ENDG 511 Lab 3 Assignment: Advanced Model Optimization

This colab notebook provides code and a framework for ***Lab 3***. You can work out your solutions here.

Goals

In this lab, you will be introduced to advanced implmentations of the model optimization methods presented in Lab 2, and you will learn how to use them to create more efficient deep learning models. Model optimization is key when deploying deep learning models in resource-constrained IoT hardware and for low-latency sensing applications. You will also have the opportunity to explore other advanced methods. The goals of this lab are:

- Understand the basics of iterative pruning and how to develop an iterative pruning schedule
- Apply iterative pruning and weight clustering to an MNIST model.
- Become familiar with applying any method of TensorFlow Model Optimization Toolkit's (https://www.tensorflow.org/model_optimization/guide/) by reviewing the examples and implementing them yourself.
- Understand and apply quantization aware training to an MNIST model (optional)

Layout

This lab is split into **three** parts.

- **Part 1**: Apply iterative pruning to an MNIST model and evaluate the pruned model.
- **Part 2 (Optional - no marks)**: Apply quantization aware training and evaluate the quantized model.
- **Part 3**: Apply weight clustering alone and then combine weight clustering with iterative pruning.

How to submit the Assignment

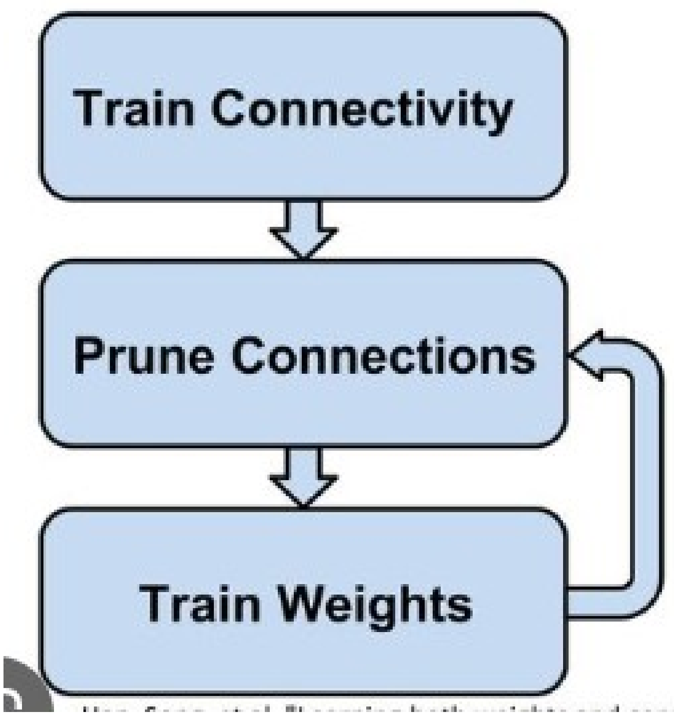
- You are required to submit the completed python notebook and a pdf version of it in a Dropbox folder on D2L.
- This is an individual assignment, and all the assignments must be submitted individually.
- This assignment can be completed directly on Google Colab, but you are free to choose any other computing resource.
- Lab sessions will be held to go over the main concepts and help you with the assignment.

Part 1: Iterative Pruning

This part of the lab demonstrates applying iterative pruning to a neural network to reduce size and inference while maintaining a high accuracy compared to the original neural network. At a high level, the steps required to implement iterative pruning and evaluate a model are as follows:

- Build and train the dense baseline
- Prune the model (but not till the target sparsity)
- Train the pruned model
- Repeat steps 2 and 3 until the target sparsity is reached.
- Evaluate the final model

One advantage of iterative pruning over one-shot pruning (which was demonstrated in Lab 2) is that it allows for more fine-grained control over the compression process. By retraining the network after each pruning step, the model is able to adapt to the pruning and maintain its performance on the target task. The image below can help you visualize iterative pruning:



Import and install all required modules

```
!pip install -q tensorflow-model-optimization

import tempfile
import os
import time

import tensorflow as tf
```

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

from tensorflow import keras
import tensorflow_model_optimization as tfmot


[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Build and Train a neural network for MNIST without pruning

Similar to the examples presented in Lab1 we will build and train a neural network for the MNIST dataset without any model optimization. This will be our base model for the remainder of this lab. We also save the model before training - you can choose to use this untrained model in the exercises.

```
# Load MNIST dataset
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

# Normalize the input image so that each pixel value is between 0 and
1.
train_images = train_images / 255.0
test_images = test_images / 255.0

# Define the model architecture.
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

# Save untrained model
model.save('untrained_base_model.h5')

# Compile the model
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
model.fit(
  train_images,
  train_labels,
```

```
  epochs=5,
  validation_split=0.1,
)
```

Let's display the architecture of our model:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Save model

Let us save the trained model so that we can evaluate it at a later stage.

```
# Save your trained model
model.save('trained_base_model.h5')
```

Define Iterative Pruning Function

We define a function that takes an unpruned model along with other parameters, performs iterative pruning and returns the pruned model. These are parameters of iterative pruning along with their explanation.

The **Polynomial Decay** pruning schedule: the degree of sparsity is changed during training and it is not kept constant.

initial_sparsity: The initial sparsity is the sparsity of the model at the beginning of the iterative pruning procedure. If a none zero value is provided, the model is one-shot pruned to the initial sparsity at the beginning.

final_sparsity: This is the final target sparsity of the model.

begin_step: The training step where iterative pruning will start to be applied. At this step the model is pruned to the initial sparsity value.

end_step: The last training step where iterative pruning will be applied. After this step has been completed the model would have reached its final sparsity.

frequency: How often to apply pruning

power: The default is linear. This is the power of the polynomial function and how the sparsity changes from the initial sparsity at the begin step to the final sparsity at the end step

Number of Steps per Epoch = (Total Number of Training Samples) / (Batch Size)

You need to make sure to choose a begin_step and end_step that are not out of the range of the training steps.

```
def iterative_pruning(model, initial_sparsity, final_sparsity,
begin_step, end_step, train_images, train_labels, epochs):
  prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

  # Define model for pruning.
  pruning_params = {
      'pruning_schedule':
tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=initial_sparsity
,
        final_sparsity=final_sparsity, begin_step=begin_step,
end_step=end_step, frequency=100)
  }

  pruned_model = prune_low_magnitude(model, **pruning_params)

  # `prune_low_magnitude` requires a recompile.
  optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
  pruned_model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])


  callbacks = [
    tfmot.sparsity.keras.UpdatePruningStep(),
  ]

  pruned_model.fit(train_images, train_labels, epochs=epochs,
validation_split=0.1,
                    callbacks=callbacks)

  # Strip pruning wrappers
  stripped_pruned_model =
tfmot.sparsity.keras.strip_pruning(pruned_model)

  return pruned_model, stripped_pruned_model

pruned_model, stripped_pruned_model = iterative_pruning(model, 0, 0.5,
0, 3000, train_images, train_labels, 3)
```

```
Epoch 1/3
1688/1688 [==============================] - 9s 4ms/step - loss:
0.0707 - accuracy: 0.9780 - val_loss: 0.0644 - val_accuracy: 0.9808
Epoch 2/3
1688/1688 [==============================] - 6s 4ms/step - loss:
0.0618 - accuracy: 0.9810 - val_loss: 0.0658 - val_accuracy: 0.9822
Epoch 3/3
1688/1688 [==============================] - 6s 3ms/step - loss:
0.0504 - accuracy: 0.9850 - val_loss: 0.0600 - val_accuracy: 0.9833
```

```
stripped_pruned_model.save('stripped_pruned_model.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.
```

Confirm that pruning was correctly applied

```
def print_model_weights_sparsity(model):

    for layer in model.layers:
        if isinstance(layer, tf.keras.layers.Wrapper):
            weights = layer.trainable_weights
        else:
            weights = layer.weights
        for weight in weights:
            if "kernel" not in weight.name or "centroid" in
weight.name:
                continue
            weight_size = weight.numpy().size
            zero_num = np.count_nonzero(weight == 0)
            print(
                f"{weight.name}: {zero_num/weight_size:.2%} sparsity
",
                f"({zero_num}/{weight_size})",
            )
print_model_weights_sparsity(stripped_pruned_model)
```

Evaluate the model

Finally, we compare the pruned model to the base model. We can see that the accuracy and inference time is comparable, however, the pruned model is much smaller in size.

Note: Pruning is capable of improving inference time significantly, however, additional libraries and modifications are needed to see inference improvements as a result of pruning (Pruning inference improvements is very hardware specific!). If you are curious you can read this paper which explains how sparse models can be used to accelerate inference (https://arxiv.org/pdf/1911.09723.pdf)

```
# Evaluate prediction accuracy
model = tf.keras.models.load_model('trained_base_model.h5')
test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=0)
test_loss_pruned_50, test_acc_pruned_50 =
pruned_model.evaluate(test_images,  test_labels, verbose=0)


# Evaluate Model Size
def get_gzipped_model_size(file):
  # Returns size of gzipped model, in bytes.
  import os
  import zipfile

  _, zipped_file = tempfile.mkstemp('.zip')
  with zipfile.ZipFile(zipped_file, 'w',
compression=zipfile.ZIP_DEFLATED) as f:
    f.write(file)

  return os.path.getsize(zipped_file)


# Evaluate Inference Time
startTime = time.time()
prediction = model.predict(test_images)
executionTime = (time.time() - startTime)/len(test_images)

startTime = time.time()
prediction = pruned_model.predict(test_images)
executionTimePruned50 = (time.time() - startTime)/len(test_images)

base_model_size = get_gzipped_model_size('untrained_base_model.h5')
## Print without stripping
pruned_model_size = get_gzipped_model_size('stripped_pruned_model.h5')

# Print
print('\nBase Model Accuracy:', test_acc*100, '%')
print("Base Model Size: %.2f bytes" % (base_model_size))
print("Base Inference Time is", executionTime, "s")

print('\nPruned Model Accuracy:', test_acc_pruned_50*100, '%')
print("Pruned Model Size: %.2f bytes" % (pruned_model_size))
print("Pruned Inference Time is", executionTimePruned50, "s")
```

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step

Base Model Accuracy: 97.97999858856201 %
Base Model Size: 374919.00 bytes
Base Inference Time is 8.92139196395874e-05 s
```

```
Pruned Model Accuracy: 98.12999963760376 %
Pruned Model Size: 234456.00 bytes
Pruned Inference Time is 9.187750816345214e-05 s
```

Excercise: (1 points)

Question 1: Apply the iterative pruning function provided above with a final sparsity of 90%. For the parameters of the iterative pruning function choose suitable values that make sense and give reasons for your choices. Compare the accuracy of the iterative pruned 90% model to the one-shot pruned 90% model from Lab2. Next, apply iterative pruning to a final sparsity of 95% and try to minimize any accuracy loss. ***(1.5 points)***

Defining Evaluation Methods

```
# Evaluate Model Size
def get_gzipped_model_size(file):
  # Returns size of gzipped model, in bytes.
  import os
  import zipfile

  _, zipped_file = tempfile.mkstemp('.zip')
  with zipfile.ZipFile(zipped_file, 'w',
compression=zipfile.ZIP_DEFLATED) as f:
    f.write(file)

  return os.path.getsize(zipped_file)

def evaluate(model, test_images, test_labels,
model_path='untrained_base_model.h5'):
  # Evaluates the model by providing the model accuracy, size, and
inference time.
  test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=0)

  # Evaluate inference time.
  startTime = time.time()
  prediction = model.predict(test_images)
  executionTime = (time.time() - startTime)/len(test_images)

  # Retrieve the model size.
  model_size = get_gzipped_model_size(model_path)
  return test_acc, model_size, executionTime
```

Iterative Pruning with Final Sparsity at 90%

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
pruned_90_model, stripped_pruned_90_model =
iterative_pruning(base_model, 0.20, 0.90, 1500, 6000, train_images,
```

```
train_labels, 4)
stripped_pruned_90_model.save('stripped_pruned_90_model.h5')
```

```
Epoch 1/4
1688/1688 [==============================] - 8s 4ms/step - loss:
0.0678 - accuracy: 0.9785 - val_loss: 0.0667 - val_accuracy: 0.9792
Epoch 2/4
1688/1688 [==============================] - 6s 4ms/step - loss:
0.0835 - accuracy: 0.9738 - val_loss: 0.0825 - val_accuracy: 0.9768
Epoch 3/4
1688/1688 [==============================] - 6s 4ms/step - loss:
0.1836 - accuracy: 0.9470 - val_loss: 0.1262 - val_accuracy: 0.9658
Epoch 4/4
1688/1688 [==============================] - 6s 4ms/step - loss:
0.2078 - accuracy: 0.9398 - val_loss: 0.1092 - val_accuracy: 0.9703
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.

c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Choices of Parameters

- Initial Sparsity: 0.20 is chosen to add a jumpstart to the pruning process as 0.90 is the targeted final sparsity, the earlier values of the weights will not matter as much.
- Final Sparsity: 0.90 is chosen due to the requirements of the question.
- Begins step: 1500 to start pruning just before the end of the first epoch to perform one shot pruning of 20% of the weights with values close to zero.
- End step: 6000 to stop pruning just before the end of the last epoch with 752 more steps left without pruning to maintain minimal agression with pruning at the last steps and allowing the weights to fine tune with the training images.
- Epoch: 4 is chosen because it was observed that validation accuracy starts to decrease and validation loss starts to increase at this point when training the base model without any pruning.

Model Evaluation and Comparison of Pruning Methods

```
test_acc, model_size, executionTime = evaluate(pruned_90_model,
test_images, test_labels, 'stripped_pruned_90_model.h5')
print('\nPruned Model Accuracy:', test_acc*100, '%')
print("Pruned Model Size: %.2f bytes" % (model_size))
print("Pruned Inference Time is", executionTime, "s")
```

```
313/313 [==============================] - 1s 2ms/step
```

```
Pruned Model Accuracy: 95.95999717712402 %
Pruned Model Size: 77191.00 bytes
Pruned Inference Time is 8.017294406890869e-05 s
```

One Shot 90% Pruning Lab 2

- Pruned Model Accuracy: 95.3499972820282 %
- Pruned Model Size: 80123.00 bytes
- Pruned Inference Time is 9.749836921691894e-05 s

Iterative Pruning with Final Sparsity at 90%

- Pruned Model Accuracy: 95.95999717712402 %
- Pruned Model Size: 77191.00 bytes
- Pruned Inference Time is 8.017294406890869e-05 s

Comparisons

Both models are close in accuracy with iterative pruning having a slightly higher accuracy by ~0.61%. Iterative pruning also produces a smaller model than the one shot pruning model by 2932 bytes. Finally iterative pruning produces a slightly faster model by 17.3us.

Iterative Pruning with Final Sparsity at 95%

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
pruned_95_model, stripped_pruned_95_model = 
iterative_pruning(base_model, 0, 0.95, 0, 4000, train_images, 
train_labels, 7)
stripped_pruned_95_model.save('stripped_pruned_95_model.h5')

Epoch 1/7
1688/1688 [==============================] - 9s 4ms/step - loss: 
0.0891 - accuracy: 0.9719 - val_loss: 0.0907 - val_accuracy: 0.9748
Epoch 2/7
1688/1688 [==============================] - 6s 4ms/step - loss: 
0.3284 - accuracy: 0.9031 - val_loss: 0.2744 - val_accuracy: 0.9347
Epoch 3/7
1688/1688 [==============================] - 7s 4ms/step - loss: 
0.4580 - accuracy: 0.8636 - val_loss: 0.2315 - val_accuracy: 0.9383
Epoch 4/7
1688/1688 [==============================] - 7s 4ms/step - loss: 
0.4114 - accuracy: 0.8739 - val_loss: 0.2174 - val_accuracy: 0.9418
Epoch 5/7
1688/1688 [==============================] - 7s 4ms/step - loss: 
0.3890 - accuracy: 0.8792 - val_loss: 0.2104 - val_accuracy: 0.9430
Epoch 6/7
1688/1688 [==============================] - 6s 4ms/step - loss: 
0.3783 - accuracy: 0.8820 - val_loss: 0.2049 - val_accuracy: 0.9438
Epoch 7/7
1688/1688 [==============================] - 6s 4ms/step - loss: 
```

```
0.3700 - accuracy: 0.8845 - val_loss: 0.2010 - val_accuracy: 0.9442
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.
```

- When starting with an initial sparsity at 0.40 and starting at 1000 step, there is a large drop in validation accuracy at the second epoch from 0.98 to 0.94 and an increase in validation loss from 0.08 to 0.25. The metric remains marginally the same throughout the epochs.
- When keeping the same parameters but lowering the initial sparsity to 0.10, there is a large drop in validation accuracy at the second epoch from 0.98 to 0.93 and an increase in validation loss from 0.07 to 0.29.

- When choosing the keep initial sparsity to 0% and the begin step at 0, then the validation accuracy drops at the second epoch from 0.97 to 0.93 and an increase in validation loss from 0.09 to 0.27. The metrics slightly increase by 1% keeping marginally the same at ~94% throughout the epochs.

Part 2: Quantization Aware Training - Optional (Code Provided)

This part of the lab demonstrates applying quantization aware training to a neural network to reduce size and inference while maintaining a high accuracy.

In quantization aware training, the quantization process is integrated into the training loop, so the model is optimized to perform well after quantization. This is different from post-training quantization (Lab 3) where quantization is applied in the end after training. This helps to reduce the gap between floating-point and quantized performance, and enables the deployment of deep learning models on low-power, low-memory devices.

At a high level, the steps required to quantize and evaluate a model are as follows:

- Build and train the dense baseline
- Fine tune the model by applying the quantization aware training API, see the accuracy, and export a quantization aware model.
- Apply quantization during conversion to TFLite
- Evaluate the model

Note: a quantization aware model is not actually quantized. Creating a quantized model is a separate step.

Load base model

First, let us load the base model we have trained earlier.

```
model_to_quantize =
tf.keras.models.load_model('trained_base_model.h5')
model_to_quantize.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 784)               0         
                                                                 
 dense (Dense)               (None, 128)               100480    
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Define Quantization Aware Training function

For quantization aware training, we first train a quantization aware model (from the original trained model) using a subset of the training data. The quantization aware model is then converted to a TFLite model and quantized. The quantized weights are usually more accurate as the model was "quantization aware" prior to the quantization operation.

By default, the TensorFlow QAT APIs assume 8-bit quantization.

```
def quantization_aware_training(model, x_train, y_train):
    quantize_model = tfmot.quantization.keras.quantize_model

    # q_aware stands for for quantization aware.
    q_aware_model = quantize_model(model)

    # `quantize_model` requires a recompile.
    q_aware_model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

    q_aware_model.summary()

    train_images_subset = x_train[0:10000] # out of 60000
    train_labels_subset = y_train[0:10000]

    ## Note that the resulting model is quantization aware but not
quantized (e.g. the weights are float32 instead of int8)
```

```
    q_aware_model.fit(train_images_subset, train_labels_subset,
                  batch_size=500, epochs=5, validation_split=0.1)

    # Note: a quantization aware model is not actually quantized.
Creating a quantized model is a separate step.
    ## Convert the quantization aware model to TFLite and apply
quantization through the optimization options
    converter =
tf.lite.TFLiteConverter.from_keras_model(q_aware_model)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]

    quantized_tflite_model = converter.convert()

    return quantized_tflite_model

quantized_tflite_model =
quantization_aware_training(model_to_quantize, train_images,
train_labels)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 quantize_layer (QuantizeLa  (None, 28, 28)            3
 yer)

 quant_flatten (QuantizeWra  (None, 784)               1
 pperV2)

 quant_dense (QuantizeWrapp  (None, 128)               100485
 erV2)

 quant_dropout (QuantizeWra  (None, 128)               1
 pperV2)

 quant_dense_1 (QuantizeWra  (None, 10)                1295
 pperV2)

=================================================================
Total params: 101785 (397.60 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 15 (60.00 Byte)
_________________________________________________________________
Epoch 1/5
18/18 [==============================] - 1s 29ms/step - loss: 0.0898 -
accuracy: 0.9770 - val_loss: 0.0323 - val_accuracy: 0.9930
Epoch 2/5
18/18 [==============================] - 0s 12ms/step - loss: 0.0608 -
accuracy: 0.9830 - val_loss: 0.0286 - val_accuracy: 0.9920
Epoch 3/5
```

```
18/18 [==============================] - 0s 12ms/step - loss: 0.0494 -
accuracy: 0.9870 - val_loss: 0.0270 - val_accuracy: 0.9940
Epoch 4/5
18/18 [==============================] - 0s 11ms/step - loss: 0.0452 -
accuracy: 0.9878 - val_loss: 0.0278 - val_accuracy: 0.9950
Epoch 5/5
18/18 [==============================] - 0s 10ms/step - loss: 0.0357 -
accuracy: 0.9910 - val_loss: 0.0260 - val_accuracy: 0.9950
INFO:tensorflow:Assets written to: C:\Users\johns\AppData\Local\Temp\
tmpyo9jphe5\assets

INFO:tensorflow:Assets written to: C:\Users\johns\AppData\Local\Temp\
tmpyo9jphe5\assets
c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\tensorflow\lite\python\convert.py:953: UserWarning:
Statistics for quantized inputs were expected, but not specified;
continuing anyway.
  warnings.warn(
```

Save TFLite model and load model into an interpeter

First we save the TFLite model, this will allow us to load it into an interpreter. To perform an inference with a TensorFlow Lite model, you must run it through an interpreter. The TensorFlow Lite interpreter is designed to be lean and fast. You can find more information on the TFLite interpeter here: https://www.tensorflow.org/lite/guide/inference

```
# Save TFLite Model
with open('quantized_tflite_model.tflite', 'wb') as f:
  f.write(quantized_tflite_model)
# Load model into interpeter
interpreter_quant =
tf.lite.Interpreter(model_path=str('quantized_tflite_model.tflite'))
interpreter_quant.allocate_tensors()
```

Evaluate the model

Finally, we evaluate the quantization aware model in terms of accuracy, inference time and model size. There is a very slight difference accuracy compared to Lab2, the reason is the model performs extremely well with post-training quantization that even when doing quantization aware training the benefits are minimal. Furthermore, the model trained is not very complex and therefore the added benefits of QAT are not very evident. For models that see a significant drop in accuracy due to post training quatnization, quantization aware training may be capable of producing better accuracy with no impact to model size or inference time.

```
# A helper function to evaluate the TF Lite model using "test"
dataset.
def evaluate_model(interpreter, model_path):
  input_index = interpreter.get_input_details()[0]["index"]
```

```
  output_index = interpreter.get_output_details()[0]["index"]

  # Run predictions on every image in the "test" dataset.
  prediction_digits = []
  for test_image in test_images:
    # Pre-processing: add batch dimension and convert to float32 to
match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    startTime = time.time()
    interpreter.invoke()
    executionTime = (time.time() - startTime)/len(test_images)

    # Post-processing: remove batch dimension and find the digit with
highest
    # probability.
    output = interpreter.tensor(output_index)
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

  # Compare prediction results with ground truth labels to calculate
accuracy.
  accurate_count = 0
  for index in range(len(prediction_digits)):
    if prediction_digits[index] == test_labels[index]:
      accurate_count += 1
  accuracy = accurate_count * 1.0 / len(prediction_digits)

  model_size = get_gzipped_model_size(model_path)
  # Print
  print('\nModel Accuracy:', accuracy*100, '%')
  print("Model Size: %.2f bytes" % (model_size))
  print("Inference Time is", executionTime, "s")
  return accuracy, model_size, executionTime

evaluate_model(interpreter_quant, 'quantized_tflite_model.tflite')
```

```
Model Accuracy: 97.97 %
Model Size: 83743.00 bytes
Inference Time is 0.0 s

(0.9797, 83743, 0.0)
```

Part 3: Weight Clustering & Iterative Pruning Compression

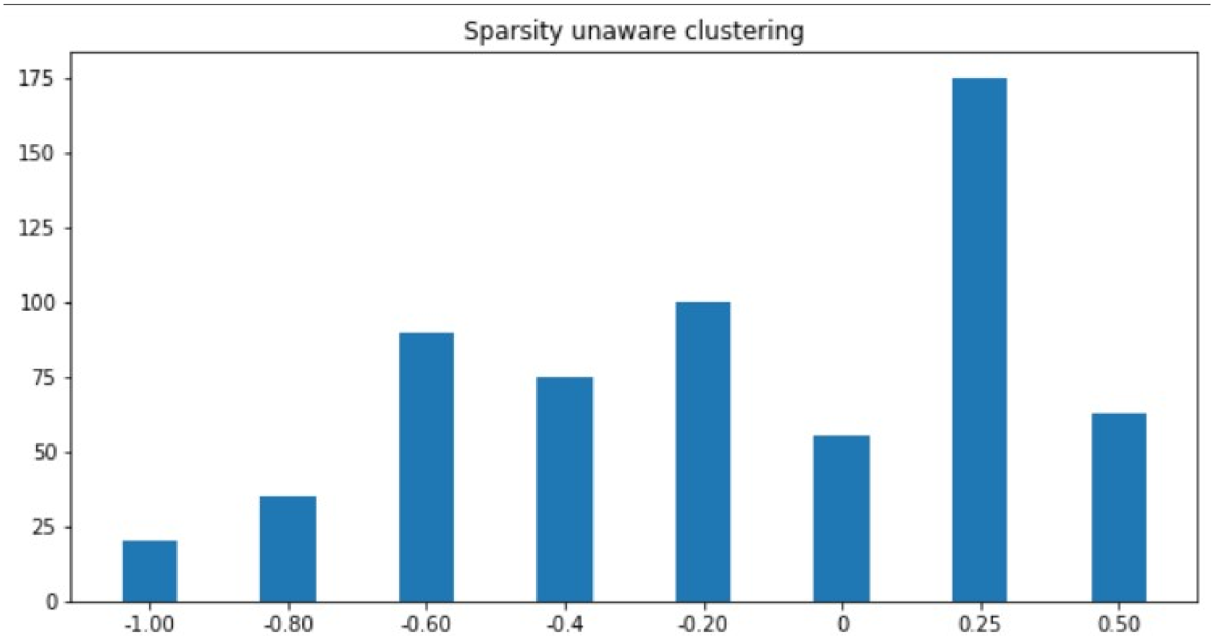
There are other common model compression techniques that are constantly being researched and improved upon. Weight clustering is one of them that was discussed in class. There are many websites and research papers that discuss weight clustering including the Deep Compression paper on D2L. In this part of the lab the exercises will be focused on Weight Clustering. Here are two websites that can help you with the lab exercises (Feel free read further papers!).

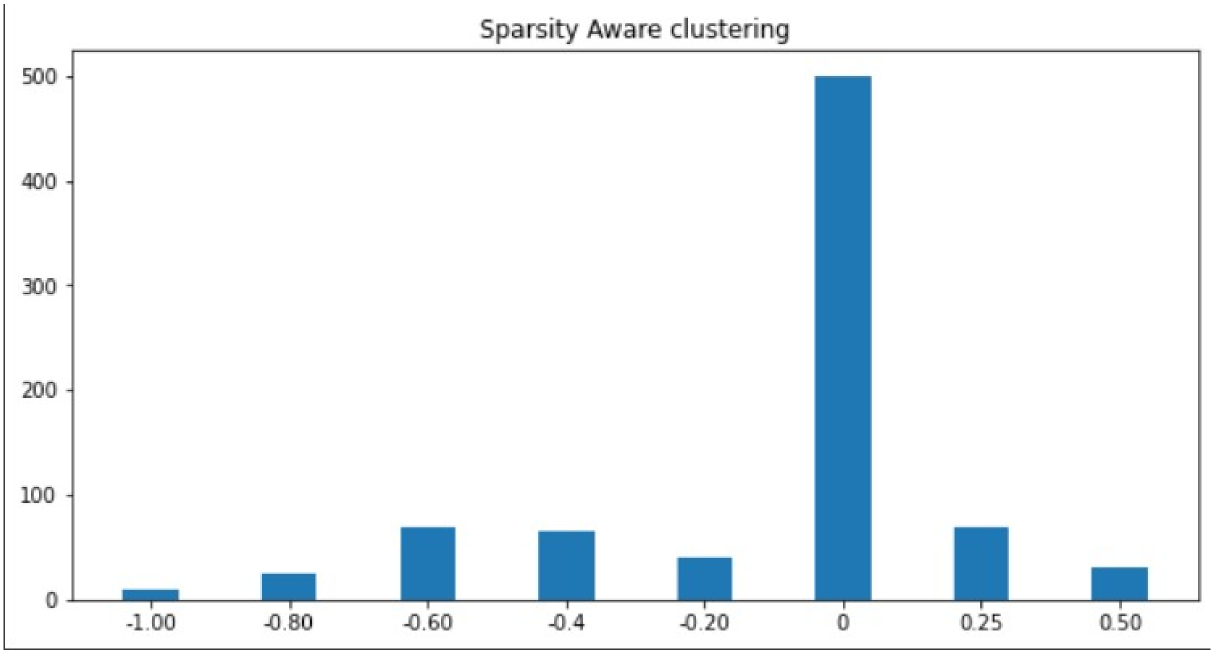
https://www.tensorflow.org/model_optimization/guide/clustering/clustering_example

https://www.tensorflow.org/model_optimization/guide/combine/sparse_clustering_example

Clustering, or weight sharing, reduces the number of unique weight values in a model, leading to benefits for deployment. It first groups the weights of each layer into N clusters, then shares the cluster's centroid value for all the weights belonging to the cluster.

The two images below highlight how the weights are distributed when 8 clustered are used. The first image is when clustering is done without maintaining sparsity and the second image is when sparsity is maintained (these two methods are discussed in the questions below).





Exercises (5.5 points)

Question 1 (2 points): Using the **first** link above and the content of all previous labs, write a function that applies weight clustering. Your function must take in 3 parameters (model, number of clusters and centroid initialization). Experiment and evaluate your model (size and accuracy) with 2 clusters and 16 clusters, and with KMEANS_PLUS_PLUS centroid initialization and comment on your observations.

```
def apply_weight_clustering(model, nc: int=16, centroid_init:
str=None):
    """
    This function applies weight clustering to a model.

    Parameters
    ----------
        model: Keras model
          This is a loaded keras model.

        nc: int
          This is the number of clusters.

        centroid_init: str
          This is the centroid initialization type.
    """
    cluster_weights = tfmot.clustering.keras.cluster_weights

    if centroid_init is None:
      CentroidInitialization =
tfmot.clustering.keras.CentroidInitialization
      centroid_init = CentroidInitialization.LINEAR
```

```
    clustering_params = {
      'number_of_clusters': nc,
      'cluster_centroids_init': centroid_init
    }

    # Cluster a whole model
    clustered_model = cluster_weights(model, **clustering_params)

    # Use smaller learning rate for fine-tuning clustered model
    opt = keras.optimizers.Adam(learning_rate=1e-5)

    clustered_model.compile(

loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      optimizer=opt,
      metrics=['accuracy'])

    clustered_model.summary()

    # Fine-tune model
    clustered_model.fit(
      train_images,
      train_labels,
      batch_size=32,
      epochs=1,
      validation_split=0.1)

    # Stripped clustering is necessary to see the benefits of
clustering.
    stripped_clustered_model =
tfmot.clustering.keras.strip_clustering(clustered_model)
    return clustered_model, stripped_clustered_model
```

Clustering Model with 2 Clusters

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization
centroid_init = CentroidInitialization.KMEANS_PLUS_PLUS
clustered_2_model, stripped_clustered_2_model =
apply_weight_clustering(base_model, nc=2, centroid_init=centroid_init)
stripped_clustered_2_model.save('stripped_clustered_2_model.h5')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cluster_flatten (ClusterWe  (None, 784)               0
 ights)
```

```
 cluster_dense (ClusterWeig  (None, 128)               200834
 hts)

 cluster_dropout (ClusterWe  (None, 128)               0
 ights)

 cluster_dense_1 (ClusterWe  (None, 10)                2572
 ights)

=================================================================
Total params: 203406 (1.16 MB)
Trainable params: 101774 (397.55 KB)
Non-trainable params: 101632 (794.00 KB)
_________________________________________________________________
1688/1688 [==============================] - 17s 9ms/step - loss:
6.4698 - accuracy: 0.4700 - val_loss: 2.5068 - val_accuracy: 0.6825
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.
```

```
test_acc, model_size, executionTime = evaluate(clustered_2_model,
test_images, test_labels, 'stripped_clustered_2_model.h5')
print('\nClustered Model Accuracy:', test_acc*100, '%')
print("Clustered Model Size: %.2f bytes" % (model_size))
print("Clustered Inference Time is", executionTime, "s")
```

```
313/313 [==============================] - 1s 4ms/step

Clustered Model Accuracy: 66.75000190734863 %
Clustered Model Size: 24481.00 bytes
Clustered Inference Time is 0.00015413177013397217 s
```

Clustering Model with 16 Clusters

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization
centroid_init = CentroidInitialization.KMEANS_PLUS_PLUS
clustered_16_model, stripped_clustered_16_model =
apply_weight_clustering(base_model, nc=16,
centroid_init=centroid_init)
stripped_clustered_16_model.save('stripped_clustered_16_model.h5')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cluster_flatten (ClusterWe  (None, 784)               0
 ights)
```

```
 cluster_dense (ClusterWeig  (None, 128)               200848
 hts)

 cluster_dropout (ClusterWe  (None, 128)               0
 ights)

 cluster_dense_1 (ClusterWe  (None, 10)                2586
 ights)

=================================================================
Total params: 203434 (1.16 MB)
Trainable params: 101802 (397.66 KB)
Non-trainable params: 101632 (794.00 KB)
_________________________________________________________________
1688/1688 [==============================] - 23s 13ms/step - loss:
0.0393 - accuracy: 0.9891 - val_loss: 0.0676 - val_accuracy: 0.9797
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.

c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
test_acc, model_size, executionTime = evaluate(clustered_16_model,
test_images, test_labels, 'stripped_clustered_16_model.h5')
print('\nClustered Model Accuracy:', test_acc*100, '%')
print("Clustered Model Size: %.2f bytes" % (model_size))
print("Clustered Inference Time is", executionTime, "s")
```

```
313/313 [==============================] - 2s 6ms/step

Clustered Model Accuracy: 98.00999760627747 %
Clustered Model Size: 75470.00 bytes
Clustered Inference Time is 0.00021111106872558595 s
```

Observations Cluster 2 and Cluster 16 Models

Model with 2 clusters achieved an accuracy of 66.75%, a size of 24481 bytes, and an inference time of 154.13us. The model with 16 clusters acheived an accuracy of 98.01%, a model size of 75470 bytes, and an inference time of 211.11us.

It was observed that having more clusters allowed the model to have a better generalization in the classes which resulted in a higher accuracy because more clusters meant more variation in the weight values, thus more precision in the classifications. However, a tradeoff is that this will increase the model size as shown which almost tripled the size. This is because more clusters means storing more weight values. However, this did not really change the total number of parameters, it was only observed to have a slight increase because the actual model layers did not change. Lastly, more clusters means higher inference times because there more cases to check to determine which cluster a value corresponds to, as oppose to having two clusters only had 2 comparisons to match the value to the closest cluster.

Question 2 (1 point): Using the resources present above and the content of all previous labs, apply weight clustering with iterative pruning (50% sparsity). Do this by calling your iterative pruning (Part1) followed by your weight clustering functions (of Part-3 Question 1). Print the final sparsity of your model and commment on it, and the performance and size versus what you observed in Q1.

Perform Pruning with 50% Sparsity

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
pruned_model, stripped_pruned_model = iterative_pruning(base_model, 0,
0.5, 0, 3000, train_images, train_labels, 3)
print("Model sparsity pre-clustered")
print_model_weights_sparsity(stripped_pruned_model)
stripped_pruned_model.save("trained_stripped_pruned_base_model.h5")
```

```
Epoch 1/3
1688/1688 [==============================] - 8s 4ms/step - loss:
0.0689 - accuracy: 0.9782 - val_loss: 0.0639 - val_accuracy: 0.9813
Epoch 2/3
1688/1688 [==============================] - 6s 4ms/step - loss:
0.0613 - accuracy: 0.9801 - val_loss: 0.0682 - val_accuracy: 0.9810
Epoch 3/3
1688/1688 [==============================] - 6s 4ms/step - loss:
0.0519 - accuracy: 0.9836 - val_loss: 0.0634 - val_accuracy: 0.9802
Model sparsity pre-clustered
dense/kernel:0: 50.00% sparsity  (50176/100352)
dense_1/kernel:0: 50.00% sparsity  (640/1280)
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.

c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Perform 2 Clustering on Pruned Model

```
stripped_pruned_model =
tf.keras.models.load_model('trained_stripped_pruned_base_model.h5')
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization
centroid_init = CentroidInitialization.KMEANS_PLUS_PLUS
pruned_clustered_2_model, pruned_stripped_clustered_2_model =
```

```
apply_weight_clustering(stripped_pruned_model, nc=2,
centroid_init=centroid_init)
pruned_stripped_clustered_2_model.save('pruned_stripped_clustered_2_mo
del.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file,
so the model was *not* compiled. Compile it manually.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cluster_flatten (ClusterWe  (None, 784)               0
 ights)

 cluster_dense (ClusterWeig  (None, 128)               200834
 hts)

 cluster_dropout (ClusterWe  (None, 128)               0
 ights)

 cluster_dense_1 (ClusterWe  (None, 10)                2572
 ights)

=================================================================
Total params: 203406 (1.16 MB)
Trainable params: 101774 (397.55 KB)
Non-trainable params: 101632 (794.00 KB)
_________________________________________________________________
1688/1688 [==============================] - 16s 9ms/step - loss:
1.9659 - accuracy: 0.2851 - val_loss: 1.0654 - val_accuracy: 0.7607
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.

c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
test_acc, model_size, executionTime =
evaluate(pruned_clustered_2_model, test_images, test_labels,
'pruned_stripped_clustered_2_model.h5')
print('\nPruned Clustered Model Accuracy:', test_acc*100, '%')
print("Pruned Clustered Model Size: %.2f bytes" % (model_size))
print("Pruned Clustered Inference Time is", executionTime, "s")
```

```
print("Model sparsity post-clustered")
print_model_weights_sparsity(pruned_stripped_clustered_2_model)
```

```
313/313 [==============================] - 1s 4ms/step

Pruned Clustered Model Accuracy: 73.94999861717224 %
Pruned Clustered Model Size: 20002.00 bytes
Pruned Clustered Inference Time is 0.00014613535404205323 s
Model sparsity post-clustered
kernel:0: 0.00% sparsity  (0/100352)
kernel:0: 0.00% sparsity  (0/1280)
```

Perform 16 Clustering on Pruned Model

```
stripped_pruned_model =
tf.keras.models.load_model('trained_stripped_pruned_base_model.h5')
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization
centroid_init = CentroidInitialization.KMEANS_PLUS_PLUS
pruned_clustered_16_model, pruned_stripped_clustered_16_model =
apply_weight_clustering(stripped_pruned_model, nc=16,
centroid_init=centroid_init)
pruned_stripped_clustered_16_model.save('pruned_stripped_clustered_16_
model.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file,
so the model was *not* compiled. Compile it manually.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 cluster_flatten (ClusterWe  (None, 784)               0
 ights)

 cluster_dense (ClusterWeig  (None, 128)               200848
 hts)

 cluster_dropout (ClusterWe  (None, 128)               0
 ights)

 cluster_dense_1 (ClusterWe  (None, 10)                2586
 ights)

=================================================================
Total params: 203434 (1.16 MB)
Trainable params: 101802 (397.66 KB)
Non-trainable params: 101632 (794.00 KB)
_________________________________________________________________
1688/1688 [==============================] - 27s 15ms/step - loss:
0.0250 - accuracy: 0.9927 - val_loss: 0.0627 - val_accuracy: 0.9812
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

test_acc, model_size, executionTime = evaluate(pruned_clustered_16_model, test_images, test_labels, 'pruned_stripped_clustered_16_model.h5')
print('\nPruned Clustered Model Accuracy:', test_acc*100, '%')
print("Pruned Clustered Model Size: %.2f bytes" % (model_size))
print("Pruned Clustered Inference Time is", executionTime, "s")

print("Model sparsity post-clustered")
print_model_weights_sparsity(pruned_stripped_clustered_16_model)

313/313 [==============================] - 2s 6ms/step

Pruned Clustered Model Accuracy: 98.14000129699707 %
Pruned Clustered Model Size: 61097.00 bytes
Pruned Clustered Inference Time is 0.0002096734285354614 s
Model sparsity post-clustered
kernel:0: 0.00% sparsity  (0/100352)
kernel:0: 0.00% sparsity  (0/1280)
```

Observations Pruning with 50% Sparsity and Clustering with 2 and 16 Clusters.

When performing pruning on the base model, it was verified that the sparsity was 50% using the `print_model_weights_sparsity` function. However after performing sparsity unaware clustering, the model sparsity was lost and was confirmed to be 0% using the same function to verify.

Performance wise, when pruning and clustering methods are applied with 50% sparsity and 2 clusters. The model achieved a 73.95% accuracy, 20002 bytes, and 146.14us inference time. In relation to Q1 when only applying 2 clusters without pruning, the model achieved 66.75% accuracy, 24481 bytes, and 154.13us inference time. In summary applying 50% pruning and 2 clusters, resulted in 7.2% increase in model accuracy, 4479 bytes decrease in model size, and 7.99us decrease in inference time.

Lastly, when applying 50% pruning and 16 clusters, the model achieved 98.14% accuracy, 61097 bytes, and 209.67us inference time. In relation to Q1, when only applying 16 clusters without pruning, the model achieved 98.01% accuracy, 75470 bytes, and 211.11us inference time. In summary, the model accuracies remained marginally the same but a slight 0.13% increase in accuracy is observed when applying both pruning and clustering. There is also significant decrease of 14373 bytes in model size and 1.44us decrease in inference time when applying both pruning and clustering.

Question 3 (2.5 points): Now apply iterative pruning (50% sparsity) followed by **sparsity preserving clustering** (https://www.tensorflow.org/model_optimization/guide/combine/sparse_clustering_example).

Print the final sparsity of your model and evaluate your final model. Compare the effects of combining the techniques together vs the techniques individually. Comment on how sparsity-preserving clustering differs from regular weight clustering.

```
# Sparsity preserving clustering
from
tensorflow_model_optimization.python.core.clustering.keras.experimenta
l import (
    cluster,
)

def sparsity_aware_clustering(model, nc: int=16, centroid_init:
str=None):
    """
    This function applies weight clustering to a model.

    Parameters
    ----------
        model: Keras model
          This is a loaded keras model.

        nc: int
          This is the number of clusters.

        centroid_init: str
          This is the centroid initialization type.
    """
    if centroid_init is None:
      CentroidInitialization =
tfmot.clustering.keras.CentroidInitialization
      centroid_init = CentroidInitialization.LINEAR

    cluster_weights = cluster.cluster_weights

    clustering_params = {
      'number_of_clusters': nc,
      'cluster_centroids_init': centroid_init,
      'preserve_sparsity': True
    }

    sparsity_clustered_model = cluster_weights(model,
**clustering_params)

    sparsity_clustered_model.compile(
          optimizer='adam',

loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])

    sparsity_clustered_model.fit(train_images, train_labels,epochs=3,
```

```
validation_split=0.1)

    # Stripped clustering is necessary to see the benefits of
clustering.
    stripped_clustered_model =
tfmot.clustering.keras.strip_clustering(sparsity_clustered_model)
    return sparsity_clustered_model, stripped_clustered_model
```

Perform Pruning with 50% Sparsity

```
base_model = tf.keras.models.load_model('trained_base_model.h5')
pruned_model, stripped_pruned_model = iterative_pruning(base_model, 0,
0.5, 0, 3000, train_images, train_labels, 3)
print("Model sparsity pre-clustered")
print_model_weights_sparsity(stripped_pruned_model)
stripped_pruned_model.save("trained_stripped_pruned_base_model.h5")

Epoch 1/3
1688/1688 [==============================] - 16s 6ms/step - loss:
0.0697 - accuracy: 0.9777 - val_loss: 0.0656 - val_accuracy: 0.9812
Epoch 2/3
1688/1688 [==============================] - 13s 8ms/step - loss:
0.0610 - accuracy: 0.9809 - val_loss: 0.0636 - val_accuracy: 0.9828
Epoch 3/3
1688/1688 [==============================] - 12s 7ms/step - loss:
0.0511 - accuracy: 0.9841 - val_loss: 0.0651 - val_accuracy: 0.9813
Model sparsity pre-clustered
dense/kernel:0: 50.00% sparsity  (50176/100352)
dense_1/kernel:0: 50.00% sparsity  (640/1280)
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.

c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Perform 16 Clustering with Sparsity Awareness on the Pruned Model

Note: clustering of 2 is unable to be performed with sparsity awareness due to a ValueError. Clustering Minimum Error

```
stripped_pruned_model =
tf.keras.models.load_model('trained_stripped_pruned_base_model.h5')
```

```
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization
centroid_init = CentroidInitialization.KMEANS_PLUS_PLUS
pruned_clustered_16_model, pruned_stripped_clustered_16_model =
sparsity_aware_clustering(stripped_pruned_model, nc=16,
centroid_init=centroid_init)
pruned_stripped_clustered_16_model.save('pruned_stripped_clustered_16_
model.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file,
so the model was *not* compiled. Compile it manually.
```

```
Epoch 1/3
1688/1688 [==============================] - 23s 13ms/step - loss:
0.0260 - accuracy: 0.9920 - val_loss: 0.0694 - val_accuracy: 0.9817
Epoch 2/3
1688/1688 [==============================] - 29s 17ms/step - loss:
0.0210 - accuracy: 0.9936 - val_loss: 0.0715 - val_accuracy: 0.9782
Epoch 3/3
1688/1688 [==============================] - 24s 14ms/step - loss:
0.0205 - accuracy: 0.9938 - val_loss: 0.0716 - val_accuracy: 0.9815
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you
train or evaluate the model.
```

```
c:\Users\johns\Documents\EngineeringRepositories\ENDG511\lab-env\Lib\
site-packages\keras\src\engine\training.py:3103: UserWarning: You are
saving your model as an HDF5 file via `model.save()`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
test_acc, model_size, executionTime =
evaluate(pruned_clustered_16_model, test_images, test_labels,
'pruned_stripped_clustered_16_model.h5')
print('\nPruned Clustered Model Accuracy:', test_acc*100, '%')
print("Pruned Clustered Model Size: %.2f bytes" % (model_size))
print("Pruned Clustered Inference Time is", executionTime, "s")

print("Model sparsity post-clustered")
print_model_weights_sparsity(pruned_stripped_clustered_16_model)
```

```
313/313 [==============================] - 2s 7ms/step
```

```
Pruned Clustered Model Accuracy: 97.97000288963318 %
Pruned Clustered Model Size: 45731.00 bytes
Pruned Clustered Inference Time is 0.00025818610191345216 s
Model sparsity post-clustered
kernel:0: 64.39% sparsity  (64616/100352)
kernel:0: 53.59% sparsity  (686/1280)
```

Observations Sparsity Awareness Clustering with 16 Clusters

When sparsity awareness clustering is applied, the model achieves an accuracy of 97.97%, model size of 45731 bytes, and an inference time of 258.19us. Additionally, using the function `print_model_weights_sparsity` verifies that the model preserves its sparsity, but does not keep it exactly at 50%, providing 64.39% and 53.59% sparsities instead at the kernels.

For clustering a model without sparsity awareness results in a model with 98.14% accuracy, 61097 bytes, and 209.67us inference time. Yet using the same function to verify the sparsity level shows 0% sparsity indicating sparsity was not preserved.

Lab Summary of Observations

One shot pruning and Iterative pruning did not have considerable differences in the model accuracy, size, and inference time. However it was observed that iterative pruning had slightly better performances in the aspects above, but not a whole lot of margin.

Having more clusters generally improves the model performances in terms of accuracy, but it does increase the model size and the inference time.

Including pruning with low clusters improves accuracy, but it still does not achieve a higher accuracy than the model with many clusters. However, including pruning with many clusters yeilds exceptional results by keeping the accuracy the same, but decreases both the model size and the inference time.

Implementing sparsity awareness clustering after pruning shows a slight decrease in model accuracy in comparison to non-sparsity awareness clustering. However, it shows even more reduction in model size, but a slight increase in the inference time of the model.

Sparsity preserving clustering differs from regular clustering in the sense that it maintains the % of pruning of the model, keeping the values of the weights intact that are close to zero when redistributing the weights during the clustering operations.

Summary Results Shown Below

One Shot 90% Pruning Lab 2

- Pruned Model Accuracy: 95.3499972820282 %
- Pruned Model Size: 80123.00 bytes
- Pruned Inference Time is 9.749836921691894e-05 s

Iterative Pruning with Final Sparsity at 90%

- Pruned Model Accuracy: 95.95999717712402 %
- Pruned Model Size: 77191.00 bytes
- Pruned Inference Time is 8.017294406890869e-05 s

Clustering with 2 Clusters

- Clustered Model Accuracy: 66.75000190734863 %
- Clustered Model Size: 24481.00 bytes
- Clustered Inference Time is 0.00015413177013397217 s

Clustering with 16 Clusters

- Clustered Model Accuracy: 98.00999760627747 %
- Clustered Model Size: 75470.00 bytes
- Clustered Inference Time is 0.00021111106872558595 s

50% Pruning with 2 Clusters

- Pruned Clustered Model Accuracy: 73.94999861717224 %
- Pruned Clustered Model Size: 20002.00 bytes
- Pruned Clustered Inference Time is 0.00014613535404205323 s
- Model sparsity post-clustered
- kernel:0: 0.00% sparsity (0/100352)
- kernel:0: 0.00% sparsity (0/1280)

50% Pruning with 16 Clusters

- Pruned Clustered Model Accuracy: 98.14000129699707 %
- Pruned Clustered Model Size: 61097.00 bytes
- Pruned Clustered Inference Time is 0.0002096734285354614 s
- Model sparsity post-clustered
- kernel:0: 0.00% sparsity (0/100352)
- kernel:0: 0.00% sparsity (0/1280)

Sparsity Awareness Clustering with 50% Pruning

- Pruned Clustered Model Accuracy: 97.97000288963318 %
- Pruned Clustered Model Size: 45731.00 bytes
- Pruned Clustered Inference Time is 0.00025818610191345216 s
- Model sparsity post-clustered
- kernel:0: 64.39% sparsity (64616/100352)
- kernel:0: 53.59% sparsity (686/1280)