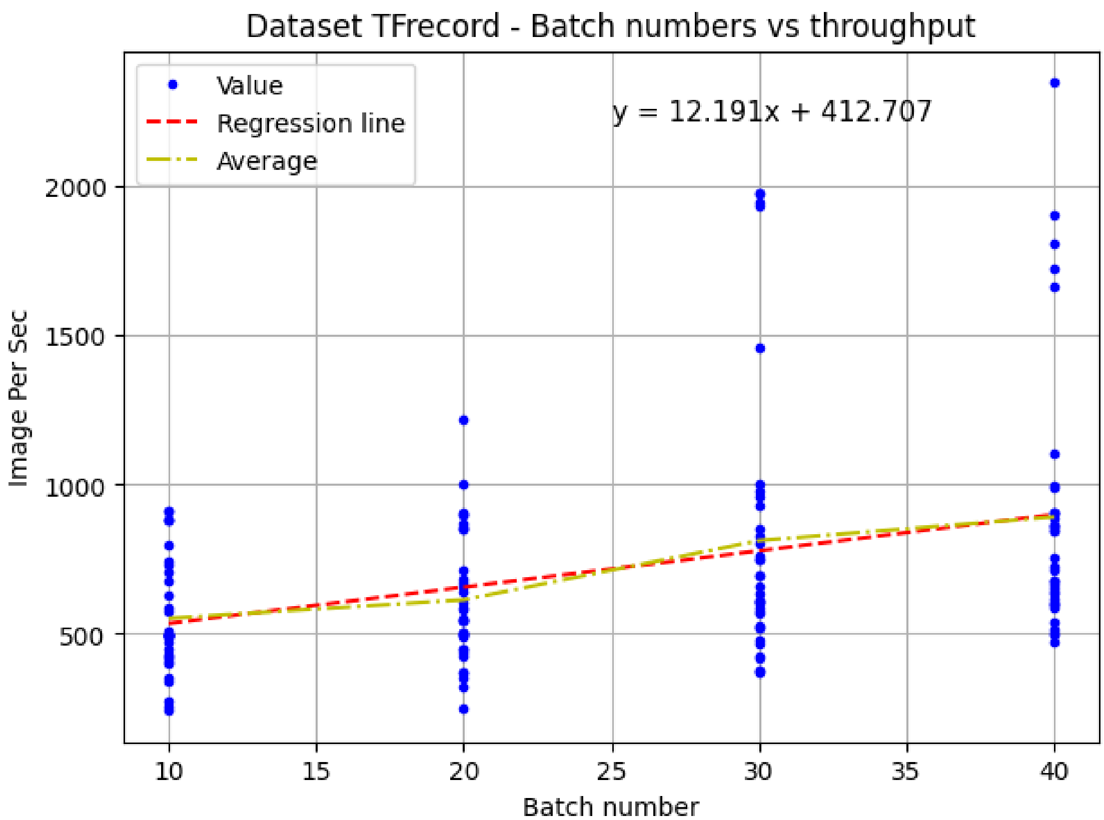
```
# Repititions vs throughput
regression_analysis(results_2c[6], results_2c[14], 'Dataset images - Repitition␣
↪vs throughput','Repititions')
# Datasize vs throughput
regression_analysis(results_2c[7], results_2c[15], 'Dataset images - Dataset␣
↪size vs throughput','Dataset size')
```

```
Slope: 2.7722628647284746
Intercept: 484.6034745483995
r-value: 0.6410267662429828
p-value: 4.972234511370804e-18
std-err: 0.27854985541694627
```

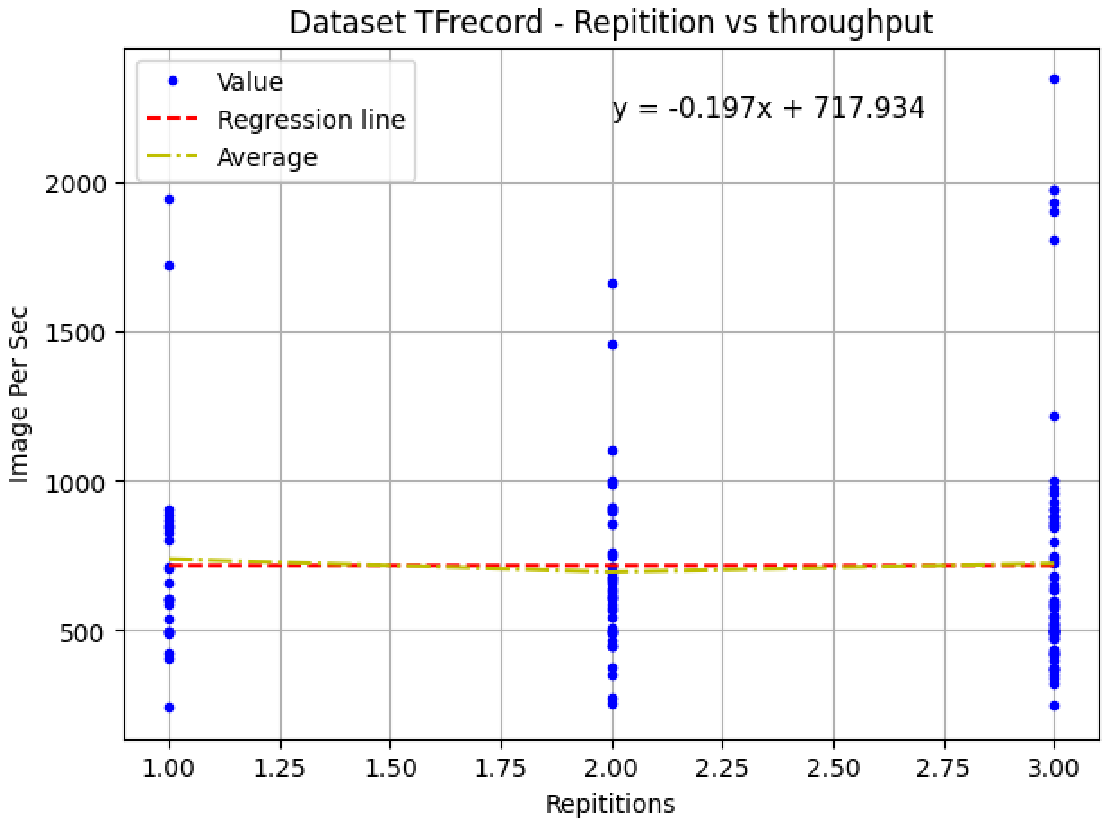



```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: 12.190651711011888
Intercept: 412.7072624102942
r-value: 0.36381221287145193
p-value: 7.385113433996048e-06
```

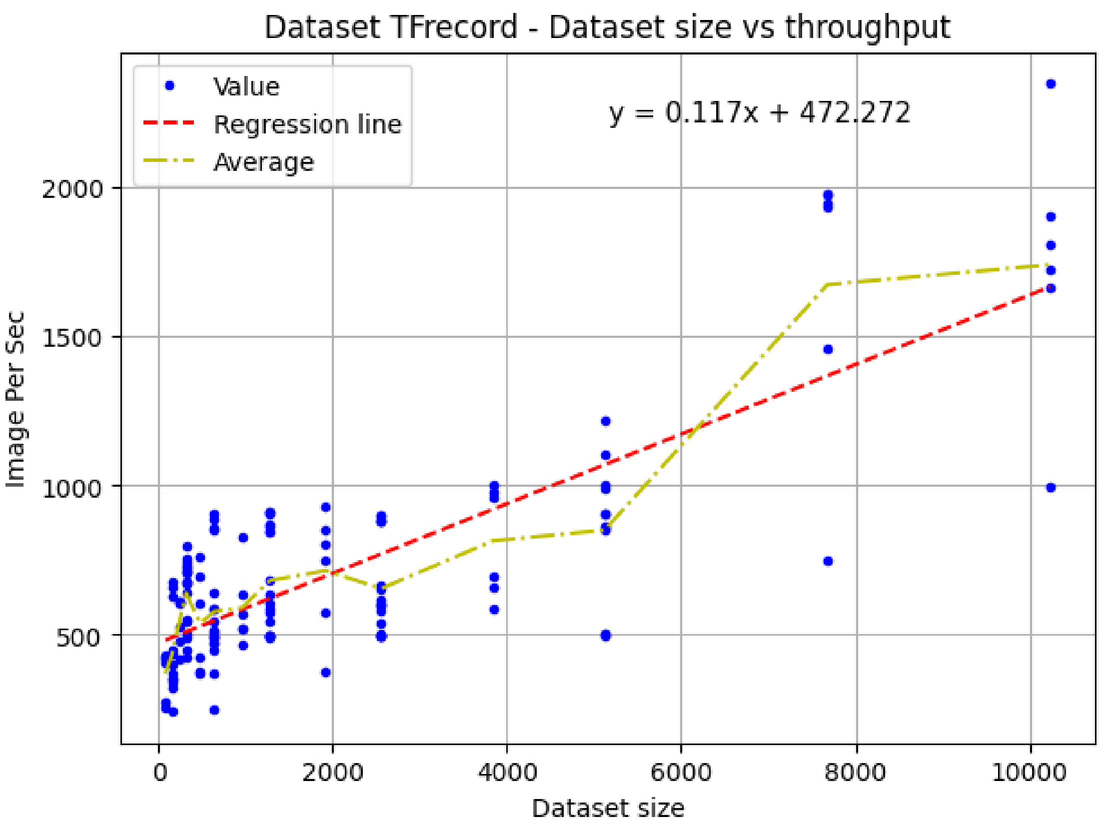
```
std-err: 2.6192408110792726
```



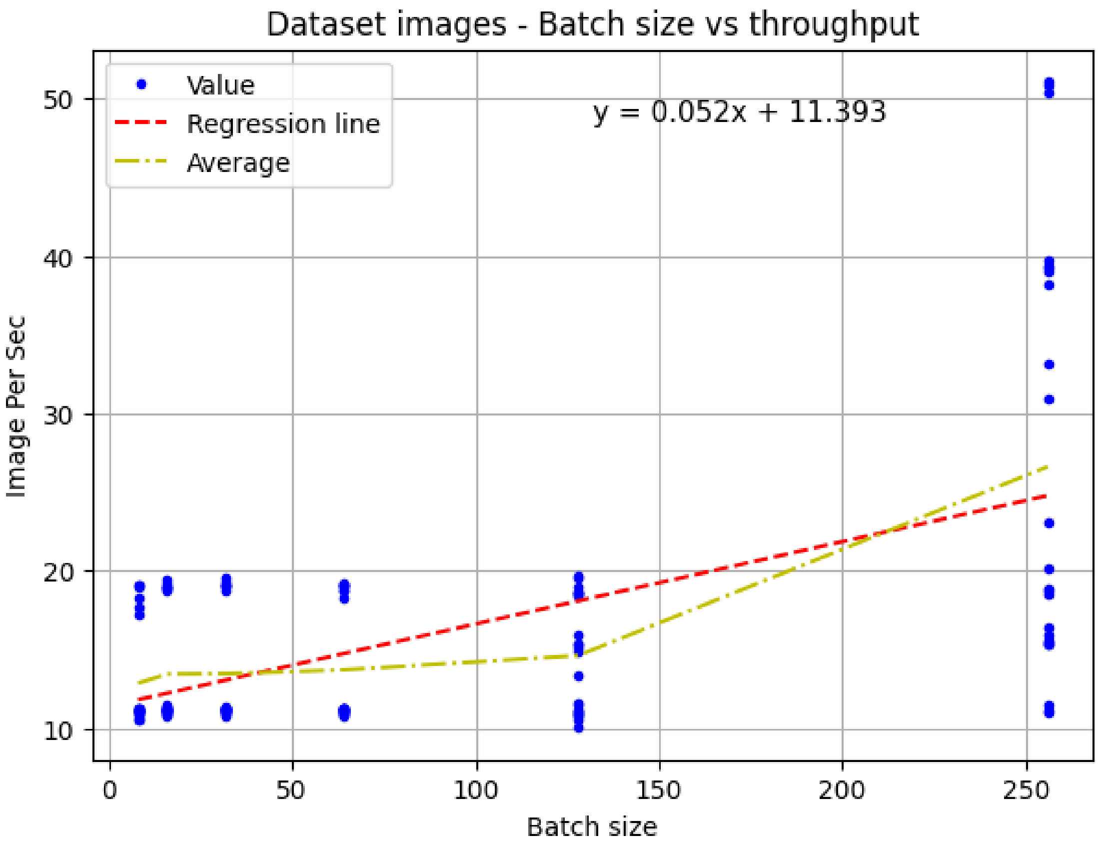

```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: -0.19742973805810554
Intercept: 717.9342245743936
r-value: -0.0003928001641O535523
p-value: 0.9962718812775484
std-err: 42.179041179276396
```



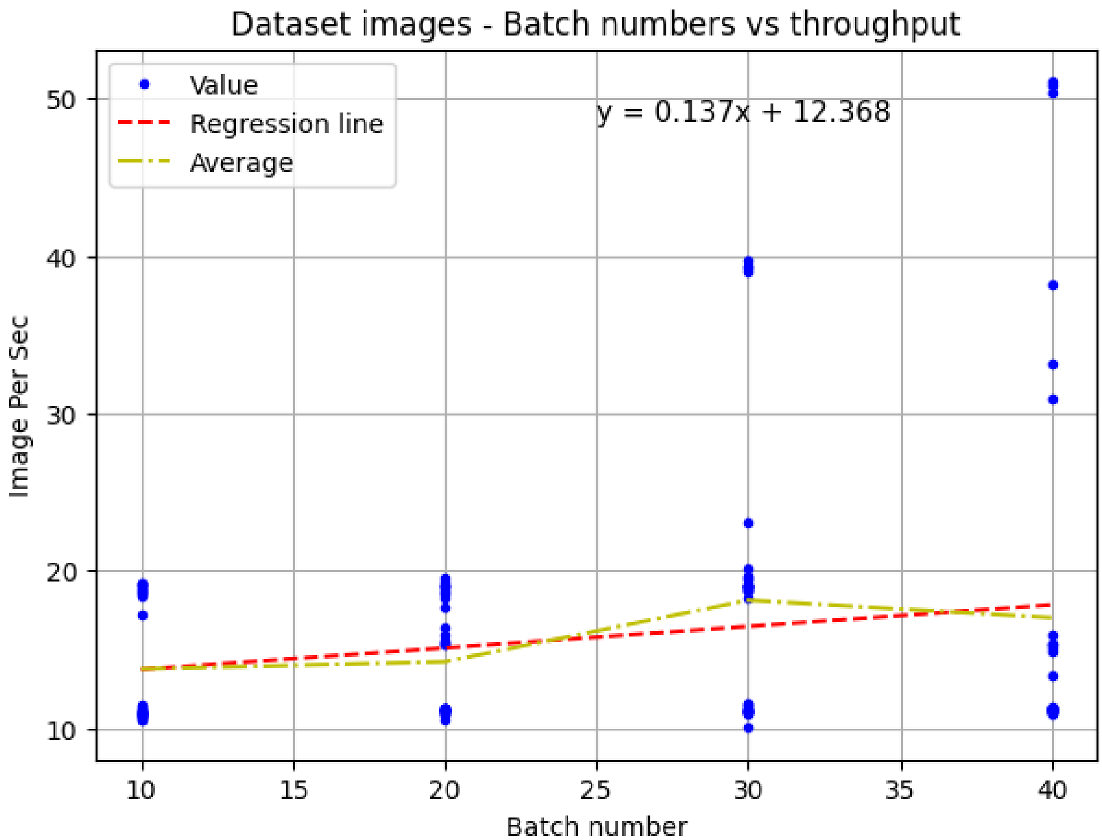

```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: 0.11676254751077336
Intercept: 472.2722054129673
r-value: 0.79522385572608
p-value: 1.1642915665499093e-32
std-err: 0.007470832093836426
```



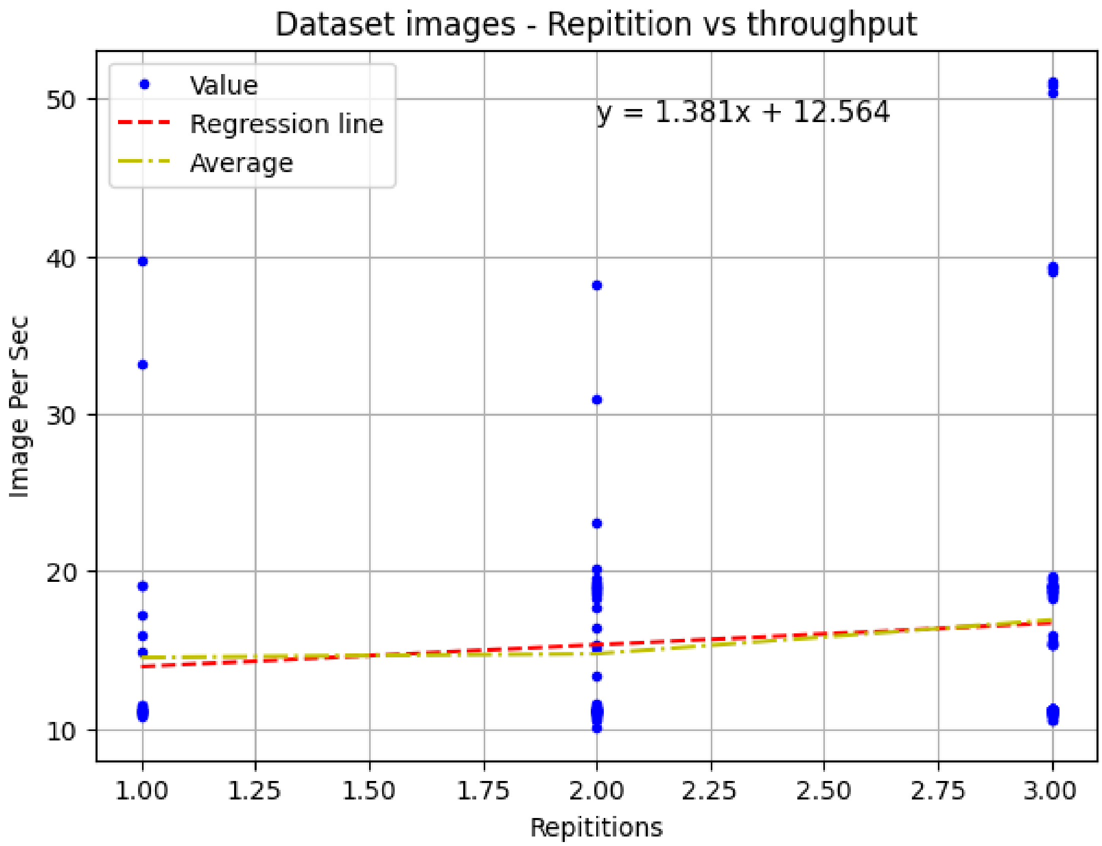

```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: 0.05230087787859328
Intercept: 11.393162846438978
r-value: 0.5675615608186098
p-value: 1.192495757918909e-13
std-err: 0.006366874366876663
```



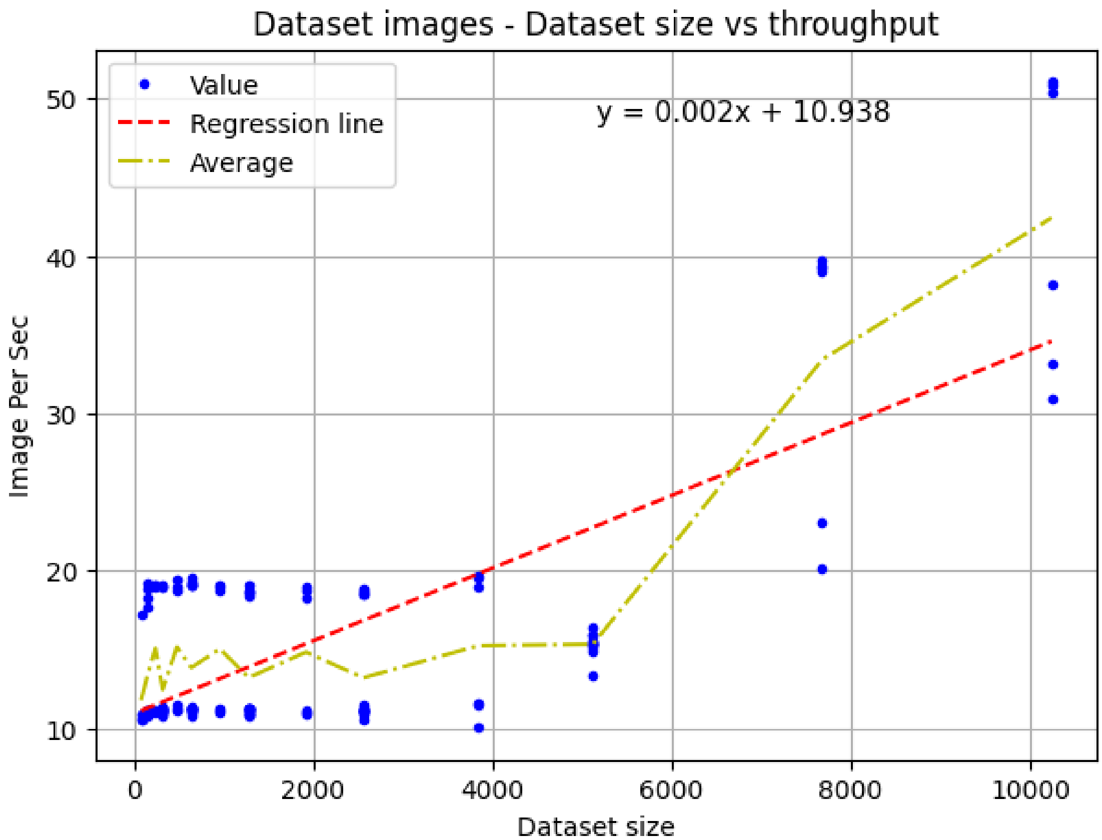

```
---------------------------------------------------------------------------------
----------------------------------------
---------------------------------------------------------------------------------
----------------------------------------
Slope: 0.13672738306093055
Intercept: 12.36825201171755
r-value: 0.19149969698486513
p-value: 0.02148843382738677
std-err: 0.058807172666624975
```




```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: 1.3809682094528921
Intercept: 12.56417743285073
r-value: 0.1289451499451333
p-value: 0.12348878724768406
std-err: 0.8912379479448461
```




```
--------------------------------------------------------------------------------
----------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
Slope: 0.0023089130991036633
Intercept: 10.937719080123122
r-value: 0.737998815805416
p-value: 4.9834832319069324e-26
std-err: 0.000177167523983206
```



----

6 Section 3. Theoretical discussion

6.1 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.. 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.

6.1.1 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%)

6.1.2 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.

6.2 Final cleanup

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

[ ]:

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

[ ]:

```

```