Big Data Coursework - Questions

Data Processing and Machine Learning in the Cloud

This is the **INM432 Big Data coursework 2025**. This coursework contains extended elements of **theory** and **practice**, mainly around parallelisation of tasks withSpark and a bit about parallel training using TensorFlow.

Code and Report

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

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

Submission

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

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

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

Introduction and Description

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

This coursework contains **3 sections**.

Section 0

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

Section 1

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

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

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

Section 2

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

Section 3

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

General points

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

The **tasks** are **mostly independent** of each other. The later tasks can mostly be addressed without needing the solution to the earlier ones.

Section 0: Set-up

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

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

Imports

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

```
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 2.18.0
```

Cloud and Drive authentication

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

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

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

```
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...
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/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 `europe-west2` as it is closest to us geographically. This way we don't have to specify this information every time we access the cloud.

```
PROJECT = 'bigdata06'  ### USE YOUR GOOGLE CLOUD PROJECT ID HERE. ###
!gcloud config set project $PROJECT
REGION = 'europe-west2'
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 = europe-west2
[core]
account = sara.iqbal@city.ac.uk
project = bigdata06
[dataproc]
region = europe-west2

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://bigdata06-storage/...
```

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.

Utility functions for image display **[RUN THIS TO ACTIVATE]**

Show code

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-3.5.0-bin-hadoop3.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-3.5.0-bin-hadoop3"
import findspark
findspark.init()
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)
```

```
/root
W: Skipping acquire of configured file 'main/source/Sources' as repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does r
3.5.0
<SparkContext master=local[*] appName=pyspark-shell>
```

Section 1: Data pre-processing

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

1.1 Getting started

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

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

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

```
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.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_si
```

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

Map functions

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

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

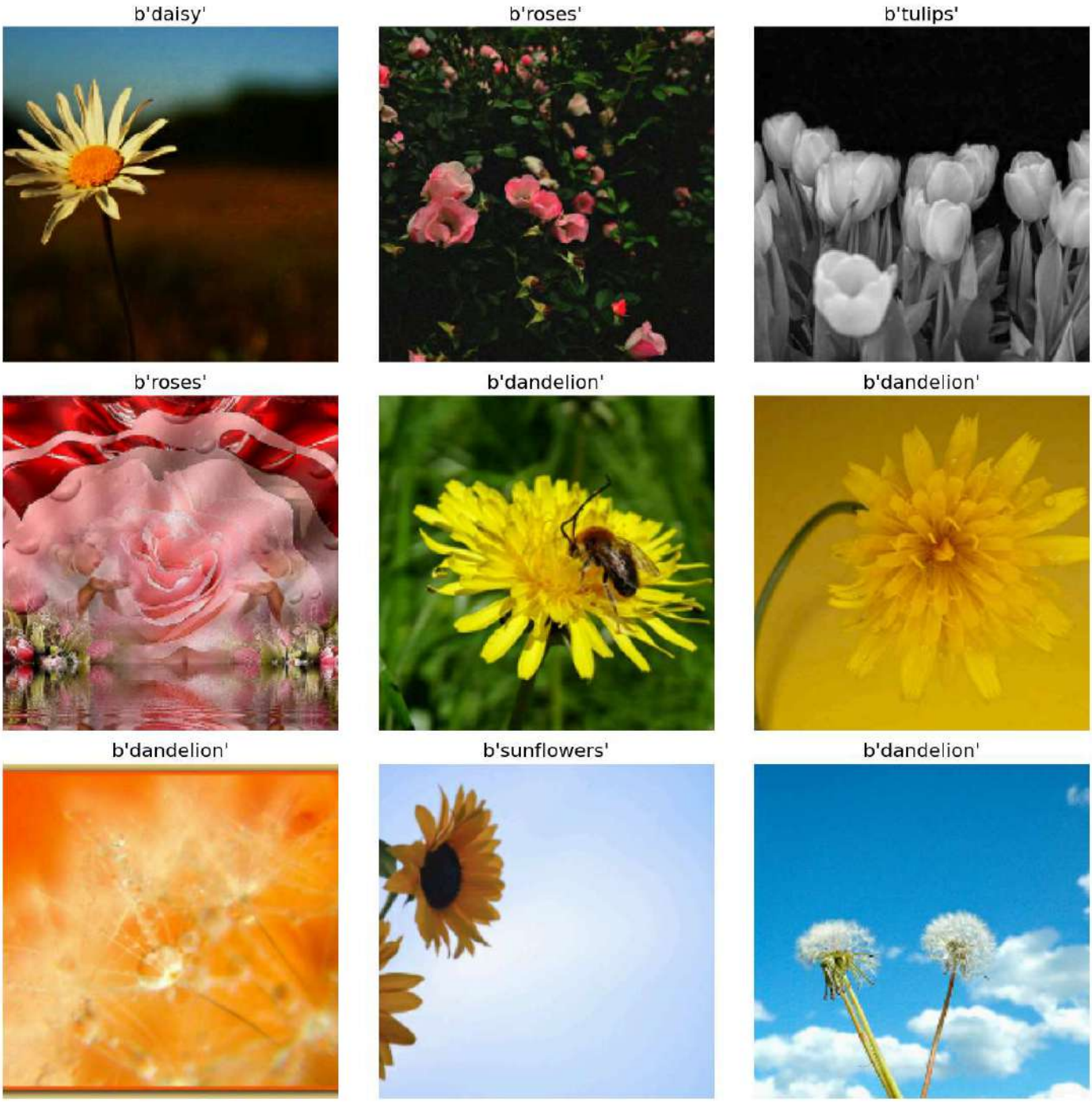
```
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label
```

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

```
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).

```
display_9_images_from_dataset(dsetResized)
```



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

```
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), ['daisy', 'daisy', 'dandelion', 'dandelion', 'tulips', 'daisy', 'roses', 'dandelion', 'tulips', 'rc
Image batch shape (10, 192, 192, 3), ['dandelion', 'dandelion', 'dandelion', 'roses', 'sunflowers', 'daisy', 'dandelion', 'daisy', 'tuli
Image batch shape (10, 192, 192, 3), ['dandelion', 'roses', 'sunflowers', 'roses', 'roses', 'roses', 'daisy', 'dandelion', 'dandelion',
Image batch shape (10, 192, 192, 3), ['roses', 'tulips', 'dandelion', 'tulips', 'dandelion', 'daisy', 'daisy', 'daisy', 'daisy', 'dandel
Image batch shape (10, 192, 192, 3), ['sunflowers', 'sunflowers', 'tulips', 'daisy', 'sunflowers', 'tulips', 'roses', 'dandelion', 'rose
Image batch shape (10, 192, 192, 3), ['roses', 'tulips', 'daisy', 'tulips', 'daisy', 'dandelion', 'tulips', 'daisy', 'sunflowers', 'tuli
Image batch shape (10, 192, 192, 3), ['roses', 'tulips', 'sunflowers', 'roses', 'daisy', 'roses', 'tulips', 'dandelion', 'sunflowers', '
Image batch shape (10, 192, 192, 3), ['dandelion', 'tulips', 'tulips', 'sunflowers', 'tulips', 'roses', 'tulips', 'daisy', 'tulips', 'tu
Image batch shape (10, 192, 192, 3), ['dandelion', 'roses', 'sunflowers', 'daisy', 'sunflowers', 'dandelion', 'dandelion', 'tulips', 'rc
Image batch shape (10, 192, 192, 3), ['daisy', 'sunflowers', 'daisy', 'roses', 'tulips', 'roses', 'roses', 'roses', 'roses', 'tulips'])
```

```
# This is formatted as code
```

1.2 Improving Speed

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

Recompress the images

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

```
# 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,), ['sunflowers', 'roses', 'tulips', 'daisy', 'daisy', 'sunflowers', 'sunflowers', 'sunflowers', 'dandelion', 'dai
Image batch shape (10,), ['daisy', 'tulips', 'roses', 'sunflowers', 'sunflowers', 'daisy', 'sunflowers', 'sunflowers', 'roses', 'roses']
Image batch shape (10,), ['tulips', 'tulips', 'roses', 'tulips', 'sunflowers', 'roses', 'daisy', 'daisy', 'dandelion', 'sunflowers'])
Image batch shape (10,), ['tulips', 'tulips', 'tulips', 'tulips', 'tulips', 'daisy', 'sunflowers', 'daisy', 'dandelion', 'tulips'])
Image batch shape (10,), ['roses', 'daisy', 'tulips', 'tulips', 'dandelion', 'roses', 'daisy', 'sunflowers', 'daisy', 'dandelion'])
Image batch shape (10,), ['daisy', 'daisy', 'sunflowers', 'sunflowers', 'sunflowers', 'dandelion', 'dandelion', 'dandelion', 'roses', 's
Image batch shape (10,), ['sunflowers', 'tulips', 'dandelion', 'dandelion', 'sunflowers', 'tulips', 'dandelion', 'tulips', 'dandelion',
Image batch shape (10,), ['tulips', 'roses', 'sunflowers', 'sunflowers', 'daisy', 'roses', 'roses', 'dandelion', 'roses', 'tulips'])
Image batch shape (10,), ['tulips', 'dandelion', 'dandelion', 'tulips', 'tulips', 'dandelion', 'daisy', 'tulips', 'dandelion', 'dandelic
Image batch shape (10,), ['sunflowers', 'daisy', 'sunflowers', 'roses', 'dandelion', 'tulips', 'tulips', 'daisy', 'sunflowers', 'sunflow
```

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://cloud-samples-data/ai-platform/flowers_tfrec/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
```

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.

```
GCS_OUTPUT = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/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**.

```
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 :-)

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

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

1a) Create the script (14%)

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

You need to:

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

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

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

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

v) The code for **writing to the TFRecord files** needs to be put into a function, that can be applied to every partition with the 'RDD.mapPartitionsWithIndex' 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%)

```
# import required libraries
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 pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row
```

```
Tensorflow version 2.18.0
```

```
# define variables
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg' # glob  pattern for input files
PROJECT = 'bigdata06' # project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # bucket storage
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output file names
PARTITIONS = 16 # no of partitions we will use later
```

```
# i) Copy over the mapping functions (see section 1.1) and adapt the resizing and recompression function 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(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



    import tensorflow as tf

    def decode_image_and_extract_label(filepath):
        # Reads the image and extracts the class label based on the file path
        image_data = tf.io.read_file(filepath)
        image_tensor = tf.image.decode_jpeg(image_data)
        # Extracts the flower class name from the folder name
        parts = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
        flower_label = parts.values[-2]
        return image_tensor, flower_label

    def resize_and_center_crop(image_tensor, flower_label):
        # Resize and crop the image to fit within the target size
        image_width = tf.shape(image_tensor)[0]
        image_height = tf.shape(image_tensor)[1]
        target_width = TARGET_SIZE[1]
        target_height = TARGET_SIZE[0]

        # Determine the resize factor based on aspect ratio
        resize_factor = (image_width * target_height) / (image_height * target_width)
        image_tensor = tf.cond(resize_factor < 1,
                               lambda: tf.image.resize(image_tensor, [image_width * target_width / image_width, image_height * target_width / im
                               lambda: tf.image.resize(image_tensor, [image_width * target_height / image_height, image_height * target_height /
                              )
        resized_width = tf.shape(image_tensor)[0]
        resized_height = tf.shape(image_tensor)[1]
        # Crop the image to the target size, centered
        cropped_image = tf.image.crop_to_bounding_box(image_tensor, (resized_width - target_width) // 2, (resized_height - target_height) // 2,
        return cropped_image, flower_label

    def compress_and_encode_image(image_tensor, flower_label):
        # Converts the image to uint8 and compresses it to JPEG format
        image_tensor = tf.cast(image_tensor, tf.uint8)
        compressed_image = tf.image.encode_jpeg(image_tensor, optimize_size=True, chroma_downsampling=False)
        return compressed_image, flower_label
```

```
# ii) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list of image filenames. (3%)
# retrieve a list of files that match the specified pattern.
filenames = tf.io.gfile.glob(GCS_PATTERN)
# spark context for rdds
sc = pyspark.SparkContext.getOrCreate()
# create RDD for files
rdd1_filenames = sc.parallelize(filenames)


# 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%
# sampling the rdd
rdd1_sample = rdd1_filenames.sample(False, 0.02)
# rdd for decode jpeg and label
rdd2_decode_jpeg_and_label=rdd1_filenames.map(decode_jpeg_and_label)
# rdd for resize and crop image
rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(lambda z: resize_and_crop_image(z[0],z[1]))
# rdd for recompress image
rdd4_recompress_image = rdd3_resize_and_crop_image.map(lambda z:recompress_image(z[0],z[1]))



# iv) Then use the functions from above to write the TFRecord files, using an RDD as the vehicle for parallelisation but not for storing the
# 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): #, height, width):
    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))

print("Writing TFRecords")
def write_tfrecords(partition_index,partition):
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # re-compressed image: already a byte string
                            label.numpy()
                            )
      out_file.write(example.SerializeToString())
  return [filename]
```

```
Writing TFRecords
```

```
import tensorflow as tf
import numpy as np

# Helper function for encoding bytestring features
def _bytestring_feature(list_of_bytestrings):
    """
    Convert a list of byte strings into a tf.train.Feature of type bytes_list.
    """
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of_bytestrings))

# Helper function for encoding integer features
def _int_feature(list_of_ints):
    """
    Convert a list of integers into a tf.train.Feature of type int64_list.
    """
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of_ints))

# Convert image and label into a tf.train.Example
def to_tfrecord(img_bytes, label):
    """
    Converts an image byte array and label into a tf.train.Example.
```

```
        """
        class_num = np.argmax(np.array(CLASSES) == label)  # 'roses' => 2 (index in CLASSES)

        # Create a one-hot encoded class array (this is not used in the current function)
        one_hot_class = np.eye(len(CLASSES))[class_num]

        # Prepare the feature dictionary
        feature = {
            "image": _bytestring_feature([img_bytes]),  # One image in the list
            "class": _int_feature([class_num])  # Class index as an integer
        }

        # Return a serialized tf.train.Example
        return tf.train.Example(features=tf.train.Features(feature=feature))

    # Function to write TFRecord files in parallel
    def write_tfrecords(partition_index, partition):
        """
        Write images and labels from a partition to a TFRecord file.
        The partition_index allows each partition to be saved with a unique filename.
        """
        filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)  # Set output filename with partition index

        # Create a TFRecord writer in the context manager to handle file closing automatically
        with tf.io.TFRecordWriter(filename) as out_file:
            for element in partition:
                image = element[0]  # Extract the image (already in byte format)
                label = element[1]  # Extract the label

                # Convert the image and label to TFRecord Example
                example = to_tfrecord(image.numpy(), label.numpy())

                # Write the serialized example to the file
                out_file.write(example.SerializeToString())

        # Return the list of filenames for this partition
        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.mapPar
    # return a new RDD by applying a function to each partition of this RDD, while tracking the index of the original partition.
    rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS)
    rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords)



    rdd2_decode_jpeg_and_label.take(1) #Take the first num elements of the RDD.
```

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

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

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

         ...,

         [[ 44,  48,  25],
          [ 44,  48,  25],
          [ 44,  48,  25],
          ...,
```

```
                [127, 126, 122],
                [127, 126, 122],
                [127, 126, 122]],

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

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

```
rdd3_resize_and_crop_image.take(1) #Take the first num elements of the RDD.
```

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

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

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

         ...,

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

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

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

```
rdd4_recompress_image.take(1) #Take the first num elements of the RDD.
```

```
[(<tf.Tensor: shape=(), dtype=string,
numpy=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xff\xdb\x00C\x00\x02\x01\x01\x01\x01\x01\x02\x01\x01\x01\x02\x
\x00\xfd5\xbc:l\xbfg\xfbK[)\xaa\xa3\x1d\xfb@;\x8f\xe9\x91\xf9cM9\x96)\x00r\xa2+6F\xd9e+\x1d\x03\xd1\xb6\x7fq\xb9(\xef\xf2n\xff\x00\xd
```

```
c\xda\x97\xad\xb1X\x97\x8bd)_N\xa0\xf3Z`y\x05>A_?
\xe6>\xba\x10\x89\xd1\xba\xc2\x92A]\xb7\x15\x8e\x92\xedo[\xb5\xa2Q,N2\x08\x1d\xd4\xfc\x8f\xd4j\xb2g*\x1b\x81I\r\n\xb2\x1cH0\xf1v`}\xd
\xf6\x98\xc1Pb\x0b\x8eC\xf7|gJ=\xd96\xac\x17]\xb9C\xf6\xa8\x1aR0Q\xc0\xe4?
\xb6\x8d\x05l*\x1c/(\xeb\x9d9\x92"Ja\x91\xbb\xfduw\xb8\xaaU)\xc9\xe3\rP"9\xc6y\x03\xee>\x9aQ\xce\xf7"\x80i.\xdd5\xeb%\x17\xd9\xd8\x10
Kc\x07Khi{r\\\xfe\'N@\x08\x88!\xbb(m\xedsx\xa2g\x07\xca\x80\x07\xd7R\xf7P\xc2\x86\x82\xb2\x1b\xbc\x8a\xd5/\x17\x12\xceX\xfa\x87K;!\x1
X\xdb@\xc7\xef\x01\xfd\xf4\x8c\xa0\xf6R\xbcm\xbd\xd1u\xda\x97\x11]l\x9c\xa9=\x99s\xf0\xba\xff\x00\t\xf9\xfb\xe8qN\xe6\x11\x95\x05k\x9
nJq\xd8\x8d\nRM\x95\x00R\'f*Esk{7\xc3P\x84\x00|\x07\x1d\xc7\xfa\x8f\xcfQ\xa5\x97k\xe8\xa9Ln\xb4,\xa9\x95\x18*pA\xf9i\x99\r\x1bU\xabR\
@w\x990\xa0\xfe\x1a\x1e\xf5e\xcaA
\x94,hr\x17\xe1\\\xe8w}\xd7&\x14\x14\xce"\x12N\x01o\x90\xd1ZqJ\x08\xb2\x98\xc6\xe8\xa7\xb0\xfcs\xa30P\\\x05(\x8b}\xd6\xb2\xdd\xc6XKI\
s\xb3p\xcac\xb1S\xdb\x1aZAH\xa1!\xa2\xdckAX)%\x9f\xd1Wo\x86F\x19U?\xcd\xf3_\x9f\xd3\xbf\xb6\x96~M.<#\xea\xd2*\xd4z\x88#
(c8\x9a\x12Fco\x97\xd4\x1f
\xfc\xb5\x9c\xf1\xb4\xa9D\xec\xdd\xe5_\xb2o\xf1\xdc\xe92T\x10\'\x84\xf8u\xf7\xfc\xfeZ\x985\x067\xd8\xe1q\x00\x85\xb8\xcd\xbb\xf6\xadU
\x83\xfd\xb4f.A\xde\xef\xd1\xc0\x1a\x9a\x9c\x8e\xfd\xd9\xbd\xb4n\xd8\\\x92\x8bl\xf4\xbd\xc2\xfe\xf6Y\x08\xf6\xd5\xa8\xa8\xdc\x17\x89)
(\xcap%\x1e\xe3\xf1\xd5ZK\x0e\x17*\x07\xb9Gw\x89+\xa3B\x0e\n\xca\x87\xf7[\xfe\xfaOR\xd3\x1b\xb77\x82\xb9,\xb8\xb4t\xb4\xed3/\xde\xec>
-\xaa\xed\xcd^\x88\x87\x93\x1f\x80\x11\xc7\xe7\xadi%\xb7*\x0c\x05\x19Zfw`\xa3\x00\x8c\xaa\x01\xef\xf5\xd0\x8c\x96)
[\x94\xa7ql\xaf\xf6\x81\x0c[\x1a\xb0+\xdb&\x91%\xbf+\xb7\xc3QN\xbd\xc5)_,$n<\x87\x8fLH\xa7\xbb\xae\x9d\x88S,\xaeZm\x0c\x11\xa4h\xbe\x
hs\xf7F\x8c\x1c8\\\x91\xa2UV\xb9\xee\xccI\xee\x07\xb9\xf9h\x8d"\x89\\p\xb4\xa6\xdbi[\x18u\x88\x06#\xe1c\xe0\xe9\xb2/\x84\x0b!)\xbam:\
\xd3K\x13\xe4\x16\xe5\x8c*\x8c\xea\xb2\xe3\x8c\xd9+\xb6\x94\xf6\x8e\x07\xb8\xc4\xd3\xbcA@\x18=\xbb\xe99\xb5\xb2m\xdc\xd1aue\x01p\xa4
{\xd9\xb5\xa8\xf9\x1b44\x80A\x08\x8d\xc1Z\x1a\x826\x0c2;\x90O\xd7^zMH\xf4\xf6\x92\xae\xd6\x95\x1dM<\x95{\xda\x95\xd0\x90#\x91X\xf7\xf
s\x8a\r\x14\x96i\x11eW\xa9\x1f\x070\xaa\x84\xfd\xf2H\x00\x7f\xa7\xe7\xa3\xe9]\xeb\xbdA\x14\xa8vm\x84[\xe9~!\xceY~9e\x03\x1c\xdb\xf8\x
\x9f\x1a4N\xb1er\x12\xe5wX\xa1\xfb=;\x92|gF\x04\x95\xcb\x95\x9a\xcb-
\xca\xa3\x99\r\x8f,\xc4\xe8\xcd\xa6\x8c\xaeH:\xb7I\xd4\r\xdf\x1b\xf4\xf7\xa5\x8d\x15\xba\x9d\x93\x8d\xdbpT\xbb(\x8dO\x98
U\x1c\xa4r\x0eK\x8f\x80\x03\x8e\\\xb2\x16\xee\x8d\xd2\x8a\x06\x82\xa6\xe2\x80\xe97\xe91\xb86\x8bEl\xbd\x13s\xb7\xa8\xc2\xd3\xcb&$\x88
c\xdb\xca#\x98\xd3\xc2\xdf6\x87V\xfaO\xd4\x1at\x8e\x8e\xfd\x145\x0e0\xf4u\x9f\xb2\x91I\xf6\x1c\xbb7\x7fu$kA\x9a\x88\xe5\x18(%\x94\x8f
<H\xecA\xf3\xc4\xfc\x88\xd2:\xb7\xb5\xa7{\r\xb4\xf7\xf0\xa6;\xaa!H\xd5\xcd%+\x11\x13\x11\xdf\xdck\x0bW#\x1f\x19.Eo(\xcb|\xb1VQ\x8a\xb
\xc6\xb6\x95G\x04/\x8f#\xdf\x1e\xff\x00\xfc\xef\xae\xb2\x15\x83|\xab\xde\x9f~\x90[\xe7i2%%\xdeG\x85p\x1a\x9e\xa9\xc9\x18\xf9\x03\xe4}
<\x81\xf2\xd31j\xa5guGF;-sl~\x91\x9bos\xc4)o\xb4)\x14\x8d\xd8\x82\x00\xe4\x0f\xd7\xee\xb7\xf6?
M:\xcd\\r|\xc8E\x85\xab\xb6\xe2\xda\xf6\xbb\xad!\xb9m\xe7Y\xe3#>\x988*~X\xf6:\xab\xda\xceB\x80J\xcd\xf7\x05\x96JY\x19\r+\x18\xcfu+\xf
<\xf6W\r<\x85\xf6\xe9\x19\xb8\xc5%bFc\xab\xa68\xab\x88\x0e\xe7\xf9\xc7\xd3\xe7\xaf1\xd4\'s#.o\xe6\x99`\xceWk\x1e\xe1\xaf\xb71\x92d\xc
GY:\xc7\xbbPI\r\x00c\xb8#\xe9\xf5)\x83\x17\xa6\xd0\xd1\xca\xa3\xd8\xf50\\.t\xd2\x00\xca\x19\x89R\xc3\x00\x80pH\'\xc8\x04\x11\xadM7\xc
8!\xd3\xb7\x10\x1c0\xc9\'8\xf6\xf7\xf1\x1a\xae\xbb\xaa=RMk\t\xb7\x1bp\xff\x00\x89\xec\x0f\x9e\nz=&\xf8\xb6\xd7\nN\xc7MW\r\xfe-
\xbfo\xa6\x91\x9a\x97\x94\xf7I\xc2abf\xe3\xe9\xa7o\x04\xa9$\x0f8BN5\xeb\xfe\x19\xea:\x89\xb5\xac\x8d\xc6\xb7\x1a\x04\x9ey\xb2O\x81@ZK
\xdc\xf5\xfb\x9e\x96\xc756\xd5\xb1\x9a\xea\xf2\x9e\x95\xa6\xdf,\xe2\x05\xac\xa9
p\x0f+ab\x8f\x969HH\x00\x03\xdf\xb6\xbe\x95\'\xc5\xfa&k\xbd\x00\xea\x8d\xb4\x1c@\xb3Wg\x8b>
(\x01d\xac\xf6\xe9\x0e\xc2{\xac\xdf\xff\x00\xa7\x0e\xc7\xbbmn\x94\xdd\xb7>\xea\xbb-
u\xd7q\xee\x9a\xdb\x95mcS\xc9\x13I!+\x1b\xe5\x1c\x021"I\xf4\xd7\xd3\xe2\xeaq\x17\x86\x0c\x10\x06\rw\xfb\x1f\x14\xb3\x8c/h\xca\xdd\xf7
[\x95=>*\x04\xb2qI\x00\x03\xb9o\x1d\xfe\x9f]|\x87\xe2\x8f\x8b\x84\xba\x11\xe8]I\x8b\x1d\x88\xf29\xbf!j\xe9\xb4\x80>\xdc8N\xf7%\x8e\x8
(gU*\xe8\xc5\x10\x80\x88Y\xbb\x02y\x85\x01\xb2\x18\x8dL\xba\xe7G\xd4\x8e\xc8\xc1c\x8dpM\x8el\xe6\xed\xdc};f\x95\x1c\xfb\x01\xc52\xdb;
\xae\x82\xa3\xec\xea\xb5\xee\xe0\xbc\x91\x86*LG\x93\xb9\x1f\x03c\xe1\x00d\x1d/\xa9\x97I\x0b\x1dd\xee.$8\x8b\xf6\x0b\xc1\xf0p\x05w\xe7
\xd2\xba-
\xb4\xe2Nn\xf1\xf4\xfd\xfe\xc9\xb8\xe4{l\x03\x82\x8c\x8bh\xdc\x92\xfd]O@!\x10\x96T\xb6\xd2L\x9e\x9dF\x02\xb1nm\x8e,\xc4\xf8r{\xf2\x00
I\xc9"R\x00\xecG\x8f\xa8?
\xdb\x1a\xf6\x9a/\xfcC\xebR\xea\x19\x06\xa0\x02l\x0b7u\xe1f\xcb\xd3\xe2km\xaa~\xfb\xb9!\x8e\xa3\x84\xcc\xc5\x87\x80\xbd\xf1\xaf\xb1\x
(\xa2?
\xd4\x8d{]4\xefx\xb9\r%\x0bE\xaf\x17\x1b\xf5\x1d\xa2\x98\xd3\xd1\xa0g\xf7o\xae\xb6c\xd44\x8c!\xed)\x0c\xf55\x17i\x83\xd6\xd5\xb1\xc9\
\xb0o.\xa3g\x04\x9c\xd8\xcf#\x91\xd9z\x00\xc6\xb8\x84\xef\xa6\x97\xbd\x87\xbcx\xeem\xbfZ\xb6\xea\xbb-
\xbaj*J\xfb\x94i.e\xe2\t\x95\x80\xe3\xe0\x10\xa3\x1f\x17\xc4{\xb1\xc6\x15\xd7\xc3\xaa\xd2;\xd1\x7f\xcb!\x04\x86\xd85|s\x9b\xef~\x06\x
i6\xc6\xe7N\xe8\x9a\x08\xc7z\xc0\xc1\xcf\xf1\xf5!\x05\xee
]\xaf\x14\xb77\x16\x99\xf7\x17L\xb7\nz\x12\xd4\xc6\x8bew\xe4\xf2F\xec\x0bpyXrnD\xbe\x0ep1\x96\xc6\xac\xed8\x91\xdbg\xf6\xb8w\xae\x0f\
\xe2\xab\xf7q\xf7\xf0\xf9\x00v$\x1f\x85\x87\xd3$\xdc\x83\xb8\x06\x8bG\x8f\xe7\xf6*\xa2P8\tu\xa7\x7f\xd9\xae\x16jK5\x81\x8d%\xcaj\xb6Z
\xeb4\xc2\x1a\xd6\xbe1l\xb0h\xfd\x0ew_a\xe4\n\xe4r\x8e$\xdc,\x1aG\xdc\xe4\xaf\xbfl\xd2\x94\x94\xf4\x91\xd7\xd2\xc4XQ\xdc\xe9@s\x84#\x
\x17l\xf6\xca\xf9\xf7\xd0\xe3\r\x87QN\'i<\xb7<\xf7\xfc\xd5\xc1\x04\x8caF\xeeK\xce\xe0\xb4\xd2\xd1\xd9\xb7\r\xb2\xe8k\xea\xe02\xb5\r%\
\xba\xc1O\xe9n\xd6\xa8T-
```

```
rdd5_partitions.take(1) #Take the first num elements of the RDD.
```

```
[(<tf.Tensor: shape=(), dtype=string,
numpy=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xff\xdb\x00C\x00\x02\x01\x01\x01\x01\x01\x02\x01\x01\x01\x02\x
\x10\x00\x01\x03\x03\x04\x01\x02\x04\x04\x04\x04\x06\x03\x01\x00\x01\x02\x03\x04\x05\x06\x11\x00\x07\x12!1\x13A\x08\x14"Q\x152aq#
\x00\xc4\xd6tx\x94zz\x12\xc9\x01i\x18#\x1a\x8e\xa5\x80\xfd\xa1SS\x90\xe4C!n`\xe3\xc0:\xd1\xa3<KT\x06\xbdj(2\x04\xa6\x15\xf4\xa4x_\xbe
\xe3\xcep5\x13\xe4\x05\xa5\x00X\x81\x91,\xba\xb5\x12\xaf\xf8\x99_!\xc8\x103\xa7cF;\x93\x91F\x17T\xef\xf6\x15E0\x96rR0H\xd3\xcb\x18\xb
<\x84B\r;5j\\s\xe2\x96V\tI\xef\x07\xc8\xd4m\xeb\x03BQ\xc9O\xcc\xa5\xa8\xd8Si\x8er[d\xfb\xe7\x18\xd3\x107fx\x15\x12e\x10\xbc\x15\xe8%J
\xcb1\xe4Z\xdc\xe9R\xde\x16\'A\n\x08NH\x19\x1c|\xea\xc5u@\x14\xc4\xb5\x93\xa83K\xb8\x8d^\xa8\xa6\xd4\xac\xa5\xd3\x82\x07\x81\xa7/\x16
\x80}\xf4C\x16 z\x8a7%B\xb7)\xf2@Qd\x1c\x1f cN\\8\xcf\xc4\x02\xe4Kx\xf4JS,\x1e-
\xa7\xa1\xdfZNO\x0f\x1c5\xcaz\x837u\r\x89@\xfc\xb3`\xf7\xec4\xa3\xe3\xe3\x02412\x82\x9db<\x97\x14\xf8\x8f\xef\xed\xa9\x8e!\xca\x10b5,
\x89\x15\x9aS\xe0\x1ec\xf6\xd3B\xdfp>g\xef\x90J\xdf\x01\xc6@\x1fr5\xe3\x8fSgsBe)\xcbm\x01\x93\xfc\xbaO\x03s\xd3r|6|en\xc4\x8a\xc4\xfd
\xc5R\x9do\x19_"?
\xe0\x9eX\x19$\xa0\x8c\x12A\x18\xe5y~\x11J\xc8\x9d\x19\xd0\xf1<\xd0\xfe\xc6\xeef\xfa\x81C\xee\x92\xa5g\xaf\x03\xef\xa8Qu\xb9[\x13)\xa
j\xb3^y\x05\xc0\xa4\xf2\xef\x97\x9d \x13\xceyT\xd4\x89r\xdc\x8c|\x88`\'\xeb\xf2\xac\x9dV\r\x08\'pf-
a1d\x17PR\x00\xfd25\xe0\xc6\x00\x12\xd5\xaa\xf32\x9b\xedI#>\xfazd\xad\xdcS\x89\x1a\xb3YD
\x16\xdar\x9cx\x1eF\x9d\xccT\x1e3\x8d2\xedq\xc2P\x140;\xec\xeb\xc33,\xf1E=\xc9S\xee\x99\r3\xf4\xa3\xdb\xb1\xa6\x1c\xacD\xce\n%O\xf9\x
\xbaT\xa2\x82J\x87\xa8\x87FRx\x83\xf4/\xc7\x98\x9cE\xbe;\x84v\xadnbnH[[\xb8\x97\x9c\xf5"\xe6Z\xbf\x1a\x83!\r5Or\x7f\x13\xea6\xa4%\x1e
3\x03x\xd9,tg_\xc5\xce3\xa5\x1e\xe2>\x0b\xa9x\x84\xbc0\xaf\x07\xbd)h\xac|\x9e\x88\r\x95sk\xa0GD\xf9\xd7\x95a\x9dN\xe9p28$\xe4\xfd\xb4
[\x96&\xa6t}\xc6\x99\xa8K\x9a\xcb^\x14p1\xa8\xf9\x0eR\x81j\x9dAZ\xfd\xc2\xe4\x99>\x8a\x10R\xacw\xde\x9fb(\xf5r\xa6T\x97\xdb!)ec#\xbc\
\x06\x19[\x8e(+!A\xb2\xc9=\x8c\x9d}\x16D\xf51\xd4\xe3c!\x1a\xe2\xd5T\xba^\xf1\xdaT=\xd7\x99!\xb7n{e\xc6D\xd8\x0c\xce,\xa1\xc7R\xe0KHI
\xb6\xf8\xb7\xd9\xe5\xd1.
{^\x15~m5J\x9a\xfcZ\xa2H\ty\xa7\x03N\x82\x9c\xe5*A\xc2\t\xc9\xca\x16\x08R\x808\xd2\x132\xf1apw\x8c\xda\x98\x91\xde\x1f\xf0\xac\xda\xf
O\xc3u\xd2:B\x8f\x1c\xb6z\xf25>o\x07\x13\x8bM\x18\xfc^[\xa9\xa3\xb9\x827\x16\xc0\xbcv\x86\xee\x9d`\xdf\xb4\x95\xc2\xa9\xd3\xdc-
\xbc\xd9W$+\x07\x1c\x90\xaf\xe6N=\xc6\xb9\x8c\xad\x89\xb8\xb0\xdc\xe9+\x86NB\x0e.
[KW\x8c\x1c\xff\x00}i\x06\xa6\x86\x12|zQ\x92\xd0!\xae\x88\xe8\xe3Ha\xf7\x9a\r\xceT\x8aR\xa9\xb51\x86\xf2\x0b\x83R\xe4Qq\x8a#\xf6\xc5\
\xf0Y\xe3\xc4\x1d\x0b\x1d\xee\x1e3m Rh\x94W)\x1c\xdfZ\x0b\xab9
```

```
\x9e\xf5=\x0b\xb9M\xda\x91\x05e\xd8\xf0$\xd7\xd6\xfcD\x85)^\xdfa\xa2T%\xaeNh\x1a\x96\xb2\xf6\x8ag\xc8\x054\xc8\nP\xc9\x1f}X\x14(\xa8\
\x9f\xac\x13\xa1%\x87S\xdd\xce-]E\x84%\n\nO\xd3\xe0\x9f:!\x91\xaez\x80\x9c]\xbd#8\xe0\x0f\xb6@\x07\xc9\x1a?
Ve\x89\x7f@\xa9Se\xe0\xb201\x801\xa3\x19\x16\xfa\x81F\xe7+\xbe\x8d&|%|\xaa\x8f\x8e\xba\xd1eR\xcbbz\xa2\xf3\xf0\x8a\x9d2\xa0U\'>|\x9dJ
\xea\x9a\xe3pC\x8b9\x1d\x85j\xa4n3J\xee\x18\xd2n\x14\xba\x03n\x00A\x18\xf3\xa7\x87\xb8\x05D\x972\x81\x16\xaa\xc9ZR\x02\xbc\xf7\xa2\xe
\x02\xa0\xf3jBS!\xb7HSj\xe0\xa2W\xf9\xc9l$+\x008\xac\xe4\x11\x8f\xa4\x04\t\xf3\xd4\xc5bWz\xe3\xca\xb5\xee\xb9\x82\xdf\xa5\xb7U\xb6j\x
\x9d,\x1amu4\xaf\xb7q\xd5\xbfp\x9an\xe1\x84\x9be\xca\x93\x10nP\xca\x9a\xa9\xc1\x96\xa2Q
\xaf\x86\x16\x80Bx\xf8\xefDT\x91\xa8\n\xc1w\x07\xf7\xb7\xe1\x0e\xc2\xdc\xeb\x15\x8d\xba\xbd\x9eq\x17+l\xad\xd8\x15\xd5\xa0\x07\x92\xe
\xce\x9e\xb1\xfbh_\x08\xca\xbcXu\r2\xb63\xc8O\x9bWf\xdb]{\x7fxJ\xb3\xee\x88\x05\xa9\x90d)\xa7\x12|
(\x03\xd2\x87\xdc\x11\xd8\xfbk\x92\xc0\xe1%L\xeb!\x19\x16\xc4$\xb7iJL,>\x8c\x1cx\x07K\xbeGq\x95R\x05^\x94\xf4I^\xab
\x80T1\xfb\xe9\x19S\xe6\x185\n-
[\xa6dH\x0b\x89\xeb\x10\x0fA#KU\xd4\xd2A\x94\x97]\x1e}A\xc4\xcaJI\x1c\xb2u>Eo\x88X\xcf\x19D\xd56\xb8f\xf2\x8f\xcb\x888\x03>\xfa\xe7\x
\xdad\xf4?,\xa4)^\xe4\xea\xccn\x0f}\xc9\xc8#\xb8\xc4\x8fQ\x84\xf5\x18\xbc\xcc\x8c\xab\x18\xc0:w\xccg\xb6\xa2\x86\xf7\x8d\xf35\x07J\xd
\x191\xbb\x82\xea\xa6#\xd4\xe2\xa4\x0f
\x0c\x8d\x08Z0\xee\xbb\x9e\xe7\xda1\xe52\x1ci\x00\x9f|h\x99.\t\xeeE\xa6S\xa6S\x9f\xe2\x84\x90>\xc4\xe96Gp\xbb\x10\x86\x1dq\xbe>\x8c\x
SN!I)nBT\x91\x85\x02\xa2\n\xd3\xd2J\x82u\xdc\x1e\xed\x89\xc4!\x81:\x8c\x1b\xfa\xcd\xb9\xe6m\xedJ\xf9[\xe9\x14\xf8s\x95&L9m%\x84K\x8f\
rR|\x9e\xb3\x81\x93\xac7\x8d5<\xb4\xec9N\x9b}\xf1\xbb\xb4\xb7\x8b\x14\x8a5V\x87T\xa8\x11\t\x1f4\xf1G\xfc5\x80\x02\x9e\x05\x7fR\x92\n\
E\xc9\x92\xaaq\xdaB\xd2\xb6\x86G\x8f\xb6\xb3\x80mMf5
\xd3&@K\xa9[\xd2\x1ag.\x00\\t\xe1)\xc9\xc6I\xf6\xff\x00\xef\x8f~~\x7f\x16\xcdMG7\xa9\xbf\xbe\x1d\xbf\xc2{ow[d\x11]\xbe\xee\xc4\xcb\xa
\x96\xa4\xbe\x94!xWm\xb88\xbc%%\xc4\xe5*CC\xb3\xc8\x81f\x0f\xa6\xa2\xe3\xf7\xf7%\xcd\xe6\xf2zQ\xaa\xfe\xf2\x07\xc6_\xf8_\xdb\x1byhP\x
<YU\x85\x88/\x8e\x8c\xec\xba\xa9d)-\xe4\x121\xfb\xe9\xe5\xaa.\x8c\x876\xa8=,qYs\x1e\x123\xa9\xb2\x1eGQ\x8b@J\x8f\x9c\xab\xf2/?
\x01\xd4\xb6\x15\xda\x8e\x87\x8f\xdex\x9b\x84T)\x11\xdd\x01\xc4\xa7 \x0e\xf9\x8d+\xc8\xc8\x13\x16\x8c \xa2\xe7[\x8e\xbdH\x87\x14\xa5
\x05\x0e\xfa:\xf8\xe7\xf2\xfc\x86\xf2j2\x94\x08\x1d\x16\xe1*\xa8\x9fK%\'\xecu\xf6\x1e)\xca1\x02\xd1d\x8a\x97\xd1g\xcbq\\\x83\x84\x00:
\x11s\xc5F|\x84\xb6\x14\xea\x89\xc1\xec\x13\xaa\x14\x98\x04Q\x90\x9e\xab\xa9\t\xfaNS\xfb\xe8\xc1\xdcS\x83>\xa4\x7f\x875:\xd9\xba\xec\
b\xcc\xb5\xc3\x14Z\x8a\x96\xbb}\xe6\x1d\xf4\x92\xf4W\xcfHiC\xf9Fq\xf6\x03:\x95\xb1\xaf\xad_\x07\xa9H\xc8}\x10D~Y0k\xd5)q\xb6\xd1\xab\
\x9c\xf6{\xf7\xd5\x01J\x8a\xf8\x8ag\xae\xa4\xfd\xb7\xdeG-\xcb\xfe&\xde3V\x8e\xe5\x0e\xb0\xd3\xad&
<\xf2\x946\xfb\x888%>\xc8=\xfe]o\xa8\x01\xe3\x04co\xdc;\x8a\xef\x89?
\x847\xaf\xcb\x9aU\xc9\xb6\xb4\x06\xa3\xcem\x05n\xc7`}\x12\x02F0@\xe8+\xf5\xd2sx\xde\xa6\xd7F;\x0f\x91\xc0\xd3L\xc4\xd5}4\xb5\xc9\xa2
\x99Q\x9cSO0\xb4\xf1R\x16\x0f`\x8dr\t\xa6\xe2gUx\x95\xb1(\xea2\x9d}~\xa3I +\xb0\x08\xc9\xd6\xa9
\xdc\xc6\xaa\xdc9\xf82\x89\xb9R\xbe(md\xedu\xbe\x9a\x94\xe4\xce\xe2\xf4w\xa2\xad\xe6Lu\xfd\x0e\x87\x02Zp\xa1\x05*
\xaf\x89\x03=\xf5\x9d9y6ABO\x93\x8f\xa6I\x9feh{\x97wm\x8c4=\xb8v\xaa\xe3\xd4\x08C\x14\xc9\xb1\xa2\xb6\x811%D\xa5\x97\x83?
B]B\x8a\xfe\xa6\xf0\x87\x07\xd4\x90\x92Ki\xf9?\xd7\xdf\xabq~\x89\xfa\x06O\xa9:s+@/@\x93\xf7?
```


```
rdd1_filenames.take(1)
```

```
['gs://bigdata06-storage/tfrecords-jpeg-192x192-2/flowers0.tfrec']
```

```
### CODING TASK ###
# import required libraries
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 pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg' # glob  pattern for input files
PROJECT = 'bigdata06' # project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # bucket storage
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output file names
PARTITIONS = 16 # no of partitions we will use later

# i) Copy over the mapping functions (see section 1.1) and adapt the resizing and recompression function to Spark (only one argument). (2%)
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 recompress_image(image, label):
        # this reduces the amount of data, but takes some time
        image = tf.cast(image, tf.uint8)
        image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
        return image, label

    # ii) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list of image filenames. (2%)
    # retrieve a list of files that match the specified pattern.
    filenames = tf.io.gfile.glob(GCS_PATTERN)
    # spark context for rdds
    sc = pyspark.SparkContext.getOrCreate()
    # create RDD for files
    rdd1_filenames = sc.parallelize(filenames)
    # 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%
    # sampling the rdd
    rdd1_sample = rdd1_filenames.sample(False, 0.02)
    # rdd for decode jpeg and label
    rdd2_decode_jpeg_and_label=rdd1_filenames.map(decode_jpeg_and_label)
    # rdd for resize and crop image
    rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(lambda z: resize_and_crop_image(z[0],z[1]))
    # rdd for recompress image
    rdd4_recompress_image = rdd3_resize_and_crop_image.map(lambda z:recompress_image(z[0],z[1]))

    # iv) Then use the functions from above to write the TFRecord files, using an RDD as the vehicle for parallelisation but not for storing the
    # 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): #, height, width):
        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))

    print("Writing TFRecords")
    def write_tfrecords(partition_index,partition):
      filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
      with tf.io.TFRecordWriter(filename) as out_file:
        for element in partition:
          image=element[0]
          label=element[1]
          example = to_tfrecord(out_file,
                                image.numpy(), # re-compressed image: already a byte string
                                label.numpy()
                                )
          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.mapPar
    # return a new RDD by applying a function to each partition of this RDD, while tracking the index of the original partition.
    rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS)
    rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords)
```

```
Tensorflow version 2.18.0
Writing TFRecords
```

1b) Testing (3%)

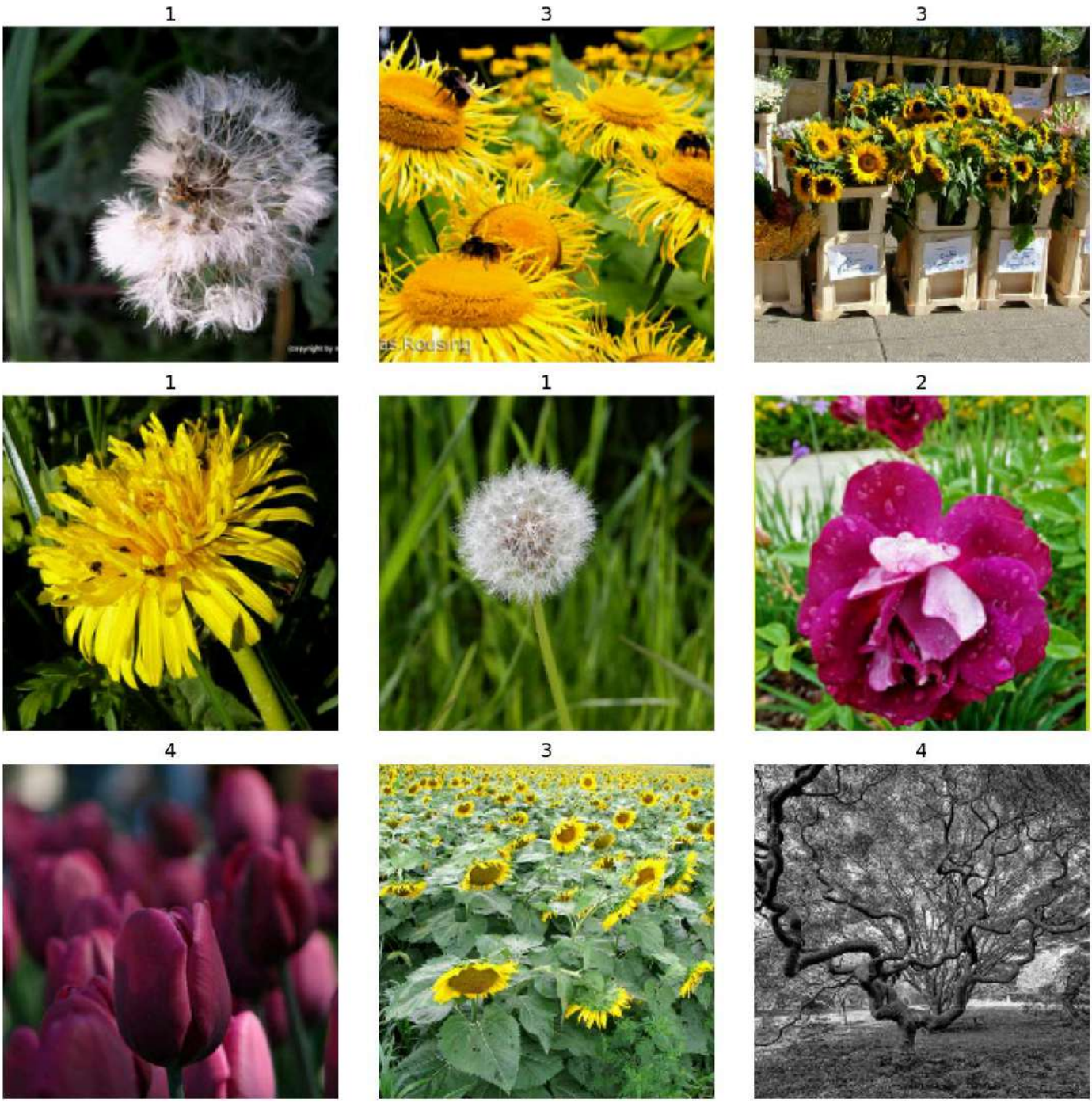
i) Read from the TFRecord Dataset, using `load_dataset` and `display_9_images_from_dataset` to test. \

```
### CODING TASK ###
GCS_OUTPUT = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/tfrecords-jpeg-192x192-2/'  # 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")
datasetDecoded = load_dataset(filenames)
display_9_images_from_dataset(datasetDecoded)
```



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
# import required libraries
```

```
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 pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row

# define variables
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg' # glob  pattern for input files
PROJECT = 'crypto-avenue-347713' # project id
BUCKET = 'gs://{}-storage'.format(PROJECT) # bucket storage
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output file names
PARTITIONS = 16 # no of partitions we will use later

# i) Copy over the mapping functions (see section 1.1) and adapt the resizing and recompression function to Spark (only one argument). (2%)
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 recompress_image(image, label):
    # this reduces the amount of data, but takes some time
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label

# ii) Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list of image filenames. (2%)
# retrieve a list of files that match the specified pattern.
filenames = tf.io.gfile.glob(GCS_PATTERN)
# spark context for rdds
spark_con = pyspark.SparkContext.getOrCreate()
# create RDD for files
rdd1_filenames = spark_con.parallelize(filenames)
# 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%
# sampling the rdd
# rdd1_sample = rdd1_filenames.sample(False, 0.02)
# rdd for decode jpeg and label
rdd2_decode_jpeg_and_label=rdd1_filenames.map(decode_jpeg_and_label)
# rdd for resize and crop image
rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(lambda z: resize_and_crop_image(z[0],z[1]))
# rdd for recompress image
rdd4_recompress_image = rdd3_resize_and_crop_image.map(lambda z:recompress_image(z[0],z[1]))

# iv) Then use the functions from above to write the TFRecord files, using an RDD as the vehicle for parallelisation but not for storing the
# 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): #, height, width):
    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))

print("Writing TFRecords")
def write_tfrecords(partition_index,partition):
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index)
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # re-compressed image: already a byte string
                            label.numpy()
                            )
      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.mapPar
# return a new RDD by applying a function to each partition of this RDD, while tracking the index of the original partition.
rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS)
rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords)
```

```
Overwriting spark_write_tfrec.py
```

```
%%writefile spark_write_tfrec.py
```

```
Overwriting spark_write_tfrec.py
```

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

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

link text

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%)

```
# Create a single-node Dataproc cluster with specific resources and TensorFlow installed
!gcloud dataproc clusters create $CLUSTER \
  --image-version 1.5-ubuntu18 --single-node \
  --master-machine-type n1-standard-8 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
  --metadata PIP_PACKAGES="tensorflow numpy" \
  --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/e50a2e4f-95f2-3da2-94cc-2fbcf3ac5632].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-dataproc-initialization-actions
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

Run the script in the cloud and test the output.

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

```
clusterName: bigdata06-cluster
clusterUuid: a9c40e52-53d6-46fb-a3f3-a0bf7105bbdd
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-02T11:47:11.517250Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: bigdata06-cluster-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
```

Run the script in the cloud and test the output.

```
# submitting spark job for single machine cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER spark_write_tfrec.py
%time
```

```
Job [9e2e2456f03542bd98e381cb71cb0ede] submitted.
Waiting for job output...
2025-05-02 12:24:34.536547: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-05-02 12:24:34.685818: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
2025-05-02 12:24:34.685859: I tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
2025-05-02 12:24:35.473580: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
2025-05-02 12:24:35.473691: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic libra
2025-05-02 12:24:35.473703: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libra
Tensorflow version 2.11.0
25/05/02 12:24:38 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/02 12:24:38 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/02 12:24:38 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/02 12:24:38 INFO org.spark_project.jetty.util.log: Logging initialized @5929ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/02 12:24:38 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/02 12:24:38 INFO org.spark_project.jetty.server.Server: Started @6050ms
25/05/02 12:24:38 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@37c5f2b5{HTTP/1.1, (http/1.1)}{0.0.0
25/05/02 12:24:39 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.0.9:8032
25/05/02 12:24:39 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154
25/05/02 12:24:39 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/02 12:24:39 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/02 12:24:39 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type
25/05/02 12:24:39 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COU
25/05/02 12:24:41 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746186519966_0002
Writing TFRecords
25/05/02 12:24:48 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@37c5f2b5{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [9e2e2456f03542bd98e381cb71cb0ede] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/a9c40e52-53d6-46fb-a3f3
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/a9c40e52-53d6-46fb-a3
jobUuid: 456360fc-eb89-3b5d-9f16-39e2688d893d
placement:
  clusterName: bigdata06-cluster
  clusterUuid: a9c40e52-53d6-46fb-a3f3-a0bf7105bbdd
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/a9c40e52-53d6-46fb-a3f3-a
reference:
  jobId: 9e2e2456f03542bd98e381cb71cb0ede
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-02T12:24:51.493565Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-02T12:24:31.501354Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-02T12:24:31.521398Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-02T12:24:31.694981Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746186519966_0002/
CPU times: user 6 µs, sys: 1 µs, total: 7 µs
Wall time: 13.8 µs
```

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

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

See here 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%)

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-cluster' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/001703ae-1671-32d8-8190-ecf290e3e199].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster].
```

```
### CODING TASK ###
#REGION='europe-west2'

!gcloud dataproc clusters create $CLUSTER \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --worker-machine-type n1-standard-1 \
    --num-workers 7 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --worker-boot-disk-type pd-standard --worker-boot-disk-size $(( 2000 / 7 )) \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
    --max-idle 3600s
```

```
ERROR: (gcloud.dataproc.clusters.create) INVALID_ARGUMENT: Insufficient 'IN_USE_ADDRESSES' quota. Requested 8.0, available 4.0. Your res
```

```
### CODING TASK ###
# cluster with a single machine using the maximal SSD size (100) and 1 master with 1 vCPU + 7 workers with 1 vCPU
#REGION='europe-west2'

#!gcloud dataproc clusters create $CLUSTER \
    #--image-version 1.5-ubuntu18 \
    #--master-machine-type n1-standard-1 \
    #-worker-machine-type n1-standard-1 \
    #--num-workers 7 \
    #--master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    #--initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    #--metadata PIP_PACKAGES="tensorflow-cpu numpy" \
    #--max-idle 3600s
```

```
ERROR: (gcloud.dataproc.clusters.create) INVALID_ARGUMENT: Multiple validation errors:
  - Insufficient 'CPUS' quota. Requested 8.0, available 4.0. Your resource request exceeds your available quota. See https://cloud.google
  - Insufficient 'DISKS_TOTAL_GB' quota. Requested 1995.0, available 1193.0. Your resource request exceeds your available quota. See http
  - Insufficient 'INSTANCES' quota. Requested 8.0, available 4.0. Your resource request exceeds your available quota. See https://cloud.g
  - Insufficient 'IN_USE_ADDRESSES' quota. Requested 8.0, available 0.0. Your resource request exceeds your available quota. See https://
  - This request exceeds CPU quota. Some things to try: request fewer workers (a minimum of 2 is required), use smaller master and/or wor
```

```
### CODING TASK ###
# Note - Error of insufficient CPUs quota occured when I tried to create cluster with 1 master and 7 worker nodes.
# cluster with a single machine using the maximal SSD size (100) and 1 master with 1 vCPU + 3 workers with 1 vCPU
#Create a Dataproc cluster with 1 master and 3 workers, each using 1 vCPU (quota-limited setup).
# Installs TensorFlow and NumPy on all nodes via initialization actions.
# Uses SSD for the master and divides available standard disk space across workers.

!gcloud dataproc clusters create $CLUSTER \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --worker-machine-type n1-standard-1 \
    --num-workers 3 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --worker-boot-disk-type pd-standard --worker-boot-disk-size $(( 2000 / 7 )) \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
    --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/b4d42de2-f280-313c-9a8b-4fc811b457fe].

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
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performanc
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER spark_write_tfrec.py
%time
```

```
Job [fdf8e8c405154a4a8a2230eb09bb5bb3] submitted.
Waiting for job output...
2025-05-02 12:32:52.051535: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Ne
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Tensorflow version 2.11.0
25/05/02 12:32:59 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/02 12:32:59 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/02 12:32:59 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/02 12:32:59 INFO org.spark_project.jetty.util.log: Logging initialized @11836ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/02 12:32:59 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/02 12:33:00 INFO org.spark_project.jetty.server.Server: Started @12089ms
25/05/02 12:33:00 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@45915c69{HTTP/1.1, (http/1.1)}{0.0.0.0:
25/05/02 12:33:02 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.15.233:8032
25/05/02 12:33:02 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154.15
25/05/02 12:33:02 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/02 12:33:02 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/02 12:33:02 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = C
25/05/02 12:33:02 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTA
25/05/02 12:33:07 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746189027932_0001
Writing TFRecords
25/05/02 12:33:29 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@45915c69{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [fdf8e8c405154a4a8a2230eb09bb5bb3] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/83ad6acc-a6ac-4f9c-b1b2-03
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/83ad6acc-a6ac-4f9c-b1b2-
jobUuid: e90ad49c-8502-3150-8939-2def24854e3d
placement:
  clusterName: bigdata06-cluster
  clusterUuid: 83ad6acc-a6ac-4f9c-b1b2-03509d72b1b7
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/83ad6acc-a6ac-4f9c-b1b2-0356
reference:
  jobId: fdf8e8c405154a4a8a2230eb09bb5bb3
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-02T12:33:34.492540Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-02T12:32:45.473610Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-02T12:32:45.506086Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-02T12:32:45.862261Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746189027932_0001/
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 8.82 µs
```

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

```
clusterName: bigdata06-cluster
clusterUuid: 83ad6acc-a6ac-4f9c-b1b2-03509d72b1b7
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow-cpu numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
```

```
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-02T12:33:34.492540Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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: '40'
      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: bigdata06-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
```

:### 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.

```
%%writefile spark_write_tfrec.py

# Import necessary libraries for Spark, TensorFlow, file handling, and timing
import pyspark
import tensorflow as tf
import numpy as np
import time
import pickle
import os

# Define global project parameters
PROJECT = 'bigdata06'
CLUSTER = f'{PROJECT}-cluster'
BUCKET = f'gs://{PROJECT}-storage'
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg'  # Input image pattern
GCS_OUTPUT = f'{BUCKET}/tfrecords-jpeg-192x192-2/flowers'                  # Output path for TFRecords
PARTITIONS = 16  # Number of RDD partitions for parallel processing
TARGET_SIZE = [192, 192]  # Resize target for images

# Function to decode JPEG image and extract label from path
def decode_jpeg_and_label(filepath):
```

```
        bits = tf.io.read_file(filepath)
        image = tf.image.decode_jpeg(bits)
        label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/').values[-2]
        return image, label

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

    # Function to recompress image as JPEG
    def recompress_image(image, label):
        image = tf.cast(image, tf.uint8)
        image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
        return image, label

    # Function to convert to TFRecord format
    def to_tfrecord(out_file, image, label):
        feature = {
            'image': tf.train.Feature(bytes_list=tf.train.BytesList(value=[image])),
            'class': tf.train.Feature(int64_list=tf.train.Int64List(value=[label]))
        }
        return tf.train.Example(features=tf.train.Features(feature=feature))

    # List all input file paths matching the GCS pattern
    filenames = tf.io.gfile.glob(GCS_PATTERN)

    # Create SparkContext
    spark_con = pyspark.SparkContext.getOrCreate()

    ### TASK 1d ###
    # Improve parallelism by specifying number of partitions explicitly when parallelizing file list
    rdd1_filenames = spark_con.parallelize(filenames,16)

    # Step-by-step image processing RDD pipeline
    rdd2_decode_jpeg_and_label = rdd1_filenames.map(decode_jpeg_and_label)
    rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(lambda z: resize_and_crop_image(z[0], z[1]))
    rdd4_recompress_image = rdd3_resize_and_crop_image.map(lambda z: recompress_image(z[0], z[1]))

    print("Writing TFRecords")

    # Function to write TFRecord file for each RDD partition
    def write_tfrecords(partition_index, partition):
        filename = f"{GCS_OUTPUT}{partition_index}.tfrec"
        with tf.io.TFRecordWriter(filename) as out_file:
            for image, label in partition:
                example = to_tfrecord(out_file, image.numpy(), label.numpy())
                out_file.write(example.SerializeToString())
        return [filename]

    # Repartition the RDD to match target parallelism
    rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS)

    # Apply TFRecord writing in parallel by partition
    rdd_output = rdd5_partitions.mapPartitionsWithIndex(write_tfrecords)
```

```
Overwriting spark_write_tfrec.py
```

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-cluster' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/6a6db378-b8a9-3825-be74-eb6f8e97c1c5].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster].
```

```
### CODING TASK ###
# cluster with a single machine using the maximal SSD size (100) and 1 master with 1 vCPU + 3 workers with 1 vCPU
#REGION='europe-west2'

!gcloud dataproc clusters create $CLUSTER \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --worker-machine-type n1-standard-1 \
    --num-workers 3 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --worker-boot-disk-type pd-standard --worker-boot-disk-size $(( 2000 / 3 )) \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
    --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/062fd717-c3db-3b2e-93ee-6fc71a8af976].

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
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performanc
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

```
# submitting spark job
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER spark_write_tfrec.py
%time
```

```
Job [9b2f7d8604c749a3a4c8cbe6d641786d] submitted.
Waiting for job output...
2025-05-02 12:43:14.295962: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Ne
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
25/05/02 12:43:20 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/02 12:43:20 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/02 12:43:21 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/02 12:43:21 INFO org.spark_project.jetty.util.log: Logging initialized @11557ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/02 12:43:21 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/02 12:43:21 INFO org.spark_project.jetty.server.Server: Started @11807ms
25/05/02 12:43:21 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@51f17779{HTTP/1.1, (http/1.1)}{0.0.0.0:
25/05/02 12:43:24 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.0.13:8032
25/05/02 12:43:24 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154.0.
25/05/02 12:43:24 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/02 12:43:24 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/02 12:43:24 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = C
25/05/02 12:43:24 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTA
25/05/02 12:43:29 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746189658617_0001
Writing TFRecords
25/05/02 12:43:48 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@51f17779{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [9b2f7d8604c749a3a4c8cbe6d641786d] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/e1f01d26-080d-403d-8bdc-57
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/e1f01d26-080d-403d-8bdc-
jobUuid: 43abc34e-bf62-34ca-9e5e-1d607a391785
placement:
  clusterName: bigdata06-cluster
  clusterUuid: e1f01d26-080d-403d-8bdc-57869b399e71
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/e1f01d26-080d-403d-8bdc-578
reference:
  jobId: 9b2f7d8604c749a3a4c8cbe6d641786d
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-02T12:43:53.948673Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-02T12:43:06.555768Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-02T12:43:06.576538Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-02T12:43:06.949783Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746189658617_0001/
CPU times: user 4 µs, sys: 1e+03 ns, total: 5 µs
Wall time: 16 µs
```

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

```
clusterName: bigdata06-cluster
clusterUuid: e1f01d26-080d-403d-8bdc-57869b399e71
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow-cpu numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-02T12:43:53.948673Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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: '40'
      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: bigdata06-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
```

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-cluster' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/b196a0de-7aaa-3a88-9172-6af35c05d406].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster].
```

```
### EXPERIMENTING WITH CLUSTER CONFIGURATIONS ###

# 4 machines with double the resources each (2 vCPUs, memory, disk)

!gcloud dataproc clusters create $CLUSTER \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-2 \
    --worker-machine-type n1-standard-2 \
```

```
    --num-workers 3 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --worker-boot-disk-type pd-standard --worker-boot-disk-size $(( 2000 / 3 )) \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
    --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/a2667bde-5cdc-3c4d-94b8-36d1261f682d].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-dataproc-initialization-actions
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performanc
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

```
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: bigdata06-cluster
clusterUuid: 47c15645-7381-4a0a-9b03-ef1169ae5301
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow-cpu numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-01T22:51:01.467110Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-2
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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: '40'
      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: bigdata06-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
```

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER spark_write_tfrec.py
%time
```

```
Job [5e643f40633e456c8c580e1b2590af2c] submitted.
Waiting for job output...
2025-05-01 22:54:59.048224: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
25/05/01 22:55:03 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/01 22:55:03 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/01 22:55:03 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/01 22:55:04 INFO org.spark_project.jetty.util.log: Logging initialized @7857ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/01 22:55:04 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/01 22:55:04 INFO org.spark_project.jetty.server.Server: Started @7990ms
25/05/01 22:55:04 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@7f17fdf6{HTTP/1.1, (http/1.1)}{0.0.0
25/05/01 22:55:06 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.15.227:8032
25/05/01 22:55:06 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154
25/05/01 22:55:06 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/01 22:55:06 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/01 22:55:06 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type
25/05/01 22:55:06 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COU
25/05/01 22:55:07 WARN org.apache.hadoop.hdfs.DataStreamer: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1257)
        at java.lang.Thread.join(Thread.java:1331)
        at org.apache.hadoop.hdfs.DataStreamer.closeResponder(DataStreamer.java:980)
        at org.apache.hadoop.hdfs.DataStreamer.endBlock(DataStreamer.java:630)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:807)
25/05/01 22:55:09 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746139993826_0001
Writing TFRecords
25/05/01 22:55:23 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@7f17fdf6{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [5e643f40633e456c8c580e1b2590af2c] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/47c15645-7381-4a0a-9b03
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/47c15645-7381-4a0a-9b
jobUuid: 22ec7b48-7c68-3107-8f76-0796a02fb9fe
placement:
  clusterName: bigdata06-cluster
  clusterUuid: 47c15645-7381-4a0a-9b03-ef1169ae5301
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/47c15645-7381-4a0a-9b03-e
reference:
  jobId: 5e643f40633e456c8c580e1b2590af2c
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-01T22:55:24.745743Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-01T22:54:54.156097Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-01T22:54:54.176100Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-01T22:54:54.422825Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746139993826_0001/
```

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-cluster' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/a1d1d0aa-1ca2-3eae-a2e7-26737ec01c0e].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster].
```

```
# Cluster with 1 machine with eightfold resources

!gcloud dataproc clusters create $CLUSTER \
  --image-version 1.5-ubuntu18 --single-node \
  --master-machine-type n1-standard-8 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
  --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
  --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/1a8939dc-9be2-3d12-be68-43359648ea37].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-dataproc-initialization-actions
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
```

```
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

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

```
clusterName: bigdata06-cluster
clusterUuid: 2e8be18a-7ef2-4be2-a897-c2844f92ba6e
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow-cpu numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-01T22:56:51.408943Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: bigdata06-cluster-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
```

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER spark_write_tfrec.py
%time
```

```
Job [b7ff75506a264277a8de99a975997f32] submitted.
Waiting for job output...
2025-05-01 23:00:13.190888: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Ne
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
25/05/01 23:00:18 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/01 23:00:18 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/01 23:00:18 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/01 23:00:18 INFO org.spark_project.jetty.util.log: Logging initialized @7699ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/01 23:00:18 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/01 23:00:18 INFO org.spark_project.jetty.server.Server: Started @7825ms
25/05/01 23:00:18 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@25ef055b{HTTP/1.1, (http/1.1)}{0.0.0.0:
25/05/01 23:00:19 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.15.230:8032
```

```
25/05/01 23:00:20 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154.15
25/05/01 23:00:20 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/01 23:00:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/01 23:00:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = C
25/05/01 23:00:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTA
25/05/01 23:00:23 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746140323849_0001
Writing TFRecords
25/05/01 23:00:33 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@25ef055b{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [b7ff75506a264277a8de99a975997f32] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/2e8be18a-7ef2-4be2-a897-c2
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/2e8be18a-7ef2-4be2-a897-
jobUuid: 33dc3aee-d5a6-3c34-af81-216ee5b002aa
placement:
  clusterName: bigdata06-cluster
  clusterUuid: 2e8be18a-7ef2-4be2-a897-c2844f92ba6e
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/2e8be18a-7ef2-4be2-a897-c284
reference:
  jobId: b7ff75506a264277a8de99a975997f32
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-01T23:00:37.250949Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-01T23:00:08.704030Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-01T23:00:08.728733Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-01T23:00:09.009611Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746140323849_0001/
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 9.06 µs
```

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-cluster' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/78480b48-c62b-3522-8ab0-e8dccda75391].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster].
```

Section 2: Speed tests

> Add blockquote

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

2.1 Speed test implementation

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

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

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

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

```
# 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'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'roses'", "b'roses'"]) <_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: 2, ds: 6, rep: 1
Image batch shape (2,), ["b'tulips'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'tulips'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'dandelion'", "b'sunflowers'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'sunflowers'", "b'roses'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='utf-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4,), ["b'daisy'", "b'sunflowers'", "b'roses'", "b'daisy'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding=
Image batch shape (4,), ["b'daisy'", "b'tulips'", "b'dandelion'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encod
Image batch shape (4,), ["b'dandelion'", "b'dandelion'", "b'dandelion'", "b'roses'"]) <_io.TextIOWrapper name='/dev/null' mode='w' en
bs: 4, ds: 6, rep: 1
Image batch shape (4,), ["b'tulips'", "b'roses'", "b'roses'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='ut
Image batch shape (4,), ["b'roses'", "b'dandelion'", "b'roses'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encodi
Image batch shape (4,), ["b'roses'", "b'tulips'", "b'sunflowers'", "b'daisy'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
Image batch shape (4,), ["b'sunflowers'", "b'sunflowers'", "b'tulips'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w'
Image batch shape (4,), ["b'dandelion'", "b'roses'", "b'roses'", "b'daisy'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='
Image batch shape (4,), ["b'roses'", "b'tulips'", "b'sunflowers'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encodin
total time: 3.4398131370544434
[[[ 9.14862383]
  [14.90850011]]

 [[16.72974257]
  [19.30988804]]]
[[[[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'>
```

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

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

2a) Create the script (14%)

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

More specifically:

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

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

```
%%writefile spark_job.py
import os
import tensorflow as tf
import numpy as np
import time
import pickle
from pyspark import SparkContext, SparkConf
from datetime import datetime

# Define globals
PROJECT = 'bigdata06'  ### My GCP project ID
BUCKET = 'gs://bigdata06-storage'  # My GCS bucket
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg'  # Input JPEG files path (modify this if needed)
GCS_OUTPUT = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/tfrecords-jpeg-192x192-2/'  # TFRecord output path
TFRECORD_DIR = os.path.join(BUCKET, "tfrecords_spark")  # (Unused here)
OUTPUT_PICKLE = os.path.join(BUCKET, "{}_spark_speed_test_results.pkl".format(datetime.now().strftime("%Y%m%d-%H%M%S")))  # Output pickle pa
TARGET_SIZE = [192, 192]
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
PARTITIONS = 16
RANDOM_SEED = 42
NUM_IMAGES = 3670
```

```
# === DATA PREPROCESSING UTILS ===
# These functions handle decoding, resizing, recompression, and feature encoding.

def decode_jpeg_and_label(filepath):
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/').values[-2]
    return image, label

def resize_and_crop_image(image, label):
    w, h = tf.shape(image)[0], tf.shape(image)[1]
    tw, th = TARGET_SIZE[1], TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(
        resize_crit < 1,
        lambda: tf.image.resize(image, [w * tw / w, h * tw / w]),
        lambda: tf.image.resize(image, [w * th / h, h * th / h])
    )
    nw, nh = tf.shape(image)[0], 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):
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label

def _bytes_feature(value):
    if isinstance(value, type(tf.constant(0))):
        value = value.numpy()
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _int64_feature(value):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),
        "class": tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    return image, example['class']

def load_dataset(filenames):
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False
    dataset = tf.data.TFRecordDataset(filenames).with_options(option_no_order)
    return dataset.map(read_tfrecord)


# === PERFORMANCE TEST FUNCTION ===
# Used for benchmarking JPEG vs TFRecord loading

def time_configs(dataset_type, batch_size, batch_number, repetition, gcs_pattern, gcs_output):
    if dataset_type == "jpeg":
        dset_files = tf.data.Dataset.list_files(gcs_pattern)
        dset_decoded = dset_files.map(decode_jpeg_and_label)
        dset_resized = dset_decoded.map(resize_and_crop_image)
        dataset = dset_resized.map(recompress_image)
    elif dataset_type == "tfrec":
        filenames = tf.io.gfile.glob(gcs_output + "*.tfrec")
        dataset = load_dataset(filenames)
    else:
        raise ValueError("Invalid dataset_type. Choose 'jpeg' or 'tfrec'.")

    results = []
    with open("/dev/null", mode='w') as null_file:
        t0 = time.time()
        batched_dataset = dataset.batch(batch_size)
        timing_set = batched_dataset.take(batch_number)
        for image, label in timing_set:
            print("Image batch shape {}".format(image.numpy().shape), file=null_file)
        td = time.time() - t0
```

```
            images_per_second = (batch_size * batch_number) / td
            results.append((dataset_type, batch_size, batch_number, repetition, images_per_second))
    return results


# === MAIN FUNCTION ===
def main():
    # Initialize Spark
    conf = SparkConf().setAppName("TFRecordSpeedTest")
    sc = SparkContext.getOrCreate(conf=conf)

    ### (i) Create RDD with combinations of parameters for benchmarking ###
    batch_sizes = [2, 4, 8]
    batch_numbers = [3, 6, 12]
    repetitions = [1, 2, 3]

    params_tfrec = [("tfrec", b, n, r) for b in batch_sizes for n in batch_numbers for r in repetitions]
    params_jpeg = [("jpeg", b, n, r) for b in batch_sizes for n in batch_numbers for r in repetitions]
    params = params_tfrec + params_jpeg

    # Broadcast the input/output paths
    gcs_pattern_bc = sc.broadcast(GCS_PATTERN)
    gcs_output_bc = sc.broadcast(GCS_OUTPUT)

    # Parallelize parameter combinations for distributed benchmarking
    params_rdd = sc.parallelize(params, PARTITIONS)

    ### (ii) Run timing tests for each configuration in parallel using Spark ###
    results_rdd = params_rdd.flatMap(lambda x: time_configs(
        x[0], x[1], x[2], x[3], gcs_pattern_bc.value, gcs_output_bc.value))

    ### (iii) Extract results into key-value format: ((data_type, batch_size, num_batches, repetition), images/sec) ###
    def extract_results(results):
        data_type, bs, ds, rep, i_p_s = results
        return [((data_type, bs, ds, rep), i_p_s)]
    results_rdd = results_rdd.flatMap(lambda x: extract_results(x))

    ### (iv) For each individual parameter, create RDD: ((data_type, param_name, value), images/sec) ###
    def get_param_results(x):
        (data_type, bs, ds, rep), images_per_second = x
        return [
            ((data_type, 'batch_size', bs), images_per_second),
            ((data_type, 'batch_number', ds), images_per_second),
            ((data_type, 'repetition', rep), images_per_second)
        ]
    all_params_rdd = results_rdd.flatMap(get_param_results)

    ### (v) Compute average speed for each parameter value ###
    def calculate_average(x):
        (data_type, param_name, param_value), values = x
        return ((data_type, param_name, param_value), sum(values) / len(values))
    average_speeds_rdd = all_params_rdd.groupByKey().mapValues(list).map(calculate_average)

    ### (vi) Save full results and average results as pickle file to GCS ###
    results = {
        "all_results": results_rdd.collect(),
        "average_speeds": average_speeds_rdd.collect()
    }
    with tf.io.gfile.GFile(OUTPUT_PICKLE, 'wb') as f:
        pickle.dump(results, f)

    print(f"Results saved to: {OUTPUT_PICKLE}")


# Entry point
if __name__ == "__main__":
    main()
```

```
Overwriting spark_job.py
```

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

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

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

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

```
%run spark_job.py
```

```
Results saved to: gs://bigdata06-storage/20250502-125307_spark_speed_test_results.pkl
<Figure size 640x480 with 0 Axes>
```

```
import datetime

# Generate timestamp string
timestamp = datetime.datetime.now().strftime("%y%m%d-%H%M")

# Create a unique output filename using the timestamp
output_filename = f"results_{timestamp}.txt"

# Now use `output_filename` wherever you're saving your results
with open(output_filename, 'w') as f:
    f.write("Your results go here...\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.

```
!gcloud dataproc clusters create $CLUSTER \
  --image-version 1.5-ubuntu18 --single-node \
  --master-machine-type n1-standard-8 \
  --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
  --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
  --metadata PIP_PACKAGES="tensorflow-cpu numpy" \
  --max-idle 3600s
```

```
Waiting on operation [projects/bigdata06/regions/europe-west2/operations/520fe810-8d36-3175-8624-8b221a6c9674].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-dataproc-initialization-actions
WARNING: Failed to validate permissions required for default service account: '96475368285-compute@developer.gserviceaccount.com'. Clust
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security ri
WARNING: The specified custom staging bucket 'dataproc-staging-europe-west2-96475368285-wwktuyat' is not using uniform bucket level acce
Created [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-cluster] Cluster placed in zone [
```

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_job.py
%time
```

```
Job [89a3329b4ecd4dabad770905d908f5f4] submitted.
Waiting for job output...
2025-05-02 12:58:15.046167: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Ne
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
25/05/02 12:58:18 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/02 12:58:18 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/02 12:58:18 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/02 12:58:18 INFO org.spark_project.jetty.util.log: Logging initialized @5525ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/02 12:58:18 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/02 12:58:18 INFO org.spark_project.jetty.server.Server: Started @5641ms
25/05/02 12:58:18 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@37c5f2b5{HTTP/1.1, (http/1.1)}{0.0.0.0:
25/05/02 12:58:20 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.0.10:8032
25/05/02 12:58:20 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154.0.
25/05/02 12:58:20 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/02 12:58:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/02 12:58:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = C
25/05/02 12:58:20 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTA
25/05/02 12:58:23 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746190605314_0001
Results saved to: gs://bigdata06-storage/20250502-125817_spark_speed_test_results.pkl
25/05/02 13:00:14 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@37c5f2b5{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [89a3329b4ecd4dabad770905d908f5f4] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/08ed70c1-937e-4c75-9dd4-f5
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/08ed70c1-937e-4c75-9dd4-
jobUuid: fce93929-b14b-3c5b-a6b3-8c4d16a48404
placement:
```

```
  clusterName: bigdata06-cluster
  clusterUuid: 08ed70c1-937e-4c75-9dd4-f51c19e69c24
pysparkJob:
  mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/08ed70c1-937e-4c75-9dd4-f51c
reference:
  jobId: 89a3329b4ecd4dabad770905d908f5f4
  projectId: bigdata06
status:
  state: DONE
  stateStartTime: '2025-05-02T13:00:15.500212Z'
statusHistory:
- state: PENDING
  stateStartTime: '2025-05-02T12:58:11.246876Z'
- state: SETUP_DONE
  stateStartTime: '2025-05-02T12:58:11.266408Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2025-05-02T12:58:11.503503Z'
yarnApplications:
- name: TFRecordSpeedTest
  progress: 1.0
  state: FINISHED
  trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746190605314_0001/
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 9.3 µs
```

```
!gcloud dataproc clusters describe $CLUSTER
```

```
clusterName: bigdata06-cluster
clusterUuid: 3386c045-27f2-42e0-9db8-d9cc3e64f27b
config:
  configBucket: dataproc-staging-europe-west2-96475368285-wwktuyat
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow-cpu numpy
    networkUri: https://www.googleapis.com/compute/v1/projects/bigdata06/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
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-europe-west2/python/pip-install.sh
    executionTimeout: 600s
  lifecycleConfig:
    idleDeleteTtl: 3600s
    idleStartTime: '2025-05-01T23:11:26.739312Z'
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20230909-165100-rc01
    instanceNames:
    - bigdata06-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/bigdata06/zones/europe-west2-c/machineTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-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
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
```

```
            hdfs:dfs.namenode.http-address: 0.0.0.0:9870
            hdfs:dfs.namenode.https-address: 0.0.0.0:9871
            hdfs:dfs.namenode.lifeline.rpc-address: bigdata06-cluster-m:8050
            hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
```

```
!gcloud dataproc clusters delete $CLUSTER
```

```
The cluster 'bigdata06-single' and all attached disks will be deleted.

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

Waiting on operation [projects/bigdata06/regions/europe-west2/operations/7a5d0249-569a-3633-a26a-2301116bfc38].
Deleted [https://dataproc.googleapis.com/v1/projects/bigdata06/regions/europe-west2/clusters/bigdata06-single].
```

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.py
import os
import tensorflow as tf
import numpy as np
import time
import pickle
from pyspark import SparkContext, SparkConf
from datetime import datetime

# Define globals
PROJECT = 'bigdata06'  ### My GCP project ID
BUCKET = 'gs://bigdata06-storage'  # My GCS bucket
GCS_PATTERN = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/*/*.jpg'  # Input JPEG files path (modify this if needed)
GCS_OUTPUT = 'gs://cloud-samples-data/ai-platform/flowers_tfrec/tfrecords-jpeg-192x192-2/'  # TFRecord output path
TFRECORD_DIR = os.path.join(BUCKET, "tfrecords_spark")  # (Unused here)
OUTPUT_PICKLE = os.path.join(BUCKET, "{}_spark_speed_test_results.pkl".format(datetime.now().strftime("%Y%m%d-%H%M%S")))  # Output pickle pa
TARGET_SIZE = [192, 192]
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
PARTITIONS = 16
RANDOM_SEED = 42
NUM_IMAGES = 3670


# === DATA PREPROCESSING UTILS ===
# These functions handle decoding, resizing, recompression, and feature encoding.

def decode_jpeg_and_label(filepath):
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/').values[-2]
    return image, label

def resize_and_crop_image(image, label):
    w, h = tf.shape(image)[0], tf.shape(image)[1]
    tw, th = TARGET_SIZE[1], TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(
        resize_crit < 1,
        lambda: tf.image.resize(image, [w * tw / w, h * tw / w]),
        lambda: tf.image.resize(image, [w * th / h, h * th / h])
    )
    nw, nh = tf.shape(image)[0], 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):
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label
```

```
    def _bytes_feature(value):
        if isinstance(value, type(tf.constant(0))):
            value = value.numpy()
        return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

    def _int64_feature(value):
        return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

    def read_tfrecord(example):
        features = {
            "image": tf.io.FixedLenFeature([], tf.string),
            "class": tf.io.FixedLenFeature([], tf.int64)
        }
        example = tf.io.parse_single_example(example, features)
        image = tf.image.decode_jpeg(example['image'], channels=3)
        image = tf.reshape(image, [*TARGET_SIZE, 3])
        return image, example['class']

    def load_dataset(filenames):
        option_no_order = tf.data.Options()
        option_no_order.experimental_deterministic = False
        dataset = tf.data.TFRecordDataset(filenames).with_options(option_no_order)
        return dataset.map(read_tfrecord)


    # === PERFORMANCE TEST FUNCTION ===
    # Used for benchmarking JPEG vs TFRecord loading

    def time_configs(dataset_type, batch_size, batch_number, repetition, gcs_pattern, gcs_output):
        if dataset_type == "jpeg":
            dset_files = tf.data.Dataset.list_files(gcs_pattern)
            dset_decoded = dset_files.map(decode_jpeg_and_label)
            dset_resized = dset_decoded.map(resize_and_crop_image)
            dataset = dset_resized.map(recompress_image)
        elif dataset_type == "tfrec":
            filenames = tf.io.gfile.glob(gcs_output + "*.tfrec")
            dataset = load_dataset(filenames)
        else:
            raise ValueError("Invalid dataset_type. Choose 'jpeg' or 'tfrec'.")

        results = []
        with open("/dev/null", mode='w') as null_file:
            t0 = time.time()
            batched_dataset = dataset.batch(batch_size)
            timing_set = batched_dataset.take(batch_number)
            for image, label in timing_set:
                print("Image batch shape {}".format(image.numpy().shape), file=null_file)
            td = time.time() - t0
            images_per_second = (batch_size * batch_number) / td
            results.append((dataset_type, batch_size, batch_number, repetition, images_per_second))
        return results


    # === MAIN FUNCTION ===
    def main():
        # Initialize Spark
        conf = SparkConf().setAppName("TFRecordSpeedTest")
        sc = SparkContext.getOrCreate(conf=conf)

        ### (i) Create RDD with combinations of parameters for benchmarking ###
        batch_sizes = [2, 4, 8]
        batch_numbers = [3, 6, 12]
        repetitions = [1, 2, 3]

        params_tfrec = [("tfrec", b, n, r) for b in batch_sizes for n in batch_numbers for r in repetitions]
        params_jpeg = [("jpeg", b, n, r) for b in batch_sizes for n in batch_numbers for r in repetitions]
        params = params_tfrec + params_jpeg

        # Broadcast the input/output paths
        gcs_pattern_bc = sc.broadcast(GCS_PATTERN)
        gcs_output_bc = sc.broadcast(GCS_OUTPUT)

        # Parallelize parameter combinations for distributed benchmarking
        params_rdd = sc.parallelize(params, PARTITIONS)

        ### (ii) Run timing tests for each configuration in parallel using Spark ###
        results_rdd = params_rdd.flatMap(lambda x: time_configs(
```

```
            x[0], x[1], x[2], x[3], gcs_pattern_bc.value, gcs_output_bc.value))

        ### TASK 2c ###
        # Cache the results_rdd. This is important because this RDD is used multiple times below for:
        # - extracting structured results
        # - parameter-wise performance analysis
        # Without caching, Spark would recompute time_configs for each action, leading to redundant computation and slow performance.
        results_rdd = results_rdd.cache()


        ### (iii) Extract results into key-value format: ((data_type, batch_size, num_batches, repetition), images/sec) ###
        def extract_results(results):
            data_type, bs, ds, rep, i_p_s = results
            return [((data_type, bs, ds, rep), i_p_s)]
        results_rdd = results_rdd.flatMap(lambda x: extract_results(x))

        ### (iv) For each individual parameter, create RDD: ((data_type, param_name, value), images/sec) ###
        def get_param_results(x):
            (data_type, bs, ds, rep), images_per_second = x
            return [
                ((data_type, 'batch_size', bs), images_per_second),
                ((data_type, 'batch_number', ds), images_per_second),
                ((data_type, 'repetition', rep), images_per_second)
            ]
        all_params_rdd = results_rdd.flatMap(get_param_results)

        ### (v) Compute average speed for each parameter value ###
        def calculate_average(x):
            (data_type, param_name, param_value), values = x
            return ((data_type, param_name, param_value), sum(values) / len(values))
        average_speeds_rdd = all_params_rdd.groupByKey().mapValues(list).map(calculate_average)

        ### (vi) Save full results and average results as pickle file to GCS ###
        results = {
            "all_results": results_rdd.collect(),
            "average_speeds": average_speeds_rdd.collect()
        }
        with tf.io.gfile.GFile(OUTPUT_PICKLE, 'wb') as f:
            pickle.dump(results, f)

        print(f"Results saved to: {OUTPUT_PICKLE}")


    # Entry point
    if __name__ == "__main__":
        main()
```

```
Overwriting spark_job.py
```

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER \spark_job.py
%time
```

```
Job [e1a19dd25b334b9d9d90046fd5ac7447] submitted.
Waiting for job output...
2025-05-01 23:14:14.594043: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Ne
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
25/05/01 23:14:17 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/01 23:14:17 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/01 23:14:17 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/01 23:14:17 INFO org.spark_project.jetty.util.log: Logging initialized @4932ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/01 23:14:17 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/01 23:14:17 INFO org.spark_project.jetty.server.Server: Started @5056ms
25/05/01 23:14:17 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@600eaa78{HTTP/1.1, (http/1.1)}{0.0.0.0:
25/05/01 23:14:18 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at bigdata06-cluster-m/10.154.0.2:8032
25/05/01 23:14:18 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at bigdata06-cluster-m/10.154.0.
25/05/01 23:14:19 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/01 23:14:19 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/01 23:14:19 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = C
25/05/01 23:14:19 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTA
25/05/01 23:14:21 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746140870617_0002
Results saved to: gs://bigdata06-storage/20250501-231416_spark_speed_test_results.pkl
25/05/01 23:15:22 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@600eaa78{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
Job [e1a19dd25b334b9d9d90046fd5ac7447] finished successfully.
done: true
driverControlFilesUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/3386c045-27f2-42e0-9db8-d9
driverOutputResourceUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/3386c045-27f2-42e0-9db8-
jobUuid: 9244300e-459a-3fdf-90fa-39a0af9194b7
placement:
  clusterName: bigdata06-cluster
```

```
    clusterUuid: 3386c045-27f2-42e0-9db8-d9cc3e64f27b
  pysparkJob:
    mainPythonFileUri: gs://dataproc-staging-europe-west2-96475368285-wwktuyat/google-cloud-dataproc-metainfo/3386c045-27f2-42e0-9db8-d9cc
  reference:
    jobId: e1a19dd25b334b9d9d90046fd5ac7447
    projectId: bigdata06
  status:
    state: DONE
    stateStartTime: '2025-05-01T23:15:26.911804Z'
  statusHistory:
  - state: PENDING
    stateStartTime: '2025-05-01T23:14:11.281942Z'
  - state: SETUP_DONE
    stateStartTime: '2025-05-01T23:14:11.299889Z'
  - details: Agent reported job success
    state: RUNNING
    stateStartTime: '2025-05-01T23:14:11.455338Z'
  yarnApplications:
  - name: TFRecordSpeedTest
    progress: 1.0
    state: FINISHED
    trackingUrl: http://bigdata06-cluster-m:8088/proxy/application_1746140870617_0002/
```

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**.

```
from sklearn.linear_model import LinearRegression
from collections import defaultdict
import numpy as np
import matplotlib.pyplot as plt
import pickle
import tensorflow as tf

# ------------------------------
# Load Pickled Spark Benchmark Results
# ------------------------------
output_pickle = 'gs://bigdata06-storage/20250501-230337_spark_speed_test_results.pkl'

with tf.io.gfile.GFile(output_pickle, 'rb') as f:
    results_data = pickle.load(f)

all_results = results_data["all_results"]
average_speeds = results_data["average_speeds"]

# Split JPEG and TFRecord results
jpeg_results = [r for r in all_results if r[0][0] == 'jpeg']
tfrec_results = [r for r in all_results if r[0][0] == 'tfrec']

# Print basic info
print("\n--- All Results (JPEG and TFRecord) ---")
for result in all_results:
    print(result)
```

```
print("\n--- Average Speeds ---")
for param, avg_speed in average_speeds:
    print(f"{param}: {avg_speed:.2f}")


# ------------------------------
# Linear Regression Analysis
# ------------------------------
def perform_linear_regression(data, parameter_name):
    """Perform linear regression on one parameter."""
    param_index = {'batch_size': 1, 'batch_number': 2, 'repetition': 3}[parameter_name]
    x = np.array([d[0][param_index] for d in data]).reshape(-1, 1)
    y = np.array([d[1] for d in data])
    model = LinearRegression()
    model.fit(x, y)
    return model.coef_[0], model.intercept_

def analyze_and_print_regression(results, data_type):
    """Run regression on each parameter and print summary."""
    print(f"\n--- Linear Regression Analysis for {data_type} ---")
    regression_results = {}
    parameters = ['batch_size', 'batch_number', 'repetition']

    for param in parameters:
        coef, intercept = perform_linear_regression(results, param)
        regression_results[param] = {'coef': coef, 'intercept': intercept}
        print(f"{param}: Coefficient = {coef:.2f}, Intercept = {intercept:.2f}")

    # Special case: batch_size x batch_number
    x_product = np.array([d[0][1] * d[0][2] for d in results]).reshape(-1, 1)
    y_product = np.array([d[1] for d in results])
    model = LinearRegression()
    model.fit(x_product, y_product)
    regression_results['batch_size_x_batch_number'] = {
        'coef': model.coef_[0],
        'intercept': model.intercept_
    }
    print(f"batch_size x batch_number: Coefficient = {model.coef_[0]:.2f}, Intercept = {model.intercept_:.2f}")
    return regression_results

# Run regression for both types
jpeg_regression_results = analyze_and_print_regression(jpeg_results, "JPEG")
tfrec_regression_results = analyze_and_print_regression(tfrec_results, "TFRecord")


# ------------------------------
# Plot Results with Clear Layout
# ------------------------------
def plot_results(jpeg_data, tfrec_data, jpeg_avg_speeds, tfrec_avg_speeds, jpeg_reg, tfrec_reg, title):
    """Generate scatter + regression plots for JPEG and TFRecord benchmarks."""
    parameters = ['batch_size', 'batch_number', 'repetition']
    fig, axes = plt.subplots(4, 2, figsize=(15, 18))
    fig.suptitle(title, fontsize=16)

    def plot_param(ax, param_data, param, avg_speeds, regression, label_prefix):
        """Helper for plotting one parameter (scatter, avg, regression line)."""
        param_data.sort()
        x_vals = [p[0] for p in param_data]
        y_vals = [p[1] for p in param_data]
        ax.scatter(x_vals, y_vals, label='Raw data', color='blue')
        avg_x = [p[1] for p in avg_speeds if p[0][0] == param]
        avg_y = [p[1] for p in avg_speeds if p[0][0] == param]
        if avg_x:
            ax.plot(avg_x, avg_y, 'ro-', label='Average')
        coef = regression[param]['coef']
        intercept = regression[param]['intercept']
        x_line = np.array(sorted(set(x_vals)))
        y_line = coef * x_line + intercept
        ax.plot(x_line, y_line, 'g--', label='Regression')
        ax.set_xlabel(param)
        ax.set_ylabel('Images/sec')
        ax.set_title(f"{label_prefix}: {param}")
        ax.legend()

    # JPEG plots
    jpeg_param_data = defaultdict(list)
    for (dtype, bs, bn, rep), speed in jpeg_data:
```

```
            jpeg_param_data['batch_size'].append((bs, speed))
            jpeg_param_data['batch_number'].append((bn, speed))
            jpeg_param_data['repetition'].append((rep, speed))

        for i, param in enumerate(parameters):
            plot_param(axes[i][0], jpeg_param_data[param], param, jpeg_avg_speeds, jpeg_reg, "JPEG")

        # TFRecord plots
        tfrec_param_data = defaultdict(list)
        for (dtype, bs, bn, rep), speed in tfrec_data:
            tfrec_param_data['batch_size'].append((bs, speed))
            tfrec_param_data['batch_number'].append((bn, speed))
            tfrec_param_data['repetition'].append((rep, speed))

        for i, param in enumerate(parameters):
            plot_param(axes[i][1], tfrec_param_data[param], param, tfrec_avg_speeds, tfrec_reg, "TFRecord")

        # batch_size × batch_number plots
        def plot_product(ax, data, reg, label):
            x_vals = np.array([d[0][1] * d[0][2] for d in data])
            y_vals = np.array([d[1] for d in data])
            ax.scatter(x_vals, y_vals, color='purple', label='Raw data')
            coef = reg['batch_size_x_batch_number']['coef']
            intercept = reg['batch_size_x_batch_number']['intercept']
            x_line = np.array(sorted(set(x_vals)))
            y_line = coef * x_line + intercept
            ax.plot(x_line, y_line, 'm--', label='Regression')
            ax.set_xlabel('batch_size x batch_number')
            ax.set_ylabel('Images/sec')
            ax.set_title(f"{label}: batch_size x batch_number")
            ax.legend()

        plot_product(axes[3][0], jpeg_data, jpeg_reg, "JPEG")
        plot_product(axes[3][1], tfrec_data, tfrec_reg, "TFRecord")

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Leave space for suptitle
        plt.show()

    # Split average speeds by data type
    jpeg_average_speeds = [p for p in average_speeds if p[0][0] != 'tfrec']
    tfrec_average_speeds = [p for p in average_speeds if p[0][0] == 'tfrec']

    # Plot final results
    plot_results(
        jpeg_results, tfrec_results,
        jpeg_average_speeds, tfrec_average_speeds,
        jpeg_regression_results, tfrec_regression_results,
        "Performance vs Parameters (JPEG vs TFRecord)"
    )

    # ------------------------------
    # Final Summary Print
    # ------------------------------
    print("\n--- Final Regression Summary ---")
    print("\nJPEG:")
    for param, vals in jpeg_regression_results.items():
        print(f"  {param}: Coef = {vals['coef']:.2f}, Intercept = {vals['intercept']:.2f}")

    print("\nTFRecord:")
    for param, vals in tfrec_regression_results.items():
        print(f"  {param}: Coef = {vals['coef']:.2f}, Intercept = {vals['intercept']:.2f}")
```

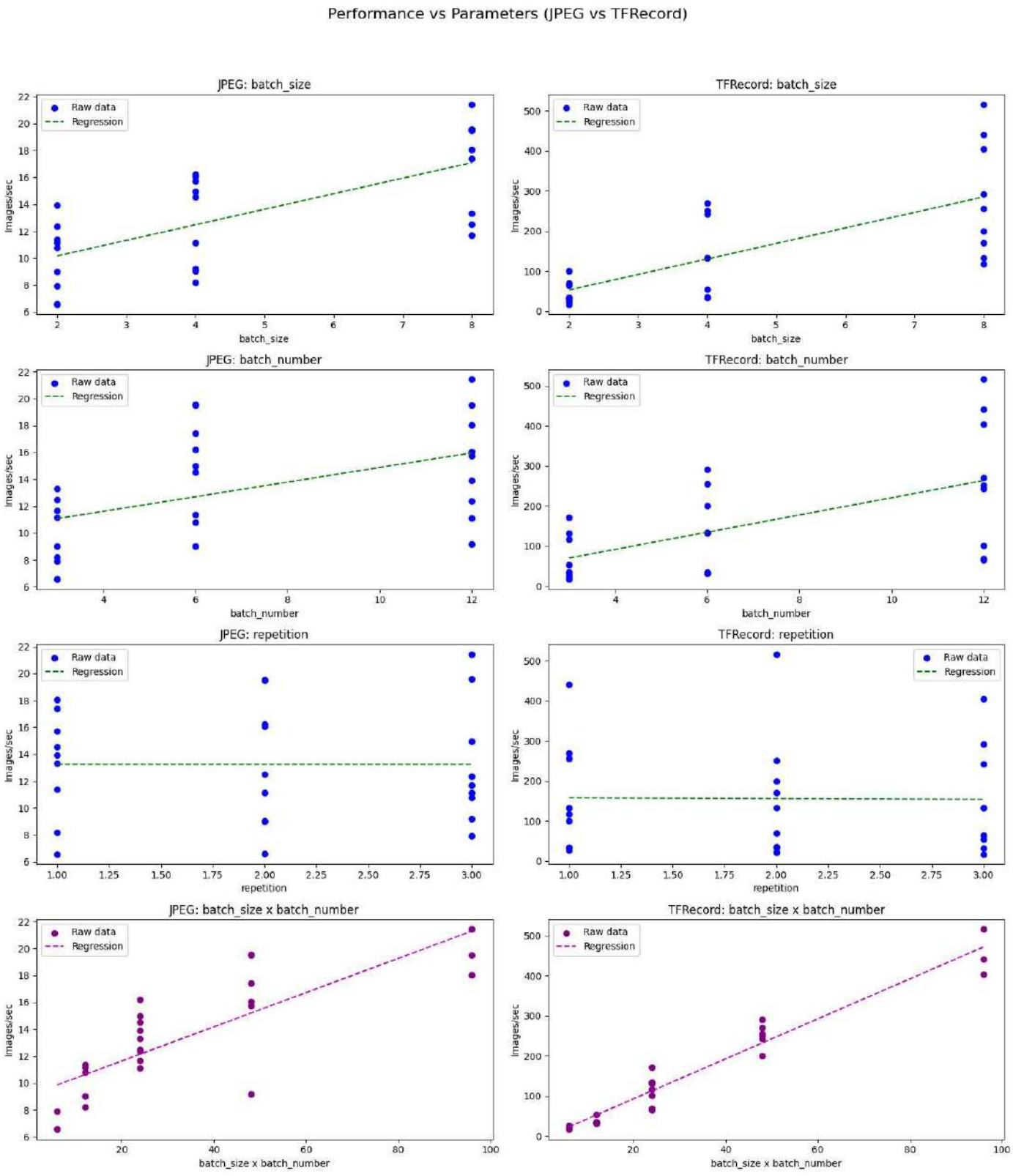
```
--- All Results (JPEG and TFRecord) ---
(('tfrec', 2, 3, 1), 26.333834913765138)
(('tfrec', 2, 3, 2), 20.788941124573434)
(('tfrec', 2, 3, 3), 17.274858610272318)
(('tfrec', 2, 6, 1), 34.24126614382357)
(('tfrec', 2, 6, 2), 33.1235171245543)
(('tfrec', 2, 6, 3), 31.636560433406057)
(('tfrec', 2, 12, 1), 100.79190647265911)
(('tfrec', 2, 12, 2), 69.27757540554546)
(('tfrec', 2, 12, 3), 64.57502607678185)
(('tfrec', 4, 3, 1), 33.86251314458948)
(('tfrec', 4, 3, 2), 35.19641516004649)
(('tfrec', 4, 3, 3), 53.56504859328506)
(('tfrec', 4, 6, 1), 132.86375773286363)
(('tfrec', 4, 6, 2), 133.31068211886324)
(('tfrec', 4, 6, 3), 132.33625928959341)
(('tfrec', 4, 12, 1), 269.79811635299353)
(('tfrec', 4, 12, 2), 250.94774419923593)
(('tfrec', 4, 12, 3), 242.85329039028716)
(('tfrec', 8, 3, 1), 116.77754640612801)
(('tfrec', 8, 3, 2), 170.7697381367606)
(('tfrec', 8, 3, 3), 132.09660872155507)
(('tfrec', 8, 6, 1), 255.64405029922784)
(('tfrec', 8, 6, 2), 199.70617608722824)
(('tfrec', 8, 6, 3), 291.87961592382266)
(('tfrec', 8, 12, 1), 441.39630310909797)
(('tfrec', 8, 12, 2), 516.2835876585285)
(('tfrec', 8, 12, 3), 404.7628077540139)
(('jpeg', 2, 3, 1), 6.577570448138346)
(('jpeg', 2, 3, 2), 6.600994798853959)
(('jpeg', 2, 3, 3), 7.909169936772712)
(('jpeg', 2, 6, 1), 11.368692959343514)
(('jpeg', 2, 6, 2), 9.000779517450264)
(('jpeg', 2, 6, 3), 10.791029224229087)
(('jpeg', 2, 12, 1), 13.915415677291257)
(('jpeg', 2, 12, 2), 11.12545266819761)
(('jpeg', 2, 12, 3), 12.360899430442256)
(('jpeg', 4, 3, 1), 8.192076000705084)
(('jpeg', 4, 3, 2), 9.04008796599472)
(('jpeg', 4, 3, 3), 11.146037318182483)
(('jpeg', 4, 6, 1), 14.538006775419413)
(('jpeg', 4, 6, 2), 16.201143828324355)
(('jpeg', 4, 6, 3), 14.974097563479036)
(('jpeg', 4, 12, 1), 15.722084627682973)
(('jpeg', 4, 12, 2), 16.06234654327393)
(('jpeg', 4, 12, 3), 9.176433491609322)
(('jpeg', 8, 3, 1), 13.302611553070674)
(('jpeg', 8, 3, 2), 12.48747026476639)
(('jpeg', 8, 3, 3), 11.690872150457736)
(('jpeg', 8, 6, 1), 17.411723884977413)
(('jpeg', 8, 6, 2), 19.530343409258467)
(('jpeg', 8, 6, 3), 19.5691993298556)
(('jpeg', 8, 12, 1), 18.04026453050835)
(('jpeg', 8, 12, 2), 19.501850396877558)
(('jpeg', 8, 12, 3), 21.423725754642312)

--- Average Speeds ---
('tfrec', 'batch_number', 6): 107.91
('jpeg', 'repetition', 1): 14.00
('jpeg', 'batch_size', 4): 13.19
('jpeg', 'batch_number', 6): 13.66
('tfrec', 'batch_size', 8): 179.33
('jpeg', 'batch_number', 3): 11.61
('tfrec', 'repetition', 1): 123.91
('tfrec', 'batch_number', 12): 181.27
('jpeg', 'repetition', 3): 14.41
('jpeg', 'batch_size', 8): 18.26
('tfrec', 'batch_size', 4): 89.00
('tfrec', 'batch_number', 3): 62.48
('tfrec', 'batch_size', 2): 83.33
('jpeg', 'batch_number', 12): 17.45
('jpeg', 'repetition', 2): 14.31
('tfrec', 'repetition', 3): 106.98
('tfrec', 'repetition', 2): 120.77
('jpeg', 'batch_size', 2): 11.28

--- Linear Regression Analysis for JPEG ---
batch_size: Coefficient = 1.16, Intercept = 7.86
batch_number: Coefficient = 0.54, Intercept = 9.44
repetition: Coefficient = -0.00, Intercept = 13.25
batch_size x batch_number: Coefficient = 0.13, Intercept = 9.10

--- Linear Regression Analysis for TFRecord ---
```

```
batch_size: Coefficient = 38.77, Intercept = -24.92
batch_number: Coefficient = 21.51, Intercept = 5.41
repetition: Coefficient = -2.26, Intercept = 160.53
batch_size x batch_number: Coefficient = 5.00, Intercept = -7.25
```




```
--- Final Regression Summary ---

JPEG:
  batch_size: Coef = 1.16, Intercept = 7.86
  batch_number: Coef = 0.54, Intercept = 9.44
  repetition: Coef = -0.00, Intercept = 13.25
  batch_size_x_batch_number: Coef = 0.13, Intercept = 9.10

TFRecord:
  batch_size: Coef = 38.77, Intercept = -24.92
  batch number: Coef = 21.51, Intercept = 5.41
```

```
repetition: Coef = -2.26, Intercept = 160.53
batch_size_x_batch_number: Coef = 5.00, Intercept = -7.25
```

```
from sklearn.linear_model import LinearRegression
from collections import defaultdict
import numpy as np
import matplotlib.pyplot as plt
import pickle
import tensorflow as tf

# === Load Pickled Result File ===
output_pickle = 'gs://bigdata06-storage/20250501-230337_spark_speed_test_results.pkl'

with tf.io.gfile.GFile(output_pickle, 'rb') as f:
    results_data = pickle.load(f)

all_results = results_data["all_results"]
average_speeds = results_data["average_speeds"]

# === Separate Results by Data Type ===
jpeg_results = [r for r in all_results if r[0][0] == 'jpeg']
tfrec_results = [r for r in all_results if r[0][0] == 'tfrec']

# === Print Loaded Results ===
print("\n--- All Results ---")
for result in all_results:
    print(result)

print("\n--- Average Speeds ---")
for param, avg_speed in average_speeds:
    print(f"{param}: {avg_speed:.2f}")

# === Linear Regression Function ===
def perform_linear_regression(data, parameter_name):
    index_map = {'batch_size': 1, 'batch_number': 2, 'repetition': 3}
    param_index = index_map[parameter_name]
    x = np.array([d[0][param_index] for d in data]).reshape(-1, 1)
    y = np.array([d[1] for d in data])
    model = LinearRegression().fit(x, y)
    return model.coef_[0], model.intercept_

# === Run Regression and Print Results ===
def analyze_and_print_regression(results, data_type):
    print(f"\n--- Linear Regression Analysis for {data_type} ---")
    regression_results = {}
    for param in ['batch_size', 'batch_number', 'repetition']:
        coef, intercept = perform_linear_regression(results, param)
        regression_results[param] = {'coef': coef, 'intercept': intercept}
        print(f"{param}: Coefficient = {coef:.2f}, Intercept = {intercept:.2f}")

    # Additional regression for batch_size × batch_number
    x = np.array([d[0][1] * d[0][2] for d in results]).reshape(-1, 1)
    y = np.array([d[1] for d in results])
    model = LinearRegression().fit(x, y)
    regression_results['batch_size_x_batch_number'] = {
        'coef': model.coef_[0], 'intercept': model.intercept_
    }
    print(f"batch_size x batch_number: Coef = {model.coef_[0]:.2f}, Intercept = {model.intercept_:.2f}")
    return regression_results

# Run regression analysis for both data types
jpeg_regression_results = analyze_and_print_regression(jpeg_results, "JPEG")
tfrec_regression_results = analyze_and_print_regression(tfrec_results, "TFRecord")

# === Plot Results Cleanly ===
def plot_results(jpeg_data, tfrec_data, jpeg_avg_speeds, tfrec_avg_speeds, jpeg_reg, tfrec_reg, title):
    fig, axs = plt.subplots(nrows=3, ncols=4, figsize=(24, 14))
    fig.suptitle(title, fontsize=18)

    param_list = ['batch_size', 'batch_number', 'repetition']
    index_map = {'batch_size': 1, 'batch_number': 2, 'repetition': 3}

    for i, param in enumerate(param_list):
        # === JPEG ===
        ax_jpeg = axs[i][0]
        x_jpeg = [d[0][index_map[param]] for d in jpeg_data]
        y_jpeg = [d[1] for d in jpeg_data]
        ax_jpeg.scatter(x_jpeg, y_jpeg, color='blue', label='Raw data')
        avg_x_jpeg = [p[0][1] for p in jpeg_avg_speeds if p[0][0] == param]
```

```
            avg_y_jpeg = [p[1] for p in jpeg_avg_speeds if p[0][0] == param]
            if avg_x_jpeg:
                ax_jpeg.plot(avg_x_jpeg, avg_y_jpeg, 'ro-', label='Average speed')
            coef = jpeg_reg[param]['coef']
            intercept = jpeg_reg[param]['intercept']
            x_line = np.linspace(min(x_jpeg), max(x_jpeg), 100)
            ax_jpeg.plot(x_line, coef * x_line + intercept, 'g--', label='Regression line')
            ax_jpeg.set_title(f"JPEG: {param}")
            ax_jpeg.set_xlabel(param)
            ax_jpeg.set_ylabel("Images/sec")
            ax_jpeg.legend()

            # === TFRecord ===
            ax_tfrec = axs[i][1]
            x_tfrec = [d[0][index_map[param]] for d in tfrec_data]
            y_tfrec = [d[1] for d in tfrec_data]
            ax_tfrec.scatter(x_tfrec, y_tfrec, color='blue', label='Raw data')
            avg_x_tfrec = [p[0][1] for p in tfrec_avg_speeds if p[0][0] == param]
            avg_y_tfrec = [p[1] for p in tfrec_avg_speeds if p[0][0] == param]
            if avg_x_tfrec:
                ax_tfrec.plot(avg_x_tfrec, avg_y_tfrec, 'ro-', label='Average speed')
            coef = tfrec_reg[param]['coef']
            intercept = tfrec_reg[param]['intercept']
            x_line = np.linspace(min(x_tfrec), max(x_tfrec), 100)
            ax_tfrec.plot(x_line, coef * x_line + intercept, 'g--', label='Regression line')
            ax_tfrec.set_title(f"TFRecord: {param}")
            ax_tfrec.set_xlabel(param)
            ax_tfrec.set_ylabel("Images/sec")
            ax_tfrec.legend()

        # === batch_size x batch_number plots ===
        x_j = [d[0][1] * d[0][2] for d in jpeg_data]
        y_j = [d[1] for d in jpeg_data]
        ax_j_product = axs[0][2]
        ax_j_product.scatter(x_j, y_j, color='purple', label='Raw data')
        coef = jpeg_reg['batch_size_x_batch_number']['coef']
        intercept = jpeg_reg['batch_size_x_batch_number']['intercept']
        x_line = np.linspace(min(x_j), max(x_j), 100)
        ax_j_product.plot(x_line, coef * x_line + intercept, 'm--', label='Regression line')
        ax_j_product.set_title("JPEG: batch_size x batch_number")
        ax_j_product.set_xlabel("batch_size × batch_number")
        ax_j_product.set_ylabel("Images/sec")
        ax_j_product.legend()

        x_t = [d[0][1] * d[0][2] for d in tfrec_data]
        y_t = [d[1] for d in tfrec_data]
        ax_t_product = axs[0][3]
        ax_t_product.scatter(x_t, y_t, color='purple', label='Raw data')
        coef = tfrec_reg['batch_size_x_batch_number']['coef']
        intercept = tfrec_reg['batch_size_x_batch_number']['intercept']
        x_line = np.linspace(min(x_t), max(x_t), 100)
        ax_t_product.plot(x_line, coef * x_line + intercept, 'm--', label='Regression line')
        ax_t_product.set_title("TFRecord: batch_size x batch_number")
        ax_t_product.set_xlabel("batch_size × batch_number")
        ax_t_product.set_ylabel("Images/sec")
        ax_t_product.legend()

        # Remove empty plots
        axs[1][2].axis('off')
        axs[1][3].axis('off')
        axs[2][2].axis('off')
        axs[2][3].axis('off')

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        plt.show()

    # Separate average speeds
    jpeg_avg = [p for p in average_speeds if p[0][0] != 'tfrec']
    tfrec_avg = [p for p in average_speeds if p[0][0] == 'tfrec']

    # Plot
    plot_results(jpeg_results, tfrec_results, jpeg_avg, tfrec_avg, jpeg_regression_results, tfrec_regression_results, "Performance Analysis (JF

    # Final regression summary
    print("\n--- Final Regression Summary ---")
    print("\nJPEG:")
    for param, res in jpeg_regression_results.items():
```