Auto-tuning TensorFlow Threading Model for CPU Backend

Niranjan Hasabnis
Intel

HPCaML workshop, 2019.

https://arxiv.org/abs/1812.01665

Introduction

- TensorFlow is a popular deep learning framework from Google.

- Applications in many areas

Image recognition

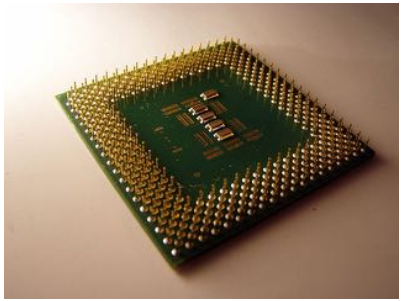
Speech recognition

NLP

MiniGo

Magenta – making art and music

TensorFlow supports



CPU

GPU

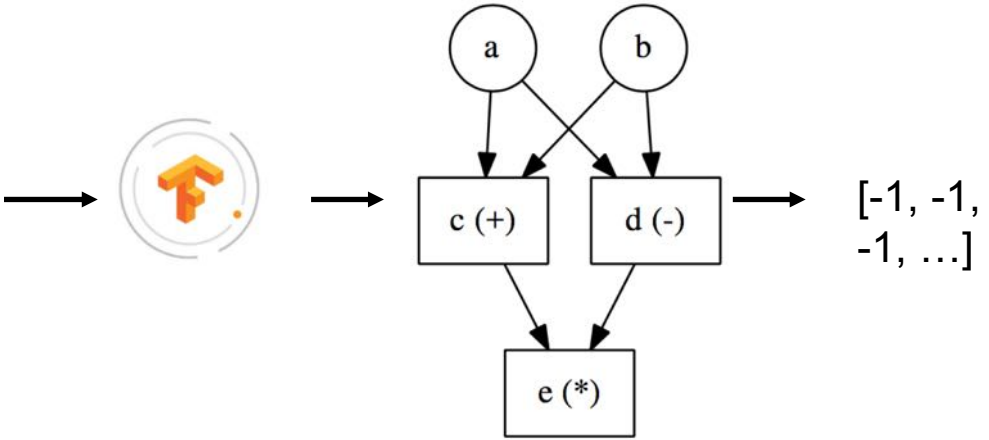
TPU*

Raspberry Pi*

All images taken from public domain or Pexels.com. TPU image taken from cloud.google.com.

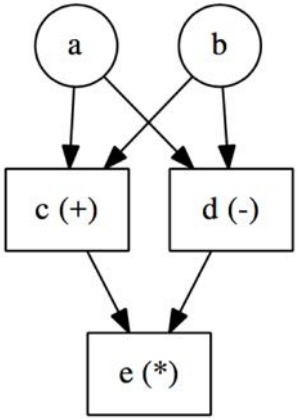
Model execution: example

```
import tensorflow as tf

a = tf.Variable(tf.zeros([100]))
b = tf.Variable(tf.ones([100]))
c = a + b
d = a - b
e = c * d

s = tf.Session()
s.run(tf.initialize_all_variables())

result = s.run([e])
```



Currently, TensorFlow has 2 CPU backends: Eigen (default), and MKL (Intel-optimized)

TensorFlow graph execution

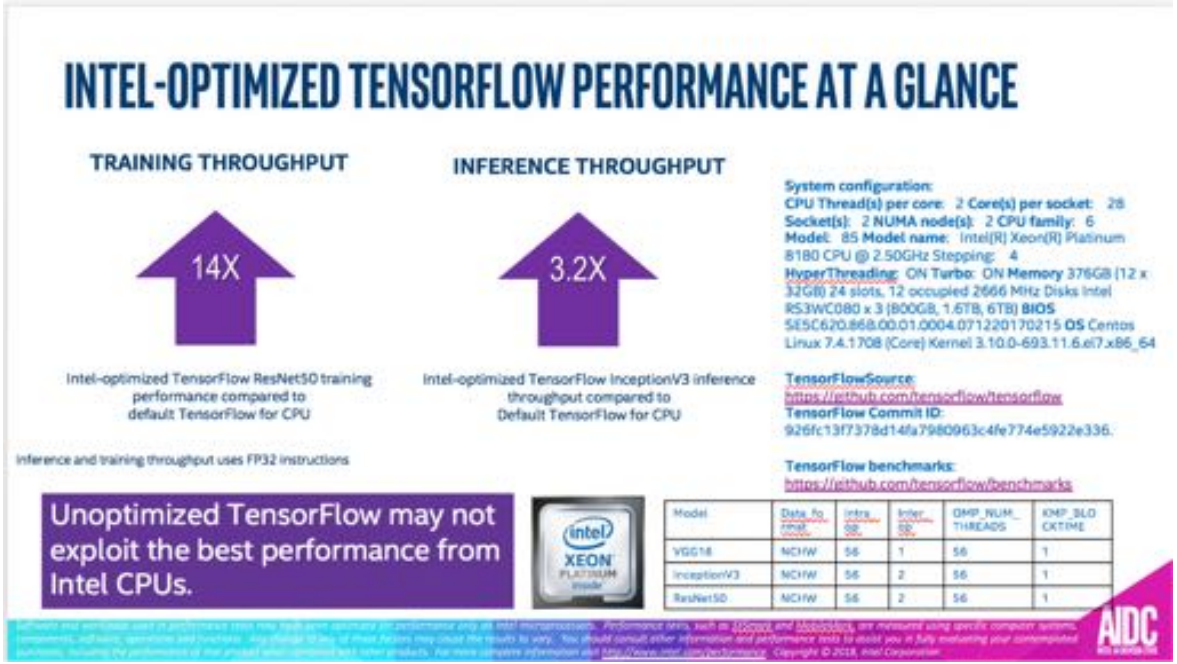


Inter_op	Node execution order
1	a, b, c, d, e
2	[a, b], [c, d], e

- Threading model offers an ability to exploit graph-level parallelism for execution.
- Threading model parameters
  1. inter_op_parallelism_threads
  2. intra_op_parallelism_threads
  3. OMP_NUM_THREADS

In TensorFlow, performance of a model on CPU backend relies on the threading model parameters.

Getting better TensorFlow performance on Intel Xeon CPU (with MKL backend)



Source - https://www.intel.com/content/www/us/en/events/ai/devcon.html

TensorFlow performance guide for CPU devices (applicable to MKL and Eigen backends)

The two configurations listed below are used to optimize CPU performance by adjusting the thread pools.

- `intra_op_parallelism_threads` : Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.
- `inter_op_parallelism_threads` : All ready nodes are scheduled in this pool.

These configurations are set via the `tf.ConfigProto` and passed to `tf.Session` in the `config` attribute as shown in the snippet below. For both configuration options, if they are unset or set to 0, will default to the number of logical CPU cores. Testing has shown that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative optimization is to set the number of threads in both pools equal to the number of physical cores rather than logical cores.

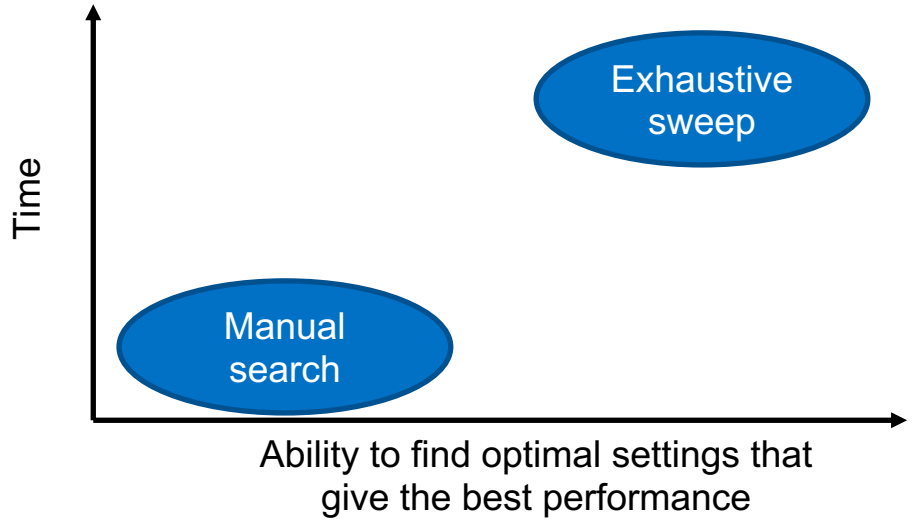
$1M question is …

Q: How easy it is to find parameter values for best performance?

A: Unfortunately, it is not easy!

Image taken from commons.wikimedia.org. Created by author Idea SV

Why it is not easy ..



- Exhaustive sweep does not work for exponentially-growing search space.
- Manual search does not explore search space systematically.

Can there be a tool that can
1) find better parameter values than manual search
2) and find the values quickly?

TensorTuner: contributions

- First attempt to address the problem
- TensorTuner could find better parameter values
  - That deliver 1.5% to 123% (2X) improved performance over the best-known
  - 2X – 10X more efficiently than exhaustive sweep

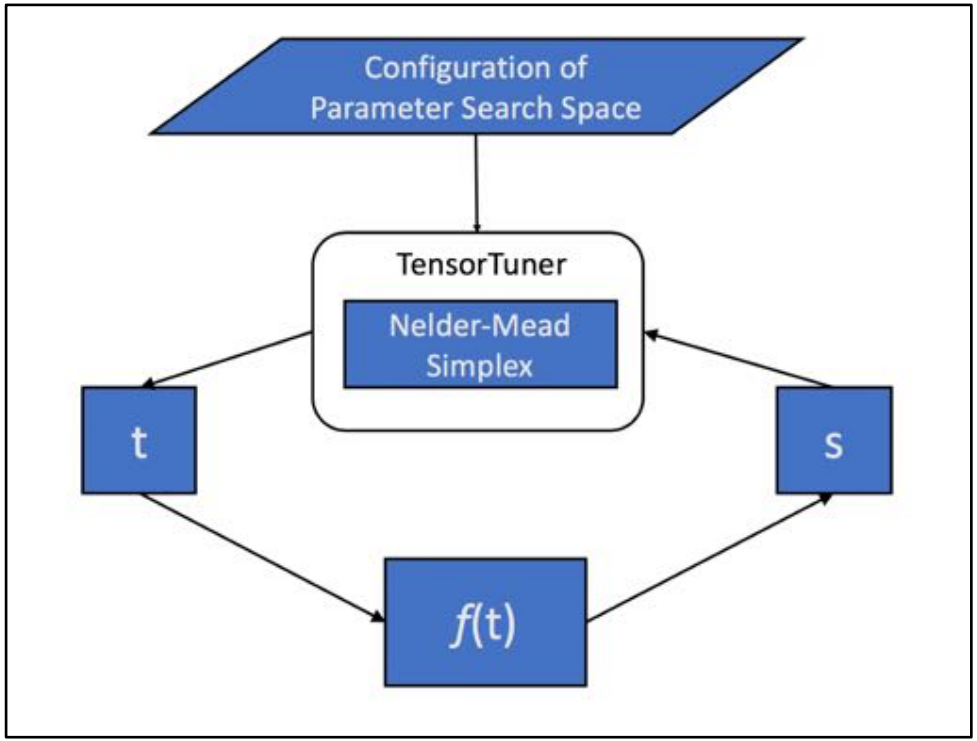
Problem formulation

- Formulated as a **function maximization problem**.
- Performance $\boldsymbol{f}$ can be defined as:

$$s = f_C(\Sigma)$$

- Where
  - C represents set of constants (e.g., input neural network with hyper-parameters, input dataset, hardware, software configuration, etc)
  - s is performance score (e.g., imgs/sec for CNNs)
  - $\boldsymbol{\Sigma}$ is the set of params that we want to tune

Design


Configuration of
Parameter Search Space
TensorTuner
Nelder-Mead
Simplex
t
s
f(t)

Evaluation: criteria

1. Tuning Quality
   - Measures ability of the algorithm to find optimum setting
   - Measured as $\boldsymbol{f}(\boldsymbol{t}_{\text{suggested}})$
   - Compare with $\boldsymbol{f}(\boldsymbol{t}_{\text{best-known}})$
     - comes from Intel AI blogs and
     - running Eigen backend the with default settings
2. Tuning Efficiency
   - Measures ability of the algorithm to converge to optimum quickly
   - Measured as % of the parameter space explored

Evaluation: setup

- Xeon 8180, Cent OS, GCC-6.3, Python-2.7.5
- Eigen backend: TF-1.7 wheel
- MKL backend: built wheel from TF master (sometime in March)
- Tensorflow tf_cnn_benchmarks

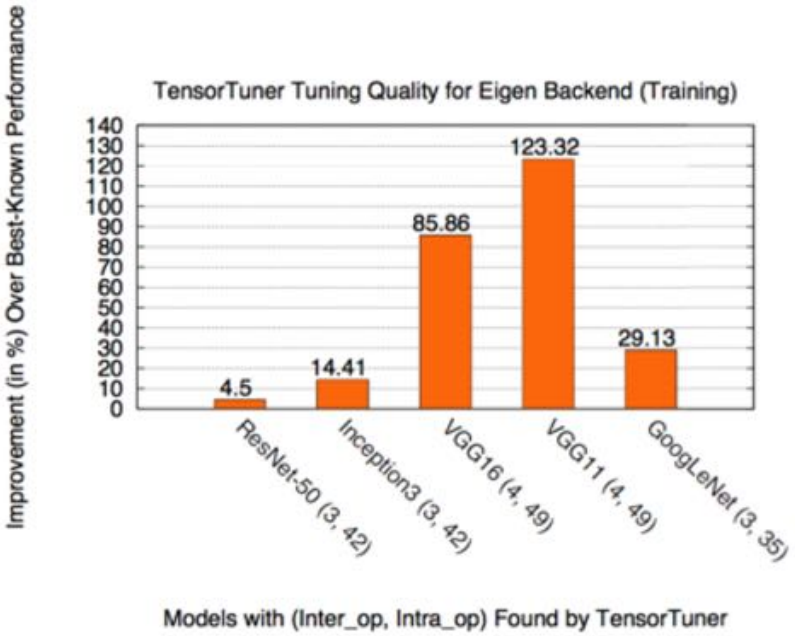
Backend	Model	Batch Size	Data Format
MKL CPU	ResNet-50	128	NCHW
MKL CPU	Inception3	64	NCHW
MKL CPU	VGG16	128	NCHW
MKL CPU	VGG11	128	NCHW
MKL CPU	GoogLeNet	96	NCHW
Eigen CPU	ResNet-50	128	NHWC
Eigen CPU	Inception3	64	NHWC
Eigen CPU	VGG16	128	NHWC
Eigen CPU	VGG11	128	NHWC
Eigen CPU	GoogLeNet	96	NHWC

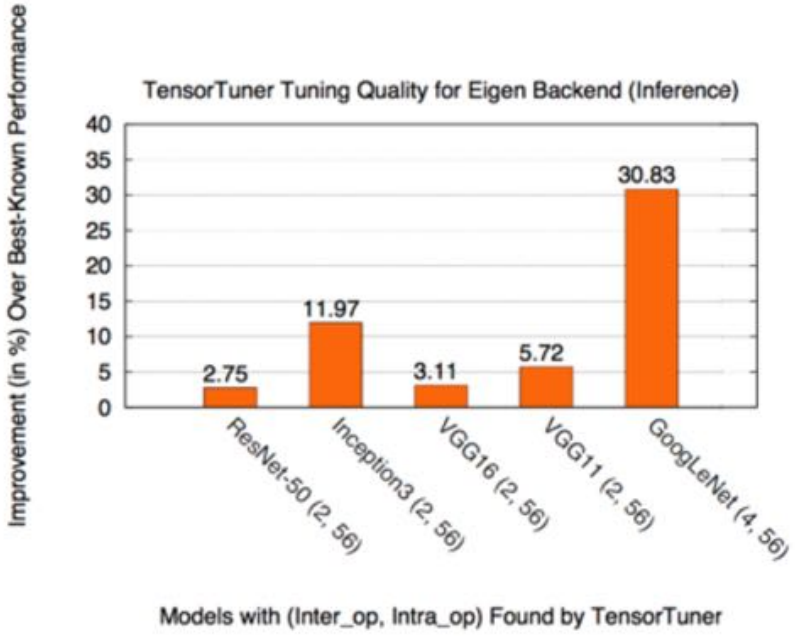
Fig. 5: Models used for evaluation

Backend	inter_op	intra_op	OMP_NUM_THREADS
MKL	[1, 4, 1]	[14, 56, 7]	[14, 56, 7]
Eigen	[1, 4, 1]	[14, 56, 7]	-

Fig. 7: [Lower bound, upper bound, step size] for parameter search

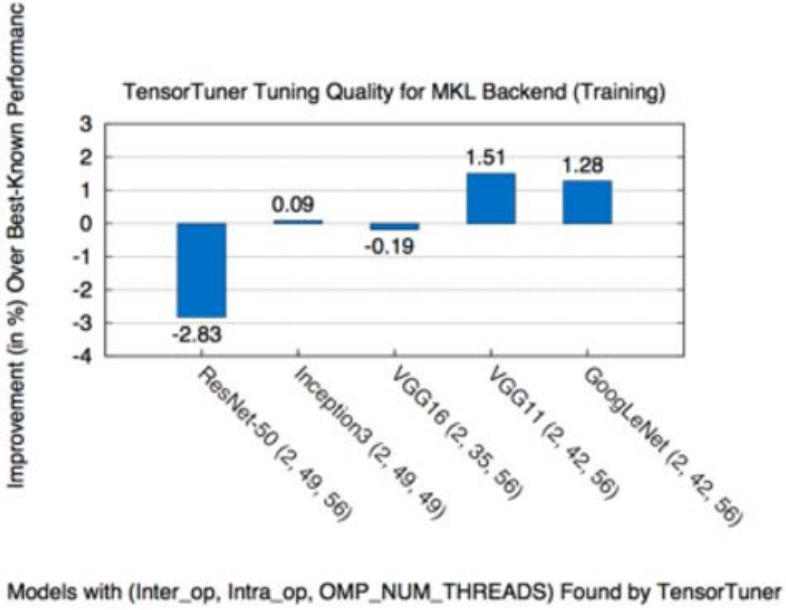
Evaluation: tuning quality with Eigen backend

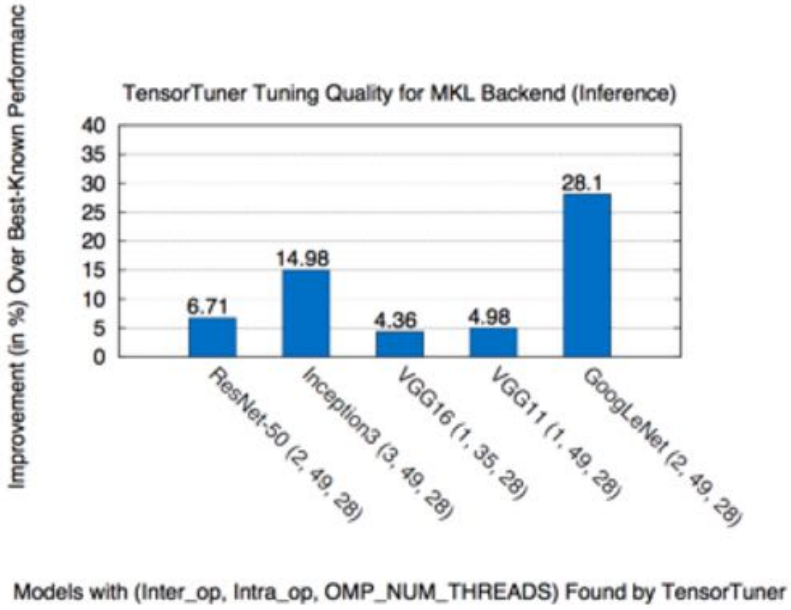




TensorTuner suggested settings perform better than default settings on Eigen backend.

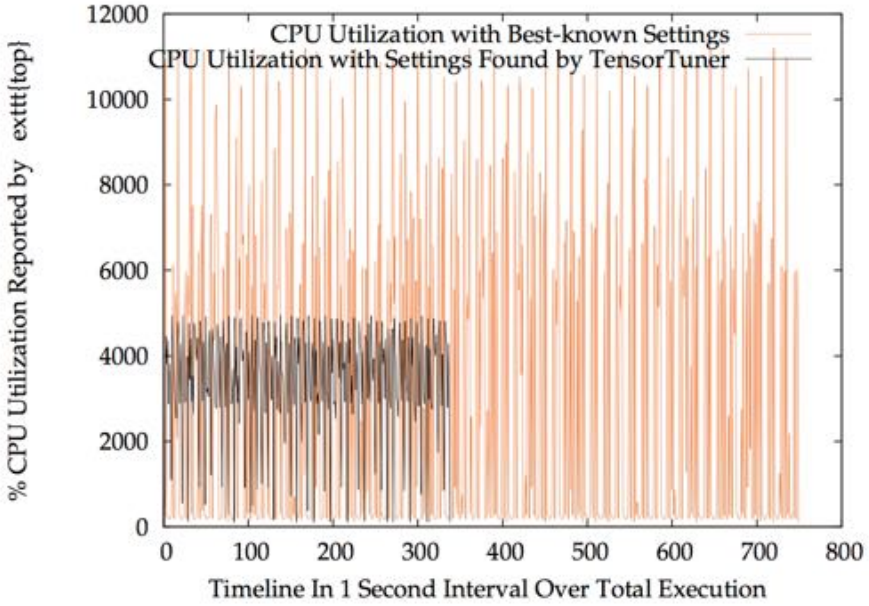
Evaluation: Tuning quality for MKL backend





TensorTuner suggested settings perform better than default settings on MKL backend.

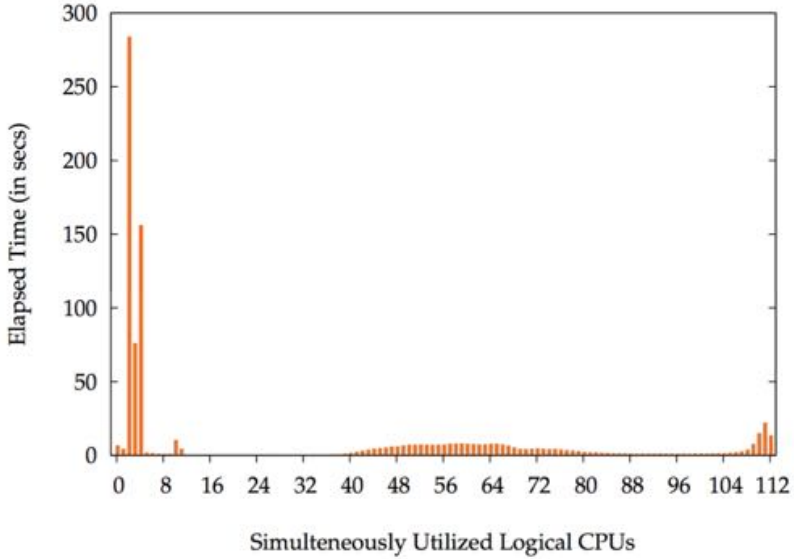
Analysis: 123% better performance with settings found by TensorTuner



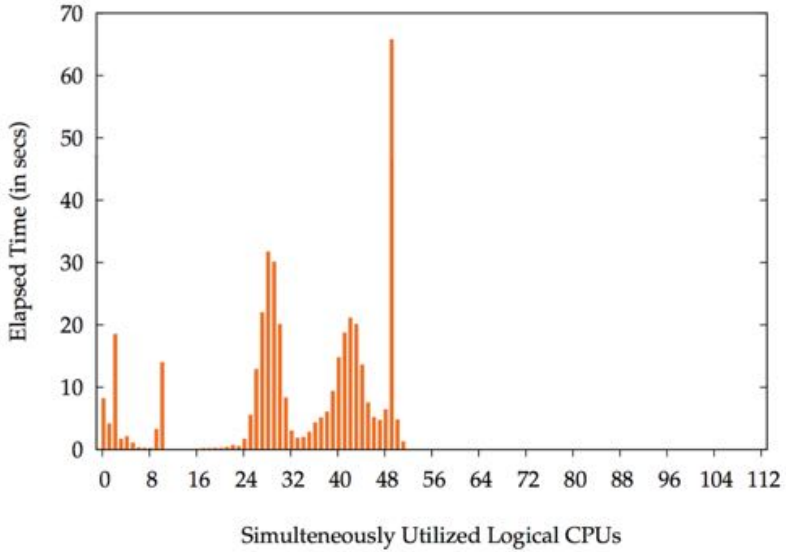
Default settings in TensorFlow (112,112) lead to thread over-subscription issue for VGG11 training case.

TensorTuner suggested settings (4,49) reduced over-subscription issue.

Analysis: 123% better performance with settings found by TensorTuner

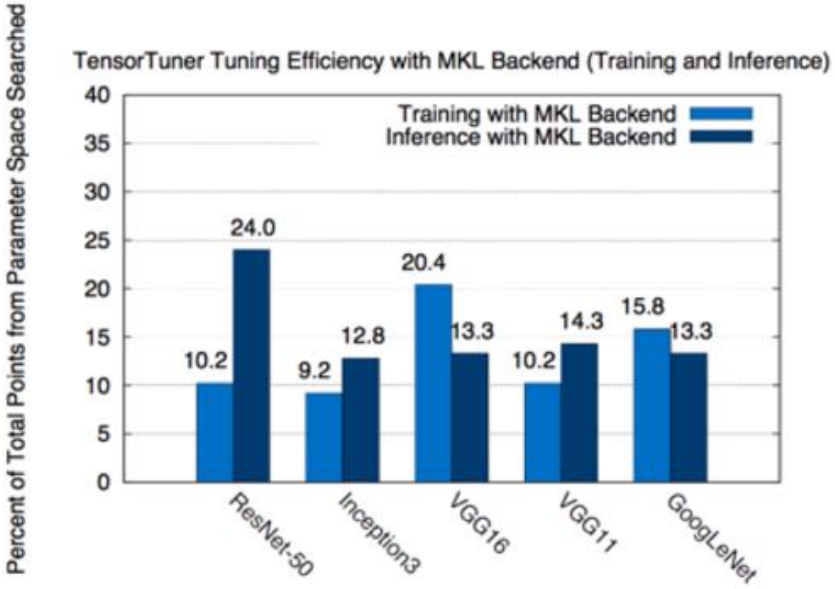


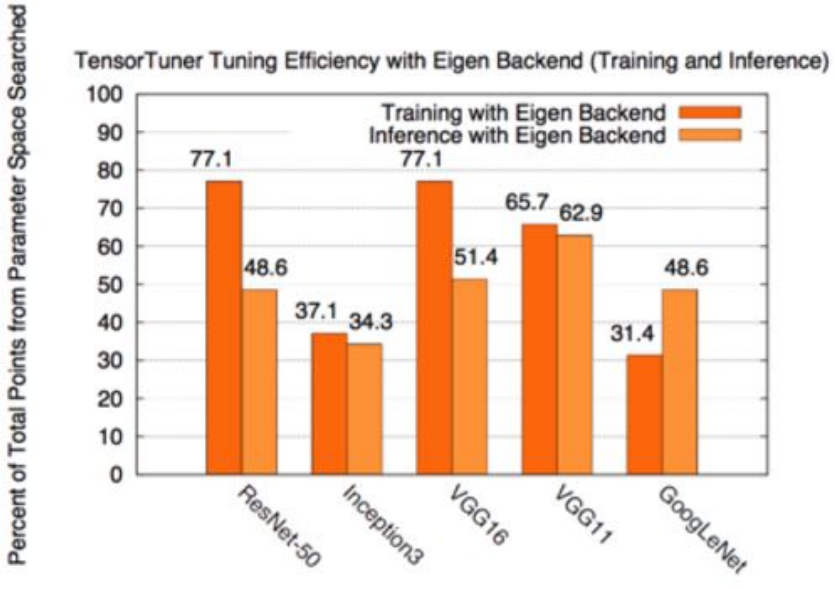
With the default settings



With the settings suggested by TensorTuner

Evaluation: Tuning efficiency





TensorTuner is able to find better-performing setting by pruning large search spaces.

Related work

- **Optimization algorithms**
  - **Gradient-based optimizers** (gradient descent, Newton's method, BFGS method)
  - **Gradient-free optimizers** (Nelder-Mead, Simulated Annealing, Genetic Algorithms)
- **Auto-tuning in HPC**
  - David H Bailey, et al. *Performance tuning of scientific applications*
  - Auto-tuning matrix multiplication
  - In Machine Learning
    - Hyper-parameter tuning: HyperOpt, MOE, AutoWeka, HyperTune
    - Automatically generate efficient kernels (Tensor Comprehension, TVM)

Future work

- Compare Nelder-Mead algorithm with other gradient-free optimization algorithms
- Explore convergence behavior of Nelder-Mead for large number of parameters
- Applicability to wider models/workloads

Conclusion

- Existing approaches for TensorFlow parameter tuning are either expensive or may leave performance on table.

- TensorTuner could suggest better parameter values
  - That improve CPU backend performance from 2% to 123%
  - Efficiently by exploring subset of the search space (2X – 10X more efficiently)

Thank you!