The Journal of Open Source Software

Datum: Creating and loading TFRecord datasets.

Mrinal Haloi[*1, 2] **and Shashank Shekhar**[2]

1 OpenAGI **2** Subex Limited

DOI: 10.21105/joss.03446

Software

- Review
- Repository
- Archive

Editor: Jacob Schreiber

Reviewers:

- @pmeier
- @younglululu

Submitted: 14 June 2021
Published: 02 July 2021

Summary

Deep learning model training efficiency depends on the performance of the input pipeline. Especially when training very deep neural network using GPU servers, efficient input pipeline can significantly help reducing overall learning time. Tensorflow provides TFRecord format to store data in a sequence of serialized protocol buffers as a binary record and `tf.data.Dataset` API for building input pipeline. TFRecord files are very read efficient and light on hard disk space usage. TFRecord files can be loaded as `tf.data.Dataset` for training neural network models. Creating and loading the TFRecord dataset involves writing a lot of complex codes and a time-consuming process. We develop `Datum` to automate this complex process.

Statement of need

`Datum` is a Tensorflow (Abadi, 2016) based Python package for creating and loading TFRecord datasets. It uses `tf.data` API to load TFRecord files as `tf.data.Dataset`, which is used for training models on a single machine or distributed servers. Large datasets can be handled easily using TFRecord and `Datum` by splitting the dataset into multiple smaller chunks of TFRecord files called shards. These shards can also be stored in different data servers.

When datasets don't fit in the RAM, using a python generator-based approach can become a huge bottleneck in training complex GPU compute-intensive models. GPU sitting idle waiting for data slow down model the training process by a huge margin. That's where Tensorflow's TFRecord format and `tf.data.Dataset` API come to the rescue.

`Datum` is designed to build an efficient input pipeline for training deep learning models using Tensorflow. `Datum` provides two high-level APIs for creating and loading TFRecord files. For creating a TFRecord dataset without writing a lot of codes, `Datum` provides `datum.export.export_to_tfrecord` API. `Datum` export API automatically identifies the sample data type and shape and chooses the correct serialization module. It also identifies whether the input dataset contains sparse or dense samples and chooses feature types accordingly. Samples meta information such as data types, shapes, and feature types are written as JSON(Pezoa et al., 2016) files in the disk. For loading TFRecord files, `Datum` makes use of the `tf.data` API. Using `datum.load` API TFRecord datasets can directly be load as `tf.data.Dataset`. `Datum` load API identifies the content of the TFRecord files based on the JSON metadata files written in the export step. `Datum` also provides utilities for sample transformation or augmentation. `Datum` provides `pre_batching_callback` for transforming samples before batching and `post_batching_callback` for transforming samples after batching.

`Datum` is designed to be used by deep learning researchers, software developers, data scientists, and data engineers to build efficient input pipelines. The combination of transformation features, fast run-time, and distributed training functionality in `Datum`(Mrinal Haloi, 2020) will enable researchers and developers to build input pipelines for complex problems.

*corresponding author

Features

One of the key features of `Datum` is its TFRecord export functionality, making the complex and tedious process easy. For training large models in distributed servers TFRecord dataset is key to run-time performance. `Datum` load functionality offers an easy and concise way to load TFRecord files. In addition, `Datum` makes it easy to alter the dataset samples individually or batch-wise by exposing callback functionalities. Furthermore, `Datum` supports loading TFRecord files split-wise, for training, validation and test it allows users to create different instances of `tf.data.Dataset` and apply split specific transformation on the fly.

Example Usage

`Datum` can be imported and used as a package:

Convert a dataset to the TFRecord format for a image classification problem as follows:

```
import datum

# Initialize a config object to define dataset properties

write_configs = datum.configs.TFRWriteConfigs()
write_configs.splits = {
    "train": {
        "num_examples": <num of train examples in the dataset>
    },
    "val": {
        "num_examples": <num of validation examples in the dataset>
    },
}

# Convert the dataset to TFRecord format
datum.export.export_to_tfrecord(input_path, output_path, datum.problem.IMAGE_CLF,

# Where input_path is the dataset location and output_path is directory to store T
```

Load the previously converted TFRecord dataset for a image classification problem

```
import datum

# Load the dataset as `tf.data.Dataset`
dataset = datum.load(<path to tfreord files folder>)

# Get dataset for training the model
train_dataset = dataset.train_fn('train', shuffle=True)

# Get dataset for validating the model
val_dataset = dataset.val_fn('val', shuffle=False)

# Augment the dataset samples
```

```python
# Define a augmentation function, for example
def augment_image(example):
  image = tf.image.resize(example["image"], IMG_SIZE)
  image = tf.image.random_flip_left_right(image)
  image = tf.image.random_flip_up_down(image)
  example.update({"image": image})
  return example

# Get the dataset configs
dataset_configs = dataset.dataset_configs

# Set pre batching callback for samplewise augmentation
datset_configs.pre_batching_callback = lambda example: augment_image(example)

# Reload the training dataset
train_dataset = dataset.train_fn('train', shuffle=True)
```

Conclusions

Since its launch in 2020, `Datum` is being continuously optimized, and new features are added. For each of the new `Tensorflow` releases, `Datum` is also updated to include enhancements and bug fixes. We are not aware of any equivalent open-source software in existence.

Acknowledgements

We acknowledge the contribution from Subex AI Labs in the development of `Datum`.

References

Abadi, B., M. (2016). *TensorFlow: Large-scale machine learning on heterogeneous systems*. arXiv. https://www.tensorflow.org/

Mrinal Haloi. (2020). Datum. *GitHub Repository*. https://github.com/openagi/datum

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016). Foundations of JSON schema. *Proceedings of the 25th International Conference on World Wide Web*.