Training Software Guide for: Planetary-scale Surface Feature Detection and Mapping for Future Exploration Missions

Mark Wronkiewicz
January 18, 2021

1 Overview

This guide provides an overview to the training software for the NASA STTR project entitled: "Planetary-scale Surface Feature Detection and Mapping for Future Exploration Missions". Specifically, this covers the training data format, ML toolboxes, and cloud-based platforms used to train ML models capable of detecting craters in satellite images. It is meant to complement the final report with some extra technical details for users of our model training software.

2 Processing pipeline

The following section outlines the step-by-step processing needed to execute this training pipeline at scale. See 2.1 for a graphical illustration.

2.1 TensorFlow Object Detection API

2.1.1 Overview

The TensorFlow Object Detection API (TF OD API) is a software tool box built to support the training and evaluation of object detection models. These models are typically aimed at identifying and localizing semantic objects within images using convolutional neural networks. The TF OD API allows users to test out many different model architectures and contains tools

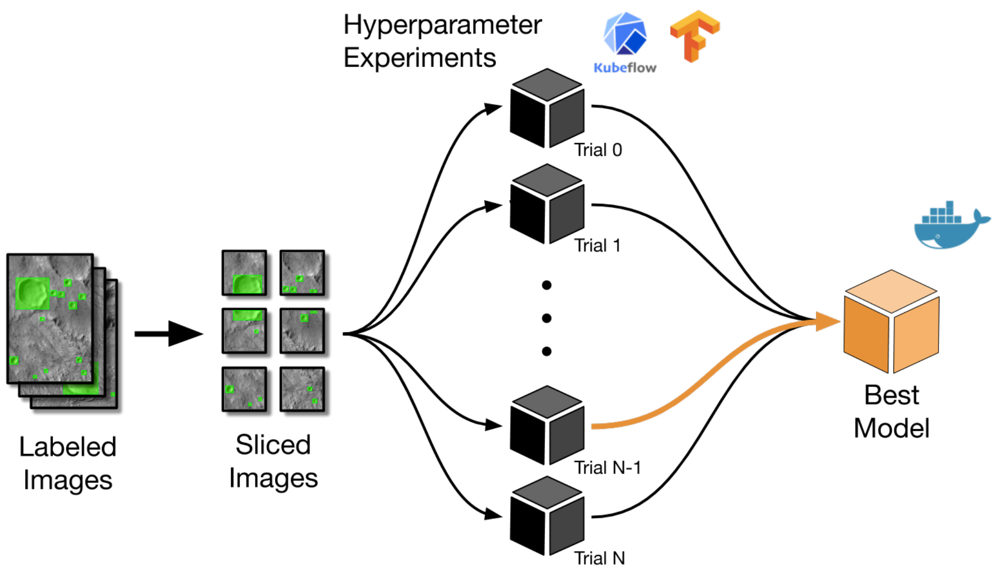


Figure 2.1: Illustration of all steps in the training process. A set of images and expertly annotated craters serve as the data needed to train an object detection model. Using this data, we can run multiple training sessions to test out different hyperparameter combinations. Finally, we automatically select the best performing model and package it into a TF Serving Docker image so that it's stored and easily deployable in the inference pipeline.

for visualizing and debugging during model development. This flexibility and variety of support tools are the primary reasons we chose this library over other options.

2.2 Training Data Format

To use the TF OD API, any training data must be compiled into TFRecords. TFRecord files are a specialized type of data format optimized for training ML models. Compared to standard file formats, they take up less space and disk and can be read more quickly. This makes for a measurable improvement on ML model training time where we want to iterate on training examples as quickly as possible. Unfortunately, these files must be compiled in a special way and aren't easily interrogated manually. For a guide on how to compile data sets for use within TF OD API, see the guide on building a custom data set.

Briefly, each single satellite image has some number (usually tens to hundreds) of labeled craters. The labels are polygon outlines (or "masks") that trace the rim of each crater within the image. We compiled these images, annotations, and other relevant information into TFRecord files before beginning model training. In this project, we accomplish this in the "TFRecorder.py" file, which can be found in the project GitHub repository.

2.3 Mask R-CNN

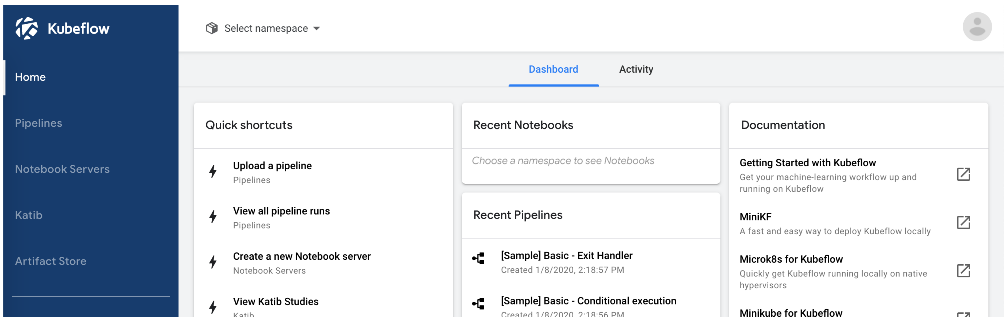
We use the Mask R-CNN model He et al. 2018 here to identify craters within satellite images). The Mask R-CNN is similar to the Faster R-CNN model, which identifies objects by predicting bounding boxes. However, Mask R-CNN has an additional branch in its architecture that predicts a segmentation mask for each object in addition to its bounding box. This means we can get a per-pixel boundary for all predicted objects and not just a rectangular extent. For craters, this is advantageous because we can extract more detailed information about the shape and morphology of craters using this mask. The trade off is that this extra mask-prediction component of the model adds some additional compute time to the training and inference steps.

2.4 Kubeflow Pipelines

2.4.1 Advantages of Kubeflow

Kubeflow is a set of Kubernetes tools mean to support repeatable and scalable ML workflows. Once set up, it can lower the barrier to training new ML models and keep track of past experiments. Kubeflow is built on Kubernetes, so it comes with advantages for running, scaling, and maintaining containerized applications.

We chose to build a system with Kubeflow because we wanted to create a repeatable, stable way train object detection models. Using the same framework, users can train multiple models with different hyperparameters or swap to different data sets (e.g., detect recurring slope lineae instead of craters). This repeatability improves efficiency during the training process because it doesn't require a ML engineer develop a custom pipeline every time a new model is needed. The code is all stored within Docker containers, which makes the code more easily stored and portable because Docker containers contain all the necessary code and libraries needed to run



Quick shortcuts

Upload a pipeline
Pipelines

View all pipeline runs
Pipelines

Create a new Notebook server
Notebook Servers

View Katib Studies

Recent Notebooks

Choose a namespace to see Notebooks

Recent Pipelines

[Sample] Basic - Exit Handler
Created 1/8/2020, 2:18:57 PM

[Sample] Basic - Conditional execution

Documentation

Getting Started with Kubeflow
Get your machine-learning workflow up and running on Kubeflow

MiniKF
A fast and easy way to deploy Kubeflow locally

Microk8s for Kubeflow
Quickly get Kubeflow running locally on native hypervisors

Minikube for Kubeflow

Figure 2.2: Kubeflow UI Landing Page

internally. For our project, this made it simple to develop and test our code locally and then deploy to the cloud. Overall, setting up our Kubeflow system took more initial investment than simply creating a set of Python scripts, but the reusability and scalability of such an approach made it an attractive long term solution.

2.4.2 Running Kubeflow Pipelines

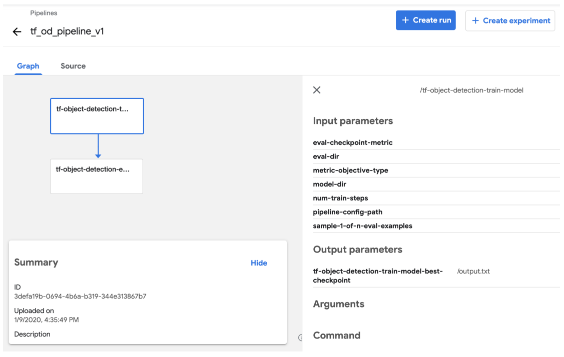
1. **Launch Kubernetes cluster and install Kubeflow**. Set up a Kubernetes cluster to prepare a platform for training ML models. The Kubeflow documentation is exceptionally thorough, so follow the Kubeflow on Google Cloud Platform guide to launch a cluster. Once deployed, it's possible to navigate to the Kubeflow UI shown in 2.2, which provides a web interface for interacting with the cluster.

2. **Upload a Kubeflow Pipeline file**. Upload a pre-built pipeline to the Kubeflow Pipelines interface. These are stored under the "kfpipelines" folder within the GitHub repository. Once uploaded, the pipeline can be investigate as shown in 2.3.

3. **Launch a Kubeflow Pipeline run**. To train a new ML model, select the pipeline and click the "Create Run". Enter the location of your desired data set, training hyperparameters, and model export details. The dialog box will appear similar to that shown in 2.4.

4. **Monitor the training process**. To track the model training run, you can watch either the experiment overview (2.5) or select the pipeline and observe the logs within individual running containers (2.6). The latter is especially useful if you need to track down errors within the running container.

5. **Ensure model files correctly uploaded to GCS**. During training, it is customary to specify a Google Cloud Storage (GCS) location as the model checkpoint directory. This ensures that all model weights are directly backed up to the cloud and available for downstream use. During or after training, it is worth investigating your GCS bucket to make sure they were exported as expected. See 2.7 for an example of what the generated model files should look like.



Summary

ID
3defa19b-0694-4b6a-b319-344e313867b7

Uploaded on
1/9/2020, 4:35:49 PM

Description

/tf-object-detection-train-model

Input parameters

eval-checkpoint-metric
eval-dir
metric-objective-type
model-dir
num-train-steps
pipeline-config-path
sample-1-of-n-eval-examples

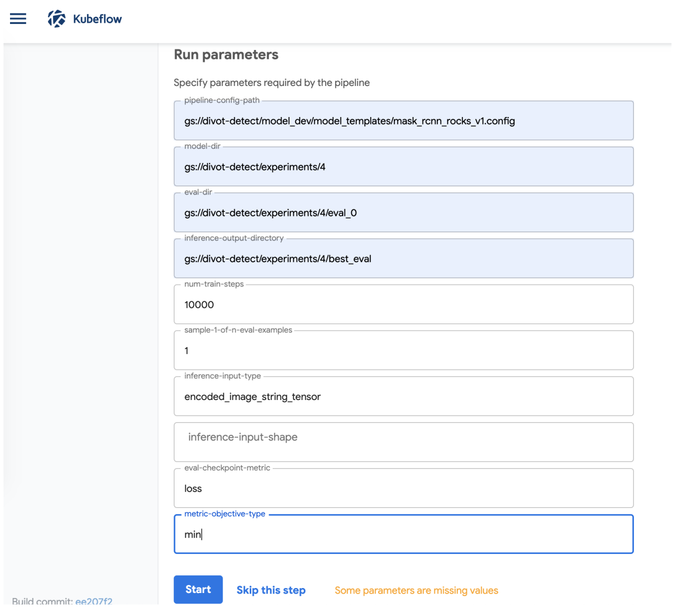
Output parameters

tf-object-detection-train-model-best-checkpoint	/output.txt

Arguments

Command

Figure 2.3: Example Kubeflow Pipeline

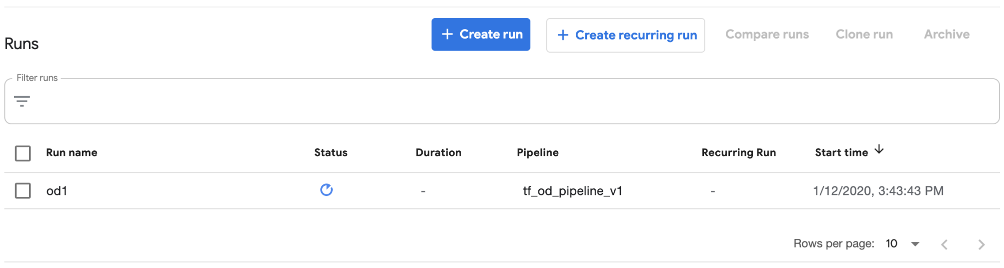


Run parameters

Specify parameters required by the pipeline

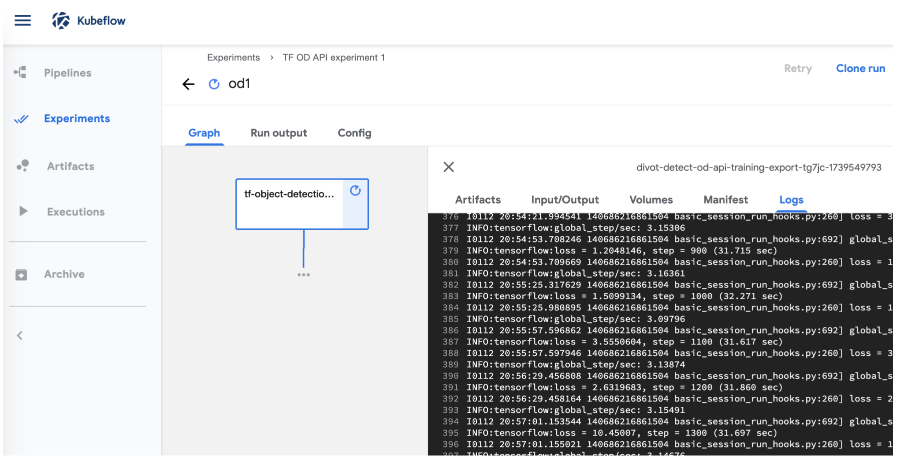
pipeline-config-path	gs://divot-detect/model_dev/model_templates/mask_rcnn_rocks_v1.config
model-dir	gs://divot-detect/experiments/4
eval-dir	gs://divot-detect/experiments/4/eval_0
inference-output-directory	gs://divot-detect/experiments/4/best_eval
num-train-steps	10000
sample-1-of-n-eval-examples	1
inference-input-type	encoded_image_string_tensor
inference-input-shape	
eval-checkpoint-metric	loss
metric-objective-type	min

Figure 2.4: Example Kubeflow Pipeline Parameters



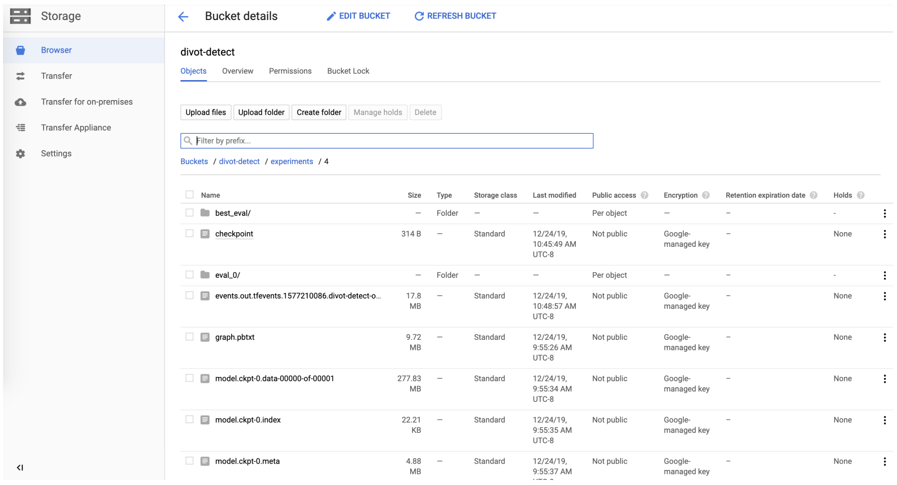
Run name	Status	Duration	Pipeline	Recurring Run	Start time
od1		-	tf_od_pipeline_v1	-	1/12/2020, 3:43:43 PM

Figure 2.5: Example of Running Kubeflow Pipeline



```
I0112 20:54:21.994541 140686216861504 basic_session_run_hooks.py:260] loss = 3
INFO:tensorflow:global_step/sec: 3.15306
I0112 20:54:53.708246 140686216861504 basic_session_run_hooks.py:692] global_s
INFO:tensorflow:loss = 1.2048146, step = 900 (31.715 sec)
I0112 20:54:53.709669 140686216861504 basic_session_run_hooks.py:260] loss = 1
INFO:tensorflow:global_step/sec: 3.16361
I0112 20:55:25.317629 140686216861504 basic_session_run_hooks.py:692] global_s
INFO:tensorflow:loss = 1.5099134, step = 1000 (32.271 sec)
I0112 20:55:25.980895 140686216861504 basic_session_run_hooks.py:260] loss = 1
INFO:tensorflow:global_step/sec: 3.09796
I0112 20:55:57.596862 140686216861504 basic_session_run_hooks.py:692] global_s
INFO:tensorflow:loss = 3.5550604, step = 1100 (31.617 sec)
I0112 20:55:57.597946 140686216861504 basic_session_run_hooks.py:260] loss = 3
INFO:tensorflow:global_step/sec: 3.13874
I0112 20:56:29.456808 140686216861504 basic_session_run_hooks.py:692] global_s
INFO:tensorflow:loss = 2.6319683, step = 1200 (31.860 sec)
I0112 20:56:29.458164 140686216861504 basic_session_run_hooks.py:260] loss = 2
INFO:tensorflow:global_step/sec: 3.15491
I0112 20:57:01.153544 140686216861504 basic_session_run_hooks.py:692] global_s
INFO:tensorflow:loss = 10.45007, step = 1300 (31.697 sec)
I0112 20:57:01.155021 140686216861504 basic_session_run_hooks.py:260] loss = 1
INFO:tensorflow:global_step/sec: 3.14676
```

Figure 2.6: Example Logs from Running Kubeflow Pipeline.



Name	Size	Type	Storage class	Last modified	Public access	Encryption	Retention expiration date	Holds
best_eval/	–	Folder	–	–	Per object	–	–	-
checkpoint	314 B	–	Standard	12/24/19, 10:45:49 AM UTC-8	Not public	Google-managed key	–	None
eval_0/	–	Folder	–	–	Per object	–	–	-
events.out.tfevents.1577210086.divot-detect-o...	17.8 MB	–	Standard	12/24/19, 10:48:57 AM UTC-8	Not public	Google-managed key	–	None
graph.pbtxt	9.72 MB	–	Standard	12/24/19, 9:55:26 AM UTC-8	Not public	Google-managed key	–	None
model.ckpt-0.data-00000-of-00001	277.83 MB	–	Standard	12/24/19, 9:55:34 AM UTC-8	Not public	Google-managed key	–	None
model.ckpt-0.index	22.21 KB	–	Standard	12/24/19, 9:55:35 AM UTC-8	Not public	Google-managed key	–	None
model.ckpt-0.meta	4.88 MB	–	Standard	12/24/19, 9:55:37 AM	Not public	Google-managed key	–	None

Figure 2.7: Files generated during ML training experiment.

Note that the Kubeflow Pipeline strategy described here is most appropriate for large scale or repeated ML training runs. There is more monitoring and debugging to do than simply running the TF OD API training from the command line. Therefore, for one off jobs, or research settings, it is likely strongly recommended to consider how much effort and expertise is available to get this more advanced strategy to work.

2.5 TensorFlow Serving

After we trained a Mask R-CNN model to sufficient performance, we packaged up the final model in a TF Serving Docker image. TF Serving is a tool built to support inference in production environments. It allows a ML model to be wrapped in a Docker image that supports a server interface. Creating a Docker image means that the model can be executed in multiple environments with all its libraries and supporting software pre-bundled together. The server interface brings with it a consistent way to request predictions. Here, we used specifically used the RESTful API, which provides a web interface for sending images to the model via a POST request and then receiving predictions as JSON responses.

The inference software guide provides more details on how to create a pool of multiple GPU instance each initialized with one of these TF Serving Docker images.

References

[He+18] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].