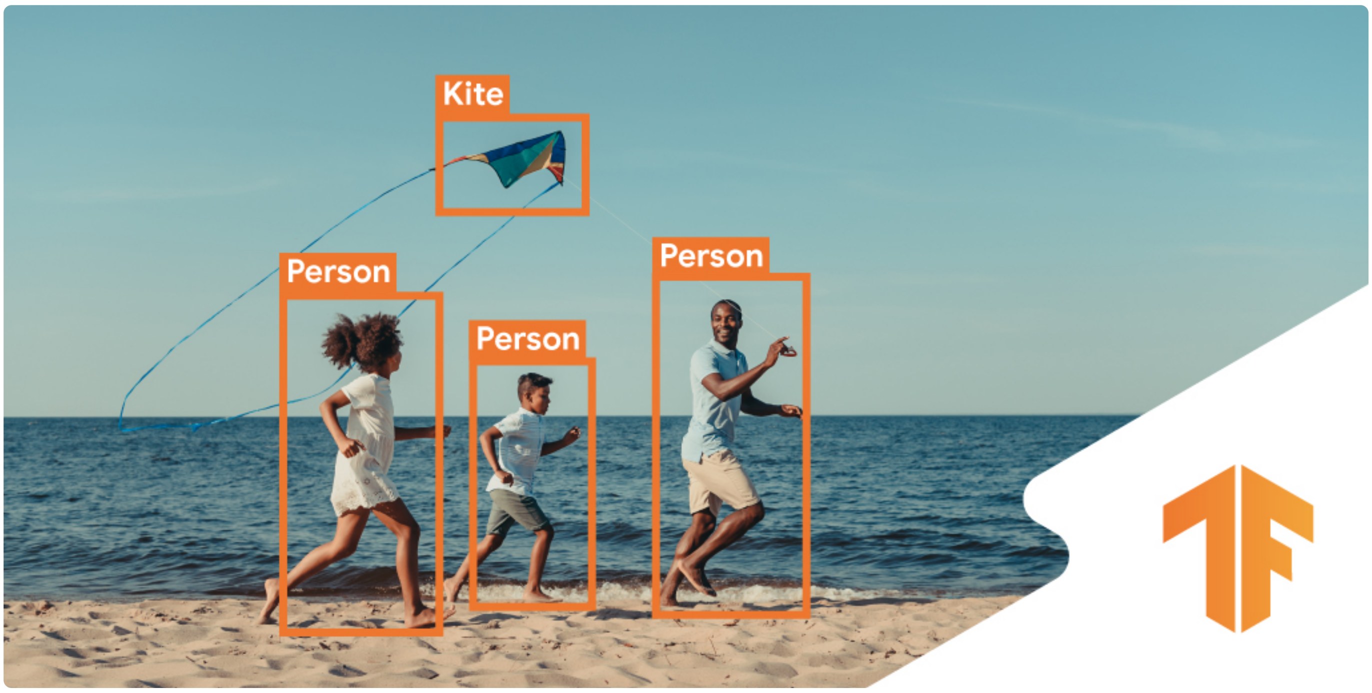
TensorFlow Core

TensorFlow 2 meets the Object Detection API

July 10, 2020

Posted by Vivek Rathod and Jonathan Huang, Google Research



At the TF Dev Summit earlier this year, we mentioned that we are making more of the TF ecosystem compatible so your favorite libraries and models work with TF 2.x. Today we are happy to announce that the TF Object Detection API (OD API) officially supports TensorFlow 2!

Over the last year we've been migrating our TF Object Detection API models to be TensorFlow 2 compatible. If you are a frequent visitor to the Object Detection API GitHub repository, you may have already seen bits and pieces of these new models. Our codebase offers tight Keras

- New binaries for train/eval/export that are eager mode compatible.
- A suite of TF2 compatible (Keras-based) models; this includes migrations of our most popular TF1 models (e.g., SSD with MobileNet, RetinaNet, Faster R-CNN, Mask R-CNN), as well as a few new architectures for which we will only maintain TF2 implementations: (1) CenterNet - a simple and effective anchor-free architecture based on the recent Objects as Points paper by Zhou et al, and (2) EfficientDet --- a recent family of SOTA models discovered with the help of Neural Architecture Search.
- COCO pre-trained weights for all of the models provided as TF2 style object-based checkpoints
- Access to DistributionStrategies for distributed training: traditionally, we have mainly relied on asynchronous training for our TF1 models. We now support synchronous training as the primary strategy; Our TF2 models are designed to be trainable using sync multi-GPU and TPU platforms
- Colab demonstrations of eager mode compatible few-shot training and inference
- First-class support for keypoint estimation, including multi-class estimation, more data augmentation support, better visualizations, and COCO evaluation.

If you'd like to get your feet wet immediately, we recommend checking out our shiny new Colab demos (for inference and few-shot training). As a fun example, we've included a tutorial demonstrating how to train a rubber ducky detector using fine-tuning based few-shot training (with just five example images!).

Our philosophy for this migration was to expose all the benefits of TF2 and Keras, while continuing to support our wide user base still using TF1. We believe that there might be many teams out there grappling with similar migration projects, so we thought that a few words about our thought process and approach here might be useful even for non object-detection TensorFlow users.

Users to our codebase now belong to three categories: (1) New users who want to leverage new features (eager mode training, Distribution Strategies) and new models, (2) Existing TF1 users who want to migrate to TF2, and (3) Existing TF1 users who prefer not to migrate just yet. To support all three categories of users, we have followed a number of strategies detailed below:

tried to ensure that our code is agnostic about whether it is run under TF1 or TF2.

- *Treat feature extractors/backbones as being specific to either TF1 or TF2*. We continue to maintain our TF1 backbones which are implemented in tf-slim, and introduce TF2 backbones implemented in Keras. Then depending on the version of TensorFlow that a user is running, these models will be either enabled or disabled.
- *Leverage community-maintained existing backbone implementations*. Instead of re-implementing backbone architectures (e.g. MobileNet or ResNet) in Keras, our models depend on implementations in the Keras applications collection - a set of community-maintained canned architectures. We have also verified that our new Keras backbones maintain or surpass the accuracy of comparable tf-slim backbones (at least for the models that were already in the OD API).
- *Increase unit test coverage to cover GPU/TPU, TF1 and TF2*. Given that we now need to ensure functionality on multiple platforms (GPU and TPU) as well as across TF versions, we've designed a new and flexible unit testing framework that tests OD API functions under all four settings ({GPU, TPU}x{TF1, TF2}), while allowing for certain tests to be disabled (e.g. input pipelines are not tested on TPU)
- *Separate front-end binaries (training loops, exporters) for TF1 and TF2*. We have added a separate entry point for TF2 models (in the form of new TF2 training and export binaries) which can be run in eager mode, leveraging various DistributionStrategies.
- *No changes to the frontend config language*. In order to make migration from TF1 to TF2 as easy as possible for our users, we've worked hard to ensure that model specifications using OD API's config language produce equivalent model architectures in both TF1 and TF2 and that models can be trained to the same level of numerical performance under both TF versions. As an example, if you have an existing ResNet-50 based RetinaNet model config that is trainable using TF1 binaries, then to train the same model with TF2 binaries, you would simply change the name of the feature extractor in the config (in this case from `ssd_resnet50_v1_fpn` to `ssd_resnet50_v1_fpn_keras`); all other hyperparameter specifications would remain unchanged.

This release is just one example of making the TF ecosystem TF2 compatible and easier to use. Over the next few months, we will continue to migrate large-scale codebases from TF1 to TF2. In addition, we are working to provide a more integrated, end-to-end experience in the TF ecosystem for researchers looking for easy-to-use modeling, starting with a unified computer vision library coming soon.

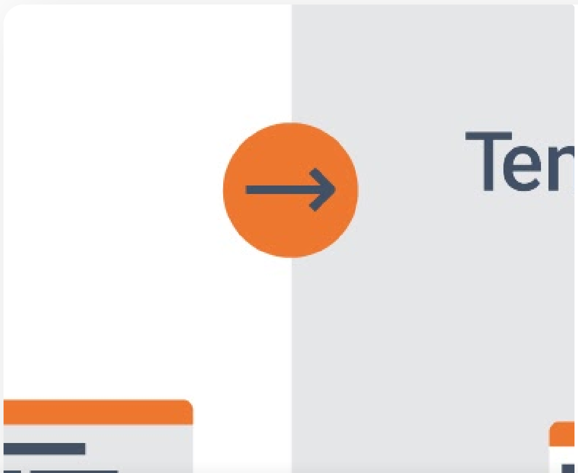
supported in the TF2 pipelines, we encourage you to let us know as this may help us to prioritize as we continue to release features/models.

Acknowledgements

This release is the result of a close collaboration among a number of teams within Google Research. In particular we want to highlight the contributions of the following individuals: first, a special thanks to Tomer Kaftan and Yanhui Liang for initiating this entire effort and doing most of the early heavy lifting. We also thank our main OD API contributors: Vighnesh Birodkar, Ronny Votel, Zhichao Lu, Yu-hui Chen, Sergi Caelles Prat, Jordi Pont-Tuset, Austin Myers. We are also grateful to many other contributors including: Sudheendra Vijayanarasimhan, Sara Beery, Shan Yang, Anjali Sridhar, Karmel Allison, Allen Lavoie, Lu He, Yixin Shi, Derek Chow, David Ross, Pengchong Jin, Jaeyoun Kim, Jing Li, Mingxing Tan, Dan Kondratyuk and Tina Tian. Finally we also thank our interns and summer of code students for their contributions: Kathy Ruan, Kaushik Shivakumar, Yiming Shi, Vishnu Banna, Akhil Chinnakotla, and Anirudh Vegesana.

TensorFlow Core

Next post



TensorFlow Core · TensorFlow Lite

TensorFlow operation fusion in the TensorFlow Lite converter