Counting Sheep with Drones and AI

Abstract: This whitepaper describes the steps taken to install Tensorflow and an Object Detection model to create a machine learning engine to count sheep from a DJI drone’s video feed on an Android phone.

Prepared by:
RIIS LLC
1250 Stephenson Hwy, Suite 200
Troy, MI 48083
Contact:
Godfrey Nolan
(248) 286 1227
# Table of Contents

**The Challenge**

3

**The Solution**

3

- Step 1: Prepare the dataset ........................................................................... 3
- Step 2: Set up Google Cloud Account ............................................................. 4
- Step 3: Set up your Docker environment ......................................................... 4
- Step 4: Configure your local Google Cloud environment .................................. 5
- Step 5: Set up Object Detection API ............................................................... 6
- Step 6: Train your model .............................................................................. 8
- Step 7: Evaluate your Model ....................................................................... 11
- Step 8: Export your model to Android with TensorFlow Lite ......................... 12
- Step 9: Running on Android With a DJI Drone ............................................ 14
The Challenge

The challenge was defined as follows:

1. Create a mobile app that uses the DJI Mobile SDK to fly in an automated fashion in a field of sheep.
2. Create a machine learning algorithm using Tensorflow that will do image detection on the drone’s video feed to detect and count the sheep.
3. The learning can be done offline, but the detection should be real time.
4. Because this is a rural connection it’s likely that there will be no connection to any cloud services.

The Solution

This is a companion whitepaper to the following presentation [https://www.slideshare.net/godfreynolan/counting-sheep-with-drones-and-ai](https://www.slideshare.net/godfreynolan/counting-sheep-with-drones-and-ai). We describe the steps and commands needed to train Tensorflow with the labeled images and then upload the trained engine onto an Android phone.

Step 1: Prepare the dataset

1. Gather images
2. Label the images with LabellImg found at [https://github.com/tzutalin/labellimg](https://github.com/tzutalin/labellimg)
4. Convert the CSV files into your TFRecord Dataset using script found at [https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py](https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py)
5. Create your label map file, see Figure 1.

```python
    1   item {
    2     id: 1
    3         name: 'sheep'
    4   }
```

_Figure 1. Sheep Label Map_
Step 2: Set up Google Cloud Account

1. Sign up for a Google Cloud Account
2. Login and set up a project named CountingSheep
3. Enable ML Engine for your project at
4. Click “Storage” on the side bar, see Figure 2, and then create a new bucket called CountingSheepData
5. Create a sub-directory called “data” in your storage bucket.

Figure 2. Google Cloud

Step 3: Set up your Docker environment

Setting up your environment with Docker.

1. Ensure Docker is downloaded on your machine, see docker.io for more information
2. Download the Dockerfile at
3. Build the docker image with the following command:
   
   docker build .
4. View the current docker images with the following command:
   
   `docker images`

5. Copy the Image ID, then run the image with the following command:

   `docker run -it IMAGE_ID_HERE`

6. You should now be in running your docker image locally in your terminal, see Figure 3.

   ![Figure 3. Docker terminal](image)

---

**Step 4: Configure your local Google Cloud environment**

Run the following commands to set up your google cloud project locally on your docker container:

1. Login with the following command:

   `gcloud auth login`

2. Export Google Cloud project/bucket names to environment variables by executing:

   `export PROJECT="YOUR_PROJECT_ID"`
   
   `export YOUR_GCS_BUCKET="YOUR_UNIQUE_BUCKET_NAME"`

3. Set your current Google Cloud project and storage bucket with the following commands (Make sure you use the Project ID displayed on Google Cloud Console, see Figure 4):
gcloud config set project $PROJECT

gsutil mb gs://$YOUR_GCS_BUCKET

4. Set up Cloud TPU access for the project. Get the name of your service account with the following command:


5. Extract the service account from the tpuServiceAccount value from the response of the previous command and export the TPU service account to an environment variable with the following command:

   export TPU_ACCOUNT=your-service-account

6. Grant ML service agent to TPU service account:

   gcloud projects add-iam-policy-binding $PROJECT \\ --member serviceAccount:$TPU_ACCOUNT --role roles/ml.serviceAgent

---

**Figure 4. Google Cloud Project ID**

**Step 5: Set up Object Detection API**

1. Make sure your local copy of the object detection API works with following commands:

   cd /tensorflow/models/research

   python object_detection/builders/model_builder_test.py

2. It should respond with the following, see Figure 5:
Figure 5. Object Detection tests working correctly

3. Move your TFRecord files and label map from your host machine into your docker image with the docker cp tool:

```bash
docker cp train.record CONTAINERNAME:/train.record
docker cp test.record CONTAINERNAME:/test.record
docker cp sheep_label_map.pbtxt CONTAINERNAME:/sheep_label_map.pbtxt
```

4. Copy the data over to your bucket (from your docker container) with gsutil:

```bash
gsutil -m cp -r /train.record gs://${YOUR_GCS_BUCKET}/data/
gsutil -m cp -r /test.record gs://${YOUR_GCS_BUCKET}/data/
gsutil cp /sheep_label_map.pbtxt gs://${YOUR_GCS_BUCKET}/data/
```

5. Download the object detection model and upload it to Google cloud with the following commands:

```bash
mkdir /tmp && cd /tmp

curl -O http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_0.75_depth_300x300_coco14_sync_2018_07_03.tar.gz
tar xzf ssd_mobilenet_v1_0.75_depth_300x300_coco14_sync_2018_07_03.tar.gz
```
gsutil cp
/tmp/ssd_mobilenet_v1_0.75_depth_300x300_coco14_sync_2018_07_03/model.ckpt.*
gs://$YOUR_GCS_BUCKET)/data/

6. Edit the object detection configuration using your preferred editor:

   vim
/tensorflow/models/research/object_detection/samples/configs/ssd_mobilenet_v1_0.75_depth_quantized_300x300_pets_sync.config

7. Update the following in the config file, see Table 1:

<table>
<thead>
<tr>
<th>Original Value (All instances)</th>
<th>New Value (All instances)</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes: 37</td>
<td>num_classes: 1</td>
</tr>
<tr>
<td>PATH TO BE CONFIGURED</td>
<td>gs://YOUR_GCS_BUCKET/data/</td>
</tr>
<tr>
<td>pet_faces_train.record-?????-of-00010</td>
<td>train.record</td>
</tr>
<tr>
<td>pet_faces_val.record-?????-of-00010</td>
<td>test.record</td>
</tr>
<tr>
<td>pet_label_map.pbtxt</td>
<td>sheep_label_map.pbtxt</td>
</tr>
</tbody>
</table>

   Table 1. Object Detection configuration

8. Upload the new file to Google Cloud bucket with the following command:

   gsutil cp
/tensorflow/models/research/object_detection/samples/configs/ssd_mobilenet_v1_0.75_depth_quantized_300x300_pets_sync.config
gs://$YOUR_GCS_BUCKET)/data/pipeline.config

9. Package the object detection API with the following commands:

   bash object_detection/dataset_tools/create_pycocotools_package.sh
/tmp/pycocotools

   python setup.py sdist

**Step 6: Train your model**

1. Start your training job on Google Cloud TPU with the following command:

   cd /tensorflow/models/research
gcloud ml-engine jobs submit training `whoami`_object_detection_`date +%s` \
   --job-dir=gs://$YOUR_GCS_BUCKET)/train \
   \
   --tpu-zone=us-central1-a
   \
   --tpu-program=script
   \
   --tpu-type=tpu-v3
   \
   --tpu-attached=1
   \
   --tpu-count=1
   \
   --tpu-program-args="--model=efficientnet
   --job-dir=gs://$YOUR_GCS_BUCKET)/train
   --tpu-attached=1
   --tpu-type=tpu-v3
   --tpu-count=1
   --tpu-program-args="--model=efficientnet
   --job-dir=gs://$YOUR_GCS_BUCKET)/train
   \
   &

   bash object_detection/dataset_tools/create_pycocotools_package.sh
/tmp/pycocotools

   python setup.py sdist
Next, start your evaluation job on Google Cloud GPU with the following command:

```
gcloud ml-engine jobs submit training \
'whoami'_'object_detection_eval_validation_'date +%s' \
--job-dir=gs://${YOUR_GCS_BUCKET}/train \
--packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz,/tmp/pycocotools/pycocotools-2.0.tar.gz \
--module-name object_detection.model_main \
--runtime-version 1.9 \
--scale-tier BASIC_GPU \
--region us-central1 \
--model_dir=gs://${YOUR_GCS_BUCKET}/train \
--pipeline_config_path=gs://${YOUR_GCS_BUCKET}/data/pipeline.config \
--checkpoint_dir=gs://${YOUR_GCS_BUCKET}/train
```

3. Open up the Google Cloud console in your browser and click the ML Engine category (see Figure 6):
4. You should be able to see that your training job and evaluation job are both running. The run
time is dependent on the size of your dataset. For our use case, both jobs should be done within
30 minutes (see Figure 7).

5. Ensure that both jobs were successful upon completion with the green indicator (see Figure 8).
Step 7: Evaluate your Model

If you would like to view some metrics of your model after training has completed, you can view them in TensorBoard. TensorBoard is browser based, so this must be done on your host machine and not in your docker container. You must have TensorFlow installed on your host machine to use TensorBoard. You must also have Google Cloud CLI installed.

1. Install Python on your machine (can be downloaded at https://www.python.org/downloads/) and use the following command to install TensorFlow:
   
   ```bash
   pip install tensorflow
   ```

2. You can download and install Google Cloud CLI at https://cloud.google.com/sdk/docs/quickstarts. Login to your Google Cloud account with the following command:
   
   ```bash
   gcloud auth application-default login
   ```

3. Start a TensorBoard from your GCS bucket’s train directory:
   
   ```bash
   tensorboard --logdir=gs://YOUR_GCS_BUCKET/train
   ```


5. In the ‘Scalars’ section you can view that the loss decreased during training (this indicates learning), see Figure 9).

Figure 8. Completed jobs on ML Engine Console
Figure 9. Tensorboard Scalars

6. Click on the 'Images' Section to view an example image from your testing. The left image shows the ground truth example (see Figure 10). It obviously isn't perfect, but it seems to work reasonably well.

Figure 10. Tensorboard Images

**Step 8: Export your model to Android with TensorFlow Lite**

1. Export some values to environment variables using the following commands:

   ```bash
   export CONFIG_FILE=gs://${YOUR_GCS_BUCKET}/data/pipeline.config
   export CHECKPOINT_PATH=gs://${YOUR_GCS_BUCKET}/train/model.ckpt-2000
   export OUTPUT_DIR=/tmp/tflite
   ```
2. Before we export, we need to make a small change to the export script to increase the number of boxes we can draw on the scene at one time. To do so, run:

```
vim object_detection/export_tflite_ssd_graph.py
```

Modify line 106, changing the 10 to 100:

```
flags.DEFINE_integer('max_detections', 10, 'Maximum number of detections (boxes) to show. ')
```

3. Export the frozen inference graph with the following command:

```
python object_detection/export_tflite_ssd_graph.py \
  --pipeline_config_path=$CONFIG_FILE \
  --trained_checkpoint_prefix=$CHECKPOINT_PATH \
  --output_directory=$OUTPUT_DIR \
  --add_postprocessing_op=true \
  --max_detections=100
```

4. We now have the files “tflite_graph.pb” and “tflite_graph.pbfstxt” in our /tmp/tflite folder. Now, we will get the optimized model using the TensorFlow Lite Optimizing Converter (TOCO), see Figure 11 for more information on TOCO. This will convert the resulting frozen graph (tflite_graph.pb) to the TensorFlow Lite flatbuffer format (detect.tflite). This will take several minutes. Execute the following commands:

```
cd /tensorflow/
bazel run -c opt tensorflow/contrib/lite/toco:toco -- \
  --input_file=$OUTPUT_DIR/tflite_graph.pb \
  --output_file=$OUTPUT_DIR/detect.tflite \
  --input_shapes=1,300,300,3 \
  --input_arrays=normalized_input_image_tensor \
  --inference_type=QUANTIZED_UINT8 \
  --mean_values=128 \
  --std_values=128 \
  --change_concat_input_ranges=false \
```
Where the converter fits in the TensorFlow landscape

Once an application developer has a trained TensorFlow model, TOCO will accept that model and generate a TensorFlow Lite FlatBuffer file. TOCO currently supports SavedModels, frozen graphs (models generated via freeze_graph.py), and tf.Keras model files. The TensorFlow Lite FlatBuffer file can be shipped to client devices, generally mobile devices, where the TensorFlow Lite interpreter handles them on-device. This flow is represented in the diagram below.

**Figure 11.** TensorFlow Lite Optimizing Converter

5. Our new TFLite file will be placed in the /tmp/tflite directory called detect.tflite. This file contains the graph and all model parameters and can be run via the TensorFlow Lite interpreter on the Android device and should be less than 4 Mb in size.

Step 9: Running on Android with a DJI Drone

1. You’re going to need a DJI developer account to use the drone’s SDK. You can make one at [https://developer.dji.com](https://developer.dji.com)

2. Once you’ve made an account, create a new app from the Developer Center. Give the app a name and an android package name (see Figure 12). Make sure you remember this package name. Once created, DJI will ask you to confirm creation of the app via email. Once you’ve done so, move on to the next step.
3. Now we can run drone camera frames through our model on an Android device using a premade app containing code from the TensorFlow repositories that we’ve modified. Start by cloning the repository:
   `git clone git@github.com:riis/sheep_demo.git`
4. Once cloned, open the application in Android Studio.
5. Add your model and label files (detect.tflite and sheep_labels_list.txt) to the assets folder of the project (sheep_demo/app/src/main/assets/).
6. Open `AndroidManifest.xml` and change the package attribute in the manifest tag to match the package name you used when you created the DJI project.
7. Change the value of the API key on line 41 in the `android:value` tag to the one DJI provides you on their site.

![Create App Form](image)

*Figure 12. Creation of DJI application.*
8. In the application’s build.gradle file, change the applicationId to match the one in the manifest.

9. Correct the imports by pressing Ctrl+Shift+R (Command+Shift+R on Mac) and replacing the phrase “import com.riis.sheepcounter.R;” with “import com.your.package.name.R;” (replace the package with the one you used, see Figure 13).

**Figure 13. Replacing some incorrect imports.**

10. Click File > Sync Project with Gradle Files to update the project configuration.
11. Plug in an android device and click Run > Run ‘app’.
12. Once installed, connect your phone to the drone and select the application when prompted by Android. Open the application and wait for it to recognize the drone. Pressing the open button will allow you to see your model in action!