OpenCV with Android

OpenCV is a very useful library for performing image processing and computer vision related tasks. It is supported on various development platform and is highly stable. For Android, it comes with an OpenCV4Android SDK. To work with OpenCV on Android, we need to integrate the library in our application. We will look at how to do that, as well as learn how to use it in our application.

Include OpenCV SDK in an Android Studio project

We need to follow the following steps to include OpenCV SDK in our Android Studio projects. For this tutorial, we are using "*opencv-3.2.0-android-sdk*" with Android Studio 2.3.3.

1. Download the OpenCV SDK from OpenCV official download page http://opencv.org/releases.html. We can get the most recent version of OpenCV library. Click on the "*Android Pack*" corresponding to your chosen version. Download and save the zip file in an easily accessible location, such as your Desktop or home folder. Unzip the folder.
2. To import the OpenCV SDK, go to *File > New > Import Module*. Locate the unzipped "*OpenCV-android-sdk*" folder that you downloaded and import "*sdk/java/*" folder in your project.
3. The library will take a while to import and should be imported without any error. Next, go to the "*Gradle Scripts*" tab in Projects view and open the "build.gradle" file for the imported OpenCV module.
   Change four properties in the file to match your project settings, viz.

   ```
   compileSdkVersion
   buildToolsVersion
   minSdkVersion
   targetSdkVersion
   ```

   It is important that these 4 properties are same in both OpenCV module as well as your application module.
4. Now, we need to add the module dependency to out application module. To do so, go to your application settings. You can do so by right-clicking on your application module in the Project view and selecting "Open Module Settings" option. Alternatively, you can also go via "File > Project Structure". Go to your application module and select the "Dependencies" tab. Click on the '+' sign, select "Module Dependency" from the menu and choose "openCVLibraryXXX" module from the dialog box to add the dependency.
5. To use the imported OpenCV SDK, we need to import all the precompiled static openCV libraries. Simply go to the "Opencv-android-sdk/sdk/" and copy the "libs" folder under "sdk/native" to your Android Studio project under "/app/src/main/". The folder contains all the precompiled static libraries for a number of platforms. You can either copy all the platforms or only the ones for which you are developing the application. We will later need to link these static libraries to our project for native use, if we are using Java Native Interface(jni) in our project.
6. Rename the copied "libs" directory to "jniLibs" and we are ready to use OpenCV in our Java class.

Import and Check OpenCV in an application

In this section we will see how to use OpenCV in your android application. We will create a new project and load openCV in it. We will also create checks to see if OpenCV is loaded successfully or not.

Follow the steps mentioned in Section 1, to include the library in your project. Next, follow the given steps to use it in your application.

1. Go to the MainActivity class in MainActivity.java file and create a new String variable called "TAG". This is a general good practice which helps us in debugging the app. We will post all message of significance using this tag, which will make debugging using "logcat" (see Android Application Usage Profiling) easier.
2. Next, include

   ```
   static {System.loadLibrary("opencv_java3");}
   ```

   in your mainActivity.java file.
3. Add the following lines of code inside the static module.

```
if(OpenCVLoader.initDebug()){
    Log.d(TAG, "OpenCV successfully imported!");
}
else{
    Log.d(TAG,"OpenCV couldn't be imported successfully!");
}
```

4. You may see a few errors, indicated by red line under some keywords. Move your cursor to those keywords and press "Alt+Return" on Linux and Windows or "Option+Return" on Mac. The required libraries will be imported automatically.

We are now ready to check whether OpenCV was imported in our app.

1. Open the Android Monitor and switch to logcat pane as mentioned on this page Android Application Usage Profiling
2. For our ease of use, we will create a filter to see only the log messages which we want to see. On the right side of logcat window, you will see a drop-down box. Click on it and select "Edit Filter Configuration". Enter the filter name "MyApp" and in the Log Tag field, enter the String stored in "TAG" variable. Click on Ok to create the filter and select it.
3. Compile and run the application on a device or emulator. Once the application is loaded, you should see the log message on the logcat, stating whether OpenCV was loaded or not. If it is not successfully loaded, you can switch the filter from your filter to Only Selected Application, and check the logcat for further information on why the library was not loaded and debug it.

Sometimes you may get and error saying NDK integration is deprecated. Just open your "gradle.properties" file from Gradle Scripts in Project view. Add the line "*android:useDeprecatedNdk=true*".

Using OpenCV to Capture frames from Camera

Now we will see how to use the camera on your phone to capture frames from your device and process it. If you do not have a device, you can use create a device emulator which provides you with a simulated camera.

To capture frames and display them on screen, we need to use the "javaCameraView" class from OpenCV. There are two steps to this process.

- Define the UI for showing the frames in the "activity_main.xml" file, and
- Define the functions to capture frames and handle all possible cases in MainActivity.java file.

1. **User Interface**: To define the view on the screen you first need to go to the "activity_main.xml" file.
    a. Select the "Text" tab on "activity_main.xml" window. You should be able to see a window on the right showing your application layout and to its left an xml file.
    b. In the xml file, add the following code snippet before the file ends.

```
<org.opencv.android.JavaCameraView
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:id="@+id/java_camera_view"
    />
```

    c. You should be able to see a JavaCameraView on the Preview pane in the application layout. That is where your frames will be rendered on the screen. You can reshape it as you want on the screen by click and drag on the Preview pane layout itself. Note the id of the view we have specified as "java_camera_view". We will be needing it to link this widget to the code.
2. **Code**: Now go to the MainActivity.java file and in the MainActivity class add the following steps of code.
    a. First we need to define an object of the JavaCameraView class. So, below your static block which we used to check whether OpenCV was imported or not, add the given declaration.

```
JavaCameraView javaCameraView;
```

    b. Next we go to the onCreate function which is already provided by Android Studio when you create a new project in the MainActivity.java class. We need to link our created object to the widget on the layout. We can do so by adding the following code snippet.

```
javaCameraView =
(JavaCameraView)findViewById(R.id.java_camera_view);
javaCameraView.setVisibility(SurfaceView.VISIBLE);
javaCameraView.setCvCameraViewListener(this);
```

The first line links our object to the widget. Now whatever we do with the object will be reflected on the widget. The next two line sets the visibility of the widget of the "VISIBLE" and creates a Listener for the widget referencing the class. Now we haven't setup a listener which we will be doing in a moment. So, you should see "this" underlined in red.

c. But before we setup our callback listener, we need to override some functions like onPause, onResume and onDestroy, in order to tell the application what to do when such situation arise. We will basically override these function such that our javaCameraView widget knows what to do. We can do so by adding the following snippet after the onCreate function.

```
@Override
protected void onPause(){
    super.onPause();
    if(javaCameraView!=null)
        javaCameraView.disableView();
}

@Override
protected void onDestroy(){
    super.onDestroy();;
    if(javaCameraView!=null)
        javaCameraView.disableView();
}

@Override
protected void onResume(){
    super.onResume();
    if(OpenCVLoader.initDebug()){
        Log.d(TAG, "OpenCV successfully imported!");

    }
    else{
        Log.d(TAG,"OpenCV couldn't be imported successfully!");

    }
}
```

We need to invoke the super Class of each of these class and then override them. The onPause and onDestory functions essentially notifies the javaCameraView to be disabled. On resuming the application, we need to check whether OpenCV is loaded again or not. That we do exactly like we checked before.

d. Now, if the library is loaded successfully, we need to enable the javaCameraView again. To handle this we define an object of class BaseLoaderCallback, called mLoaderCallback. To define the object add the following lines of code at the start of your MainActivity class.

```
BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this)
{
    @Override
    public void onManagerConnected(int status) {
        switch (status){
            case BaseLoaderCallback.SUCCESS:{
                javaCameraView.enableView();
                break;
            }
            default:{
                super.onManagerConnected(status);
            }
        }
        super.onManagerConnected(status);
    }
};
```

Since we haven't included the BaseLoaderCallback class, it will be underlined by red. Just use "Alt+Return"(Linux/Windows) or "Option+Return"(Mac) to import the required classes. This is another useful feature that Android Studio provides, in case you miss importing the required classes.

e. Now, we can modify our onResume function to enable the javaCameraView if the library is loaded successfully. If the library is not loaded successfully, we will try to load it again on an asynchronized thread. Go ahead and modify the onResume function with the given function int he snippet.

```
@Override
protected void onResume(){
    super.onResume();
    if(OpenCVLoader.initDebug()){
        Log.d(TAG, "OpenCV successfully imported!");

mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCE
SS);
    }
    else{
        Log.d(TAG,"OpenCV couldn't be imported successfully!");

OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_2_0,this,
mLoaderCallback);
    }
}
```

The "OpenCVLoader.OPENCV_VERSION_X_X_X" parameter may differ depending on what version of OpenCV you included in the application.

f. Now, going back to the onCreate function, we will notice we have not setup a camera view listener yet. We can do this by modifying the class signature to implement "CvCameraViewListener2" class like this.

```
public class MainActivity extends AppCompatActivity implements
CameraBridgeViewBase.CvCameraViewListener2 {
...
} //end class
```

As soon as you modify the line, you will see the line underlined in red. On using our shortcut "Alt+Return"(Linux/Windows) or "Option+Return"(Mac), we will seeAndroid Studio pointing us to implement a bunch of required function. Go ahead and add them to our code. We will see the following code snippet added at the end of our class.

```
@Override
public void onCameraViewStarted(int width, int height) {

}

@Override
public void onCameraViewStopped() {

}

@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
inputFrame) {
    return null;
}
```

This is where we define how to capture a frame and what to do with them once captured.

g. To get the frame, first we will need to create a placeholder for it. OpenCV supports "Mat" format as image object type. So first we create a Mat object in our class which will hold the captured frame. Then, we modify the 3 functions that we need to implement as shown.

```
private Mat myFrame;
...
@Override
public void onCameraViewStarted(int width, int height) {
    myFrame = new Mat(height, width, CvType.CV_8UC4);
}

@Override
public void onCameraViewStopped() {
    myFrame.release();
}

@Override
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
inputFrame) {
    myFrame = inputFrame.rgba();
    return myFrame;
}
```

The codes are self-explanatory. On camera startup we allocate memory to the Mat object, as much as required by the frame. On camera view stop, we release the memory. The onCameraFrame function is the most relevant function for us. This is where we

capture the frame and process it. The captured frame from camera is stored in inputFrame and we can point a Mat to it as we have done in the code snippet. We can specify the format of frame we want to save. Here we have saved the image in RGBA format, while we can also save them in YUV or other formats that are supported. Finally, we return the frame which is displayed on the widget on screen.

If we need to do some processing on the frame, we will do it in the onCameraFrame function and then return the modified frame to be viewed on the screen through our javaCameraView widget.

Everything is done save one thing. We haven't notified anywhere that the application uses camera to work. We do so by adding the permissions in "AndroidManifest.xml" file. We can open the file from the Project view in the folder "app/manifests/AndroidManifest.xml". Add the following code snippet at the start after the package declarations.

```
...
<uses-permission android:name="android.permission.CAMERA"/>
...
```

We can run the application now and after giving it camera permissions, we can see the application show us frames captured from the camera on screen.

Compiling OpenCV to use in JNI files

Sometimes we require to process data in C/C++. For the same, we take the help of Java Native Interface (JNI) which lets us compile native files written in C/C++ and use it with our Java codes. We will not get into detail about how to compile these files. You can do so by specifying the application that you will be using native functions in C++ and Android Studio will compile a project for you including a native C++ file called "native-lib.cpp". It also provides with a demo function about how to use the JNI to code in C++ and pass the processed data back to Java.

We want to focus on how to use OpenCV functions in the C++ files to performs image processing and vision tasks. We can do this using the "External Build Files" by specifying in the "CMakeLists.txt". We need to add three things in the "CMakeLists.txt" file as specified below.

1. **Add path to OpenCV4Android SDK**: We need to first add the path to the Opencv4Android SDK.

   ```
   include_directories(/home/OpenCV-android-sdk/sdk/native/jni/include)
   ```

2. **Add library and set target properties**: Next we need to specify the openCV library and set the target to the the compiled static libraries (.a, .so files) that we included in the "jniLibs" folder while importing the library in our application.

   ```
   add_library( lib_opencv
       SHARED
       IMPORTED )

   set_target_properties(lib_opencv
       PROPERTIES IMPORTED_LOCATION

   ${CMAKE_CURRENT_SOURCE_DIR}/src/main/jniLibs/${ANDROID_ABI}/libopencv
   _java3.so)
   ```

   The ${ANDROID_ABI} will take care of all the platforms required by the application. We just need to ensure the static libraries for the required platforms are copied to the "jniLibs" folder.

3. **Link the libraries**: Once we have specified the target libraries, we need to link it to our application.

```
target_link_libraries(
                       native-lib
                       lib_opencv
                       ${log-lib}
                       ${android-lib}
                       )
```

The "lib_opencv" will link the OpenCV library to our application and the JNI. We can now use OpenCV functions in the "native-lib.cpp" file.