

# Compile and Tune a Vision Transformer Model using HyperParameter Tuner on a Single Node

1. [SageMaker Training Compiler Overview](#)
  - A. [Introduction](#)
2. [Working with the Caltech-256 dataset](#)
  - A. [Installation](#)
  - B. [SageMaker environment](#)
3. [How effective is SageMaker Training Compiler ?](#)
  - A. [SageMaker Training Job](#)
  - B. [Training Setup](#)
  - C. [Experimenting with Native TensorFlow](#)
  - D. [Experimenting with Optimized TensorFlow](#)
  - E. [Wait for tuning jobs to complete](#)
  - F. [Fastest Training Job](#)
4. [Further tuning with SageMaker Training Compiler](#)
  - A. [Wait for tuning jobs to complete](#)
  - B. [Fastest Convergence](#)
5. [Conclusion](#)
6. [Clean up](#)

## SageMaker Training Compiler Overview

SageMaker Training Compiler is a capability of SageMaker that makes hard-to-implement optimizations to reduce training time on GPU instances. The compiler optimizes DL models to accelerate training by more efficiently using SageMaker machine learning (ML) GPU instances. SageMaker Training Compiler is available at no additional charge within SageMaker and can help reduce total billable time as it accelerates training.

SageMaker Training Compiler is integrated into the AWS Deep Learning Containers (DLCs). Using the SageMaker Training Compiler enabled AWS DLCs, you can compile and optimize training jobs on GPU instances with minimal changes to your code. Bring your deep learning models to SageMaker and enable SageMaker Training Compiler to accelerate the speed of your training job on SageMaker ML instances for accelerated computing.

For more information, see [SageMaker Training Compiler](#) (<https://docs.aws.amazon.com/sagemaker/latest/dg/training-compiler.html>) in the *Amazon SageMaker Developer Guide*.

## Introduction

In this demo, you'll use SageMaker Training Compiler and SageMaker Hyperparameter Tuner to speed up training the `vision transformer` model on the `caltech-256` dataset. To get started, we need to set up the environment with a few prerequisite steps, for permissions,

configurations, and so on.

**NOTE:** You can run this demo in SageMaker Studio, SageMaker notebook instances, or your local machine with AWS CLI set up. If using SageMaker Studio or SageMaker notebook instances, make sure you choose one of the TensorFlow-based kernels, Python 3 (TensorFlow x.y Python 3.x CPU Optimized) or `conda_tensorflow_p39` respectively.

**NOTE:** This notebook uses `m1.p3.2xlarge` instances, each with a single GPU. However, it can easily be extended to multiple GPUs on a single node. If you don't have enough quota, see [Request a service quota increase for SageMaker resources](https://docs.aws.amazon.com/sagemaker/latest/dg/regions-quotas.html#service-limit-increase-request-procedure) (<https://docs.aws.amazon.com/sagemaker/latest/dg/regions-quotas.html#service-limit-increase-request-procedure>).

## Development Environment

### Installation

This example notebook requires **SageMaker Python SDK v2.95.0 or later**

```
In [1]: !pip install pip --upgrade
!pip install "sagemaker>=2.95.0" boto3 boto3 awscli --upgrade
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://pip.repos.neuron.amazonaws.com (https://pip.repos.neuron.amazonaws.com)
Requirement already satisfied: pip in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (22.1.2)
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://pip.repos.neuron.amazonaws.com (https://pip.repos.neuron.amazonaws.com)
Requirement already satisfied: sagemaker>=2.95.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (2.96.0)
Requirement already satisfied: boto3 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (1.27.15)
Requirement already satisfied: boto3 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (1.24.15)
Requirement already satisfied: awscli in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (1.25.15)
Requirement already satisfied: protobuf<1.0,>=0.1.5 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (0.1.5)
Requirement already satisfied: numpy<2.0,>=1.9.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (1.20.3)
Requirement already satisfied: importlib-metadata<5.0,>=1.4.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (1.7.0)
Requirement already satisfied: pandas in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (1.3.4)
Requirement already satisfied: google-pasta in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (0.2.0)
Requirement already satisfied: packaging>=20.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (21.0)
Requirement already satisfied: smdebug-rulesconfig==1.0.1 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (1.0.1)
Requirement already satisfied: protobuf<4.0,>=3.1 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (3.19.1)
Requirement already satisfied: pathos in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (0.2.8)
Requirement already satisfied: attrs==20.3.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.95.0) (20.3.0)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto3) (0.10.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto3) (1.26.8)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto
```

```

core) (2.8.2)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto3) (0.6.0)
Requirement already satisfied: docutils<0.17,>=0.10 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (0.15.2)
Requirement already satisfied: rsa<4.8,>=3.1.2 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (4.7.2)
Requirement already satisfied: PyYAML<5.5,>=3.10 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (5.4.1)
Requirement already satisfied: colorama<0.4.5,>=0.2.5 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (0.4.3)
Requirement already satisfied: zipp>=0.5 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from importlib-metadata<5.0,>=1.4.0->sagemaker>=2.95.0) (3.6.0)
Requirement already satisfied: pyparsing>=2.0.2 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from packaging>=20.0->sagemaker>=2.95.0) (3.0.6)
Requirement already satisfied: six in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from protobuf3-to-dict<1.0,>=0.1.5->sagemaker>=2.95.0) (1.16.0)
Requirement already satisfied: pyasn1>=0.1.3 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from rsa<4.8,>=3.1.2->awscli) (0.4.8)
Requirement already satisfied: pytz>=2017.3 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pandas->sagemaker>=2.95.0) (2021.3)
Requirement already satisfied: dill>=0.3.4 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=2.95.0) (0.3.4)
Requirement already satisfied: multiprocessing>=0.70.12 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=2.95.0) (0.70.12.2)
Requirement already satisfied: ppft>=1.6.6.4 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=2.95.0) (1.6.6.4)
Requirement already satisfied: pox>=0.3.0 in /home/ec2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=2.95.0) (0.3.0)

```

```

In [2]: import botocore
import boto3
import sagemaker

print(f"botocore: {botocore.__version__}")
print(f"boto3: {boto3.__version__}")
print(f"sagemaker: {sagemaker.__version__}")

```

```

botocore: 1.27.15
boto3: 1.24.15
sagemaker: 2.96.0

```

## SageMaker environment

```
In [3]: import sagemaker

sess = sagemaker.Session()

# SageMaker session bucket -> used for uploading data, models and logs
# SageMaker will automatically create this bucket if it does not exist
sagemaker_session_bucket = None
if sagemaker_session_bucket is None and sess is not None:
    # set to default bucket if a bucket name is not given
    sagemaker_session_bucket = sess.default_bucket()

role = sagemaker.get_execution_role()
sess = sagemaker.Session(default_bucket=sagemaker_session_bucket)

print(f"sagemaker role arn: {role}")
print(f"sagemaker bucket: {sagemaker_session_bucket}")
print(f"sagemaker session region: {sess.boto_region_name}")
```

```
sagemaker role arn: arn:aws:iam::875423407011:role/SageMakerRole
sagemaker bucket: sagemaker-us-west-2-875423407011
sagemaker session region: us-west-2
```

## Working with the Caltech-256 dataset

We have hosted the [Caltech-256 \(https://authors.library.caltech.edu/7694/\)](https://authors.library.caltech.edu/7694/) dataset in S3 in us-west-2. We will transfer this dataset to your account and region for use with SageMaker Training.

The dataset consists of JPEG images organized into directories with each directory representing an object category.

```
In [4]: import os

source = "s3://sagemaker-sample-files/datasets/image/caltech-256/256_Object
destn = f"s3://{sagemaker_session_bucket}/caltech-256"
local = "caltech-256"

os.system(f"aws s3 sync {source} {local}")
os.system(f"aws s3 sync {local} {destn}")
```

Out[4]: 0

## How effective is SageMaker Training Compiler ?

The effectiveness of SageMaker Training Compiler depends on the model architecture, model size, input shape, and the training loop. Please refer to our [Best Practices \(https://docs.aws.amazon.com/sagemaker/latest/dg/training-compiler-tips-pitfalls.html\)](https://docs.aws.amazon.com/sagemaker/latest/dg/training-compiler-tips-pitfalls.html) documentation to understand how to get the most out of your training job using SageMaker Training Compiler. In this section, we will compare and contrast a training job with and without SageMaker Training Compiler.

## SageMaker Training Job

To create a SageMaker training job, we use a `TensorFlow` estimator. Using the estimator, you can define which training script should SageMaker use through `entry_point`, which `instance_type` to use for training, which `hyperparameters` to pass, and so on.

When a SageMaker training job starts, SageMaker takes care of starting and managing all the required machine learning instances, picks up the `TensorFlow` Deep Learning Container, uploads your training script, and downloads the data from `sagemaker_session_bucket` into the container at `/opt/ml/input/data`.

In the following section, you learn how to set up two versions of the SageMaker `TensorFlow` estimator, a native one without the compiler and an optimized one with the compiler.

## Training Setup

In this section, we set our hyperparameters to a naive first guess. Notice the low value for `EPOCHS` - this is because we are just experimenting with our hyperparameters to find the best setting that will lead to the fastest training. The effectiveness of SageMaker Training Compiler is often apparent within the first few steps. In the example below we will inspect the speed of the training job after every epoch.

```
In [5]: EPOCHS = 3
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4
```

## Experimenting with Native TensorFlow

We attempt to find the largest `BATCH_SIZE` that can fit into the memory of an `ml.p3.2xlarge` instance. This will consequently give us the fastest training speed.

```
In [6]: from sagemaker.tensorflow import TensorFlow

estimator_args = dict(
    source_dir="scripts",
    entry_point="vit.py",
    model_dir=False,
    instance_type="ml.p3.2xlarge",
    instance_count=1,
    framework_version="2.9.1",
    py_version="py39",
    debugger_hook_config=None,
    disable_profiler=True,
    max_run=60 * 20, # 20 minutes
    role=role,
)

# Configure the training job
native_estimator = TensorFlow(
    hyperparameters={
        "EPOCHS": EPOCHS,
        "LEARNING_RATE": LEARNING_RATE,
        "WEIGHT_DECAY": WEIGHT_DECAY,
    },
    **estimator_args,
)
```

## SageMaker Hyperparameter Tuning Job

We use the `sagemaker.tuner.HyperparameterTuner` object to define a Hyperparameter Tuning Job. It will import the training job configuration specified in the `estimator`. We additionally specify some `metric_definitions` to extract training metrics from the training logs. From these `metric_definitions` we select a single metric as the `objective_metric_name` and configure the tuning job to Minimize or Maximize it. We further provide a constrained search space through the `hyperparameter_ranges` argument.

We can limit the number of training jobs spawned concurrently in the `max_parallel_jobs` argument and limit the total number of training jobs spawned in the `max_jobs` argument.

For more information regarding SageMaker Hyperparameter Tuner refer to [the documentation\(\)](#).

In the example below, we are trying to find the best batch size between 32 and 80 that will result in the smallest possible epoch latency, by launching 40 training jobs, 10 at a time. The range for batch sizes is our best guess. You can always [reuse and restart a tuning job with an extended range](https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-warm-start.html) (<https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-warm-start.html>).

```
In [7]: from sagemaker.tuner import HyperparameterTuner, IntegerParameter

tuner_args = dict(
    objective_metric_name="training_latency_per_epoch",
    objective_type="Minimize",
    metric_definitions=[
        {"Name": "training_loss", "Regex": "loss: ([0-9.]*)"},
        {"Name": "training_accuracy", "Regex": "accuracy: ([0-9.]*)"},
        {"Name": "training_latency_per_epoch", "Regex": "- ([0-9.]*)s/epoch"},
        {"Name": "training_avg_latency_per_step", "Regex": "- ([0-9.]*)ms/"},
        {"Name": "training_avg_latency_per_step", "Regex": "- ([0-9.]*)s/s"}
    ],
    max_jobs=40,
    max_parallel_jobs=10,
    early_stopping_type="Auto",
)

# Define a Hyperparameter Tuning Job
native_tuner = HyperparameterTuner(
    estimator=native_estimator,
    hyperparameter_ranges={
        "BATCH_SIZE": IntegerParameter(32, 80, "Linear"),
    },
    base_tuning_job_name="native-tf29-vit",
    **tuner_args
)

# Start the tuning job with the specified input data
native_tuner.fit(inputs=destn, wait=False)

# Save the name of the tuning job
native_tuner.latest_tuning_job.name
```

No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config

```
Out[7]: 'native-tf29-vit-220623-1523'
```

**Tip:** You can reduce the cost of tuning by restricting the batch size to be multiple of 8. Refer to Nvidia's article on the [significance of the number 8 \(https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/\)](https://developer.nvidia.com/blog/optimizing-gpu-performance-tensor-cores/) when training with Automatic Mixed Precision.

```
from sagemaker.tuner import CategoricalParameter
hyperparameter_ranges={
    "BATCH_SIZE": CategoricalParameter(list(range(32, 80, 8))),
}
```

This can restrict the search space to just 6 training jobs as opposed to 40 !

## Experimenting with Optimized TensorFlow



Compilation through SageMaker Training Compiler changes the memory footprint of the model. Most commonly, this manifests as a reduction in memory utilization and a consequent increase in the largest batch size that can fit on the GPU. In the example below we will find the new batch size with SageMaker Training Compiler enabled and the resultant latency per epoch.

**Note:** We recommend you to turn the SageMaker Debugger's profiling and debugging tools off when you use compilation to avoid additional overheads.

```
In [8]: from sagemaker.tensorflow import TensorFlow, TrainingCompilerConfig

# Configure the training job
optimized_estimator = TensorFlow(
    hyperparameters={
        "EPOCHS": EPOCHS,
        "LEARNING_RATE": LEARNING_RATE,
        "WEIGHT_DECAY": WEIGHT_DECAY,
    },
    compiler_config=TrainingCompilerConfig(),
    **estimator_args,
)
```

```
In [9]: from sagemaker.tuner import HyperparameterTuner, IntegerParameter

# Define the tuning job
optimized_tuner = HyperparameterTuner(
    estimator=optimized_estimator,
    hyperparameter_ranges={"BATCH_SIZE": IntegerParameter(20, 60, "Linear")},
    base_tuning_job_name="optimized-tf29-vit",
    **tuner_args,
)

# Start the tuning job with the specified input data
optimized_tuner.fit(inputs=destn, wait=False)

# Save the name of the tuning job
optimized_tuner.latest_tuning_job.name
```

No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config

```
Out[9]: 'optimized-tf29-vit-220623-1523'
```

## Wait for tuning jobs to complete

The tuning jobs described above typically take around 50 mins to complete

**Note:** If the tuner object is no longer available due to a kernel break or refresh, you need to directly use the training job name and manually attach the tuning job to a new tuner. For example:

```
native_tuner = HyperparameterTuner.attach("<your_tuning_job_name>")
```

```
In [10]: native_tuner.wait()
         optimized_tuner.wait()
```

.....  
!

```
In [21]: from sagemaker.tuner import HyperparameterTuner

         native_tuner = HyperparameterTuner.attach(native_tuner.latest_tuning_job.name)
         optimized_tuner = HyperparameterTuner.attach(optimized_tuner.latest_tuning_job.name)

         native_tuner.wait()
         optimized_tuner.wait()
```

!  
!

## Fastest Training Job

Let us collate and analyze the results from the tuning jobs. The tuner provides the results as a Pandas dataframe. We combine the results from both the tuners, sort them according to the epoch latency and display the top 5 results.

```
In [22]: import pandas as pd

pd.set_option("display.max_rows", None, "display.max_columns", None)

# Collect the results from the tuners
native_results = native_tuner.analytics().dataframe()
optimized_results = optimized_tuner.analytics().dataframe()

# Sort results according to Epoch Latency
native_results = native_results.sort_values(
    ["FinalObjectiveValue", "BATCH_SIZE"], ascending=[True, False]
)
optimized_results = optimized_results.sort_values(
    ["FinalObjectiveValue", "BATCH_SIZE"], ascending=[True, False]
)

# Combine the top N results for viewing
N = 3
results = pd.concat([native_results.head(N), optimized_results.head(N)])
results
```

Out[22]:

	BATCH_SIZE	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	Trainin
31	77.0	native-tf29-vit-220616-1656-009-b7ed5852	Completed	113.0	2022-06-16 16:59:06+00:00	17:1:
22	75.0	native-tf29-vit-220616-1656-018-f5cd7d6e	Completed	113.0	2022-06-16 17:14:42+00:00	17:2:
21	74.0	native-tf29-vit-220616-1656-019-3c4788ad	Completed	114.0	2022-06-16 17:15:09+00:00	17:2:
30	56.0	optimized-tf29-vit-220616-1656-010-6aa056a0	Completed	64.0	2022-06-16 16:59:33+00:00	17:1:
27	48.0	optimized-tf29-vit-220616-1656-013-f9746c55	Completed	66.0	2022-06-16 17:11:40+00:00	17:2:
24	51.0	optimized-tf29-vit-220616-1656-016-2d1066f8	Completed	69.0	2022-06-16 17:11:57+00:00	17:2:

```
In [23]: # Calculating potential percentage Savings from Training Compiler
difference = (
    native_results.iloc[0]["FinalObjectiveValue"] - optimized_results.iloc[0]
)
percentage = difference * 100 / native_results.iloc[0]["FinalObjectiveValue"]
f"With the SageMaker Training Compiler the epoch latency is {percentage:.1f}%"
```

Out[23]: 'With the SageMaker Training Compiler the epoch latency is 38.9% lower meaning the training job could be upto 38.9% faster!'

## Further tuning with SageMaker Training Compiler

Now that we have the fastest batch size and compiler configuration, we need to tune the associated hyperparameters to get the fastest convergence.

**Remember**  $\text{Total\_Training\_Time} \approx \text{Latency\_per\_epoch} * \text{Number\_of\_epochs}$

First, we tuned to reduce the `Latency_per_epoch`. Now we will tune to reduce the number of epochs required for convergence. Since, hyperparameters that directly affect convergence (like learning rate, weight decay, learning schedule, etc.) are dependent on batch size, we decouple the 2 steps as described.

We now train for a higher number of epochs since we are testing the speed of convergence. Ideally, you should tune learning rate and weight decay to minimize validation loss, but for the sake of example let's minimize the training loss.

```
In [14]: EPOCHS = 10
         BATCH_SIZE = 56

         estimator_args["max_run"] = 60 * 60 # 60 minutes
         tuner_args["objective_metric_name"] = "training_loss"
```

```
In [15]: from sagemaker.tensorflow import TensorFlow, TrainingCompilerConfig

         # Configure the training job
         convergence_estimator = TensorFlow(
             hyperparameters={
                 "EPOCHS": EPOCHS,
                 "BATCH_SIZE": BATCH_SIZE,
             },
             compiler_config=TrainingCompilerConfig(),
             **estimator_args,
         )
```

```
In [16]: from sagemaker.tuner import HyperparameterTuner, ContinuousParameter

# Define the tuning job
convergence_tuner = HyperparameterTuner(
    estimator=convergence_estimator,
    hyperparameter_ranges={
        "LEARNING_RATE": ContinuousParameter(1e-6, 1e-3, "Logarithmic"),
        "WEIGHT_DECAY": ContinuousParameter(1e-6, 1e-3, "Logarithmic"),
    },
    base_tuning_job_name="optimized-tf29-vit",
    **tuner_args,
)

# Start the tuning job with the specified input data
convergence_tuner.fit(inputs=destn, wait=False)

# Save the name of the tuning job
convergence_tuner.latest_tuning_job.name
```

No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config

```
Out[16]: 'optimized-tf29-vit-220623-1625'
```


## Wait for tuning jobs to complete

The tuning jobs described above typically take around 2 hours to complete

**Note:** If the tuner object is no longer available due to a kernel break or refresh, you need to directly use the training job name and manually attach the tuning job to a new tuner. For example:

```
tuner = HyperparameterTuner.attach("<your_tuning_job_name>")
```

```
In [17]: convergence_tuner.wait()
```

.....  


```
In [18]: from sagemaker.tuner import HyperparameterTuner

convergence_tuner = HyperparameterTuner.attach(convergence_tuner.latest_tuning_job.name)
convergence_tuner.wait()
```

!

## Fastest Convergence

Let us analyze the results from the tuning jobs. The tuner provides the results as a Pandas dataframe. We sort by training loss and display the top 5 results.

```
In [19]: import pandas as pd

pd.set_option("display.max_rows", None, "display.max_columns", None)

# Gather results from the tuner
results = convergence_tuner.analytics().dataframe()

# Sort according to Training Loss
results = results.sort_values(["FinalObjectiveValue"], ascending=[True])

# Display the top 5 results
results.head(5)
```

```
Out[19]:
```

	LEARNING_RATE	WEIGHT_DECAY	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	Tra
7	0.000086	0.000016	optimized-tf29-vit-220623-1625-033-150957a2	Completed	3.4129	
2	0.000085	0.000011	optimized-tf29-vit-220623-1625-038-d0fa3d30	Completed	3.4508	
13	0.000072	0.000010	optimized-tf29-vit-220623-1625-027-176f5cbb	Completed	3.4530	
12	0.000066	0.000002	optimized-tf29-vit-220623-1625-028-e873cb35	Completed	3.5116	
15	0.000042	0.000011	optimized-tf29-vit-220623-1625-025-de6dcfef	Completed	3.5155	

Having obtained the best configuration for your training job, you can now train to completion. Please consider [checkpointing \(https://docs.aws.amazon.com/sagemaker/latest/dg/model-checkpoints.html\)](https://docs.aws.amazon.com/sagemaker/latest/dg/model-checkpoints.html) in order to resume training from the best performing job indicated by the tuner.

## Conclusion

In conclusion, we first arrived at the batch size and compiler configuration that leads to the highest training throughput. Then, we tuned the associated hyperparameters to arrive at the configuration that leads to the fastest convergence. The resultant combinations leads to maximum savings !

## Clean up

Stop all tuning jobs launched if the jobs are still running.

```
In [20]: native_tuner.stop_tuning_job()
optimized_tuner.stop_tuning_job()
convergence_tuner.stop_tuning_job()
```

Also, to find instructions on cleaning up resources, see [Clean Up \(https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html\)](https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html) in the *Amazon SageMaker*

*Developer Guide.*