MIDS 251 Spring 2020
HW5
Suhas Gupta

1. TensorFlow is an open source software framework that contains math libraries useful for performing machine learning and deep learning programming tasks. It provides a set of APIs and programming framework that can be used for development of machine learning training and inference applications.
    a. Google is the developer and leading contributor to TensorFlow though other companies like Nvidia also contribute to the hardware accelerated libraries of TensorFlow for optimized run times on their proprietary hardware.

2. TensorRT is an SDK for optimizing the performance of deep learning inference when run on Nvidia (or other) CUDA enabled GPUs. It is built on Nvidia's CUDA programming model and provides optimizations for deep learning inference. TensorFlow is the general framework for both training and inference and doesn't natively contain optimizations for accelerated inference on GPUs. TensorRT integration with TensorFlow provides this additional optimization for high performance deep learning inference.

3. ImageNet is a database of quality-controlled and human-annotated images organized according to the WordNet hierarchy. The aim of ImageNet contributors is to provide approximately 1000 images per "synonym set" present in WordNet.
    a. Total number of images = **14,197,122**
    b. Number of classes > **20,000**

4. **MobileNet vs GoogleNet**
    a. MobileNet is a set of CNNs that were designed to be computationally efficient on embedded and mobile devices (less resource hungry). Compared to a traditional CNN, MobileNet architecture uses depth-wise separable convolutions and a 1x1 convolution called pointwise convolution. Figure 1 [1] illustrates the difference between standard CNN & MobileNet CNN architectures.

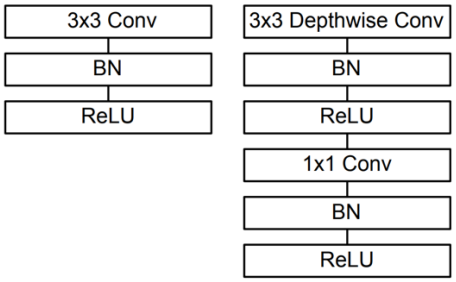
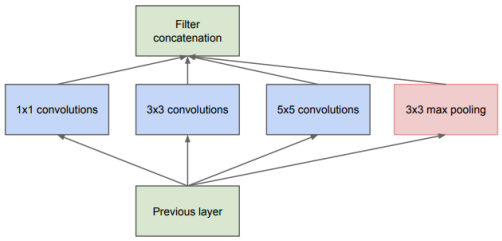


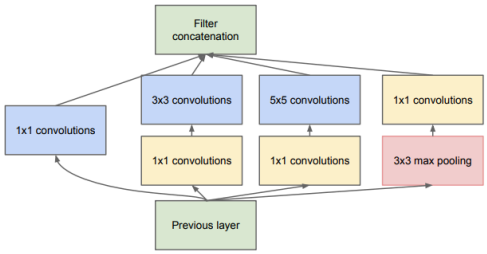
Figure 1[1]: Left: Standard CNN with batch normalization and ReLU. Right: Depth wise Separable convolutions with Depth wise and Pointwise layers followed by batch normalization and ReLU.

    b. GoogLeNet architecture is also motivated with a need to increase computational efficiency on mobile/embedded devices. However, instead of simplifying the network architecture like MobileNet, GoogLeNet's architecture aims to use the computational

resources inside the devices more efficiently. The architecture is shown in Figure 2[2]. The concept is also termed "inception" because the networks are intended to be very deep resulting in a significant reduction in number of parameters of the model.



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2[2]: Inception architecture with (a) and without (b) dimensionality reduction

We can see that both MobileNet & GoogLeNet intend to improve computational efficiency of image classification algorithms to run on mobile and resource constrained embedded devices. However, the main difference between the two is the tradeoff of network complexity and accuracy. MobileNet is a small/less complex network with a lower accuracy compared to the inception architecture that is intended to be very deep and have higher accuracy.

5. Bottleneck is the term used for the network layer just before the final output layer that does the classification. This is a layer which is compressed and uses the computed outputs of the pre-trained (frozen) layers before itself to run efficient evaluation on multiple sets of data. The outputs of the frozen layers are computed only once, and they form the "bottleneck" features of the network that can be used in the last layer of the network to re-compute on each dataset. Thus, bottleneck layer contains the cached results from the frozen layers of a pre-trained network.
6. Bottleneck feature and layer freezing concepts are the same. In both of these techniques, the pre-trained model weights are computed and cached to be used by the final classification layer.
7. The lab uses the pre-trained network layers as a part of the frozen model downloaded from the MobileNet (or GoogLeNet) repository. When the re-training is run in the lab, the script downloads the pre-trained model from the *'tensorflow/models'* GitHub repo and adds are final classification layer and trains the final layer on the images that are downloaded in the lab.
    a. The scripts compute the bottleneck checkpoints when the script is run for the first time. For each successive run, the bottleneck outputs are uses as input to the newly added final classification layer and re-training is done on the user dataset.

8. The reduced learning rate increased the training time by about 6 minutes on the Jetson TX2 when using the GPUs. The accuracy increased by about 3% (from 88% to 91%)

Learning Rate	Training Time (secs)	Validation Accuracy (%)
0.1 (Default)	648.172	88

0.005	983.769	91

9. Training with learning rate of 1.0 and the accuracy was comparable to other values:

Learning Rate	Training Time (secs)	Validation Accuracy (%)
0.1 (Default)	648.172	88
0.005	983.769	91
1.0	688.679	90

10. I used animal pictures download from Apple's coreML developer website:
(https://developer.apple.com/documentation/vision/classifying_images_with_vision_and_core_ml)
The model accuracy was 90% - 92% on test set inference using this dataset indicating that the model generalized extremely well using the bottleneck approach. I was able to train it using 2579 images which is not a very high number.
11. Training times comparison between GPU vs CPU runs:

Compute Units	Training Time (secs)	Validation Accuracy (%)
GPU	688.679	88
CPU	871.904	88

The training on Jetson TX2 CPUs take longer. This is expected since the TensorRT flow, that we are using in this lab, is optimized by Nvidia for CUDA based GPU cores. Besides, computations in CNNs can be readily parallelized thereby taking advantage of the large number of small GPU cores that are present in the Jetson TX2. The ARM based CPUs in the dev kit on the other hand are much smaller in number and do not have x86 features like hyperthreading.

12. Training using the 'inception_v3' network:

Compute Units	Learning Rate	Training Time (secs)	Validation Accuracy (%)
GPU	1.0	1277.172	88.01
CPU		992.036	90.67

The GPU training times are longer than CPU training times. This is an interesting observation and might be due to the nature of the inception architecture. Since the architecture is very deep and uses 1x1 convolution operations only, parallelization due a GPU doesn't provide benefit over the faster CPU cores that can execute the network faster.

13. To evaluate the trained inception model on the flower images, we need to execute the label script with the following arguments(highlighted in **red**):

```
python -m scripts.label_image --input_layer="input" --input_height=299 --
input_width=299  --graph=tf_files/retrained_graph.pb --
image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```