Model Optimization 101

Sayak Paul (@RisingSayak)

$whoami

- I call `model.fit()` @ [PyImageSearch]
- Netflix Nerd 👀
- My coordinates are here - https://sayak.dev/

Acknowledgement

- ML-GDEs
- Khanh LeViet from Google

Ideal audience

- ML Developers having worked on image models (in Keras).
- Engineers looking for ways to optimize models for deployment purposes.

What are we up to today?

- What is model optimization?
- Why should we care about it?
- Different areas for a model to optimize
- Mapping areas to optimization techniques
  - Quantization
  - Pruning
- Considerations
- Further directions and QA

About the conventions used

- tf - tensorflow
- tfmot & MOT - tensorflow_model_optimization_toolkit
- TF Lite - TensorFlow Lite

What is model optimization?

Model optimization can be an umbrella term for:

- Reducing model size

What is model optimization?

Model optimization can be an umbrella term for:

- Reducing model size
- Speeding up inference time

What is model optimization?

Model optimization can be an umbrella term for:

- Reducing model size
- Speeding up inference time
- Reducing the power usage to run a model

Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.

Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.
- Consider deploying your models to Raspberry Pis, Mobile Phones, Microcontrollers...

Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.
- Consider deploying your models to Raspberry Pis, Mobile Phones, Microcontrollers...
- Moreover, heavier models tend to have ***latency***.

Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.
- Consider deploying your models to Raspberry Pis, Mobile Phones, Microcontrollers...
- Moreover, heavier models tend to have ***latency***.
    - Latency during serving is not suitable for critical applications.

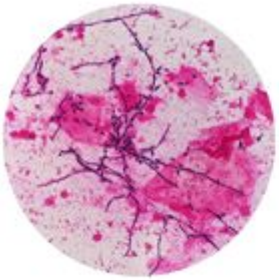
Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.
- Consider deploying your models to Raspberry Pis, Mobile Phones, Microcontrollers...
- Moreover, heavier models tend to have ***latency***.
- Heavier models can affect the infrastructure costs significantly.

Why should we care about it?

- Your heavy but world-class models are likely not suitable for deployment.
- Consider deploying your models to Raspberry Pis, Mobile Phones, Microcontrollers...
- Moreover, heavier models tend to have ***latency***.
- Heavier models can affect the infrastructure costs significantly.
- What if cloud-based model hosting is not an option (cost, network connectivity, privacy concerns, etc.)?

Different areas for a model to optimize



Different areas for a model to optimize

- Reduced numerical precision?
  - A model's parameters and the activations are generally represented in `float32`.

Different areas for a model to optimize

- Reduced numerical precision?
- Are all the operations of a model's graph needed during inference?

Different areas for a model to optimize

- Reduced numerical precision?
- Are all the operations of a model's graph needed during inference?
- Do all the parameters of a model contribute to its performance?

Different areas for a model to optimize

- Reduced numerical precision?
- Are all the operations of a model's graph needed during inference?
- Do all the parameters of a model contribute to its performance?
- Back and forth between GPU and non-GPU kernels (explained here wonderfully).

Mapping areas to optimization techniques

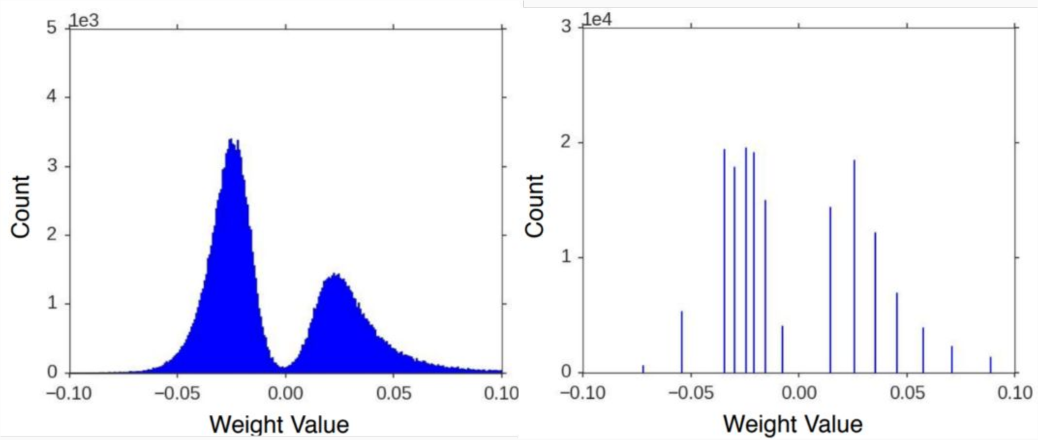
- Reduced numerical precision? - **Quantization**
- Are all the operations of a model's graph needed during inference? - **Layer fusion, constant folding**
- Do all the parameters of a model contribute to its performance? - **Pruning**

Mapping areas to optimization techniques

- Reduced numerical precision? - **Quantization**
- Are all the operations of a model's graph needed during inference? - **Layer fusion, constant folding**
- Do all the parameters of a model contribute to its performance? - **Pruning**

Quantization in deep learning

- Works by reducing the precision of the numbers used to represent a model's parameters and sometimes activations too (`float32` mostly).
- This results in smaller model sizes and faster computations.



[Source]

Quantization mostly come in two flavors

- Post-training quantization
- Quantization-aware training

Post-training quantization

- Happens ***after*** a model is trained.

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]
```

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
```

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')


# Train your model

model.fit(x, y, epochs=50)
```

```python
# Data
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]

# Define and compile your model
model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')


# Train your model

model.fit(x, y, epochs=50)

# Optimize your model

converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

tflite_model = converter.convert()
```

```python
# Optimize your model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
```

- **tf.lite.Optimize.DEFAULT**
- tf.lite.Optimize.OPTIMIZE_FOR_SIZE
- tf.lite.Optimize.OPTIMIZE_FOR_LATENCY

```
# Serialize the TF Lite model
f = open("model.tflite", "wb")
f.write(tflite_model)
f.close
```

Post-training quantization

Different forms of post-training quantization available in TF Lite:

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2-3x speedup, accuracy	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, etc.
Float16 quantization	2x smaller, potential GPU acceleration	CPU/GPU

Check out here: [Post-training quantization]

Notebook demo: https://bit.ly/edge-tpu

Quantization-aware training

- Quantization introduces information loss that might lead to drop in accuracy.

Quantization-aware training

- Quantization introduces information loss that might lead to drop in accuracy.
- What if we could somehow inform the model about this while training it?

Quantization-aware training

- Quantization introduces information loss that might lead to drop in accuracy.
- What if we could somehow inform the model about this while training it?
- Enter quantization-aware training!

```python
# Define the model.
model = tf.keras.Sequential([...])
```

Source

```python
# Define the model.
model = tf.keras.Sequential([...])

# Quantize the entire model.
quantized_model = tfmot.quantization.keras.quantize_model(model)
```

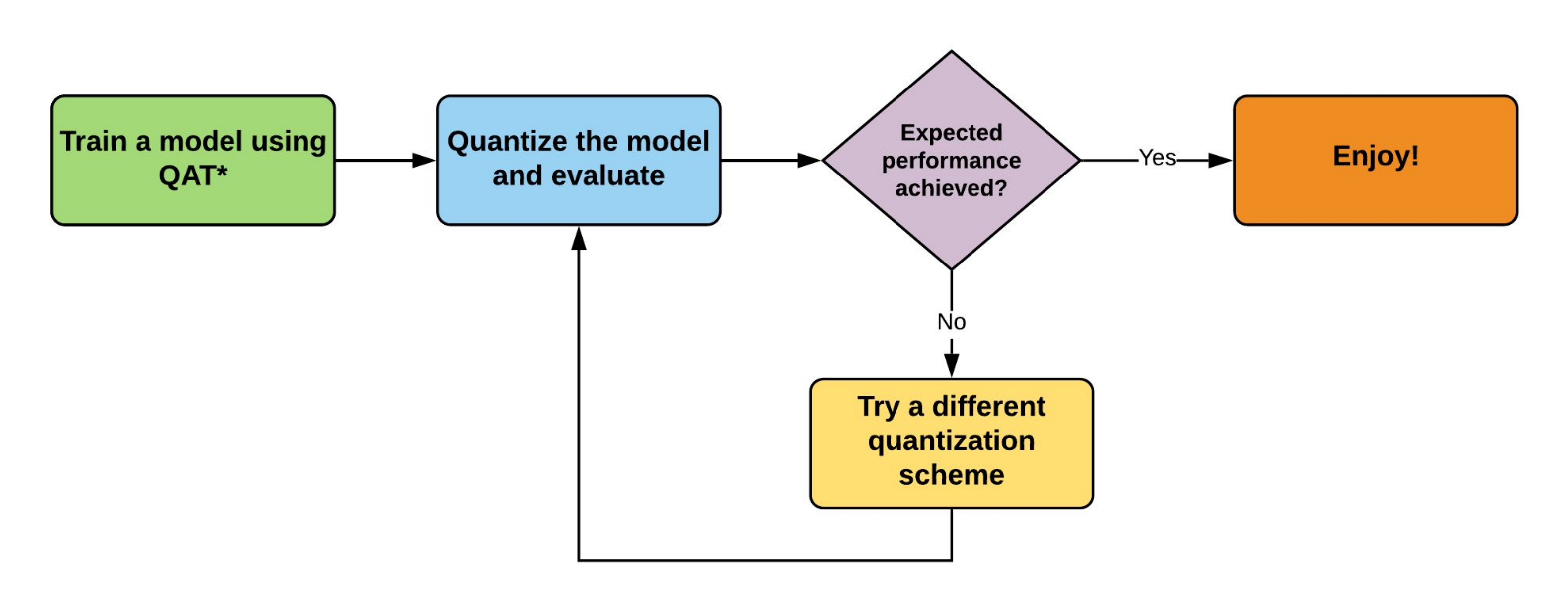
Source

```python
# Define the model.
model = tf.keras.Sequential([...])

# Quantize the entire model.
quantized_model = tfmot.quantization.keras.quantize_model(model)

# Continue with training as usual.
quantized_model.compile(...)
quantized_model.fit(...)
```

Source

Notebook demo:
https://bit.ly/tale-quantization

So, the recipe so far



* one can train parts of a model using QAT as well (see here)

Quantization best practices in TF Lite and MOT

- Performance best practices
- Quantization aware training comprehensive guide

Do all the parameters of a model contribute to its performance?

Switching gears to model's parameters

- **Low-magnitude parameters** usually don't contribute to model's performance and can be discarded.

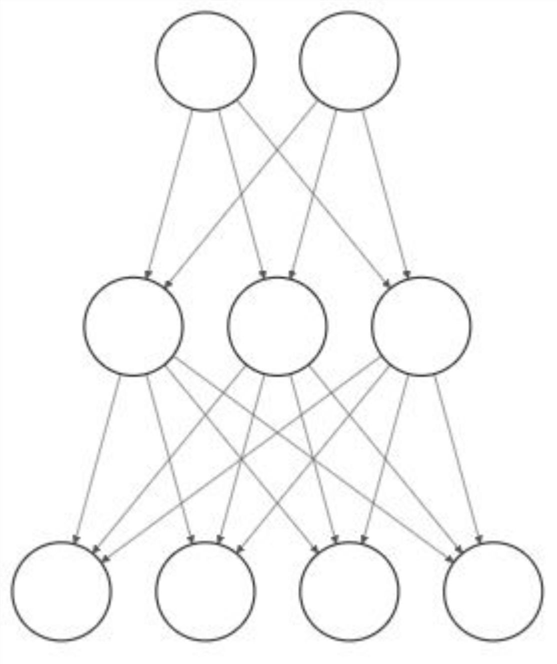
Switching gears to model's parameters

- **Low-magnitude parameters** usually don't contribute to model's performance and can be discarded.
- Zeroing those parameters out is referred to as *Pruning*.

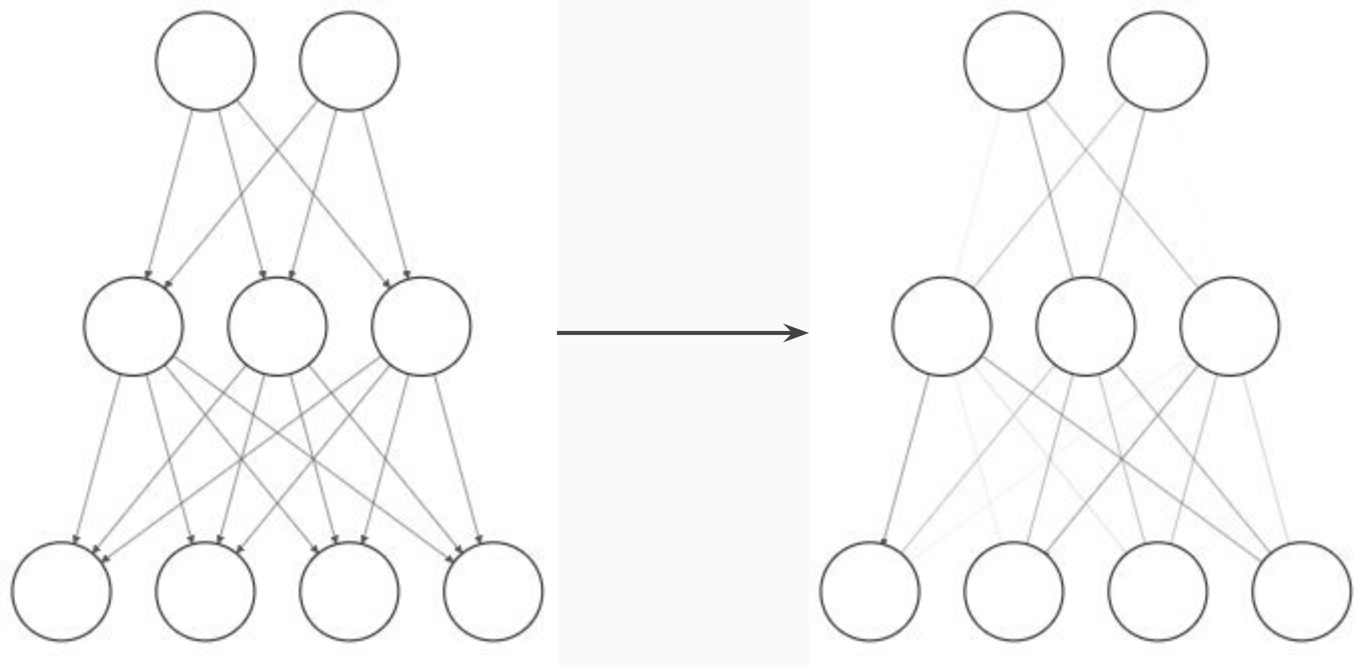
Switching gears to model's parameters

- **Low-magnitude parameters** usually don't contribute to model's performance and can be discarded.
- Zeroing those parameters out is referred to as *Pruning*.
- Can take place *while* and *after* training.

Switching gears to model's parameters



Switching gears to model's parameters



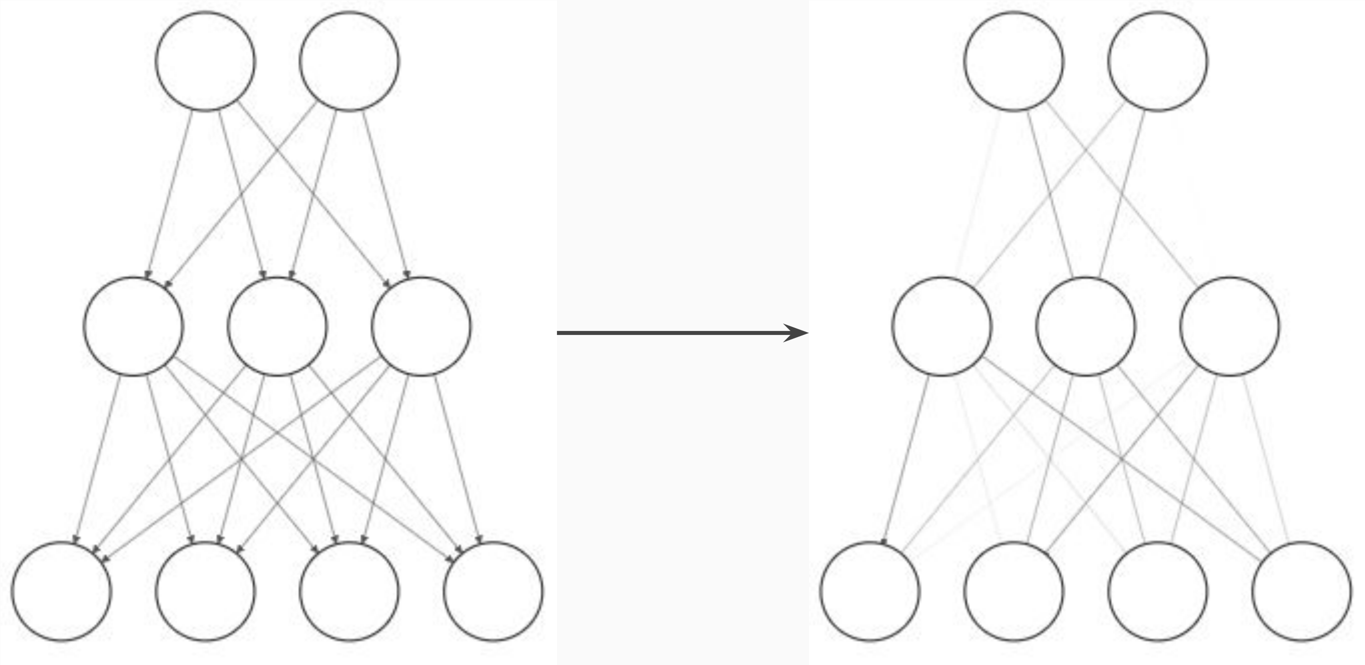
```python
# Define the model.
model = tf.keras.Sequential([...])

# Compile and train the model.
model.compile(...)
model.fit(...)
```

```python
# Define the model.
model = tf.keras.Sequential([...])

# Compile and train the model.
model.compile(...)
model.fit(...)

# Prune, compile, and re-train.
model_for_pruning = prune_low_magnitude(model, **pruning_params)
model_for_pruning.compile(...)
model_for_pruning.fit(...)
```

Get on-boarded: Pruning in Keras example

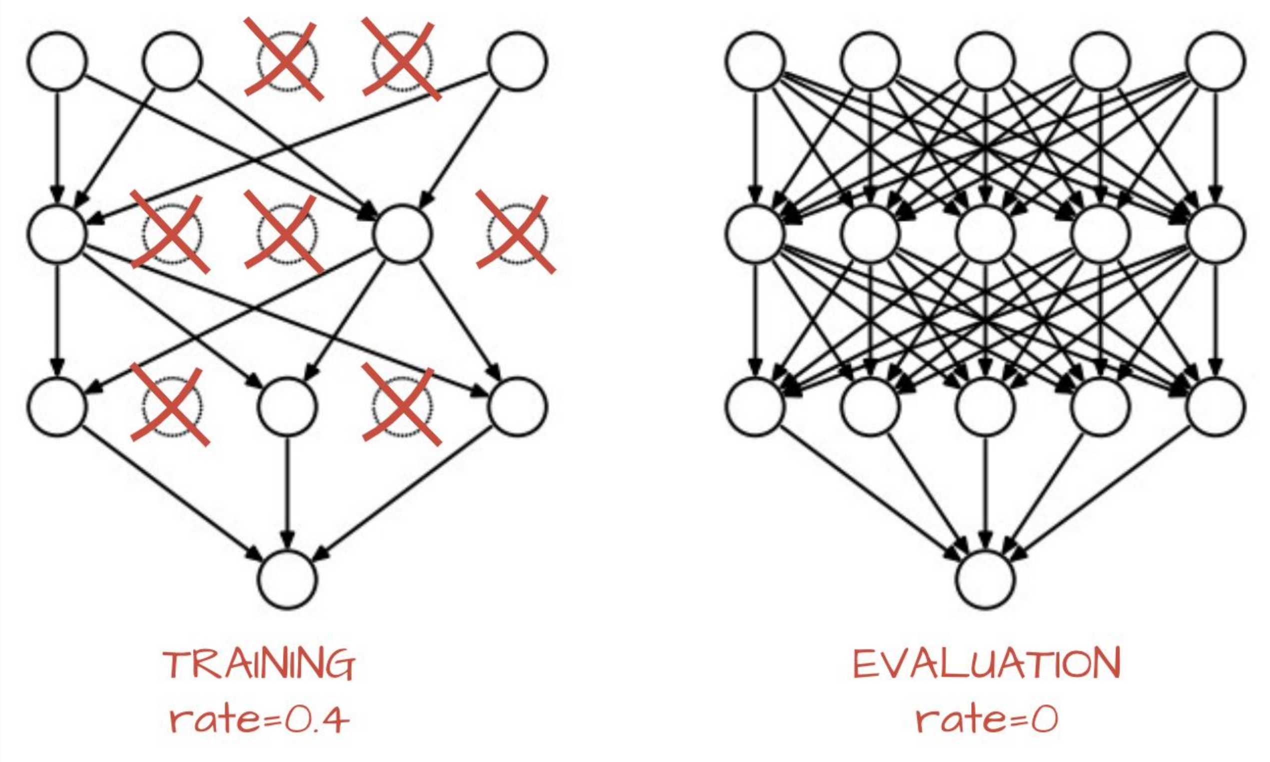
You might be wondering …

This thingy ...




Isn't this Dropout?
imgflip.com

Well...



[Source]

Something to think about

- A pruned version of a heavy network is yielding almost same performance.

Something to think about

- A pruned version of a heavy network is yielding almost same performance.
- So, it's safe to say that the initial network is indeed overparameterized w.r.t given data.

Something to think about

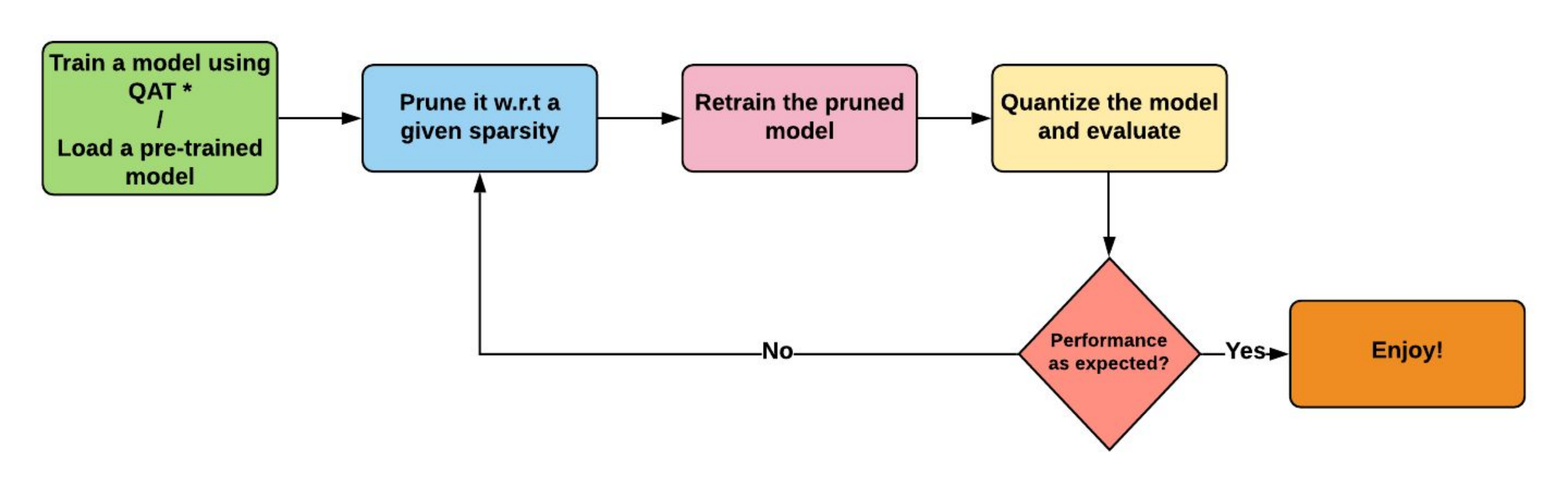
- A pruned version of a heavy network is yielding almost same performance.
- So, it's safe to say that the initial network is indeed overparameterized w.r.t given data.

For the interested ones: [The Lottery Ticket Hypothesis with Jonathan Frankle]

Pruning best practices in MOT

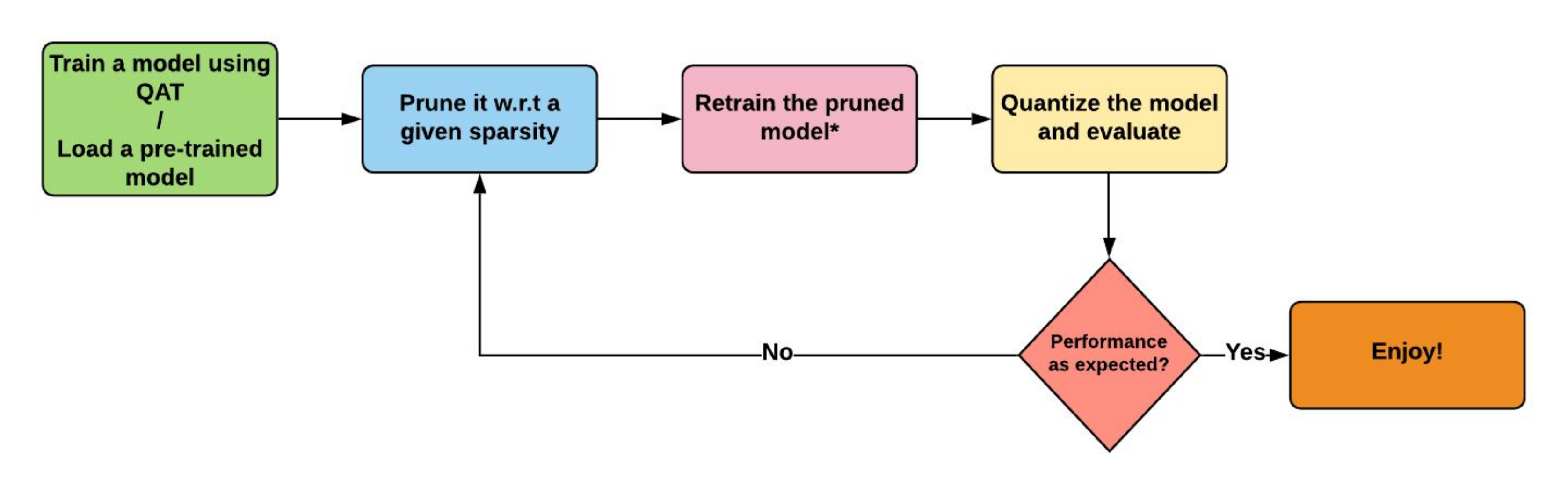
- [Pruning comprehensive guide]

Here's another recipe



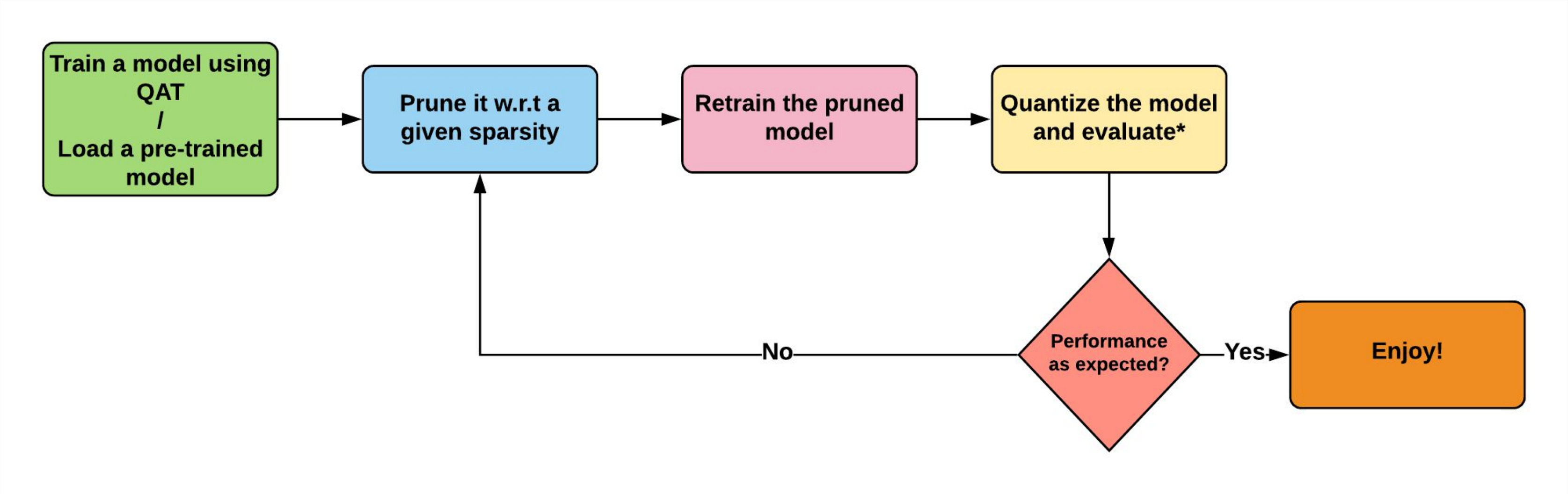
* one can train models using pruning callbacks (see [here]())
one can prune parts of a model (see [here]())

Here's another recipe



* to recover any lost accuracy

Here’s another recipe



* inference will be faster for pruned models with TF Lite in future
(see here)


MAKE YOUR OWN OPTIMIZATION
RECIPES AND EXPERIMENT!
imgflip.com

Resources

- TensorFlow Lite guide
- TensorFlow Model Optimization
- Device-based Models with TensorFlow Lite
- Introduction to TensorFlow Lite
- Awesome TF Lite
- TinyML

Join tflite@tensorflow.org and participate in the discussions!

Some further things to explore

- Movement Pruning*
- Lottery Ticket Hypothesis
- Weight Rewinding (generalization of Lottery Ticket Hypothesis)
- Knowledge Distillation
- Early Exit
- Deep Compression
- Low Rank Approximation

* particularly useful for models in Transfer Learning regime

Deck available here: https://bit.ly/mo-101

Let's get connected on Twitter! I am [@RisingSayak](https://twitter.com/RisingSayak).