EEP 596 Quiz 2 – Week 3, Apr 10 (Wed.) Spring 2024

1. How does the number of parameters change when pruning is applied to a neural network, and what is the reason for this change?

```
[46]: model_base.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dense_1 (Dense)             (None, 12)                6156

=================================================================
Total params: 27,566,412
Trainable params: 12,851,724
Non-trainable params: 14,714,688
```

```python
import tensorflow as tf
import tensorflow_model_optimization as tfmot
import numpy as np

# Load the original pre-trained model
model = tf.keras.models.load_model('vgg16_plant_leaves.h5')

# Define the pruning parameters
pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(initial_sparsity=0.5,
                                                        final_sparsity=0.75,
                                                        begin_step=0,
                                                        end_step=len(train_ds)*5)
pruning_params = {'pruning_schedule': pruning_schedule}

# Apply pruning to the whole model
prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude
pruned_model = prune_low_magnitude(model, **pruning_params)

# Compile the pruned model
pruned_model.compile(optimizer='adam',
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy'])

# Fit the pruned model
pruning_callbacks = [tfmot.sparsity.keras.UpdatePruningStep()]
pruned_model.fit(train_ds, epochs=5, validation_data=test_ds, callbacks=pruning_callbacks)

# After training, remove the pruning wrappers and use this for quantization-aware training
final_model = tfmot.sparsity.keras.strip_pruning(pruned_model)
```

```
pruned_model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         29425183

 prune_low_magnitude_flatten  (None, 25088)            1
  (PruneLowMagnitude)

 prune_low_magnitude_dense (  (None, 512)              25690626
 PruneLowMagnitude)

 prune_low_magnitude_dense_1  (None, 12)               12302
  (PruneLowMagnitude)

=================================================================
Total params: 55,128,112
Trainable params: 12,851,724
Non-trainable params: 42,276,388
_________________________________________________________________
```

Why?

```
final_model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dense_1 (Dense)             (None, 12)                6156

=================================================================
Total params: 27,566,412
Trainable params: 12,851,724
Non-trainable params: 14,714,688
_________________________________________________________________
```

```python
# Calculate the model size in bytes first: number of parameters * 4 (bytes per parameter)
model_size_bytes = model.count_params() * 4

# Convert bytes to megabytes
model_size_mb = model_size_bytes / (1024 * 1024)

print(f'Model Size: {model_size_mb:.2f} MB')
```

```
Model Size: 105.16 MB
Test Accuracy: 0.948888897895813
```

```python
import os

# Compile the stripped model before evaluation
final_model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

# Evaluate the pruned model
pruned_accuracy = final_model.evaluate(test_ds)[1]
print(f'Pruned Model Test Accuracy: {pruned_accuracy}')

# After pruning and fine-tuning, convert the model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(final_model)
tflite_model = converter.convert()

# Save the TensorFlow Lite model
with open('vgg16_plant_leaves_pruned.tflite', 'wb') as f:
    f.write(tflite_model)

# Calculate the size of the TensorFlow Lite model
tflite_model_size = os.path.getsize('vgg16_plant_leaves_pruned.tflite') / (1024 * 1024)
print(f'Pruned Model Size (in TFLite format): {tflite_model_size:.2f} MB')
```

```
29/29 [==============================] - 39s 1s/step - loss: 0.2527 - accuracy: 0.9456
Pruned Model Test Accuracy: 0.945555567741394
Pruned Model Size (in TFLite format): 105.17 MB
```

```python
import os
import tensorflow as tf

# Compile the stripped model before evaluation
final_model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

# Evaluate the pruned model
pruned_accuracy = final_model.evaluate(test_ds)[1]
print(f'Pruned Model Test Accuracy: {pruned_accuracy}')

# After pruning and fine-tuning, convert the model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(final_model)
converter.optimizations = [tf.lite.Optimize.EXPERIMENTAL_SPARSITY]  # Enable sparsity optimization

# Convert the model
tflite_model = converter.convert()

# Save the TensorFlow Lite model
tflite_model_path = 'vgg16_plant_leaves_pruned.tflite'
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)

# Calculate the size of the TensorFlow Lite model
tflite_model_size = os.path.getsize(tflite_model_path) / (1024 * 1024)
print(f'Pruned Model Size (in TFLite format): {tflite_model_size:.2f} MB')
```

```
29/29 [==============================] - 38s 1s/step - loss: 0.2527 - accuracy: 0.9456
Pruned Model Test Accuracy: 0.945555567741394
Pruned Model Size (in TFLite format): 38.97 MB
```

2. Which quantization methods does TensorFlow Lite support?

TensorFlow Lite supports several quantization methods, including post-training quantization (Dynamic range quantization, full integer quantization, float16 quantization) and quantization-aware training.

3. Does TensorFlow Lite support structured or unstructured pruning, and what is the key difference between these methods?

   TensorFlow Lite primarily supports unstructured pruning, which involves individually removing weights based on their magnitude or importance, as opposed to structured pruning, which removes entire filters or channels.

4. What distinguishes feature-based knowledge distillation from relation-based knowledge distillation?

- Feature-based knowledge distillation focuses on transferring knowledge by matching features (such as intermediate layer activations) between the teacher and student networks.

- In contrast, relation-based knowledge distillation aims to transfer relational knowledge by aligning the relationships (e.g., pairwise distances - Euclidean distance or angles between data points – Cosine Similarity) between the layers or outputs of the teacher and student networks.