Deep Lite: Bringing Tensor flow model to Edge Device for offline-Inferencing

Rohith Kumar Nagulapati[1], Nageswara Rao Nandigam[2]
School of Computing and Engineering,
University of Missouri - Kansas City
Kansas City, USA

Abstract— The gap between existing heavy weight neural network models to work on mobile platform made us to bring this paper. In this paper, we proposed a Tensor flow architecture that brings pre- trained model to user's device for custom training and inferencing taking advantage of tflite version of the existing model. The 1001 categories dataset is used to build the mobile-net model on the server and then brought to user's device using established connection through socket. An Android Application is developed to load the model to user's device for situation learning

I. INTRODUCTION

Tensor flow was designed to be a good deep learning solution for mobile platforms. Currently there are two solutions for deploying deep learning applications on mobile and embedded devices they are Tensor flow for Mobile and Tensor Flow Lite. Tensor flow Lite is an evolution of Tensor flow Mobile enabling on-device machine learning inference more importantly with low latency and a small binary size.

Tensor flow Lite supports a set of core operators, both quantized and float, which have been tuned for mobile platforms. They incorporate pre-fused activations and biases to further enhance performance and quantized accuracy supporting custom operations in models. It defines a new model file format, based on Flat Buffers that does not need parsing/unpacking step for secondary representation. Also, the code footprint of Flat Buffers is smaller than protocol buffers.

Tensor flow Lite has a new mobile-optimized interpreter, which has the key goals of keeping apps lean and fast. The interpreter uses a static graph ordering and a custom (less-dynamic) memory allocator to ensure minimal load, initialization, and execution latency. Java API: A convenience wrapper around the C++ API on Android.

II. RELATED WORK

Single-machine frameworks:

Many machine learning researchers carry out their work on a single—often GPU equipped—computer [1, 2], and several single-machine frameworks support this scenario. Caffe [3] is a high-performance framework for training declaratively specified neural networks on multicore CPUs and GPUs. It is easy to compose models from existing layers, but relatively difficult to add new layers or optimizers. Theano [4]] allows programmers to express a model as a dataflow graph of primitive operators, and generates efficient compiled code for training that model. Its programming model is closest to Tensor flow, and it provides much of the same flexibility in a single machine. Unlike Caffe, Theano, and Tensor flow, Torch [17] offers a powerful imperative programming model for scientific computation and machine learning. It allows fine-grained control over the execution order and memory utilization, which enables power users to optimize the performance of their programs. While this flexibility is useful for research, Torch lacks the advantages of a dataflow graph as a portable representation across small- scale experimentation, production training, and deployment. From the aspect of on situation deep learning, our project can come in handy.

III. PROPOSED WORK

We focused on training and testing 1000 categories PASCAL dataset. The deep learning model is built on a server machine. All the built models can be downloaded to Android APP. The download model button is for downloading the model from the server. This downloaded model gets converted to lite version of Tensor flow to make less resource utilization of end devices. This lite version may be used for inferencing or context learning and continuous learning on end devices offline.

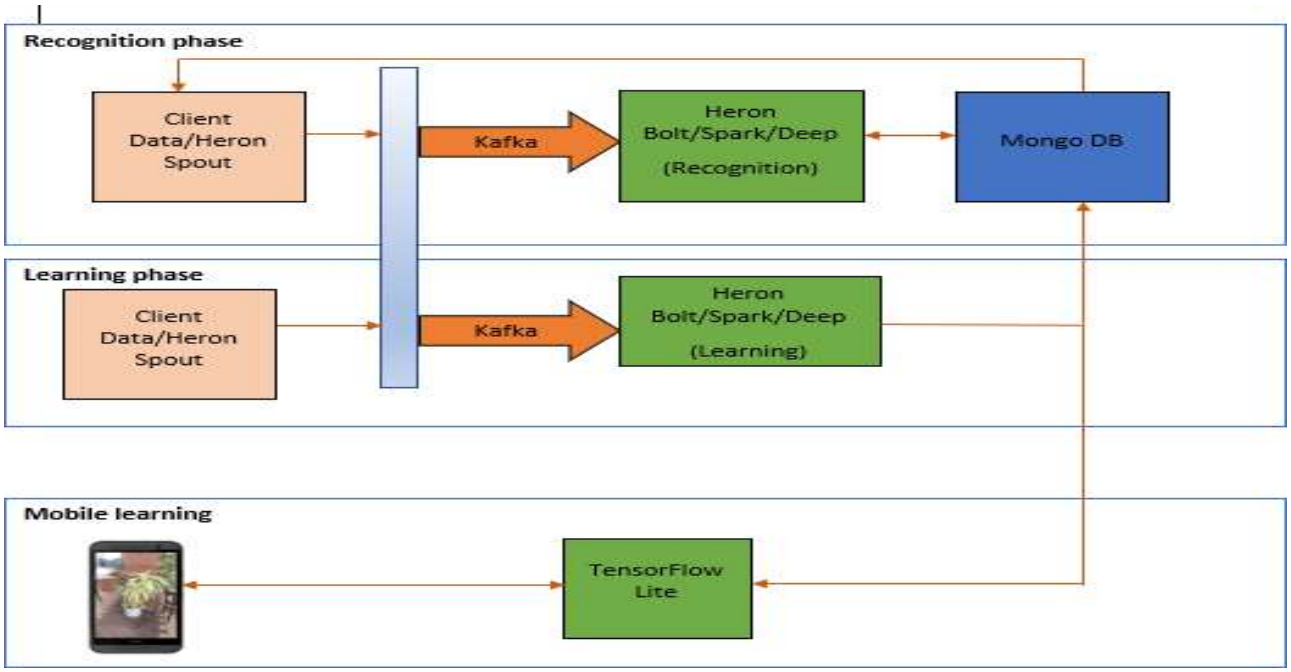


Figure 1: Architecture Diagram

Tensor flow model

We are using keras which is an open source neural network library written in Python. It is capable of running on top of Tensor flow. This functional API allow us to build model comfortably.

Tensor flow lite model:

Tensor Flow Lite is a lightweight solution for mobile and embedded devices. It enables on-device machine learning inference with low latency and a small binary size. A set of core operators, both quantized and float, many of which have been tuned for mobile platforms. These can be used to create and run custom models. Developers can also write their own custom operators and use them in models.

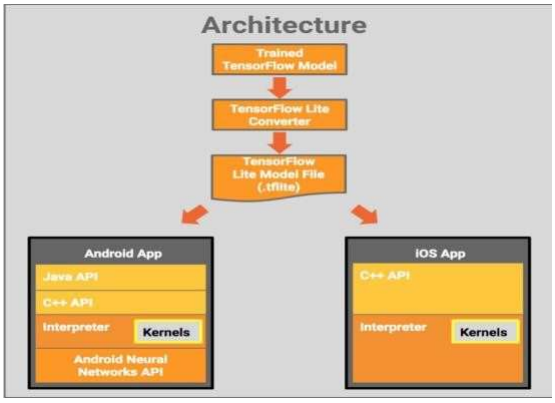


Figure 2: TFlite Architecture used for inferencing

libraries to model neural networks. So, we used convolutional neural network to find patterns in the image and finally to classify the objects in image. This part of logic runs in server and saving onto cloud repository. Below is the snippet for build model in server

IV. IMPLEMENTATION AND EVALUATION

A. System Design and Implementation

The overall workflow is in Figure 7 above. Pascal visual object detection dataset is used to train and test for three selected modules-Clarifai and Tensor flow. A pre-trained model mobile net is used and its last layer is for further trained with the new inputs. The user will see the prediction of the model on the device then.

B. Evaluation and Results

1) Evaluation plan:

- Datasets:

  The dataset consists of 150,000 photographs, collected from Flickr and other search engines labeled with 1000 object categories. A random subset of 50,000 of the images with labels is used for building model and the remaining images are used for evaluation.

- Model:
  MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how

other popular large-scale models, such as Inception, are used. MobileNets can be run efficiently on mobile devices with Tensor Flow Mobile.

Model	Million MACs	Million Parameters	Top-1 Accuracy	Top-5 Accuracy
MobileNet_v1_1.0_224	569	4.24	70.9	89.9
MobileNet_v1_1.0_192	418	4.24	70.0	89.2
MobileNet_v1_1.0_160	291	4.24	68.0	87.7
MobileNet_v1_1.0_128	186	4.24	65.2	85.8
MobileNet_v1_0.75_224	317	2.59	68.4	88.2
MobileNet_v1_0.75_192	233	2.59	67.2	87.3

Figure 3: MobileNet model Accuracies

- System Specifications:
  We used i7-6700HQ processor, 16GB RAM CPU to build model on server machine and we inferenced the model on Nexus 7

Experimental Setup:

Processor: Qualcomm snapdragon 625 CPU Memory: 2GB memory, Operating System: android 7.1.1 Dataset: Pascal Dataset Models: Pre-trained models150,000 photographs (trained +validation), 1000 categories

Model	Model	Type	Size	Architecture	Top 5 Accuracy
quant.tflite	Mobile net	CNN	4MB	28 layers	89.5%
float.tflite	Mobile net	CNN	16MB	28 layers	89.9%
slim.tflite	Inception	CNN	93MB	48 layers	93.9%

Figure 4: Model comparisons

Evaluation Results:

Performance:

Execution time to do inference for mobilenet_quant taking close to 200 milliseconds where as mobilenet_float tflite models taking 350 msec and inception model running almost 2 seconds because of it has 48 layers architecture

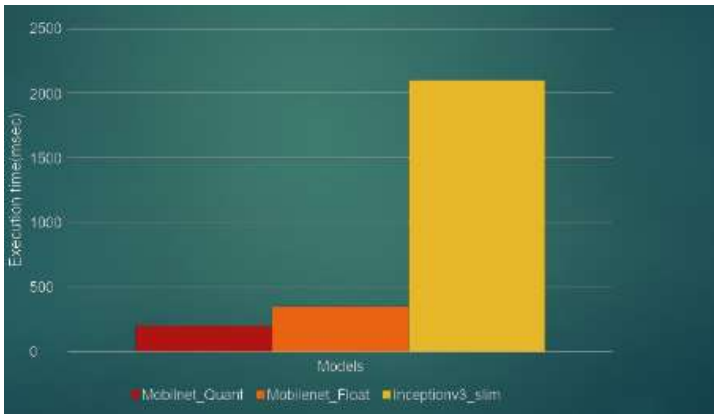
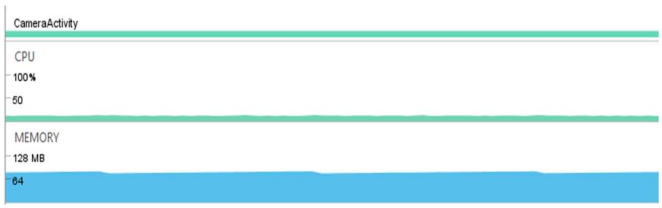


Figure 5: Execution time comparison

CPU utilization:

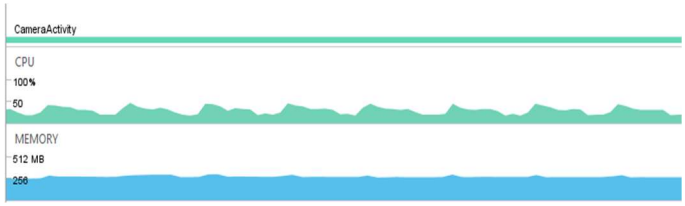
We continuously ran inference over period time to understand the CPU utilization. On average Mobilenet_quant model utilize 15% of CPU ,Mobilenet_float utilize 25% of CPU and inceptionV3 model need 40% of CPU resources.

Mobilenet_Quant model:



Mobilenet_Float Model:

InceptionV3 model:



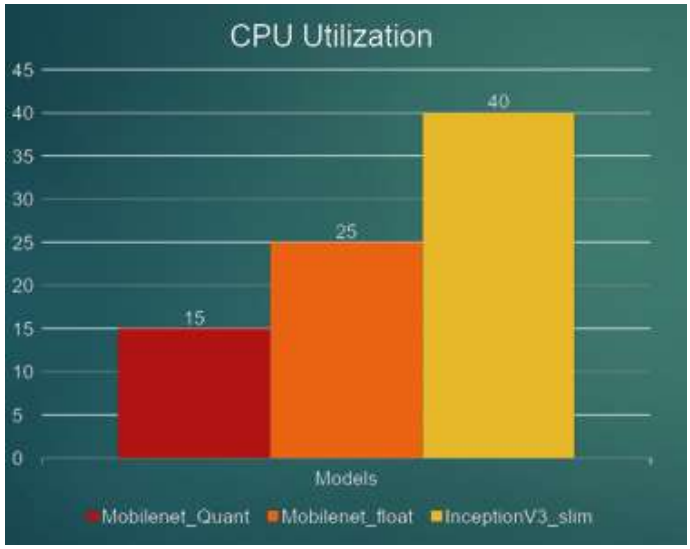


Figure 6: CPU utilization comparison

Memory utilization:

Even for memory the utilization is directly propotinal to size of model and no of layers it has. So mobilenet_quant using 64MB of RAM for inferncing, 130MB will needed for mobilnet_float and inception v3 utilizing 256MB of RAM to infer.

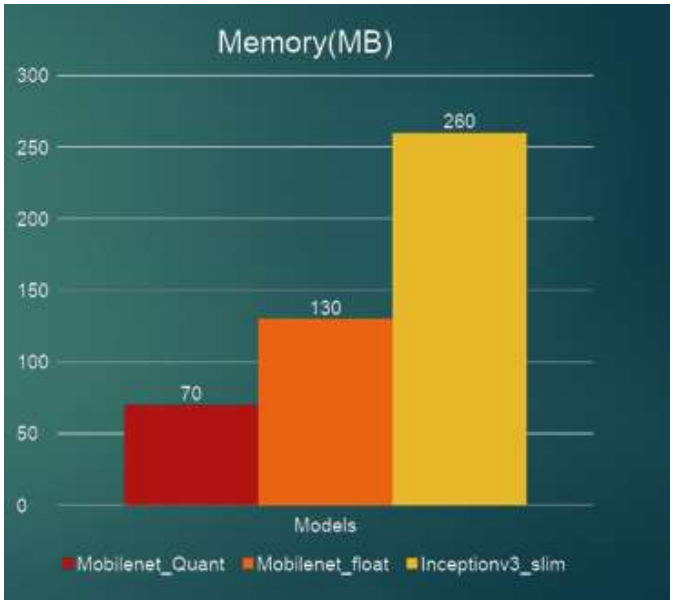


Figure 7: Memory Utilization comparison

Future work:

We would like to take leverage of collaborative recognition in multiple devices where each mobile device responsible for one type of model inference and combine all results into one device. For example, if you would like to infer the image of water bottle, then one device will do image classification, one device will tell you text in that image, one device will give you color of the bottle and etc.

Figure 8: Collaborative recognition

REFERENCES

1.Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. and Ghemawat, S., 2016. Tensorflow: Large- scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

3. Ran, Xukan, Haoliang Chen, Zhenming Liu, and Jiasi Chen. "Delivering Deep Learning to Mobile Devices via Offloading." In Proceedings of the Workshop on Virtual Reality and Augmented Reality Network, pp. 42-47. ACM, 2017.

4 Lane, N.D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L. and Kawsar, F., 2016, April. Deepx: A software accelerator for low-power deep learning inference on mobile device

5 Lane, Nicholas D., Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices." In Proceedings of the 2015 International Workshop on Internet of Things towards Applications, pp. 7-12. ACM, 2015.

6 Lane, Nicholas D., Petko Georgiev, and Lorena Qendro. "DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learning." In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 283-294. ACM, 2015.