CUSTOM OBJECT DETECTION USING OPENCV, KERAS, AND DEEP LEARNING(SQUEEZENET)

Methodology –

1) Image Dataset collection
2) Selecting/Designing a Neural Net
3) Train the Model
4) Test the Model
5) Deploy Model
6) Convert Keras Model to TFLite using Google Colab
7) Deploy TFLite in Android Studio

Pre-requisites are provided in the requirements.txt file –

Major requirements are - Python3 virtualenv, keras, keras-squeezenet 0.4, TF 2.0 and 1.13, numpy, and OpenCV.

Initial steps –

1. Create Python Virtual Environment so that there are minimal errors and discrepancies.
2. Check for prerequisites, and install any missing libraries.
3. Then, run gatherimage.py in command line virtual env, along with "name of object shown" and number of photos to be taken, e.g.,

python gatherimage.py 200notefrontside 100

This way, '200notefrontside' will be the label of folder, and 100 photos will be saved in it.

It starts capturing images when you press “a” on keyboard, and will close with “q” on keyboard. Of course wait till all images are captured before pressing q. Make sure you show multiple angles of the object, and if it’s a 3D object, then take more pictures, above 200.

4. After gathering images of various objects in a wide range of angles, we can proceed to train the model with the image datasets.
5. In training file trainn.py , we defined classes to the neural net, by mapping the labels of the folders and assigning integers to them.
    a. Sequential model is used in Keras inside TF, because sequential model creates neural net layer by layer.
    b. Seq model does not share any values outside of itself, through its inception to the end of running the code.
    c. Seq model cannot create multiple output destinations or even taken multiple input paths, therefore for certain types of deep learning, sequential model is good since its got only one job to do.
    d. Convolution 2D net makes neurons based on number of classes provided.
    e. Then rELU and softmax reduces data outflow by averaging each vector and then providing a probability to each of the average.
6. Now that the neural net is created, using keras in TF, we can process the images to make them ready for the neural net. OpenCV is used for this, and the images are resized to reduce data size and save time, then it is saved in a dataset with image vectors and their corresponding labels.
7. This dataset is then split into vectors and labels, using the mapper defined earlier. The mapper takes the image file name and assigns it an integer.
8. This new list dataset is fed into the neural net, and is trained using CPU by Tensorflow. GPU is not used since we are using Tensorflow CPU version.

9. The learning rate (lr) is optimized since Adam optimizer is being used. It provides weights to each neuron as it completes one iteration. Lower the learning rate, the longer the initial training takes.
10. The model is the saved. In this case as ".h5" model of Keras.
11. Now the model can be tested. We can take random images from different class folders and input them in the cmd line along with test.py file, e.g., python test.py "image path + .jpg"
12. In "testt.py" file, the mapper, which maps image labels to integers, is reversed. It now maps, integers to labels, hence when the model predicts a series of integers as an array, it takes the index of highest integer in the array, and maps it to the label defined in the mapper, hence giving a human readable prediction output.
13. Once the testing confirms the model is reasonably accurate, we can proceed to create a web camera-based app.
14. In "whatnote.py" file, we use both a reversed map, and then the output of the predictor to be sent to a simple set of if statements to show more human readable outputs.
15. The model takes web camera stream with a defined box as input. In real time, the stream is analyzed frame by frame and an output is shown by predictor.

After the above steps, we have a working object predictor on PC. To transfer this to a mobile device, it is required to convert the '.h5' keras model to a '.tflite'. This can be done using Google Colab servers, using TF 2.0 without GPU. The code is provided in "googlecolab.ipynb" which can be run in Google Colab with a connection to your Google Drive, which can provide both input h5 model and take the output tflite model.

Sources used –

1. https://github.com/tensorflow/examples
2. https://medium.com/@saidakbarp/real-time-face-recognition-tflite-3fb818ac039a
3. https://androidkt.com/how-to-convert-pb-and-h5-into-tflite-file-python/
4. https://www.skcript.com/svr/realtime-object-and-face-detection-in-android-using-tensorflow-object-detection-api/
5. https://www.tensorflow.org/lite/convert/python_api