HW4 – W251 - abhisha@berkeley.edu

- **What was the average inference time for model and image combination? What where the returned classes their score?**

For example, in the case of efficientnet-L_quant.tflite, we see the following results.

(tflite) root@nvidia-desktop:/data/w251/v2/week04/hwPart2/tflite# python3 classify_image.py --model models/efficientnet-L_quant.tflite --labels models/imagenet_labels.txt --input /data/w251/w251_hw4/**hen.jpg** -k 5

----INFERENCE TIME----

Note: The first inference may be slow because it includes loading the model.

1633.2ms

1592.3ms

1592.8ms

1626.8ms

1612.6ms

-------RESULTS--------

9 hen: 0.80469

8 cock: 0.02734

87 partridge: 0.00781

332 hare: 0.00000

336 fox squirrel, eastern fox squirrel, Sciurus niger: 0.00000

(tflite) root@nvidia-desktop:/data/w251/v2/week04/hwPart2/tflite# python3 classify_image.py --model models/efficientnet-L_quant.tflite --labels models/imagenet_labels.txt --input /data/w251/w251_hw4/**shark.jpg** -k 5

----INFERENCE TIME----

Note: The first inference may be slow because it includes loading the model.

1591.7ms

1562.0ms

1575.2ms

1567.3ms

1573.0ms

-------RESULTS--------

3 great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias: 0.47266

4 tiger shark, Galeocerdo cuvieri: 0.37109

5 hammerhead, hammerhead shark: 0.01562

644 mask: 0.00781

720 piggy bank, penny bank: 0.00391

(tflite) root@nvidia-desktop:/data/w251/v2/week04/hwPart2/tflite# python3 classify_image.py --model models/efficientnet-L_quant.tflite --labels models/imagenet_labels.txt   --input /data/w251/w251_hw4/**robin.jpg** -k 5

----INFERENCE TIME----

Note: The first inference may be slow because it includes loading the model.

1597.9ms

1575.3ms

1566.7ms

1566.1ms

1568.3ms

-------RESULTS--------

16 robin, American robin, Turdus migratorius: 0.87891

21 water ouzel, dipper: 0.00391

913 worm fence, snake fence, snake-rail fence, Virginia fence: 0.00391

342 hog, pig, grunter, squealer, Sus scrofa: 0.00000

333 Angora, Angora rabbit: 0.00000

We notice the time spent on average for this model was around 1.5 seconds per epoch.

Similarly, the other models are summarized in the table below

Model Name	**Time per Epoch in Seconds**	**Classes incorrectly Classified**
efficientnet-L_quant.tflite	1.5	None
efficientnet-M_quant.tflite	0.6	None
efficientnet-S_quant.tflite	0.4	None
inception_v4_299_quant.tflite	1.85	None
mobilenet_v1_1.0_224_quant.tflite	0.1	None
mobilenet_v2_1.0_224_quant.tflite	0.07	All

- **In your opinion, which model is best and why?**

I think the best model should consider classification accuracy as well as speed, so the best model is mobilenet_v1_1.0_224_quant.tflite

- **Did the EfficientNet models return the correct classifiction? If not why, and how would you fix this?**

No the efficientnet models did not return the right classes. For example

(tf115) root@nvidia-desktop:/data/w251/v2/week04/hwPart2/tf1.15# sudo sh flush_buffers.sh

(tf115) root@nvidia-desktop:/data/w251/v2/week04/hwPart2/tf1.15# python3 classifier.py -m https://tfhub.dev/google/efficientnet/b0/classification/1 -i **images/parrot.jpg**

2020-05-30 19:41:04.947510: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.0

TF 1.15 testing

First inference may be slower

2020-05-30 19:44:28.387181: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1

2020-05-30 19:44:28.437624: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.438093: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1639] Found device 0 with properties:

name: NVIDIA Tegra X2 major: 6 minor: 2 memoryClockRate(GHz): 1.3

pciBusID: 0000:00:00.0

2020-05-30 19:44:28.438213: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.0

2020-05-30 19:44:28.473627: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10.0

2020-05-30 19:44:28.499806: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcufft.so.10.0

2020-05-30 19:44:28.533819: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcurand.so.10.0

2020-05-30 19:44:28.573000: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusolver.so.10.0

2020-05-30 19:44:28.593933: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusparse.so.10.0

2020-05-30 19:44:28.668909: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7

2020-05-30 19:44:28.669145: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] **ARM64 does not support NUMA - returning NUMA node zero**

2020-05-30 19:44:28.669359: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.669440: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1767] Adding visible gpu devices: 0

2020-05-30 19:44:28.709124: W tensorflow/core/platform/profile_utils/cpu_utils.cc:98] Failed to find bogomips in /proc/cpuinfo; cannot determine CPU frequency

2020-05-30 19:44:28.709891: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x313e60b0 initialized for platform Host (this does not guarantee that XLA will be used). Devices:

2020-05-30 19:44:28.709983: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): Host, Default Version

2020-05-30 19:44:28.841094: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.841700: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x32752c40 **initialized for platform CUDA (this does not guarantee that XLA will be used).** Devices:

2020-05-30 19:44:28.841804: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): NVIDIA Tegra X2, Compute Capability 6.2

2020-05-30 19:44:28.842436: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.842650: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1639] Found device 0 with properties:

name: NVIDIA Tegra X2 major: 6 minor: 2 memoryClockRate(GHz): 1.3

pciBusID: 0000:00:00.0

2020-05-30 19:44:28.842751: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.0

2020-05-30 19:44:28.842816: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10.0

2020-05-30 19:44:28.842853: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcufft.so.10.0

2020-05-30 19:44:28.842921: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcurand.so.10.0

2020-05-30 19:44:28.842954: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusolver.so.10.0

2020-05-30 19:44:28.843019: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusparse.so.10.0

2020-05-30 19:44:28.843053: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7

2020-05-30 19:44:28.843200: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.843467: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:28.843578: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1767] Adding visible gpu devices: 0

2020-05-30 19:44:28.845397: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.0

2020-05-30 19:44:32.762000: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1180] Device interconnect StreamExecutor with strength 1 edge matrix:

2020-05-30 19:44:32.762108: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1186] 0

2020-05-30 19:44:32.762156: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1199] 0: N

2020-05-30 19:44:32.764329: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:32.764839: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:950] ARM64 does not support NUMA - returning NUMA node zero

2020-05-30 19:44:32.765171: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1325] **Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 3335 MB memory**) -> physical GPU (device: 0, name: NVIDIA Tegra X2, pci bus id: 0000:00:00.0, compute capability: 6.2)

2020-05-30 19:44:46.410339: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10.0

2020-05-30 19:44:47.770009: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7

36438.4ms

116.9ms

42.0ms

41.5ms

36.0ms

Label: Confidence

African grey: 0.94196

coucal: 0.00440

sulphur-crested cockatoo: 0.00351

black stork: 0.00066

jacamar: 0.00064

In the case of classifying a parrot, the efficient net model got it wrong (although African grey is a type of parrot)

Upon investigating the script classify.py, we find that the code path for efficientnet models uses hub.module() which seems to not work for TX2

I am picking this information from here

https://github.com/MIDS-scaling-up/v2/blob/2a453df03e787655c1e76dc033d63016cfda4918/week04/hwPart2/tf1.15/classifier.py#L47

Hence, we may have to change this function for it to work

Another hypothesis could be that the efficientnet models are trained on only african grey images which is causing the issue. Its also possible that this hun.module() function converts images to grey scale which makes it seem like the parrot is an African Grey.

- **How big (in megabytes) are the models?**

Model Name	**Size in MB**
mobilenet_v1_100_224	2490
mobilenet_v2_130_224	2700 (was trying to allocate 3.06 GB, but failed, even with the flush buffer script)
efficientnet b0	3335
efficientnet b4	3399
efficientnet b7	2575 (tried to allocate 2.9 GB which failed, even after the flush buffer)

- **How did the performance compare to TFLite? Be sure to throw out the first run as it includes downloading.**

In general the mobilenet in TF 1.1.5 models are comparable to the mobilenet models in TF Lite (both being less than 100ms). They also accurately were able to detect the classes.

The efficient net models also were able to perform under 100ms, however, the accuracy of the b0, b4 and b7 models was not on point. Each predicted the parrot to be an “African Grey”.

Similarly for the hen, the classification that was made was “cock”, while this isnt technically wrong, it is not on point.

34778.1ms

116.0ms

59.5ms

54.1ms

56.6ms

Label: Confidence

cock: 0.81578

stingray: 0.06672

quail: 0.00698

centipede: 0.00316

ruffed grouse: 0.00262

- ## In your own words, what is Quantization? What is effect on performance and accuracy?

It is about approximating real-world values with a digital representation that introduces limits on the precision and range of a value. For example, rounding numbers into less precise representations by means of truncation or rounding is an example of quantization. Quantization introduces various sources of error in your algorithm, such as rounding errors, underflow or overflow, computational noise, and limit cycles. This results in numerical differences between the ideal system behavior and the computed numerical behavior.

To manage the effects of quantization, you need to choose the right data types to represent the real-world signals. You need to consider the precision, range, and scaling of the data type used to encode the signal, and also account for the non-linear cumulative effects of quantization on the numerical behavior of your algorithm. This cumulative effect is further exacerbated when you have constructs such as feedback loops.

Quantization can cause an increase in performance due to rounding numbers to lesser precision, but it can lead to errors in accuracy during prediction. However, quantization would lead to more efficient compute performance as we only are calculating less precise numbers.

- ## What models did you use?

GoogleNet, GoogleNet-12, ResNet-18 and VGG-16

- **What was the average inference time for model and image combination? What where the returned classes their score?**

For the parrot (GoogleNet)

image is recognized as 'macaw' (class #88) with 99.951172% confidence

[TRT] --

[TRT] Timing Report networks/bvlc_googlenet.caffemodel

[TRT] --

[TRT] Pre-Process CPU 0.11226ms CUDA 0.20723ms

[TRT] Network CPU 12.16874ms CUDA 12.10288ms

[TRT] Post-Process CPU 0.18471ms CUDA 0.18496ms

[TRT] Total CPU 12.46571ms CUDA 12.49507ms

[TRT] --

For the hen (GoogleNet)

image is recognized as 'hen' (class #8) with 95.166016% confidence

[TRT] --

[TRT] Timing Report networks/bvlc_googlenet.caffemodel

[TRT] --

[TRT] Pre-Process CPU 0.10765ms CUDA 0.16432ms

[TRT] Network CPU 13.56380ms CUDA 13.45898ms

[TRT] Post-Process CPU 0.25341ms CUDA 0.25248ms

[TRT] Total CPU 13.92486ms CUDA 13.87578ms

[TRT] --

- **In your opinion, which framework was best? Why?**

I think the jetson framework is the best because it allows for fast training of pre trained models with high accuracy (compared to TF that got the classes wrong). There is also likely some acceleration in the classification due to the clock speed adjustment that was made on the TX2 + native integrations provided by the jetson inference engine on NVIDIA devices. Lastly, I observed that the jetson inference was caffe based. Caffe is known to be blazingly fast and accurate for object detection. So there's multiple advantages with running jetson inference on TX2. However, if this was not being run on an NVIDIA device, we will need to revisit the answer to this question.

Updates made to the TF 1.15 classifier – remove this line 33 where we hard code "k=1". We want to be able to specify our top k classifications, with this line k=1, we are always bounded by 1 classification.

https://github.com/MIDS-scaling-up/v2/blob/2a453df03e787655c1e76dc033d63016cfda4918/week04/hwPart2/tf1.15/classifier.py#L33