Keras RFC Implementation- ImagePipeline based on tf.data API

Contact Information

Name: **Swarup Ghosh**
Email Address: **snwg@live.com**
GitHub: **swghosh** | LinkedIn: **swgghosh**

Abstract

tf.data.Dataset is the defacto way of loading and preprocessing data in TensorFlow for training machine learning models on CPUs, GPUs and TPUs. There are a wide range of advantages available with the tf.data API suitable for distribution of workers using various tf.distribute.Strategy(s). Whereas, the keras_preprocessing framework is popularly used to load and preprocess data for keras Model(s). It comprises a bunch of classes (inherited from keras.utils.Sequence) based on NumPy and SciPy implementations making it unsuitable for either multi-worker strategies or prefetching.

Recently, TensorFlow has introduced the Preprocessing Layers API that allows preprocessing layers to be serialized with the model itself. It is fundamentally aimed at unifying the functionalities available within keras.preprocessing. In this project, we would implement the API specifications (related to image operations by implementing ImagePipeline class) as discussed in the latest Keras RFC which propose changes and redesign of the Keras Preprocessing API in favour of performance and usability.

Background

As discussed by François Chollet at the Keras SIG meeting held on 13-09-2019 and on forthcoming meetings, the keras_preprocessing module would be required to be redesigned and refactored to support tf.data API. There is an existing RFC (governance/20190729-keras-preprocessing-redesign.md, currently in proposed state) in this regard which proposes various API changes in order to unify keras.preprocessing with tf.data API. The current implementation of keras_preprocessing module is based on NumPy and SciPy, making it almost impossible to provide support for multi-worker preprocessing ops as well as prefetching of data which is required for optimum model performance when models are trained on hardware accelerators like GPUs and TPUs. During training of machine learning models, it is almost always required that data preprocessing (loading, transformation, etc.) happens on the fly, such that the GPU/TPU device is continuously being fed with the required data samples to keep the training going on. Thus, requiring quite a lot of compute power to keep the data pipeline always ready for the consumer. Moreover, when we make use of Sequence loaders from keras.preprocessing, training bottlenecks are certain due to the lack of multi-worker strategies and or prefetching.

There are a bunch of improvements to the Keras API due to deprecation of multi-backend Keras (until v2.2.x) in favour of the future release cycles of tf.keras (Keras v2.3.x) and TensorFlow 2.x. The aforementioned RFC proposes a new ImagePipeline class inherited from:

- tf.keras.models.Sequential (or)
- tf.keras.layers.experimental.preprocessing.PreprocessingLayer (discussed in another RFC, governance/20190502-preprocessing-layers.md)

so as to make possible the deprecation of the existing ImageDataGenerator class, widely used by Keras users for random image data augmentation which improves regularization during training of deep learning models. The discussed ImagePipeline class can work similar to a Sequential model (readily allowing image processing ops to be chained together) as well as benefit from the existing features of the tf.data API.

Machine learning models trained using tf.data pipelines ensure that that the data preprocessing happens on the host CPU (although, improvements have been announced at the TF Dev Summit 2020 introducing the users multi-worker Kubernetes backed instances which should be able to improve scalability and performance further) while, the actual model training is done on a cluster of hardware accelerators like GPUs and TPUs. Introducing changes to the default method for image pipelining in TensorFlow models should require that these changes do not break the model computation i.e. by ensuring that ImagePipeline class runs on a CPU graph only.

The following random affine transformation operations that are currently available from ImageDataGenerator:

- Transforms that are required to be adapted based on dataset
    - Feature-wise Center
    - Sample-wise Center
    - Feature-wise Standard Normalisation
    - Sample-wise Standard Normalisation
- Transforms that are not required to be adapted based on dataset
    - Random ZCA Whitening
    - Random Rotation
    - Random Width Shift
    - Random Height Shift
    - Random Brightness
    - Random Shear
    - Random Zoom
    - Random Horizontal Flip
    - Random Vertical Flip
    - Random Channel Shifting

The master branch of the TensorFlow main repository currently includes the following random image transformation operations (tensorflow/image_preprocessing.py at master · tensorflow/tensorflow) :

- Resizing
- CenterCrop
- RandomCrop
- Rescaling
- RandomFlip
- RandomTranslation
- RandomRotation
- RandomZoom
- RandomContrast
- RandomHeight
- RandomWidth

Deliverables

The following functions will have to be added to the tf.image module in order to create the respective Layer(s) that can perform these random operations:

- tf.image.random_shear(image, intensity, seed=None)
- tf.image.random_zca_whiten(image, epsilon, seed=None)

The following PreprocessingLayer(s) which are currently inexistent are to be added:

- tf.keras.layers.experimental.preprocessing.RandomShear
- tf.keras.layers.experimental.preprocessing.RandomZCAWhiten
- tf.keras.layers.experimental.preprocessing.ImagePipeline

The ImagePipeline class will be interoperable with PreprocessingStage in the future and would readily include the capabilities that comes with the tf.data.API.

The necessary documentation for these classes and functions along with Colab tutorials to use the new ImagePipeline class will be bundled in the relevant sections. (eg. TensorFlow.org/tutorials)

Example:

```
normalization = tf.keras.layers.experimental.preprocessing.Normalization(axis=-1)
normalization.adapt(data_sample)

augmenter = tf.keras.layers.experimental.preprocessing.ImagePipeline([
    normalization,
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.5),
    tf.keras.layers.experimental.preprocessing.RandomFlip(vertical=True)
])

ds = input_pipeline.from_directory(input_dir, batch_size=32) # tf.data.Dataset

model = tf.keras.applications.ResNet50(weights=None)
model.fit(ds, epochs=10)
```

Community Benefits

The interoperability between tf.data APIs and PreprocessingLayer(s) would benefit users thus, allowing them to serialize complete models including the preprocessing layers as a part of the model itself (this is based on inspiration from NLP tasks where pre-processing ops like Word Vectorization; the same would be available for image models). The unification of keras_preprocessing module with the new PreprocessingLayer(s) API would make it easy for beginners to perform data pre-processing ops within the model itself.

The Keras team has always been taking decisions that are likely to increase usability of the Keras framework. The ImagePipeline class would substantially contribute to this goal. A majority of the API changes discussed in this project proposal are directly based on the latest RFC, making it more appropriate to the tf.keras project in general. The redesign and breaking changes to every part of the TensorFlow API would be in favour of performance and usability only.

Timelines

June 1st - June 7th (Week 1)

- Propose changes to tf.image by filing issues
- File issues for discussion regarding tf.image.random_shear and tf.image.random_zca_whiten
- Discuss with mentors of tf.image regarding these changes

June 8th - June 14th (Week 2)

- Write functions for the above two functions random_shear and random_zca_whiten
- Add the necessary documentation
- Write tests to check the working of these functions

June 15th - June 21st (Week 3)

- Write the corresponding classes, RandomShear and RandomZCAWhitening
- Look for changes and incompleteness in other PreprocessingLayer(s)
- Write tests for these classes

June 22nd - June 28th (Week 4)

- Add detailed documentation for the classes
- Publish a PR by linking the above issue
- Start creating the class for ImagePipeline

June 29th - July 5th (Week 5)

- Check compatibility and desired functionality for each Random* layer with ImagePipeline

- Write automated tests to check the working of ImagePipeline class
- Create an issue for general discussion about ImagePipeline with mentors

July 6th - July 12th (Week 6)

- Implement the from_directory function of ImagePipeline class
- Implement from_dataframe function of ImagePipeline class
- Make the ImagePipeline class suitable for tf.data.Dataset

July 13th - July 19th (Week 7)

- Make sure to ensure that the ImagePipeline class and all other Random* layers are run on a CPU graph only
- Ensure that these ops can only be run in forward mode without any weights
- Implement the preview function from the ImagePipeline class

July 20th - July 26th (Week 8)

- Write detailed documentation related to the ImagePipeline class and how they are interoperable with both Layers API as well as tf.data API
- Improvise on the implementations so as to make sure that all tests work.

July 27th - August 2nd (Week 9)

- Write Colab notebooks with tutorials describing how to use the ImagePipeline class with GPU
- Write Colab notebooks with tutorials describing how to use the ImagePipeline class with TPU

August 3rd - August 9th (Week 10)

- Write Colab notebooks with tutorials describing how to use the ImagePipeline class with custom training loops

- Write Colab notebooks with tutorials describing how to use the ImagePipeline class with tf.distribute.Strategy(s)
- Create PR to publish the API changes for ImagePipeline

August 10th - August 16th (Week 11)

- Enact upon the review of the mentor
- Make sure that all code and documentation is complete

August 17th - August 23th (Week 12)

- Enact upon the review of the mentor
- Make sure that all code and documentation is complete

Prior Experience

Recently, I had been able to replicate the model training process mentioned on paper, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification" by Taigman et al. using openly available VGGFace2 dataset and by use of Cloud TPU resources provided by TensorFlow Research Cloud (TFRC) program. Training of machine learning models using million-scale datasets have made me learn the techniques to overcome the challenges that come in the way of large-scale ML training. It has given me the necessary exposure required to build high performance TensorFlow data pipelines suitable for training models on hardware accelerators with super computing capabilities like TPUs. In my quest to train a large-scale triplet-loss based network (preferably to open-source an exact FaceNet implementation) using distributed TFRecords have made me aware about the tf.data API completely.

Over the years, I have found avid interest in open source while replicating proprietary benchmarks into the open source space. My background includes work on applying machine learning techniques to the computer vision domain and interacting with core computational systems to create high-performance cloud native applications. Previously, I have made a few contributions to the TensorFlow repositories and made bug reports and fixes to the other TensorFlow projects. My interests include programming across various paradigms with the aim to develop frameworks with a concrete API design. It would be a pleasure to make more contributions to the TensorFlow ecosystem even outside of the GSoC tenure.

References

1. *RFC - governance/keras-preprocessing-redesign.md · keras-team/governance*
2. *RFC - governance/preprocessing-layers.md · keras-team/governance*
3. *[Public] SIG Keras Meeting Agenda and Notes*
4. *Image Preprocessing - Keras*
5. *tensorflow/image_preprocessing.py at master · tensorflow/tensorflow*
6. *Module: tf.image | TensorFlow Core v2.1.0*
7. *Scaling Tensorflow data processing with tf.data (TF Dev Summit '20)*
8. *tensorflow/preprocessing_stage.py at master · tensorflow/tensorflow*
9. *Learning to read with TensorFlow and Keras (TF Dev Summit '20)*
10. *tensorflow/image_pipeline.py at master · tensorflow/tensorflow*
11. *tf.data: Build TensorFlow input pipelines*
12. *Better performance with the tf.data API*
13. *keras-team/keras-preprocessing: Utilities for working with image data, text data, and sequence data.*
14. *Google Summer of Code (TensorFlow) Ideas List*
15. *NumPy*
16. *SciPy*