**Video 2**

1. Import opencv module (file import module)
2. Add opencv to project structure (file > project structure > dependencies)
   1. Select app > press plus button > select OpenCVlibrary345
3. Copy libs folder from SDK to app’s (apps>src>main). Rename folder to ‘jnilibs’
4. AndroidManifest.xml > specify camera permissions
5. Import some OpenCV modules!

**Video3: opening camera and processing frames**

1. To add an opencv view to a layout we use the term <org.opencv.android.JavaCameraView> in activity\_main.xml
2. Our main activity class should implement CameraBridgeViewBase.CvCameraViewListener2
3. Create on cameraFrame, oncameraviewstarted, cameraviewstopped methods. You can kind of just start typing these out. These are our main triggers and they deal with what happens when we receive frames/startcamera/stopcamera
4. Define variables CameraBridgeViewBase and baseloadercallback. We use these to launch the camera view in oncreate and allow us to receive frames from camera
5. In the layout give the javacameraview an id <android:id="@+id/CameraView"/>
6. In oncreate we set up the cameraBridgeViewBase variable by declaring it as the cameraview in the layout, then setting it to visible, then setting it to a CVcameraview listener object.
7. We then create a method called baseloadercallback which checks that everything loads correctly with a switch case. if loaded correctly we enable the view
8. We added some code to onresume which delivers a error message if there’s some kind of bug or starts the app if there’s no problem using the baseloadercallback with baseLoaderCallback.SUCCESS as the argument.
9. In onpause: check if the cameraBridgeViewBase (our view) has a value. If it does not have a value ( if it is null) then we disable it.
10. Add the same stuff to onDestroy()
11. Ivan wants to build this landscape. We add this to manifest.xml android:screenOrientation="landscape"
12. We now turn our attention to onCameraFrame. This gets triggered everytime there is a new frame (as many as 20 times a second) and can be used to change the frames as they come in using matrixes (Mat). Thereis no numpy in Java so this is the equivalent.
13. In onCameraFrame we make a variable called frame and set it to inputFrame.rgba() (colour + alpha), and make it return the camera frame. The app is now just a camera.

**Video 4: using openCv with buttons**

1. We add some logic for what is going to happen when our frames come in
2. We then add a button and add an onClick marker for it – we name the onclick marker Canny (edge detection)
3. Make a public, void function called Canny that takes our button as an argument. The button just turns the startCanny variable from true to false. In our onCameraFrame method we then have a block which turns on edge detection only if startCanny is equal to true (i.e when our button is pressed)

**Video 5: Using Yolonet**

1. Ivan recommends loading the DNN weights directly from external memory because downloading them is a bit more tricky at this stage
2. We change our button to a Yolo toggler
3. We make a variable called firstTimeYolo which signals whether it’s the first time running the app (to copy over the weights). We need permissions for this first time.
4. Then we import org.opencv.dnn.Dnn, and create a Net variable called tinyYolo (this will eventually be used to load our network variables
5. We make a switch called firstTimeYolo – if this is false we load the two strings which tell us where our config files and weights are. We use Environment.getExternalStorage to give us the root file of our external storage directory. We then add in the additional parts of the path for the dnn folder. We add these variables in arguments to the reaNetfromDarknet method.
6. Now for our onCameraFrame method. We need to preprocess the image to a ‘blob’. We scale it and do not convert it to BGR (swapped blue and red channels)

**Video 6: Running custom trained yolos**