Developing Android Apps with OpenCV for Windows

EEE508

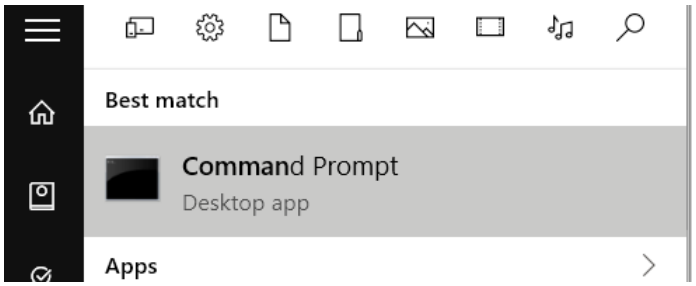
Part A: Installing Android Studio

This tutorial has been modified for Windows from the original tutorial for Macs listed in Reference [1].

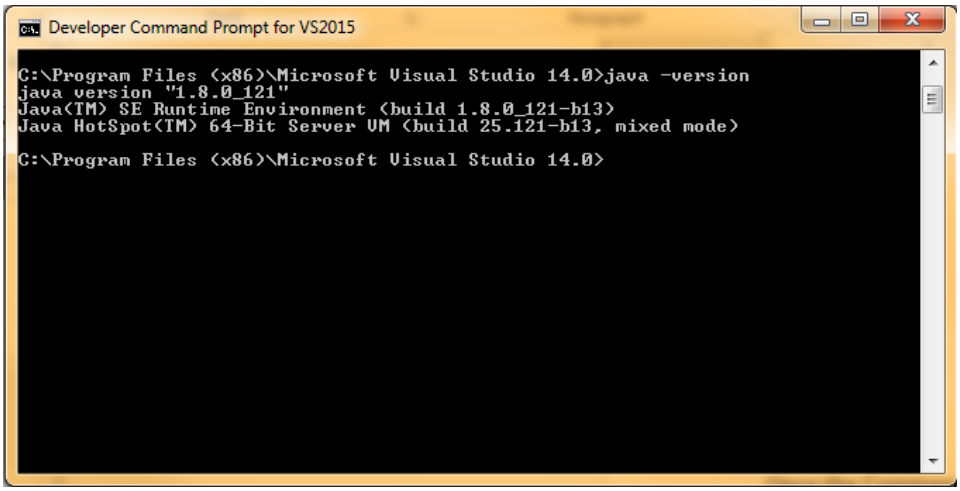
One of the most important parts of getting started with any new platform is setting up your environment. It is important to take your time and follow each step methodically. Your system configuration or product versions might for unexpected results.

First you need to check that you have the Java Development Kit (JDK) installed. For this purpose, you can make use of the Command Prompt (CP).

You can find the CP app quite easily on a PC: open Windows Search by pressing the “Windows” key and type command prompt into the search at the top of the screen and select Command Prompt when it shows up.



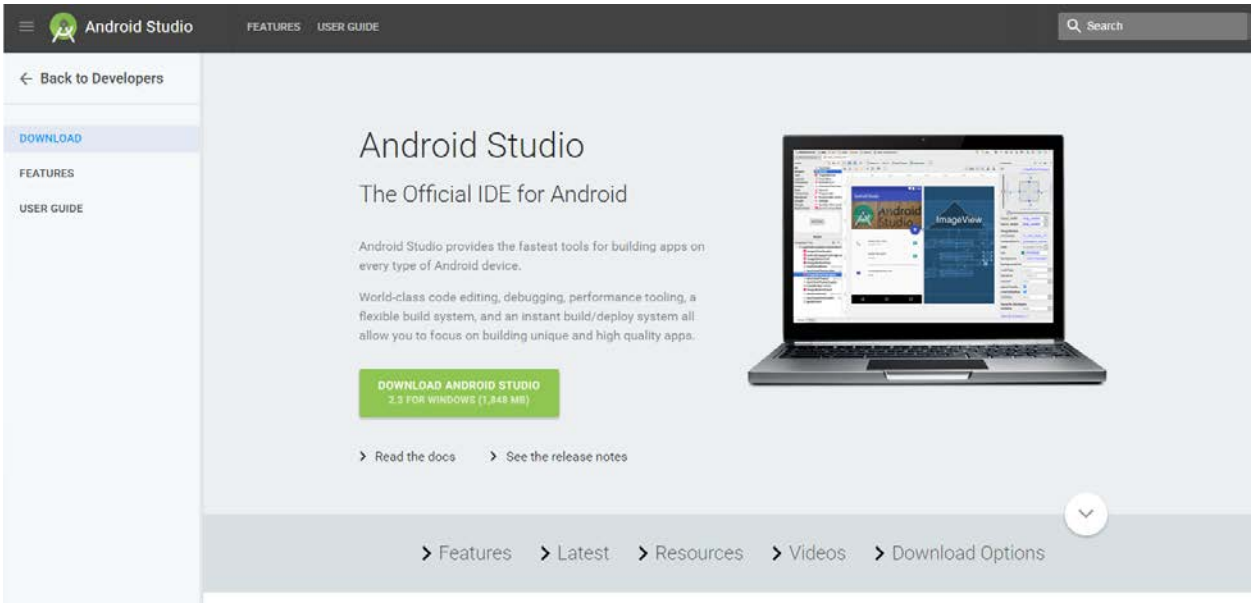
Once the Command Prompt is open, type java -version and press **Enter**. You should see some output that mentions a version number, like below.



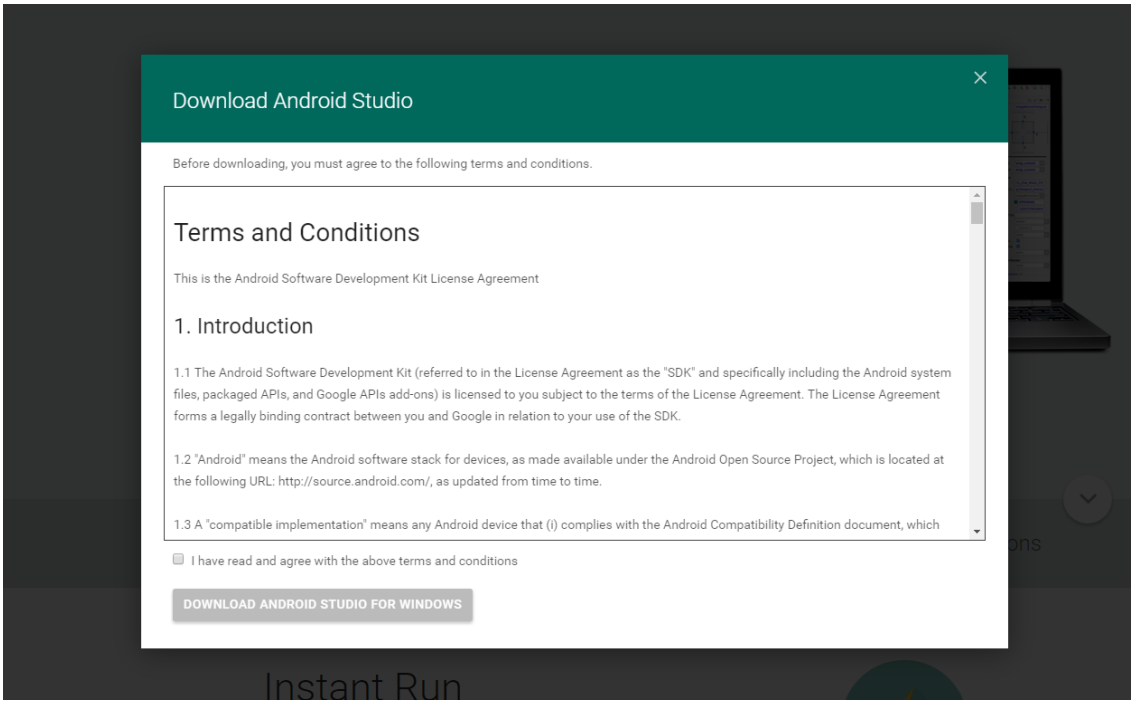
If that’s not what you see, then you don’t have the JDK installed. CP might tell you -bash: java: command not found, or it could say No Java runtime present, requesting install.

You will need to head over to Oracle to download the JDK from Oracle.

Install the JDK if needed, and once you are done, head over to the Android Studio page and click the **Download Android Studio** button.

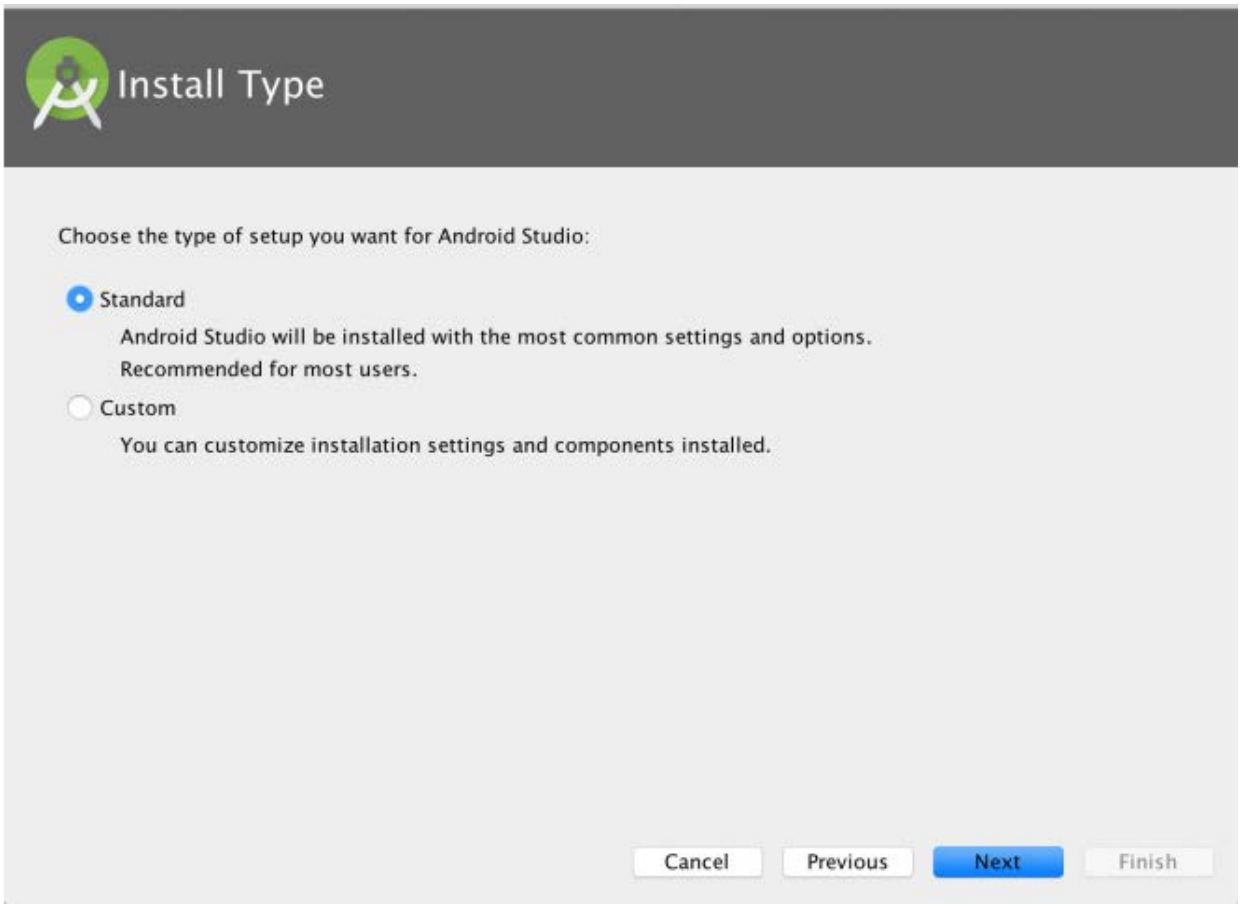


Once you click the button, you'll see a request to agree to the terms and conditions.



After reading the terms and conditions, click the button underneath titled **Download Android Studio 2.3**. Once the download is complete, you can install Android Studio.

When installing Android Studio, make sure to choose Standard in the Install Type screen.



Once installation is completed, launch Android Studio.

The Android SDK Manager

Each version of Android has its own SDK (Software Development Kit) that enables you to create applications for the Android platform. If you just went through the setup wizard, you will have the latest version of the SDK.

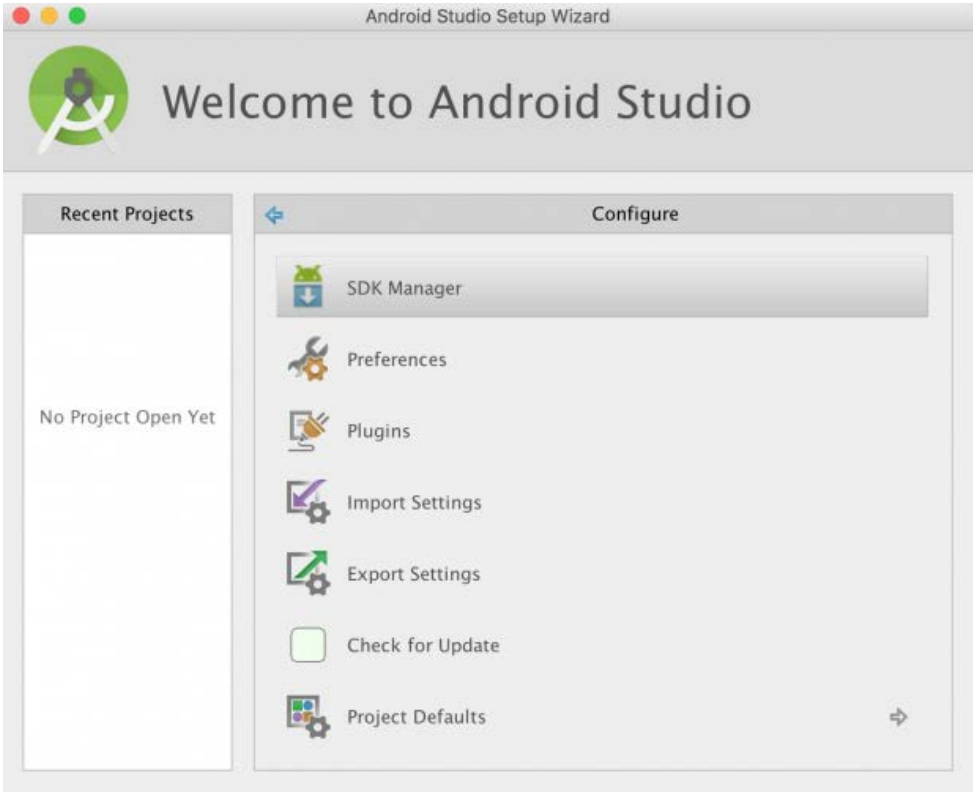
Installing a New SDK

It is useful to know how to install additional versions of the SDK so that you can develop for all supported versions of Android.

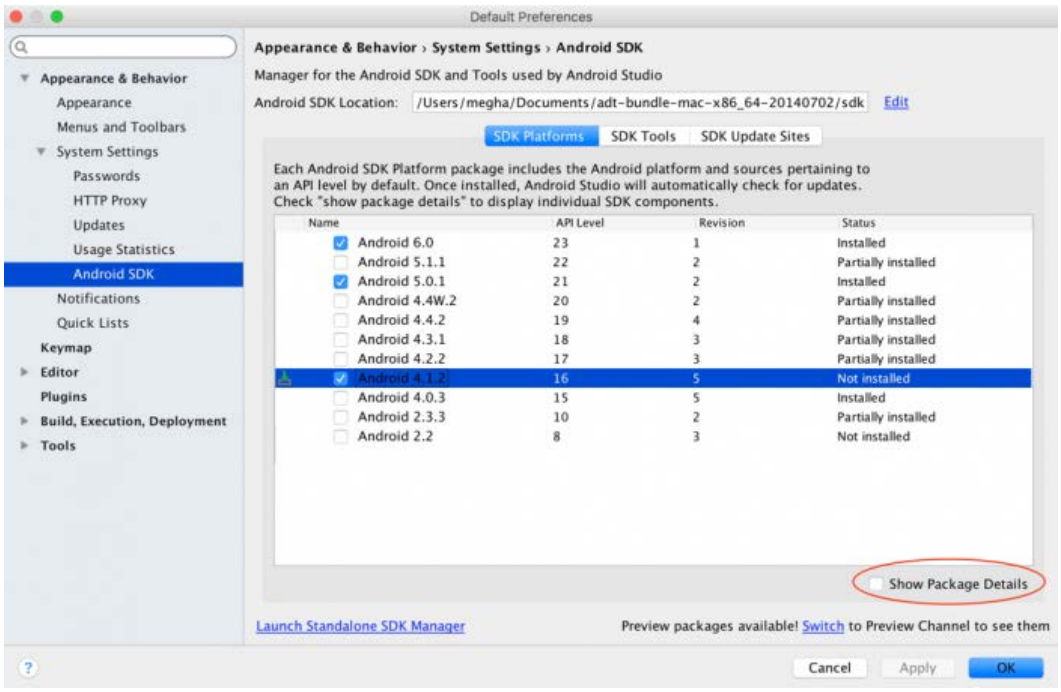
SDKs also allow you to create AVDs (Android Virtual Devices) that are customized to your personal configuration for the purpose of testing your app.

From the Android Studio welcome screen, click Configure.

The menu will slide across and present the Configure menu. Select the SDK Manager option.



Once it launches, you'll see a window like the one below:



The first tab of this window, **SDK Platforms**, lists the Android SDK platform available for download.

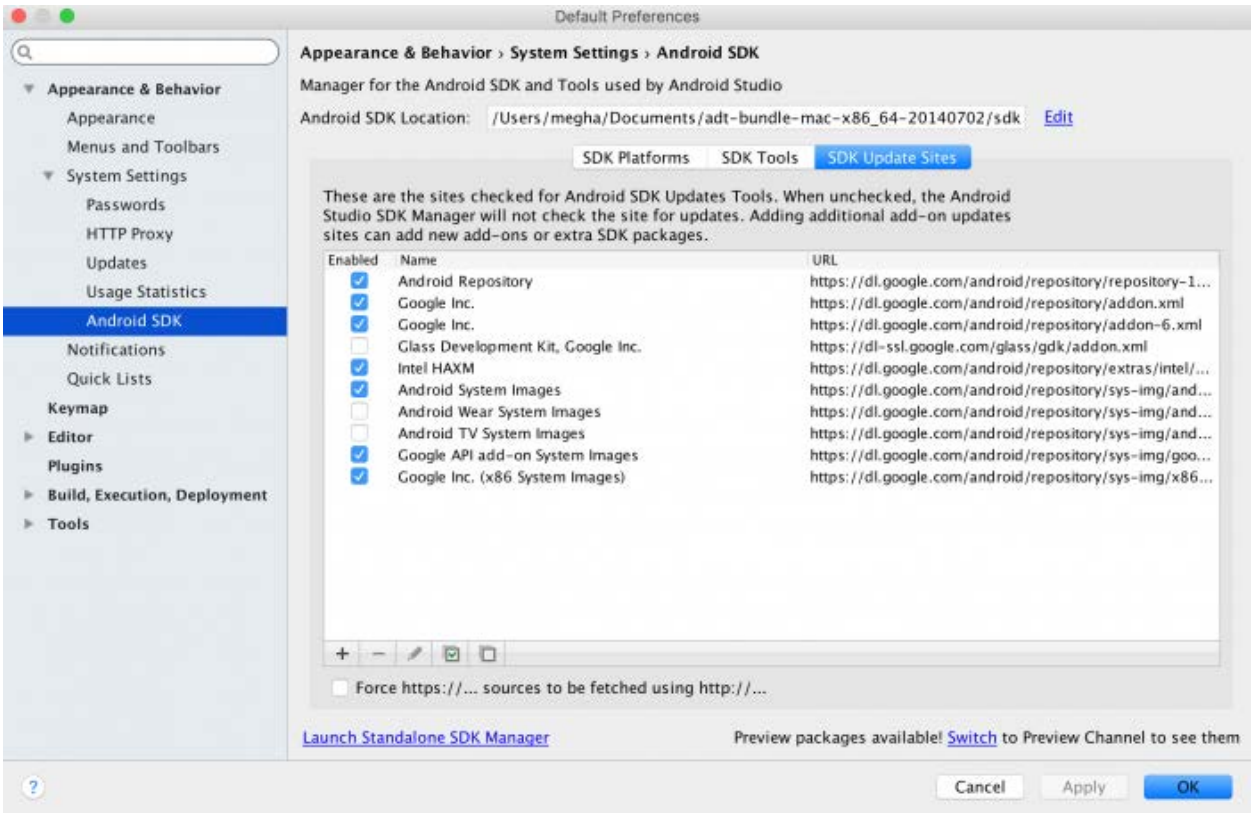
Make sure to have Android 7.1.1 marked.

Enable the **Show Package Details** option to see individual SDK components, such as the platform itself and the sources pertaining to the API level like system image. Take note of the checkbox next to the SDK platform; it will be pre-selected if an update is available.

By default, the SDK Manager installs the latest packages and tools. Select the SDKs as shown in the screenshot above. If you wish to install other SDKs, just select them for installation.

The **SDK Tools** tab lists developer tools and documentation along with the latest versions. Similar to the first tab, checking the **Show Package Details** will display available version of SDK tools.

The first three components in this list, for example, are **Android SDK Build Tools**, **Android SDK Tools** and **Android SDK Platform-Tools**. Each contains components that are designed to assist in the development of Android and work across multiple SDKs. Go with the default selection on this tab.



The **SDK Update Sites** tab displays the update sites for Android SDK tools and add-ons. You're not limited to what's listed because you can add other sites that host their own Android SDK add-ons, and then download them from those sites.

Select the options that are checked in the screenshot above. Click **Apply** at the bottom. You'll be presented with a confirmation dialog for the chosen packages; accept and continue. Click **OK** to close out the window.

The window will disappear and the SDK Manager will download and install the selected items. Once it's done, click **Finish**. You'll be directed back to the SDK Manager window where clicking **OK** will take you back to the **Welcome to Android Studio**.

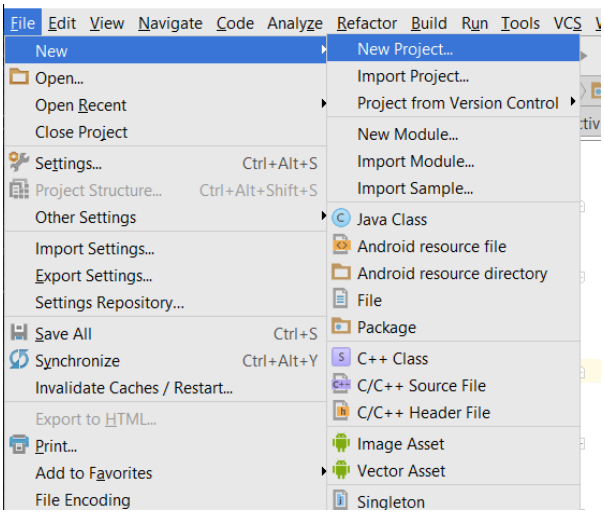
Extra Note: Reset Android Studio

If you want to Reset Android Studio settings and start over please refer to the following links

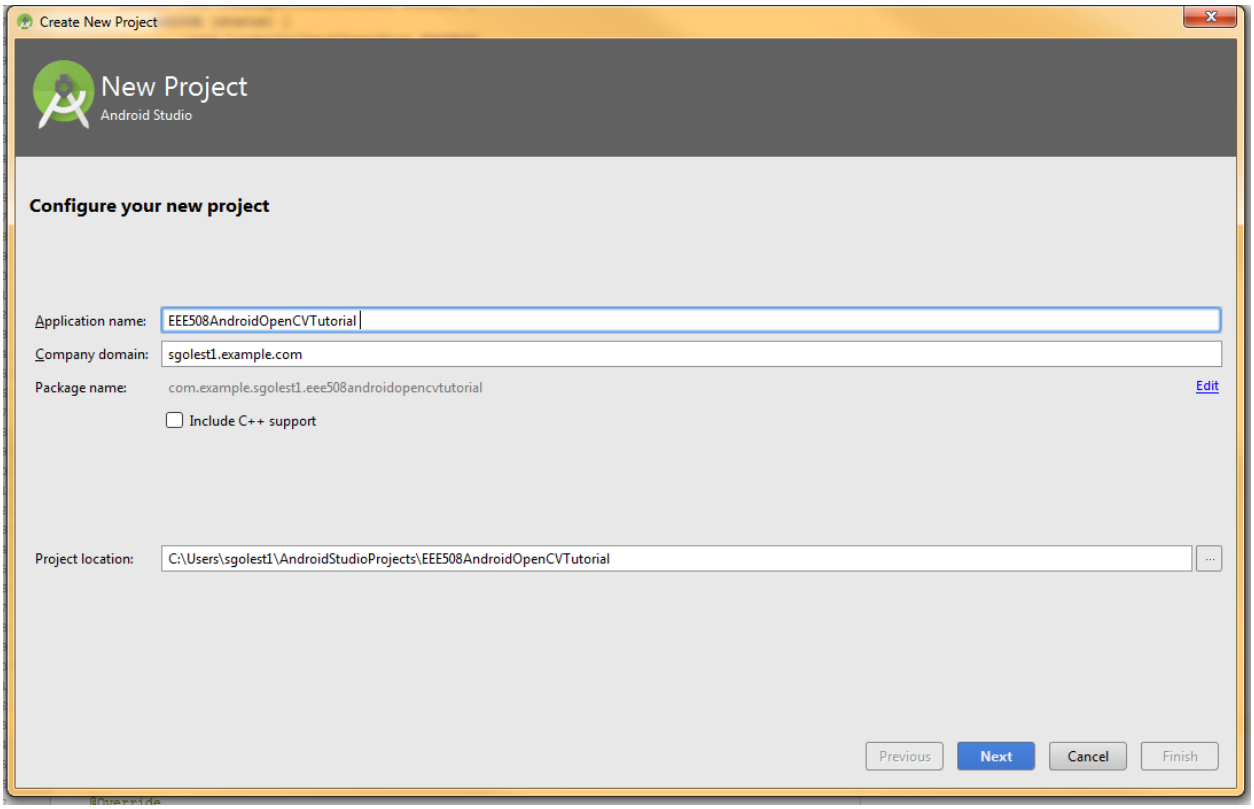
https://www.youtube.com/watch?v=AwZEliSPGwU

http://stackoverflow.com/questions/19384033/how-to-reset-android-studio

Create a new project.

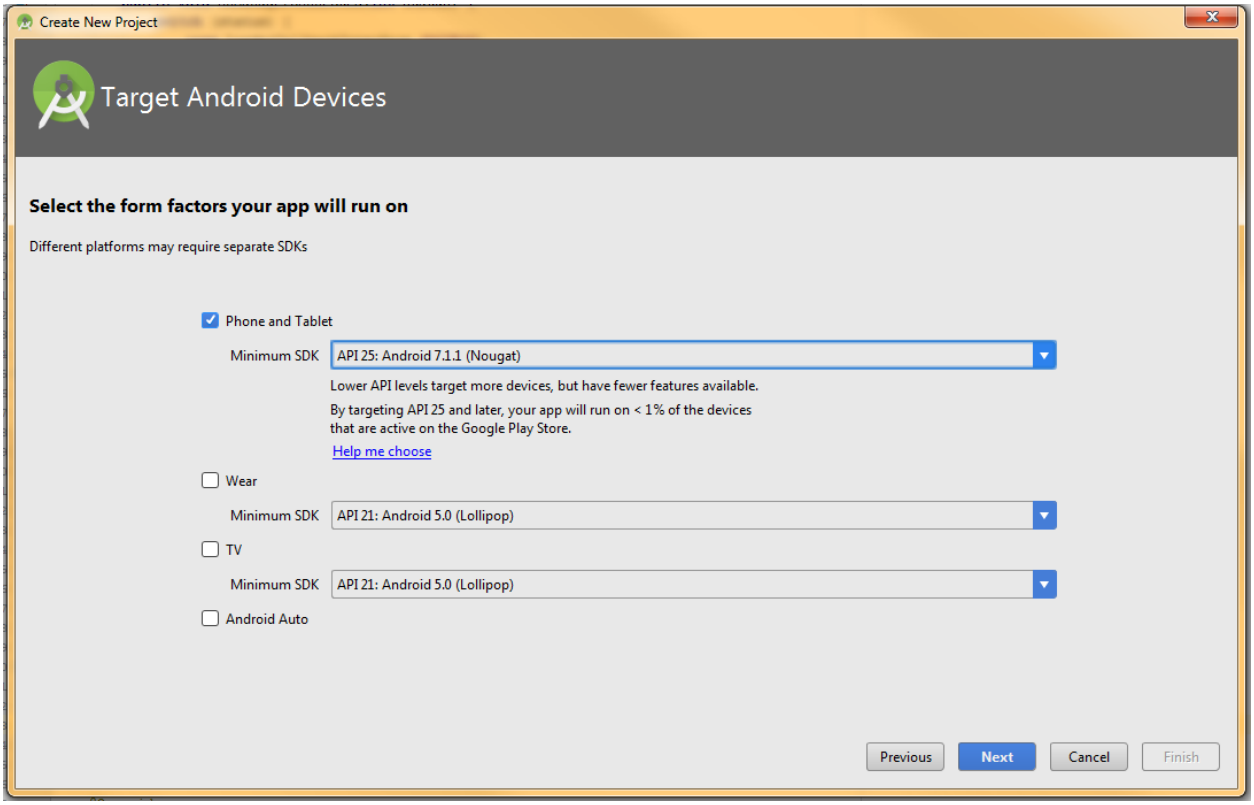


We call it EEE508AndroidOpenCVTutorial and define a "company domain" (you can choose to keep the default name).

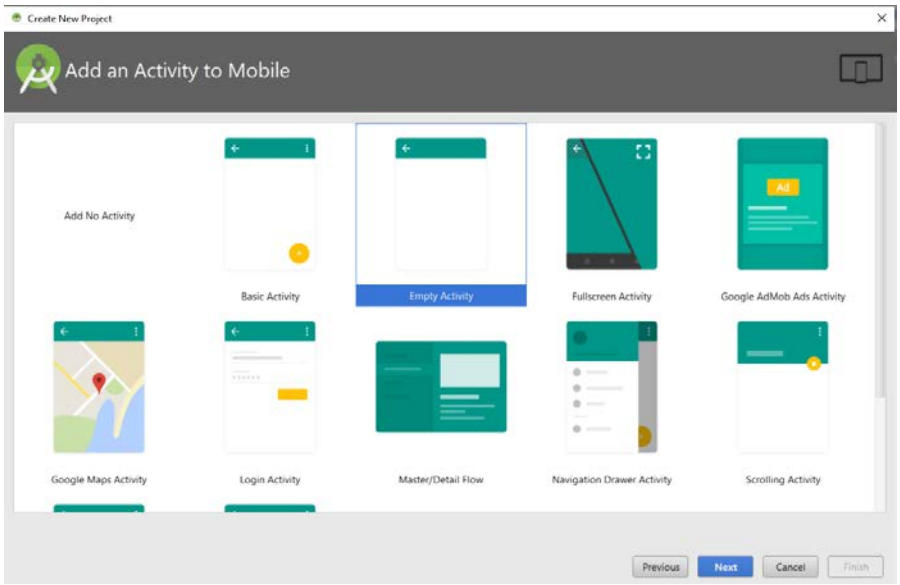


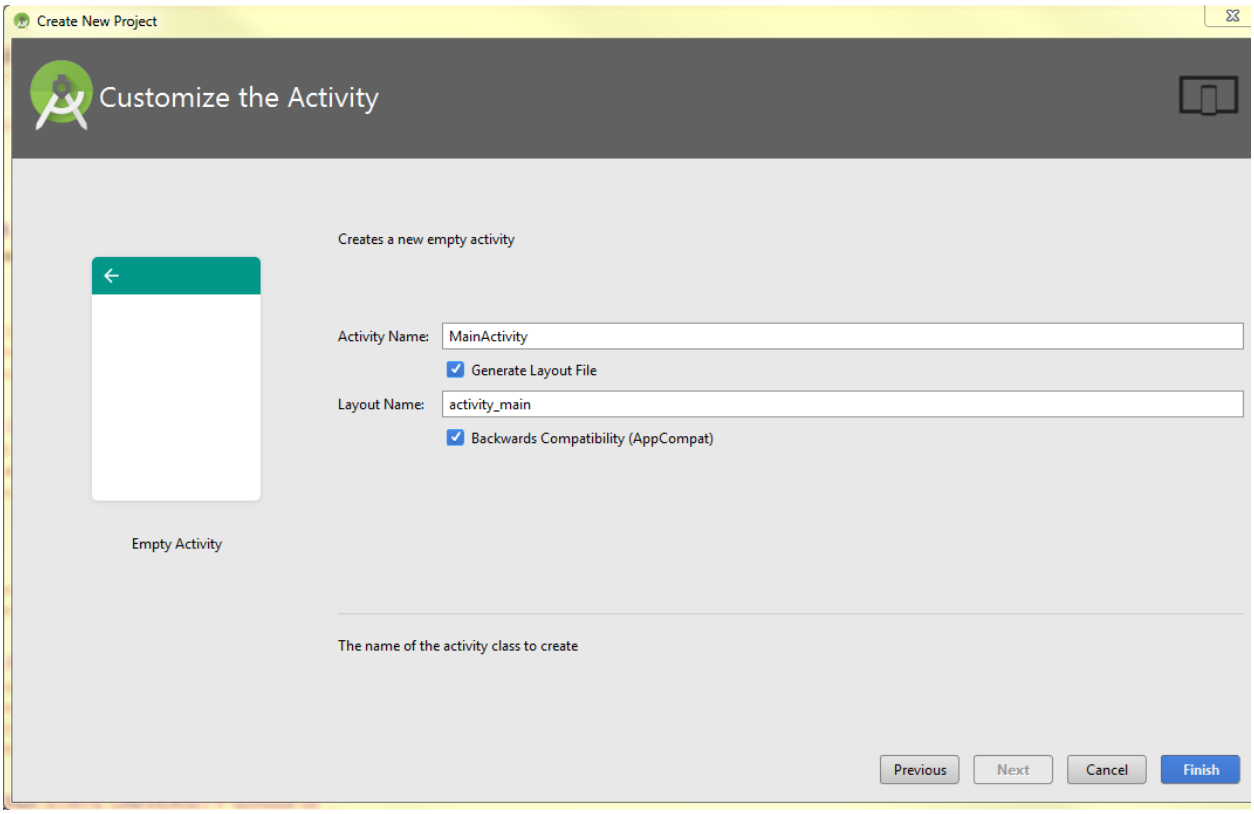
We will be using the android.hardware.camera2 package from the Android SDK. This requires a minimum SDK defined as Android API 21 or later. Usually for simulation on PC the newest version is preferred, but some of the new versions may not be compatible on phones with older versions.

Go ahead and set the minimum SDK to Android API 25 Android 7.1.1 (Nougat) for phone and tablet development.

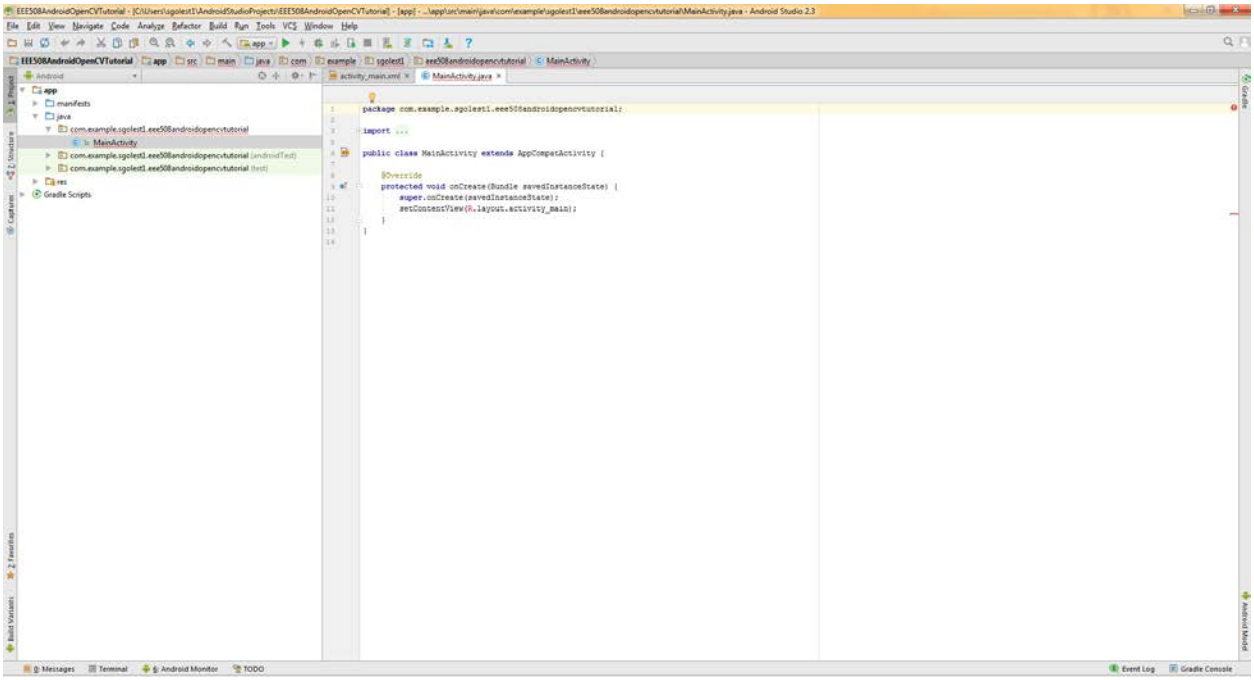


We will start from scratch, but initialize our project with an empty activity. Call that activity MainActivity, which should be the default name in Android Studio.





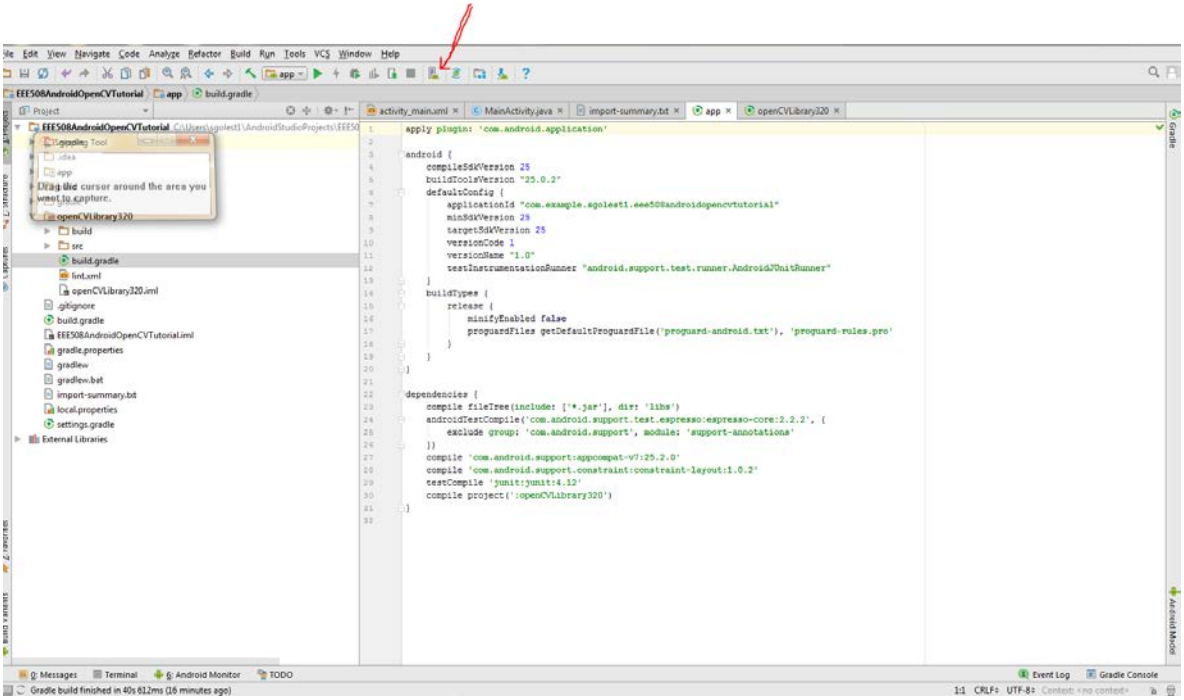
The project will initialize and leave us with a relatively empty project looking like this:

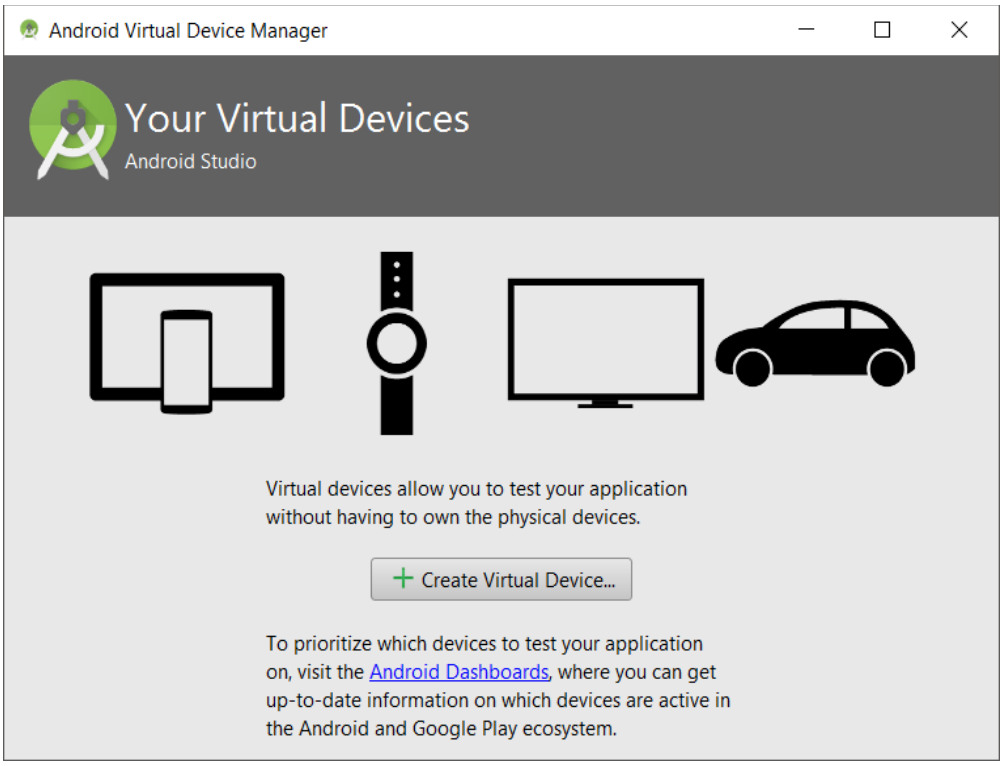


Creating a Virtual Device

Our virtual device will use the built-in webcam on your PC as the emulated phone's back camera. We will use a fully emulated camera for the front camera, which we will not use with OpenCV in this tutorial.

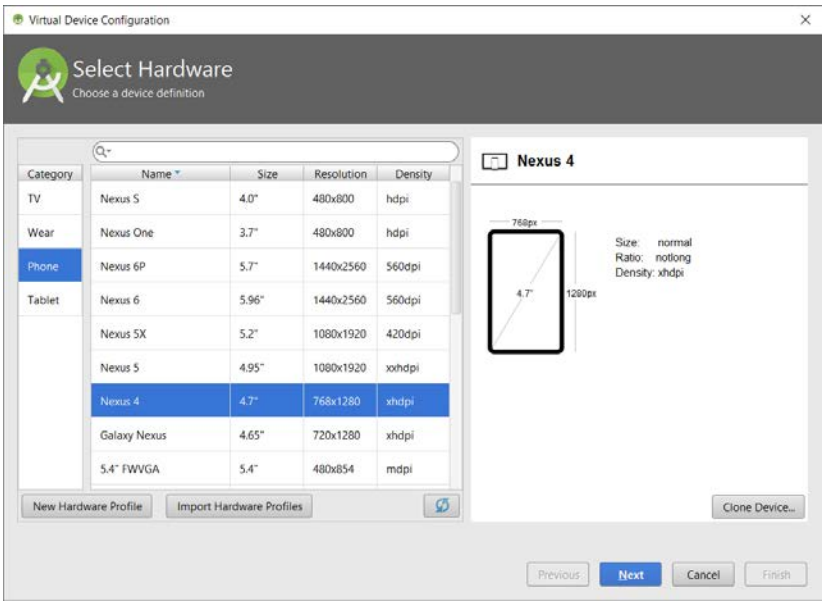
Using the Android Virtual Device Manager, let's create a new virtual device using the AVD icon pictured below.



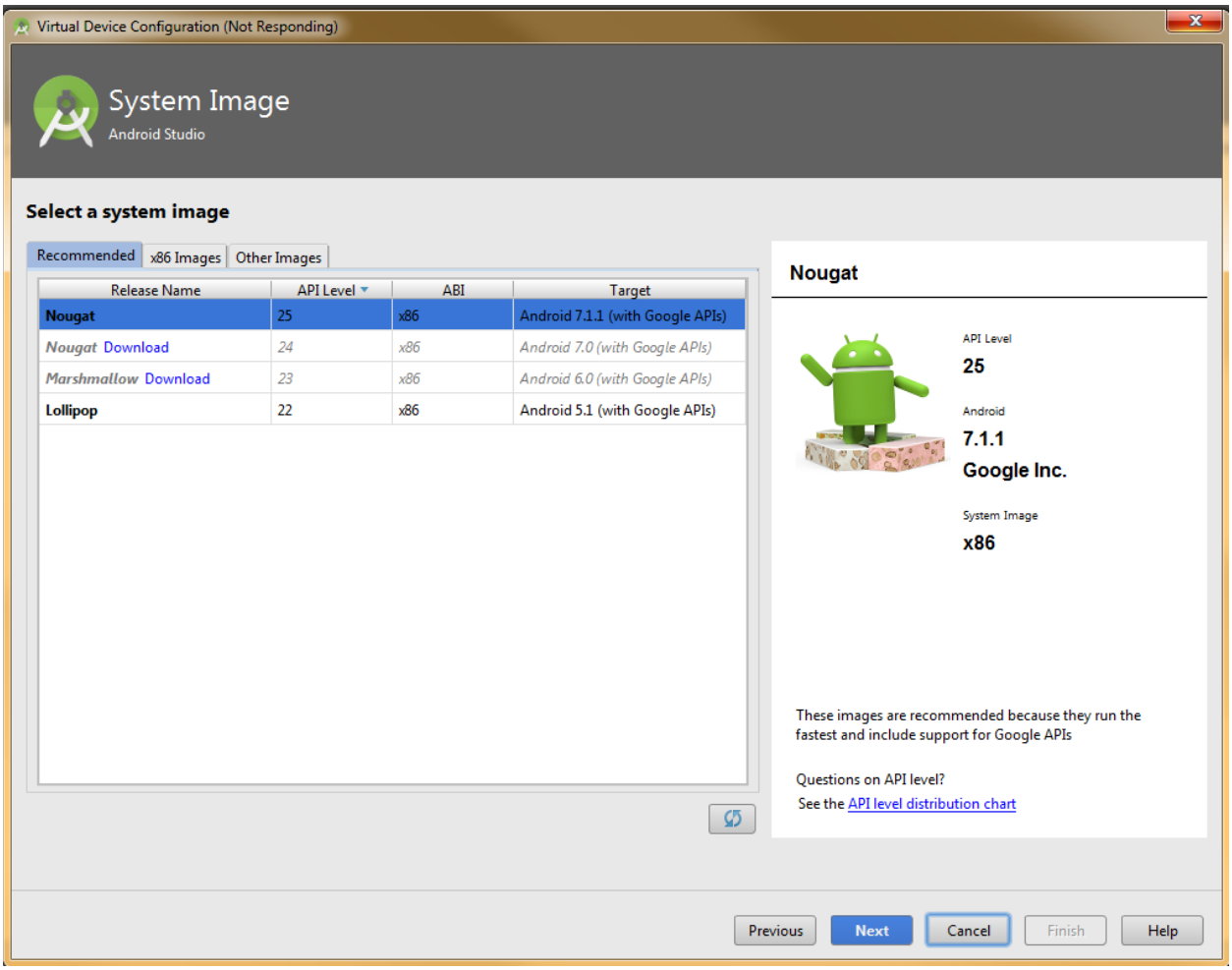


Running an Android emulation is quite memory intensive to forewarn you. Let's go ahead and create a new virtual device.

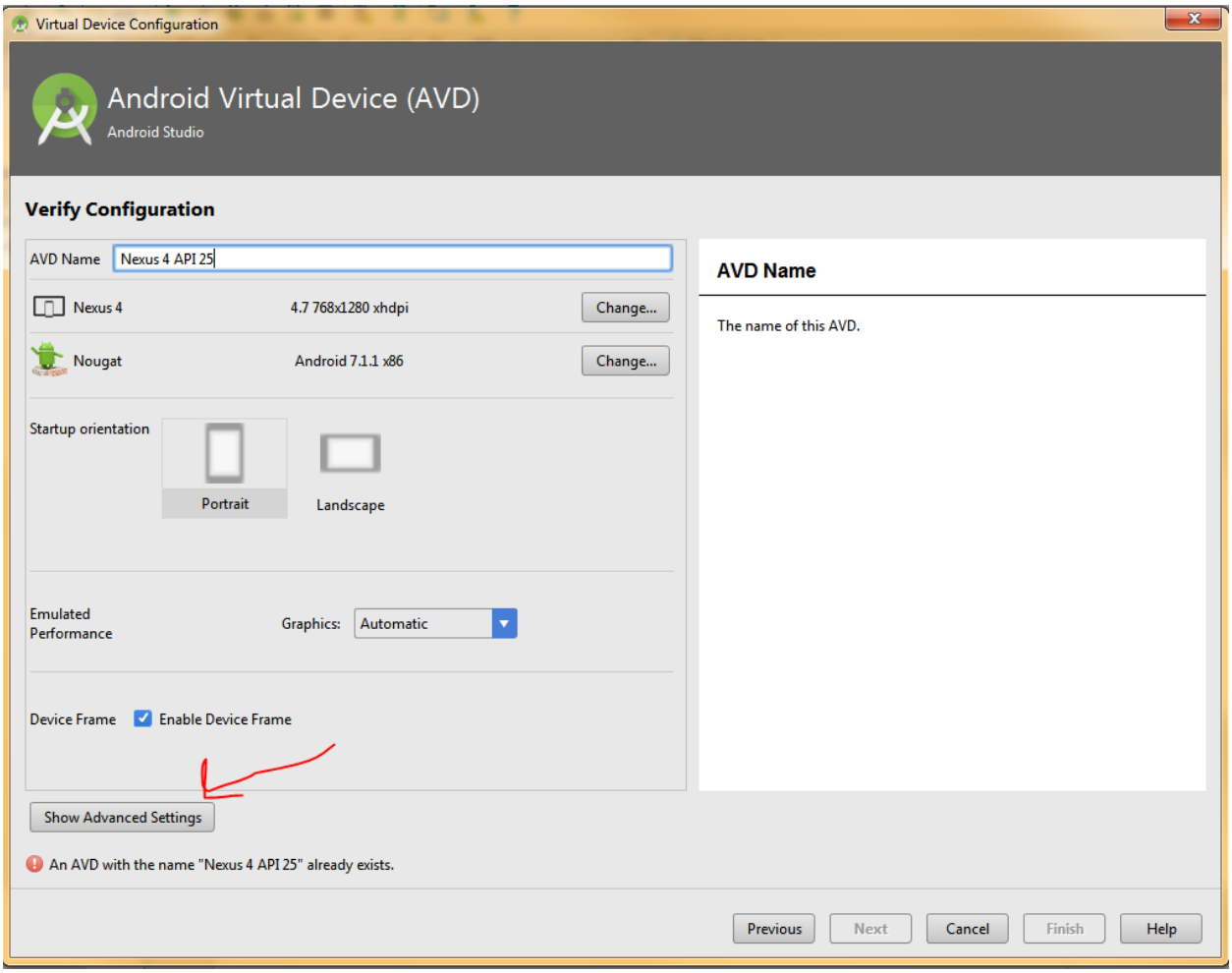
For the sake of this example a virtual Nexus 4 will be created.



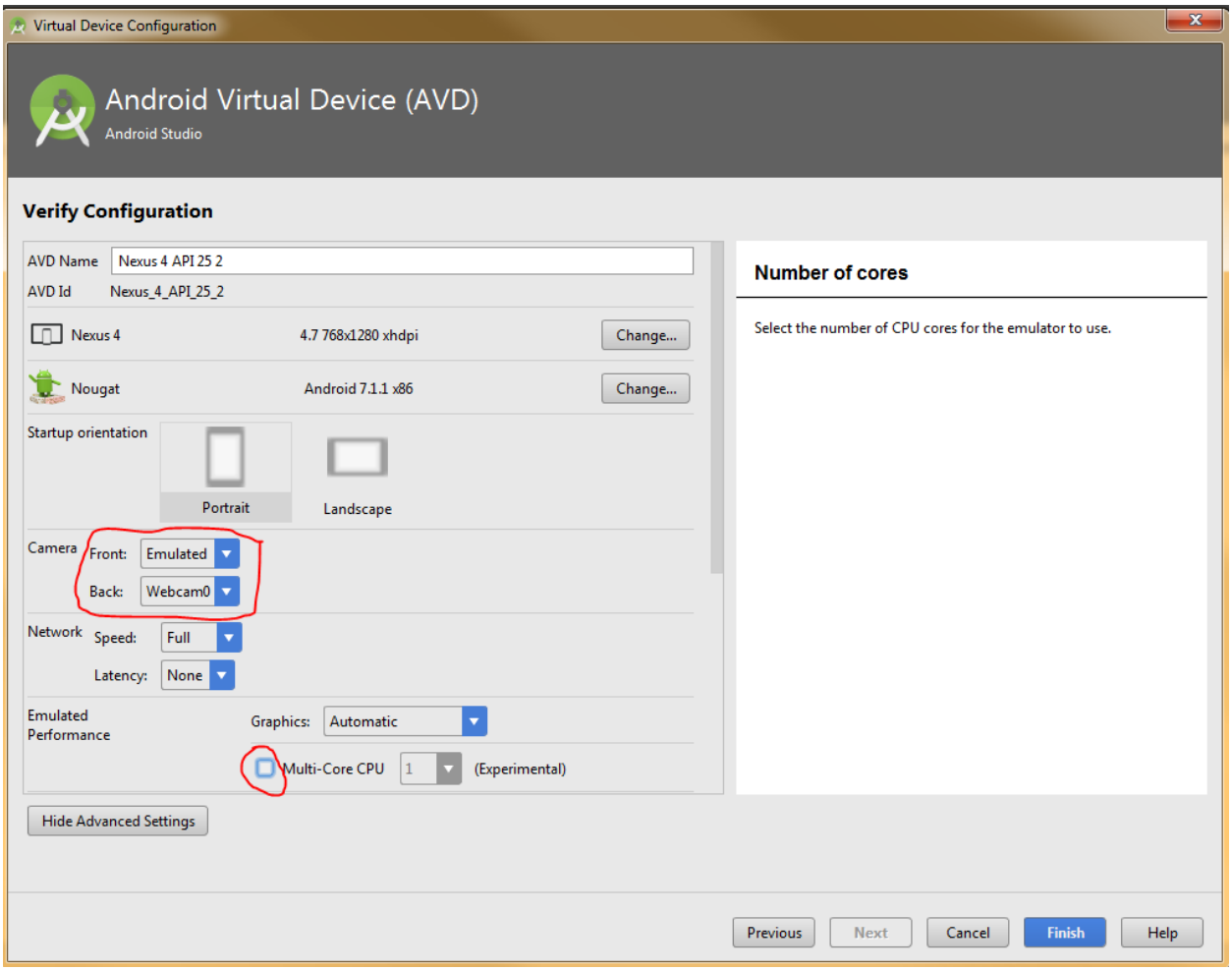
Now we want our emulated Nexus 4 to run Android Nougat 7.1.1 using an x86 system image.



You can leave the default settings on the main page of the device settings, but we need to activate the camera for our emulation to use later with OpenCV. Click Next to go to the next step where you can open the advanced settings



Select the appropriate camera settings for the front and back cameras. The choices below show that we will be emulating the front camera and using an actual physical camera for the back camera. The physical camera can be a built-in phone camera if an actual phone is connected to the computer or can be a connected webcam if the phone is being emulated.

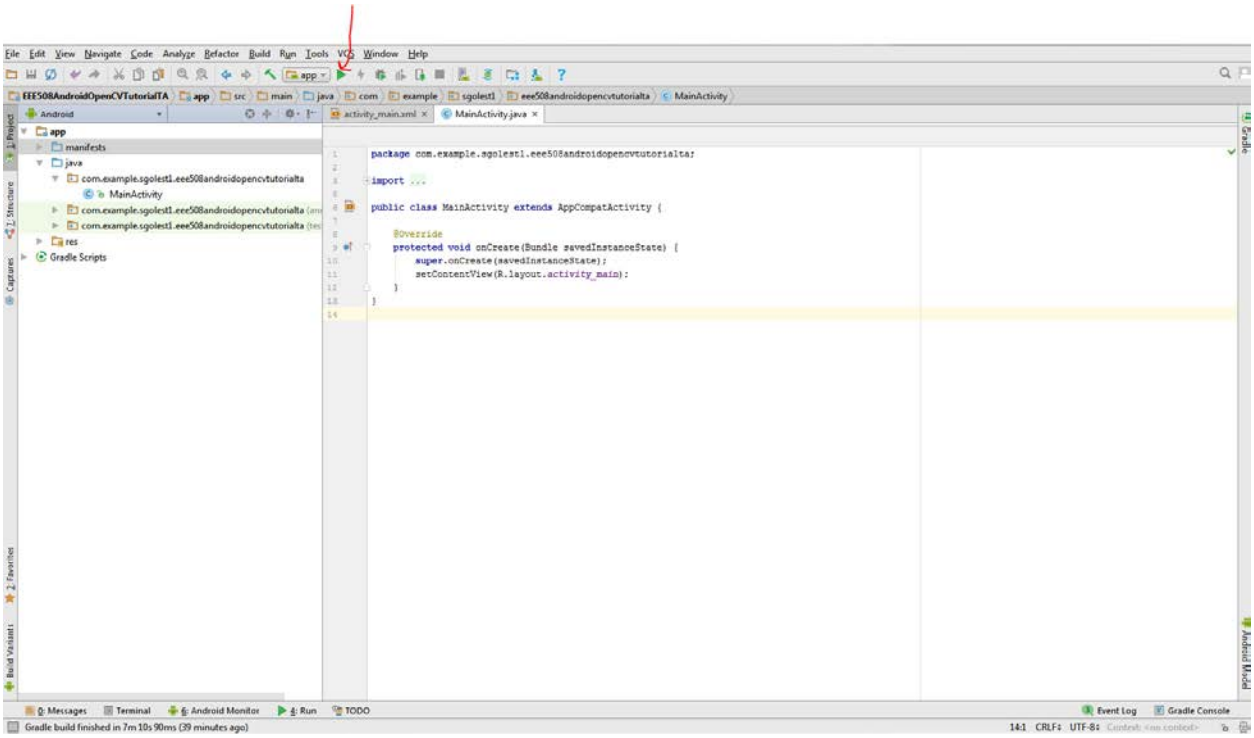


Hit Finish. It may take a minute for our new virtual device to be saved.

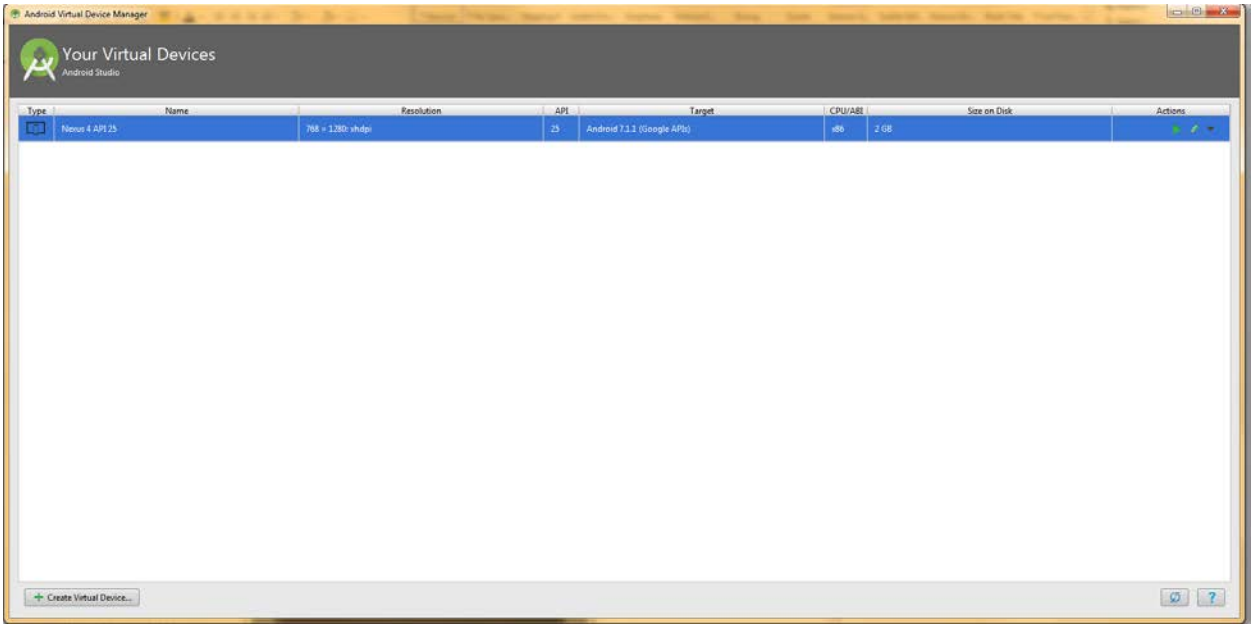
Once the virtual device is successfully saved, you can close the AVD Manager.

Now let us start our virtual device up by running our app using the run icon.

Later in this tutorial we will emulate the front camera and use a built-in webcam as the back camera for OpenCV.



A window opens up, select the device that you would like to be emulated as shown below.



The virtual device will boot up as if you had clicked the power button of a real device for the first time.

This shows a sample Hello World App that is present by default in Android Studio. In order to modify the app display you need to open activity_main.xml found in path app/main/res/layout (or app/src/main/res/layout).

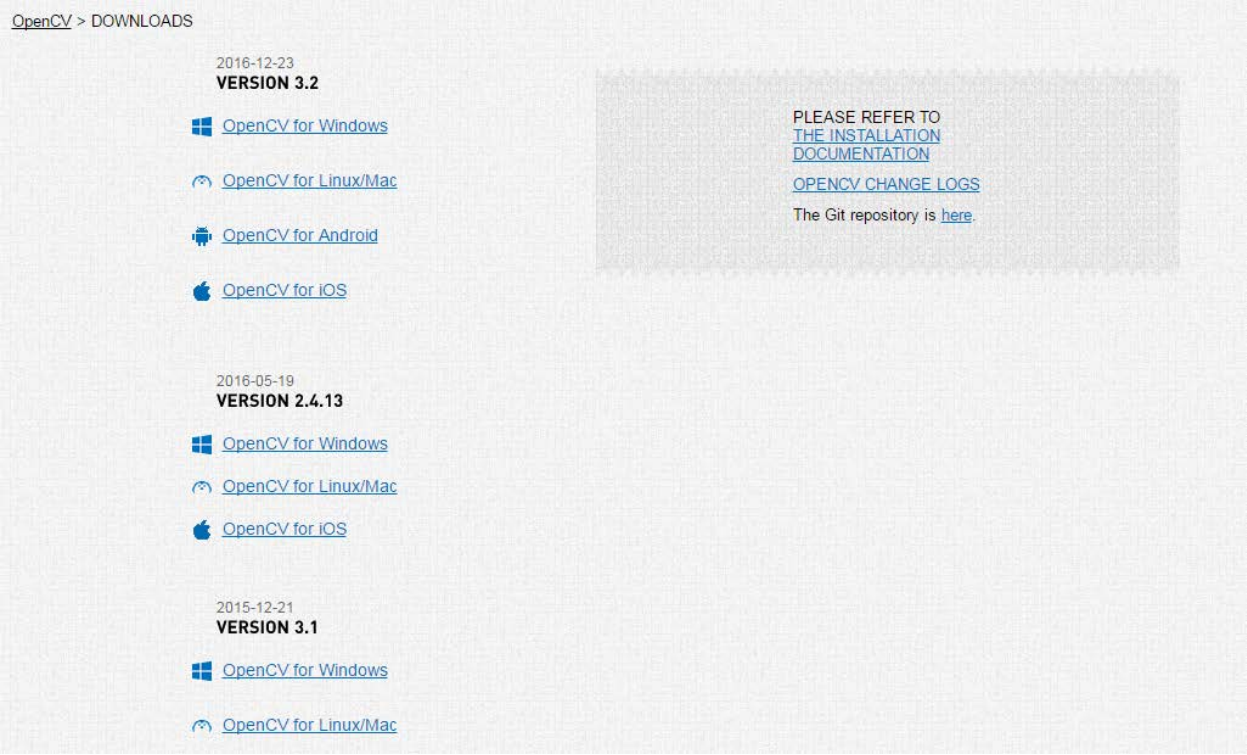
Now, modify the app to display another text of your choosing including your names.

Part B: Installing OpenCV for Android

This part of the tutorial has been modified from a tutorial for iOS listed in Reference [2]

Install the most recent version of OpenCV for Android (http://opencv.org/releases.html).

In our case it would be OpenCV 3.2 for Android



Download and unzip the zip file containing the SDK. This folder contains the Android application package (APK) files for OpenCV Manager, which we will address below, sample OpenCV applications for Android, and the OpenCV SDK itself.

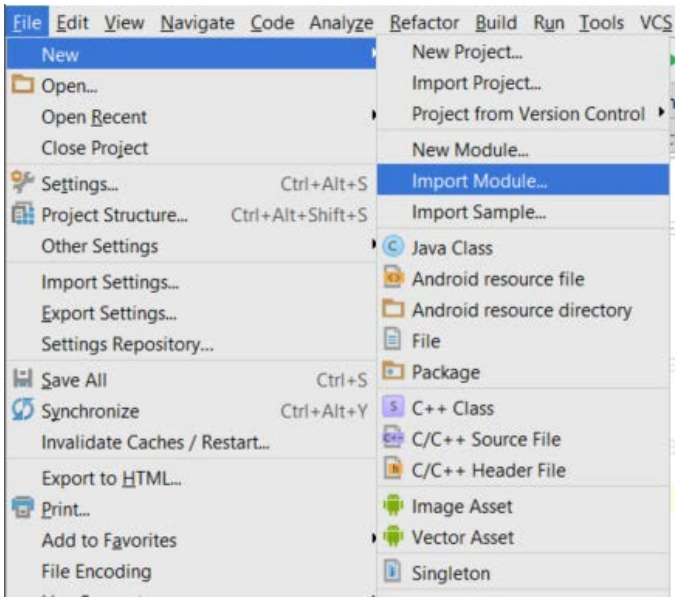
Now that we have the OpenCV library, we need to start a new Android Project and learn how to use the OpenCV library with an Android app. Open Android Studio.

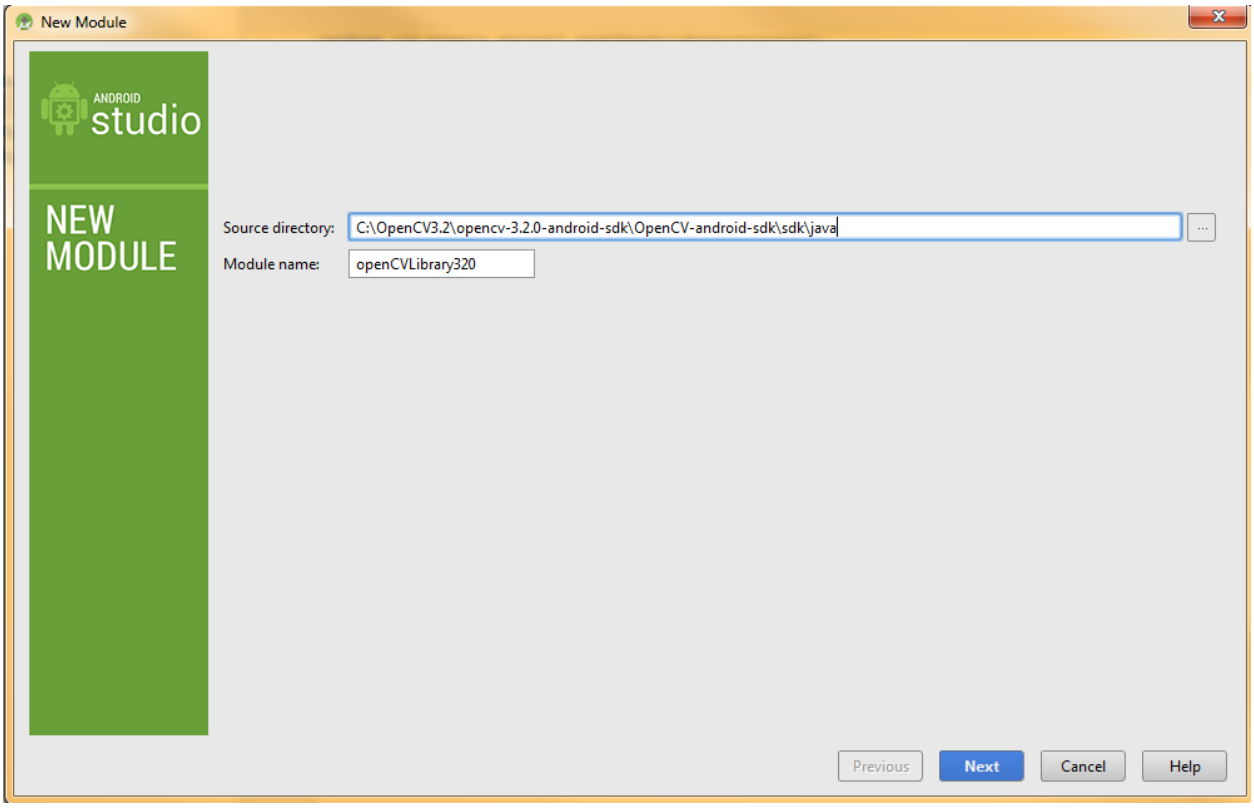
Import OpenCV Module

To import OpenCV into our project, we import OpenCV as a new module.

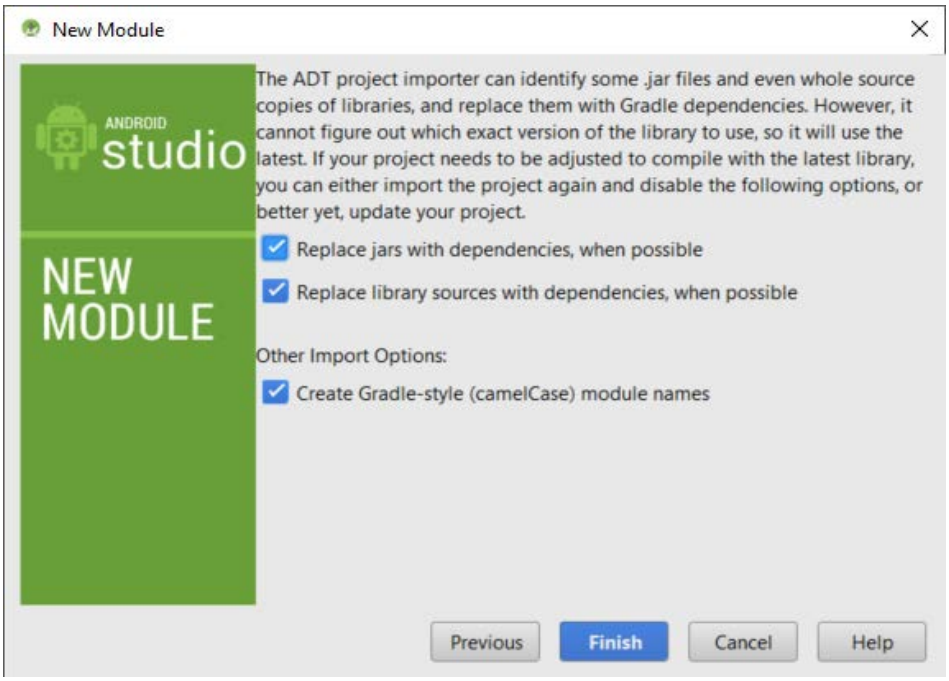
We do this with File → New → Import Module...

Navigate the source directory to where you unzipped the OpenCV SDK and add the sdk/java folder as the module to be imported.





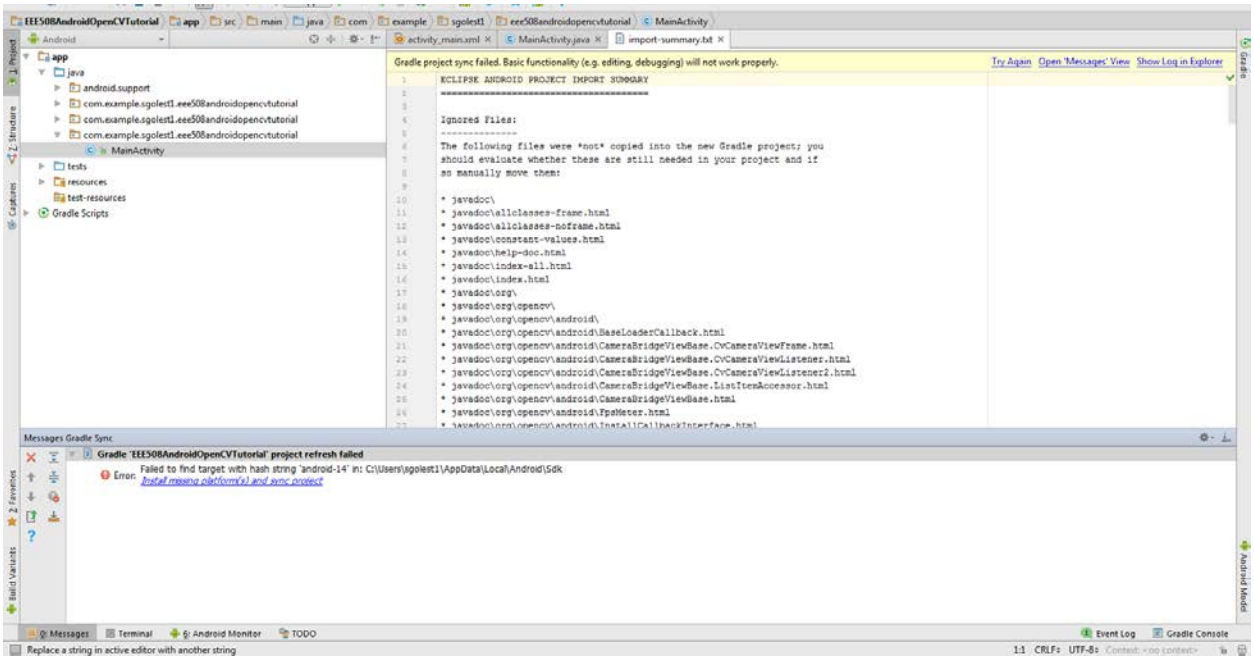
Leave the default settings checked and import the OpenCV library by clicking on **Finish**.



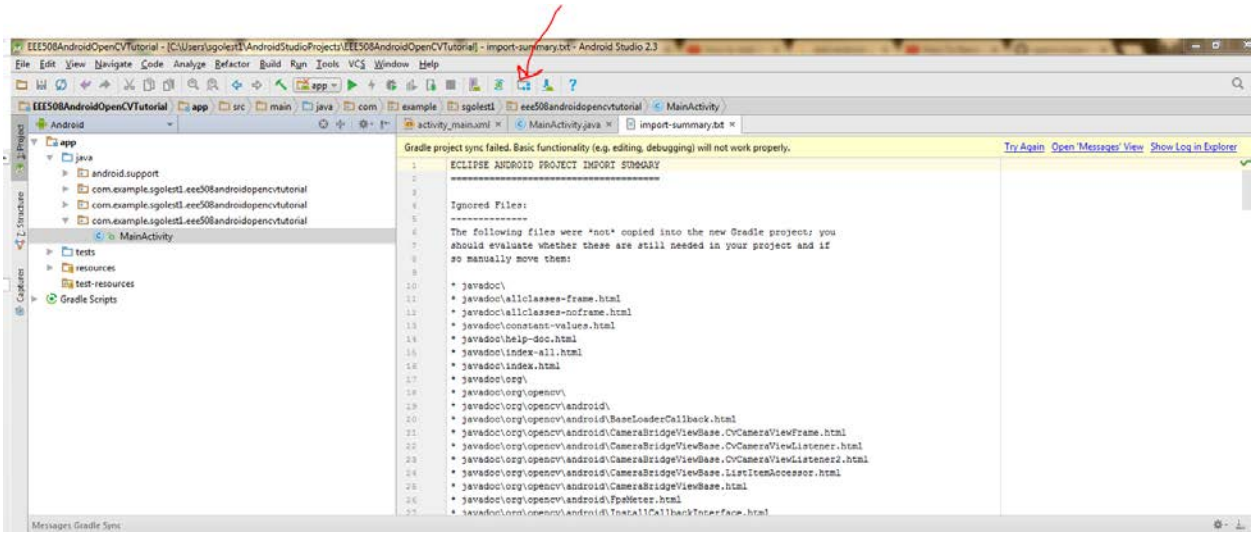
Now, you will see a summary of the module import. You will see that a few things seem broken or cannot be resolved in the OpenCV library from the initial import

You may also have an error message saying *failed to find target with hash string 'android-14'*.... This may happen because the build.gradle file in the OpenCV zip file you downloaded says to compile using another android API version 14, which might not be included with the installed Android Studio version.

This has to do with the default Android API expected by the OpenCV build, which we will fix below.

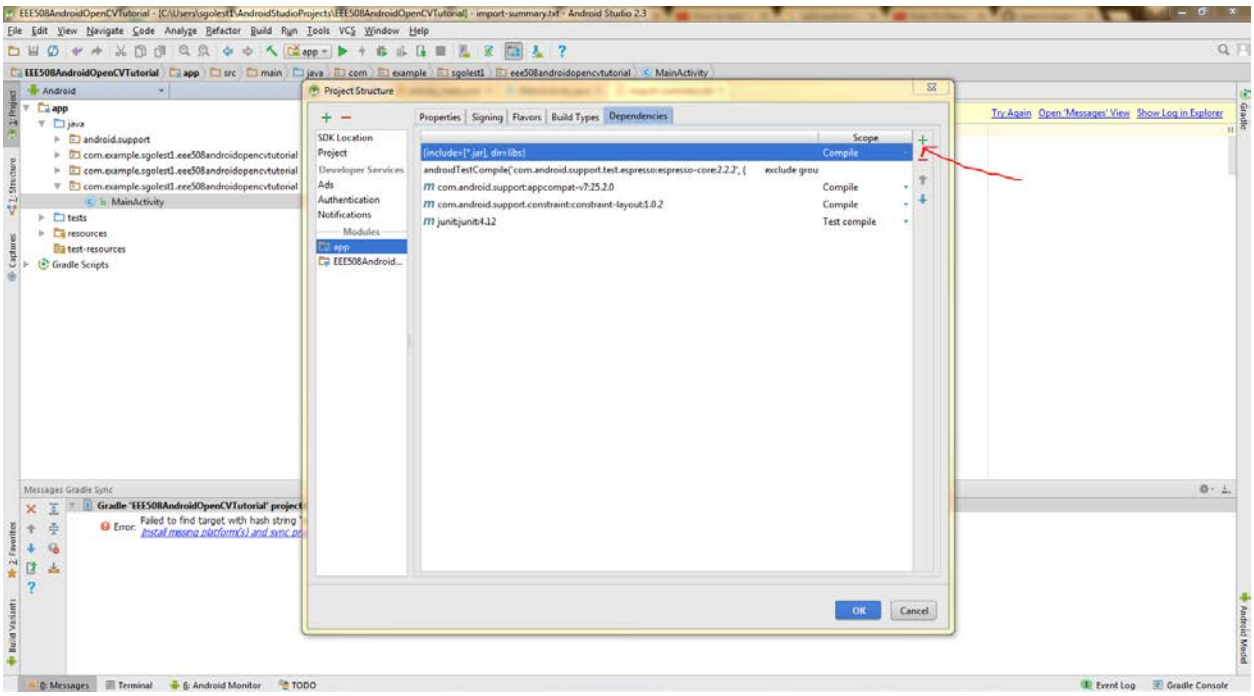


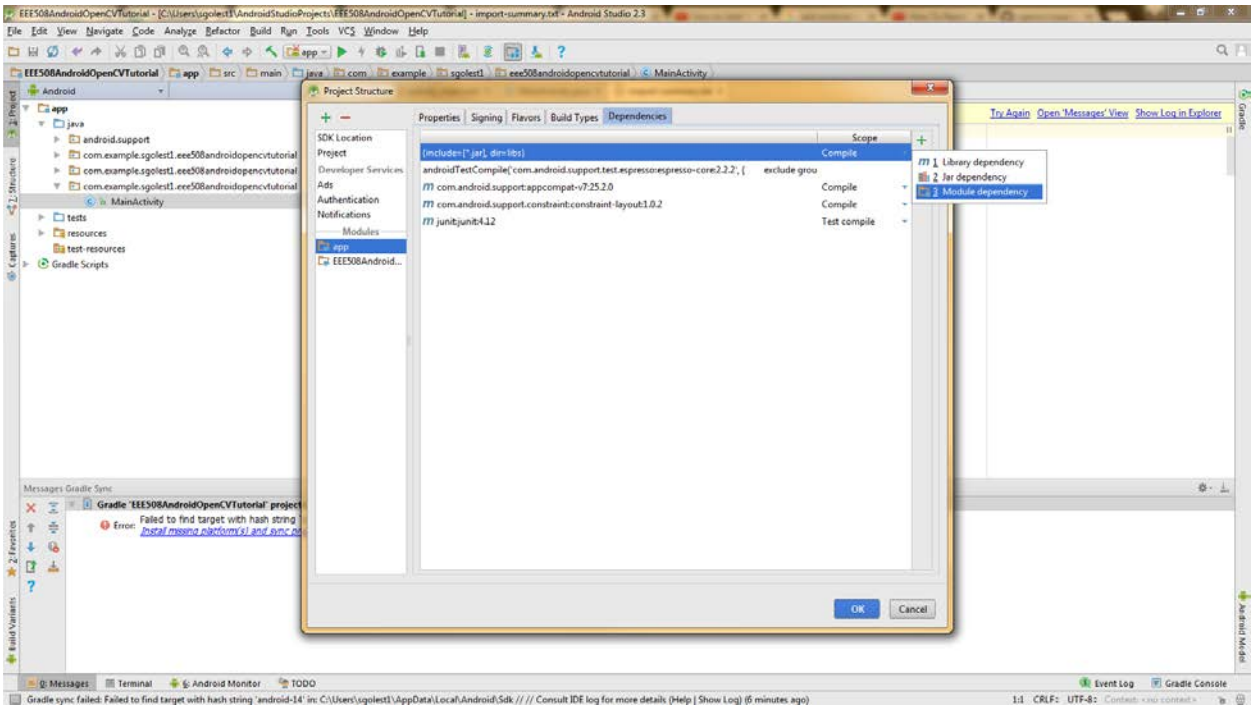
Open the project structure dialogue (*Menu:/File/Project_Structure*).

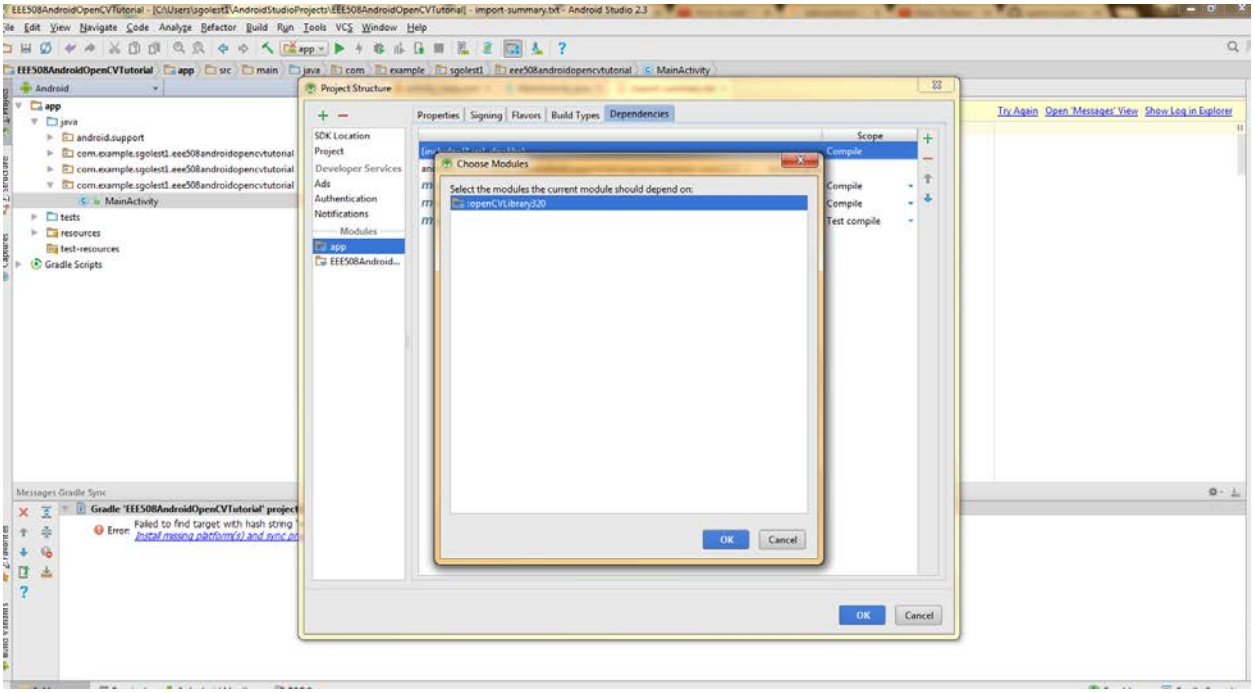


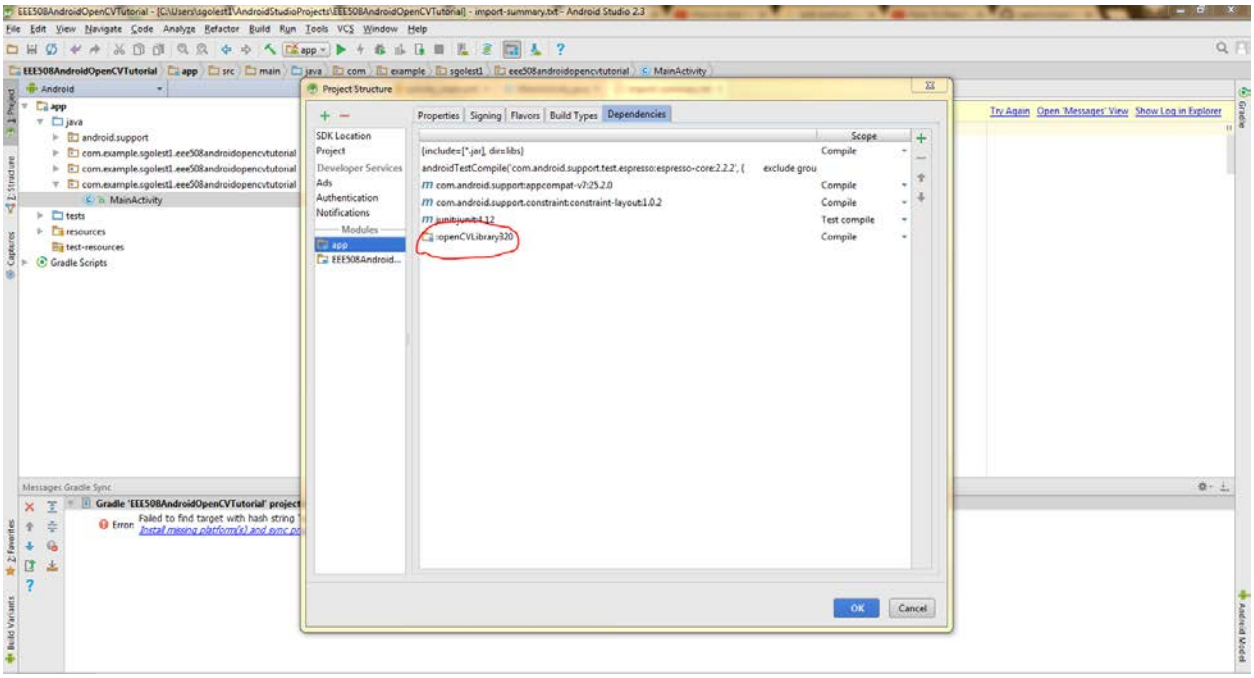
Select the "app" module, click on the *Dependencies* tab and add

:openCVLibrary320 as a Module Dependency. When you select *Add/Module_Dependency* it should appear in the list of modules you can add. It will now show up as a dependency, click okay, It may take a couple of mins. You will get a few more *cannot-find-android-14* errors in the event log.




Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Developer Services
Ads
Authentication
Notifications
Modules
app
Scope
Compile
Test compile
1 Library dependency
2 Jar dependency
3 Module dependency
OK
Cancel


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
Choose Modules
Select the modules the current module should depend on:
:openCVLibrary320
OK
Cancel

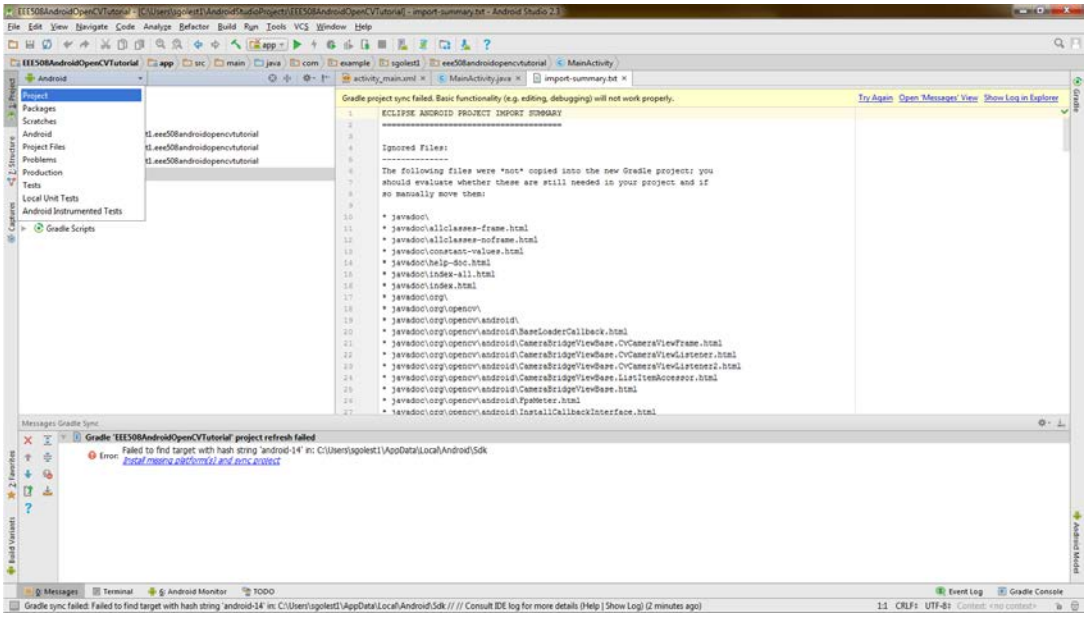


Modify build.gradle

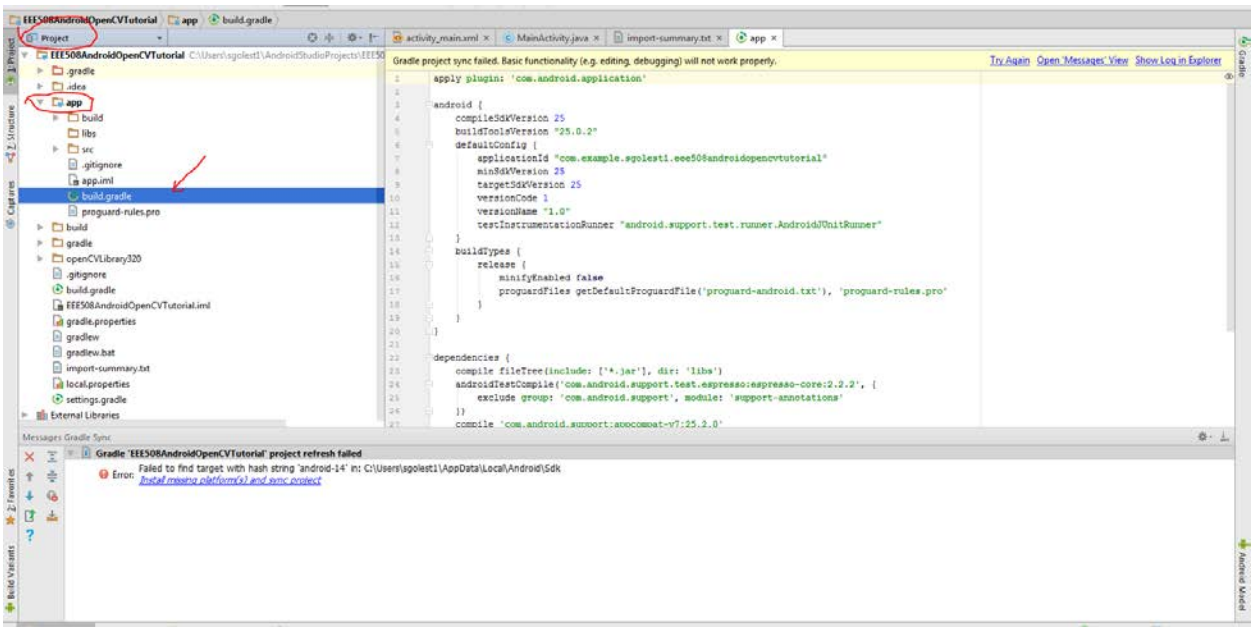
We will need Gradle to run our build configuration.

If you do not have it, you will need to install it. For installation, you can watch this video https://www.youtube.com/watch?v=kAIqcPwGo_w

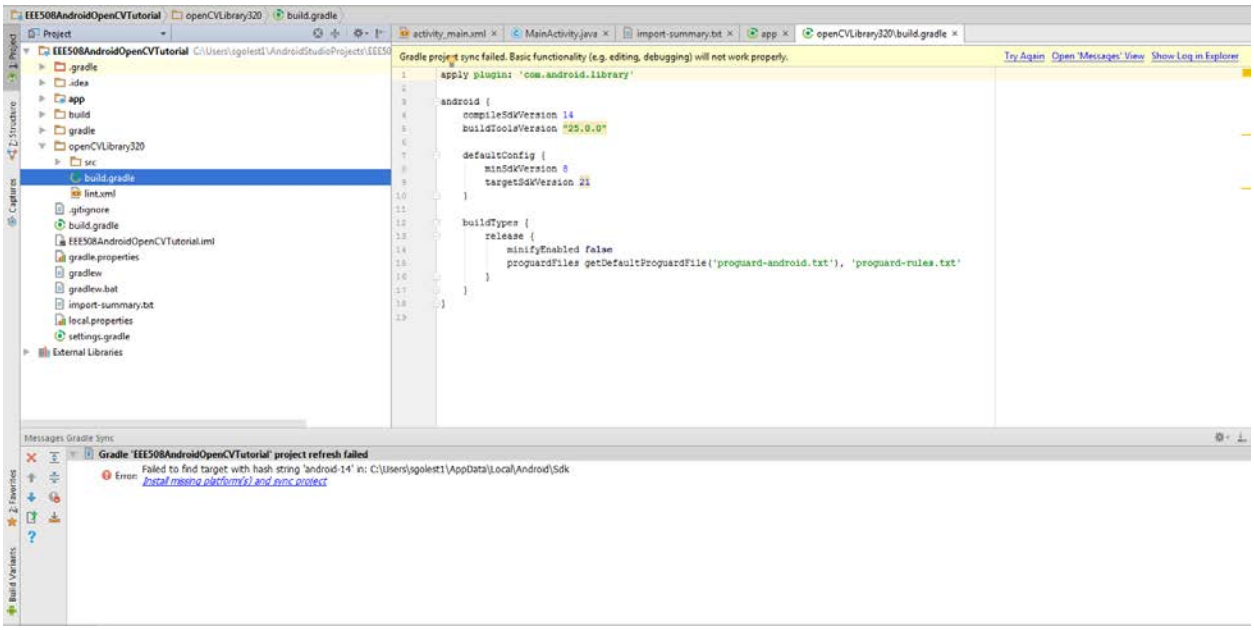
Now select Project instead of Android from the leftmost menu:



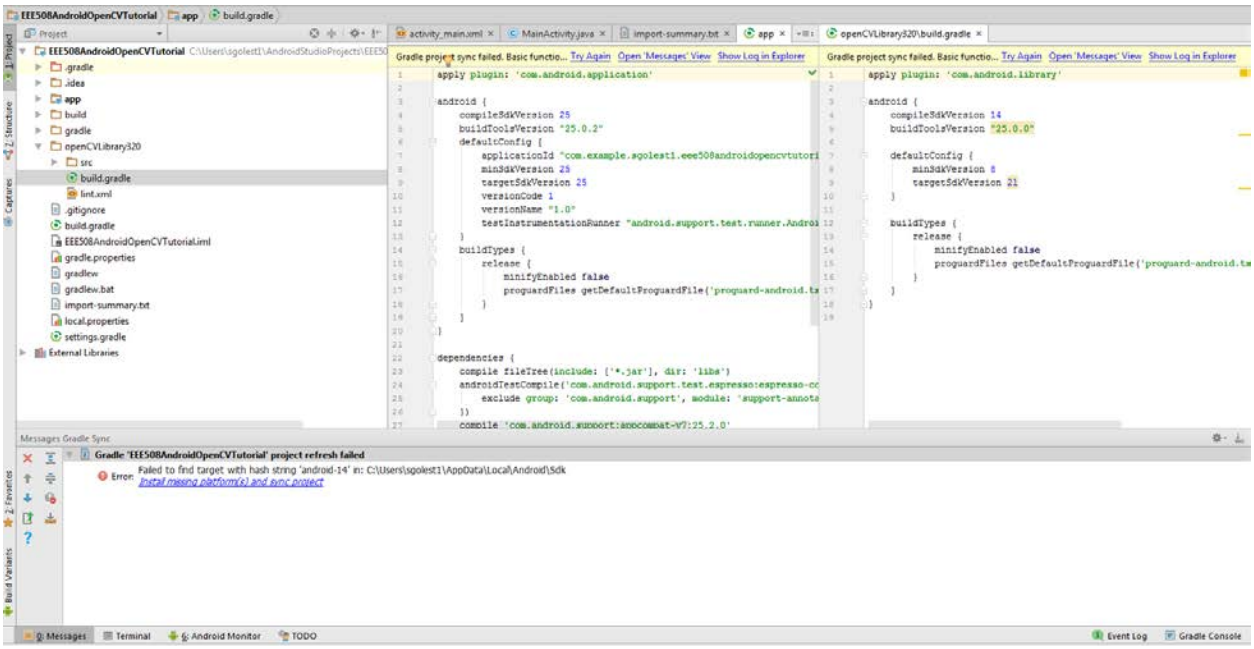
In **app** click on build.gradle and open the build.gradle file.



Now click on the openCVLibrary320 folder and open the build.gradle under that folder:



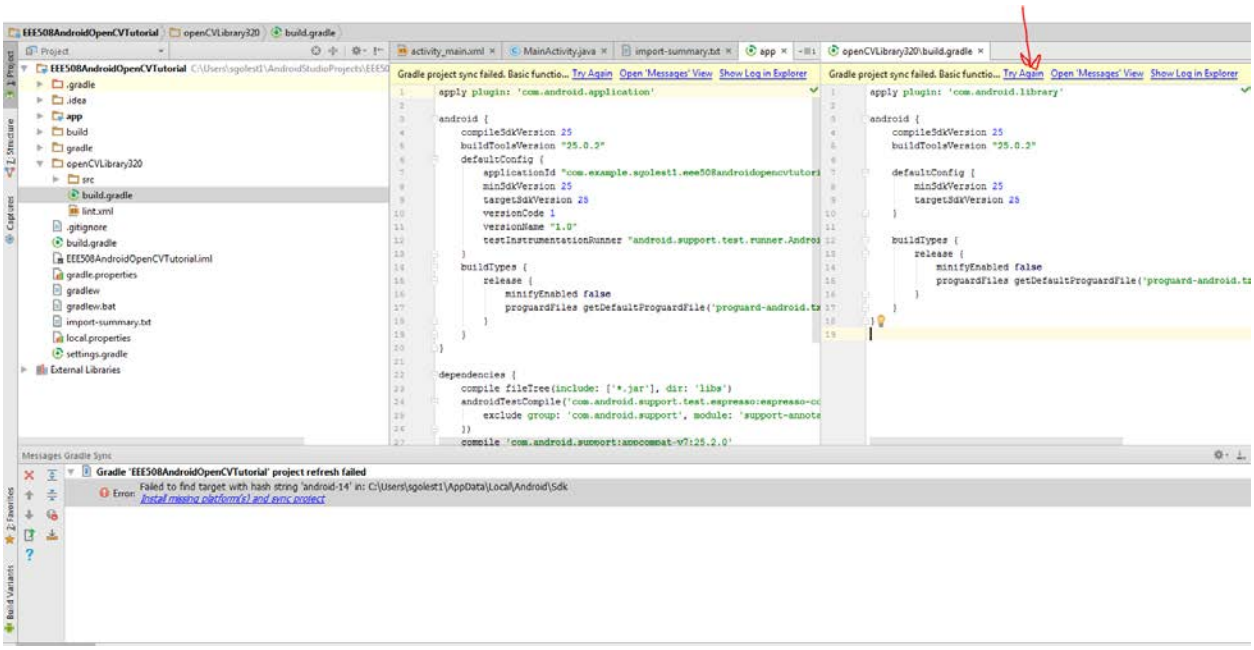
So we have two modules: one for the Android app module and one for the OpenCV module. You might notice that they do not match one another.



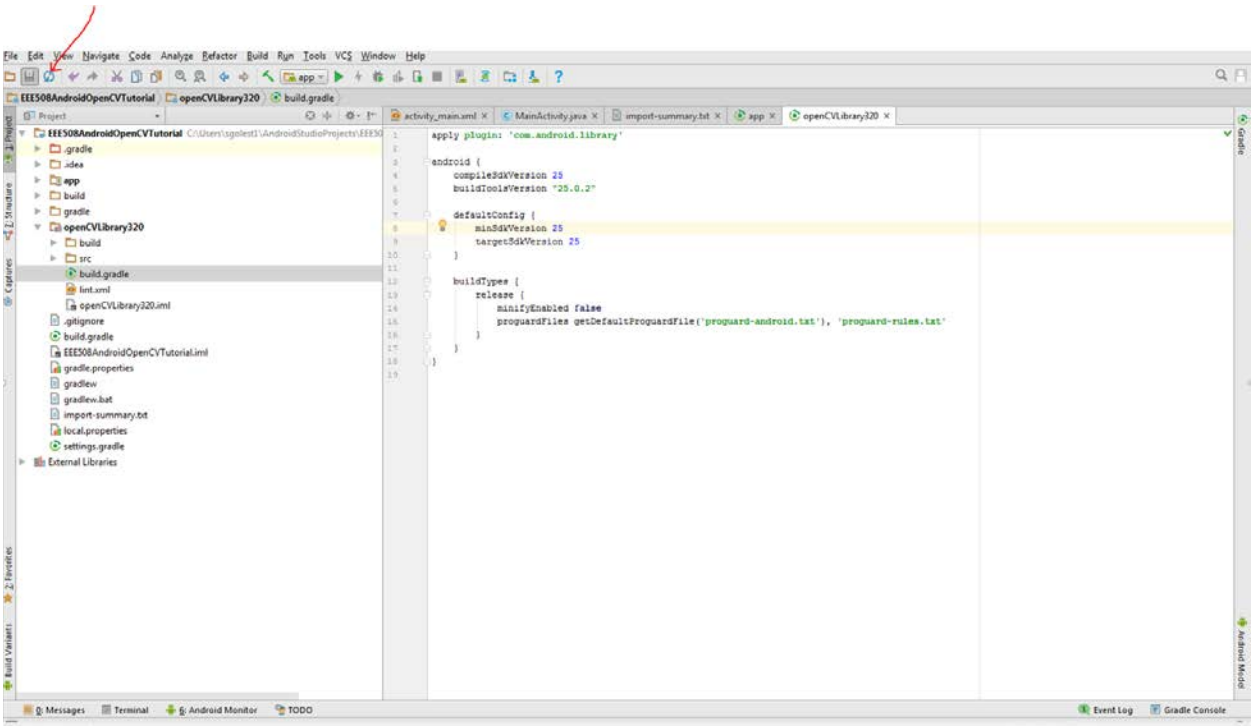
What we need to do is ensure that the openCV one match the app one.

Checkout the values of **compileSdkVersion , buildToolsVersion, minSdkVersion, and targetSdkVersion,** and they should be made equal to the values in the app**'s** build.gradle file**.**

Your build.gradle files should look like this, then click on **Try Again**:



Also, click on the sync icon as shown below and you should not see any error after that.



After synchronizing if you do not already have the necessary Android API versions downloaded, Android Studio may prompt you to download the correct build tools and correct SDK.

Clicking the appropriate SDK versions for your build and clicking install automate the download process of the SDK for you in Android Studio.

You will now notice that our the succeeded with no errors.

Add OpenCV Library

What we did above by importing OpenCV is make the OpenCV classes available to the app for use within the app. If we try to use any OpenCV classes, you'll notice that Android Studio does not know how to resolve any objects that we declare. This is because we must initialize our app's environment with the libraries that we plan to use. We overcome this by making the full OpenCV libraries available to our app. We can do this by pointing our app to the library asynchronously at runtime, or we can statically upload the library to our app's project files.

What we will do here for development is called static initialization. This is meant for the purpose of development where we, as the developer, are hosting the OpenCV library as a part of the app's project files, so the app has the entire backend library at its fingertips. The OpenCV explanation of using the library with static initialization can be found here but for the ECLIPSE environment; we are going to make the OpenCV library available to us while working with our Android Studio project.

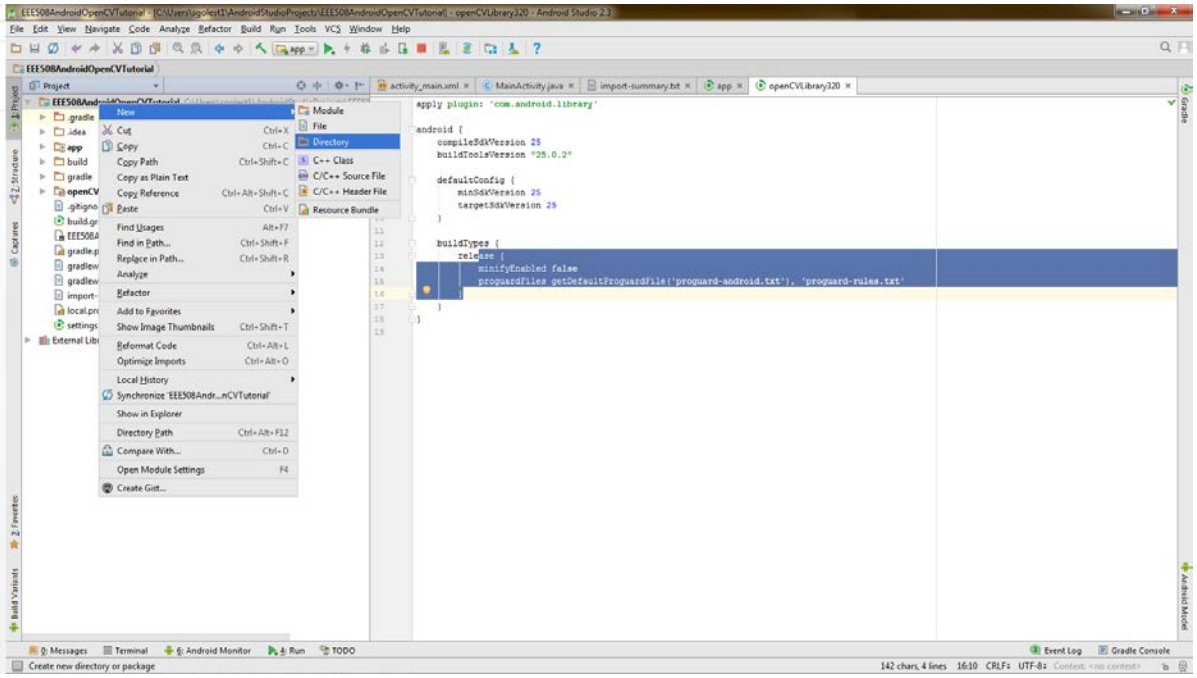
So we will be using our own copy of the library inside our app's project files while we develop in Android Studio. What we will do when the app is actually running is use the OpenCV Manager, which is an app

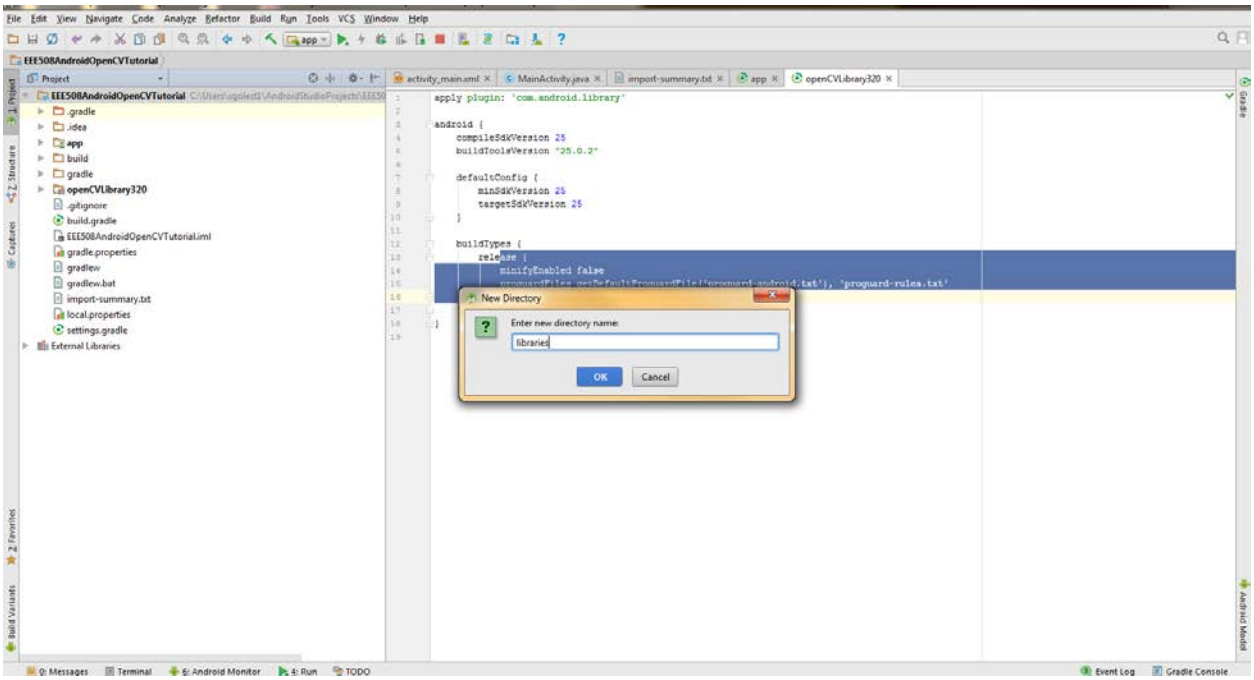
that allows us to access OpenCV libraries using async initialization. how to download the OpenCV Manager to your emulated device will be shown later on.

In order to add the OpenCV library to our Android Studio project, we will follow the instructions described here (StackOverflow).

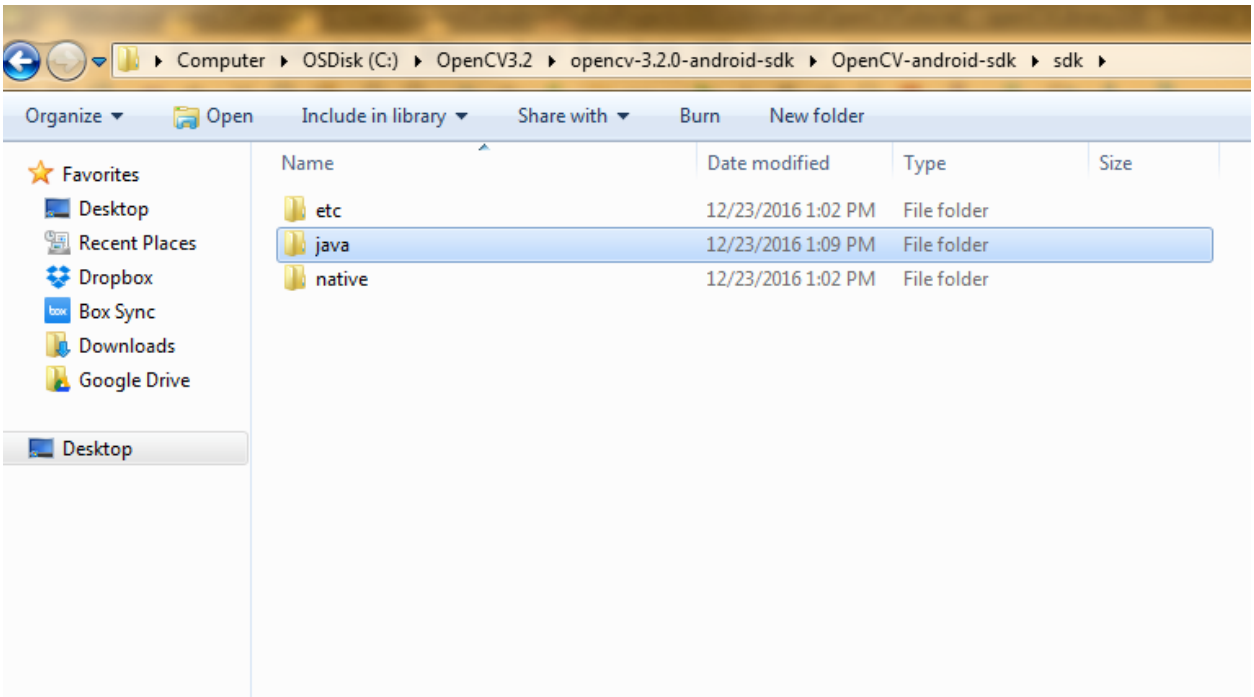
As per the StackOverflow link, do the following:

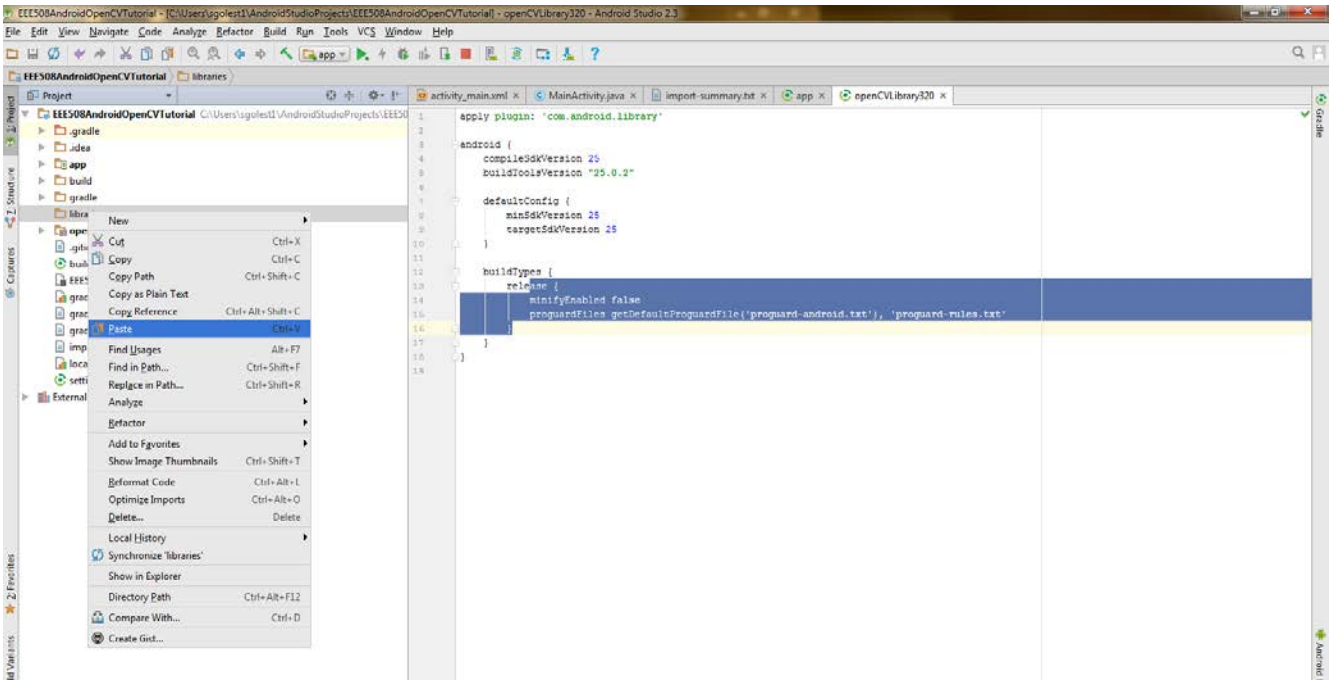
Go to the "Project" tab in the file directory listing view (we are currently in the "Android" tab). Create a libraries folder underneath your project's main directory. For our project named OpenCVTutorial, we will create an OpenCVTutorial/libraries folder.

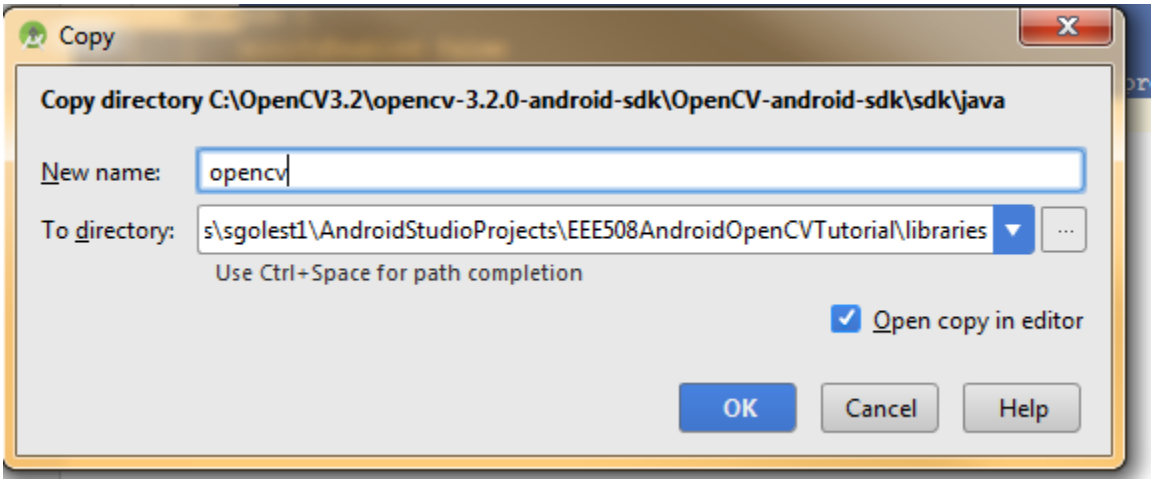




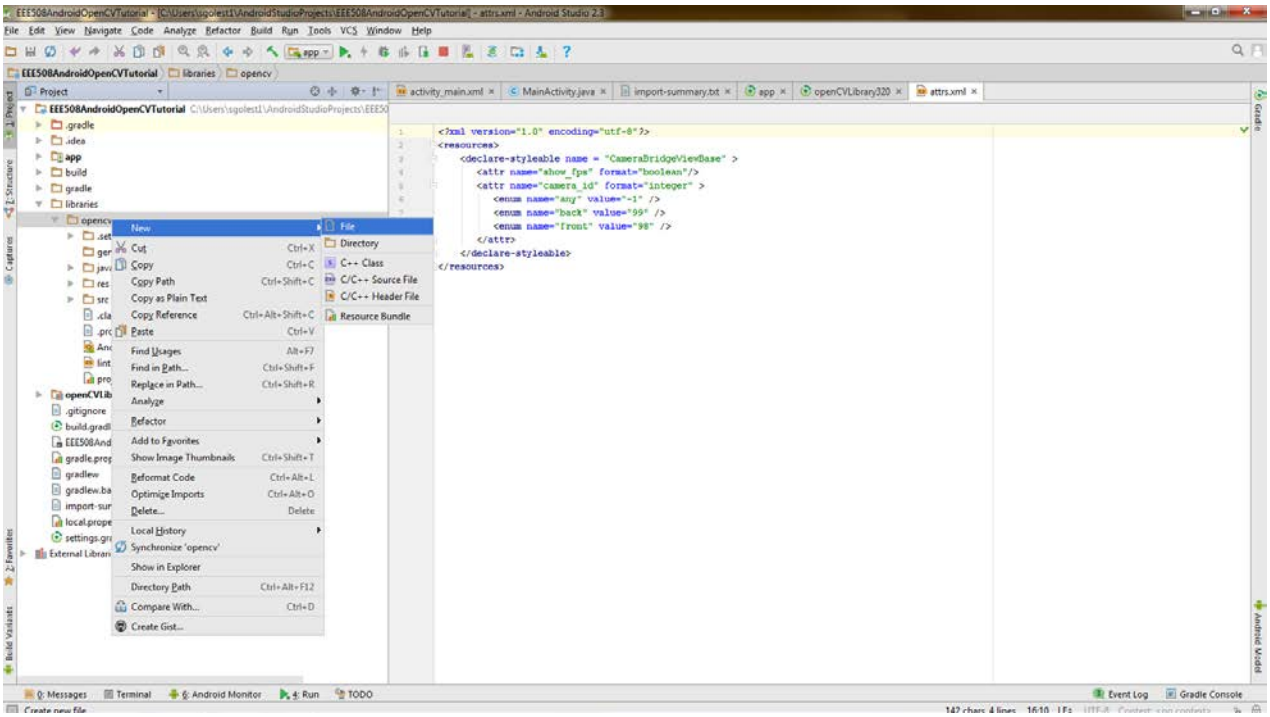
Now, from the OpenCV SDK folder that we unzipped earlier, go to the sdk/java file that we imported as the module. Copy the Java folder. Paste this folder into the new libraries directory that we just created in the project folder. Upon pasting, Android Studio will prompt us for a new name for the file. Enter the name "opencv". This is a large folder and may take a minute to paste.

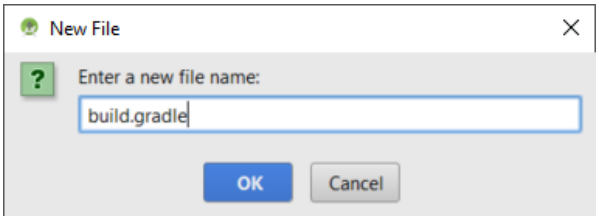






Now we must create a build.gradle file for the OpenCV library. Go ahead and create a new build.gradle file in the newly copied OpenCV directory



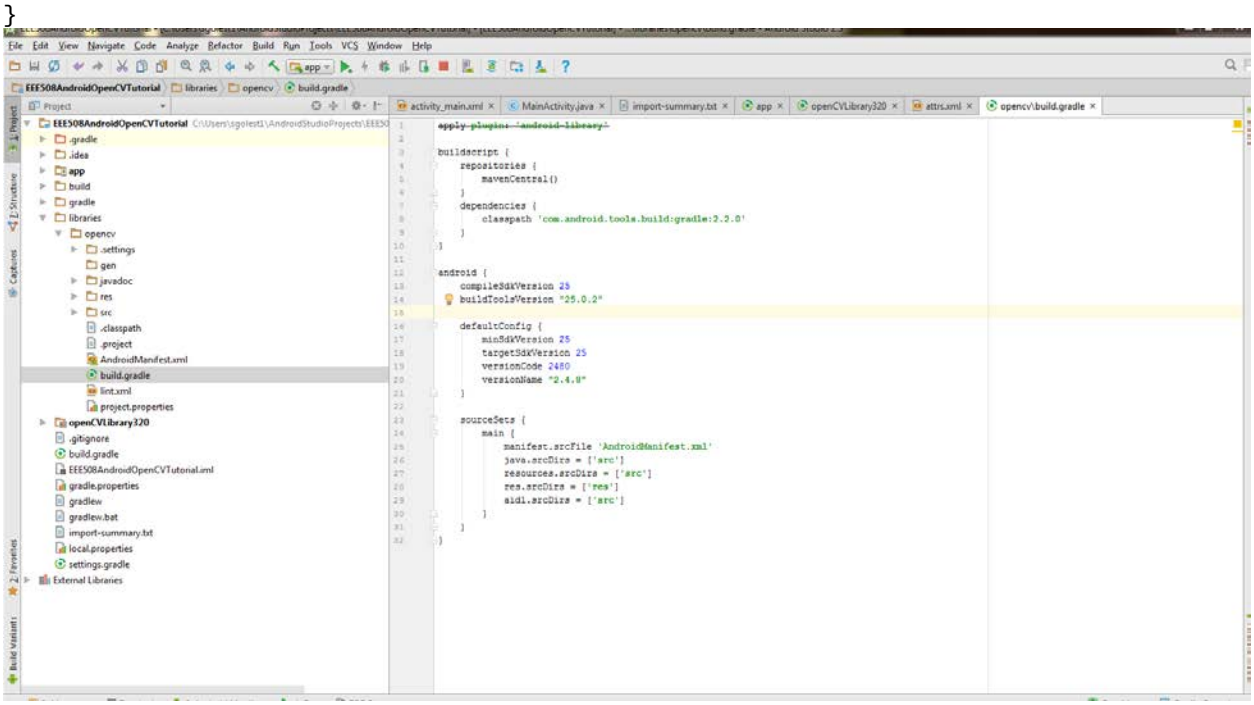


The new build.gradle file needs to be as shown below.

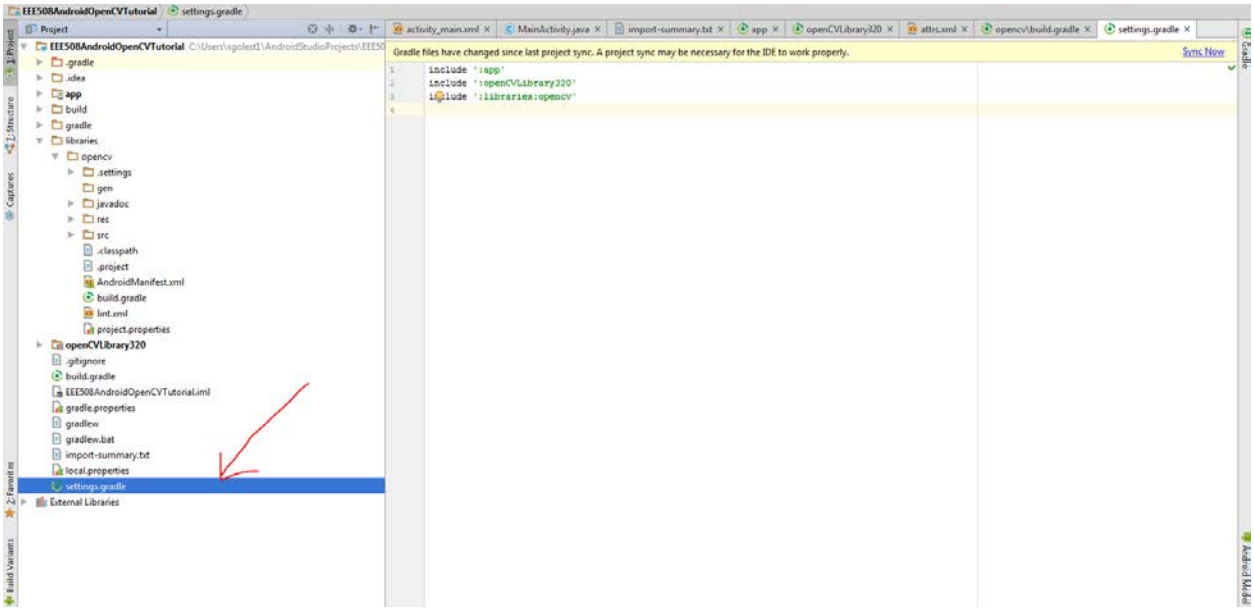
Copy and paste the following code there:

```
apply plugin: 'android-library'

buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0'
    }
}

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"

    defaultConfig {
        minSdkVersion 25
        targetSdkVersion 25
        versionCode 2480
        versionName "2.4.8"
    }

    sourceSets {
```

```
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']
            resources.srcDirs = ['src']
            res.srcDirs = ['res']
            aidl.srcDirs = ['src']
        }
    }
}
```



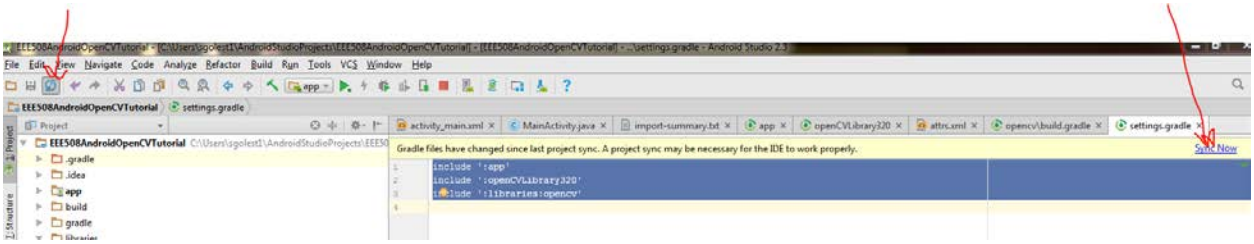
Now, the settings.gradle file for our project needs to know that we have the OpenCV library available. In our settings.gradle file in the project directory, add "include ':libraries:opencv'". Your settings.gradle file should look like this:



So make sure that setting.gradle includes:

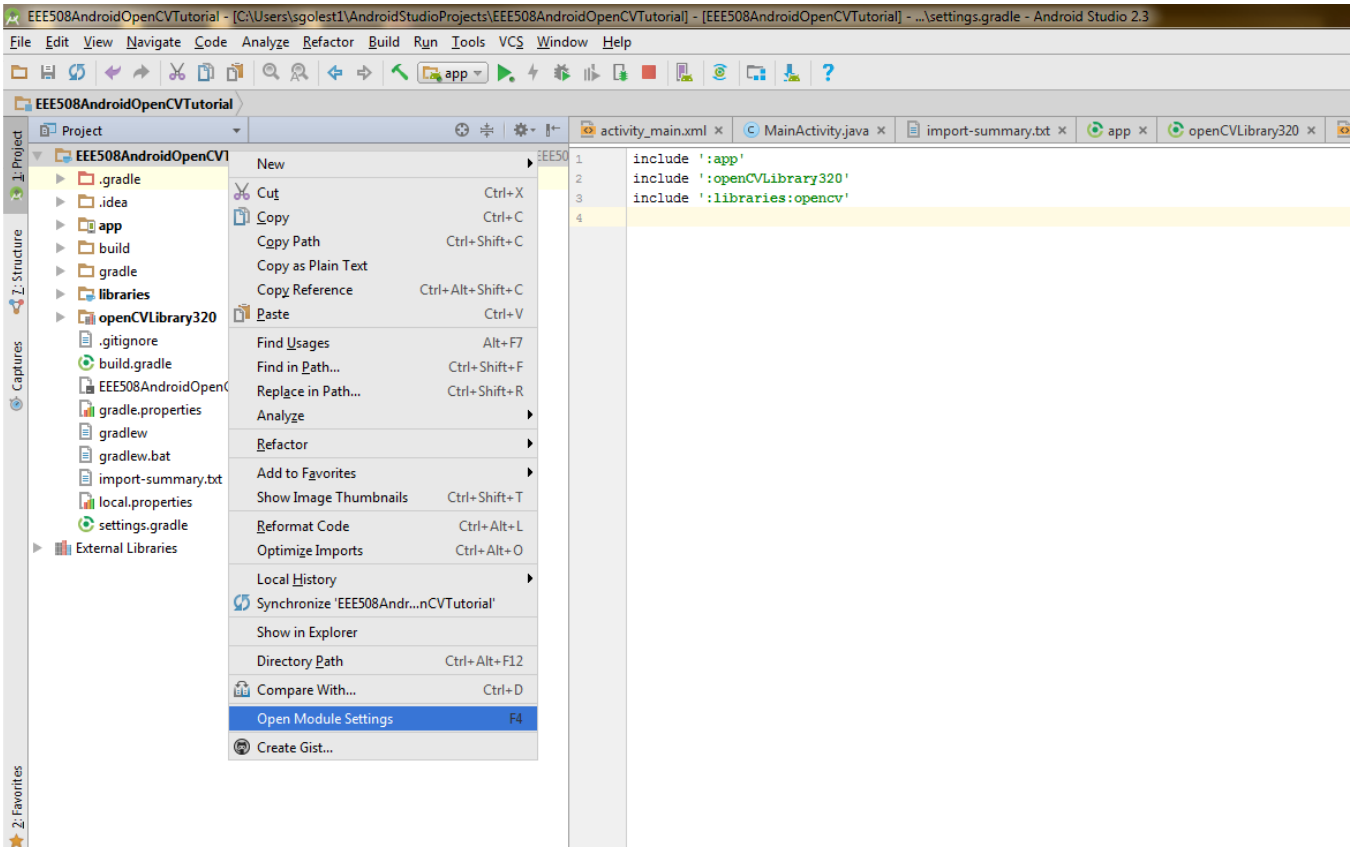
```
include ':app'
include ':openCVLibrary320'
include ':libraries:opencv'
```

Go ahead and sync your project.

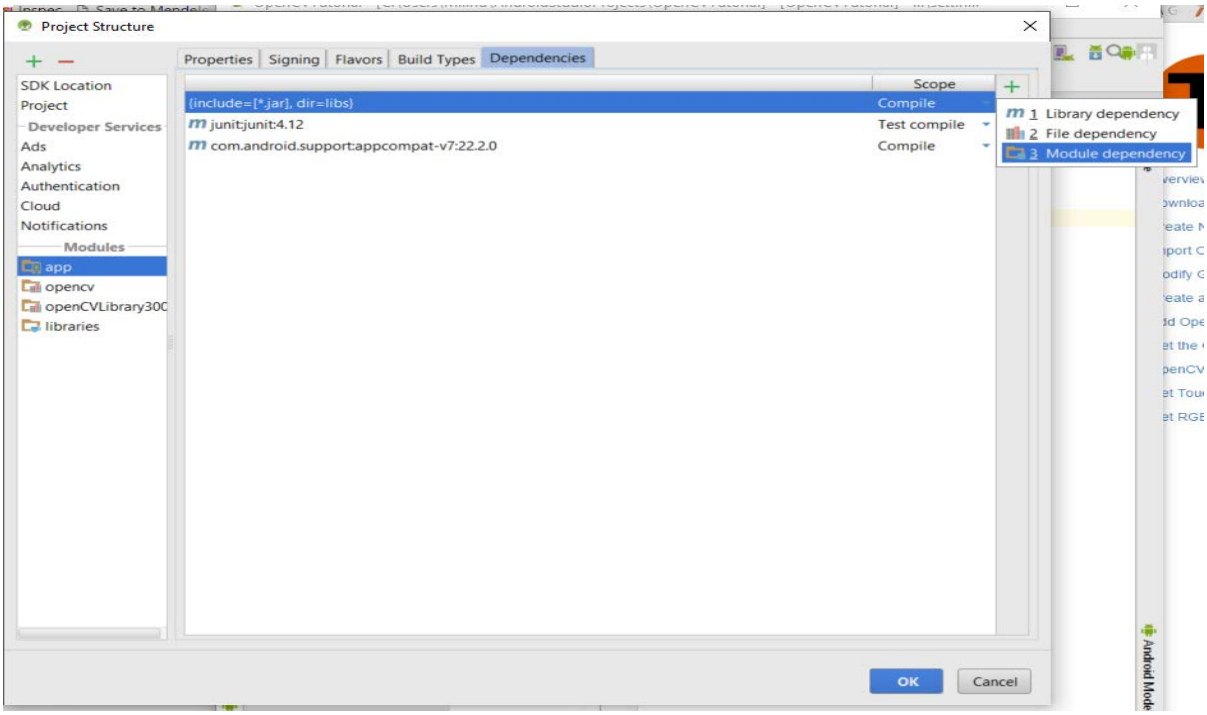


It should complete without any errors.

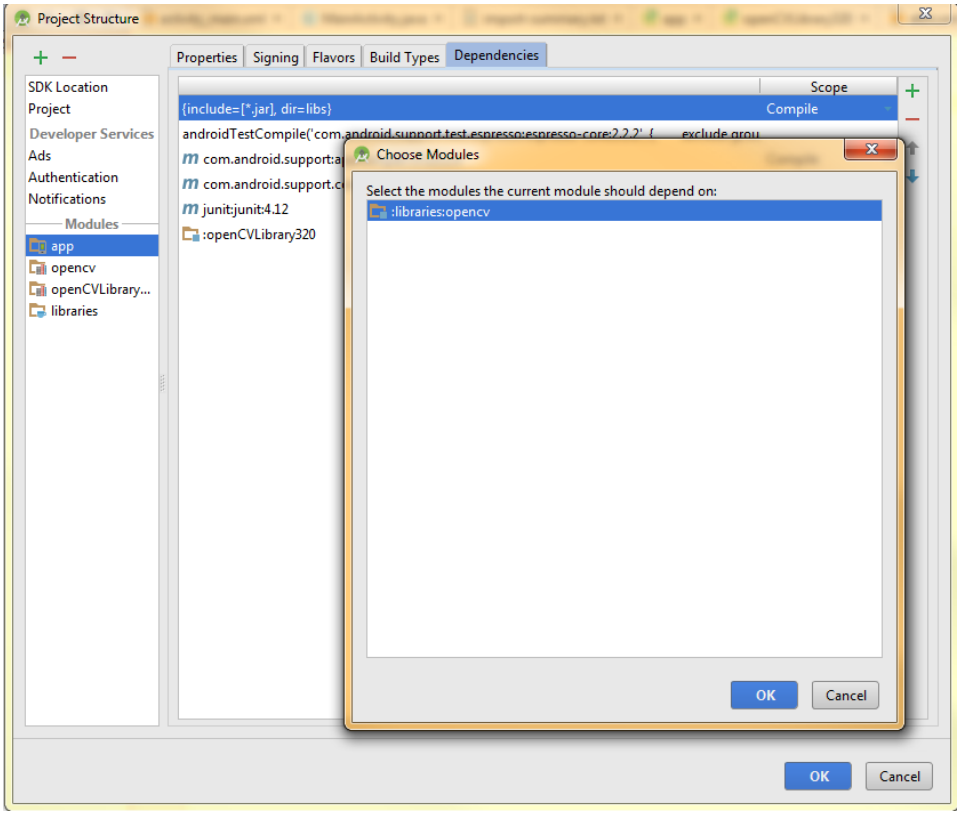
Now we must add our new library as a dependency of our app module. Right-click the project folder and click Open Module Settings.



Go to our app module, where we will add a new module dependency.

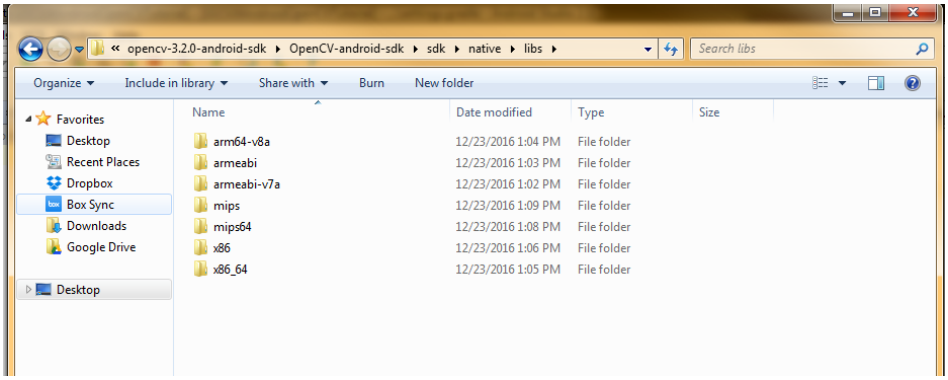


Add our newly created library as a dependency.

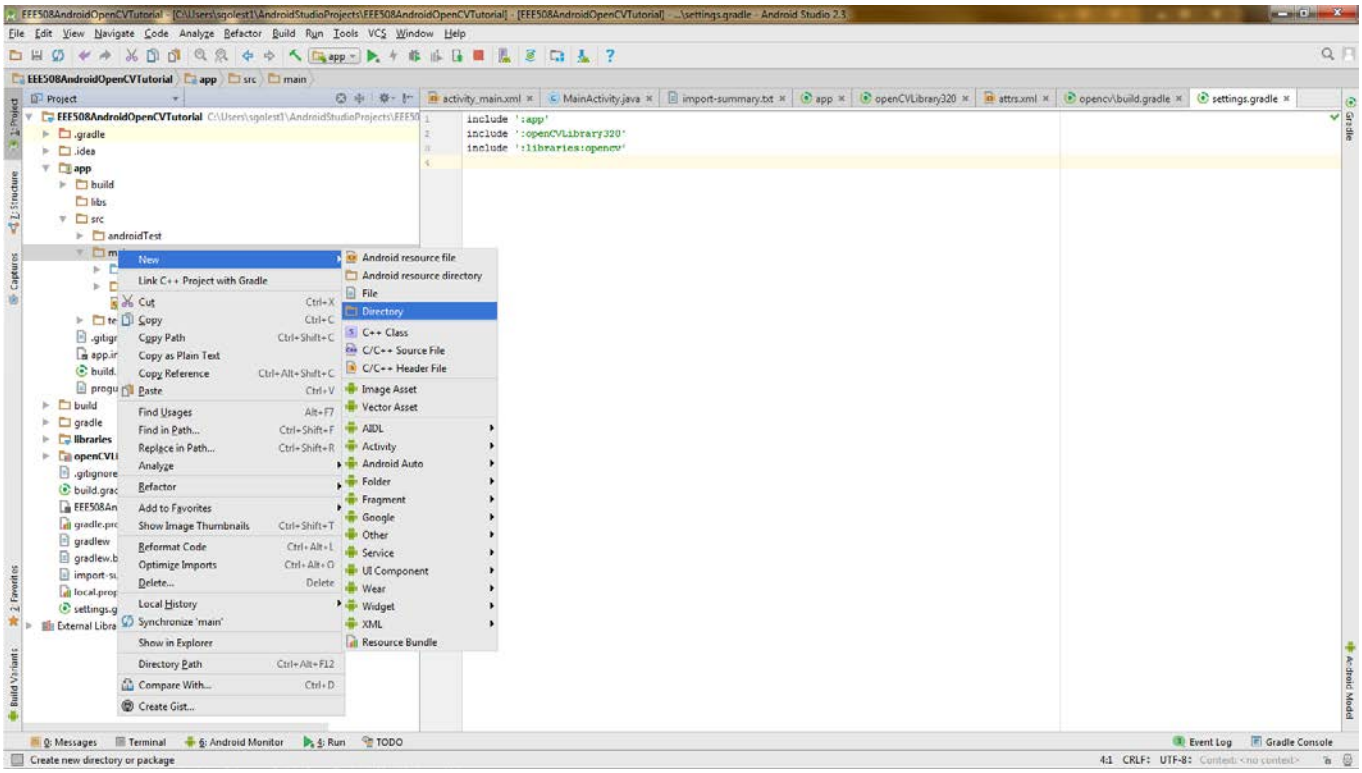


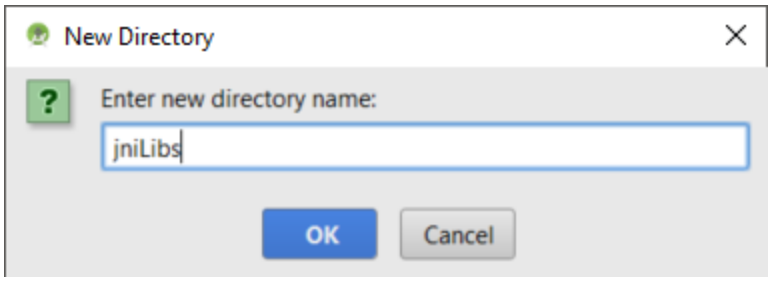
The final thing we must do in order to use the OpenCV library statically is copy all of the JNI library files for the different architectures from the OpenCV SDK into our app module.

Go to the OpenCV SDK and go to the native directory within the sdk file. We want all of the architecture files in the libs folder within the native directory. Copy all of these folders.

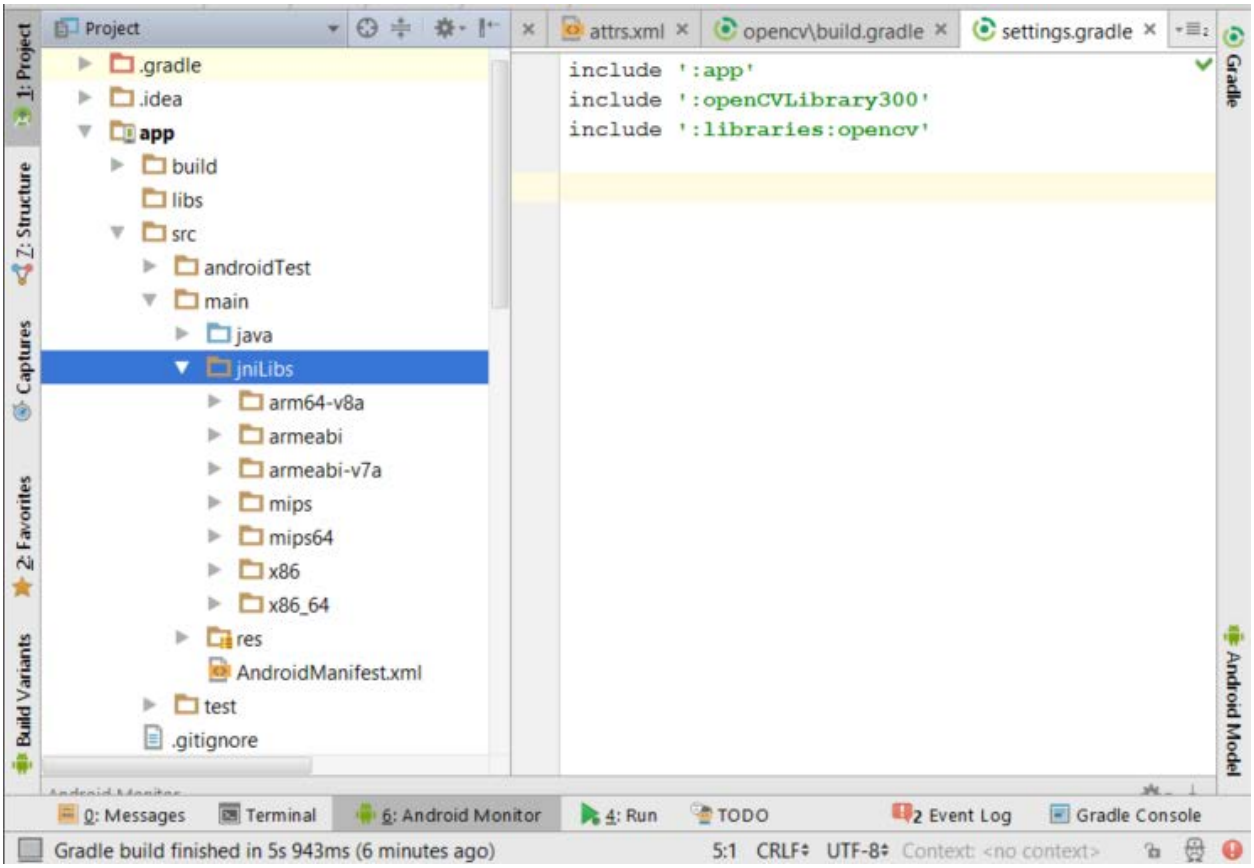


Back in the app module, go to app/src/main. Within this directory, create a new directory called "jniLibs". Paste all of the architecture JNI library files into this new directory.





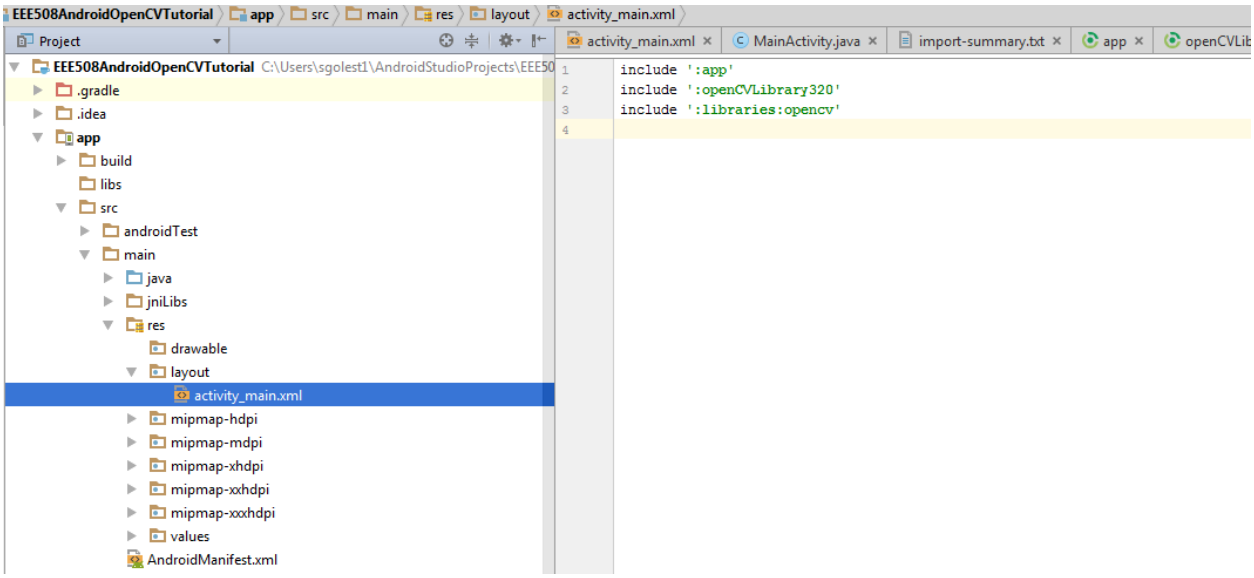
This is how our project directory should look with the newly pasted architecture library files:



Now we can get on and start coding a bit using the OpenCV library in our app! The first thing we're going to do is get the camera up and running and displaying an image for us using the webcam.

Get the Camera Working

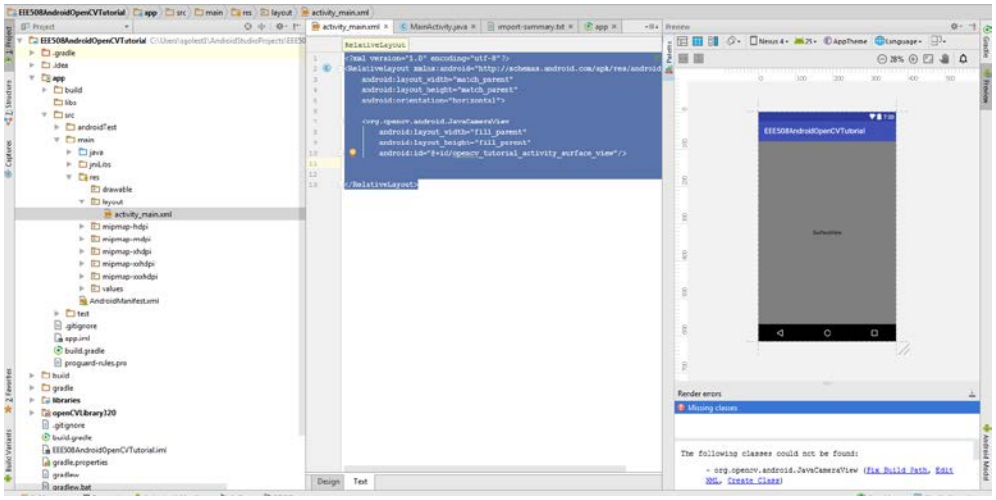
The first thing that we need to do is get our XML layout file ready to use with the camera. Open activity_main.xml found in path app/main/res/layout (or app/src/main/res/layout). Delete the Hello World TextView provided by Android Studio in this file.



Paste the following code there:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <org.opencv.android.JavaCameraView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/opencv_tutorial_activity_surface_view"/>


</RelativeLayout>
```

In the code above, we also added the new JavaCameraView from the OpenCV library that will take up the entirety of the screen. Add the new view as shown below, giving it an ID:



You can see in the layout preview that the JavaCameraView appears, taking up the entirety of the app screen.

When the app starts, we need to get the camera started and display the camera feed in this view. Now that we have the view defined in the layout, let's see how we can get access to the camera.

Open the MainActivity.java class file. We need to implement the following methods:

- onCreate
- onPause
- onResume
- onDestroy

In addition to those methods, we will also implement a BaseLoaderCallback method, which will be used with OpenCV Manager in order to operate our app with the OpenCV libraries.

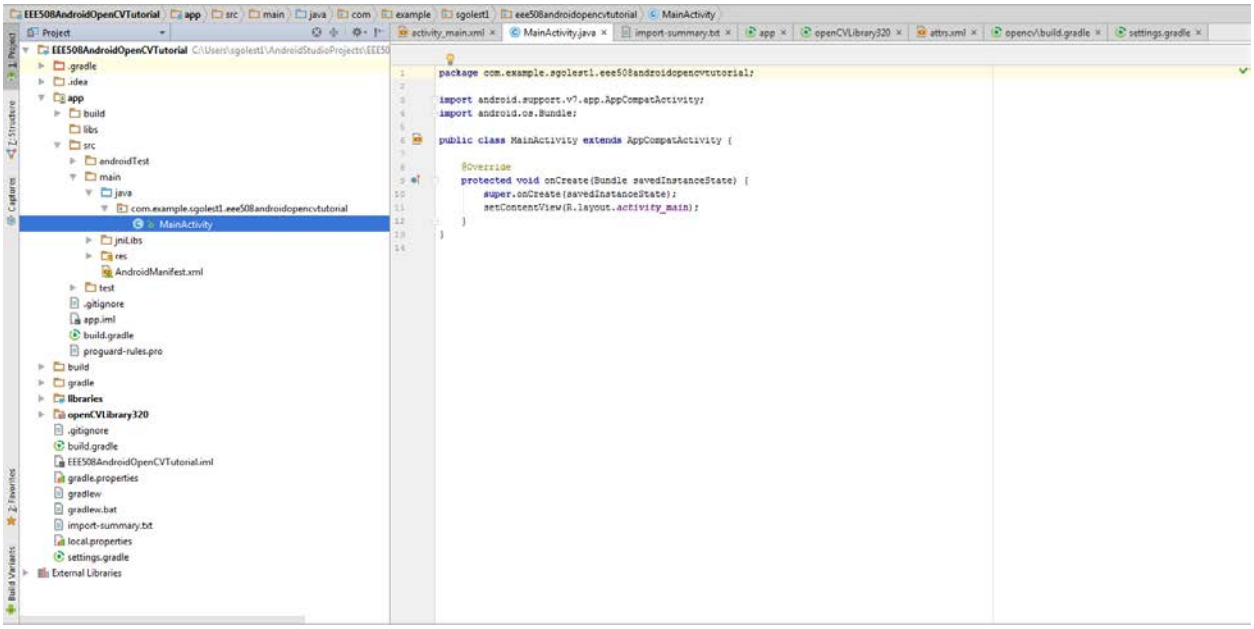
When we open the app, we want to open the camera for use with our OpenCV library and we want to display the camera's feed to the JavaCameraView.

As we implement the methods above, we will need to import a number of components. We will show a few of them, but Android Studio will let you know when you must import a new component.

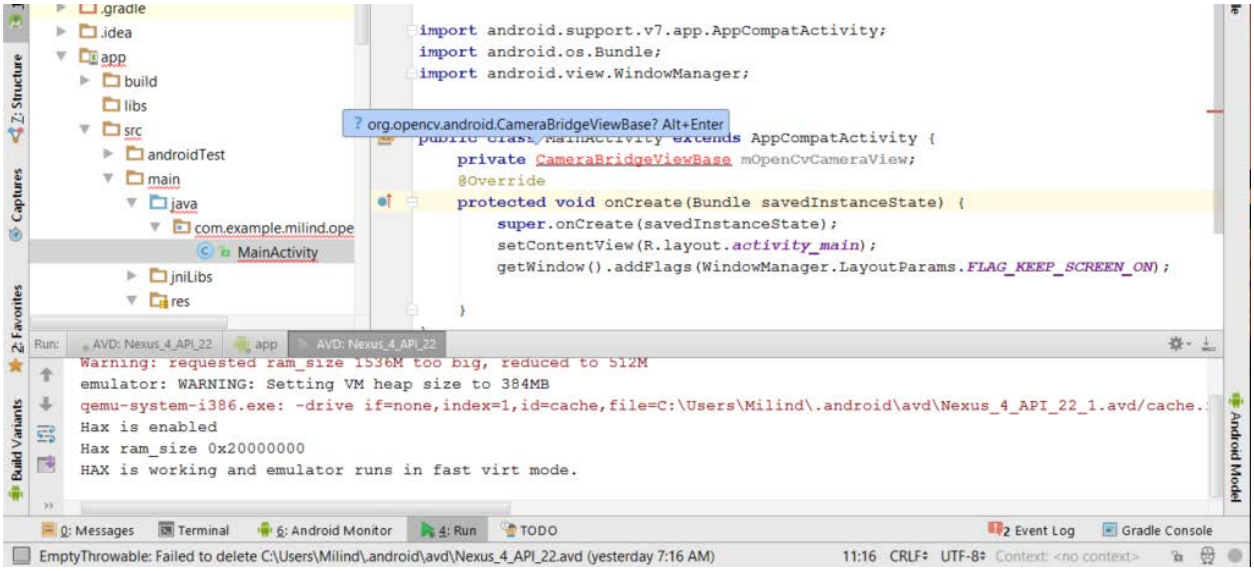
In the mainActivity.java do the following changes:

onCreate Method

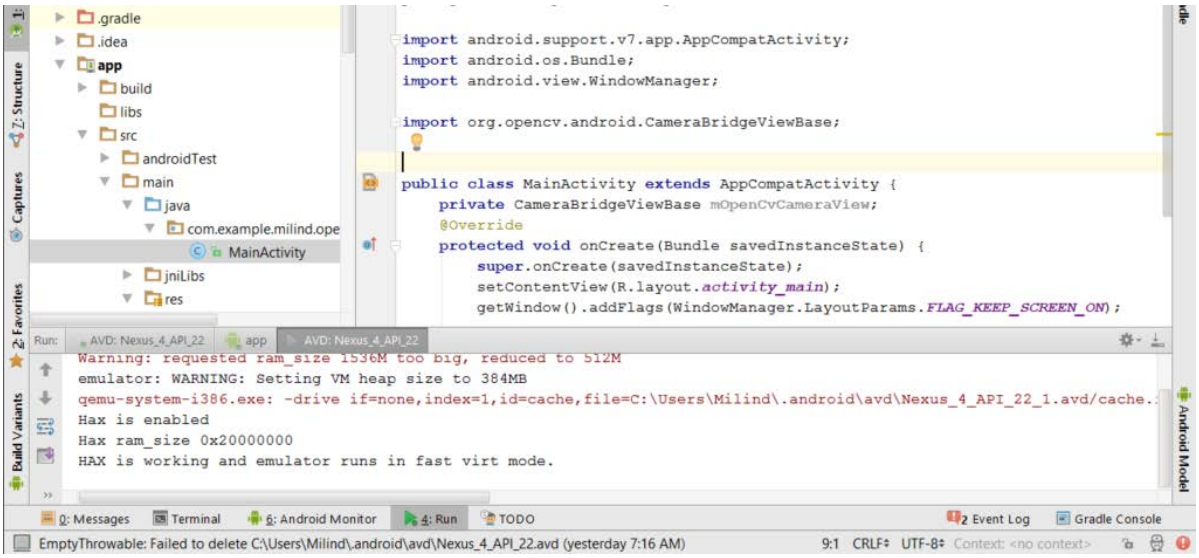
The first thing we need to do is force the screen to stay on when the app is opened



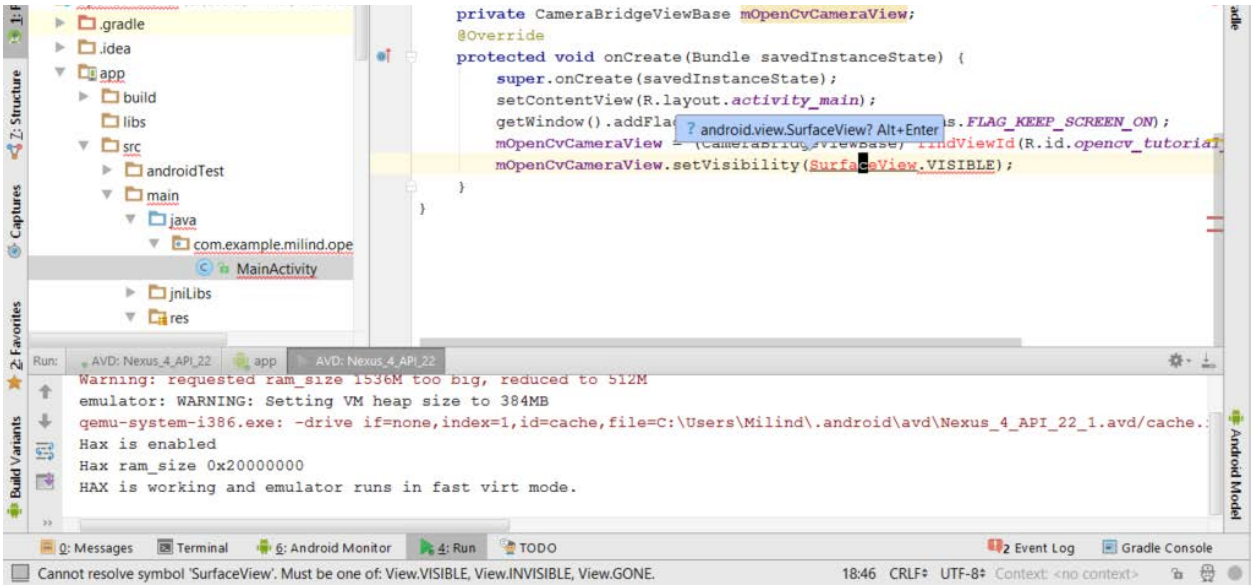
Now we need to declare a new CameraBridgeViewBase object from the OpenCV library that we will use to let our app know that we are using a camera whose feed should be displayed in the JavaCameraView defined in the activity_main.xml layout file.



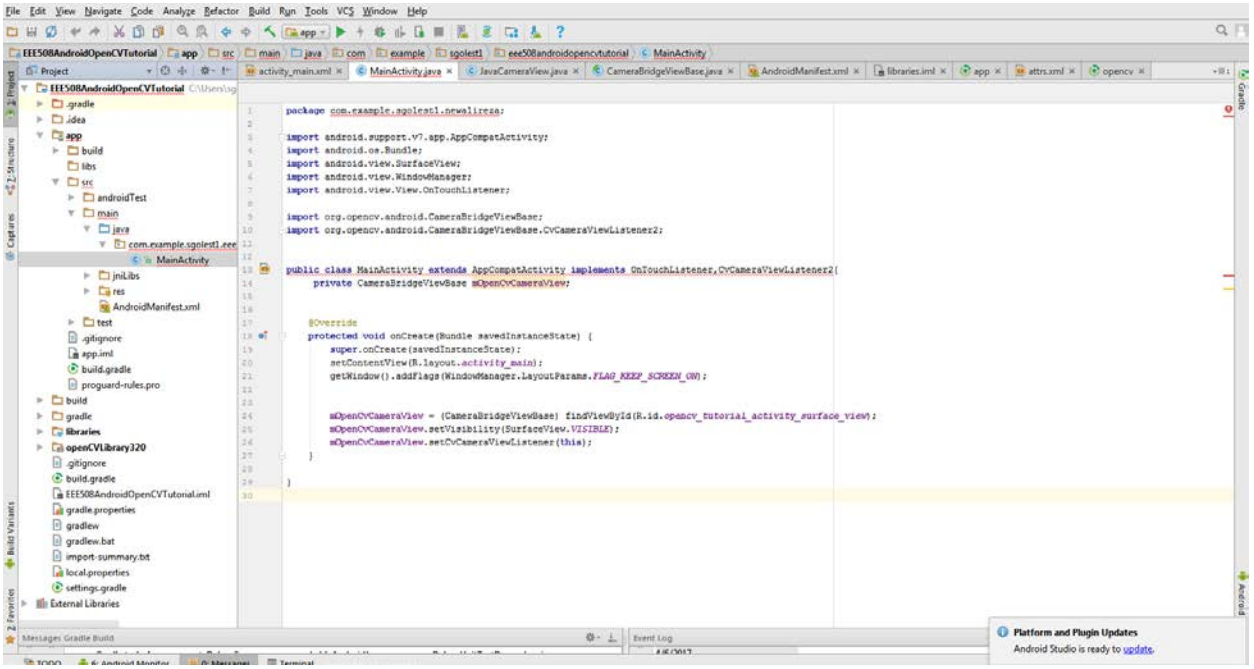
You can press Alt+Enter to automatically import the corresponding required package to import the classes.



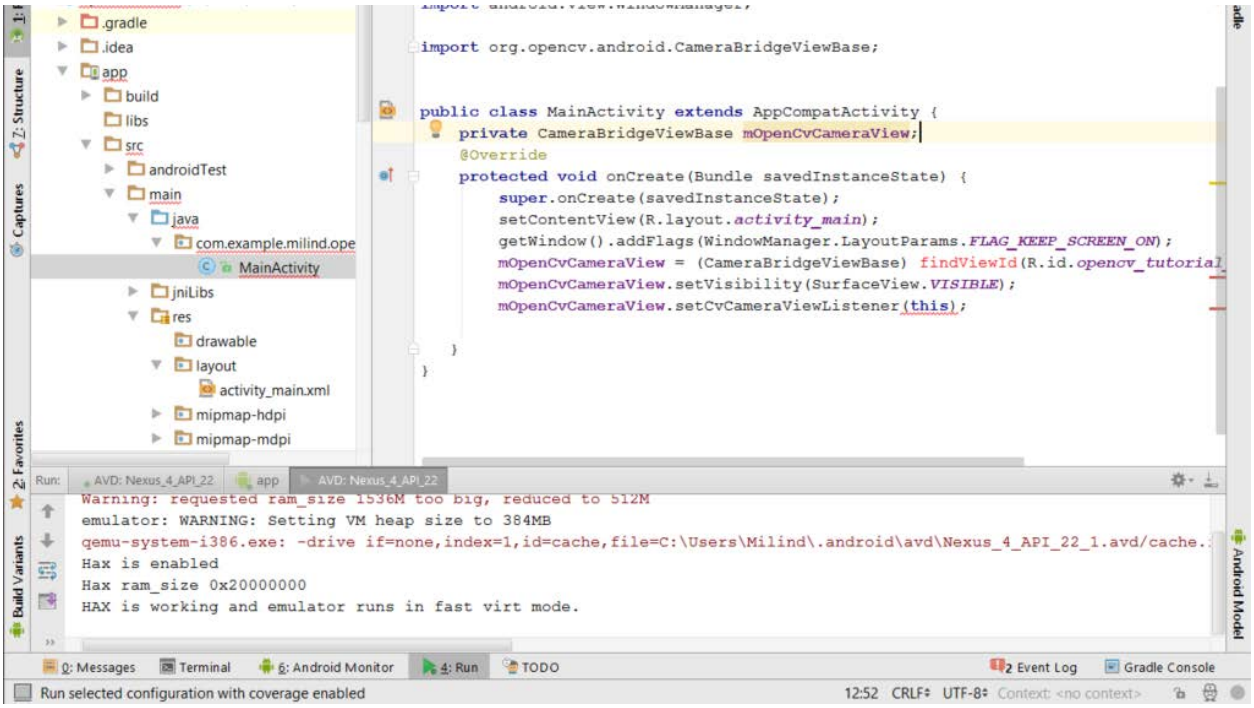
Get the JavaCameraView by its ID and set its visibility to visible. Set the camera listener to our current activity.



Again press Alt+Enter to add the necessary import statements

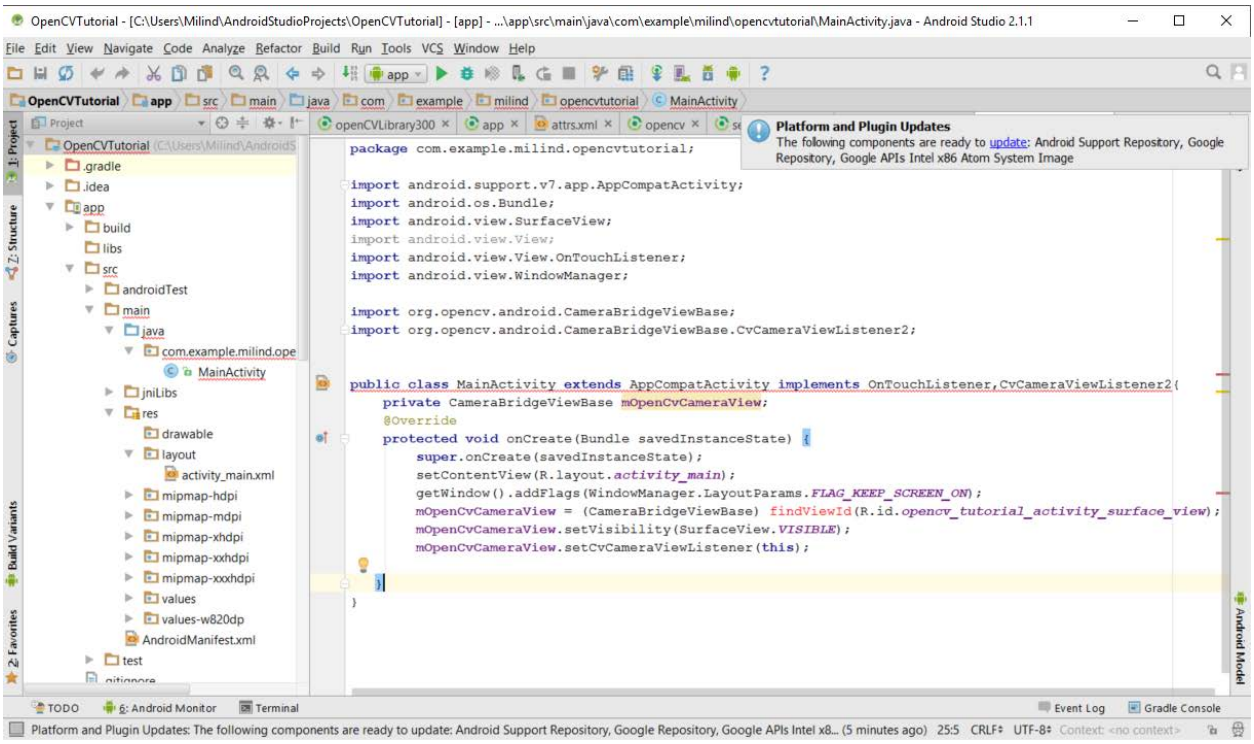


Note that we are passing "this" to the setCvCameraViewListener method.



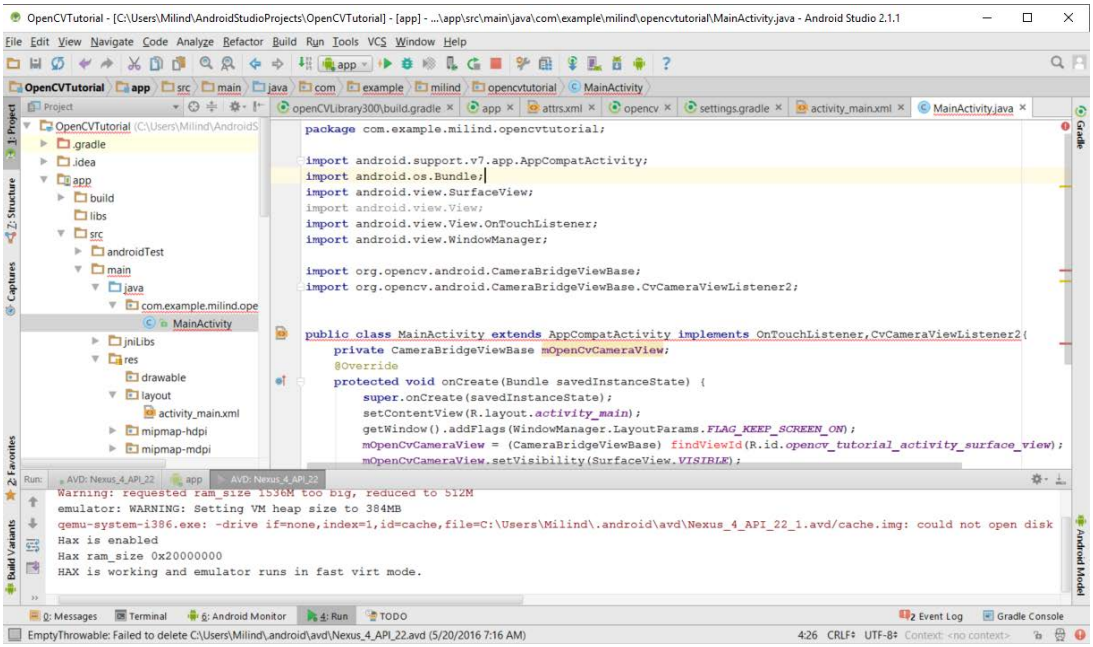
In order for this to work we must specify that our activity MainActivity implements CvCameraViewListener2.

We will also specify that MainActivity implements our OnTouchListener as well, which we will see later.



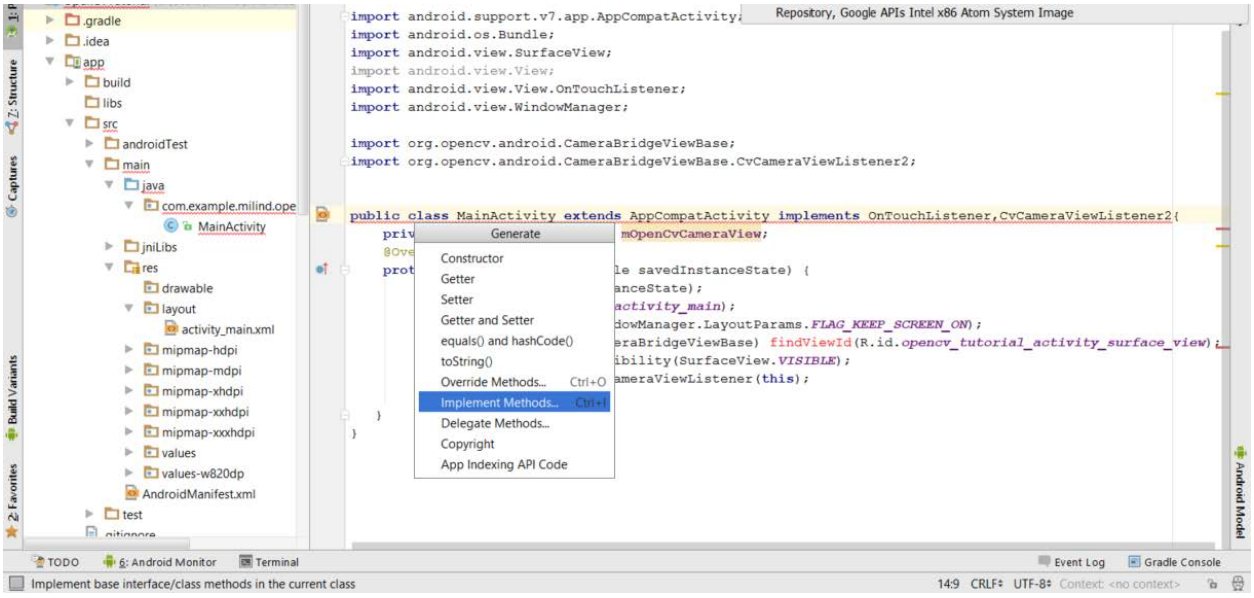
Go ahead and import:

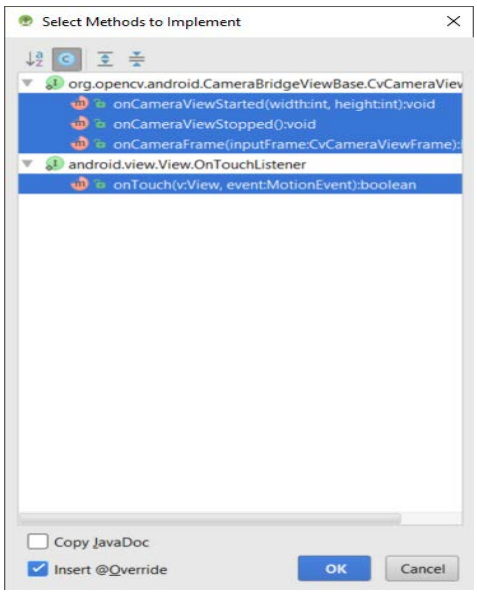
- android.view.View.OnTouchListener
- org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2

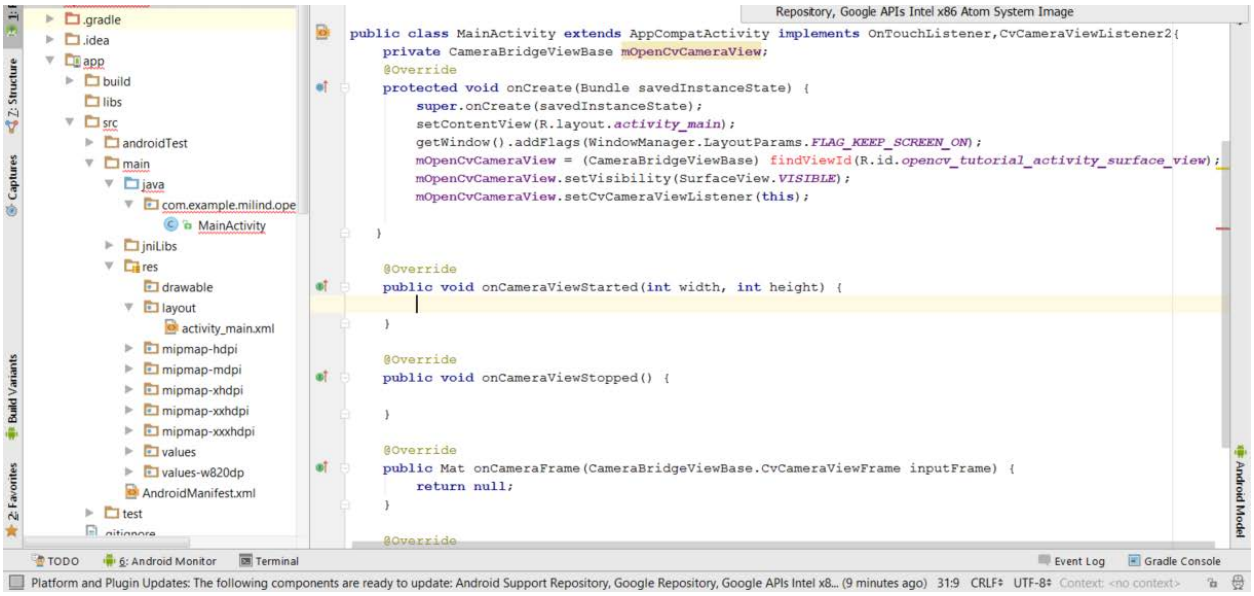


Android Studio gives us an error Aer2, we need some guaranteed methods defined. Go ahead and let Android Studio create those methods for you. We will work with these momentarily.

Right-click on the error and select “Generate” and then select Implement Methods from the menu.

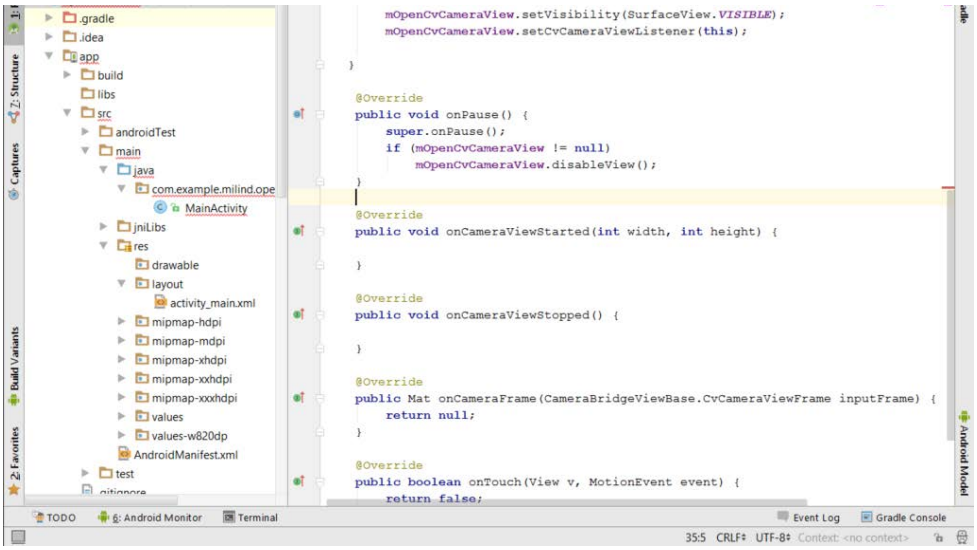






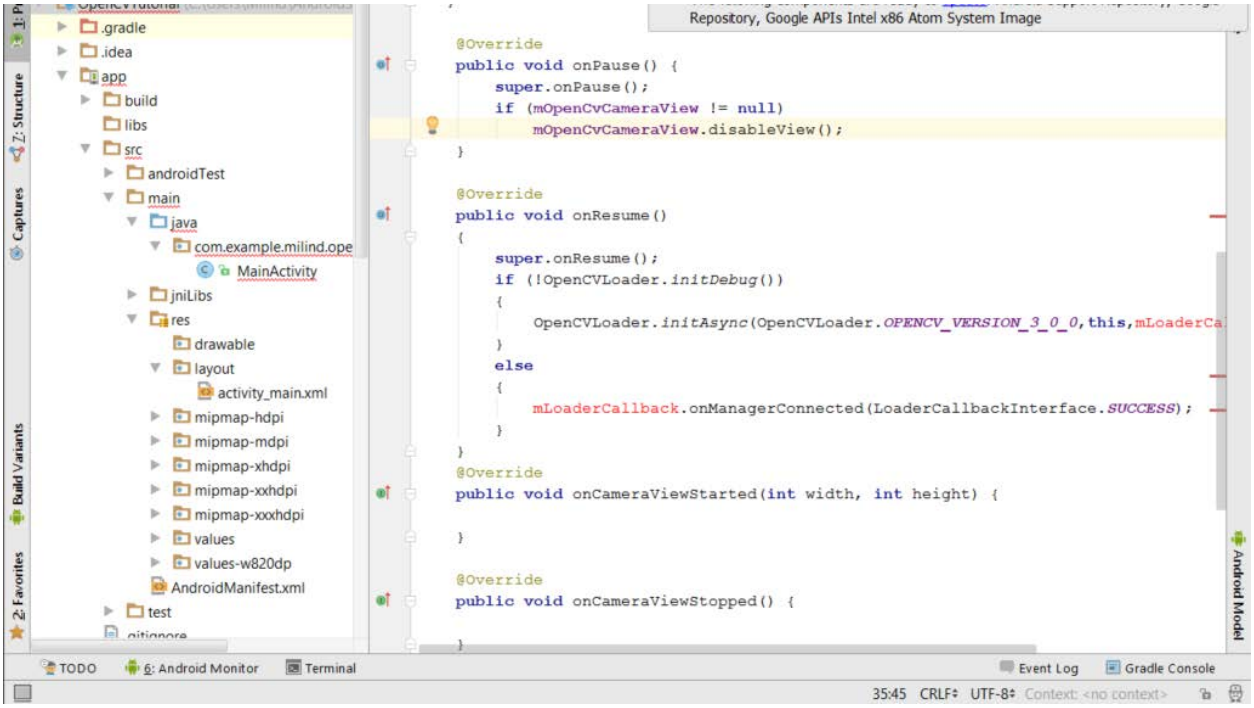
onPause Method

To implement onPause, we would simply like to disable the view that we are using. The disable view method relinquishes the camera for us (internally calling the release method on the camera) so that other apps may use it.



onResume Method

This method is a bit more complex, as we will invoke our BaseLoaderCallback method from the OpenCV library, which we will use to reinstantiate our OpenCV view appropriately. This method will be a bit clearer when we implement the BaseLoaderCallback function for our activity in a moment.

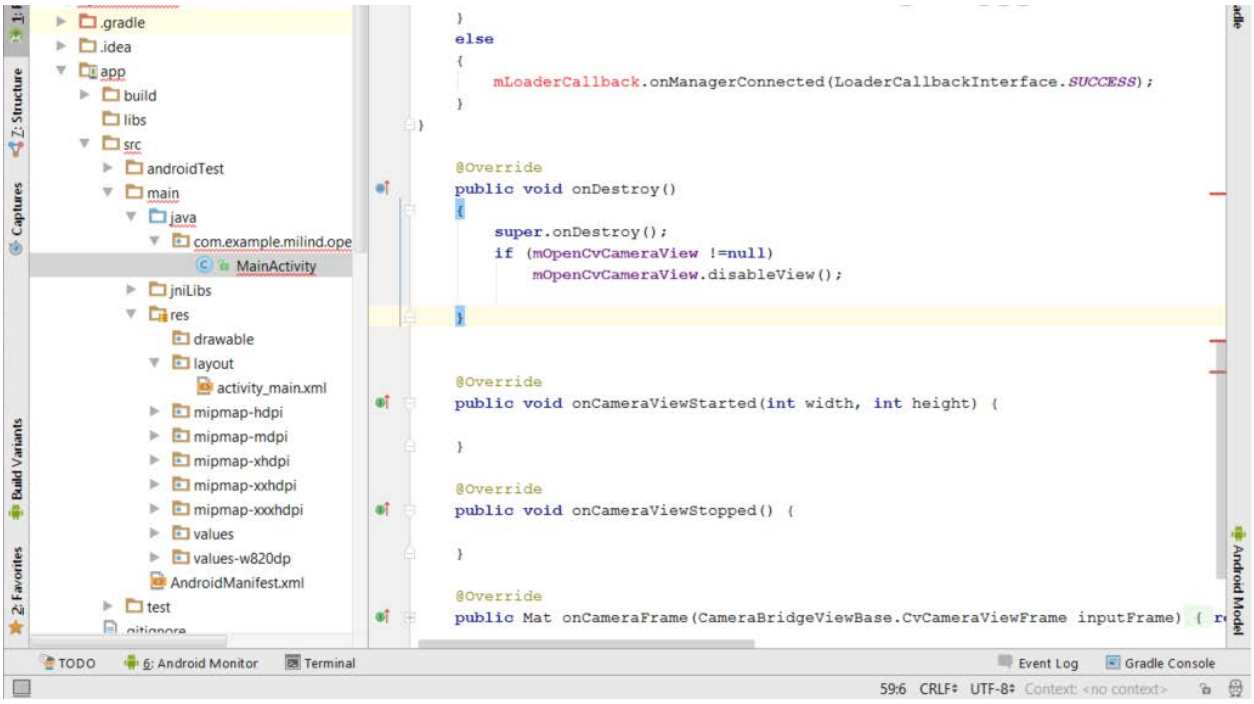


Make sure to import class.

Note that mLoaderCallback is our instance of the BaseLoaderCallback for this activity, so we will take care of that error in a moment.

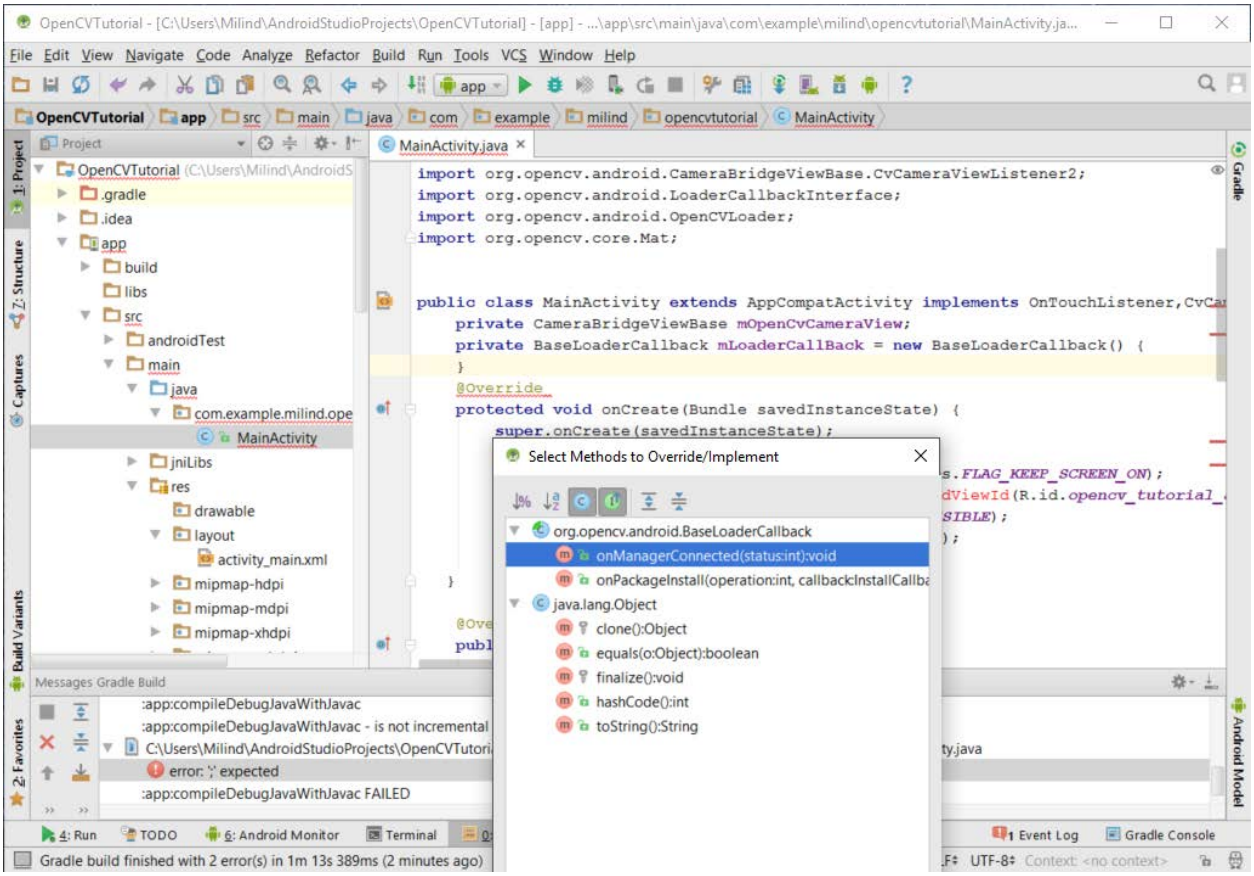
onDestroy Method

Similar to the onPause method, we simply would like to disable the view that we are using with onDestroy.



BaseLoaderCallback

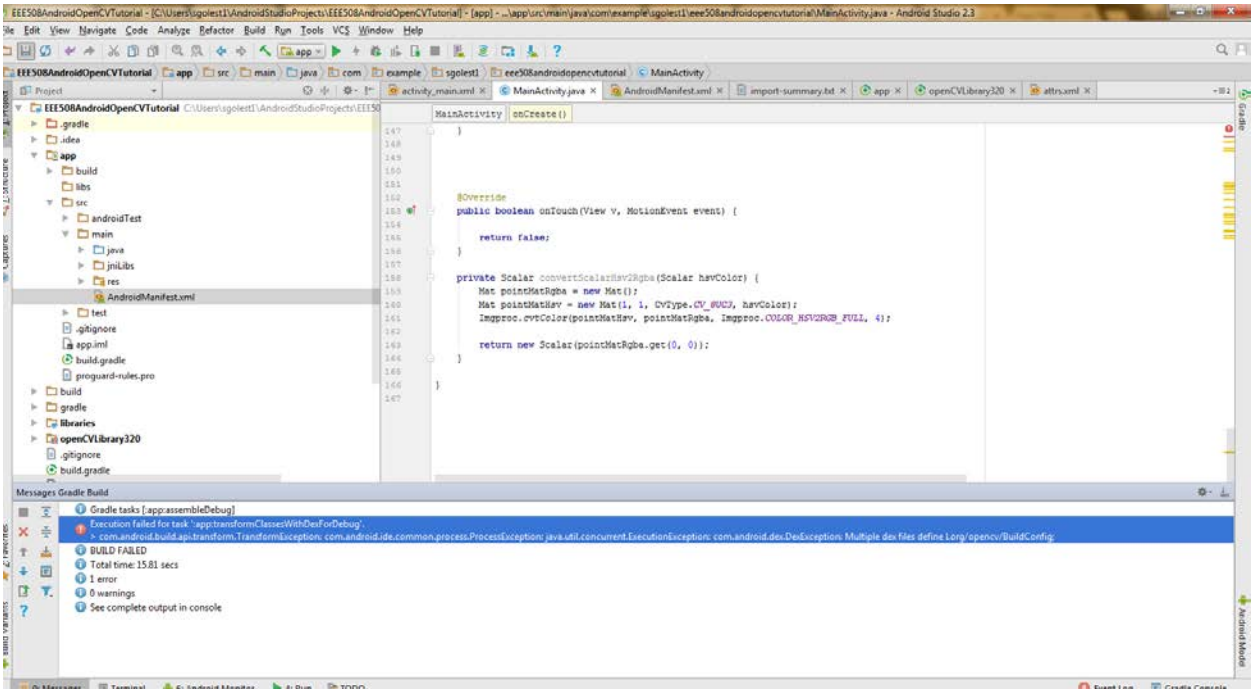
When we declare a new BaseLoaderCallback, Android Studio knows that we need at least some guaranteed methods in order to use it. For this one, we only need to add the onManagerConnected method. (shortcut key ctr+O)



If in the opened window you could not find the same things as the window above, go ahead and do it manually by copying and pasting the following code and placing it in the right position:

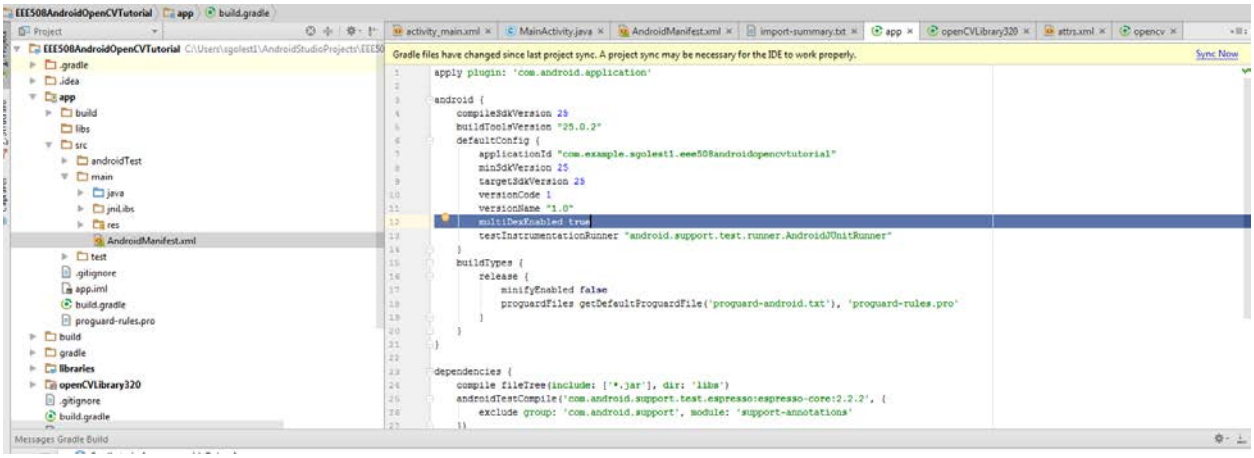
```
private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
    @Override
    public void onManagerConnected(int status) {
        switch (status) {
            case LoaderCallbackInterface.SUCCESS:
            {
                //Log.i(TAG, "OpenCV loaded successfully");

                mOpenCvCameraView.enableView();
                mOpenCvCameraView.setOnTouchListener(MainActivity.this);
            } break;
            default:
            {
                super.onManagerConnected(status);
            } break;
        }
    }
};
```

Let us run the app.



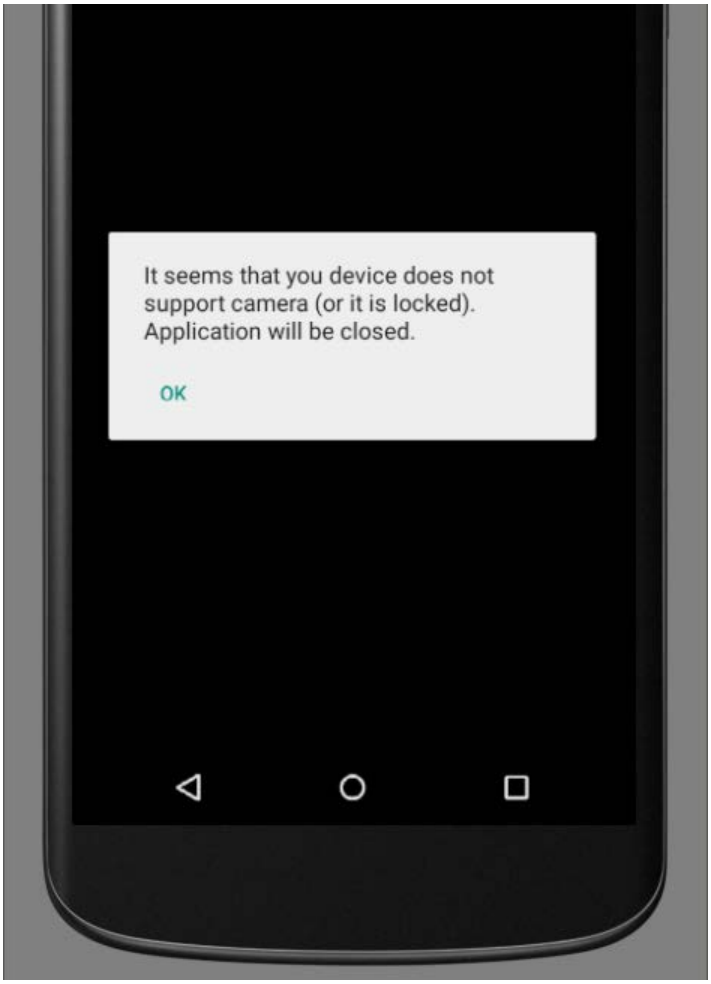
If you obtain an error message as shown in figure above, in order to solve that error you need to go to and open the build.gradle in the **app,** and add the

(multiDexEnabled true) under the defaultConfig.



Let us run the app again and see what we have so far! When you are running the app, keep the emulator open and reuse the same virtual device without relaunching it. It will act as if a device is connected by USB, more or less. You can install apps, uninstall apps, and generally use the virtual device as if it were a full Android device.

After a minute or two, your app will automatically open on your virtual device.

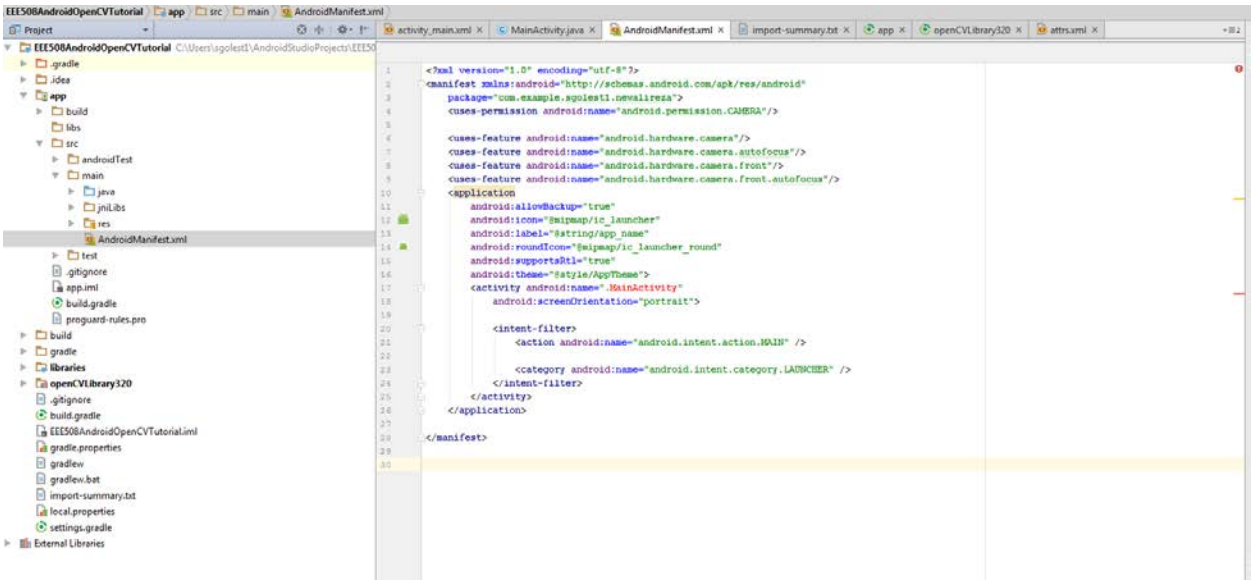


Although we specified that the emulator should use the webcam, the lack of camera support message can appear if we forget to specify that our app has permission to use the camera and that our app will be using camera features in our project's AndroidManifest.xml file. Add the needed uses-permissions and uses-feature tags to that file by copying and pasting the following code in AndroidManifest.xml:

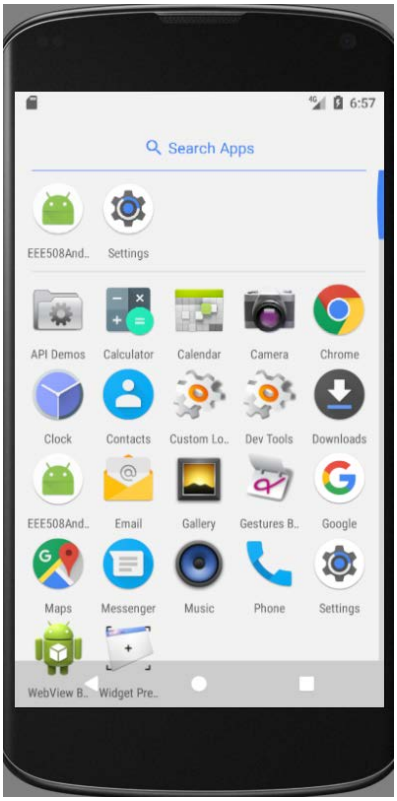
Note that the third line of the following code would be different in your system, and you should keep your package line generated by your computer file setup.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sgolest1.newalireza">
    <uses-permission android:name="android.permission.CAMERA"/>

    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-feature android:name="android.hardware.camera.front"/>
    <uses-feature android:name="android.hardware.camera.front.autofocus"/>
```

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:screenOrientation="portrait">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```



On the phone, go to Settings and then go to Apps→your app (e.g. in this tutorial it would be EEE508AndroidOpenCVTutorial)→Permissions and then enable the camera permission.

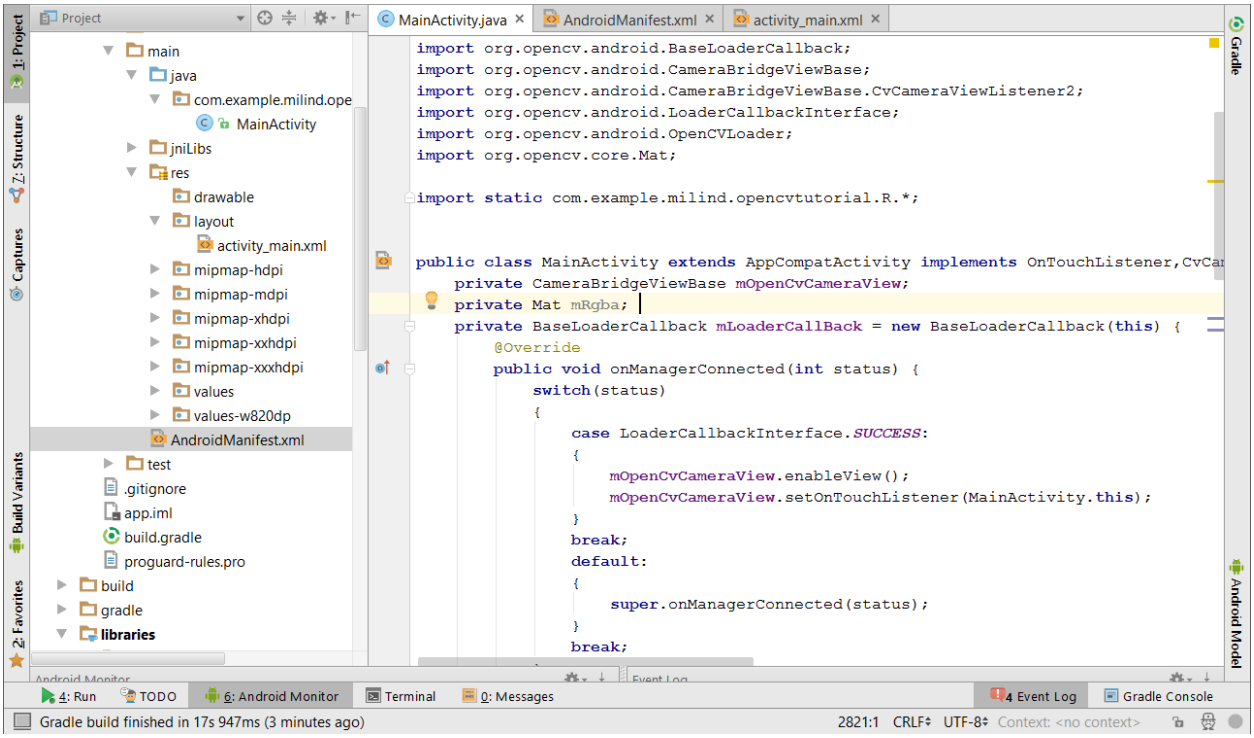


Run the app again.

Why doesn't anything appear? The app is allowed to use the camera and the webcam should be working...

This is an issue with the way that we deal with camera input with the OpenCV library. Remember the methods that were automatically created when we said this activity implements a CvCameraViewListener2? That's what we must do next.

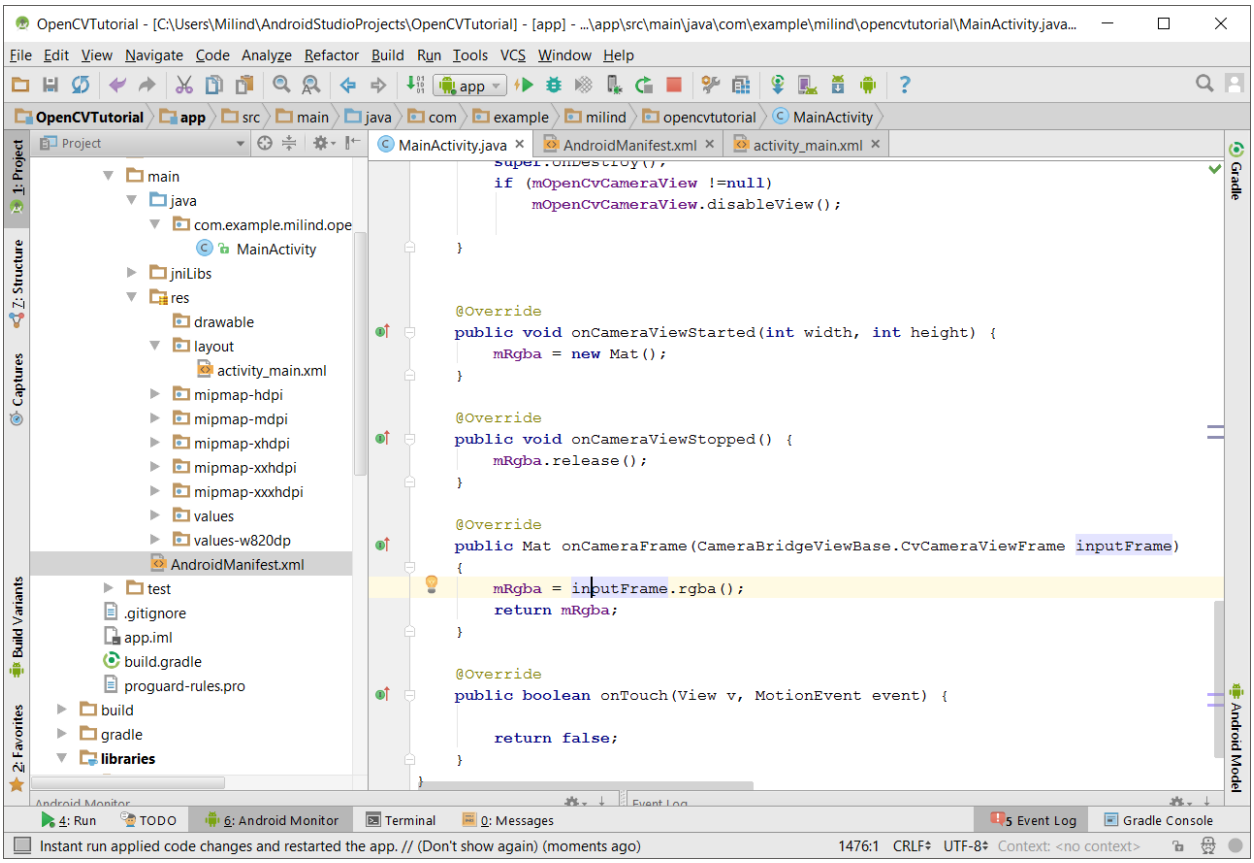
Declare a private Mat variable called mRgba. RGBA is simply the RGB color model with A (stands for Alpha) for transparency.



Now we must implement the following methods in AndroidManifest.xml:

- onCameraViewStarted
- onCameraViewStopped
- onCameraFrame

The implementation of these is straightforward, so we can do them all at once as shown in figure below:

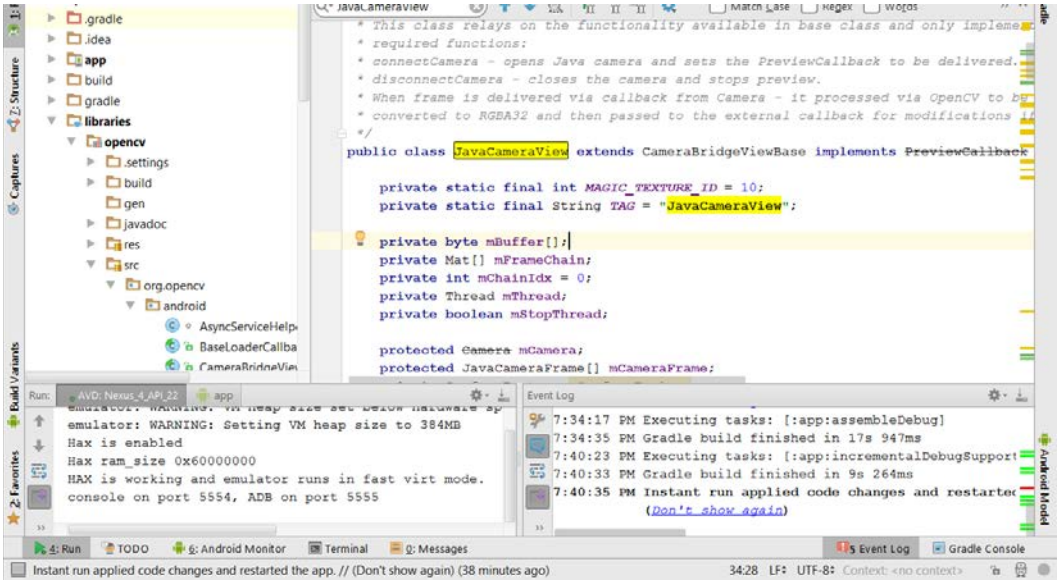


- When our camera view starts, we want to save a new matrix to our mRgba variable.
- When a camera view stops, we want to free this matrix.
- While the camera is running, we want to store the inputFrame in the matrix that we created and return it.

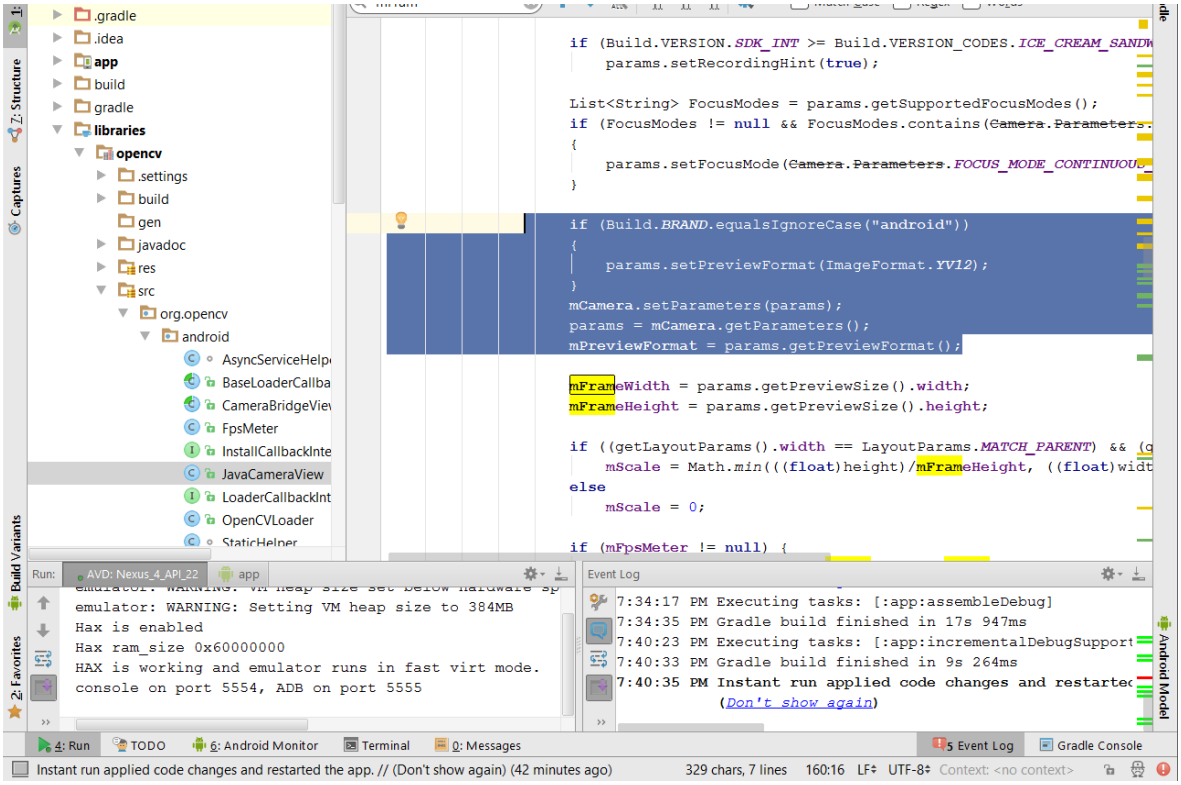
Now go ahead and run the app again.

If colors seem not as expected, there is something wrong with how the colors of the image are encoded at some layer of our abstracted camera.

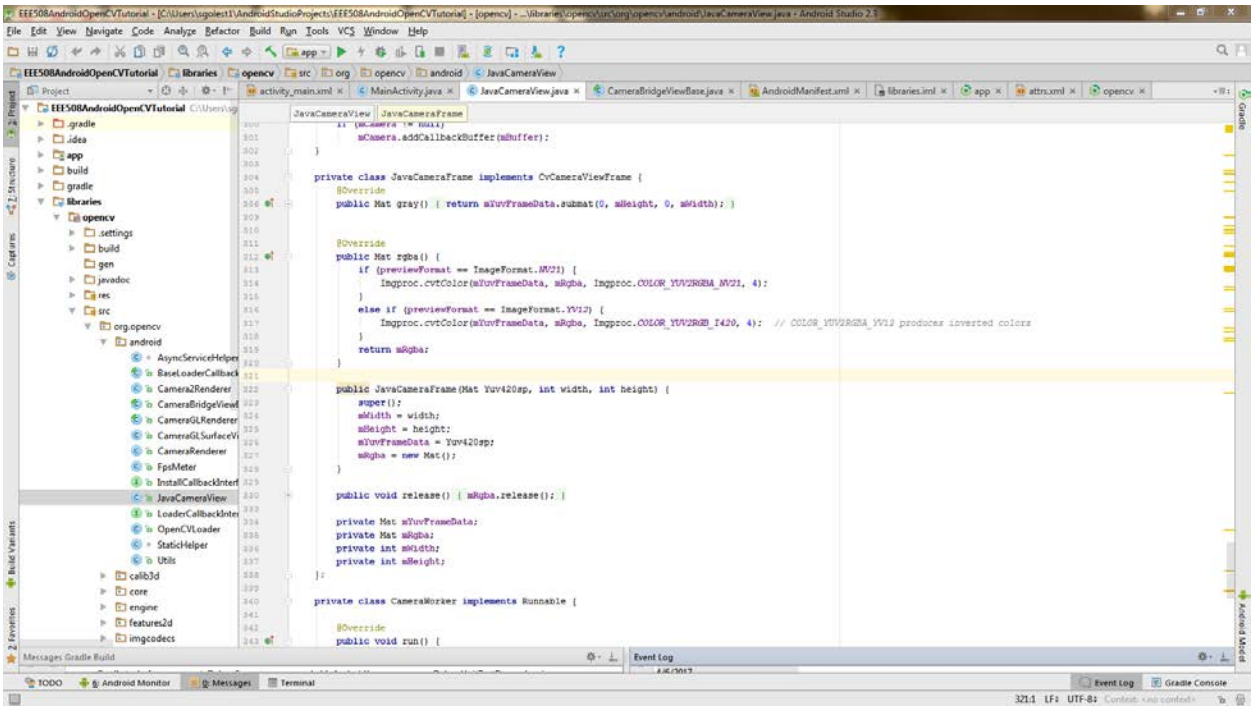
The solution to this problem is provided here (StackOverflow). Basically the encoding that we are using must be changed within the OpenCV's JavaCameraView class when we are using this type of emulator:



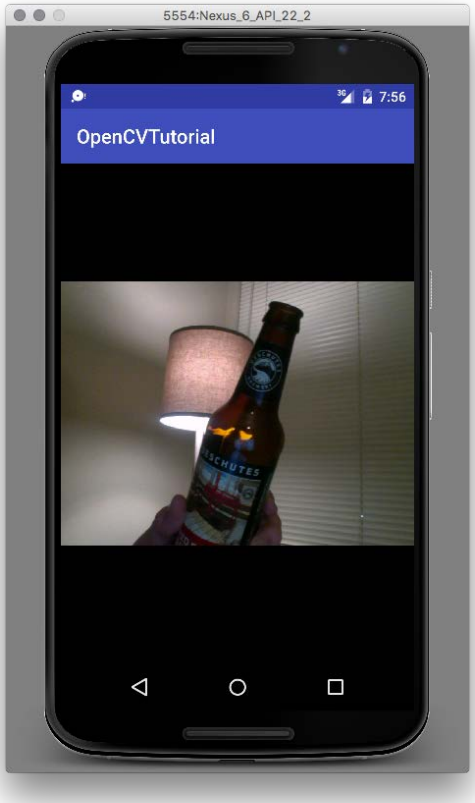
Add the following highlighted code to that class:



And modify the public Mat rgba() method:



And that should do it! Run your app again on your emulator and you should see the following:



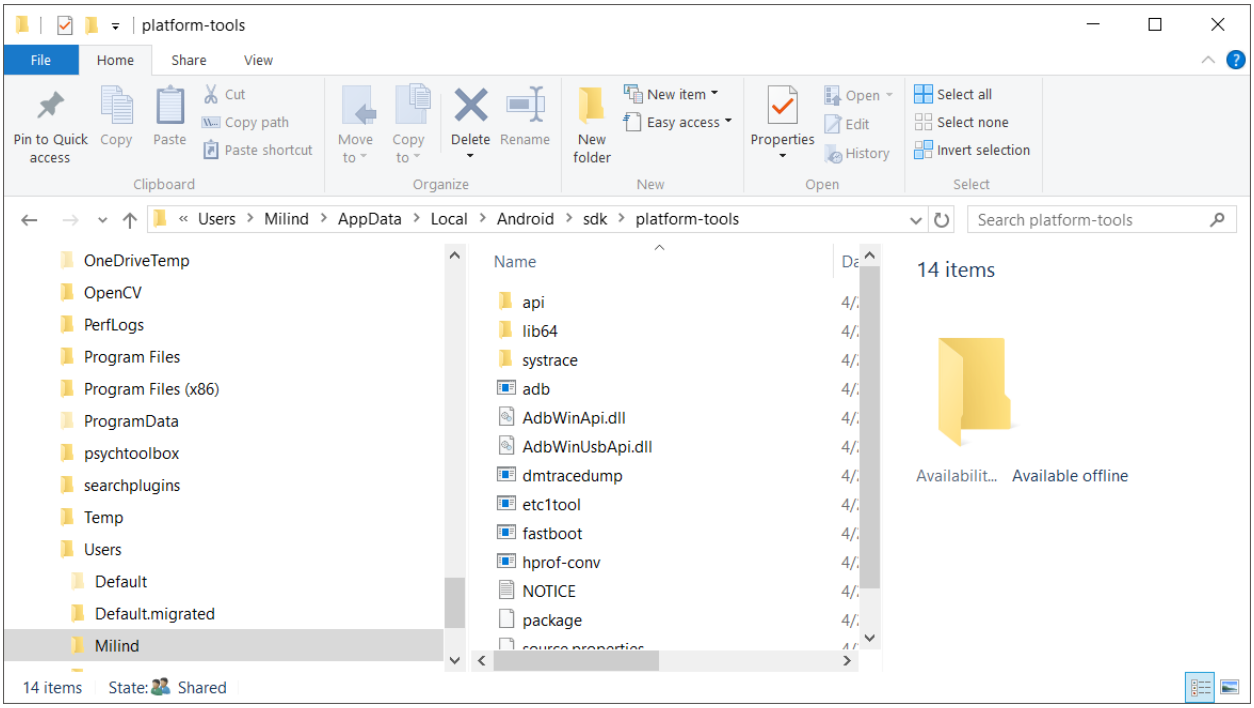
We are just about ready to move into the image processing phase of our app development! First we are going to quickly touch on how to get the OpenCV Manager installed on our emulator so that we know how to emulate a real-world working OpenCV app.

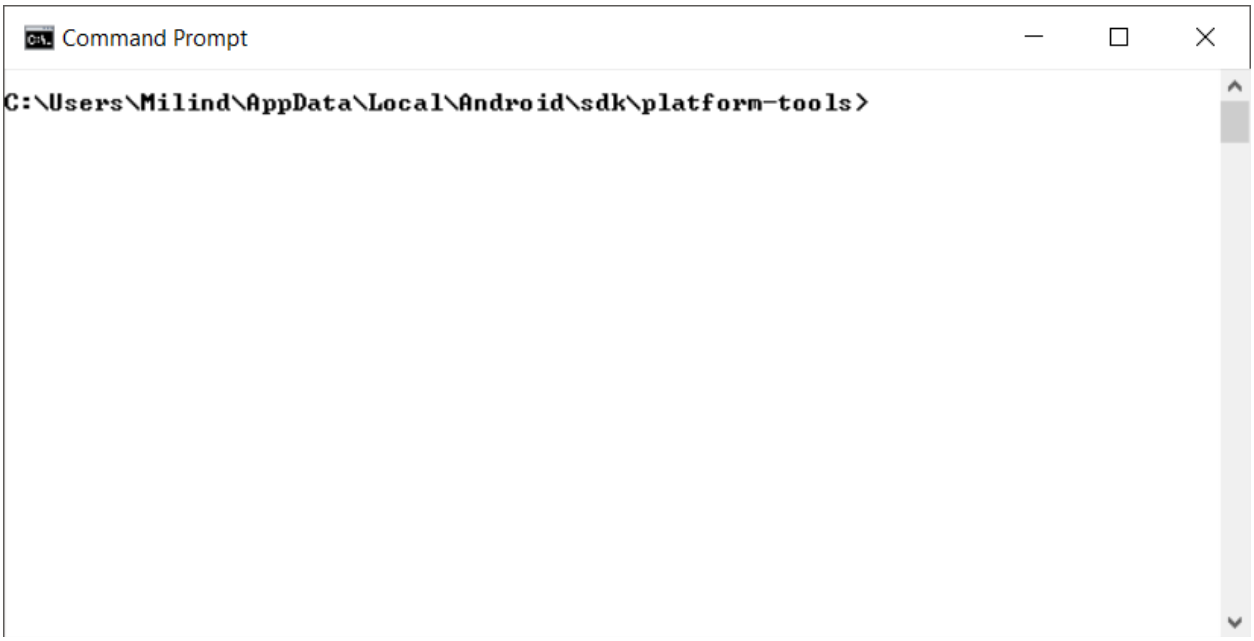
OpenCV Manager

Getting OpenCV Manager installed on the virtual device is significantly easier than the previous few steps of this tutorial.

What you need are the platform tools from the Android SDK. Go to where you have your Android SDK installed and go to the platform-tools directory. Navigate there on the command line.

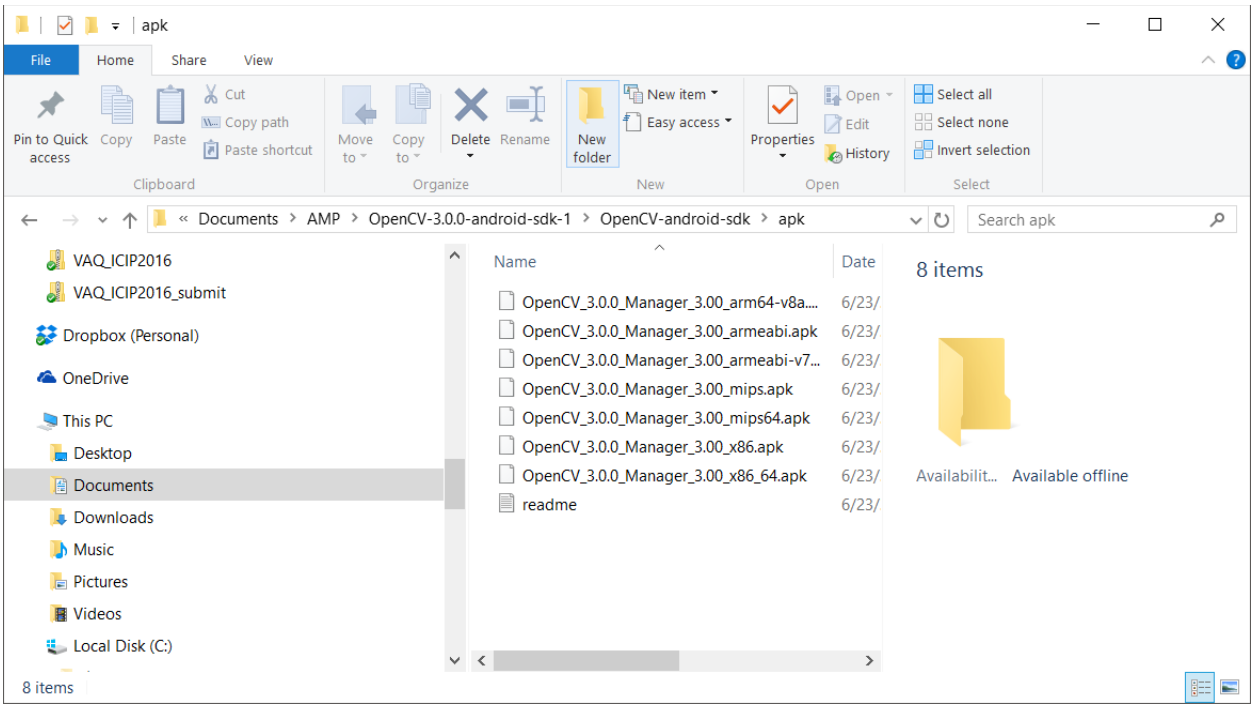
On Windows it is under Users\UserName\AppData\Local\Android\sdk\platform-tools





We will be using the adb executable in order to install the OpenCV Manager APK file onto our virtual device. The adb executable can also be used to install the OpenCV sample APK files from the OpenCV SDK if you want to play around with OpenCV Android apps.

Get the appropriate OpenCV Manager APK file for your emulator. For this tutorial it is the OpenCV_3.2.0_Manager_3.20_x86.apk file which is found in the folder where you extracted the OpenCV for Android zip file.

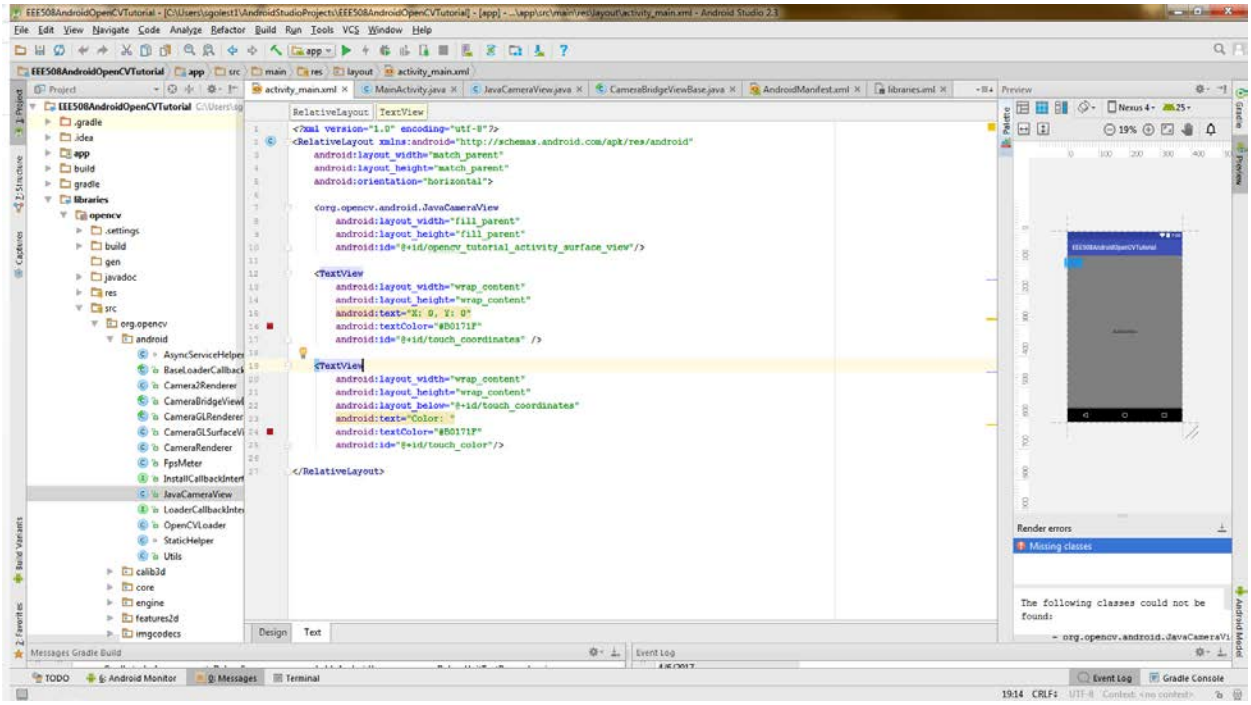


OpenCV Manager should now appear on your virtual device as its own app.

Now we are ready to move on to image processing!

Get Touch Event

In order to begin doing any image processing, we need to prepare for the readouts that we would like to see. We will do this by adding some TextViews to our layout XML file. In activity_main.xml, add the following:



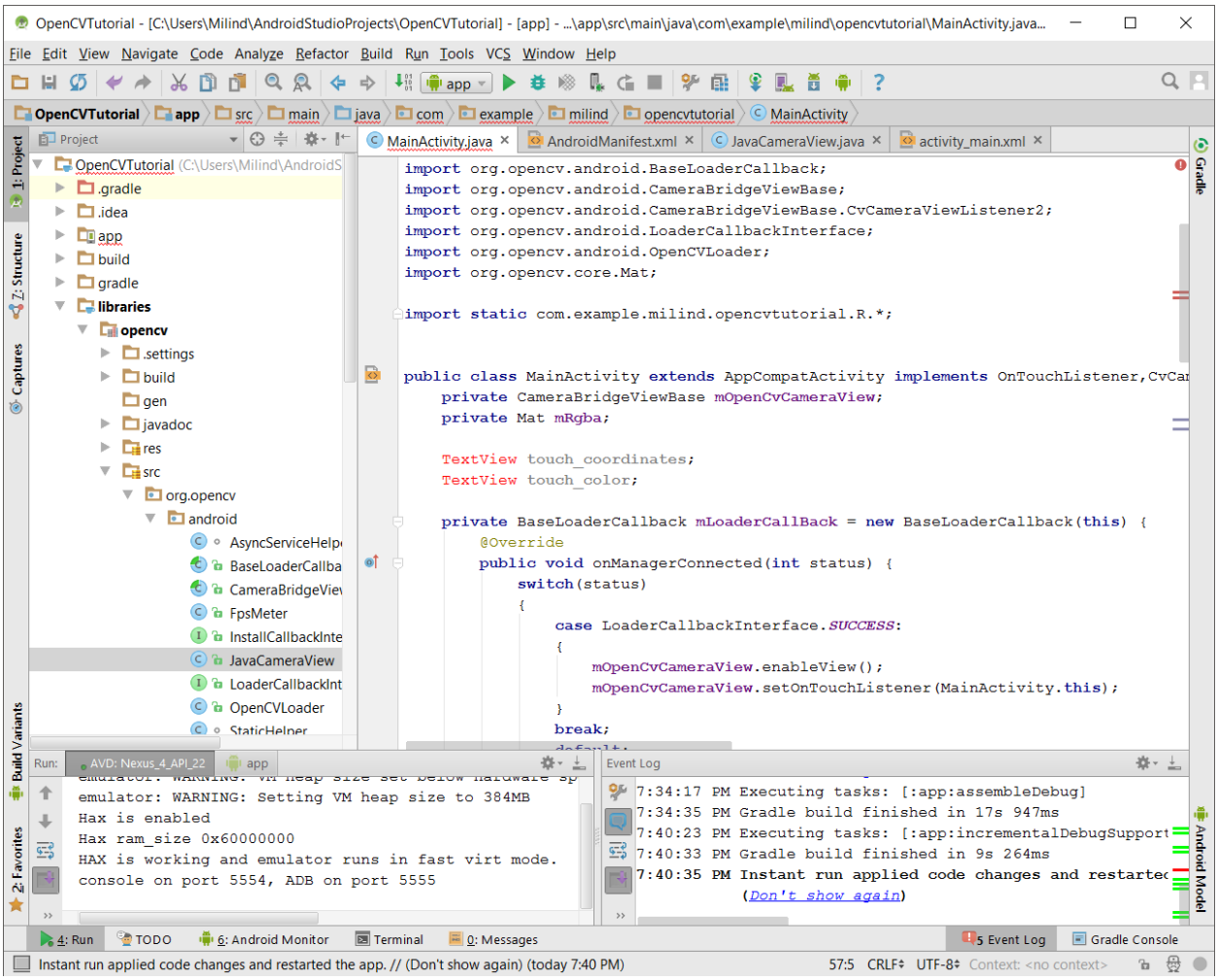
These text views will be used to display the coordinates that we touch and the immediate average of colors near where the touch event occurred.

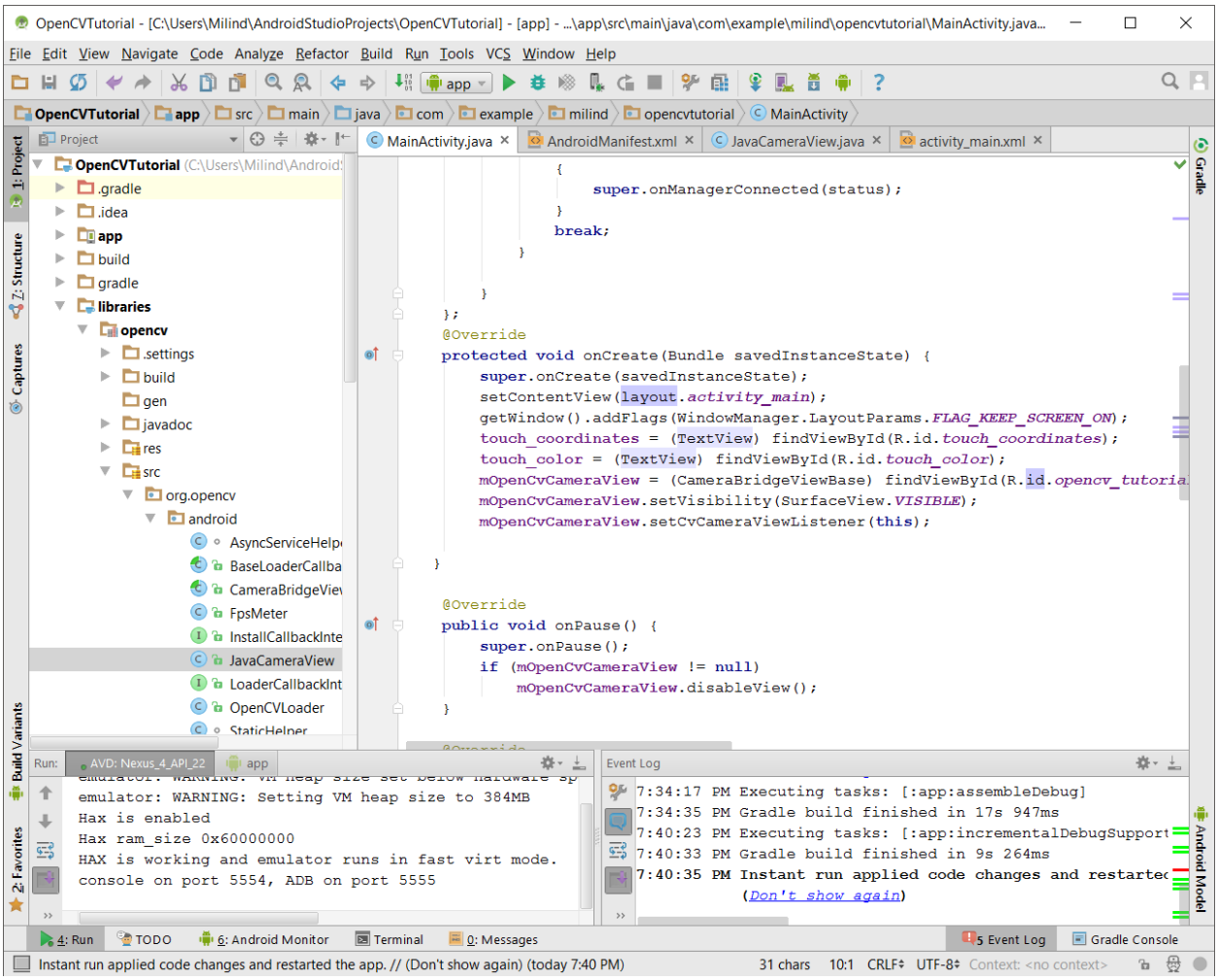
Let's first display the location that we touch to make sure that everything is working as expected. After that, we can display the color within the image that we selected.

The first thing we must do is create TextView variables in the MainActivity class so that we can modify the TextViews we just added to our layout. Find those TextViews by their ID and store them.

Make sure that you add the following lines:

```
TextView touch_coordinates;
TextView touch_color;
```


OpenCVTutorial - [C:\Users\Milind\AndroidStudioProjects\OpenCVTutorial] - [app] - ...\app\src\main\java\com\example\milind\opencvtutorial\MainActivity.java...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
OpenCVTutorial app src main java com example milind opencvtutorial MainActivity
MainActivity.java
AndroidManifest.xml
JavaCameraView.java
activity_main.xml
import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Mat;
import static com.example.milind.opencvtutorial.R.*;
public class MainActivity extends AppCompatActivity implements OnTouchListener,CvCa
private CameraBridgeViewBase mOpenCvCameraView;
private Mat mRgba;
TextView touch_coordinates;
TextView touch_color;
private BaseLoaderCallback mLoaderCallBack = new BaseLoaderCallback(this) {
@Override
public void onManagerConnected(int status) {
switch(status)
{
case LoaderCallbackInterface.SUCCESS:
{
mOpenCvCameraView.enableView();
mOpenCvCameraView.setOnTouchListener(MainActivity.this);
}
break;
emulator: WARNING: Setting VM heap size to 384MB
Hax is enabled
Hax ram_size 0x60000000
HAX is working and emulator runs in fast virt mode.
console on port 5554, ADB on port 5555
7:34:17 PM Executing tasks: [:app:assembleDebug]
7:34:35 PM Gradle build finished in 17s 947ms
7:40:23 PM Executing tasks: [:app:incrementalDebugSupport
7:40:33 PM Gradle build finished in 9s 264ms
7:40:35 PM Instant run applied code changes and restarte
(Don't show again)
Instant run applied code changes and restarted the app. // (Don't show again) (today 7:40 PM)

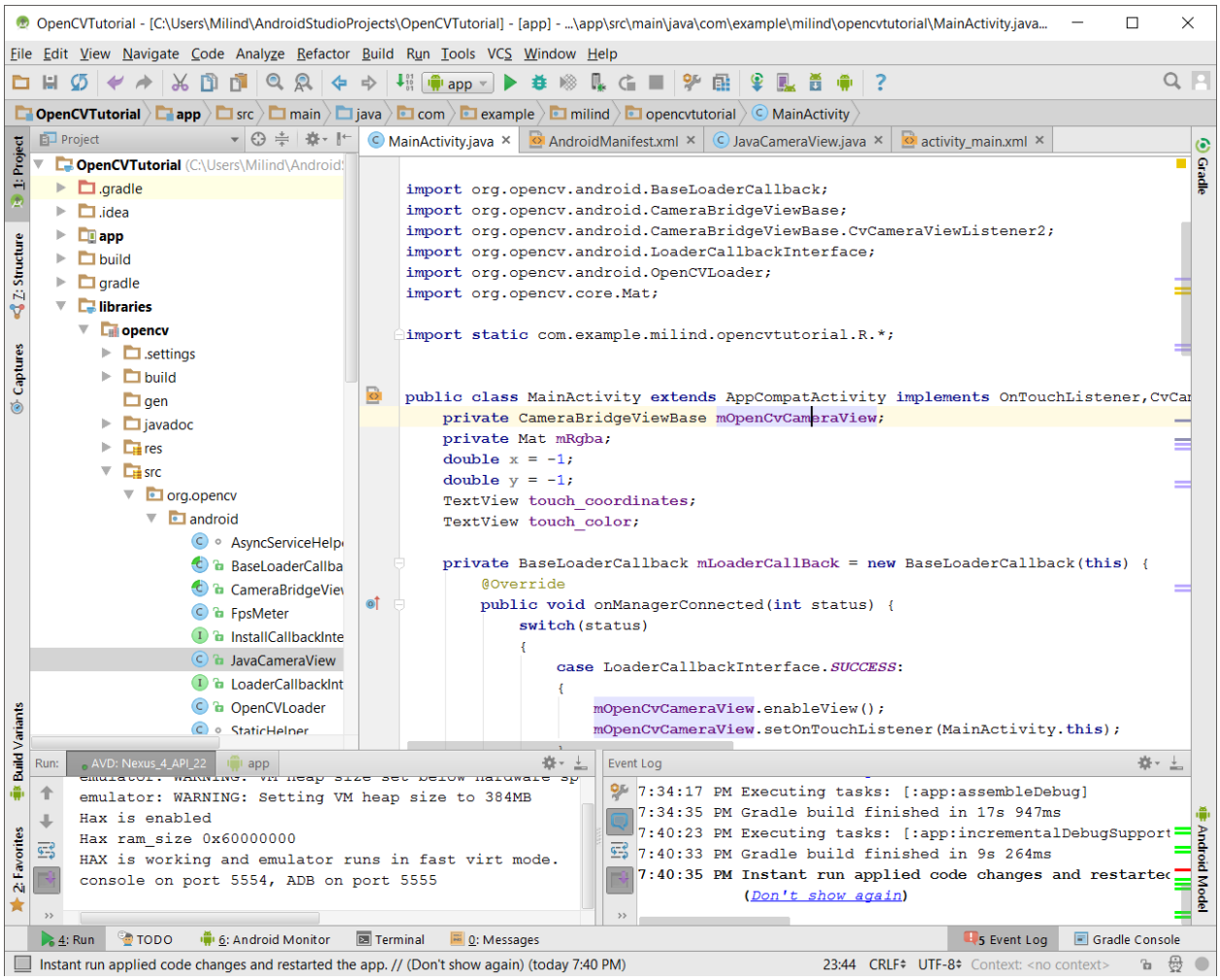


Use Alt+Enter to import the required modules for TextView **import** android.widget.TextView;

The method that we are implementing now is the one that was remaining from implementing the OnTouchListener part of this activity, the onTouch method.

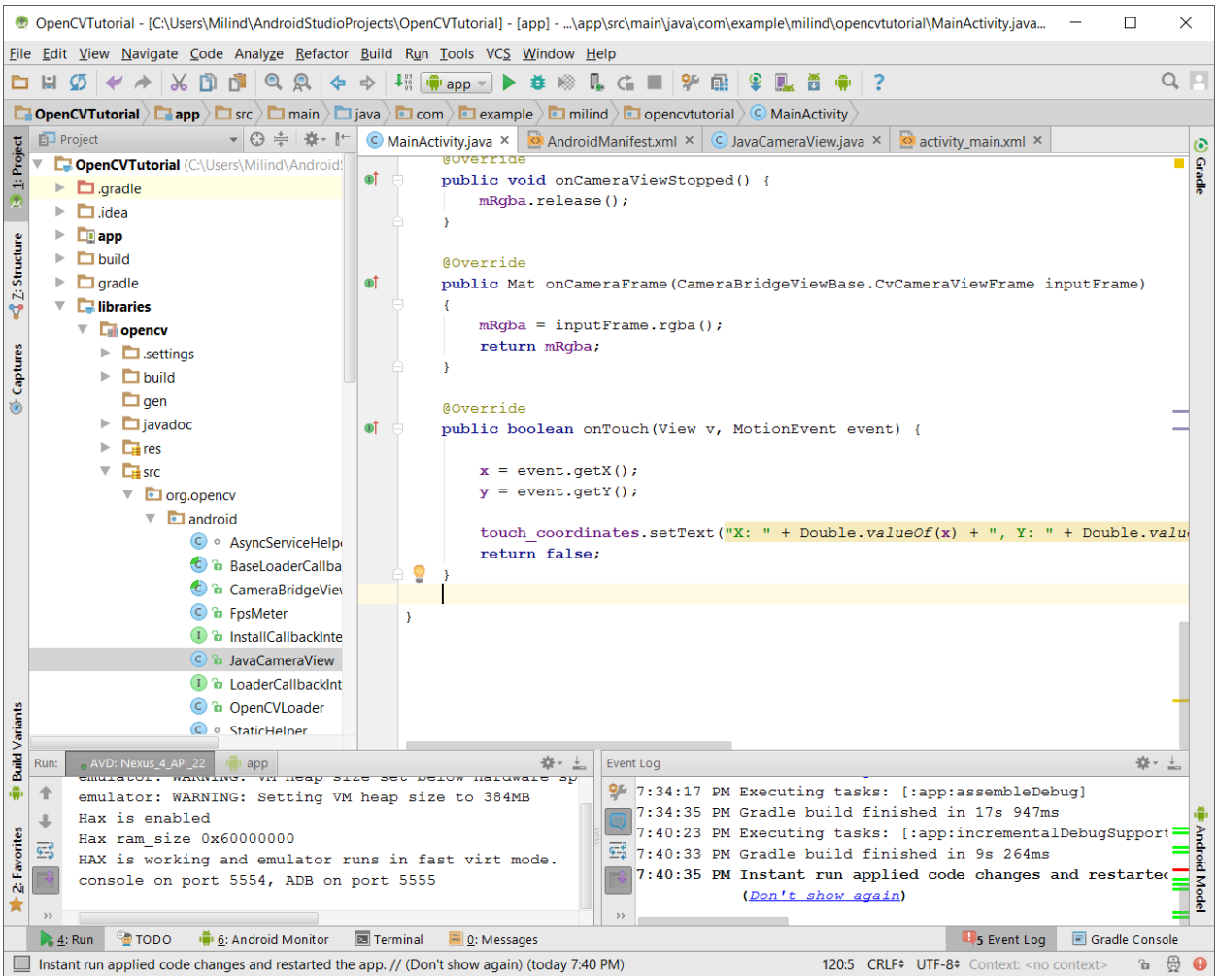
In order to display the coordinates that we touch on the screen, let us see how we process the touch event and store those coordinates.

The first thing we must do to log the touch coordinates is to declare some variables x and y that will store the location of the touch.



Now in the onTouch method, the event parameter can have getX() and getY() methods called on it. Do just that and store the results in the variables we declared above.

Now set the text of the coordinate TextView to display the coordinates that registered on the touch event.

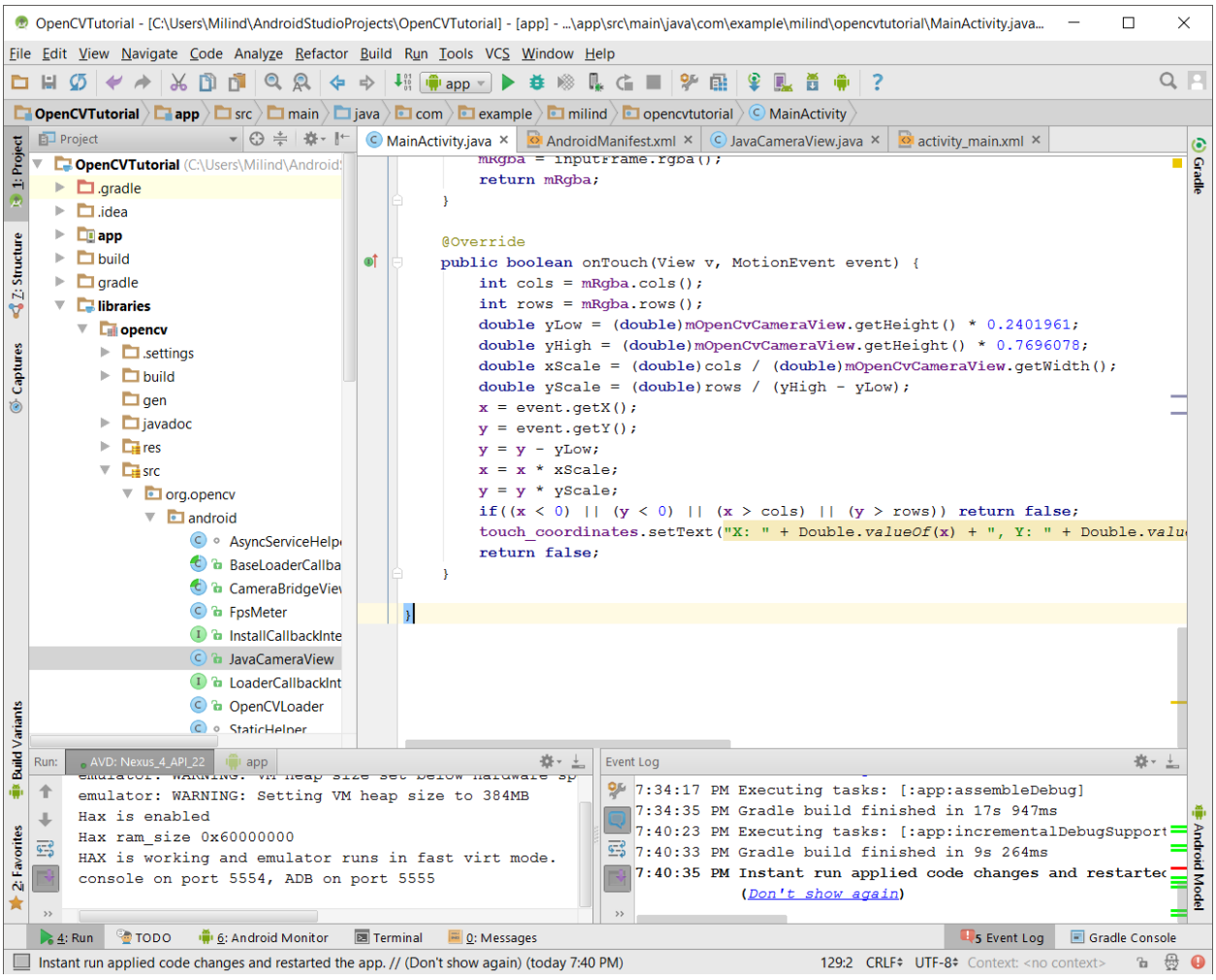


Go ahead and run your app and see that the coordinates that you touch display clearly in the TextView that we created.

Now in order to translate our touch position with the OpenCV frame matrix that we will use to process our image, we have an interesting challenge. The ratio of the mRgba matrix that we saw earlier is given in 640 rows and 480 columns (i.e., it is a 640x480 matrix). Our raw touch coordinates from the event parameter are not of this metric unfortunately.

This means that we have to normalize the touch coordinates to the RGBA matrix metric, where if we click outside of the image frame, we want to disregard that touch and not process the color there.

To do this, we must do the following:


OpenCVTutorial - [C:\Users\Milind\AndroidStudioProjects\OpenCVTutorial] - [app] - ...\app\src\main\java\com\example\milind\opencvtutorial\MainActivity.java...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
OpenCVTutorial app src main java com example milind opencvtutorial MainActivity
MainActivity.java
AndroidManifest.xml
JavaCameraView.java
activity_main.xml
mRgba = inputFrame.rgba();
return mRgba;
}
@Override
public boolean onTouch(View v, MotionEvent event) {
int cols = mRgba.cols();
int rows = mRgba.rows();
double yLow = (double)mOpenCvCameraView.getHeight() * 0.2401961;
double yHigh = (double)mOpenCvCameraView.getHeight() * 0.7696078;
double xScale = (double)cols / (double)mOpenCvCameraView.getWidth();
double yScale = (double)rows / (yHigh - yLow);
x = event.getX();
y = event.getY();
y = y - yLow;
x = x * xScale;
y = y * yScale;
if((x < 0) || (y < 0) || (x > cols) || (y > rows)) return false;
touch_coordinates.setText("X: " + Double.valueOf(x) + ", Y: " + Double.valu
return false;
}
}
Project
OpenCVTutorial (C:\Users\Milind\Android
.gradle
.idea
app
build
gradle
libraries
opencv
.settings
build
gen
javadoc
res
src
org.opencv
android
AsyncServiceHelp
BaseLoaderCallba
CameraBridgeVie
FpsMeter
InstallCallbackInte
JavaCameraView
LoaderCallbackInt
OpenCVLoader
StaticHelper
Run: AVD: Nexus_4_API_22 app
emulator: WARNING: Setting VM heap size to 384MB
Hax is enabled
Hax ram_size 0x60000000
HAX is working and emulator runs in fast virt mode.
console on port 5554, ADB on port 5555
Event Log
7:34:17 PM Executing tasks: [:app:assembleDebug]
7:34:35 PM Gradle build finished in 17s 947ms
7:40:23 PM Executing tasks: [:app:incrementalDebugSupport
7:40:33 PM Gradle build finished in 9s 264ms
7:40:35 PM Instant run applied code changes and restarte
(Don't show again)
4: Run
TODO
6: Android Monitor
Terminal
0: Messages
5 Event Log
Gradle Console
Instant run applied code changes and restarted the app. // (Don't show again) (today 7:40 PM)
129:2 CRLF UTF-8 Context: <no context>
1: Project
7: Structure
Captures
Build Variants
2: Favorites
Gradle
Android Model

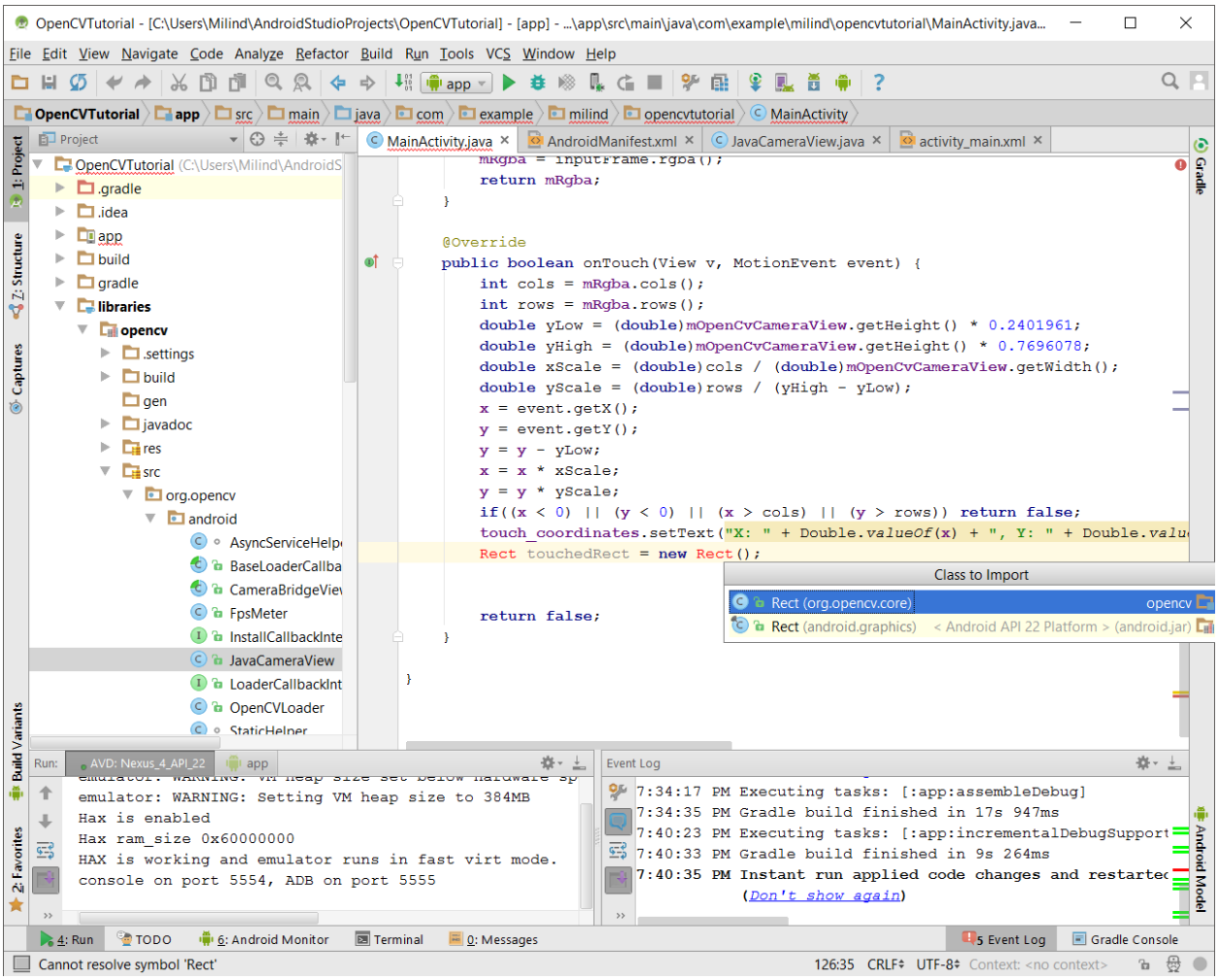
Now touching near the bottom right corner of the displayed image yields the coordinates:

These coordinates are within the bound we wanted for the RGB matrix. Now that we have coordinates that are usable in terms of the OpenCV matrix, we are ready to process the color located at the touch location

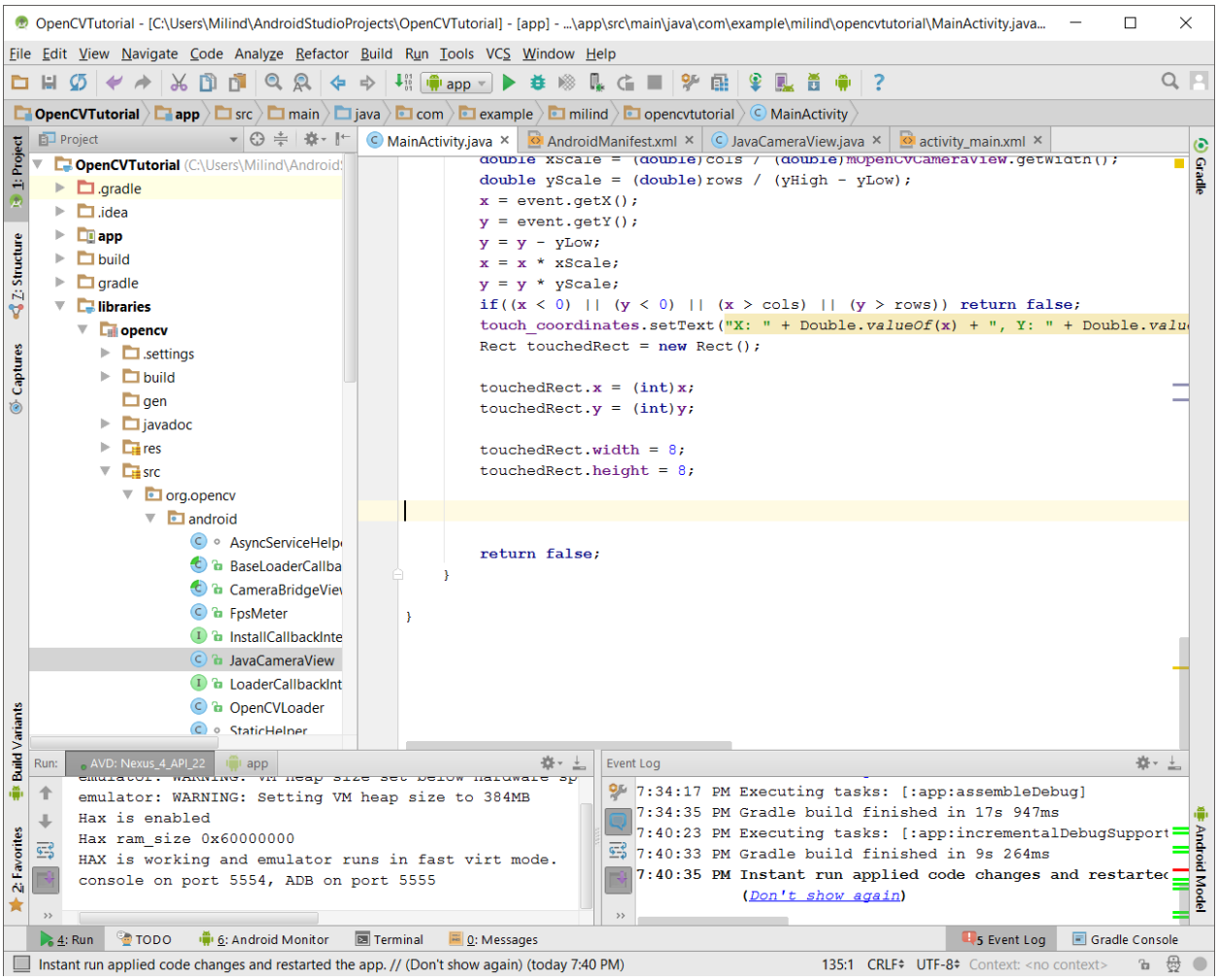
Get RGB Color on Touch

The first thing we must do to get the color at a touch location is take a rectangular slice immediately around the touch location. We want to take a small region around the touch and get the average color there.

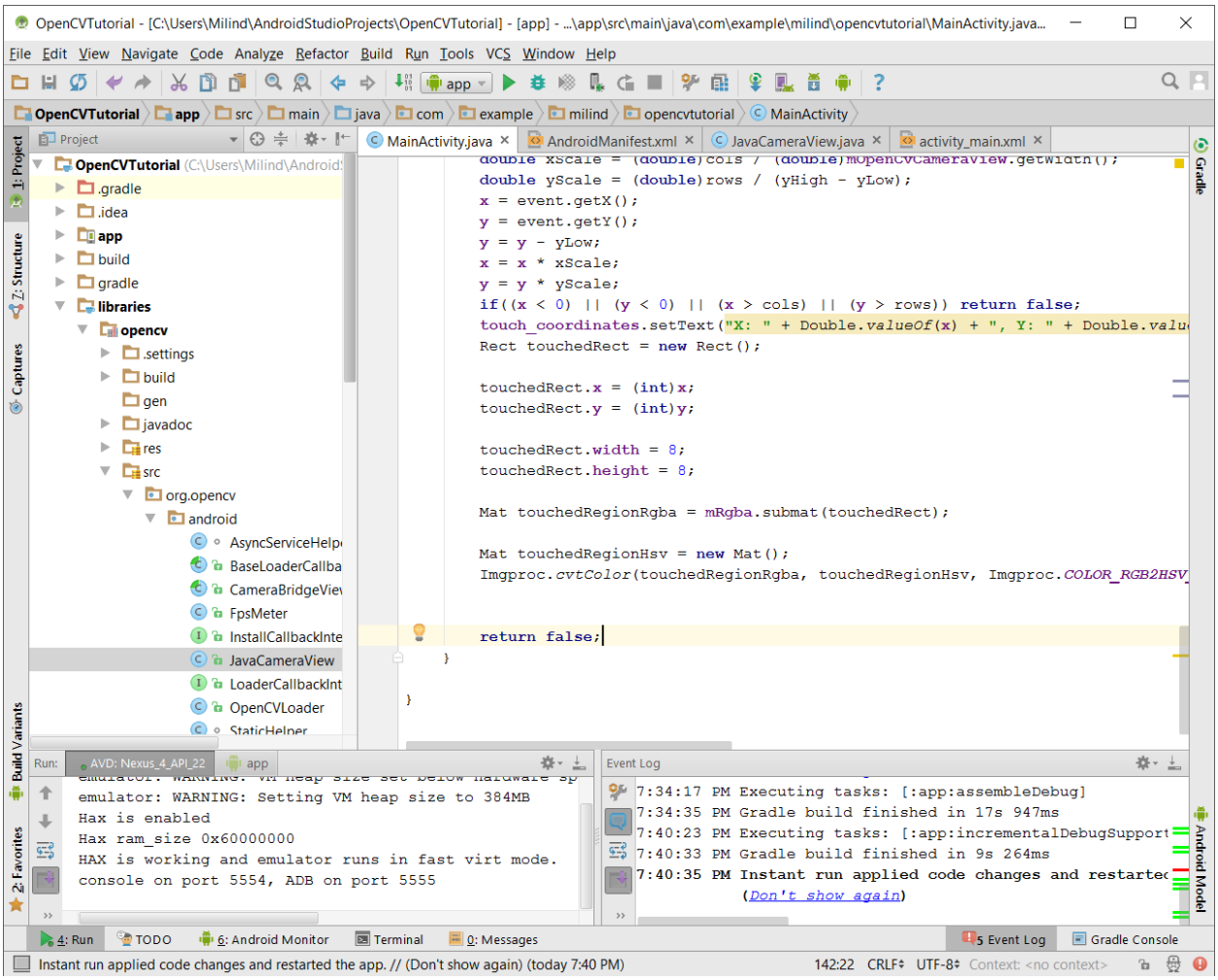
In order to do this we need the Rect class from the OpenCV core library. Be sure to import the OpenCV Rect, as there is a Rect class in the Android graphics library as well.



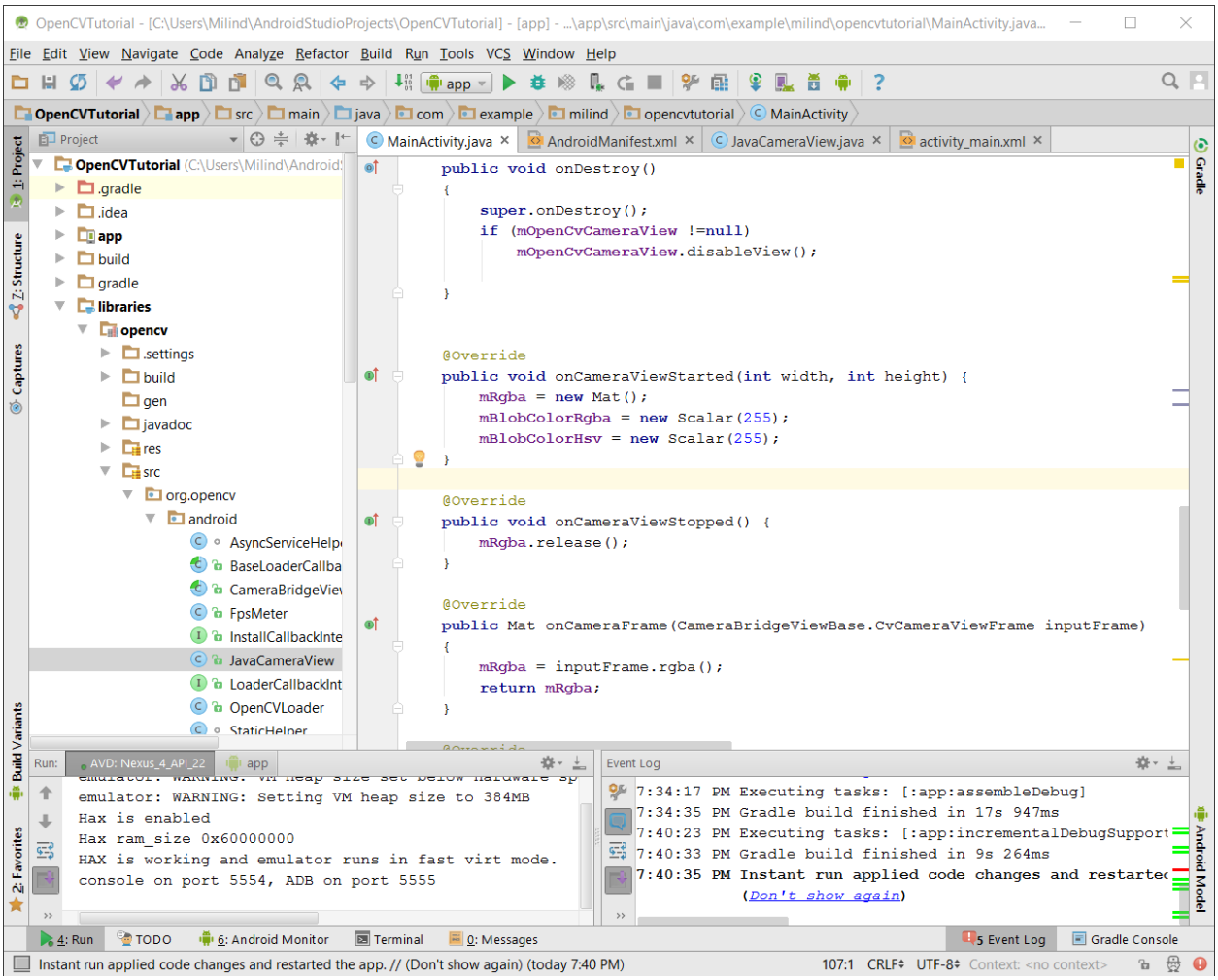
Set the x and y values of the touched Rect area to the values of the normalized touch coordinates, casted as integers. We will take the touched Rect area's height and width to be 8 rows/columns each, though we can change this depending on how large we would like the area from which to take the average color.



Now we want to get the sub-matrix from the OpenCV matrix of just the touched region. Once we do that, we should convert our RGB matrix to HSV in order to calculate the average color of the region.



Now that we have our sub-matrices of the touched regions, we need to define scalar values of RGB and HSV that we'll use in calculating the average color of the region. Initialize these variables in the onCameraViewStarted method, alongside the initialization of the mRgba matrix.

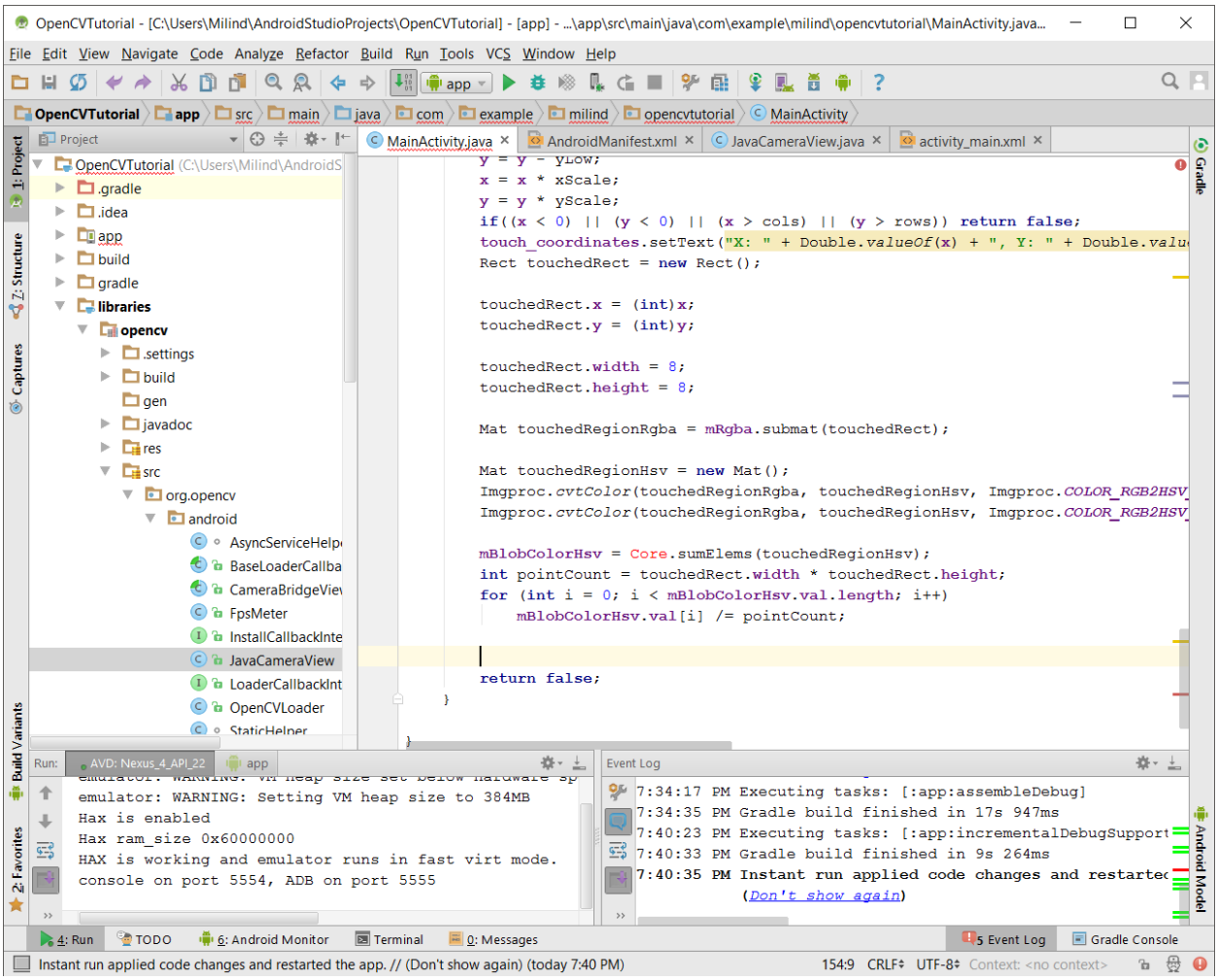


In order to get the average color of the region, use the sumElems method from the OpenCV core library, passing the HSV touched region sub-matrix as an argument. Store this value in the HSV scalar variable. Store the number of points at which we'll consider the color in an integer variable, called pointCount below.

Make sure you have

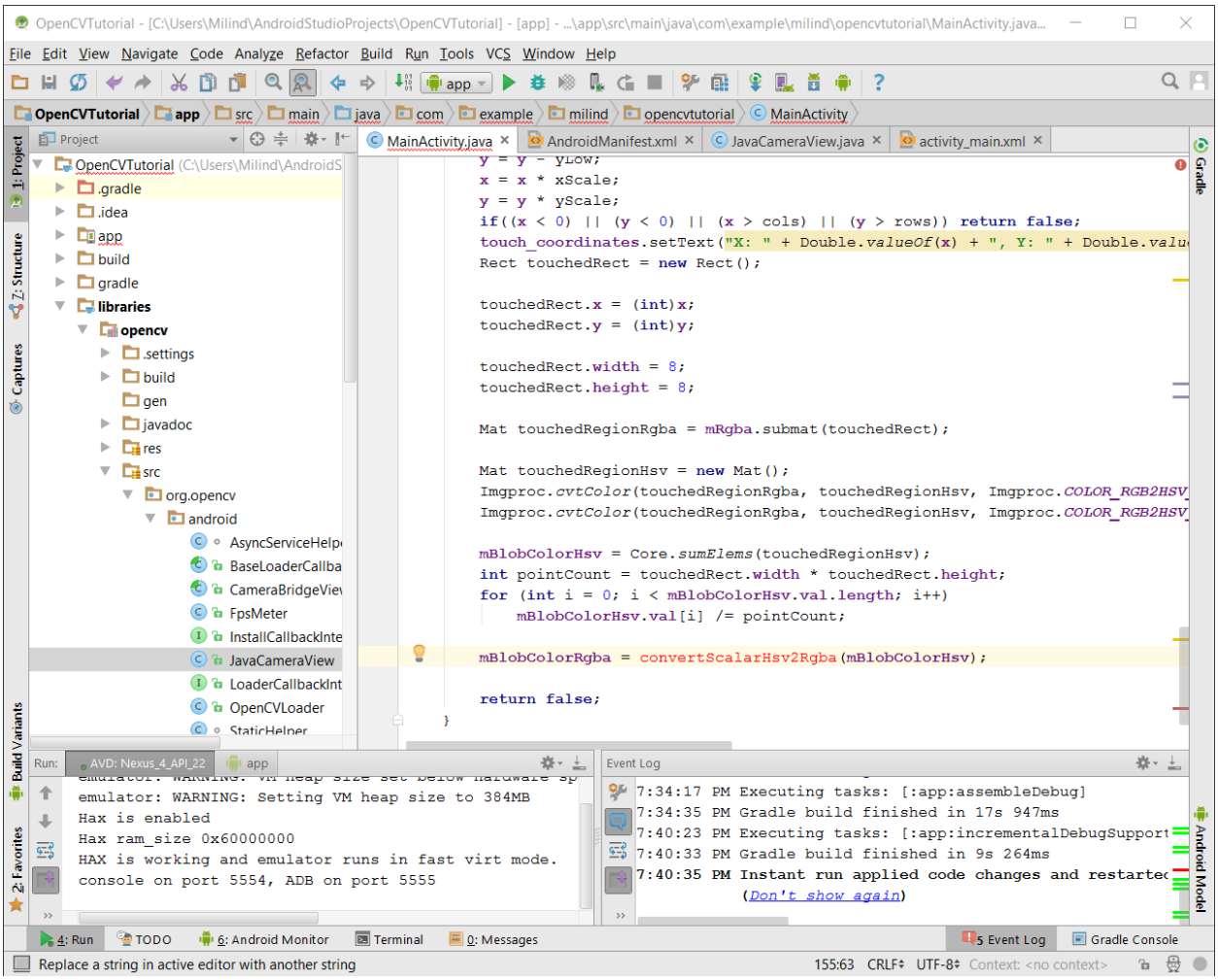
```
private Scalar mBlobColorRgba;
private Scalar mBlobColorHsv;
```

added in your code.



What's happening here in detail is that we are taking our 8x8 sub-matrix and considering the HSV value at each pixel. These 64 pixels' hue, saturation, and value are all added together with the sumElems method. We then take the average hue, saturation, and value by dividing each by the number of values we summed for each, giving us our average color for the touched region.

Now we need to convert our average color, stored as our HSV scalar and convert it to the format that we would like, RGB.



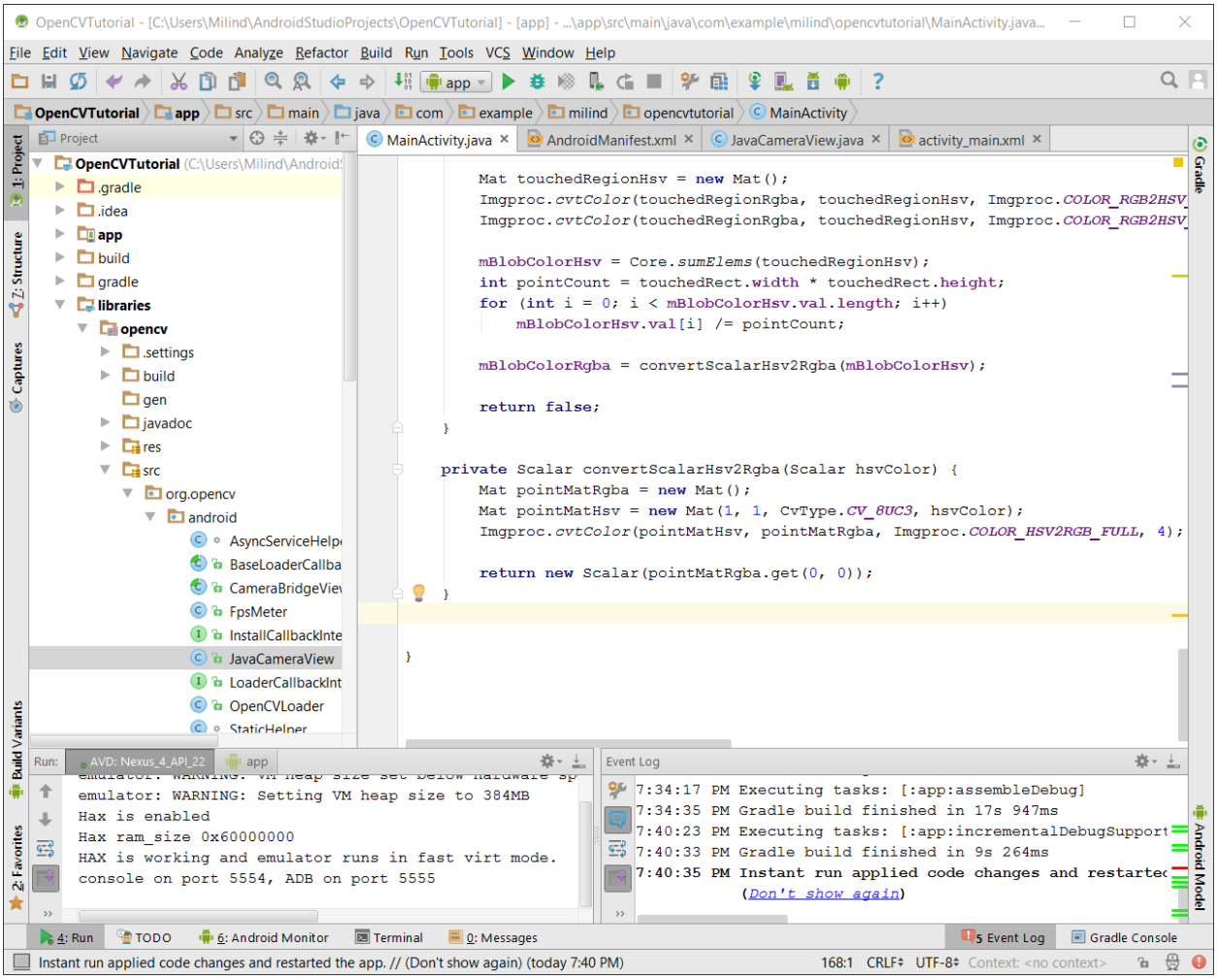
Let's implement this new conversion method.

The easiest way to convert from our HSV scalar will be to use the Imgproc cvtColor method from the OpenCV image processing library. In order to use this method, we need to turn our HSV scalar into an HSV matrix and convert the matrix to RGB.

Declare a new RGB matrix. We'll call it pointMatRgba. Note that we will only have one HSV value in our HSV matrix when we convert from the scalar, since we have already taken the average color of our touched region.

Declare a new HSV matrix, passing in the arguments 1, 1, CvType.CV_8UC3, and the HSV scalar variable (our only parameter passed into this method). The two 1's define our dimension (a 1x1 matrix) and the CvType.CV_8UC3 defines our matrix values as 8-bit (8), unsigned integer (U), with 3 channels (C3). This makes sense, as we want a 1x1 matrix containing the encoding for 1 HSV value.

Now we can call the Imgproc cvtColor method, passing our two matrices and the conversion specification as arguments. We specify 4 channels in the destination matrix, as we have the Alpha channel in addition to RGB. Return this converted RGB matrix as a new scalar.

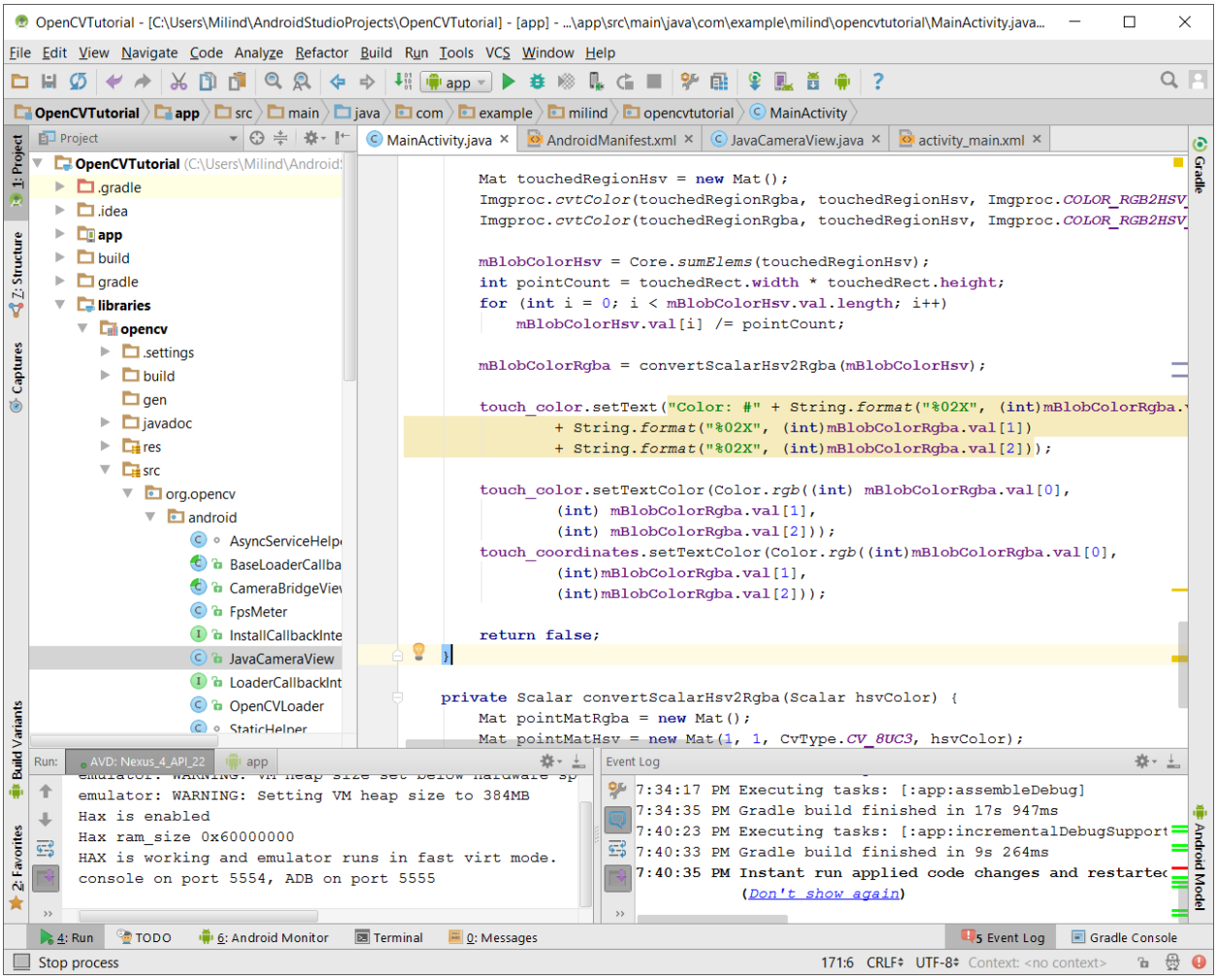


So far, we have:

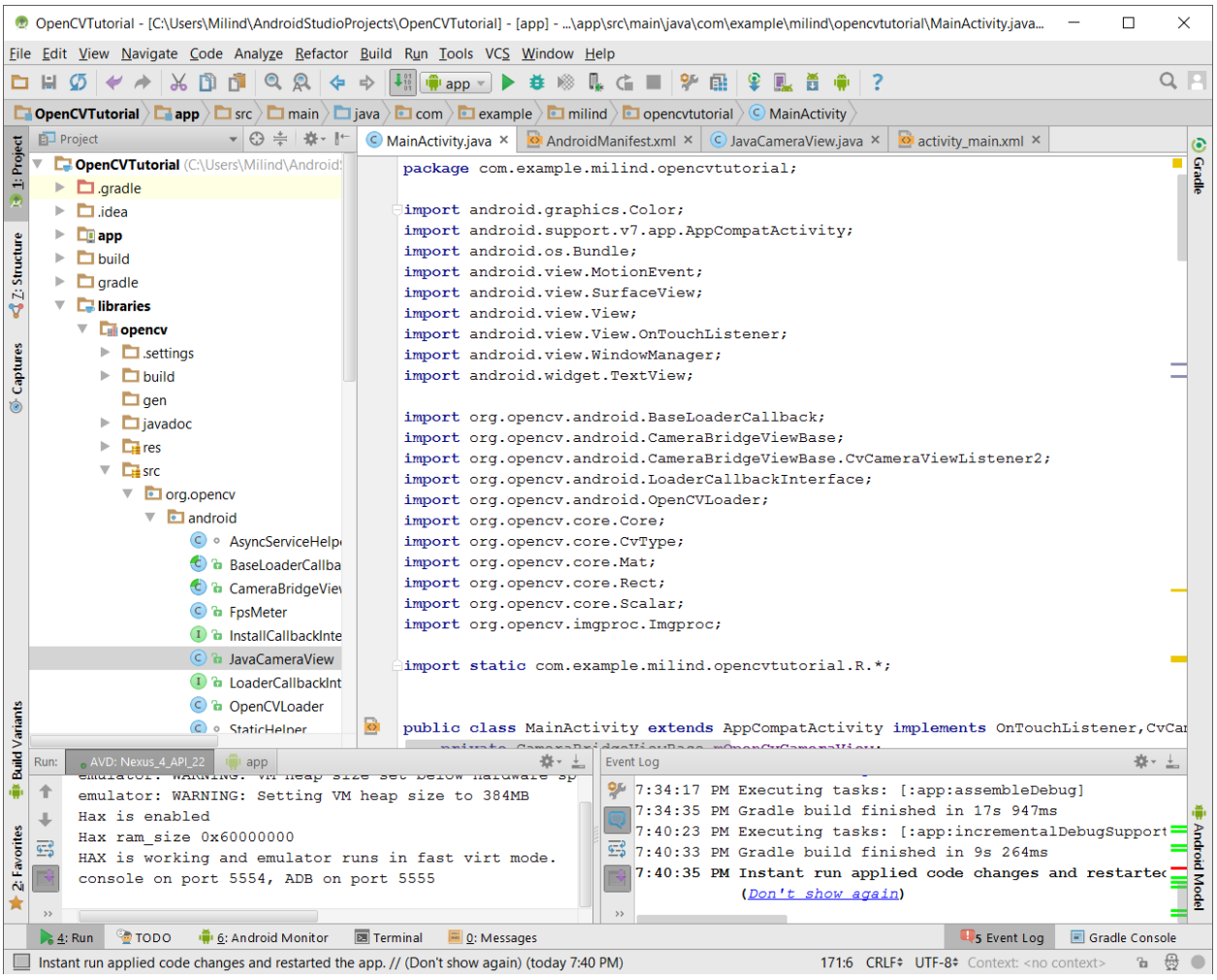
- Emulated an Android camera using our webcam
- Displayed our camera feed using the OpenCV library
- Used OpenCV to process the frames fed in by the camera
- Determined and displayed the coordinates of a touch event
- Normalized the touch event coordinates to an OpenCV RGBA matrix
- Used the RGBA matrix to calculate the average color of a touched region

Now we just have to print our hex color value that we already calculated! This is the easy part.

Use the touch_color TextView we created earlier to print our RGB values, using the TextView's setText method. The RGB values are doubles, so for the sake of formatting, let's display each RGB decimal value as a 2-digit hex value, casting each value as an integer in order to do the string formatting.



Just for good measure, these are all of the components we imported into this activity module:


OpenCVTutorial - [C:\Users\Milind\AndroidStudioProjects\OpenCVTutorial] - [app] - ...\app\src\main\java\com\example\milind\opencvtutorial\MainActivity.java...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
OpenCVTutorial app src main java com example milind opencvtutorial MainActivity
MainActivity.java
AndroidManifest.xml
JavaCameraView.java
activity_main.xml
package com.example.milind.opencvtutorial;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnTouchListener;
import android.view.WindowManager;
import android.widget.TextView;
import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;
import static com.example.milind.opencvtutorial.R.*;
public class MainActivity extends AppCompatActivity implements OnTouchListener,CvCa
emulator: WARNING: Setting VM heap size to 384MB
Hax is enabled
Hax ram_size 0x60000000
HAX is working and emulator runs in fast virt mode.
console on port 5554, ADB on port 5555
7:34:17 PM Executing tasks: [:app:assembleDebug]
7:34:35 PM Gradle build finished in 17s 947ms
7:40:23 PM Executing tasks: [:app:incrementalDebugSupport
7:40:33 PM Gradle build finished in 9s 264ms
7:40:35 PM Instant run applied code changes and restarte
(Don't show again)
Instant run applied code changes and restarted the app. // (Don't show again) (today 7:40 PM)

Setting up your phone for running the app

The tutorial uses an Android Virtual Device for running the developed app. However, it is better to test the app on an actual phone. This section describes the steps needed to setup your phone as a testing device.

Installing USB drivers

The first step is to install USB drivers for your phone on your PC. Go to the following link

You can go to https://developer.android.com/studio/run/oem-usb.html and look for your phone manufacturer to download the corresponding drivers.

Enable USB Debugging on your phone

The next step is to enable USB debugging on your phone. If you have not enabled "Developer Options" before, you will first need to enable them by going to Settings-> About Phone /About Device-> Software Info and tap on Build Number 7 times in quick succession to enable Developer Options. Next, you should see an option under Settings called Developer Options and in it you should see an option to enable USB debugging. Then, connect your phone to the PC via a USB cable and now when you run the code on Android Studio, if the drivers have been installed properly, you should see your phone listed in the "connected devices" and should be able to deploy the application on your phone and debug it in real-time.

Part C – App Development

Design an image or video processing app by extending and adding new features to the touch color app implemented in Part A. Provide a Readme file describing the App that you designed including the objectives of your app, its application, new features implemented, and how to run it/use it, and a section clearly describing the contributions of each individual in the team. Submit by the project due date in a zip folder named Lastname1Lastname2_Project2.zip, where Lastname1 and Lastname2 are the last names of the project partners, the Readme file, code, sample input and output of the app.

REFERENCES:

1. https://www.raywenderlich.com/120177/beginning-android-development-tutorial-installing-android-studio

2. http://people.oregonstate.edu/~robinsti/CS_496/Tutorial/

3. http://blog.codeonion.com/2015/11/25/creating-a-new-opencv-project-in-android-studio/

4. https://zami0xzami.wordpress.com/2016/03/17/opencv-for-mobile-devices-using-android-studio/

5. https://developer.android.com/reference/android/widget/RelativeLayout.html

6. https://thinkandroid.wordpress.com/2010/01/14/how-to-position-views-properly-in-layouts/

7. https://developer.android.com/studio/run/managing-avds.html

8. http://www.viralandroid.com/2015/08/how-to-make-android-studio-fast.html

9. http://stackoverflow.com/questions/16732021/why-emulator-is-very-slow-in-android-studio

10. http://stackoverflow.com/questions/2662650/making-the-android-emulator-run-faster

11. http://stackoverflow.com/questions/16669779/opencv-camera-orientation-issue

12. http://answers.opencv.org/question/105036/frame-image-transmitted-is-upside-down-in-android/

13. http://answers.opencv.org/question/7313/rotating-android-camera-to-portrait/

14. http://stackoverflow.com/questions/3841122/android-camera-preview-is-sideways