Track & Hardware Platform

We opted for Track 2 (DNN Accelerator and DNN Acceleration Library) utilizing a GPU on Google Colab. We compared the performance of an older model, AlexNet, and its optimized versions through various techniques to a newer model, ViT, on the CIFAR-10 dataset.

For AlexNet, we implemented the original model and various optimization techniques on the model. This includes: quantization techniques, adding residual connections in the mode, and pruning methods that applied pruning to the dense layers in the model.

For quantizations on the AlexNet, we used float16 quantization, which converts the weights to the float16 datatype, and dynamic quantization, which converts the weights to the int8 data type. After researching, we were able to verify that the Google Colab Nvidia T4 GPU supports both float16 and int8 data types. Through all of the optimization techniques used, we noticed that the pruning methods was the most effective in optimizing the model. Not only did it significantly cut down on the model size and slightly reduce the inference time, it also increased the overall inference accuracy of the model.

Problem & Challenges

Our goal was to optimize inference latency for deep learning models on resource-constrained environments, without sacrificing accuracy. We achieved this by comparing two high-accuracy models for CIFAR-10 image classification, chosen from different years on the Papers with Code leaderboard to explore potential advancements in efficiency over time. By analyzing their architectures, latency, and accuracy, we aimed to identify the model offering the best trade-off for CIFAR-10.

A significant challenge we encountered was the limited resources available on Google Colab's free tier GPU. The models we trained often exceeded the free tier's RAM and GPU usage limits, leading to interruptions and potential progress loss. This issue was especially difficult to overcome with the ViT model as it requires a large RAM space for training. To address this, we implemented a two-pronged approach. First, we utilized multiple personal Google Colab accounts to distribute the training workload across multiple sessions. We did this by saving each model after training, and manually moving the model packages through our different accounts to load them in the notebooks so that we could continue to run training and inferences on the GPU. Second, we incorporated model saving checkpoints within the training script. We then decided to break the training dataset into multiple subsets to reduce the load of each training period, and resume training from the latest saved point. This way, even if a session is timed out due to resource limitations, we could mitigate potential progress loss.

AlexNet is a model that is originally used for the data in the ImageNet dataset, which included larger image sizes and more classes. Therefore, we had to modify certain parameters within the model to accommodate for the smaller image sizes and lesser class sizes. Since we started exploring the Google Colab environment with the implementation of AlexNet, the main issues we faced were difficulties with importing packages, creating, and running the model as well as issues with modifying the model to work with the CIFAR-10 dataset. For example, we were confused on how to write inference code that properly utilized the parallelism of the GPU provided by the Google Colab environment. Another example of a challenge we faced was when trying optimization techniques, we had an error with importing a specific package that is commonly used for pruning a model. After discussing with our TA Junhao, we realized

that the issue lies within the Colab Notebook we were working on, and the solution was to create a separate new notebook and perform model pruning techniques there.

Training the ViT model presented a unique challenge. The original ViT, designed for the larger CIFAR-100 dataset, required adjustments for CIFAR-10. This, combined with its inherent complexity, resulted in slow training times. However, due to the specific architecture of the published ViT model we followed, particularly the custom patch layer implementation, we were unable to optimize the model further. This limitation prevented us from leveraging techniques like loading saved models between training subsets or quantization using TensorFlow Lite or ONNX for potentially faster inference. Consequently, we focused on training the ViT model and comparing its results with AlexNet's on the same testing data. This approach allowed us to evaluate the trade-off between model complexity, training speed, and achievable accuracy. Additionally, we noticed that the ViT has less parameters than AlexNet but performed slower during training.Transformers in general are inherently slower due to their complexity.

Design & Contribution

We divided the workload: Neha focused on training and optimizing the AlexNet model, while Ashley tackled training the ViT model and debugging its architecture to enable optimization techniques similar to those applied to AlexNet.

The goal of our project overall was to identify the difference in complexity, training, inference and optimization on older state-of-the-art DNN models compared to recent state-of-the-art models. Our contribution to the field included a clear demonstration of difference implementation difficulty, inference time, and the accuracy of these models. For example, we were able to easily improve AlexNet through various optimization techniques and improve both the inference time and accuracy results. However, even with the optimized versions of the AlexNet model the ViT model is still superior in accuracy.

Additionally, our implementation of these models on accessible resources such as Google Colab's free GPU highlights what's achievable, what modifications are necessary, and what cannot be done with these resources. We aimed to make these complex models achievable on free resources and were able to identify the limitations. For example, optimization techniques on the ViT model was a limitation to us on Google Colab with the standard model definition used by many developers online. We weren't able to try other methods of defining the model due to the complexity of training the model and the time constraints.

Evaluation & Results

AlexNet Accuracy & Results

Original Model:

```
Top 1 accuracy:  0.5386
Top 5 accuracy:  0.8949
22.634063708999975 ms
```

Original Model(tflite):

```
Top 1 accuracy:  0.5386
Top 5 accuracy:  0.8949
52.964010123002936 ms
```

Model with Float16 Quantization(tflite):

```
Top 1 accuracy:  0.5386
Top 5 accuracy:  0.8949
48.07981562799978 ms
```

Model with Dynamic Range Quantization(tflite):

```
Top 1 accuracy:  0.5381
Top 5 accuracy:  0.8964
39.00650098899723 ms
```

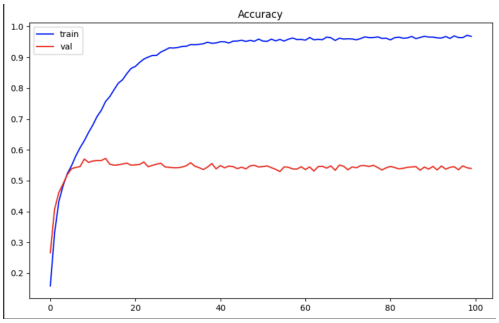
Model with Residual connections Inference results:

```
Top 1 accuracy:  0.5478
Top 5 accuracy:  0.903
36.53270785400025 ms
```

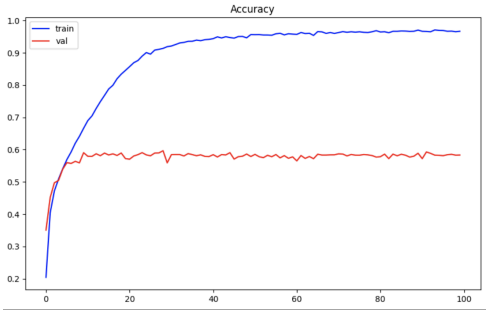
Model with Pruning Inference results:

```
Top 1 accuracy:  0.5478
Top 5 accuracy:  0.9027
19.331253641008516 ms
```

Training Results:



Training Results for the Prunning Model:



AlexNet Model Size: 282.2MB
AlexNet Model Size with Residual Connections: 261.8MB
AlexNet Model Size after Pruning Optimization: 94.2MB

<u>ViT Model Inference Accuracy & Results</u>

Model Parameters:

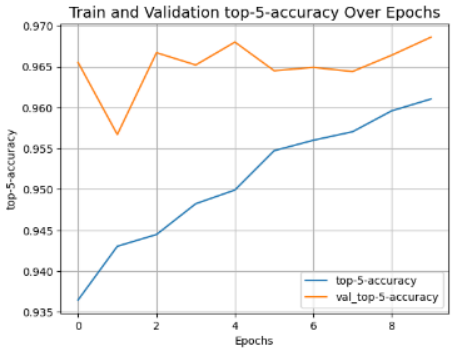
```
Total params: 6,061,585 (23.12 MB)
Trainable params: 6,061,578 (23.12 MB)
Non-trainable params: 7 (28.00 B)
time: 148 ms (started: 2024-05-06 00:46:53 +00:00)
```

ViT Inference Results:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 68s 218ms/step
Top 1 Accuracy: 64.97%
Top 5 Accuracy: 96.86%
time: 1min 14s (started: 2024-05-03 08:00:32 +00:00)
```

ViT Evaluation Results:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 68s 216ms/step - accuracy: 0.6527 - loss: 0.9807 - top-5-accuracy: 0.9695
Test accuracy: 64.97%
Test top 5 accuracy: 96.86%
```



ViT Training Results

Our model design for AlexNet and ViT does not perform better than existing models due to limiting factors of GPU and overall performance capacity. Our design overall demonstrates how to train and optimize existing models.

References & Links

Vit Model:

- https://paperswithcode.com/sota/image-classification-on-cifar-10
- https://github.com/keras-team/keras-io/blob/master/examples/vision/image_classification_with_vision_transformer.py
- https://medium.com/@arptoth/how-to-measure-execution-time-in-google-colab-707cc9aad1c8
- https://github.com/keras-team/keras/issues/18430
- https://www.tensorflow.org/guide/keras/making_new_layers_and_models_via_subclassing

AlexNet

- Architecture: https://github.com/KhuyenLE-maths/Alexnet_model_with_image_classification/blob/main/Alexnet_and_image_classification.ipynb
- Model Optimization:
  - https://www.tensorflow.org/lite/performance/model_optimization
  - https://www.tensorflow.org/lite/performance/post_training_float16_quant
  - https://studymachinelearning.com/model-quantization-methods-in-tensorflow-lite/
- Residual Networks:
  - https://medium.com/swlh/how-to-create-a-residual-network-in-tensorflow-and-keras-cd97f6c62557
  - https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-build-a-resnet-from-scratch-with-tensorflow-2-and-keras.md
  - https://stackoverflow.com/questions/64792460/how-to-code-a-residual-block-using-two-layers-of-a-basic-cnn-algorithm-built-wit
- Running on the GPU:
  - https://stackoverflow.com/questions/59616788/how-to-move-a-tensorflow-keras-model-to-gpu

Colab Notebooks

AlexNet:
https://colab.research.google.com/drive/1K6_bLhOYl37_ufMOnWd1OE_dUeRa1G2i?usp=sharing
AlexNet Pruning model:
https://colab.research.google.com/drive/11iYFKFQ3zoSpC__WMDAIBdrnDMSxEtpn?usp=sharing
ViT: https://colab.research.google.com/drive/19PX4-3xYrpbUoJuWeGiObiHMR_lDJkYE?usp=sharing