Deep Learning Model On-device Deployment

Introduction

The final stage of delivering an artificial intelligence project is the deployment. The deployment is the process for making your models available in production environments, where they can provide predictions to other software systems. Model deployment can be divided into two broad categories based on the method of inference: server-based and on-device. Server-based inference means that the model is stored in a remote server that can be accessed by any client device connected to the internet. On the other hand, on-device inference means that the model is stored in the local device where it can directly handle input data processing and inference to get the result. In Lab 2, we tried to deploy our model using the server-based inference.

In this lab guide, we will deploy our model using the on-device inference, more specifically we will deploy our model in an Android application. In the last part of this guide, the designated homework is explained.

Objective

The students are expected to learn how to deploy trained model in mobile. You will create an Android application to run a classification model trained to classify dogs and cats.

Background

The main difference when running machine learning and deep learning models on Android systems is that we do not get to use a Python interpreter in standard development and during runtime. Recall that running Python scripts require us to use the interpreter as shown below.

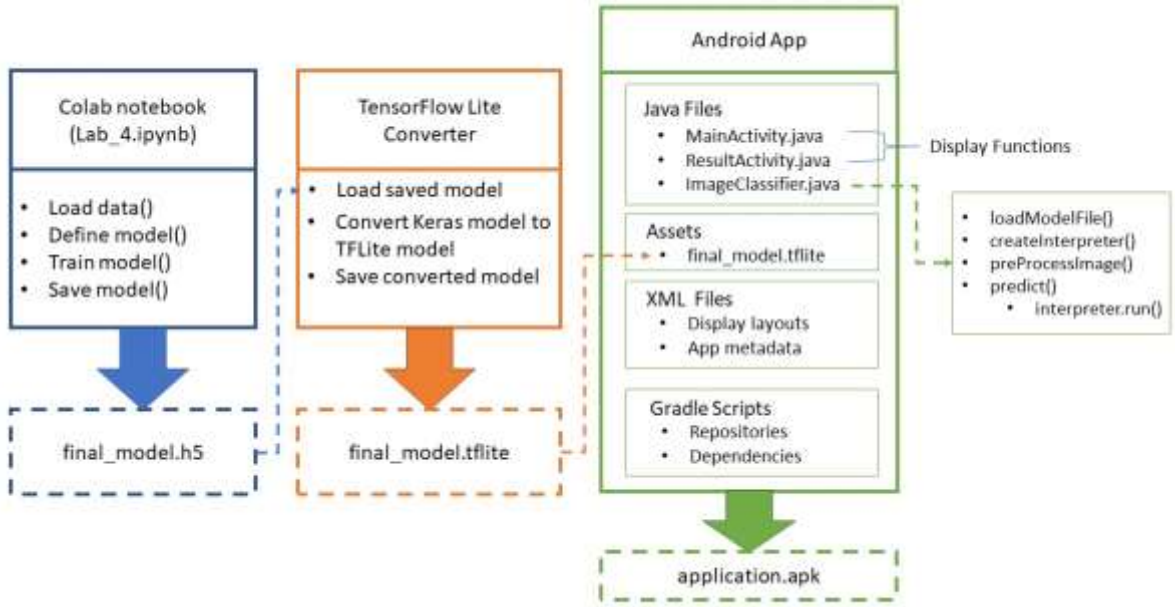
```
1   !python hello.py
```

The Python interpreter takes the path of the script as an argument and then compiles the script to a bytecode that only the Python virtual machine (PVM) understands. The PVM then executes the bytecode. In this manner, the code is portable and can run on any underlying hardware. However, Python is not a supported language for Android development.

In this lab, we will use Java to create the Android app. Java is one of the official languages for Android development. Java also works in the same manner in that the code is compiled to a bytecode that only the Java virtual machine (JVM) understands and executes. The compilation process for Android apps is slightly different from other Java applications. Instead of the standard JVM, Android has its own distinct byte-code format called Dalvik. From versions 1.0 to 4.4, Android apps run on their own Dalvik virtual machines (DVM). Since version 4.4, Android migrated to the Android Runtime (ART) from DVM. With ART, the Dalvik bytecodes are translated into native machine

code during installation of the app. When the app is launched, it runs in its own process with its own instance of ART and the machine code is run directly (native execution).

The straightforward approach to handle this issue is to either develop your models using frameworks written in Java or to rewrite your Python inference code in Java. The first option is not advisable since Python is the standard in model development and Java frameworks are not well-established yet. The second option requires you to implement the logic of your model in Java, which might be complicated in some algorithms. Fortunately, TensorFlow has a JAVA API that can run on any JVM for building, training and deploying models. However, TensorFlow developers recommend using the TensorFlow Lite (TFLite) library instead when deploying on Android devices. Your trained models will be converted to Google's proprietary optimized data format (.tflite) that is ideal for embedded systems applications and TFLite provides the JAVA API to interpret the model structure and run inference for Android applications as illustrated below.



To do this, we will follow the steps below:

1. Train and save model

2. Convert the saved model to TFLite model

3. Create Android application that includes implementation library of tensorFlow Lite

4. Build Android application installer (.apk file)

Implementation

Install Android Studio

Before preparing our model, you need to download Android Studio to develop the Android application which is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. The installation can be done by following the click-through instructions.

Model Preparation and Code Translation

In this section, you will be guided to train your model using pre-trained model VGG16 to classify dog vs. cat. After training, the model will be translated to TFLite format and loading it to your Android application for development.

Step 1: Prepare your model. You will be using the dog vs cat dataset to classify dogs from cats. To download and unzip the dataset, you can run the script below.

```
# to download the dataset
!gdown --id 1K5Ai6robPRva0eKDOrPLRvQ_6gpTIaDe
# to unzip the file
!unzip finalize_dogs_vs_cats.zip
```

And you can run the script below to train the VGG16 model with modified classifier layer. After training, the model will be saved as *.h5* file. (You can access the full code in here)

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

# define model
def define_model():
    model = VGG16(include_top=False, input_shape=(224, 224, 3))
    # mark loaded layers as not trainable
    for layer in model.layers:
        layer.trainable = False
    # add new classifier layers
    flat1 = Flatten()(model.layers[-1].output)
    class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
    output = Dense(1, activation='sigmoid')(class1)
    # define new model
    model = Model(inputs=model.inputs, outputs=output)
    # compile model
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# run the test harness for evaluating a model
def run_test_harness():
    # define model
    model = define_model()
    # create data generator
    datagen = ImageDataGenerator(featurewise_center=True)
    # specify imagenet mean values for centering
    datagen.mean = [123.68, 116.779, 103.939]
    # prepare iterator
    train_it = datagen.flow_from_directory('finalize_dogs_vs_cats/',
                                           class_mode='binary', batch_size=64, target_size=(224,224))
    # fit model
    model.fit_generator(train_it, steps_per_epoch=len(train_it), epochs=5, verbose=1)
    # save model
    model.save('final_model.h5')

# entry point, run the test harness
run_test_harness()
```

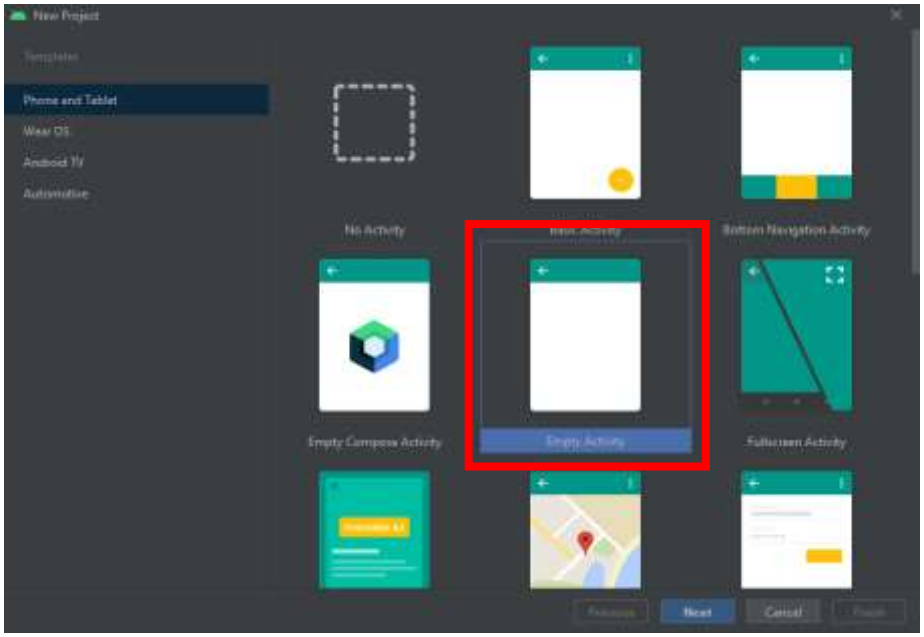
Step 2: Convert your model to a TFLite format. The Keras model can be converted using the TFLiteConverter (e.g. lines 8-9). After converting, save it which will have *.tflite* file extension.

```
1 from keras.models import load_model
2 import tensorflow as tf
3
4 # load saved model
5 model = load_model('final_model.h5')
6
7 # convert Keras model to TFLite model
8 converter = tf.lite.TFLiteConverter.from_keras_model(model)
9 tflite_model = converter.convert()
10
11 # save converted model
12 with open('dogvscat_class_model.tflite', 'wb') as f:
13   f.write(tflite_model)
```

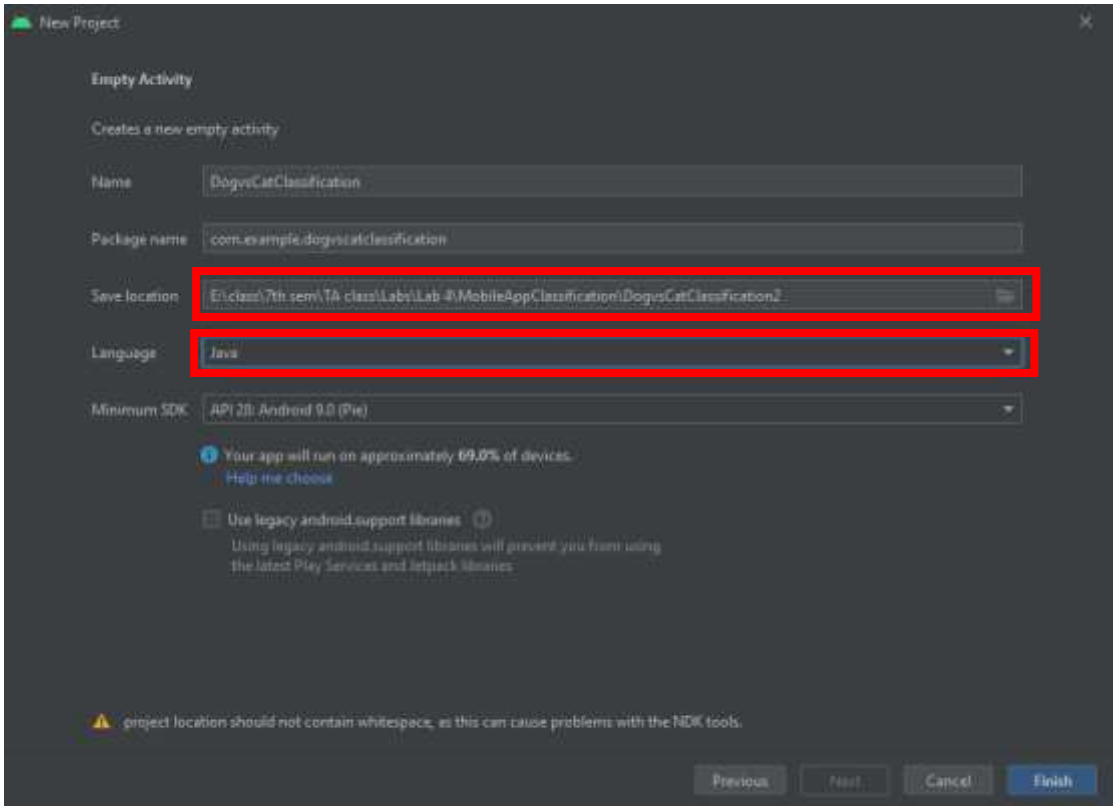
Coding the Android App

In this section, you are going to create an Android application and integrate the translated model.

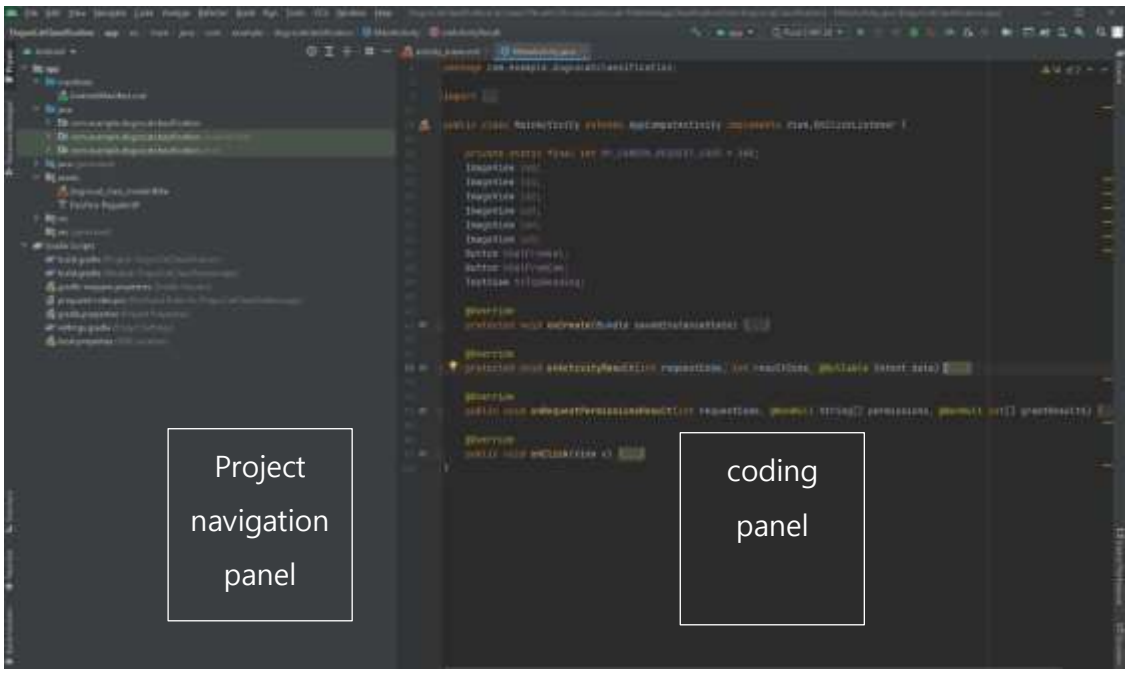
Step 1: Launch the Android Studio and click **Create New Project** and choose **Empty Activity**.



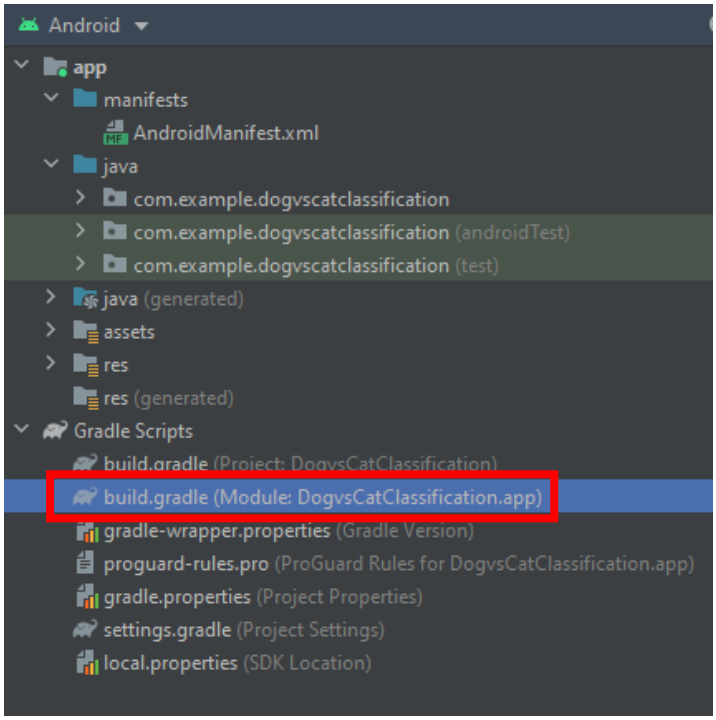
Step 2: Select **Java** as the programming language and choose **API 28** for the minimum SDK requirement. *(If your android phone has lower Android version, you can choose lower API, e.g. API 26 – Android 8.0 (Oreo))*. You are free to name your project whatever you want. Click **Finish**. *(If there are missing packages or components that are required for the Android Application, Android Studio will automatically download these. Wait for the setup to be completed before proceeding to the next section)*



When the setup is done, it will take you to the main integrated development environment (IDE). The main panel is where you write all your Java codes for the logic and XML scripts for the user interface design. The panel on the left is for project navigation.

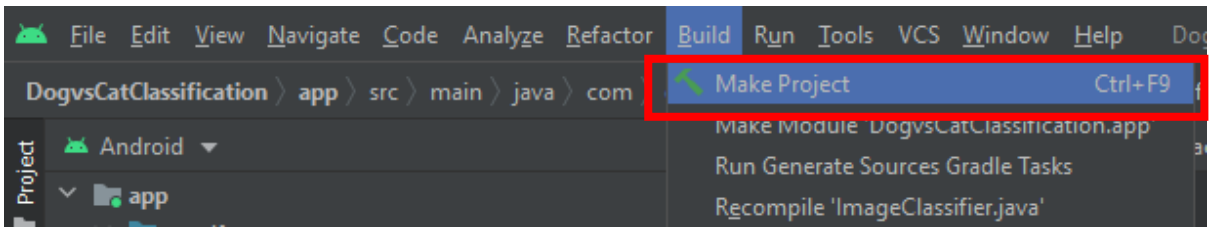


Step 3: In the project navigation panel, double click the **build.gradle** for the app. Under the **dependencies**, import the implementation for TFLite.

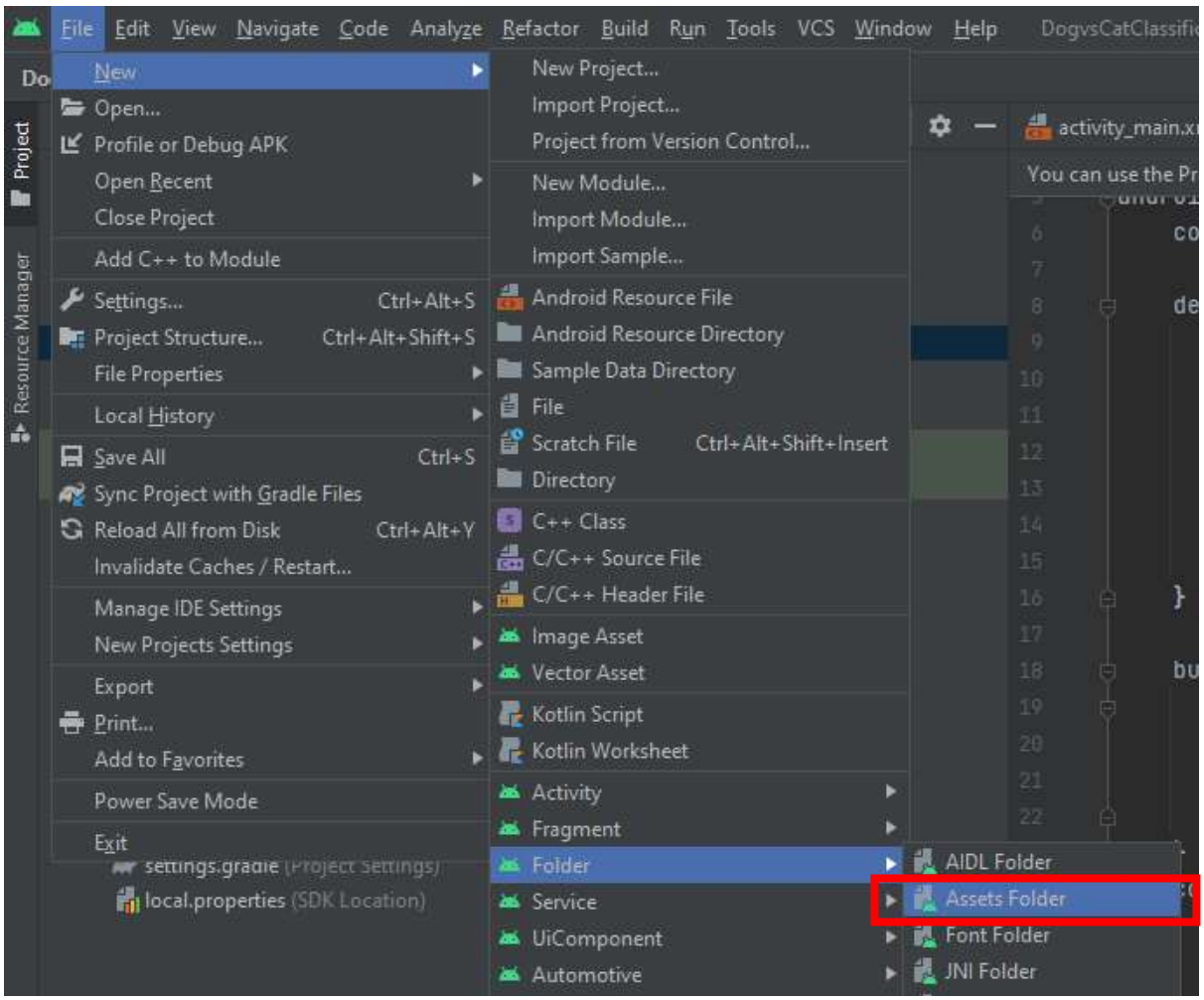



```
dependencies {

    implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'

    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

From here, **build** the project in order for the TFLite to be included in the project. *(Make sure to build the project before proceeding to avoid errors in importing the TFLite library)*

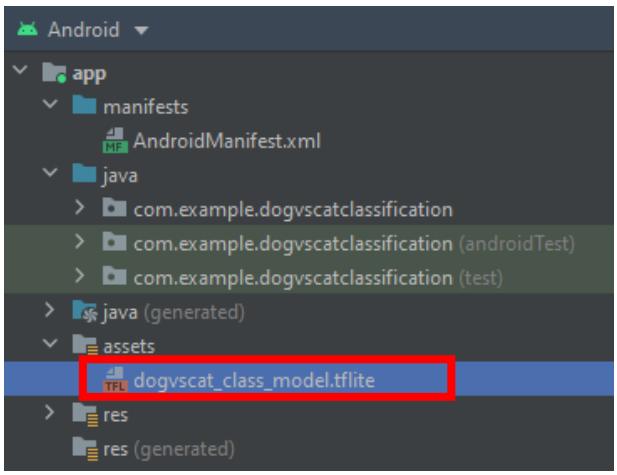


Step 4: Create an Assets folder by going to **app→ Folder → Assets Folder**.

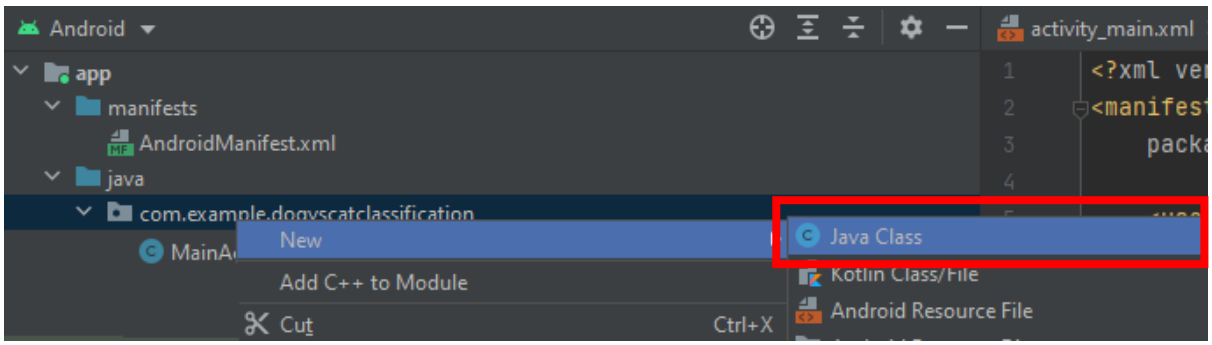


Step 5: Copy your TFLite model to this assets folder.

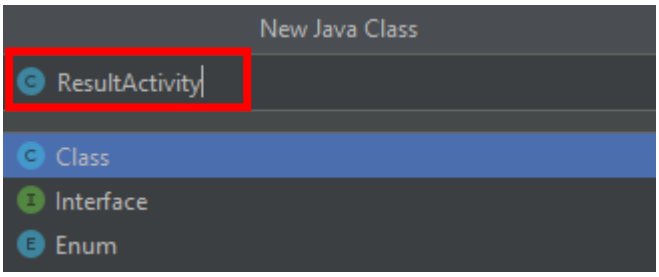
Example:
E:\class\7thsem\TAclass\Labs\Lab4\MobileAppClassificationOreo\DogvsCatClassification\app\src\main\assets



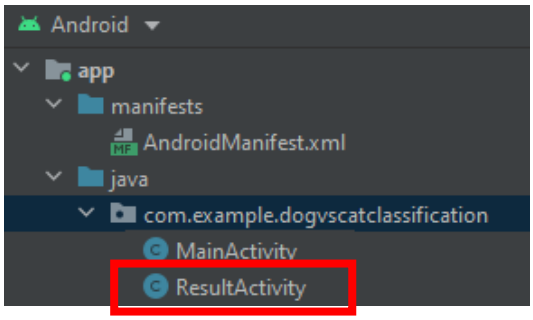
Step 6: In the project navigation go to **app → java** and right-click on the first directory which is also the package name of your project. Click **on New → Java Class**.



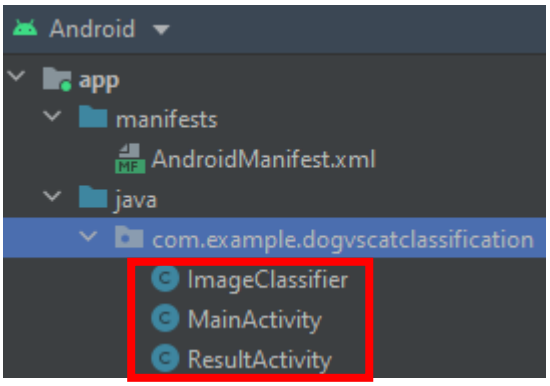
Type in **ResultActivity** as the name of your new Java Class.



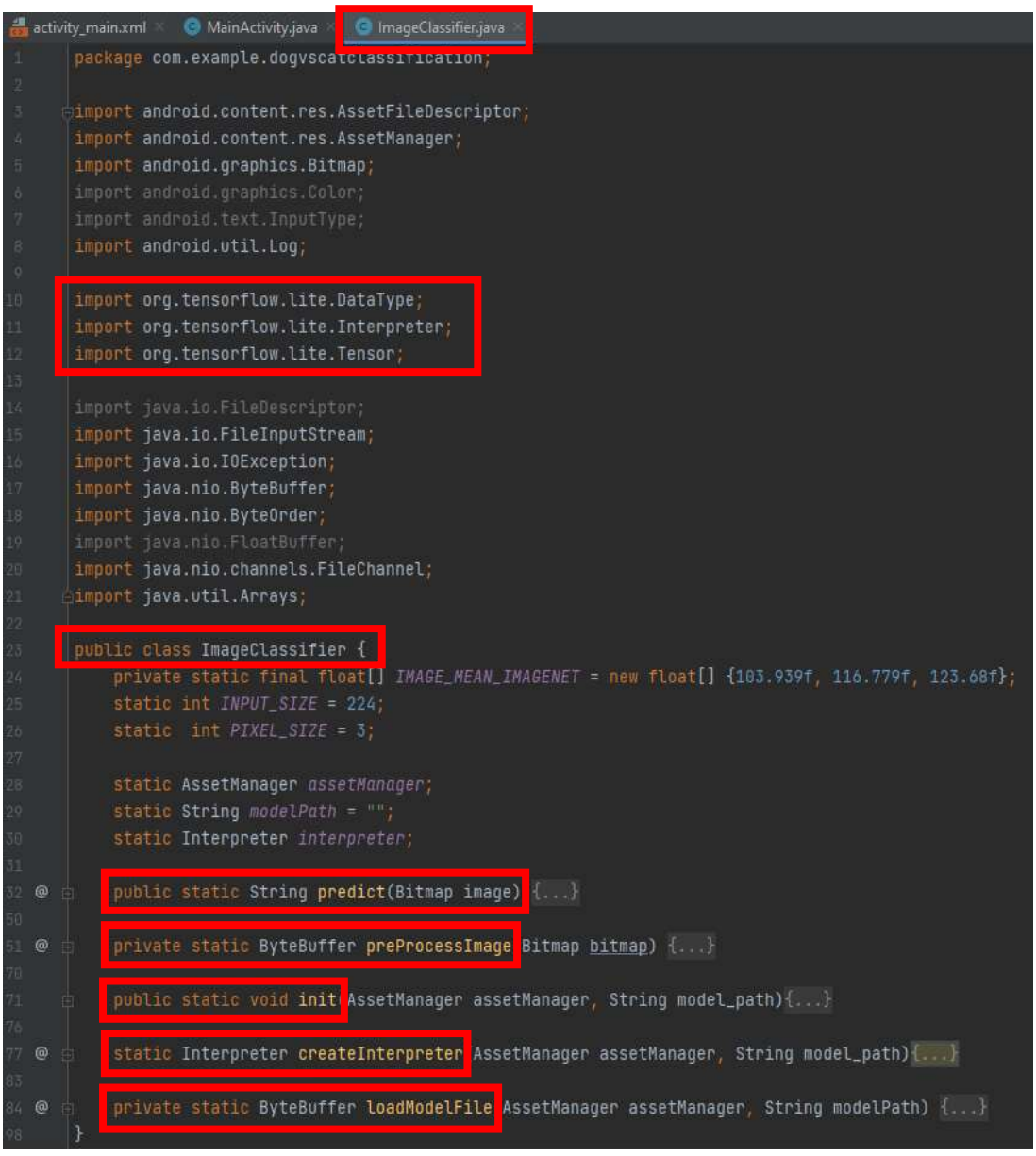
You should now see a new file **(ResultActivity.java)** appears under the same directory.



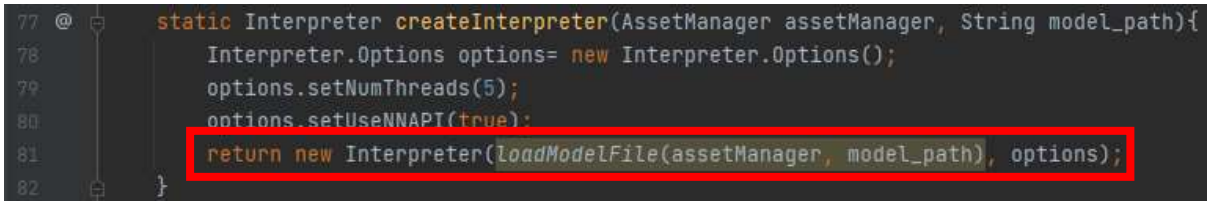
Create another one and name it as **ImageClassifier**. You should have three (3) java files as shown below.



Step 7: Double click on **ImageClassifier** file to open a new tab in the main panel. In this file, we will create the ImageClassifier class in which it loads the TFLite model, creates the interpreter for our model, preprocess our input image and predicts the classification of input image. You can copy the full code from this link. And make sure to import the TFLite library.



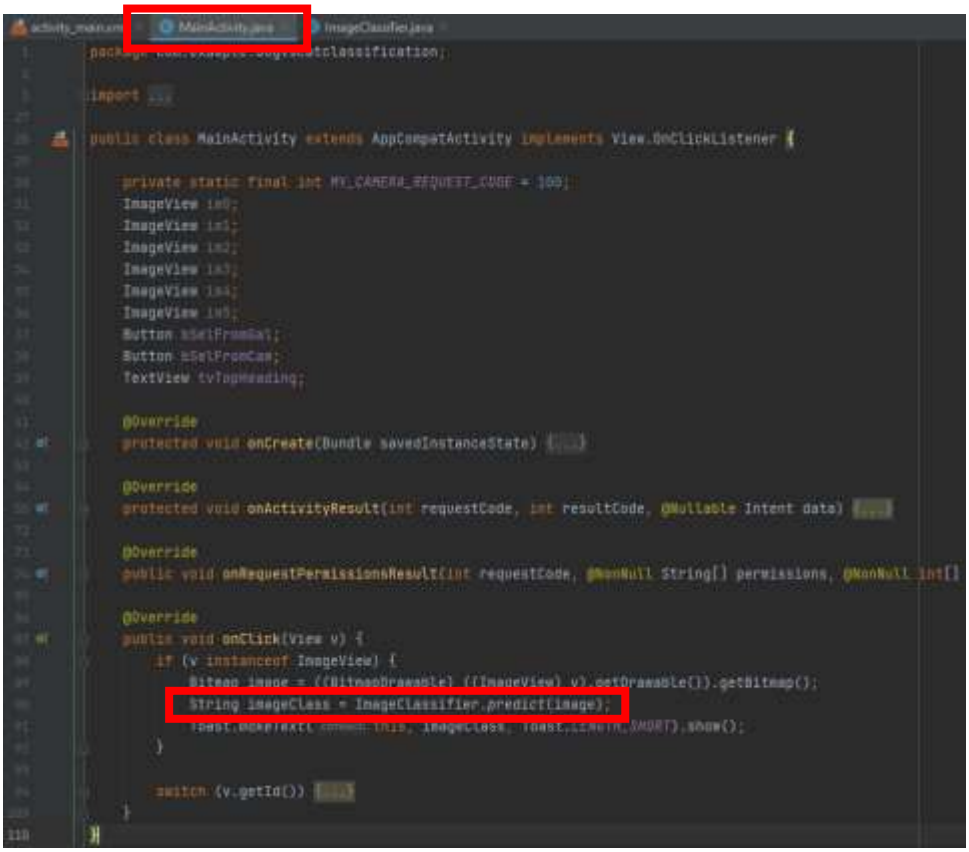
In line 81 under the createInterpreter function, an interpreter of the model is created and returned.



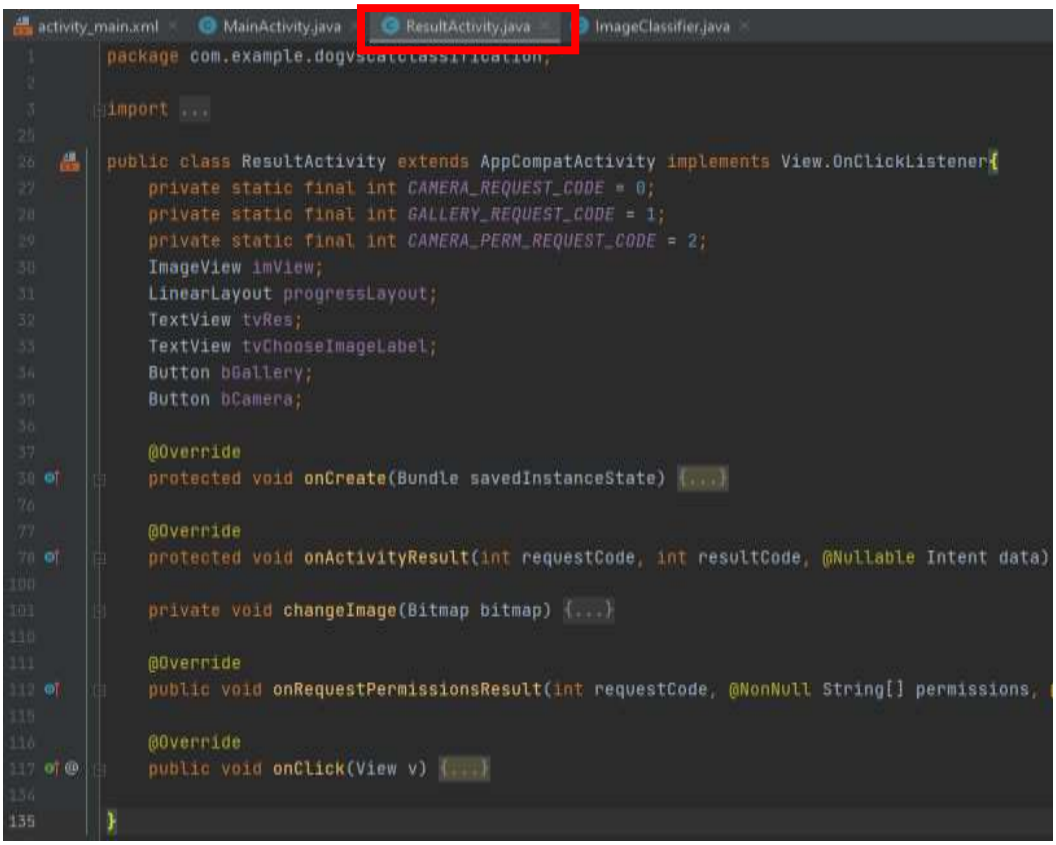
In line 39 under the predict function, the inference is implemented by calling the run method of the interpreter instance.

```
32 @    public static String predict(Bitmap image) {
33          ByteBuffer inpBuffer = preProcessImage(image);
34          Tensor outTensor = interpreter.getOutputTensor( outputIndex: 0);
35          int[] outShape = outTensor.shape();
36          DataType outType = outTensor.dataType();
37          Log.d( tag: "datatype is",  msg: "predict: "+ outType);
38          float[][] out = new float[outShape[0]][outShape[1]];
39          interpreter.run(inpBuffer, out);
40          Log.d( tag: "output is ",  msg: "predict: " + Arrays.toString(out[0]));
41          float[][] out_float = new float[1][1];
42          out_float[0][0] = 1.0f;
43          if (Arrays.equals(out[0], out_float[0])) {
44              return "Dog";
45          }
46          else {
47              return "Cat";
48          }
49      }
```

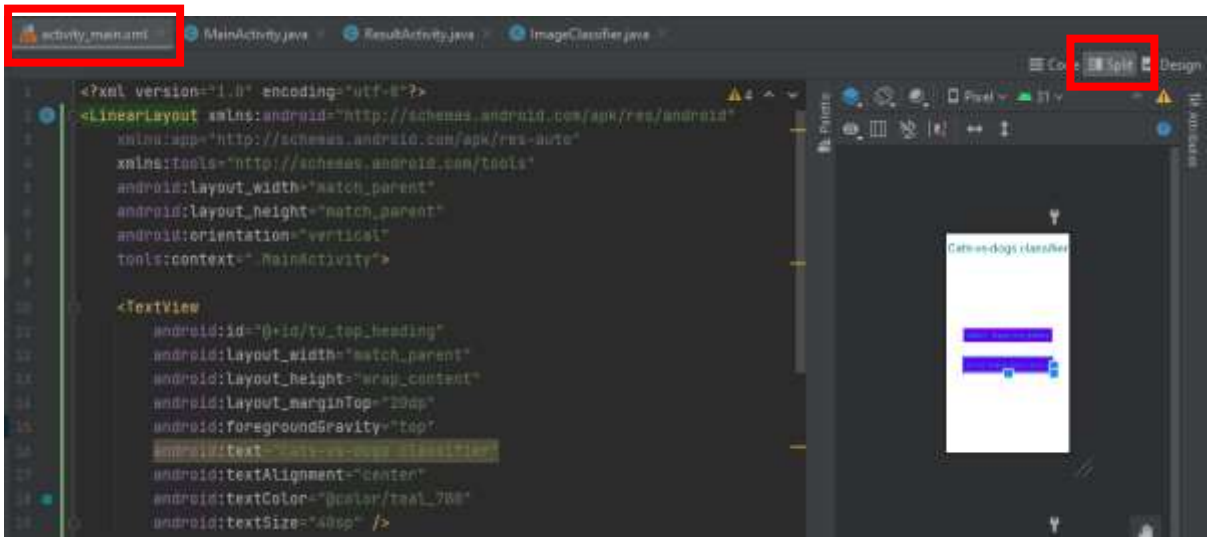
Step 7: Switch tab **to MainActivity.java** and paste the code from this link. In this file, it defines the functions of loading the image to infer either from the phone's gallery or directly from camera. And calls the ImageClassifier to predict the input image.



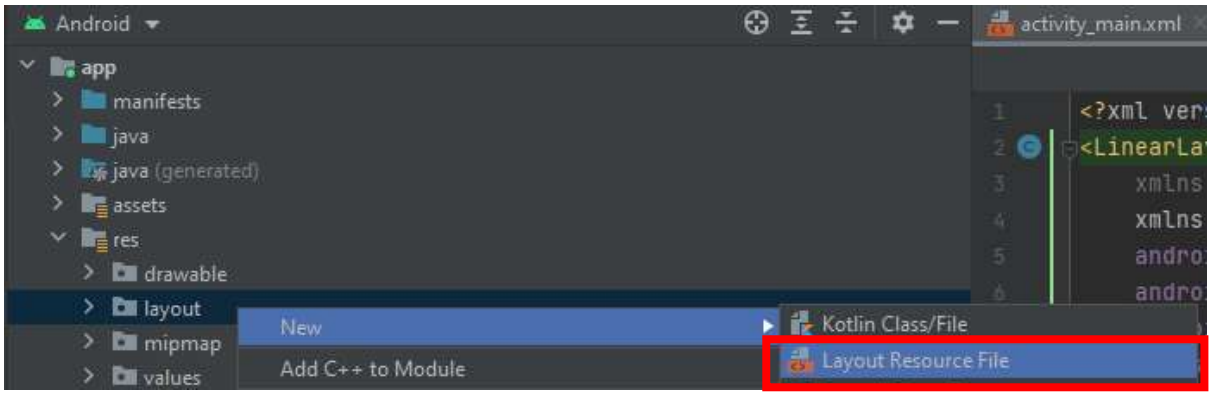
Step 8: Double click on **ResultActivity** file to open a new tab in main panel and paste the code from this link. In this file, it defines the function of displaying the result.



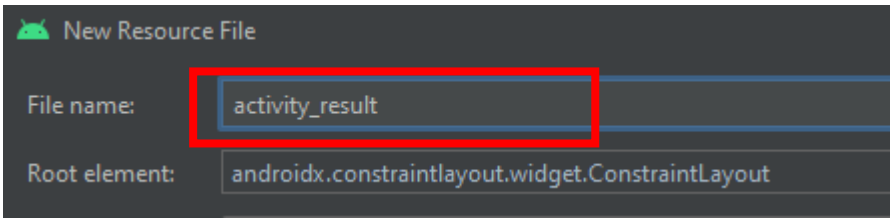
Step 9: Switch tab to **activity_main.xml** and paste the code from this link. On top-right corner of the main panel, click on Split to view the layout of the design at the same time.



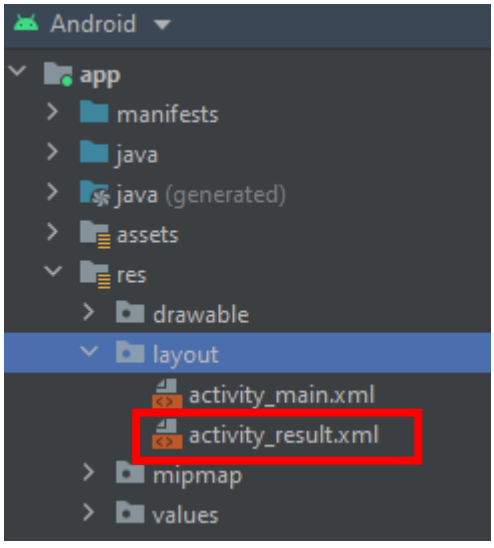
Step 10: In the project navigation go to **app → res → layout** and right-click on layout directory to create new layout resource file. Click on **New → Layout Resource File**.



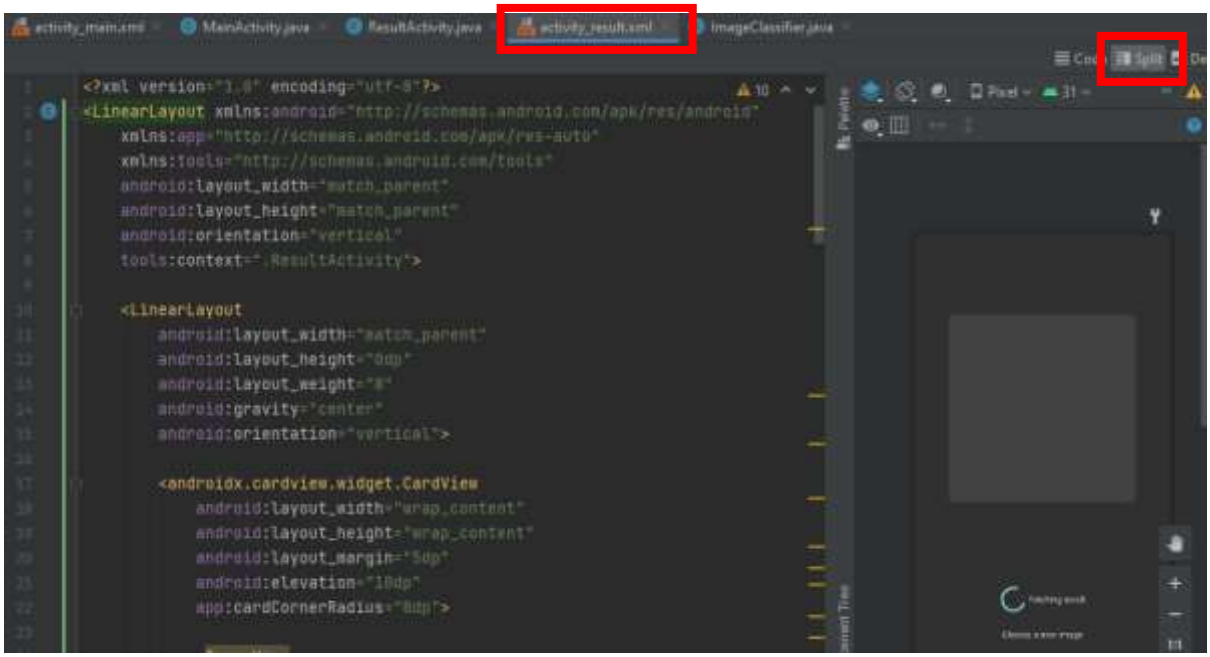
Type in **activity_result** as the name of new layout resource.



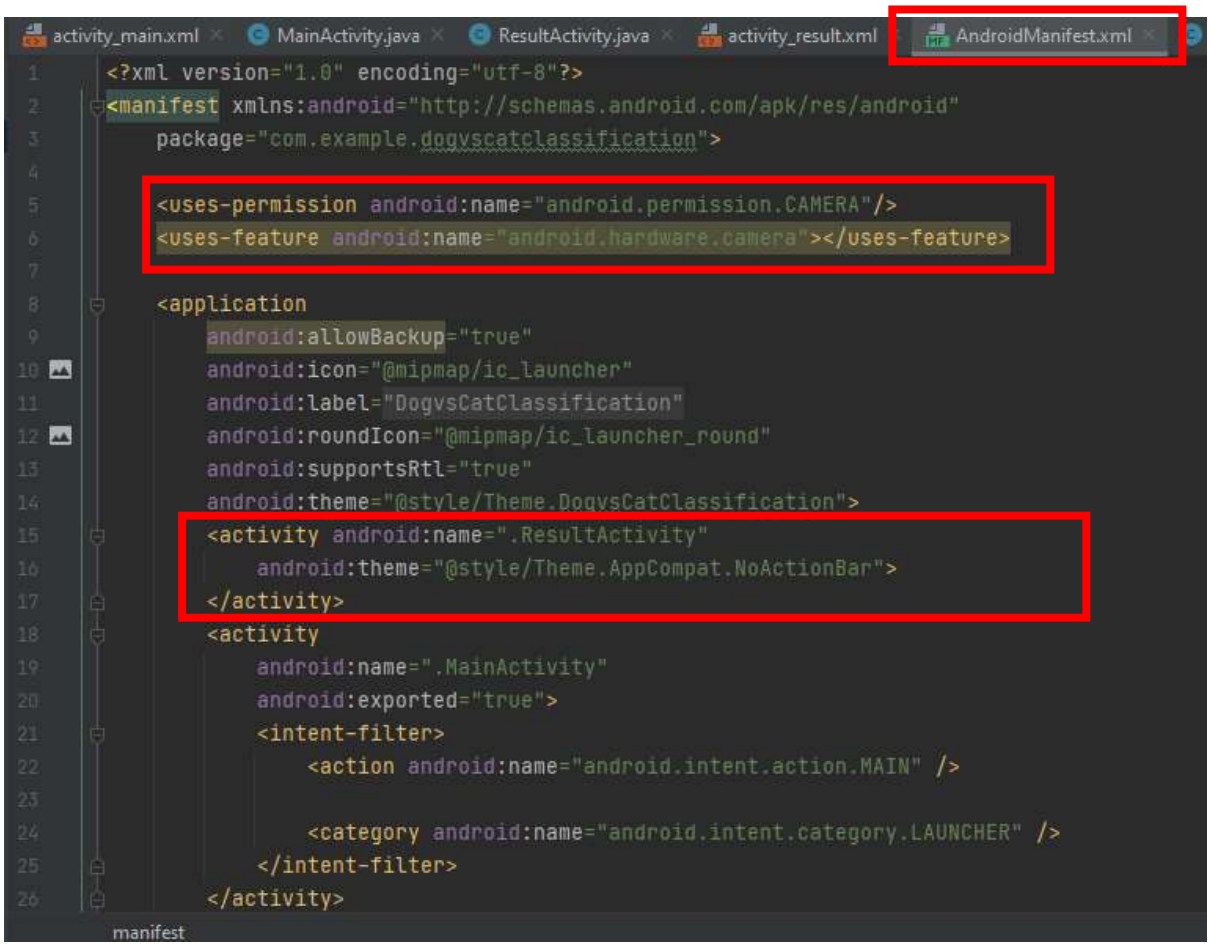
You should now see a new file **(activity_result.xml)** appear under the layout directory.



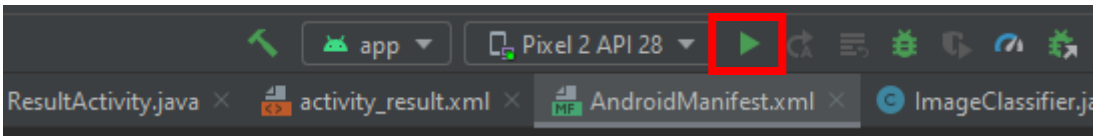
Step 11: Double click on **activity_result** file to open a new tab in the main panel then paste the code from this link. This is the layout design when displaying the result.



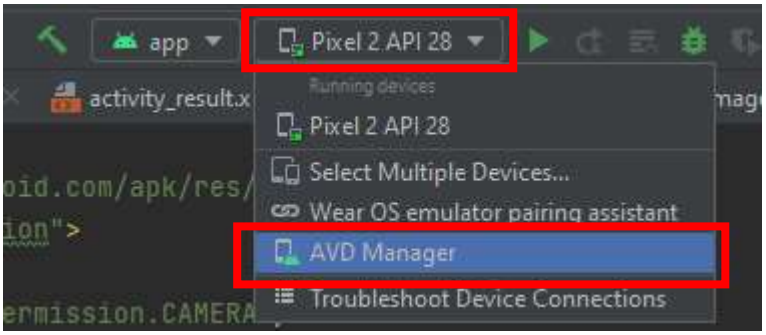
Step 12: In the project navigation go to **app → manifests** and double-click on **AndroidManifest.xml** to open in the main panel and insert the code below to access the phone's camera.



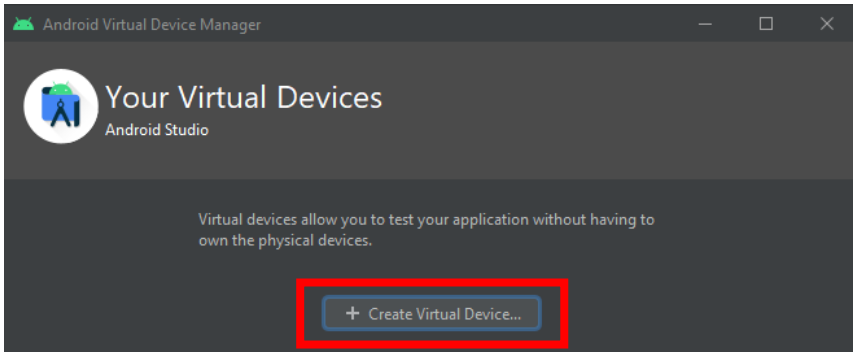
Step 13: Build and run the Android app on a virtual device by clicking on the green run icon at the tip-right corner of the IDE.



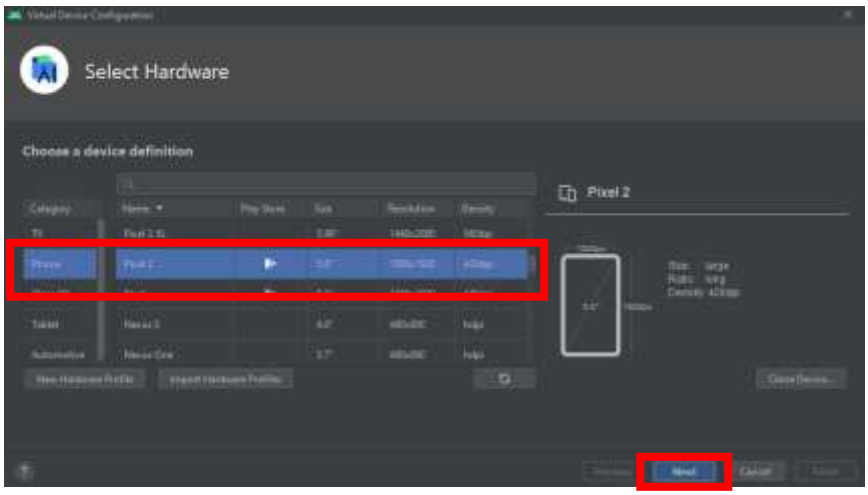
If you have not yet setup your virtual android device, click on the drop-down list beside the green run icon and select **AVD Manager**.



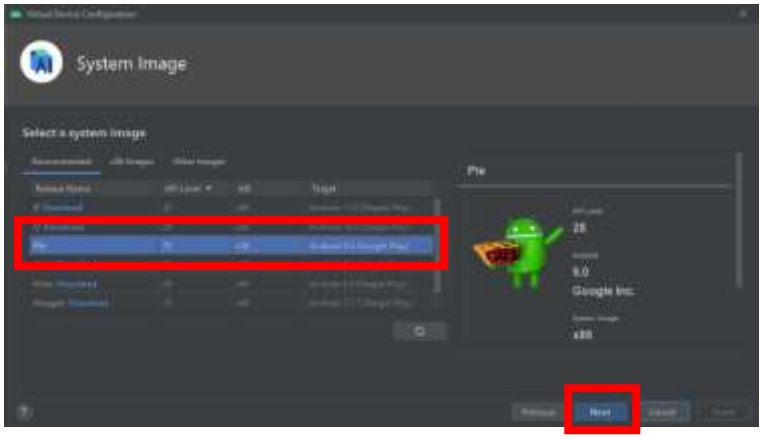
Click on **Create Virtual Device**.



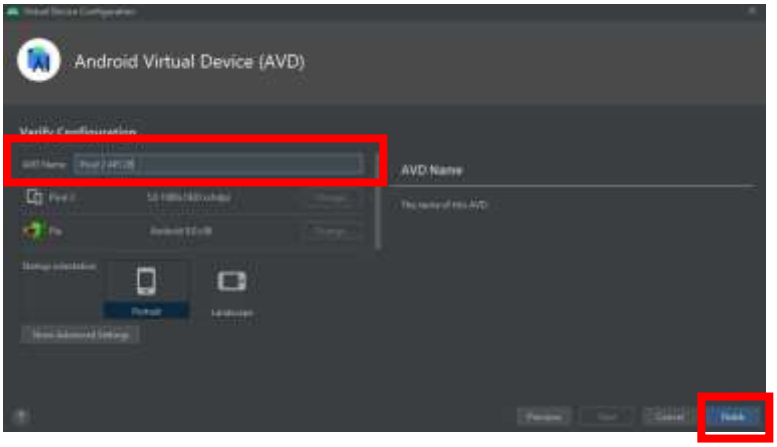
Choose **Phone** as the Category and select **Pixel 2 XL** as the model.



Select **Pie** as the system image.

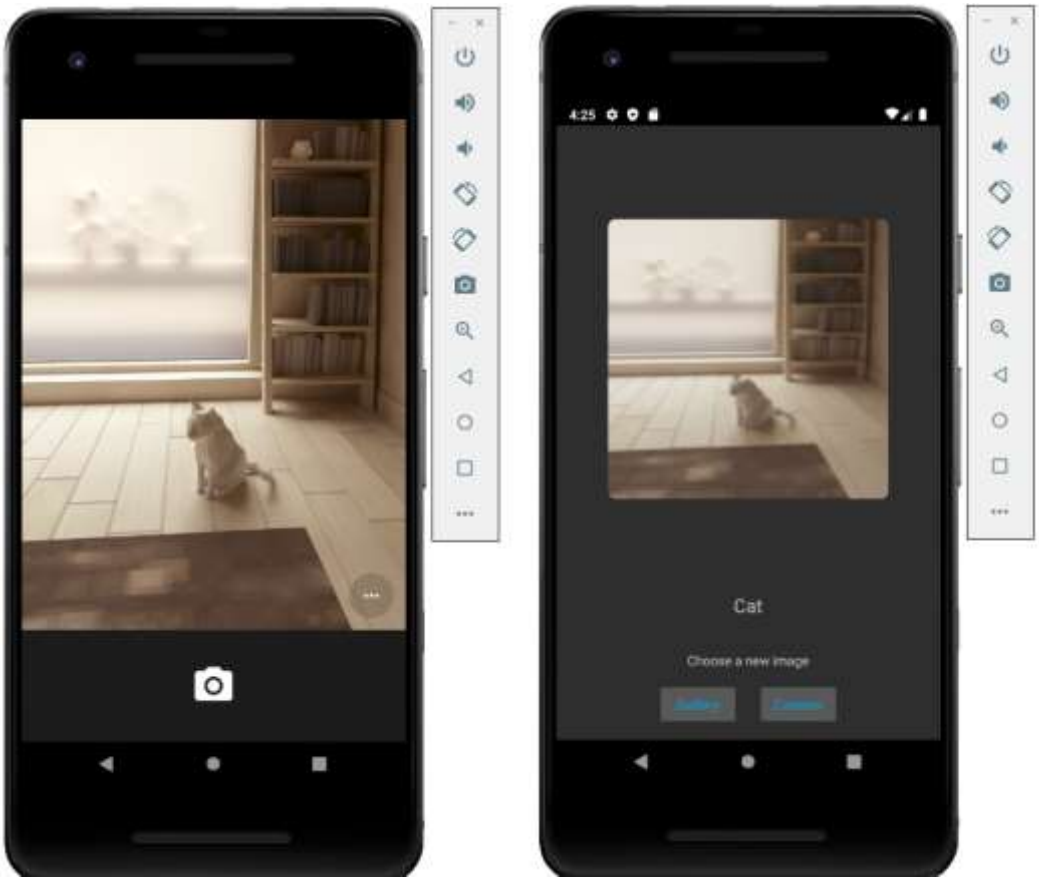


Choose a **name** for your virtual device and click Finish.

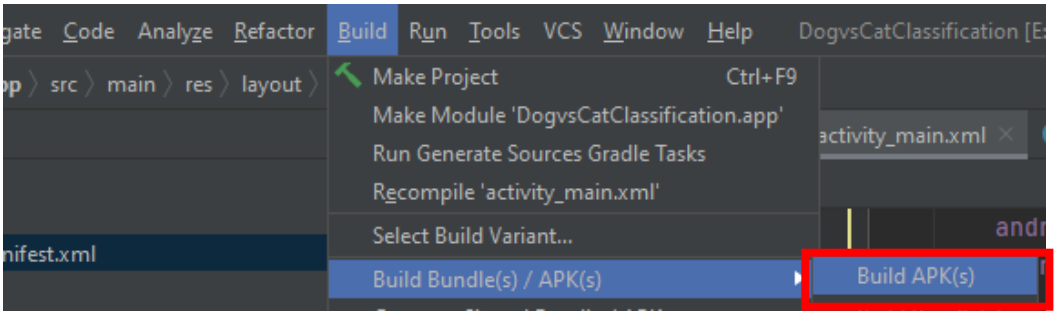


Now, you can then choose your newly created android device from the drop-down list and run your test by clicking on the green run icon. Your virtual Android device should start up as shown below.

Step 14: Test your app by capturing image from the camera and will display the result.



Step 15: Export an installable Android package (.apk) by navigating to **Build → Build Bundle(s) / APK(s) → Build APK(s).**



You can get the installable Android package (.apk) of your app by navigating through your project directory **project name → app → build → outputs → apk → debug**.

Example:
E:\class\7thsem\TAclass\Labs\Lab4\MobileAppClassificationOreo\DogvsCatClassification\app\build\outputs\apk\debug

Copy **app-debug.apk** to your Android device and install the package.

Homework

Train a model to classify dandelion vs. grass using this dataset or you can run the script below to load the data directly to your colab notebook. Deploy the trained model on an Android application for on-device inference. Implement your model by using the pre-trained VGG16 model.

```
# to download the dataset
!gdown --id 1YLaRefVhcCZGuc-3_eqlpOxUQ0Iypq7J
# to unzip the file
!unzip train.zip
```

For this homework, you are required to turn in the following:

a. Python script of training and conversion of the model to TFLite format.

b. Screen captures of the Android application. You can use the photos below to test your Android application or you can capture new images from your phone's camera.

c. Discussion of the implementation procedure.