

Deploying Tensorflow model on Andorid device for Human Activity Recognition

Posted on May 2, 2017

Fork

102

In this tutorial, we will learn how to deploy human activity recognition (HAR) model on Android device for real-time prediction. The majority of the code in this post is largely taken from Omid Alemi's simply elegant tutorial named "Build Your First Tensorflow Android App" (<https://omid.al/posts/2017-02-20-Tutorial-Build-Your-First-Tensorflow-Android-App.html>). This post could not have been possible without Omid's contribution. I am building upon his work in this post. If you have not seen the post on how to build a deep convolutional neural network for HAR, please follow this link (<http://aqibsaeed.github.io/2016-11-04-human-activity-recognition-cnn/>). Here, details on how to freeze and export the model for use in Android app are discussed.

```
1 X = tf.placeholder(tf.float32, shape=[None, input_width * num_channels], name="input")
2 X_resaped = tf.reshape(X, [-1, 1, 90, 3])
```

Now add following lines of code after model training steps to checkpoint and save the graph definition.

```
1 with tf.Session() as session:
2     /**
3     ----- Model training code goes here -----
4     **/
5     tf.train.write_graph(session.graph_def, '.', './har.pbtxt')
6     saver.save(session, save_path = './har.ckpt')
```

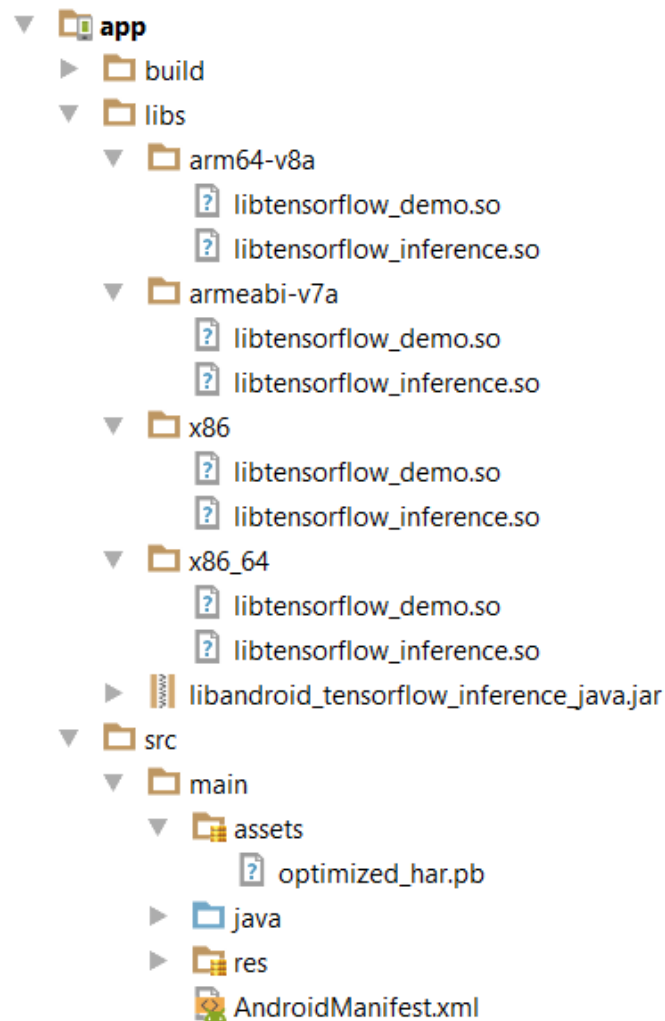
Next, the checkpointed model will be frozen and its optimized version can be saved

as follows:

```
1  from tensorflow.python.tools import freeze_graph
2  from tensorflow.python.tools import optimize_for_inference_lib
3
4  freeze_graph.freeze_graph(input_graph = "../har.pbtxt", input_saver = "",
5                           input_binary = False, input_checkpoint = "../har.ckpt", output_node_names = "y_",
6                           restore_op_name = "save/restore_all", filename_tensor_name = "save/Const:0",
7                           output_graph = "frozen_har.pb", clear_devices = True, initializer_nodes = "")
8
9  input_graph_def = tf.GraphDef()
10 with tf.gfile.Open(output_frozen_graph_name, "r") as f:
11     data = f.read()
12     input_graph_def.ParseFromString(data)
13
14 output_graph_def = optimize_for_inference_lib.optimize_for_inference(
15     input_graph_def,
16     ["input"],
17     ["y_"],
18     tf.float32.as_datatype_enum)
19
20 f = tf.gfile.FastGFile("optimized_har.pb", "w")
21 f.write(output_graph_def.SerializeToString())
```

Let's make a simple Android app to get accelerometer data and make predictions about user activities with the learned model. Before doing so, get Tensorflow libraries for Android from the following link (<https://ci.tensorflow.org/view/Nightly/job/nightly-android/>). For the purpose of this tutorial, I used the libraries with artifact number 117.

We will now create an empty app and copy *libandroid_tensorflow_inference_java.jar*, *arm64-v8a*, *armeabi-v7a*, *x86* and *x86_64* files/folders in libs directory of the project. Also, copy the frozen optimized model into an asset directory of the project. The figure shows directory structure after copying all the required files.



Afterwards, add following entry in `build.gradle` file to let build system know where files are located.

```
1 sourceSets{
2     main {
3         jniLibs.srcDirs = ['libs']
4     }
5 }
```

Now we create a class, which will use `TensorflowInferenceInterface` for feeding data to a model, running inference and getting back results.

```
1 import android.content.Context;
2 import android.content.res.AssetManager;
3 import org.tensorflow.contrib.android.TensorFlowInferenceInterface;
4
5 public class ActivityInference {
6     static {
7         System.loadLibrary("tensorflow_inference");
8     }
9
10    private static ActivityInference activityInferenceInstance;
11    private TensorFlowInferenceInterface inferenceInterface;
12    private static final String MODEL_FILE = "file:///android_asset/optimized_har.pb";
13    private static final String INPUT_NODE = "input";
14    private static final String[] OUTPUT_NODES = {"y_"};
15    private static final String OUTPUT_NODE = "y_";
16    private static final long[] INPUT_SIZE = {1,270};
17    private static final int OUTPUT_SIZE = 6;
18    private static AssetManager assetManager;
19
20    public static ActivityInference getInstance(final Context context)
21    {
22        if (activityInferenceInstance == null)
23        {
24            activityInferenceInstance = new ActivityInference(context);
25        }
26        return activityInferenceInstance;
27    }
28
29    public ActivityInference(final Context context) {
30        this.assetManager = context.getAssets();
31        inferenceInterface = new TensorFlowInferenceInterface(assetManager, MODEL_FILE);
32    }
33
34    public float[] getActivityProb(float[] input_signal)
35    {
36        float[] result = new float[OUTPUT_SIZE];
37        inferenceInterface.feed(INPUT_NODE, input_signal, INPUT_SIZE);
38        inferenceInterface.run(OUTPUT_NODES);
39        inferenceInterface.fetch(OUTPUT_NODE, result);
40        //Downstairs Jogging Sitting Standing Upstairs Walking
41        return result;
42    }
43 }
```

The rest of the code provided below is from MainActivity class. It is listening to accelerometer sensor and using getActivityProb method of ActivityInference class for getting a probability of each class to update the UI.

```
1  import android.content.Context;
2  import android.hardware.Sensor;
3  import android.hardware.SensorEvent;
4  import android.hardware.SensorEventListener;
5  import android.hardware.SensorManager;
6  import android.os.Bundle;
7  import android.support.v7.app.AppCompatActivity;
8  import android.widget.TextView;
9
10 import java.math.BigDecimal;
11 import java.util.ArrayList;
12 import java.util.List;
13
14 public class MainActivity extends AppCompatActivity implements SensorEventListener {
15
16     private final int N_SAMPLES = 90;
17     private static List x;
18     private static List y;
19     private static List z;
20     private static List input_signal;
21     private SensorManager mSensorManager;
22     private Sensor mAccelerometer;
23     private ActivityInference activityInference;
24
25     private TextView downstairsTextView;
26     private TextView joggingTextView;
27     private TextView sittingTextView;
28     private TextView standingTextView;
29     private TextView upstairsTextView;
30     private TextView walkingTextView;
31
32     @Override
33     protected void onCreate(Bundle savedInstanceState) {
34         super.onCreate(savedInstanceState);
35         setContentView(R.layout.activity_main);
36         x = new ArrayList();
37         y = new ArrayList();
38         z = new ArrayList();
39         input_signal = new ArrayList();
40
41         downstairsTextView = (TextView)findViewById(R.id.downstairs_prob);
42         joggingTextView = (TextView)findViewById(R.id.jogging_prob);
43         sittingTextView = (TextView)findViewById(R.id.sitting_prob);
44         standingTextView = (TextView)findViewById(R.id.standing_prob);
45         upstairsTextView = (TextView)findViewById(R.id.upstairs_prob);
46         walkingTextView = (TextView)findViewById(R.id.walking_prob);
47
48         mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
49         mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
50         mSensorManager.registerListener(this, mAccelerometer ,
51             SensorManager.SENSOR_DELAY_FASTEST);
52         activityInference = new ActivityInference(getApplicationContext());
53     }
54 }
```

```
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}

protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mAccelerometer,
        SensorManager.SENSOR_DELAY_FASTEST);
}

@Override
public void onSensorChanged(SensorEvent event) {
    activityPrediction();
    x.add(event.values[0]);
    y.add(event.values[1]);
    z.add(event.values[2]);
}

@Override
public void onAccuracyChanged(Sensor sensor, int i) {

}

private void activityPrediction()
{
    if(x.size() == N_SAMPLES && y.size() == N_SAMPLES && z.size() == N_SAMPLES) {
        // Copy all x,y and z values to one array of shape N_SAMPLES*3
        input_signal.addAll(x); input_signal.addAll(y); input_signal.addAll(z);

        // Perform inference using Tensorflow
        float[] results = activityInference.getActivityProb(toFloatArray(input_signal));

        downstairsTextView.setText(Float.toString(round(results[0],2)));
        joggingTextView.setText(Float.toString(round(results[1],2)));
        sittingTextView.setText(Float.toString(round(results[2],2)));
        standingTextView.setText(Float.toString(round(results[3],2)));
        upstairsTextView.setText(Float.toString(round(results[4],2)));
        walkingTextView.setText(Float.toString(round(results[5],2)));

        // Clear all the values
        x.clear(); y.clear(); z.clear(); input_signal.clear();
    }
}

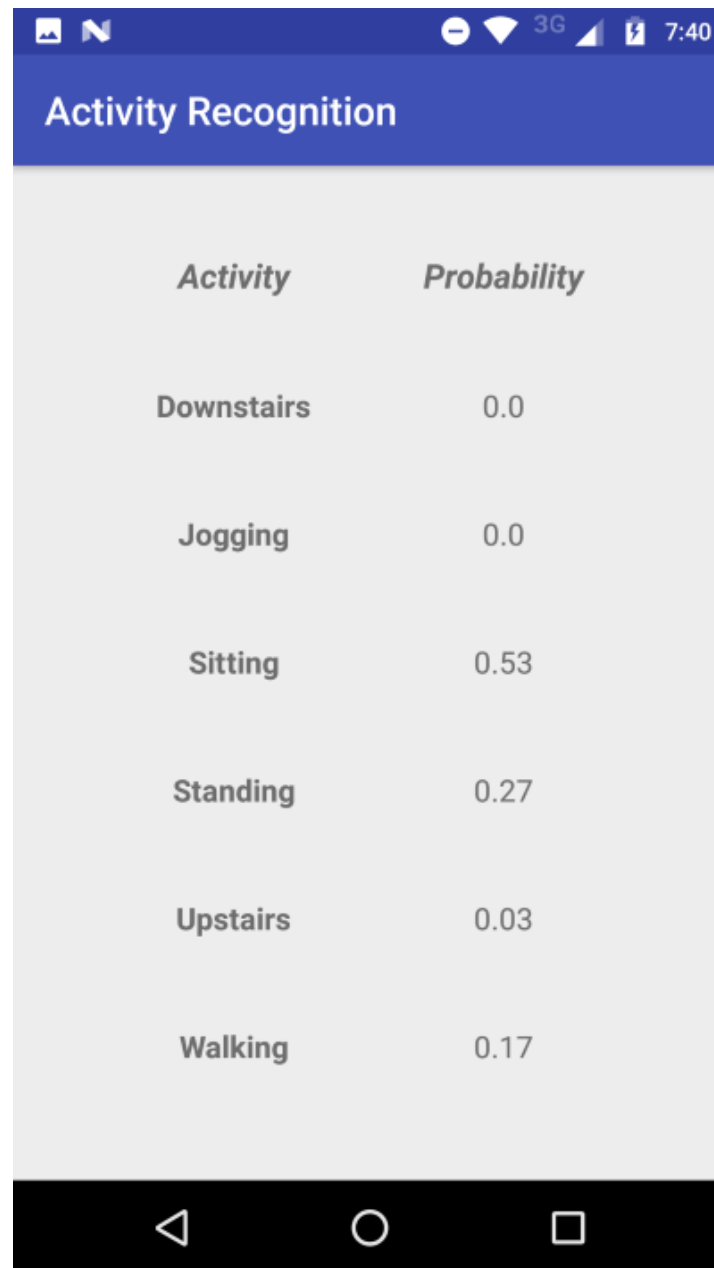
private float[] toFloatArray(List list)
{
    int i = 0;
    float[] array = new float[list.size()];

    for (Float f : list) {
        array[i++] = (f != null ? f : Float.NaN);
    }
}
```

```
        return array;
    }

    public static float round(float d, int decimalPlace) {
        BigDecimal bd = new BigDecimal(Float.toString(d));
        bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
        return bd.floatValue();
    }
}
```

This is all that we need to use Tensorflow on Android for making predictions about user activities. A simple app UI will look something as shown in the screenshot below.



The screenshot shows an Android application interface with a blue header bar containing the title 'Activity Recognition'. Below the header is a table with two columns: 'Activity' and 'Probability'. The table lists six activities with their corresponding probabilities: Downstairs (0.0), Jogging (0.0), Sitting (0.53), Standing (0.27), Upstairs (0.03), and Walking (0.17). The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

<i>Activity</i>	<i>Probability</i>
Downstairs	0.0
Jogging	0.0
Sitting	0.53
Standing	0.27
Upstairs	0.03
Walking	0.17

The complete code of the app is available at the following link (<https://github.com/aqibsaeed/Human-Activity-Recognition-using-CNN/tree/master/ActivityRecognition>). If you have any question, please comment below.



← **PREVIOUS POST (/2017-01-21-SENSOR-FUSION-AND-INPUT-REPRESENTATION/)**

NEXT POST → (/2017-08-11-GENETIC-ALGORITHM-FOR-OPTIMIZING-RNN/)

11 Comments aqibsaeed

 Login ▾ Recommend  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS **Ken Kezhi Li** • 9 months ago

good article. Would like to see more great articles from you.

1 ^ | ▾ • Reply • Share >

**Sabir Ullah** • 9 days ago

Would know about used tensorflow in android studio

^ | ▾ • Reply • Share >

**张宇** • 3 months ago

Thanks for your great work that helps me a lot!

^ | ▾ • Reply • Share >

**Chaine** • 9 months ago**@Aaqib Saeed** Can you please provide the exact code for training the model and generating the .pb file? Thanks!

^ | ▾ • Reply • Share >

**Aaqib Saeed** Mod ➔ Chaine • 9 months ago

Exact code is available in this blog post. If your app is crashing, please look into the log why it's crashing. Make sure the input output node names, dimensions etc are same. There is no silver bullet :) If your optimized_har model is failing, there could be several reasons for that! my suggestion is to debug thoroughly.

^ | ▾ • Reply • Share >

**Chaine** • 9 months ago

Hmm that's odd. Always outputting Upstairs with the highest probability even though I'm stationary.

^ | ▾ • Reply • Share >

**Aaqib Saeed** Mod ➔ Chaine • 9 months ago

The model available in github repository is not optimized. The purpose of the tutorial is to show how it is done and not to provide out of box solution. Play around with model try to optimize it yourself :)

1 ^ | ▾ • Reply • Share >



(<https://github.com/aqibsaeed>)



(<mailto:aqibsaeed@protonmail.com>)

Aaqib Saeed • 2018