This tutorial shows you how to train the Inception (https://github.com/tensorflow/tpu/tree/master/models/experimental/inception) model on Cloud TPU.

This tutorial uses a third-party dataset. Google provides no representation, warranty, or other guarantees about the validity, or any other aspects of, this dataset.

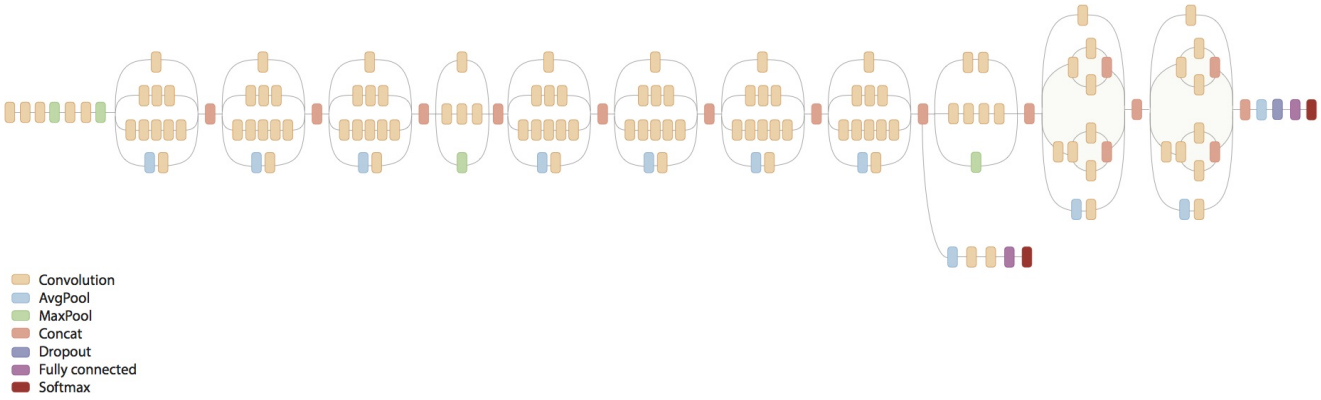
Inception v3 is a widely-used image recognition model that can attain significant accuracy. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.

The model has a mixture of symmetric and asymmetric building blocks, including:

- convolutions
- average pooling
- max pooling
- concats
- dropouts
- fully connected layers

Loss is computed via Softmax.

The following picture shows the model at a high level:



You can find more information about the model at GitHub (https://github.com/tensorflow/tpu/tree/master/models/experimental/inception).

The model is built using the high-level Estimator API (https://www.tensorflow.org/get_started/estimator).

This API greatly simplifies model creation by encapsulating most low-level functions, allowing users to focus on model development, not the inner workings of the underlying hardware that runs things.

Before starting this tutorial, check that your Google Cloud project is correctly set up.

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).

This walkthrough uses billable components of Google Cloud. Check the Cloud TPU pricing page (/tpu/docs/pricing) to estimate your costs. Be sure to clean up (#clean_up) resources you create when you've finished with them to avoid unnecessary charges.

This section provides information on setting up Cloud Storage storage, VM, and Cloud TPU resources for tutorials.

ant: Set up all resources in the same region/zone to reduce network latency and network costs.

You need a Cloud Storage bucket to store the data you use to train your model and the training results. The `ctpu up` tool used in this tutorial sets up default permissions for the Cloud TPU service account. If you want finer-grain permissions, review the access level permissions (/tpu/docs/storage-buckets).

The bucket location must be in the same region as your virtual machine (VM) and your TPU node. VMs and TPU nodes are located in specific zones (/tpu/docs/types-zones#types), which are subdivisions within a region.

1. Go to the Cloud Storage page on the Cloud Console.

   Go to the Cloud Storage page (https://console.cloud.google.com/storage/browser)

2. Create a new bucket, specifying the following options:

   - A unique name of your choosing.
   - Select `Region` for Location type and `us-central1` for the Location (zone)
   - Default storage class: `Standard`
   - Location: Specify a bucket location in the same region where you plan to create your TPU node. See TPU types and zones (/tpu/docs/types-zones#types) to learn where various TPU types are available.

This section demonstrates using the Cloud TPU provisioning tool (https://github.com/tensorflow/tpu/tree/master/tools/ctpu) (`ctpu`) for creating and managing Cloud TPU project resources. The resources are comprised of a virtual machine (VM) and a Cloud TPU resource that have the same name. **These resources must reside in the same region/zone as the bucket you just created.**

You can also set up your VM and TPU resources using `gcloud` commands or through the Cloud Console (https://console.cloud.google.com/). See the creating and deleting TPUs (/tpu/docs/creating-deleting-tpus) page to learn all the ways you can set up and manage your Compute Engine VM and Cloud TPU resources.

1. Open a Cloud Shell window.

   Open Cloud Shell (https://console.cloud.google.com/?cloudshell=true)

2. Run `gcloud config set project <var>your-project</var>` to set the project where you want to create Cloud TPU.
3. Run `ctpu up` specifying the flags shown for either a Cloud TPU device or Pod slice. If you do not specify `tpu-size`, the default is a v2-8 Cloud TPU. Refer to CTPU Reference (/tpu/docs/ctpu-reference) for flag options and descriptions.
4. Set up a Cloud TPU device:

★ **Note:** If you have more than one project, you must specify the project name. If `--name` is not specified, it defaults to your username. If `--zone` is not specified, it defaults to `us-central1-b`. Make sure the zone matches the zone where you set up the storage bucket.

5. The configuration you specified appears. Enter y to approve or n to cancel.
6. When the `ctpu up` command has finished executing, verify that your shell prompt has changed from `username@project` to `username@tpuname`. This change shows that you are now logged into your Compute Engine VM.

★ **Note:** If you are not connected to the Compute Engine instance, you can connect by running the following commands, replacing ***vm-name*** with the name of your VM:

As you continue these instructions, run each command that begins with `(vm)$` in your VM session window.

Set up the following environment variable, replacing `YOUR-BUCKET-NAME` with the name of your Cloud Storage bucket:

The training application expects your training data to be accessible in Cloud Storage. The training application also uses your Cloud Storage bucket to store checkpoints during training.

ImageNet is an image database. The images in the database are organized into a hierarchy, with each node of the hierarchy depicted by hundreds and thousands of images.

This tutorial uses a demonstration version of the full ImageNet dataset, referred to as the *fake_imagenet* dataset. This demonstration version allows you to test out the tutorial, without requiring the storage or time that required to download and run a model against the full ImageNet database. Below are the instructions for using the randomly generated fake_imagenet dataset to test the model. Alternatively, you can use the full ImageNet dataset (#full-dataset).

A `DATA_DIR` environment variable described below (#run-model) is used to specify which dataset to train on.

Note that the fake_imagenet dataset is only useful for understanding how to use a Cloud TPU, and validating end-to-end performance. The accuracy numbers and saved model will not be meaningful.

The fake_imagenet dataset is at this location on Cloud Storage:

For this tutorial, make sure you **don't** set the **`STORAGE_BUCKET`** environment variable to the path of the fake_imagene
et. You can read from **`gs://cloud-tpu-test-datasets`** but you can't write to it. As a result, you can't use it to writ
g logs. Make sure the **`STORAGE_BUCKET`** environment variable is set to your own Cloud Storage bucket, as shown ab

TensorBoard offers a suite of tools (/tpu/docs/cloud-tpu-tools) designed to present TensorFlow data visually. When used for monitoring, TensorBoard can help identify bottlenecks in processing and suggest ways to improve performance.

If you don't need to monitor the model's output at this time, you can skip the TensorBoard setup steps.

If you want to monitor the model's output and performance, follow the guide to setting up TensorBoard (/tpu/docs/tensorboard-setup).

You are now ready to train and evaluate the Inception v3 model using ImageNet data.

The Inception v3 model is pre-installed on your Compute Engine VM, in the `/usr/share/tpu/models/experimental/inception/` directory.

In the following steps, a prefix of `(vm)$` means you should run the command on your Compute Engine VM:

1. Set up a `DATA_DIR` environment variable containing one of the following values:
    - If you are using the fake_imagenet dataset:

- If you have uploaded a set of training data to your Cloud Storage bucket:

2. Run the Inception v3 model:

- `--tpu` specifies the name of the Cloud TPU. Note that `ctpu` passes this name to the Compute Engine VM as an environment variable (`TPU_NAME`).
- `--use_data` specifies which type of data the program must use during training, either fake or real. The default value is fake.
- `--data_dir` specifies the Cloud Storage path for training input. The application ignores this parameter when you're using fake_imagenet data.
- `--model_dir` specifies the directory where checkpoints and summaries are stored during model training. If the folder is missing, the program creates one. When using a Cloud TPU, the `model_dir` must be a Cloud Storage path (`gs://...`). You can reuse an existing folder to load current checkpoint data and to store additional checkpoints as long as the previous checkpoints were created using TPU of the same size and Tensorflow version.

Inception v3 operates on 299x299 images. The default training batchsize is 1024, which means that each iteration operates on 1024 of those images.

You can use the `--mode` flag to select one of three modes of operation: train, eval, and train_and_eval:

- `--mode=train` or `--mode=eval` specifies either a training-only or an evaluation-only job.

- `--mode=train_and_eval` specifies a hybrid job that does both training and evaluation.

Train-only jobs run for the specified number of steps defined in `train_steps` and can go through the entire training set, if desired.

Train_and_eval jobs cycle though training and evaluation segments. Each training cycle runs for `train_steps_per_eval` and is followed by an evaluation job (using the weights that have been trained up to that point).

The number of training cycles is defined by the floor (https://en.wikipedia.org/wiki/Floor_and_ceiling_functions) function of `train_steps` divided by `train_steps_per_eval`.

The `train_steps_per_eval` flag must be a multiple of `iterations`.

By default, Estimator API-based models report loss values every certain number of steps. The reporting format is along the lines of:

The specific modifications required to get Estimator API-based models ready for TPUs are surprisingly minimal. The program imports the following libraries:

The CrossShardOptimizer function wraps the optimizer, as in:

The function that defines the model returns an Estimator specification using:

The main function defines an Estimator-compatible configuration using:

The program uses this defined configuration and a model definition function to create an Estimator object:

Train-only jobs need only to call the train function:

Evaluation-only jobs get their data from available checkpoints and wait until a new one becomes available:

When you choose the option `train_and_eval`, the training and the evaluation jobs run in parallel. During evaluation, trainable variables are loaded from the latest available checkpoint. Training and evaluation cycles repeat as you specify in the flags::

If you used the fake_imagenet dataset to train the model, proceed to clean up (#clean-up).

You need about 300GB of space available on your local machine or VM to use the full ImageNet dataset.

If you use `ctpu up` to set up your VM, it will allocate 250GB by default.

You can increase the size of the VM disk using one of the following methods:

- Specify the `--disk-size-gb` flag on the `ctpu up` command line with the size, in GB, that you want allocated.

- Follow the Compute Engine guide to add a disk (/compute/docs/disks/add-persistent-disk) to your VM.
    - Set **When deleting instance** to **Delete disk** to ensure that the disk is removed when you remove the VM.
    - Make a note of the path to your new disk. For example: `/mnt/disks/mnt-dir`.

For the following commands, a prefix of (vm) means you should run the command on the Compute Engine VM instar
mand does not have the (vm) prefix, run it on your local workstation.

1. Sign up for an ImageNet account (http://image-net.org/signup). Remember the username and password you used to create the account.
2. Set up a `DATA_DIR` environment variable pointing to a path on your Cloud Storage bucket:
3. Download the `imagenet_to_gcs.py` script from GitHub:
4. Set a `SCRATCH_DIR` variable to contain the script's working files. The variable must specify a location on your local machine or on your Compute Engine VM. For example, on your local machine:

   Or if you're processing the data on the VM:

5. Run the `imagenet_to_gcs.py` script to download, format, and upload the ImageNet data to the bucket. Replace `[USERNAME]` and `[PASSWORD]` with the username and password you used to create your ImageNet account.

Optionally if the raw data, in JPEG format, has already been downloaded, you can provide a direct `raw_data_directory` path. If a raw data directory for training or validation data is provided, it should be in the format:

- Training images (https://github.com/awslabs/deeplearning-benchmark/blob/master/tensorflow/inception/inception/data/build_imagenet_data.py) : train/n03062245/n03062245_4620.JPEG
- Validation images (https://github.com/tensorflow/models/blob/master/research/inception/inception/data/preprocess_imagenet_validation_data.py) : validation/ILSVRC2012_val_00000001.JPEG
- Validation labels (http://data.dmlc.ml/mxnet/models/imagenet/synset.txt): synset_labels.txt

The training subdirectory names (for example, n03062245) are "WordNet IDs" (wnid). The ImageNet API (http://www.image-net.org/download-API) shows the mapping of WordNet IDs to their associated validation labels in the `synset_labels.txt` file. A synset in this context is a visually-similar group of images.

Note: Downloading and preprocessing the data can take 10 or more hours, depending on your network and computer speed. Do not interrupt the script.

When the script finishes processing, a message like the following appears:

The script produces a series of directories (for both training and validation) of the form:

and

After the data has been uploaded to your Cloud bucket, run your model and set `--data_dir=${DATA_DIR}`.

To avoid incurring charges to your GCP account for the resources used in this topic:

1. Disconnect from the Compute Engine VM:

   Your prompt should now be `user@projectname`, showing you are in the Cloud Shell.

2. In your Cloud Shell, run `ctpu delete` with the –zone flag you used when you set up the Cloud TPU to delete your Compute Engine VM and your Cloud TPU:

★ **Important:** If you set the TPU resources name when you ran `ctpu up`, you must specify that name with the `--name` flag when you run `ctpu delete` in order to shut down your TPU resources.

3. Run `ctpu status` to make sure you have no instances allocated to avoid unnecessary charges for TPU usage. The deletion might take several minutes. A response like the one below indicates there are no more allocated instances:

4. Run `gsutil` as shown, replacing ***bucket-name*** with the name of the Cloud Storage bucket you created for this tutorial:

For free storage limits and other pricing information, see the Cloud Storage pricing guide (/storage/pricing).

The Inception v4 model is a deep neural network model that uses Inception v3 building blocks to achieve higher accuracy than Inception v3. It is described in the paper "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" by Szegedy et. al.

The Inception v4 model is pre-installed on your Compute Engine VM, in the `/usr/share/tpu/models/experimental/inception/` directory.

In the following steps, a prefix of `(vm)$` means you should run the command on your Compute Engine VM:

1. If you have TensorBoard running in your Cloud Shell tab, you need another tab to work in. Open another tab in your Cloud Shell, and use `ctpu` in the new shell to connect to your Compute Engine VM:

2. Set up a `DATA_DIR` environment variable containing one of the following values:

- If you are using the fake_imagenet dataset:

- If you have uploaded a set of training data to your Cloud Storage bucket:

3. Run the Inception v4 model:

- `--tpu` specifies the name of the Cloud TPU. Note that `ctpu` passes this name to the Compute Engine VM as an environment variable (`TPU_NAME`).
- `--use_data` specifies which type of data the program must use during training, either fake or real. The default value is fake.
- `--train_batch_size` specifies the train batch size to be 256. As the Inception v4 model is larger than Inception v3, it must be run at a smaller batch size per TPU core.
- `--data_dir` specifies the Cloud Storage path for training input. The application ignores this parameter when you're using fake_imagenet data.

- `--model_dir` specifies the directory where checkpoints and summaries are stored during model training. If the folder is missing, the program creates one. When using a Cloud TPU, the `model_dir` must be a Cloud Storage path (`gs://...`). You can reuse an existing folder to load current checkpoint data and to store additional checkpoints as long as the previous checkpoints were created using TPU of the same size and Tensorflow version.

- Go in depth with an advanced view (/tpu/docs/inception-v3-advanced) of Inception v3 on Cloud TPU.
- Learn more about `ctpu` (https://github.com/tensorflow/tpu/tree/master/tools/ctpu), including how to install it on a local machine.
- Explore the TPU tools in TensorBoard (/tpu/docs/cloud-tpu-tools).