Running TensorFlow Lite Object Detection Models in Python

Good things come in (TF)lite packages!

Harshit Dwivedi
Apr 16, 2020 · 8 min read

Photo by Nik Shuliahin on Unsplash

This blog is the sixth blog in the series and a follow-up to my previous blog post on running TensorFlow Lite image classification models in Python. If you haven't read that post, you can read it here:

Running Tensorflow Lite Image Classification Models in Python

Good things come in (TF)lite packages!

heartbeat.fritz.ai

Series Pit Stops

- **Training a TensorFlow Lite Image Classification model using AutoML Vision Edge**
- **Creating a TensorFlow Lite Object Detection Model using Google Cloud AutoML**
- **Using Google Cloud AutoML Edge Image Classification Models in Python**
- **Using Google Cloud AutoML Edge Object Detection Models in Python**
- **Running TensorFlow Lite Image Classification Models in Python**
- Running TensorFlow Lite Object Detection Models in Python (You are here)
- **Optimizing the performance of TensorFlow models for the edge**

This blog post assumes that you already have a trained TFLite model on hand. If you don't or need to build one, you can take a look at the blog post that I have written here:

Using Google Cloud AutoML Edge Object Detection Models in Python

Working with edge-ready models in a Python environment

heartbeat.fritz.ai

Contrary to image classification models that classify an input image into one or more different categories, object detection models are designed to identify target objects and provide you with a bounding box around them (to track its location).

With that context established, let's jump into how to implement these models in a Python setting.

Step 1: Downloading the TensorFlow Lite model

Assuming that you've trained your TensorFlow model with Google Cloud, you can download the model from the Vision Dashboard, as shown in the screenshot here:

Once downloaded, we're ready to set up our environment and proceed with the next steps.

> Embedding machine learning models on mobile apps can help you scale while reducing costs. Subscribe to the Fritz AI Newsletter for more on this and other ways mobile ML can benefit your business.

Step 2: Installing the required dependencies

Before we go ahead and write any code, it's important that we first have all the required dependencies installed on our development machine.

For the current example, these are the dependencies we'll need:

```
tensorflow==1.13.1
pathlib
opencv-python
```

We can use pip to install these dependencies with the following command:

```
pip install dependency_name
```

> *Note: While not mandatory, it's strongly suggested that you always use a virtual environment for testing out new projects. You can read more about how to set up and activate one in the link here:*

Creating Python Virtual Environments with Conda: Why and How

An easier way to install packages and dependencies

heartbeat.fritz.ai

Step 3: Loading the model and studying its input and output

Now that we have the model and our development environment ready, the next step is to create a Python snippet that allows us to load this model and perform inference with it.

Here's what such a snippet might look like:

Here, we first load the downloaded model and then get the input and output tensors from the loaded model.

Up next, we print the input and outputs tensors we obtained earlier.
If you run the code, this is what the output might look like:

```
[{'name': 'normalized_input_image_tensor', 'index': 596, 'shape':
array([  1, 512, 512,   3], dtype=int32), 'dtype': <class
'numpy.uint8'>, 'quantization': (0.0078125, 128)}]

[{'name': 'TFLite_Detection_PostProcess', 'index': 512, 'shape':
array([], dtype=int32), 'dtype': <class 'numpy.float32'>,
```

```
'quantization': (0.0, 0)}, {'name':
'TFLite_Detection_PostProcess:1', 'index': 513, 'shape': array([],
dtype=int32), 'dtype': <class 'numpy.float32'>, 'quantization':
(0.0, 0)}, {'name': 'TFLite_Detection_PostProcess:2', 'index': 514,
'shape': array([], dtype=int32), 'dtype': <class 'numpy.float32'>,
'quantization': (0.0, 0)}, {'name':
'TFLite_Detection_PostProcess:3', 'index': 515, 'shape': array([],
dtype=int32), 'dtype': <class 'numpy.float32'>, 'quantization':
(0.0, 0)}]
```

Unlike the output of the classification model, this output looks like a bit too much to process! But let's go through it nevertheless.

Looking at the input tensor, we see that it has a single entry that takes in an RGB image of size `512 x 512` as its input at `index 596`.

Conversely, the output tensor has 4 entries, which means that unlike the previous case where we got a single-element array, here we have 4 elements in the output array.

The bounding boxes for the object that we need, along with their confidence scores, will be in two of these 4 elements. Typically, the output elements are ordered by the array of rectangles followed by the array of scores for these rectangles.

After using some trial and error, I identified that the element named `TFLite_Detection_PostProcess` contains my rectangles, and the element named `TFLite_Detection_PostProcess:2` contains the scores of these rectangles.

The element named `TFLite_Detection_PostProcess:3` contains the total number of detected items and the element `TFLite_Detection_PostProcess:1` contains the classes for the detected elements.

In our current case, printing the output of `TFLite_Detection_PostProcess:1` should print an array of zeros.
However, if you have trained an object detection to detect multiple objects; this element might have different outputs for you.

For example, here's a sample output of this node for an object detection model trained to detect 2 objects:

```
[
[0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
]
```

Over here, if a particular index has the value 0; then the box and score at that particular index belong to the first object and if it has the value 1; then the box and score at that index belong to the second object.
These values might increase if you have trained your model to detect more objects.

In the next step, we'll pass an image to the model and see the output for these outputs in action.

Step 4: Reading an image and passing it to the TFLite model

Up next, we'll use Pathlib to iterate through a folder containing images that we'll be running inference on. We'll then read each image with OpenCV, resize it to 512x512, and then pass it to our model.

Once done, we'll print the file name and the output number (0 and 2) for that file to see what that means:

Upon running this code, here's what the output might look like:

```
For file 1DSCF1649 (2)-2020-01-25T15:16:25.462Z
Rectangles are: [[
  [-0.00325413  0.47174522  0.1879216   0.61565363]
  [-0.00632208  0.24867406  0.17483354  0.3669869 ]
  [ 0.00640314  0.32291842  0.17826432  0.45696312]
  [ 0.00324218  0.17664793  0.17510337  0.31553695]
  [ 0.00893004  0.28408495  0.18565208  0.3567101 ]
```

```
  [ 0.00839117  0.22974649  0.15364154  0.33908436]
  [ 0.98135734  0.90441823  0.99792504  1.0007949 ]
  [ 0.9839504   0.8700926   0.99884427  0.9981244 ]
  [ 0.00121695  0.4290639   0.19094317  0.5823904 ]
  [ 0.460519    0.37304354  0.57724005  0.41105157]
  [ 0.9853649   0.93691933  0.99602485  0.9942064 ]
  [-0.00130505  0.24837694  0.02534466  0.34964582]
  [ 0.00229905  0.45923734  0.1424832   0.5808631 ]
  [ 0.5341829   0.54348207  0.64352083  0.57388854]
  [ 0.00227114  0.24248336  0.19199735  0.31510854]
  [ 0.98863435  0.28132698  0.9990773   0.40489325]
  [ 0.43845406  0.36674508  0.5567669   0.41571096]
  [ 0.9773656   0.8679831   0.9965315   1.0032778 ]
  [ 0.9854444  -0.02297129  0.9893893   0.01719607]
  [ 0.9880944   0.25013754  0.9989148   0.37816945]
]]
Scores are: [
[0.671875   0.54296875 0.07421875 0.02734375 0.02734375 0.02734375
  0.0234375  0.0234375  0.0234375  0.0234375  0.01953125 0.01953125
  0.01953125 0.01953125 0.01953125 0.015625   0.015625   0.015625
  0.015625   0.015625  ]
]

...
```

Upon closer inspection, you'll see that here again the number of elements in the array of rectangles and the array of scores are the same!

The elements in the scores array are all probabilities for each of the rectangles detected; so we'll only take those rectangles whose scores are greater than a particular threshold, say 0.5.

Here's a code snippet that applies this filter to the code snippet above:

Running this should result in an output like this:

```
For file 1DSCF1649 (2)-2020-01-25T15:16:25.462Z
Rectangles are: [-0.00325413  0.47174522  0.1879216   0.61565363]

For file 1DSCF1649 (2)-2020-01-25T15:16:25.462Z
Rectangles are: [-0.00632208  0.24867406  0.17483354  0.3669869 ]

For file 1IMG_20191101_183819_154-2020-01-04T11:20:44.140Z
Rectangles are: [0.01136297 0.00234886 0.53366566 0.47188312]

For file 1IMG_20191101_183819_154-2020-01-04T11:20:44.140Z
Rectangles are: [0.00472282 0.6394675  0.2634614  0.9956119 ]
```

As you can see here, after setting a filter of 0.5 on the scores, we get a substantially reduced number of rectangles (1 in most of the cases).

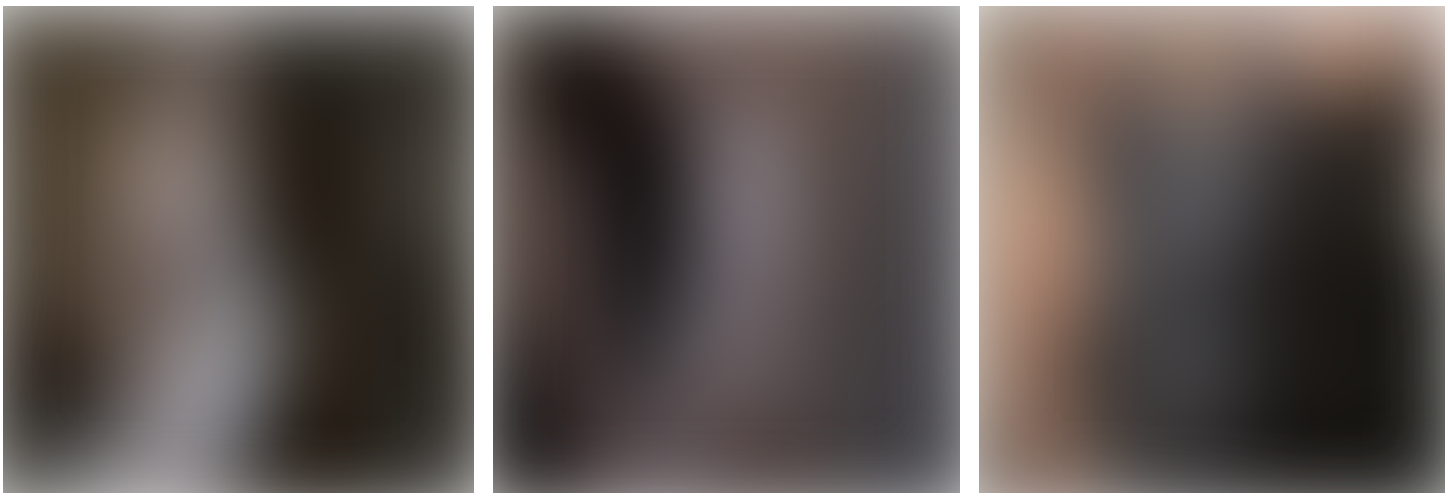
Up next, we will use OpenCV to plot these rectangles on the original image and show it on the screen.

Step 5: Plotting the detected rectangles with OpenCV

In order to calculate the x and y coordinates of the rectangle to be drawn, we'll be using a helper function that takes in the detected box and converts its elements into proper x and y coordinates of a rectangle that OpenCV can use.

Here's what that looks like:

And that's it! Upon running the complete code, integrated with this helper method above, here's what you should see:



As seen here, the model does an excellent job of identifying and detecting occluded faces in an image.

In case you're looking for the complete source code of the example, here's what it looks like:

And that's it! While not always the most effective solution, in the last post and this one we saw how easy it is to load and run TensorFlow Lite models in a Python-based setting.

If you have any questions or suggestions about this post, feel free to leave a comment down below and I'll be happy to follow up!

Thanks for reading! If you enjoyed this story, please ***click the 👏 button and share it*** *to help others find it! Feel free to leave a comment 💬 below.*

Have feedback? Let's connect ***on Twitter****.*

Editor's Note: ***Heartbeat*** *is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We're committed to supporting and inspiring developers and engineers from all walks of life.*

Editorially independent, Heartbeat is sponsored and published by ***Fritz AI****, the machine learning platform that helps developers teach devices to see, hear, sense, and think. We pay our contributors, and we don't sell ads.*

If you'd like to contribute, head on over to our ***call for contributors****. You can also sign up to receive our weekly newsletters (****Deep Learning Weekly*** *and the* ***Fritz AI Newsletter****), join us on* ***Slack****, and follow Fritz AI on* ***Twitter*** *for all the latest in mobile machine learning.*

Machine Learning Python TensorFlow Tensorflow Lite Heartbeat