Data Mining Assignment-1

Apoorva Bandi

Petals to the Metal - Flower Classification

Description of the project:

TPUs are strong hardware accelerators with a focus on deep learning applications. Google developed (and used them in the beginning) for processing huge image databases, like when it extracted all the text from Street View. For this project, we develop an image classifier model which classifies various types of flowers. Based on photographs taken from five distinct public datasets, we are categorizing 104 different kinds of flowers. The files for this competition are available in TFRecord format. Tensorflow commonly uses the TFRecord container format to group and shard data files for best training performance.

For numerous images, each file includes the id, label (the sample class for training data), and img (the actual pixels in array form) information.

training samples with labels are found in- **train/*.tfrec.**

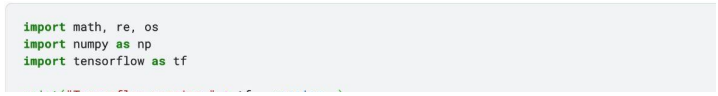
Pre-split training samples with labels (**val/*.tfrec**) are meant to assist in evaluating your model's performance on TPU. The division was arranged according to labels.

You will be making predictions about the classes of flowers that these **test/*.tfrec** samples, which lack labels, belong to.

Implementation:

1.1. Importing libraries:

Here we import the 'numpy' and 'tensorflow' libraries for data analysis and numerical operations. Deep neural network training and inference are tensorflow primary areas of focus.



```
import math, re, os
import numpy as np
import tensorflow as tf
```

```
print("Tensorflow version " + tf.__version__)
```

```
Tensorflow version 2.13.0
```

+ Code + Markdown

[57]:

```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

```
REPLICAS:  1
```

[58]:

```
GCS_DS_PATH = '/kaggle/input/tpu-getting-started'
print(GCS_DS_PATH)
```

```
/kaggle/input/tpu-getting-started
```

There are eight distinct cores in a TPU, and each core functions as an independent accelerator. (Having eight GPUs in one machine is similar to having a TPU.) We define a distribution strategy that instructs TensorFlow on how to use all of these

cores simultaneously. The distribution strategy will be applied in the development of our neural network model. TensorFlow will then create eight distinct model replicas, one for each of the eight TPU cores, and divide the training among them.

1.2. Load the data

[58]:
```
GCS_DS_PATH = '/kaggle/input/tpu-getting-started'
print(GCS_DS_PATH)
```

```
/kaggle/input/tpu-getting-started
```

Here, we plug in to the location of the dataset file and then we check whether it is properly located or not. The code is successfully implemented as we got the output.

```
IMAGE_SIZE = [192,192]
GCS_PATH = GCS_DS_PATH + '/tfrecords-jpeg-192x192'
AUTO = tf.data.experimental.AUTOTUNE

TRAINING_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/train/*.tfrec')
VALIDATION_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/val/*.tfrec')
TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/test/*.tfrec')

CLASSES = ['pink primrose',    'hard-leaved pocket orchid', 'canterbury bells', 'sweet pea',     'wild geranium',     'tiger lily',           'moon o
           'snapdragon',       "colt's foot",               'king protea',      'spear thistle', 'yellow iris',       'globe-flower',         'purple
           'fire lily',        'pincushion flower',         'fritillary',       'red ginger',    'grape hyacinth',    'corn poppy',           'prince
           'carnation',        'garden phlox',              'love in the mist', 'cosmos',        'alpine sea holly',  'ruby-lipped cattleya', 'cape f
           'barberton daisy',  'daffodil',                  'sword lily',       'poinsettia',    'bolero deep blue',  'wallflower',           'marigo
           'petunia',          'wild pansy',                'primula',          'sunflower',     'lilac hibiscus',    'bishop of llandaff',   'gaura'
           'cautleya spicata', 'japanese anemone',          'black-eyed susan', 'silverbush',    'californian poppy', 'osteospermum',         'spring
           'gazania',          'azalea',                    'water lily',       'rose',          'thorn apple',       'morning glory',        'passio
           'frangipani',       'clematis',                  'hibiscus',         'columbine',     'desert-rose',       'tree mallow',          'magnol
           'hippeastrum ',     'bee balm',                  'pink quill',       'foxglove',      'bougainvillea',     'camellia',             'mallow
           'trumpet creeper',  'blackberry lily',           'common tulip',     'wild rose']


def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)
    image = tf.cast(image, tf.float32) / 255.0  # convert image to floats in [0, 1] range
    image = tf.reshape(image, [*IMAGE_SIZE, 3]) # explicit size needed for TPU
    return image
```

Datasets are frequently serialized into TFRecords when used with TPUs. Data distribution to every TPU core is made easy with this format. Because the process takes a while, we've hidden the cell that reads the TFRecords for the dataset.

```
        'id' : tf.io.FixedLenFeature([], tf.string),  # shape [] means single element
        # class is missing, this competitions's challenge is to predict flower classes for the test dataset
    }
    example = tf.io.parse_single_example(example, UNLABELED_TFREC_FORMAT)
    image = decode_image(example['image'])
    idnum = example['id']
    return image, idnum # returns a dataset of image(s)

def load_dataset(filenames, labeled=True, ordered=False):
    # Read from TFRecords. For optimal performance, reading from multiple files at once and
    # disregarding data order. Order does not matter since we will be shuffling the data anyway.

    ignore_order = tf.data.Options()
    if not ordered:
        ignore_order.experimental_deterministic = False # disable order, increase speed

    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=AUTO) # automatically interleaves reads from multiple files
    dataset = dataset.with_options(ignore_order) # uses data as soon as it streams in, rather than in its original order
    dataset = dataset.map(read_labeled_tfrecord if labeled else read_unlabeled_tfrecord, num_parallel_calls=AUTO)
    # returns a dataset of (image, label) pairs if labeled=True or (image, id) pairs if labeled=False
    return dataset
```

+ Code + Markdown

```
def data_augment(image, label):
    # Thanks to the dataset.prefetch(AUTO)
    # statement in the next function (below), this happens essentially
    # for free on TPU. Data pipeline code is executed on the "CPU"
    # part of the TPU while the TPU itself is computing gradients.
    image = tf.image.random_flip_left_right(image)
    #image = tf.image.random_saturation(image, 0, 2)
    return image, label
```

In the last step, we will define an effective data pipeline for each of the test, validation, and training splits using the tf.data API. The datasets that we'll use with Keras for training and inference will be created in the following cell. Take note of how we match the batch sizes to the TPU core count.

```
def count_data_items(filenames):
    # the number of data items is written in the name of the .tfrec
    # files, i.e. flowers00-230.tfrec = 230 data items
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in filenames]
    return np.sum(n)

NUM_TRAINING_IMAGES = count_data_items(TRAINING_FILENAMES)
NUM_VALIDATION_IMAGES = count_data_items(VALIDATION_FILENAMES)
NUM_TEST_IMAGES = count_data_items(TEST_FILENAMES)
print('Dataset: {} training images, {} validation images, {} unlabeled test images'.format(NUM_TRAINING_IMAGES, NUM_VALIDATION_IMAGES, NUM_TEST_IMAGE
```

```
Dataset: 12753 training images, 3712 validation images, 7382 unlabeled test images
```

+ Code + Markdown

```
# Define the batch size. This will be 16 with TPU off and 128 (=16*8) with TPU on
BATCH_SIZE = 16 * strategy.num_replicas_in_sync

ds_train = get_training_dataset()
ds_valid = get_validation_dataset()
ds_test = get_test_dataset()

print("Training:", ds_train)
print ("Validation:", ds_valid)
print("Test:", ds_test)
```

```
Training: <_PrefetchDataset element_spec=(TensorSpec(shape=(None, 192, 192, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=Non
e))>
Validation: <_PrefetchDataset element_spec=(TensorSpec(shape=(None, 192, 192, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=Non
e))>
Test: <_PrefetchDataset element_spec=(TensorSpec(shape=(None, 192, 192, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.string, name=None))>
```

```
np.set_printoptions(threshold=15, linewidth=80)

print("Training data shapes:")
for image, label in ds_train.take(3):
    print(image.numpy().shape, label.numpy().shape)
print("Training data label examples:", label.numpy())
```

```
Training data shapes:
(16, 192, 192, 3) (16,)
(16, 192, 192, 3) (16,)
(16, 192, 192, 3) (16,)
Training data label examples: [ 67  67 102 ...  64  10  43]
```

```
print("Test data shapes:")
for image, idnum in ds_test.take(3):
    print(image.numpy().shape, idnum.numpy().shape)
print("Test data IDs:", idnum.numpy().astype('U')) # U=unicode string
```

```
Test data shapes:
(16, 192, 192, 3) (16,)
(16, 192, 192, 3) (16,)
(16, 192, 192, 3) (16,)
Test data IDs: ['6557acff6' '54f73d67c' 'dd20b2594' ... 'ff30e8b96' '6485b01ac' '0dd7f1f16']
```

+ Code + Markdown

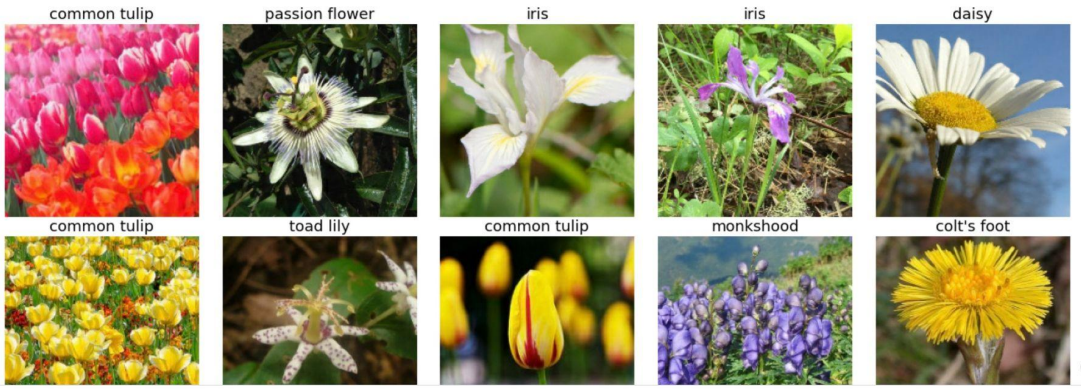
The test set consists of a stream of (image, idnum) pairs, where idnum is the image's unique identifier that we'll need to submit the file as a CSV later on.

1.3. Exploring the data

```
from matplotlib import pyplot as plt

def batch_to_numpy_images_and_labels(data):
    images, labels = data
    numpy_images = images.numpy()
    numpy_labels = labels.numpy()
    if numpy_labels.dtype == object: # binary string in this case,
                                     # these are image ID strings
        numpy_labels = [None for _ in enumerate(numpy_images)]
    # If no labels, only image IDs, return None for labels (this is
    # the case for test data)
    return numpy_images, numpy_labels

def title_from_label_and_target(label, correct_label):
    if correct_label is None:
        return CLASSES[label], True
    correct = (label == correct_label)
    return "{} [{}{}{}]".format(CLASSES[label], 'OK' if correct else 'NO', u"\u2192" if not correct else '',
                                CLASSES[correct_label] if not correct else ''), correct

def display_one_flower(image, title, subplot, red=False, titlesize=16):
    plt.subplot(*subplot)
    plt.axis('off')
    plt.imshow(image)
    if len(title) > 0:
        plt.title(title, fontsize=int(titlesize) if not red else int(titlesize/1.2), color='red' if red else 'black', fontdict={'verticalalignment':'
    return (subplot[0], subplot[1], subplot[2]+1)

def display_batch_of_images(databatch, predictions=None):
    """This will work with:
    display_batch_of_images(images)
    display_batch_of_images(images, predictions)
```

```
ds_iter = iter(ds_train.unbatch().batch(20))
```

```
one_batch = next(ds_iter)
display_batch_of_images(one_batch)
```



Here, by re-running the cell, we get a different batch of images.

1.4. Defining Model

```
def display_training_curves(training, validation, title, subplot):
    if subplot%10==1: # set up the subplots on the first call
        plt.subplots(figsize=(10,10), facecolor='#F0F0F0')
        plt.tight_layout()
    ax = plt.subplot(subplot)
    ax.set_facecolor('#F8F8F8')
    ax.plot(training)
    ax.plot(validation)
    ax.set_title('model '+ title)
    ax.set_ylabel(title)
    #ax.set_ylim(0.28,1.05)
    ax.set_xlabel('epoch')
    ax.legend(['train', 'valid.'])
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix

def display_confusion_matrix(cmat, score, precision, recall):
    plt.figure(figsize=(15,15))
    ax = plt.gca()
    ax.matshow(cmat, cmap='Reds')
    ax.set_xticks(range(len(CLASSES)))
    ax.set_xticklabels(CLASSES, fontdict={'fontsize': 7})
```

```
EPOCHS = 12
from tensorflow import keras
with strategy.scope():

    model = tf.keras.Sequential([
        keras.layers.Conv2D(128, 5, activation='relu', padding='same',input_shape=(192,192, 3)),
        keras.layers.MaxPooling2D(2),
        keras.layers.Conv2D(200, 5, activation='relu', padding='same'),
        keras.layers.Conv2D(200, 5, activation='relu', padding='same'),
        keras.layers.MaxPooling2D(3),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.3),
        keras.layers.Conv2D(300, 5, activation='relu', padding='same'),
        keras.layers.Conv2D(300, 5, activation='relu', padding='same'),
        keras.layers.MaxPooling2D(3),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.3),
        keras.layers.Conv2D(400, 5, activation='relu', padding='same'),
        keras.layers.Conv2D(400, 5, activation='relu', padding='same'),
        keras.layers.MaxPooling2D(3),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.3),

        keras.layers.Flatten(),

        tf.keras.layers.Dense(len(CLASSES), activation='softmax')
    ])
```

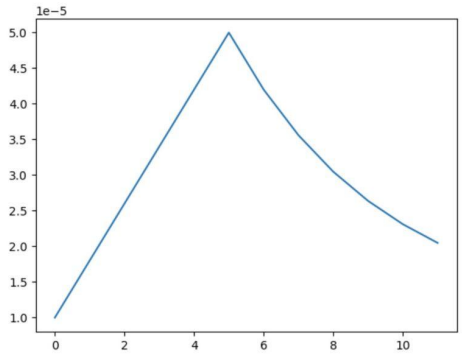
Transfer learning allows you to start off on a new dataset by reusing a portion of a pretrained model.

1.5. Training model

```
# Learning Rate Schedule for Fine Tuning #
def exponential_lr(epoch,
                   start_lr = 0.00001, min_lr = 0.00001, max_lr = 0.00005,
                   rampup_epochs = 5, sustain_epochs = 0,
                   exp_decay = 0.8):

    def lr(epoch, start_lr, min_lr, max_lr, rampup_epochs, sustain_epochs, exp_decay):
        # linear increase from start to rampup_epochs
        if epoch < rampup_epochs:
            lr = ((max_lr - start_lr) /
                  rampup_epochs * epoch + start_lr)
        # constant max_lr during sustain_epochs
        elif epoch < rampup_epochs + sustain_epochs:
            lr = max_lr
        # exponential decay towards min_lr
        else:
            lr = ((max_lr - min_lr) *
                  exp_decay**(epoch - rampup_epochs - sustain_epochs) +
                  min_lr)
        return lr
    return lr(epoch,
              start_lr,
              min_lr,
              max_lr,
              rampup_epochs,
              sustain_epochs,
              exp_decay)

lr_callback = tf.keras.callbacks.LearningRateScheduler(exponential_lr, verbose=True)

rng = [i for i in range(EPOCHS)]
y = [exponential_lr(x) for x in rng]
```

```
lr_callback = tf.keras.callbacks.LearningRateScheduler(exponential_lr, verbose=True)

rng = [i for i in range(EPOCHS)]
y = [exponential_lr(x) for x in rng]
plt.plot(rng, y)
print("Learning rate schedule: {:.3g} to {:.3g} to {:.3g}".format(y[0], max(y), y[-1]))
```

```
Learning rate schedule: 1e-05 to 5e-05 to 2.05e-05
```



This network will be trained using a unique learning rate schedule.

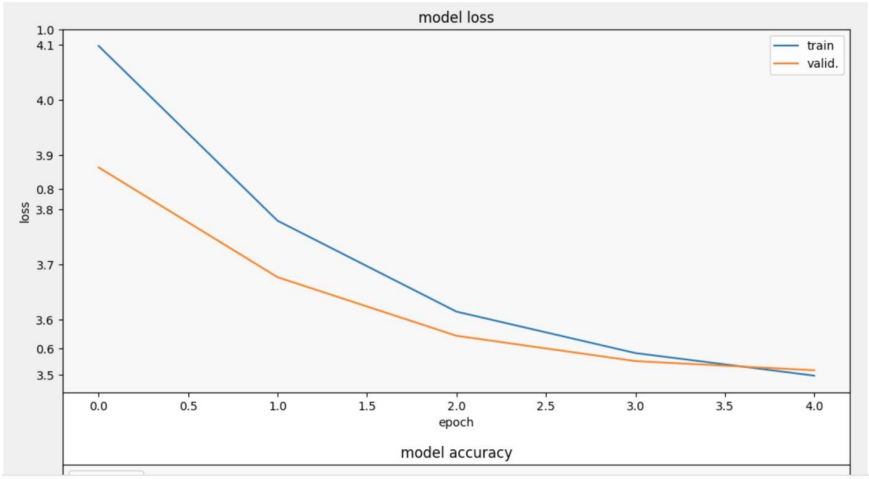
1.6. Fit Model

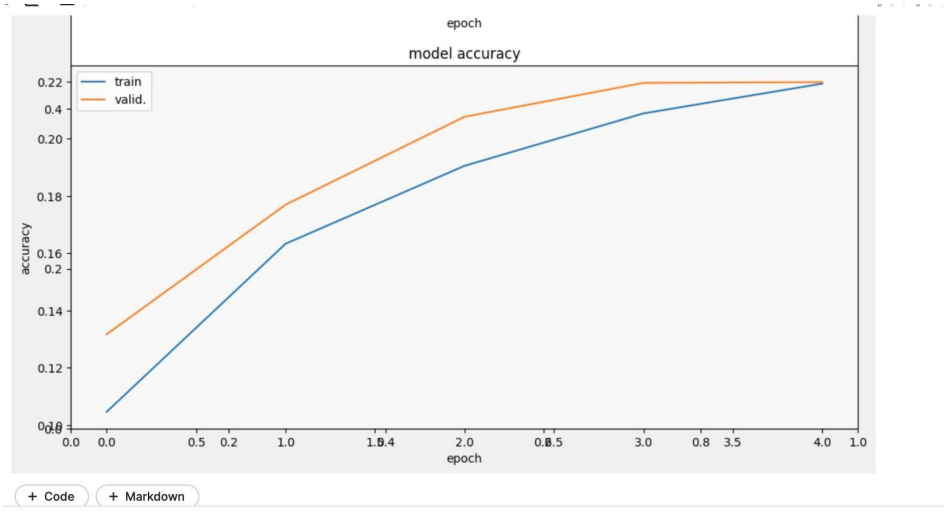
Here we train the model, by defining a few paramenters.

```
# Define training epochs
EPOCHS = 5
STEPS_PER_EPOCH = NUM_TRAINING_IMAGES // BATCH_SIZE

history = model.fit(
    ds_train,
    validation_data=ds_valid,
    epochs=EPOCHS,
    steps_per_epoch=STEPS_PER_EPOCH,
    callbacks=[lr_callback],
)
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 1/5
2023-11-14 17:18:29.122106: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.140587: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.159404: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.177126: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.195034: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.213648: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:29.232360: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
2023-11-14 17:18:30.527980: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:961] model_pruner failed: INVALID_ARGUMENT: Graph does not contain terminal no
[redacted]leOp.
[redacted]775: E ./tensorflow/compiler/xla/stream_executor/stream_executor_internal.h:124] SetPriority unimplemented for this stream.
```

The progression of the metrics and loss during training is displayed in the following cell.





+ Code + Markdown

1.6. Predictions Evaluation

```
cmdataset = get_validation_dataset(ordered=True)
images_ds = cmdataset.map(lambda image, label: image)
labels_ds = cmdataset.map(lambda image, label: label).unbatch()

cm_correct_labels = next(iter(labels_ds.batch(NUM_VALIDATION_IMAGES))).numpy()
cm_probabilities = model.predict(images_ds)
cm_predictions = np.argmax(cm_probabilities, axis=-1)

labels = range(len(CLASSES))
cmat = confusion_matrix(
    cm_correct_labels,
    cm_predictions,
    labels=labels,
)
cmat = (cmat.T / cmat.sum(axis=1)).T # normalize
```

```
2023-11-14 17:23:45.639485: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:961] model_pruner failed: INVALID_ARGUMENT: Graph does not contain terminal no
de AssignAddVariableOp.
29/29 [==============================] - 17s 365ms/step
```

1.7. Validating Visuals

Examining a few samples from the validation set to see which class your model predicted can also be beneficial. This can assist in identifying trends in the types of images that your model struggles with. The validation set will be configured in this cell to show 20 images at once.

```
dataset = get_validation_dataset()
dataset = dataset.unbatch().batch(20)
batch = iter(dataset)
```

```
images, labels = next(batch)
probabilities = model.predict(images)
predictions = np.argmax(probabilities, axis=-1)
display_batch_of_images((images, labels), predictions)
```

```
1/1 [==============================] - 11s 11s/step
```

1.8. Predictions

Here we generate a submission.csv file.

```
print('Generating submission.csv file...')

# Get image ids from test set and convert to unicode
test_ids_ds = test_ds.map(lambda image, idnum: idnum).unbatch()
test_ids = next(iter(test_ids_ds.batch(NUM_TEST_IMAGES))).numpy().astype('U')

# Write the submission file
np.savetxt(
    'submission.csv',
    np.rec.fromarrays([test_ids, predictions]),
    fmt=['%s', '%d'],
    delimiter=',',
    header='id,label',
    comments='',
)

# Look at the first few predictions
!head submission.csv
```

```
Generating submission.csv file...
id,label
0b9afbdf2,49
c37a6f3e9,103
00e4f514e,103
1c4736dea,4
252d840db,67
dfc9c6a23,103
53cfc6586,48
541c4d41e,73
59d1b6146,53
```

Kaggle Screenshot: