Compile and Train a Vision Transformer Model on the Caltech-256 Dataset using a Single Node

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 Deep Learning (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 Amazon SageMaker Training Compiler to train 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 a `ml.p3.2xlarge` instance 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 [31]:
```
!pip install "sagemaker>=2.95" botocore boto3 awscli matplotlib --upgrade
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://pip.repos.neuron.amazonaws.com (https://pip.repos.neuron.amazonaw
s.com)
Requirement already satisfied: sagemaker>=2.92 in /home/ec2-user/anaconda
3/envs/tensorflow2_p38/lib/python3.8/site-packages (2.94.0)
Requirement already satisfied: botocore in /home/ec2-user/anaconda3/envs/
tensorflow2_p38/lib/python3.8/site-packages (1.27.9)
Collecting botocore
  Downloading botocore-1.27.10-py3-none-any.whl (8.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 55.6 MB/s eta 0:
00:0000:0100:01
Requirement already satisfied: boto3 in /home/ec2-user/anaconda3/envs/ten
sorflow2_p38/lib/python3.8/site-packages (1.24.9)
Collecting boto3
  Downloading boto3-1.24.10-py3-none-any.whl (132 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.5/132.5 KB 41.3 MB/s eta
0:00:00
Requirement already satisfied: awscli in /home/ec2-user/anaconda3/envs/te
nsorflow2_p38/lib/python3.8/site-packages (1.25.9)
Collecting awscli
  Downloading awscli-1.25.10-py3-none-any.whl (3.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.9/3.9 MB 136.2 MB/s eta
0:00:00
Requirement already satisfied: matplotlib in /home/ec2-user/anaconda3/env
s/tensorflow2_p38/lib/python3.8/site-packages (3.5.2)
Requirement already satisfied: pandas in /home/ec2-user/anaconda3/envs/te
nsorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.92) (1.3.4)
Requirement already satisfied: pathos in /home/ec2-user/anaconda3/envs/te
nsorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.92) (0.2.8)
Requirement already satisfied: numpy<2.0,>=1.9.0 in /home/ec2-user/anacon
da3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.9
2) (1.20.3)
Requirement already satisfied: google-pasta in /home/ec2-user/anaconda3/e
nvs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.92)
 (0.2.0)
Requirement already satisfied: attrs==20.3.0 in /home/ec2-user/anaconda3/
envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.92)
 (20.3.0)
Requirement already satisfied: protobuf<4.0,>=3.1 in /home/ec2-user/anaco
nda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.
92) (3.19.1)
Requirement already satisfied: protobuf3-to-dict<1.0,>=0.1.5 in /home/ec2
-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sa
gemaker>=2.92) (0.1.5)
Requirement already satisfied: importlib-metadata<5.0,>=1.4.0 in /home/ec
2-user/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from s
agemaker>=2.92) (1.7.0)
Requirement already satisfied: packaging>=20.0 in /home/ec2-user/anaconda
3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagemaker>=2.92)
(21.0)
Requirement already satisfied: smdebug-rulesconfig==1.0.1 in /home/ec2-us
er/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from sagem
aker>=2.92) (1.0.1)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /home/ec2-user/a
naconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from botocore)
```

```
(0.10.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/ec2-u
ser/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto
core) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-user/an
aconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from botocore)
 (1.26.8)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in /home/ec2-use
r/anaconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from boto3)
(0.6.0)
Requirement already satisfied: rsa<4.8,>=3.1.2 in /home/ec2-user/anaconda
3/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/anacon
da3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (5.4.
1)
Requirement already satisfied: docutils<0.17,>=0.10 in /home/ec2-user/ana
conda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli) (0.
15.2)
Requirement already satisfied: colorama<0.4.5,>=0.2.5 in /home/ec2-user/a
naconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from awscli)
 (0.4.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/ec2-user/anacon
da3/envs/tensorflow2_p38/lib/python3.8/site-packages (from matplotlib)
 (1.3.2)
Requirement already satisfied: cycler>=0.10 in /home/ec2-user/anaconda3/e
nvs/tensorflow2_p38/lib/python3.8/site-packages (from matplotlib) (0.11.
0)
Requirement already satisfied: pyparsing>=2.2.1 in /home/ec2-user/anacond
a3/envs/tensorflow2_p38/lib/python3.8/site-packages (from matplotlib) (3.
0.6)
Requirement already satisfied: fonttools>=4.22.0 in /home/ec2-user/anacon
da3/envs/tensorflow2_p38/lib/python3.8/site-packages (from matplotlib)
 (4.33.3)
Requirement already satisfied: pillow>=6.2.0 in /home/ec2-user/anaconda3/
envs/tensorflow2_p38/lib/python3.8/site-packages (from matplotlib) (9.0.
1)
Requirement already satisfied: zipp>=0.5 in /home/ec2-user/anaconda3/env
s/tensorflow2_p38/lib/python3.8/site-packages (from importlib-metadata<5.
0,>=1.4.0->sagemaker>=2.92) (3.6.0)
Requirement already satisfied: six in /home/ec2-user/anaconda3/envs/tenso
rflow2_p38/lib/python3.8/site-packages (from protobuf3-to-dict<1.0,>=0.1.
5->sagemaker>=2.92) (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->a
wscli) (0.4.8)
Requirement already satisfied: pytz>=2017.3 in /home/ec2-user/anaconda3/e
nvs/tensorflow2_p38/lib/python3.8/site-packages (from pandas->sagemaker>=
2.92) (2021.3)
Requirement already satisfied: multiprocess>=0.70.12 in /home/ec2-user/an
aconda3/envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sa
gemaker>=2.92) (0.70.12.2)
Requirement already satisfied: dill>=0.3.4 in /home/ec2-user/anaconda3/en
vs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=
2.92) (0.3.4)
Requirement already satisfied: pox>=0.3.0 in /home/ec2-user/anaconda3/env
s/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>=2.
92) (0.3.0)
```

```
Requirement already satisfied: ppft>=1.6.6.4 in /home/ec2-user/anaconda3/
envs/tensorflow2_p38/lib/python3.8/site-packages (from pathos->sagemaker>
=2.92) (1.6.6.4)
Installing collected packages: botocore, boto3, awscli
  Attempting uninstall: botocore
    Found existing installation: botocore 1.27.9
    Uninstalling botocore-1.27.9:
      Successfully uninstalled botocore-1.27.9
  Attempting uninstall: boto3
    Found existing installation: boto3 1.24.9
    Uninstalling boto3-1.24.9:
      Successfully uninstalled boto3-1.24.9
  Attempting uninstall: awscli
    Found existing installation: awscli 1.25.9
    Uninstalling awscli-1.25.9:
      Successfully uninstalled awscli-1.25.9
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the foll
owing dependency conflicts.
aiobotocore 2.0.1 requires botocore<1.22.9,>=1.22.8, but you have botocor
e 1.27.10 which is incompatible.
Successfully installed awscli-1.25.10 boto3-1.24.10 botocore-1.27.10
WARNING: You are using pip version 22.0.4; however, version 22.1.2 is ava
ilable.
You should consider upgrading via the '/home/ec2-user/anaconda3/envs/tens
orflow2_p38/bin/python -m pip install --upgrade pip' command.
```

In [32]:

```
import botocore
import boto3
import sagemaker

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

```
botocore: 1.27.7
boto3: 1.24.7
sagemaker: 2.94.0
```

SageMaker environment

In [33]:
```
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-east-1. 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 [34]:
```
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[34]: 0

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 with Native TensorFlow

The `BATCH_SIZE` in the following code cell is the maximum batch that can fit into the memory of a `ml.p3.2xlarge` instance while giving the best training speed. If you change the model, instance type, and other parameters, you need to do some experiments to find the largest batch size that will fit into GPU memory.

Set `EPOCHS` to the number of times you would like to loop over the training data.

In [36]:
```
from sagemaker.tensorflow import TensorFlow

EPOCHS = 10
BATCH_SIZE = 64
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4

kwargs = 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 * 60,  # 60 minutes
    role=role,
    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/epoc
        {"Name": "training_avg_latency_per_step", "Regex": "- ([0-9.]*?)ms/
    ],
)

# Configure the training job
native_estimator = TensorFlow(
    hyperparameters={
        "EPOCHS": EPOCHS,
        "BATCH_SIZE": BATCH_SIZE,
        "LEARNING_RATE": LEARNING_RATE,
        "WEIGHT_DECAY": WEIGHT_DECAY,
    },
    base_job_name="native-tf29-vit",
    **kwargs,
)

# Start training with our uploaded datasets as input
native_estimator.fit(inputs=destn, wait=False)

# The name of the training job.
native_estimator.latest_training_job.name
```

Out[36]:
```
'native-tf29-vit-2022-06-15-23-47-24-579'
```

Training with Optimized TensorFlow

Compilation through 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. But in some cases the compiler intelligently promotes caching which leads to a decrease in the largest batch size that can fit on the GPU. Note that if you want to change the batch size, you must adjust the learning rate appropriately.

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

In [37]:
```
from sagemaker.tensorflow import TensorFlow, TrainingCompilerConfig

OPTIMIZED_BATCH_SIZE = 48
LEARNING_RATE = LEARNING_RATE / BATCH_SIZE * OPTIMIZED_BATCH_SIZE
WEIGHT_DECAY = WEIGHT_DECAY * BATCH_SIZE / OPTIMIZED_BATCH_SIZE

# Configure the training job
optimized_estimator = TensorFlow(
    hyperparameters={
        "EPOCHS": EPOCHS,
        "BATCH_SIZE": OPTIMIZED_BATCH_SIZE,
        "LEARNING_RATE": LEARNING_RATE,
        "WEIGHT_DECAY": WEIGHT_DECAY,
    },
    compiler_config = TrainingCompilerConfig(),
    base_job_name="optimized-tf29-vit",
    **kwargs,
)

# Start training with our uploaded datasets as input
optimized_estimator.fit(inputs=destn, wait=False)

# The name of the training job.
optimized_estimator.latest_training_job.name
```

Out[37]:
```
'optimized-tf29-vit-2022-06-15-23-47-25-126'
```

Wait for training jobs to complete

The training jobs described above typically take around 40 mins to complete

Note: If the estimator 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 training job to a new TensorFlow estimator. For example:

```
native_estimator = TensorFlow.attach("<your_training_job_name>")
```

In [51]:
```
waiter = sess.sagemaker_client.get_waiter("training_job_completed_or_stoppe

waiter.wait(TrainingJobName=native_estimator.latest_training_job.name)
waiter.wait(TrainingJobName=optimized_estimator.latest_training_job.name)
```

In [52]:
```
native_estimator = TensorFlow.attach(native_estimator.latest_training_job.n
optimized_estimator = TensorFlow.attach(optimized_estimator.latest_training
```

```
2022-06-16 00:17:54 Starting - Preparing the instances for training
2022-06-16 00:17:54 Downloading - Downloading input data
2022-06-16 00:17:54 Training - Training image download completed. Trainin
g in progress.
2022-06-16 00:17:54 Uploading - Uploading generated training model
2022-06-16 00:17:54 Completed - Training job completed

2022-06-16 00:09:28 Starting - Preparing the instances for training
2022-06-16 00:09:28 Downloading - Downloading input data
2022-06-16 00:09:28 Training - Training image download completed. Trainin
g in progress.
2022-06-16 00:09:28 Uploading - Uploading generated training model
2022-06-16 00:09:28 Completed - Training job completed
```

Analysis

Here we view the training metrics from the training jobs as a Pandas dataframe

Native TensorFlow

In [53]:
```
import pandas as pd

# Extract training metrics from the estimator
native_metrics = native_estimator.training_job_analytics.dataframe()

# Restructure table for viewing
for metric in native_metrics["metric_name"].unique():
    native_metrics[metric] = native_metrics[native_metrics["metric_name"] =
native_metrics = native_metrics.drop(columns=["metric_name", "value"])
native_metrics = native_metrics.groupby("timestamp").max()
native_metrics["epochs"] = range(1, 11)
native_metrics = native_metrics.set_index("epochs")

native_metrics
```

Out[53]:

	training_loss	training_accuracy	training_latency_per_epoch	training_avg_latency_per_step
epochs				
1	5.8060	0.0195	152.0	533.0
2	5.7542	0.0228	117.0	409.0
3	5.7606	0.0213	116.0	406.0
4	5.7599	0.0208	116.0	405.0
5	5.7609	0.0244	116.0	406.0
6	5.7400	0.0203	116.0	405.0
7	5.7327	0.0233	116.0	406.0
8	5.7035	0.0227	116.0	406.0
9	5.4556	0.0284	116.0	406.0
10	5.3689	0.0309	116.0	406.0

Optimized TensorFlow

In [54]:
```
import pandas as pd

# Extract training metrics from the estimator
optimized_metrics = optimized_estimator.training_job_analytics.dataframe()

# Restructure table for viewing
for metric in optimized_metrics["metric_name"].unique():
    optimized_metrics[metric] = optimized_metrics[optimized_metrics["metric
        "value"
    ]
optimized_metrics = optimized_metrics.drop(columns=["metric_name", "value"]
optimized_metrics = optimized_metrics.groupby("timestamp").max()
optimized_metrics["epochs"] = range(1, 11)
optimized_metrics = optimized_metrics.set_index("epochs")

optimized_metrics
```

Out[54]:

epochs	training_loss	training_accuracy	training_latency_per_epoch	training_avg_latency_per_step
1	5.7583	0.0198	120.0	313.0
2	5.7246	0.0217	67.0	174.0
3	5.7115	0.0226	66.0	174.0
4	5.7208	0.0215	67.0	175.0
5	5.6849	0.0226	66.0	174.0
6	5.4777	0.0292	67.0	174.0
7	5.3669	0.0300	66.0	174.0
8	5.2745	0.0383	67.0	174.0
9	5.0772	0.0486	66.0	174.0
10	4.9200	0.0625	66.0	174.0

Savings from Training Compiler

Let us calculate the actual savings on the training jobs above and the potential for savings for a longer training job.

Actual Savings

To get the actual savings, we use the describe_training_job API to get the billable seconds for each training job.

In [55]:
```
# Billable seconds for the Native TensorFlow Training job

details = sess.describe_training_job(job_name=native_estimator.latest_train
native_secs = details["BillableTimeInSeconds"]

native_secs
```

Out[55]:
```
1722
```

In [56]:
```
# Billable seconds for the Optimized TensorFlow Training job

details = sess.describe_training_job(job_name=optimized_estimator.latest_tr
optimized_secs = details["BillableTimeInSeconds"]

optimized_secs
```

Out[56]:
```
1217
```

In [57]:
```
# Calculating percentage Savings from Training Compiler

percentage = (native_secs - optimized_secs) * 100 / native_secs

f"Training Compiler yielded {percentage:.2f}% savings in training cost."
```

Out[57]:
```
'Training Compiler yielded 29.33% savings in training cost.'
```

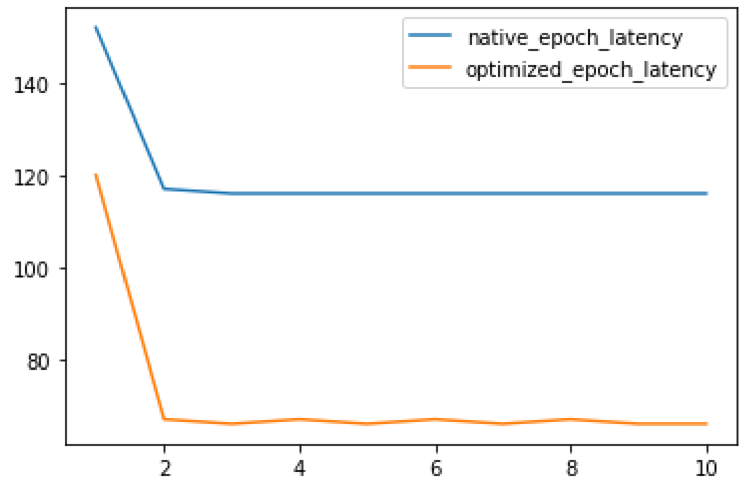
Potential savings

The Training Compiler works by compiling the model graph once per input shape and reusing the cached graph for subsequent steps. As a result the first few steps of training incur an increased latency owing to compilation which we refer to as the compilation overhead. This overhead is amortized over time thanks to the subsequent steps being much faster. We will demonstrate this below.

In [58]:
```
import matplotlib.pyplot as plt

plt.plot(native_metrics["training_latency_per_epoch"], label="native_epoch_
plt.plot(optimized_metrics["training_latency_per_epoch"], label="optimized_
plt.legend()
```

Out[58]: <matplotlib.legend.Legend at 0x7f08cbe5cbb0>



We calculate the potential savings below from the difference in steady state epoch latency between native TensorFlow and optimized TensorFlow

In [59]:
```
native_steady_state_latency = native_metrics["training_latency_per_epoch"].

native_steady_state_latency
```

Out[59]: 116.0

In [60]:
```
optimized_steady_state_latency = optimized_metrics["training_latency_per_ep

optimized_steady_state_latency
```

Out[60]: 66.0

In [61]:
```
# Calculating potential percentage Savings from Training Compiler

percentage = (
    (native_steady_state_latency - optimized_steady_state_latency)
    * 100
    / native_steady_state_latency
)

f"Training Compiler can potentially yield {percentage:.2f}% savings in trai
```

Out[61]: 'Training Compiler can potentially yield 43.10% savings in training cost for a longer training job.'

Convergence of Training

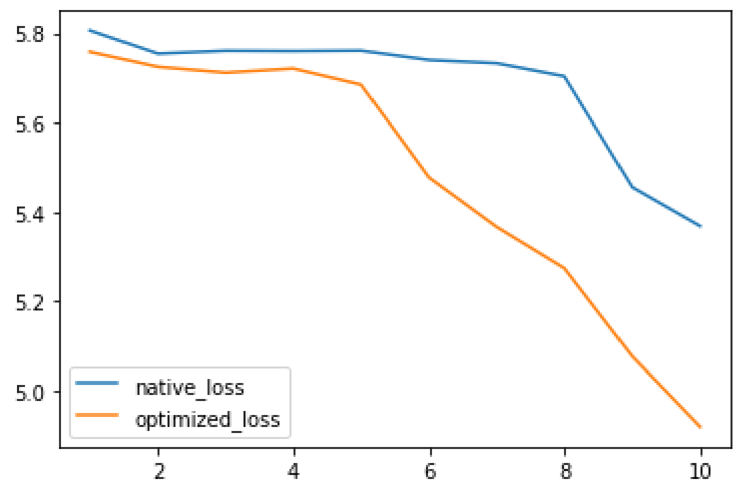
Training Compiler brings down total training time by intelligently choosing between memory utilization and core utilization in the GPU. This does not have any effect on the model arithmetic and consequently convergence of the model.

However, since we are working with a new batch size, hyperparameters like - learning rate, learning rate schedule and weight decay might have to be scaled and tuned for the new batch size

In [62]:
```
import matplotlib.pyplot as plt

plt.plot(native_metrics["training_loss"], label="native_loss")
plt.plot(optimized_metrics["training_loss"], label="optimized_loss")
plt.legend()
```

Out[62]:
```
<matplotlib.legend.Legend at 0x7f08cbdbe250>
```



We can see that the model's convergence behavior is similar with and without Training Compiler. Here we have tuned the batch size specific hyperparameters - Learning Rate and Weight Decay using a linear scaling.

Learning rate is directly proportional to the batch size:

```
new_learning_rate = old_learning_rate * new_batch_size/old_batch_si
ze
```

Weight decay is inversely proportional to the batch size:

```
new_weight_decay = old_weight_decay * old_batch_size/new_batch_size
```

Better results can be achieved with further tuning. Check out [Automatic Model Tuning (https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html)](https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html) for tuning.

Clean up

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

In [63]:
```
def stop_training_job(name):
    status = sess.describe_training_job(name)["TrainingJobStatus"]
    if status == "InProgress":
        sm.stop_training_job(TrainingJobName=name)


stop_training_job(native_estimator.latest_training_job.name)
stop_training_job(optimized_estimator.latest_training_job.name)
```

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*.