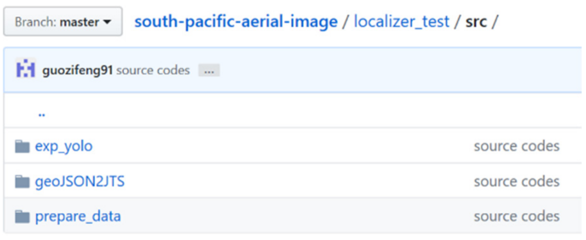
Here is a simple guide of how to run the JAVA localizer:

First you need to have a java develop platform, for example eclipse. Install the platform and you are ready to code in JAVA.

Then, start a new project and copy-paste all the code in "localizer_text/src" from github to your JAVA project. Note that the prepare_data folder is not necessary.



Branch: master south-pacific-aerial-image / localizer_test / src /

guozifeng91 source codes ...

..

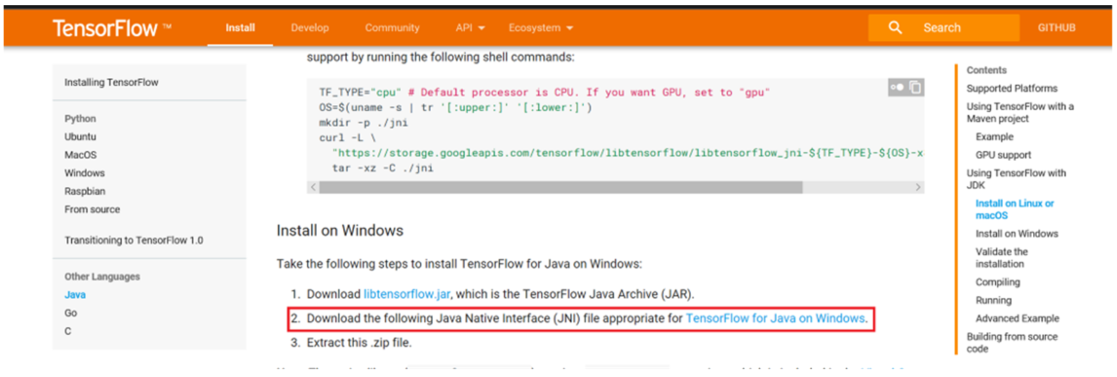
exp_yolo	source codes
geoJSON2JTS	source codes
prepare_data	source codes

Then, install three libraries: tensorflow, processing 2.2.1, and OpenCV 3.3.:

https://www.tensorflow.org/install/install_java#using_tensorflow_with_jdk (tensorflow)
https://processing.org/download/ (processing)
https://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.3.1/opencv-3.3.1-vc14.exe/download (open cv)

The jar files of these libraries must be included into the build path of your java project, in addition, they have some native runtime files (.dll or .jni) that must be included as well. For tensorflow and processing, follow these links

https://www.tensorflow.org/install/install_java#using_tensorflow_with_jdk (tensorflow)



support by running the following shell commands:

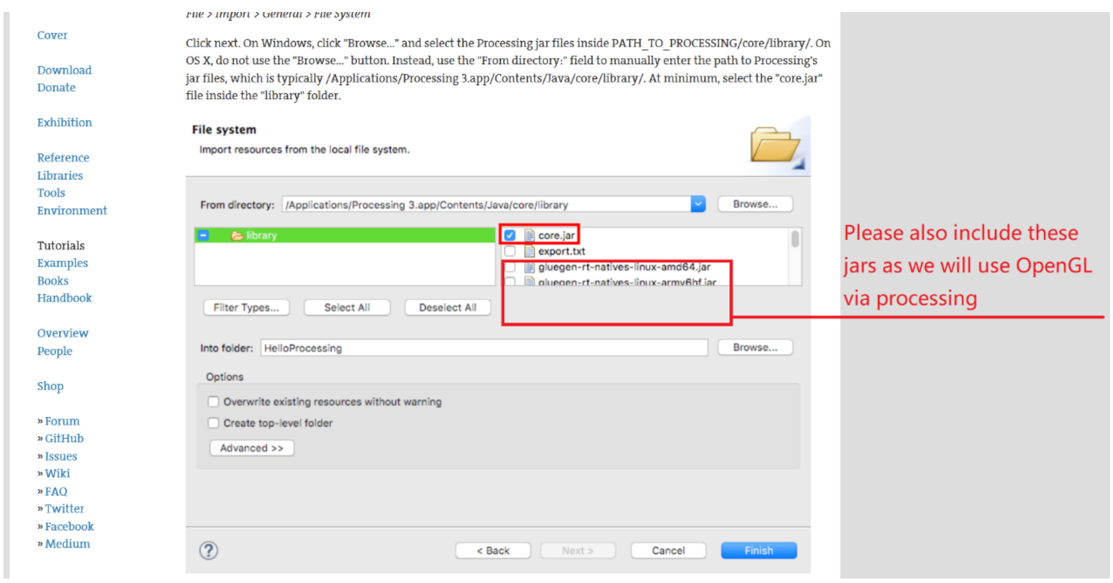
```
TF_TYPE="cpu" # Default processor is CPU. If you want GPU, set to "gpu"
OS=$(uname -s | tr '[:upper:]' '[:lower:]')
mkdir -p ./jni
curl -L \
  "https://storage.googleapis.com/tensorflow/libtensorflow/libtensorflow_jni-${TF_TYPE}-${OS}-x
  tar -xz -C ./jni
```

Install on Windows

Take the following steps to install TensorFlow for Java on Windows:

1. Download libtensorflow.jar, which is the TensorFlow Java Archive (JAR).
2. Download the following Java Native Interface (JNI) file appropriate for TensorFlow for Java on Windows.
3. Extract this .zip file.

https://processing.org/tutorials/eclipse/ (processing)



File > Import > General > File System

Click next. On Windows, click "Browse..." and select the Processing jar files inside PATH_TO_PROCESSING/core/library/. On OS X, do not use the "Browse..." button. Instead, use the "From directory:" field to manually enter the path to Processing's jar files, which is typically /Applications/Processing 3.app/Contents/Java/core/library/. At minimum, select the "core.jar" file inside the "library" folder.

File system

Import resources from the local file system.

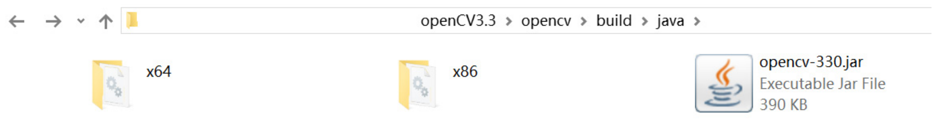
From directory: /Applications/Processing 3.app/Contents/Java/core/library

library: [✓] core.jar, [ ] export.txt, [ ] gluegen-rt-natives-linux-amd64.jar, [ ] gluegen-rt-natives-linux-armv6hf.jar

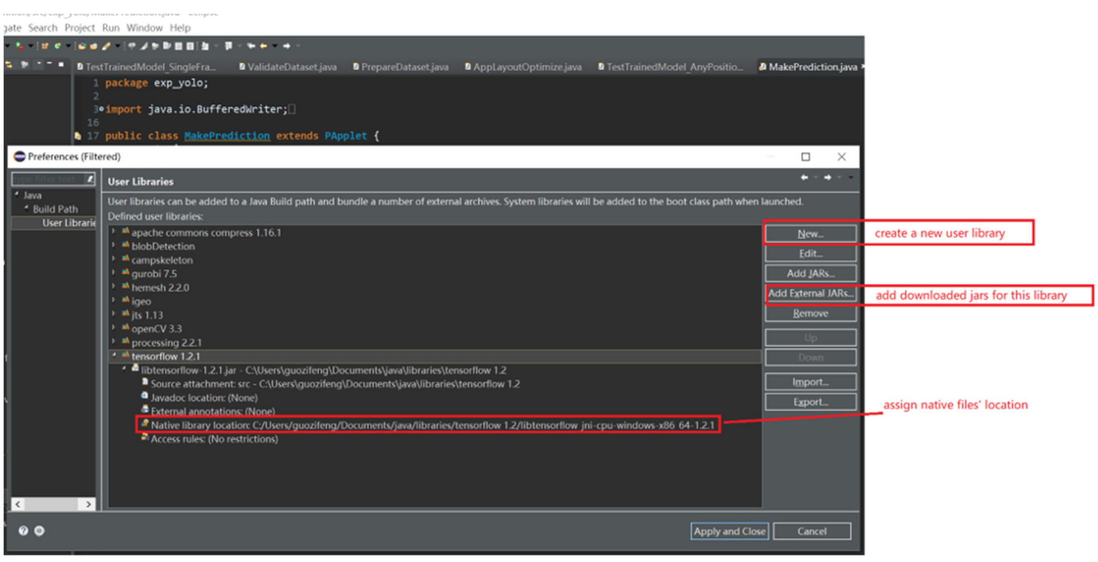
Into folder: HelloProcessing

Options
[ ] Overwrite existing resources without warning
[ ] Create top-level folder

for opencv, after download and install it on your computer, add opencv-330.jar and opencv_java330.dll (from one of the folders on the left, depend on if the system is 32 bit or 64 bit) into the build path (below).



It is recommended to use “User Library” to import the jars and the native runtime files: right click project name – building path – user libraries – next – user libraries, a window like this will show up:


Search Project Run Window Help
TestTrainedModel_SingleFra...
ValidateDataset.java
PrepareDataset.java
AppLayoutOptimize.java
TestTrainedModel_AnyPositio...
MakePrediction.java

```
package exp_yolo;

import java.io.BufferedWriter;

public class MakePrediction extends PApplet {
```

Preferences (Filtered)

Java > Build Path > User Libraries

User Libraries

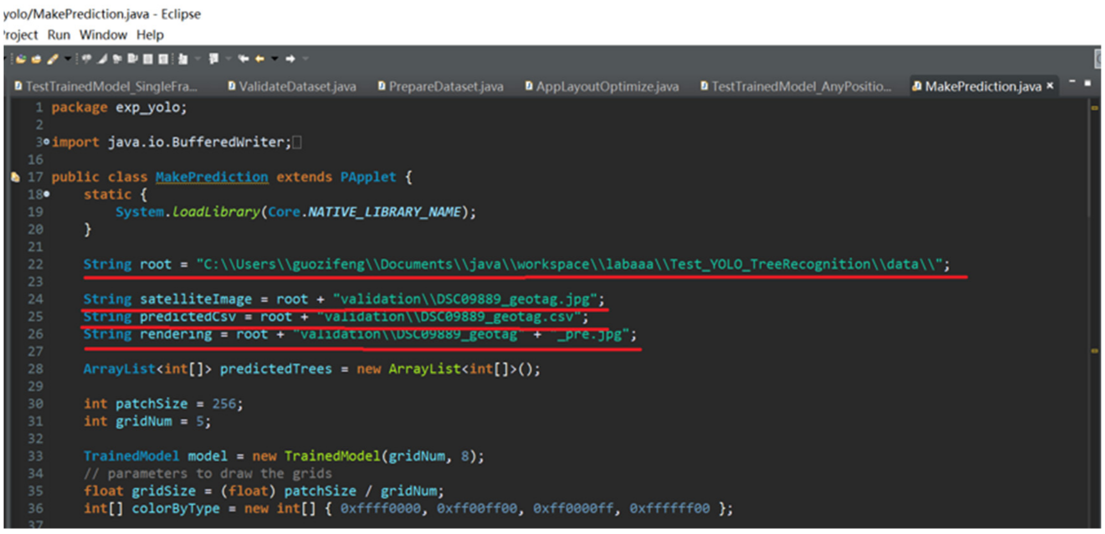
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

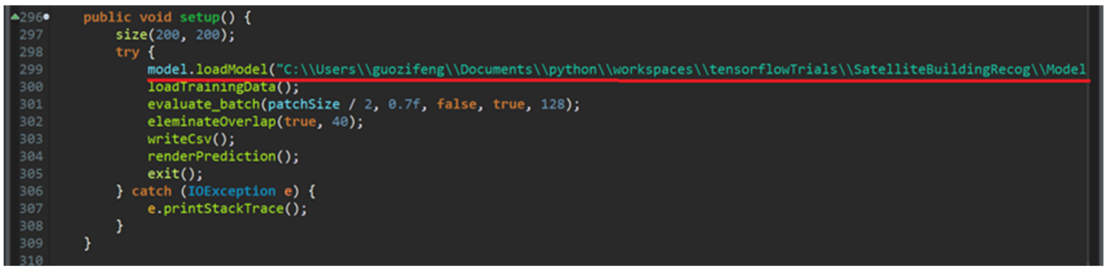
- apache commons compress 1.16.1
- blobDetection
- campskeleton
- gurobi 7.5
- hemesh 2.2.0
- igeo
- jts 1.13
- openCV 3.3
- processing 2.2.1
- tensorflow 1.2.1
  - libtensorflow-1.2.1.jar - C:\Users\guozifeng\Documents\java\libraries\tensorflow 1.2
    - Source attachment: src - C:\Users\guozifeng\Documents\java\libraries\tensorflow 1.2
    - Javadoc location: (None)
    - External annotations: (None)
    - Native library location: C:/Users/guozifeng/Documents/java/libraries/tensorflow 1.2/libtensorflow jni-cpu-windows-x86 64-1.2.1
    - Access rules: (No restrictions)

Click New to create a new user library, where you need to give a name.
Then, choose Add External JARs to include all the necessary jar files to the user library
Create one user library for each library, not for all of them
Then, set the native files' location to where they are on your drive, click Apply and Close

Now the code is ready, one more step before launch them: you need to change some paths in the code (**MakePrediction.java** and **TestTrainedModel_AnyPosition_ForVideo.java**) to where you store your jpeg imagery and the .pb file, the paths will look different if you are using Mac not Windows (below, red lines in the code)



```java
package exp_yolo;

import java.io.BufferedWriter;

public class MakePrediction extends PApplet {
    static {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    }

    String root = "C:\\Users\\guozifeng\\Documents\\java\\workspace\\labaaa\\Test_YOLO_TreeRecognition\\data\\";

    String satelliteImage = root + "validation\\DSC09889_geotag.jpg";
    String predictedCsv = root + "validation\\DSC09889_geotag.csv";
    String rendering = root + "validation\\DSC09889_geotag" + "_pre.jpg";

    ArrayList<int[]> predictedTrees = new ArrayList<int[]>();

    int patchSize = 256;
    int gridNum = 5;

    TrainedModel model = new TrainedModel(gridNum, 8);
    // parameters to draw the grids
    float gridSize = (float) patchSize / gridNum;
    int[] colorByType = new int[] { 0xffff0000, 0xff00ff00, 0xff0000ff, 0xffffff00 };
```



```java
    public void setup() {
        size(200, 200);
        try {
            model.loadModel("C:\\Users\\guozifeng\\Documents\\python\\workspaces\\tensorflowTrials\\SatelliteBuildingRecog\\Model
            loadTrainingData();
            evaluate_batch(patchSize / 2, 0.7f, false, true, 128);
            eleminateOverlap(true, 40);
            writeCsv();
            renderPrediction();
            exit();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Then that's it, click run and the program should run properly.

MakePrediction.java takes a full-size imagery as input, and exports coordinates and types of all recognized trees as csv file, and renders an equal-size bounding box image which can be overlapped with the input one.

TestTrainedModel_AnyPosition_ForVideo.java shows a user interface where you can navigate through the input imagery and see the recognition result of the place of interest.