Android Object Detection application using PowerAI R5 / Tensorflow

Introduction

This document explains the process to build an Android Object Detection application that include an AI model built on PowerAI R5 / Tensorflow

Build an image dataset

Install the '`Fatkun Download Image`' plug-in from : https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf?hl=en

to 'bulk' download images from Google Images: https://www.google.com/imghp?hl=en

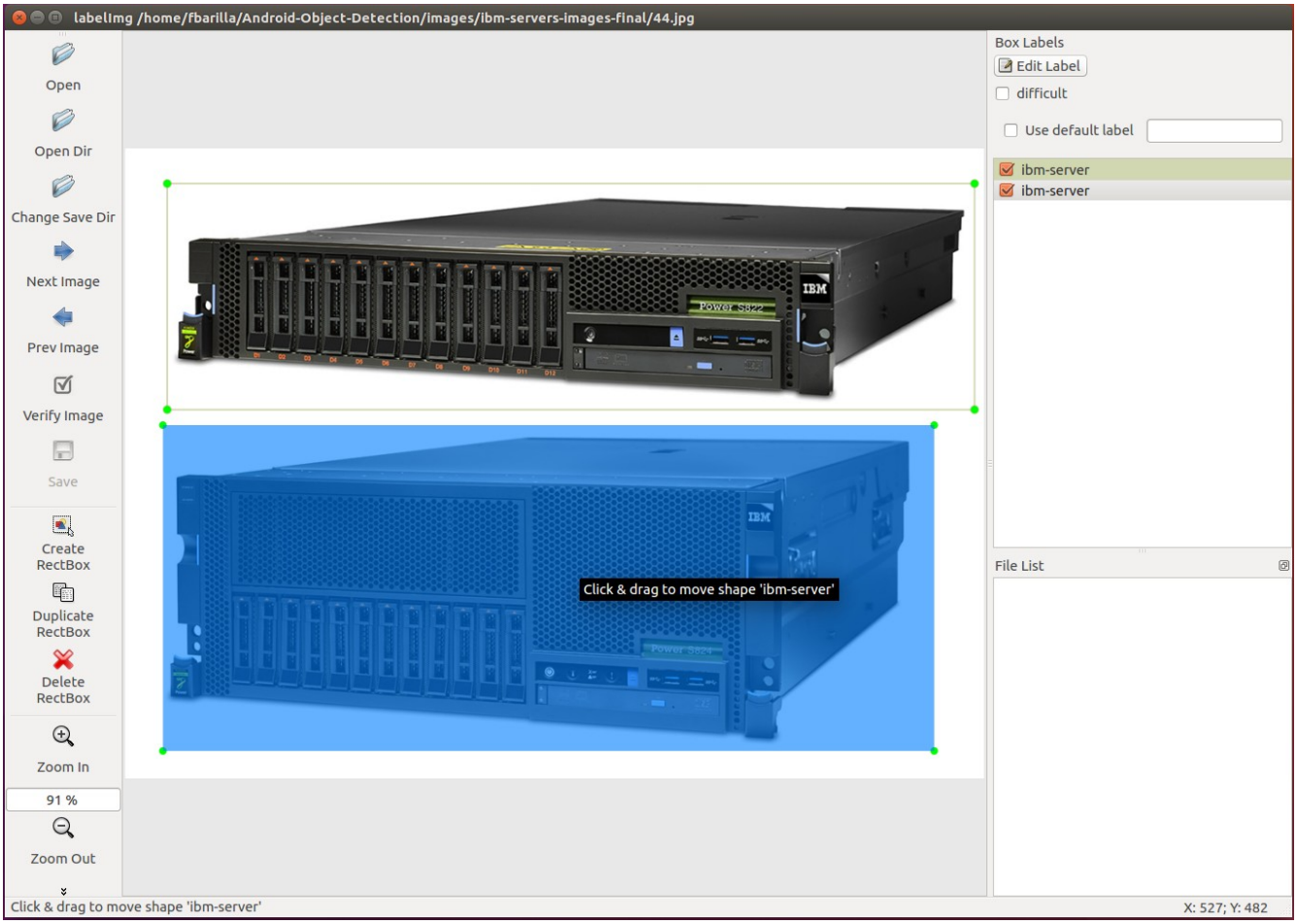
Do a search on '`IBM power server`'. Select the images you want to download and store them locally on your laptop.

Label the images

After the image dataset has been built, the next step is to label the images. What does this mean? As the goal is to do object detection, we need to identify the objects (draw a bounding box around it), to later let the Neural Network know that inside the box is the object the learning process will use.

To do it you can use `labelImg` available at: https://github.com/tzutalin/labelImg

Follow the instructions to upload your image dataset and annotate the images. It's a simple but tedious task as you'll have to go over all the images....



When the annotation process is done, tar the '`.xml`' and '`.jpg`' files and download them on the IBM Power server.

IBM Power System setup

For the experiment we use a Power 8 system (minksy) with two Nvidia Tesla P 100 GPUs and IBM PowerAI R5.

This document does not explain how to install IBM PowerAI and its prerequisites. All the documentation is available at:
https://www.ibm.com/us-en/marketplace/deep-learning-platform/resources

Load tensorflow models and dataset

On the minsky

```
# mkdir Android-Object-Detection
# cd Android-Object-Detection
# git clone https://github.com/tensorflow/tensorflow.git
# cd tensorflow/tensorflow
```

```
# git clone https://github.com/tensorflow/models.git
# cd ../../../Android-Object-Detection/
# mkdir dataset; cd dataset
# mkdir annotations
# mkdir images
```

Move the '.xml' and '.jpg' files to their respective directories ('annotations', 'images') and untar them.

Download ssd_mobilenet_v1_coco

This part is largely inspired by the work done by Dion van Velde and published at:
https://towardsdatascience.com/how-to-train-a-tensorflow-face-object-detection-model-3599dcd0c26f

Please refer to this blog for an overall understanding of the process.

The script 'download_ssd_mobilenet.py' downloads the ssd_mobilenet_v1_coco_11_06_2017 which is used to speed up the training time.

```
# cd Android-Object-Detection/demo_code
# python ./download_ssd_mobilenet.py
```

Split training and validation data

Create the 'train' and 'val' directories

```
# cd Android-Object-Detection/demo_code/data
# mkdir -p tf_train/images
# mkdir -p tf_train/annotations
# mkdir -p tf_val/images
# mkdir -p tf_val/annotations
```

Split the dataset into training and validation

```
# cp ../../dataset/images/{1..90}.jpg tf_train/images
# cp ../../dataset/annotations/{1..90}.xml tf_train/annotations
# cp ../../dataset/images/{91..130}.jpg tf_val/images
# cp ../../dataset/annotations/{91..130}.xml tf_val/annotations
```

Pascal XML to tensorflow CSV index

When the data is converted to Pascal XML, an index is created. By training and validating the dataset, we use these files as input to make TFRecords. However, it is also possible to label images with a tool like labelImg manually and use this step to create an index here.

```
# cd Android-Object-Detection/demo_code
# python xml-to-csv.py
    > tf_train - Successfully converted xml to csv.
    > tf_val -  Successfully converted xml to csv.
```

Create TFRecords

A TFRecords file is a large binary file that can be read to train the Machine Learning model. The file is sequentially read by Tensorflow in the next step. The training and validation data will be converted into binary files.

```
# cd Android-Object-Detection/demo_code/
# ./generate_tfrecord.sh
    Successfully created the TFRecords:
    /DSX/frb/Android-Object-Detection/demo_code/data/train.record
    Successfully created the TFRecords:
    /DSX/frb/Android-Object-Detection/demo_code/data/val.record
```

Setup configuration file

In the '`Android-Object-Detection/demo_code`' directory, '`ssd_mobilenet_v1.config`'is a configuration file that is used to train the model. Make sure that the following parameters reflect your current environment:

```
fine_tune_checkpoint:
train_input_reader:
eval_input_reader:
```

Furthermore, it is still possible to change the learning rate, batch size and other settings.

Training

Let's train the model.

Note: before running the training script, update `dask`

```
# conda update dask
```

Start the training process by running the following script:

```
# ./train.sh
```

or from the command line:

```
# python ../tensorflow/tensorflow/models/research/object_detection/train.py \
--logtostderr \
--pipeline_config_path=ssd_mobilenet_v1.config  \
--train_dir=model_output
```

Be patient, on a 2 Nvidia GPUs system it take a couple of hours....

Converting checkpoint to protobuf

List the latest checkpoint ID in the 'model_output' directory and run the command:

```
# python
../tensorflow/tensorflow/models/research/object_detection/export_inference_graph.py
```

```
\
--input_type image_tensor \
--pipeline_config_path=ssd_mobilenet_v1.config \
--trained_checkpoint_prefix model_output/model.ckpt-<checkpoint ID> \
--output_directory model
```

Note: if you get the

“`ValueError: Protocol message RewriterConfig has no "layout_optimizer" field.`”

error message, edit the '`tensorflow/models/research/object_detection/exporter.py`' and change line 71/72 from

```
rewrite_options = rewriter_config_pb2.RewriterConfig(
          layout_optimizer=rewriter_config_pb2.RewriterConfig.ON)
```

to

```
rewrite_options = rewriter_config_pb2.RewriterConfig()
```

and rerun the command.

Evaluation

```
# python ../tensorflow/tensorflow/models/research/object_detection/eval.py \
--logtostderr \
--pipeline_config_path=ssd_mobilenet_v1.config \
--checkpoint_dir=model_output \
--eval_dir=eval
```

Test

A Jupyter Notebook (`object_detection_tutorial.ipynb`) is provided for exercising the new trained model and display a couple of sample images with bounding boxes.

Start the Jupyter Notebook using:

```
# cd Android-Object-Detection/demo_code/
# jupyter notebook --port <forwarded port>  [--allow-root]
```

Build and deploy Android application

So far, we have trained, exported the model, and evaluated it. Now it is time embed the new model in an Android application so we can use it to detect 'IBM servers' using the phone’s camera.

First on a laptop (mine is running Ubuntu 16.04) install Android Studio from the link: https://developer.android.com/studio/index.html

When installed clone again the TensorFlow repository:

```
# mkdir Android-Object-Detection
# cd Android-Object-Detection
# git clone https://github.com/tensorflow/tensorflow.git
```

In Android Studio import a new project using the directory:

`tensorflow/tensorflow/examples/android`

from the TensorFlow repository you just cloned.

When imported, open the '`build.gradle`' file and change

```
def nativeBuildSystem = 'none'
```

by

```
def nativeBuildSystem = 'cmake'
```

Don't worry if some prerequisites are not met, Android Studio will install them for you.

Rebuild the project: `Build → Make Project`

When the build is finished, download the frozen model (`frozen_inference_graph.pb`) from the IBM server (`Android-Object-Detection/demo_code/model)` and add it to the local '`assets`' directory. Then, also in that folder, create a file called '`ibm-server.txt`' with the following content:

```
???
ibm-server
```

Next locate and open the DetectorActivity.java (under src → org.tensorflow.demo) and modify the `TF_OD_API_MODEL_FILE` and `TF_OD_API_LABELS_FILE` to reflect your environment:

```
private final static String TF_OD_API_MODEL_FILE =
"file://android_asset/frozen_inference_graph.pb";

private final static String TF_OD_API_LABELS_FILE =
"file://android_asset/ibm-server.txt";
```

We are almost ready to run! After the project has been rebuilt, click on `Run → Run 'android'` , select your Android device, and wait a couple of seconds until the application is installed on the phone. Android Studio will install four applications, however, the one that contains the new detection model is labelled '`TF Detect`' .
Launch the application and if you point the camera to an IBM Power server, you should see on the screen the bounding box around the server and the match probability.

Enjoy !