lambda-deep-learning-demo Documentation

Release 0.0.1

Lambda Labs

Apr 26, 2019

Contents

CHAPTER 1

Tutorials

1.1 Getting Started

Welcome to Lambda Lab's deep learning demo suite – the place to find ready-to-use machine learnig models. We offer the following cool features:

- A curate of open-source, state-of-the-art models that cover major machine learning applications, including image classification, image segmentation, object detection etc.
- Pure Tensorflow implementation. Efforts are made to keep the boilplate consistent across all demos.
- Examples of transfer learning and how to adapt the model to new data.

In this getting started guide, we will walk you through the glossary of our code and the steps of building tensorflow applications.

- *Glossary*
- *Example: Training a ResNet32 newtowrk on CIFAR10*

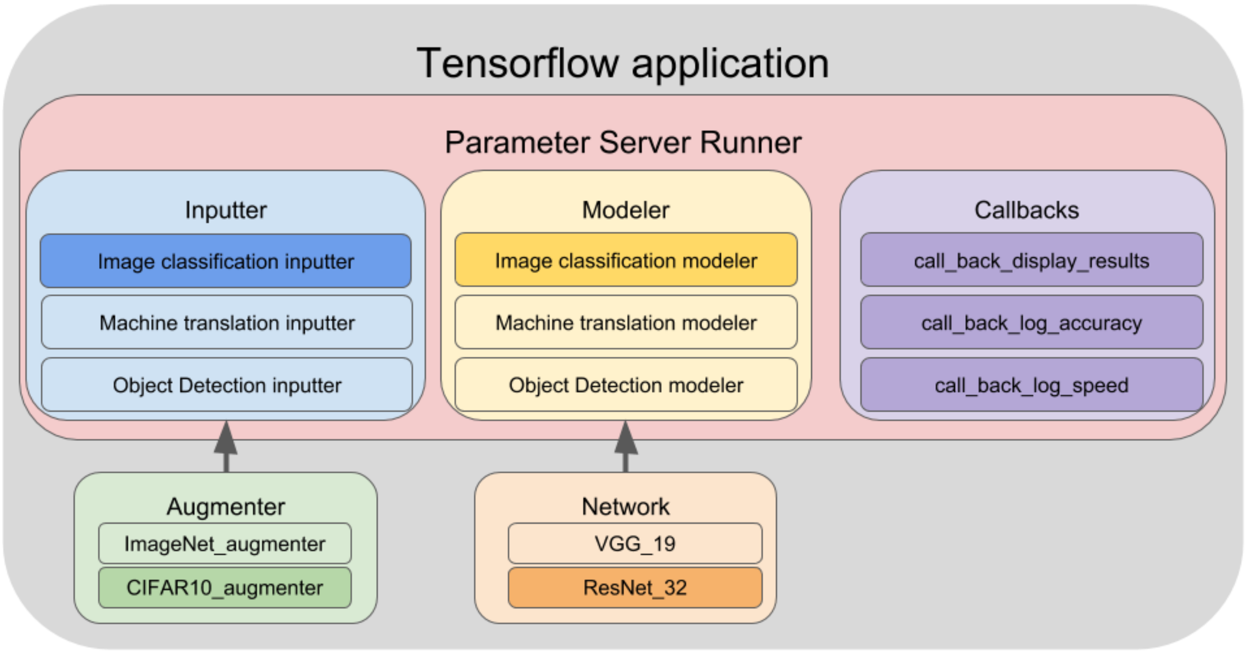
1.1.1 Glossary

Our TensorFlow application is comprised of three main components:

- **Inputter**: The data pipeline. It reads data from the disk, shuffles and preprocesses the data, creates batches, and does prefetching. An inputter is applicable to a specific problem that share the same type of input and output. For example, we have image_classification_inputter, text_classification_inputter, object_detection_inputter . . . etc. An inputter can optionally own an **augmenter** for data augmentation, for example, in the case of image related tasks, the augmenter can perform resizing, random cropping and color distortion . . . etc.
- **Modeler**: The model pipeline. The Modeler encapsulates the forward pass and the backward pass. Like the inputter, a modeler is applicable to a specific problem such as image classification or object detection. A modeler must own a **network** member that implements the network architecture, for example, an image classification modeler can choose ResNet32, VGG19 or InceptionV4 as its network architecture.

- **Runner**: The job executor. It orchestrates the execution of an Inputter and a Modeler and distributes the workload across multiple hardware devices. It also uses **callbacks** to perform auxiliary tasks such as logging, model saving and result visualization.

Fig. **??** illustrates the composition of a tensorflow application using these building blocks.



1.1.2 Example: Training a ResNet32 newtowrk on CIFAR10

Let's walk through an example of building a Tensorflow application. In this example we will use a ResNet32 model for classifying CIFAR10 images.

Before diving into details, here is all the code for this guide. It gives an overview of the boilplate we use to build Tensorflow applications.

```
# Create basic inputter configration
inputter_config = InputterConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  epochs=4,
  dataset_meta="~/demo/data/cifar10/train.csv \")

# Add additional configuration for image classification
inputter_config = ImageClassificationInputterConfig(
  inputter_config,
  image_height=32,
  image_width=32,
  image_depth=3,
  num_classes=10)


# (Optionally) Create a augmenter.
```

(continues on next page)

(continued from previous page)

```
argmenter_name = "source.augmenter.cifar_augmenter"
augmenter = importlib.import_module(argmenter_name)

# Create a Inputter.
inputter_name = "source.inputter.image_classification_csv_inputter"
inputter = importlib.import_module(inputter_name).build(inputter_config, augmenter)


# Create a ResNet32 network
network_name = "source.network.resnet32"
net = getattr(importlib.import_module(network_name), "net")

# Create basic modeler configration
modeler_config = ModelerConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  optimizer="momentum",
  learning_rate=0.01)

# Add additional configuration for image classification
modeler_config = ImageClassificationModelerConfig(
  modeler_config,
  num_classes=10)

# Create modeler
modeler_name = "source.modeler.image_classification_modeler"
modeler = importlib.import_module(modeler_name).build(modeler_config, net)

  # Create callback configuations
callback_config = CallbackConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  model_dir="~/demo/model/image_classification_cifar10",
  log_every_n_iter=10,
  save_summary_steps=10)

# Create callbacks
callback_names = ["train_basic", "train_loss", "train_accuracy",
                  "train_speed", "train_summary"]
callbacks = []
for name in callback_names:
      callback = importlib.import_module(
        "source.callback." + name).build(callback_config)
      callbacks.append(callback)

# Create run config
runner_config = RunnerConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  summary_names=["loss,accuracy", "learning_rate"])

# Create a runner
runner_name = "source.runner.parameter_server_runner"
runner = importlib.import_module(runner_name).build(runner_config, inputter, modeler,␣
↪callbacks)
```

(continues on next page)

(continued from previous page)

```
# Run the application
runner.run()
```

You can find details for each step in the follows chapters:

1.1.2.1 Define an Inputter

The inputter is the data pipeline. This example defines the data pipeline of feeding CIFAR10 data with some basic augmentations:

```
# Create basic inputter configration
inputter_config = InputterConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  epochs=4,
  dataset_meta="~/demo/data/cifar10/train.csv \")

# Add additional configuration for image classification
inputter_config = ImageClassificationInputterConfig(
  inputter_config,
  image_height=32,
  image_width=32,
  image_depth=3,
  num_classes=10)

# (Optionally) Create a augmenter.
argmenter_name = "source.augmenter.cifar_augmenter"
augmenter = importlib.import_module(argmenter_name)

# Create a Inputter.
inputter_name = "source.inputter.image_classification_csv_inputter"
inputter = importlib.import_module(inputter_name).build(inputter_config, augmenter)
```

- `cifar_augmenter` does random image cropping, flipping, brightness and contrast distortions.
- `inputter_config` sets arguments for the inputter. For example, whether it is used for training or evaluation, batch_size, the data path … etc.
- `inputter` is the data pipeline instance. It has an important `input_fn` member function that produces a data generator.

The `input_fn` of an image classification inputter looks like this:

```
def input_fn(self, test_samples=[]):

  # Get list of image paths and class labels
  samples = self.get_samples_fn()

  # Generate a Tensorflow dataset
  dataset = tf.data.Dataset.from_tensor_slices(samples)

  # Shuffle the dataset for training
  if self.config.mode == "train":
    dataset = dataset.shuffle(self.get_num_samples())
```

(continues on next page)

(continued from previous page)

```
  # Repeat the dataset for multiple epochs
  dataset = dataset.repeat(self.config.epochs)

  # Parse individal input sample, including reading image from path,
  # data augmentation
  dataset = dataset.map(
    lambda image, label: self.parse_fn(image, label),
    num_parallel_calls=4)

  # Batch data
  batch_size = (self.config.batch_size_per_gpu *
                self.config.gpu_count)
  dataset = dataset.apply(
      tf.contrib.data.batch_and_drop_remainder(batch_size))

  # Prefetch for efficiency
  dataset = dataset.prefetch(2)

  # Return data generator
  iterator = dataset.make_one_shot_iterator()
  return iterator.get_next()
```

1.1.2.2 Define a Modeler

The modeler defines the model pipeline. This example defines the computation that is needed for a ResNet32 network:

```
# Create a ResNet32 network
network_name = "source.network.resnet32"
net = getattr(importlib.import_module(network_name), "net")

# Create basic modeler configration
modeler_config = ModelerConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  optimizer="momentum",
  learning_rate=0.01)

# Add additional configuration for image classification
modeler_config = ImageClassificationModelerConfig(
  modeler_config,
  num_classes=10)

# Create modeler
modeler_name = "source.modeler.image_classification_modeler"
modeler = importlib.import_module(modeler_name).build(modeler_config, net)
```

- `net` is the function that implments ResNet32's forward pass.
- `modeler_config` contains the argments for building a ResNet32 model. Importantly, it sets up the number of classes.
- `modeler` is the model pipeline. It has an important `model_fn` member function that outputs a dictionary of operators to be run by a Tensorflow session.

The `model_fn` for an image classification modeler looks like this:

```
def model_fn(self, x):

  # Input batch of images and labels
  images = x[0]
  labels = x[1]

  # Create graph for forward pass
  logits, predictions = self.create_graph_fn(images)

  # Return modeler operators
  if self.config.mode == "train":

    # Training mode returns operators for loss, gradient and accuracy
    loss = self.create_loss_fn(logits, labels)
    grads = self.create_grad_fn(loss)
    accuracy = self.create_eval_metrics_fn(
      predictions, labels)
    return {"loss": loss,
            "grads": grads,
            "accuracy": accuracy,
            "learning_rate": self.learning_rate}
  elif self.config.mode == "eval":

    # Evalution mode returns operators for loss and accuracy
    loss = self.create_loss_fn(logits, labels)
    accuracy = self.create_eval_metrics_fn(
      predictions, labels)
    return {"loss": loss,
            "accuracy": accuracy}
  elif self.config.mode == "infer":

    # Inference mode returns the predicted classes and probabilities for the␣
↪predictions
    return {"classes": predictions["classes"],
            "probabilities": predictions["probabilities"]}
```

1.1.2.3 Define a Runner

A runner runs the inputter and the modeler. It also use callbacks for auxiliary jobs:

```
# Create callback configuations
callback_config = CallbackConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  model_dir="~/demo/model/image_classification_cifar10",
  log_every_n_iter=10,
  save_summary_steps=10)

# Create callbacks
callback_names = ["train_basic", "train_loss", "train_accuracy",
                  "train_speed", "train_summary"]
callbacks = []
for name in callback_names:
      callback = importlib.import_module(
        "source.callback." + name).build(callback_config)
```

(continues on next page)

(continued from previous page)

```
      callbacks.append(callback)

# Create run config
runner_config = RunnerConfig(
  mode="train",
  batch_size_per_gpu=64,
  gpu_count=1,
  summary_names=["loss,accuracy", "learning_rate"])

# Create a runner
runner_name = "source.runner.parameter_server_runner"
runner = importlib.import_module(runner_name).build(runner_config, inputter, modeler,␣
↪callbacks)
```

There are two main tasks for a runner: First, running some operators in a Tensorflow session. Second, distributes the computation across multiple-devices if it is needed.

The `run` member function implements the run:

```
def run(self):

  # Create the computation graph
  self.create_graph()

  # Create a Tensorflow session
  with tf.Session(config=self.session_config) as self.sess:

    # Do auxiliary jobs before running the graph
    self.before_run()

    # Set up the global step and the maximum step to run
    global_step = 0
    if self.config.mode == "train":
      # For resuming training from the last checkpoint
      global_step = self.sess.run(self.global_step_op)

    max_step = self.sess.run(self.max_step_op)

    # Run the job until max_step
    while global_step < max_step:

      # Do auxiliary jobs before running a step
      self.before_step()

      # Run a step
      self.outputs = self.sess.run(self.run_ops)

      # Do auxiliary jobs after running a step
      self.after_step()

      global_step = global_step + 1

    # Do auxiliary jobs after finishing the run
    self.after_run()
```

The second task is to distribute computation across multiple device if it is necessary. In this example we use dsynchronized multi-GPU training with a CPU as the parameter server. To do so we use a `parameter_server_runner` that splits the input data across multiple-GPUs, run computation in parallel on these GPUs, and gather the results for parameter update. The key logic is implemented in its `replicate_graph` member function.

```
def replicate_graph(self):

  # Fetch input daaa
  batch = self.inputter.input_fn()

  if self.config.mode == "infer":

    # Use a single GPU for inference
    with tf.device(self.assign_to_device("/gpu:{}".format(0),
                   ps_device="/cpu:0")):
      ops = self.modeler.model_fn(batch)
      return ops

  else:

    output = {}
    # Distribute work across multiple GPUs
    for i in range(self.config.gpu_count):
      with tf.device(self.assign_to_device("/gpu:{}".format(i),
                     ps_device="/cpu:0")):

        # Get the split for the i-th GPU
        x = self.batch_split(batch, i)
        y = self.modeler.model_fn(x)

        # Gather output from the i-th GPU
        if i == 0:
          for key in y:
            output[key] = [y[key]]
        else:
          for key in y:
            output[key].append(y[key])

    # Average results
    reduced_ops = {}
    for key in output:
      reduced_ops[key] = self.reduce_op(output[key])

    # Return the operation to run averaged results
    return reduced_ops

  # Run the application
  runner.run()
```

To run the application, simply call `runner.run()`.

1.2 Overview

1.3 Applications

The demo suite provides the following list of applications:

1.3.1 Image Classification

- *Download CIFAR10 Dataset*
- *Train ResNet32 from scratch on CIFAR10*
- *Evaluation*
- *Inference*
- *Hyper-Parameter Tuning*
- *Evaluate Pre-trained model*
- *Export*
- *Serve*

1.3.1.1 Download CIFAR10 Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/cifar10.tar.gz \
--data_dir=~/demo/data
```

1.3.1.2 Train ResNet32 from scratch on CIFAR10

```
python demo/image/image_classification.py \
--mode=train \
--model_dir=~/demo/model/resnet32_cifar10 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--batch_size_per_gpu=256 --epochs=100 \
train_args \
--learning_rate=0.5 --optimizer=momentum \
--piecewise_boundaries=50,75,90 \
--piecewise_lr_decay=1.0,0.1,0.01,0.001 \
--dataset_meta=~/demo/data/cifar10/train.csv
```

1.3.1.3 Evaluation

```
python demo/image/image_classification.py \
--mode=eval \
--model_dir=~/demo/model/resnet32_cifar10 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--batch_size_per_gpu=128 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/cifar10/eval.csv
```

1.3.1.4 Inference

```
python demo/image/image_classification.py \
--mode=infer \
--model_dir=~/demo/model/resnet32_cifar10 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
infer_args \
--callbacks=infer_basic,infer_display_image_classification \
--test_samples=~/demo/data/cifar10/test/appaloosa_s_001975.png,~/demo/data/cifar10/
↪test/domestic_cat_s_001598.png,~/demo/data/cifar10/test/rhea_s_000225.png,~/demo/
↪data/cifar10/test/trucking_rig_s_001216.png
```

1.3.1.5 Hyper-Parameter Tuning

```
python demo/image/image_classification.py \
--mode=tune \
--model_dir=~/demo/model/resnet32_cifar10 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--batch_size_per_gpu=128 \
tune_args \
--train_dataset_meta=~/demo/data/cifar10/train.csv \
--eval_dataset_meta=~/demo/data/cifar10/eval.csv \
--tune_config=source/tool/resnet32_cifar10_tune_coarse.yaml

python demo/image_classification.py \
--mode=tune \
--model_dir=~/demo/model/resnet32_cifar10 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--batch_size_per_gpu=128 \
tune_args \
--train_dataset_meta=~/demo/data/cifar10/train.csv \
--eval_dataset_meta=~/demo/data/cifar10/eval.csv \
--tune_config=source/tool/resnet32_cifar10_tune_fine.yaml
```

1.3.1.6 Evaluate Pre-trained model

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/cifar10-resnet32-20180824.
↪tar.gz | tar xvz -C ~/demo/model
```

```
python demo/image/image_classification.py \
--mode=eval \
--model_dir=~/demo/model/cifar10-resnet32-20180824 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--batch_size_per_gpu=128 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/cifar10/eval.csv
```

1.3.1.7 Export

```
python demo/image/image_classification.py \
--mode=export \
--model_dir=~/demo/model/cifar10-resnet32-20180824 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_classes
```

1.3.1.8 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_classification \
--mount type=bind,source=model_dir/export,target=/models/classification \
-e MODEL_NAME=classification -t tensorflow/serving:latest-gpu &

python client/image_classification_client.py --image_path=path_to_image
```

1.3.2 Transfer Learning

- *ResNet50 on Stanford Dogs Dataset*
- *InceptionV4 on Stanford Dogs Dataset*
- *NasNet-A-Large on Stanford Dogs Dataset*

1.3.2.1 ResNet50 on Stanford Dogs Dataset

Download Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/StanfordDogs120.tar.gz␣
↪\
--data_dir=~/demo/data
```

Download Pre-trained Model

```
(mkdir ~/demo/model/resnet_v2_50_2017_04_14;curl http://download.tensorflow.org/
↪models/resnet_v2_50_2017_04_14.tar.gz | tar xvz -C ~/demo/model/resnet_v2_50_2017_
↪04_14)
```

Train with weights restored from pre-trained model

```
python demo/image/image_classification.py \
--mode=train \
--model_dir=~/demo/model/resnet50_StanfordDogs120 \
--network=resnet50 \
--augmenter=vgg_augmenter \
```

(continues on next page)

(continued from previous page)

```
--batch_size_per_gpu=16 --epochs=10 \
--num_classes=120 --image_width=224 --image_height=224 \
train_args \
--learning_rate=0.1 --optimizer=momentum \
--piecewise_boundaries=5 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--pretrained_model=~/demo/model/resnet_v2_50_2017_04_14/resnet_v2_50.ckpt \
--skip_pretrained_var=resnet_v2_50/logits,global_step,power \
--trainable_vars=resnet_v2_50/logits
```

Hyper-Parameter Tuning

```
python demo/image/image_classification.py \
--mode=tune \
--model_dir=~/demo/model/resnet50_StanfordDogs120 \
--network=resnet50 \
--augmenter=vgg_augmenter \
--batch_size_per_gpu=16 \
--num_classes=120 --image_width=224 --image_height=224 \
tune_args \
--train_dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--eval_dataset_meta=~/demo/data/StanfordDogs120/eval.csv \
--pretrained_model=~/demo/model/resnet_v2_50_2017_04_14/resnet_v2_50.ckpt \
--skip_pretrained_var=resnet_v2_50/logits,global_step,power \
--trainable_vars=resnet_v2_50/logits \
--tune_config=source/tool/resnet50_stanforddogs120_tune_coarse.yaml
```

Evaluation

```
(mkdir ~/demo/model/resnet50_StanfordDogs120-20190303;curl https://s3-us-west-2.
↪amazonaws.com/lambdalabs-files/resnet50_StanfordDogs120-20190303.tar.gz | tar xvz -
↪C ~/demo/model)

python demo/image/image_classification.py \
--mode=eval \
--model_dir=~/demo/model/resnet50_StanfordDogs120-20190303 \
--network=resnet50 \
--augmenter=vgg_augmenter \
--batch_size_per_gpu=16 --epochs=1 \
--num_classes=120 --image_width=224 --image_height=224 \
eval_args \
--dataset_meta=~/demo/data/StanfordDogs120/eval.csv
```

1.3.2.2 InceptionV4 on Stanford Dogs Dataset

Download pre-trained model

```
(mkdir ~/demo/model/inception_v4_2016_09_09;curl http://download.tensorflow.org/
↪models/inception_v4_2016_09_09.tar.gz | tar xvz -C ~/demo/model/inception_v4_2016_
↪09_09)
```

Train with weights restored from pre-trained model

```
python demo/image/image_classification.py \
--mode=train \
--model_dir=~/demo/model/inceptionv4_StanfordDogs120 \
--network=inception_v4 \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 --epochs=4 \
--num_classes=120 --image_width=299 --image_height=299 \
train_args \
--learning_rate=0.1 --optimizer=momentum \
--piecewise_boundaries=2 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--pretrained_model=~/demo/model/inception_v4_2016_09_09/inception_v4.ckpt \
--skip_pretrained_var=InceptionV4/AuxLogits,InceptionV4/Logits,global_step,power \
--trainable_vars=InceptionV4/AuxLogits,InceptionV4/Logits
```

Hyper-Parameter Tuning

```
python demo/image/image_classification.py \
--mode=tune \
--model_dir=~/demo/model/inceptionv4_StanfordDogs120 \
--network=inception_v4 \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 \
--num_classes=120 --image_width=299 --image_height=299 \
tune_args \
--train_dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--eval_dataset_meta=~/demo/data/StanfordDogs120/eval.csv \
--pretrained_model=~/demo/model/inception_v4_2016_09_09/inception_v4.ckpt \
--skip_pretrained_var=InceptionV4/AuxLogits,InceptionV4/Logits,global_step,power \
--trainable_vars=InceptionV4/AuxLogits,InceptionV4/Logits \
--tune_config=source/tool/inceptionv4_stanforddogs120_tune_coarse.yaml
```

Evaluation

```
(mkdir ~/demo/model/inceptionv4_StanfordDogs120-20190306;curl https://s3-us-west-2.
↪amazonaws.com/lambdalabs-files/inceptionv4_StanfordDogs120-20190306.tar.gz | tar␣
↪xvz -C ~/demo/model)

python demo/image/image_classification.py \
--mode=eval \
--model_dir=~/demo/model/inceptionv4_StanfordDogs120-20190306 \
--network=inception_v4 \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 --epochs=1 \
--num_classes=120 --image_width=299 --image_height=299 \
eval_args \
--dataset_meta=~/demo/data/StanfordDogs120/eval.csv
```

1.3.2.3 NasNet-A-Large on Stanford Dogs Dataset

Download pre-trained model

```
(mkdir ~/demo/model/nasnet-a_large_04_10_2017;curl https://storage.googleapis.com/
↪download.tensorflow.org/models/nasnet-a_large_04_10_2017.tar.gz | tar xvz -C ~/demo/
↪model/nasnet-a_large_04_10_2017)
```

Train with weights restored from pre-trained model

```
python demo/image/image_classification.py \
--mode=train \
--model_dir=~/demo/model/nasnet_A_large_StanfordDogs120 \
--network=nasnet_A_large \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 --epochs=4 \
--num_classes=120 --image_width=331 --image_height=331 \
train_args \
--learning_rate=0.1 --optimizer=momentum \
--piecewise_boundaries=2 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--pretrained_model=~/demo/model/nasnet-a_large_04_10_2017/model.ckpt \
--skip_pretrained_var=final_layer,aux_logits,global_step,power \
--trainable_vars=final_layer,aux_logits
```

Hyper-Parameter Tuning

```
python demo/image/image_classification.py \
--mode=tune \
--model_dir=~/demo/model/nasnet_A_large_StanfordDogs120 \
--network=nasnet_A_large \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 \
--num_classes=120 --image_width=331 --image_height=331 \
tune_args \
--train_dataset_meta=~/demo/data/StanfordDogs120/train.csv \
--eval_dataset_meta=~/demo/data/StanfordDogs120/eval.csv \
--pretrained_model=~/demo/model/nasnet-a_large_04_10_2/ KKi017/model.ckpt \
--skip_pretrained_var=final_layer,aux_logits,global_step,power \
--trainable_vars=final_layer,aux_logits \
--tune_config=source/tool/nasnetalarge_stanforddogs120_tune_coarse.yaml
```

Evaluation

```
(mkdir ~/demo/model/nasnet_A_large_StanfordDogs120-20190306;curl https://s3-us-west-2.
↪amazonaws.com/lambdalabs-files/nasnet_A_large_StanfordDogs120-20190306.tar.gz | tar␣
↪xvz -C ~/demo/model)


python demo/image/image_classification.py \
--mode=eval \
--model_dir=~/demo/model/nasnet_A_large_StanfordDogs120-20190306 \
--network=nasnet_A_large \
--augmenter=inception_augmenter \
--batch_size_per_gpu=16 --epochs=1 \
--num_classes=120 --image_width=331 --image_height=331 \
eval_args \
--dataset_meta=~/demo/data/StanfordDogs120/eval.csv
```

1.3.3 Image Segmenation

1.3.3.1 Fully Convolutional Networks

- *Download CamVid Dataset*

1.3.3.1.1 Download CamVid Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/camvid.tar.gz \
--data_dir=~/demo/data
```

1.3.3.1.2 Train from scratch

```
python demo/image/image_segmentation.py \
--mode=train \
--model_dir=~/demo/model/fcn_camvid \
--network=fcn \
--augmenter=fcn_augmenter \
--batch_size_per_gpu=16 --epochs=200 \
train_args \
--learning_rate=0.00129 --optimizer=adam \
--piecewise_boundaries=100 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/camvid/train.csv
```

1.3.3.1.3 Evaluation

```
python demo/image/image_segmentation.py \
--mode=eval \
--model_dir=~/demo/model/fcn_camvid \
--network=fcn \
--augmenter=fcn_augmenter \
--batch_size_per_gpu=4 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/camvid/val.csv
```

1.3.3.1.4 Inference

```
python demo/image/image_segmentation.py \
--mode=infer \
--model_dir=~/demo/model/fcn_camvid \
--network=fcn \
--augmenter=fcn_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
```

(continues on next page)

(continued from previous page)

```
infer_args \
--callbacks=infer_basic,infer_display_image_segmentation \
--test_samples=~/demo/data/camvid/test/0001TP_008550.png,~/demo/data/camvid/test/
↪Seq05VD_f02760.png,~/demo/data/camvid/test/Seq05VD_f04650.png,~/demo/data/camvid/
↪test/Seq05VD_f05100.png
```

1.3.3.1.5 Hyper-Parameter Tuning

```
python demo/image/image_segmentation.py \
--mode=tune \
--model_dir=~/demo/model/fcn_camvid \
--network=fcn \
--augmenter=fcn_augmenter \
--batch_size_per_gpu=16 \
tune_args \
--train_dataset_meta=~/demo/data/camvid/train.csv \
--eval_dataset_meta=~/demo/data/camvid/val.csv \
--tune_config=source/tool/fcn_camvid_tune_coarse.yaml
```

1.3.3.1.6 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/fcn_camvid_20190125.tar.gz |␣
↪tar xvz -C ~/demo/model
```

Evaluate

```
python demo/image/image_segmentation.py \
--mode=eval \
--model_dir=~/demo/model/fcn_camvid_20190125 \
--network=fcn \
--augmenter=fcn_augmenter \
--gpu_count=1 --batch_size_per_gpu=4 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/camvid/val.csv
```

1.3.3.1.7 Export

```
python demo/image/image_segmentation.py \
--mode=export \
--model_dir=~/demo/model/fcn_camvid_20190125 \
--network=fcn \
--augmenter=fcn_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_classes
```

1.3.3.1.8 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_segmentation \
--mount type=bind,source=model_dir/export,target=/models/segmenation \
-e MODEL_NAME=segmentation -t tensorflow/serving:latest-gpu &

python client/image_segmenation_client.py --image_path=path_to_image
```

1.3.3.2 U-Net

- *Download CamVid Dataset*
- *Train from scratch*
- *Evaluation*
- *Inference*
- *Hyper-Parameter Tuning*
- *Evaluate Pre-trained model*
- *Export*
- *Serve*

1.3.3.2.1 Download CamVid Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/camvid.tar.gz \
--data_dir=~/demo/data
```

1.3.3.2.2 Train from scratch

```
python demo/image/image_segmentation.py \
--mode=train \
--model_dir=~/demo/model/unet_camvid \
--network=unet \
--augmenter=unet_augmenter \
--gpu_count=1 --batch_size_per_gpu=16 --epochs=200 \
train_args \
--learning_rate=0.00129 --optimizer=adam \
--piecewise_boundaries=100 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/camvid/train.csv
```

1.3.3.2.3 Evaluation

```
python demo/image/image_segmentation.py \
--mode=eval \
--model_dir=~/demo/model/unet_camvid \
```

(continues on next page)

(continued from previous page)

```
--network=unet \
--augmenter=unet_augmenter \
--batch_size_per_gpu=4 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/camvid/val.csv
```

1.3.3.2.4 Inference

```
python demo/image/image_segmentation.py \
--mode=infer \
--model_dir=~/demo/model/unet_camvid \
--network=unet \
--augmenter=unet_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
infer_args \
--callbacks=infer_basic,infer_display_image_segmentation \
--test_samples=~/demo/data/camvid/test/0001TP_008550.png,~/demo/data/camvid/test/
↪Seq05VD_f02760.png,~/demo/data/camvid/test/Seq05VD_f04650.png,~/demo/data/camvid/
↪test/Seq05VD_f05100.png
```

1.3.3.2.5 Hyper-Parameter Tuning

```
python demo/image/image_segmentation.py \
--mode=tune \
--model_dir=~/demo/model/unet_camvid \
--network=unet \
--augmenter=unet_augmenter \
--batch_size_per_gpu=16 \
tune_args \
--train_dataset_meta=~/demo/data/camvid/train.csv \
--eval_dataset_meta=~/demo/data/camvid/val.csv \
--tune_config=source/tool/unet_camvid_tune_coarse.yaml
```

1.3.3.2.6 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/unet_camvid_20190125.tar.gz␣
↪| tar xvz -C ~/demo/model
```

Evaluate

```
python demo/image/image_segmentation.py \
--mode=eval \
--model_dir=~/demo/model/unet_camvid_20190125 \
--network=unet \
--augmenter=fcn_augmenter \
--gpu_count=1 --batch_size_per_gpu=4 --epochs=1 \
eval_args \
--dataset_meta=~/demo/data/camvid/val.csv
```

1.3.3.2.7 Export

```
python demo/image/image_segmentation.py \
--mode=export \
--model_dir=~/demo/model/unet_camvid_20190125 \
--network=unet \
--augmenter=unet_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_classes
```

1.3.3.2.8 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_segmentation \
--mount type=bind,source=model_dir/export,target=/models/segmenation \
-e MODEL_NAME=segmentation -t tensorflow/serving:latest-gpu &

python client/image_segmenation_client.py --image_path=path_to_image
```

1.3.4 Style Transfer

1.3.4.1 Fast Neural Style

- *Download VGG backbone*
- *Download MSCOCO (sub) Dataset*
- *Train from scratch*
- *Evaluation*
- *Inference*
- *Hyper-Parameter Tuning*
- *Evaluate Pre-trained model*
- *Export*

1.3.4.1.1 Download VGG backbone

```
(mkdir ~/demo/model/vgg_19_2016_08_28;curl http://download.tensorflow.org/models/vgg_
↪19_2016_08_28.tar.gz | tar xvz -C ~/demo/model/vgg_19_2016_08_28)
```

1.3.4.1.2 Download MSCOCO (sub) Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/mscoco_fns.tar.gz \
--data_dir=~/demo/data
```

1.3.4.1.3 Train from scratch

```
python demo/image/style_transfer.py \
--mode=train \
--model_dir=~/demo/model/fns_gothic \
--network=fns \
--augmenter=fns_augmenter \
--batch_size_per_gpu=8 --epochs=100 \
train_args \
--learning_rate=0.00185 --optimizer=rmsprop \
--piecewise_boundaries=90 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/mscoco_fns/train2014.csv \
--summary_names=loss,learning_rate \
--callbacks=train_basic,train_loss,train_speed,train_summary \
--trainable_vars=FNS
```

1.3.4.1.4 Evaluation

```
python demo/image/style_transfer.py \
--mode=eval \
--model_dir=~/demo/model/fns_gothic \
--network=fns \
--augmenter=fns_augmenter \
--batch_size_per_gpu=4 --epochs=1 \
eval_args \
--callbacks=eval_basic,eval_loss,eval_speed,eval_summary \
--dataset_meta=~/demo/data/mscoco_fns/eval2014.csv
```

1.3.4.1.5 Inference

```
python demo/image/style_transfer.py \
--mode=infer \
--model_dir=~/demo/model/fns_gothic \
--network=fns \
--augmenter=fns_augmenter \
--batch_size_per_gpu=1 --epochs=1 --gpu_count=1 \
infer_args \
--callbacks=infer_basic,infer_display_style_transfer \
--test_samples=~/demo/data/mscoco_fns/train2014/COCO_train2014_000000003348.jpg,~/
↪demo/data/mscoco_fns/val2014/COCO_val2014_000000138954.jpg,~/demo/data/mscoco_fns/
↪val2014/COCO_val2014_000000015070.jpg
```

1.3.4.1.6 Hyper-Parameter Tuning

```
python demo/image/style_transfer.py \
--mode=tune \
--model_dir=~/demo/model/fns_gothic \
--network=fns \
--augmenter=fns_augmenter \
--batch_size_per_gpu=4 \
tune_args \
--train_dataset_meta=~/demo/data/mscoco_fns/train2014.csv \
--eval_dataset_meta=~/demo/data/mscoco_fns/eval2014.csv \
--train_callbacks=train_basic,train_loss,train_speed,train_summary \
--eval_callbacks=eval_basic,eval_loss,eval_speed,eval_summary \
--tune_config=source/tool/fns_gothic_tune_coarse.yaml \
--trainable_vars=FNS
```

1.3.4.1.7 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/fns_gothic_20190126.tar.gz |␣
↪tar xvz -C ~/demo/model
```

Evaluate

```
python demo/image/style_transfer.py \
--mode=infer \
--model_dir=~/demo/model/fns_gothic_20190126 \
--network=fns \
--augmenter=fns_augmenter \
--batch_size_per_gpu=1 --epochs=1 --gpu_count=1 \
infer_args \
--callbacks=infer_basic,infer_display_style_transfer \
--test_samples=~/demo/data/mscoco_fns/train2014/COCO_train2014_000000003348.jpg,~/
↪demo/data/mscoco_fns/val2014/COCO_val2014_000000138954.jpg,~/demo/data/mscoco_fns/
↪val2014/COCO_val2014_000000015070.jpg
```

1.3.4.1.8 Export

:: python demo/image/style_transfer.py –mode=export –model_dir=~/demo/model/fns_gothic_20190126 –network=fns –augmenter=fns_augmenter –gpu_count=1 –batch_size_per_gpu=1 –epochs=1 export_args –export_dir=export –export_version=1 –input_ops=input_image –output_ops=output_image

1.3.4.1.9 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_styletransfer \
--mount type=bind,source=model_dir/export,target=/models/styletransfer \
-e MODEL_NAME=styletransfer -t tensorflow/serving:latest-gpu &

python client/style_transfer_client.py --image_path=path_to_image
```

1.3.5 Text Generation

1.3.5.1 Char RNN

1.3.5.1.1 Download Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/shakespeare.tar.gz \
--data_dir=~/demo/data/
```

1.3.5.1.2 Build Vocabulary

```
python demo/text/preprocess/build_vocab_basic.py \
--input_file=~/demo/data/shakespeare/shakespeare_input.txt \
--output_vocab=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--unit=char \
--loader=char_basic
```

1.3.5.1.3 Train from scratch

```
python demo/text/text_generation.py \
--mode=train \
--model_dir=~/demo/model/char_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=256 --epochs=10 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--encode_method=basic \
--unit=char \
train_args \
--learning_rate=0.002 --optimizer=adam \
--piecewise_boundaries=5 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.1.4 Evaluation

```
python demo/text/text_generation.py \
--mode=eval \
--model_dir=~/demo/model/char_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=32 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--encode_method=basic \
--unit=char \
eval_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.1.5 Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/char_rnn_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=char \
--starter=V \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.1.6 Hyper-Parameter Tuning

```
python demo/text/text_generation.py \
--mode=tune \
--model_dir=~/demo/model/char_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=128 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=char \
tune_args \
--train_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--eval_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--tune_config=source/tool/rnn_basic_shakespeare_tune_coarse.yaml
```

1.3.5.1.7 Inference Using Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/char_rnn_shakespeare-
↪20190303.tar.gz | tar xvz -C ~/demo/model
```

Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/char_rnn_shakespeare-20190303 \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=char \
--starter=V \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.1.8 Export

```
python demo/text/text_generation.py \
--mode=export \
--model_dir=~/demo/model/char_rnn_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=char \
export_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--export_dir=export \
--export_version=1 \
--input_ops=input_item,RNN/c0,RNN/h0,RNN/c1,RNN/h1 \
--output_ops=output_logits,output_last_state
```

1.3.5.1.9 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textgeneration \
--mount type=bind,source=/home/chuan/demo/model/char_rnn_shakespeare/export,target=/
↪models/textgeneration \
-e MODEL_NAME=textgeneration -t tensorflow/serving:latest-gpu &


python client/text_generation_client.py \
--vocab_file=~/demo/data/shakespeare/shakespeare_char_basic.vocab \
--vocab_top_k=-1 \
--vocab_format=pickle \
--unit=char --starter=V --length=1000 --softmax_temperature=1.0
```

1.3.5.2 Word RNN

1.3.5.2.1 Download Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/shakespeare.tar.gz \
--data_dir=~/demo/data/
```

1.3.5.2.2 Build Vocabulary

```
python demo/text/preprocess/build_vocab_basic.py \
--input_file=~/demo/data/shakespeare/shakespeare_input.txt \
--output_vocab=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--unit=word \
--loader=word_basic
```

1.3.5.2.3 Train from scratch

```
python demo/text/text_generation.py \
--mode=train \
--model_dir=~/demo/model/word_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=128 --epochs=10 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--encode_method=basic \
--unit=word \
train_args \
--learning_rate=0.002 --optimizer=adam \
--piecewise_boundaries=5 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.2.4 Evaluation

```
python demo/text/text_generation.py \
--mode=eval \
--model_dir=~/demo/model/word_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=32 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.2.5 Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/word_rnn_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=word \
--starter=The \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.2.6 Hyper-Parameter Tuning

```
python demo/text/text_generation.py \
--mode=tune \
--model_dir=~/demo/model/word_rnn_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=128 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=word \
tune_args \
--train_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--eval_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--tune_config=source/tool/rnn_basic_shakespeare_tune_coarse.yaml
```

1.3.5.2.7 Inference Using Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/word_rnn_shakespeare-
↪20190303.tar.gz | tar xvz -C ~/demo/model
```

Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/word_rnn_shakespeare-20190303 \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=word \
--starter=The \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.2.8 Export

```
python demo/text/text_generation.py \
--mode=export \
--model_dir=~/demo/model/word_rnn_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=-1 \
--unit=word \
export_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--export_dir=export \
--export_version=1 \
--input_ops=input_item,RNN/c0,RNN/h0,RNN/c1,RNN/h1 \
--output_ops=output_logits,output_last_state
```

1.3.5.2.9 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textgeneration \
--mount type=bind,source=/home/ubuntu/demo/model/word_rnn_shakespeare/export,target=/
↪models/textgeneration \
-e MODEL_NAME=textgeneration -t tensorflow/serving:latest-gpu &


python client/text_generation_client.py \
--vocab_file=~/demo/data/shakespeare/shakespeare_word_basic.vocab \
--vocab_top_k=-1 \
--vocab_format=pickle \
--unit=word --starter=KING --length=256 --softmax_temperature=1.0
```

1.3.5.3 Word RNN with Glove Embedding

1.3.5.3.1 Download Dataset

```
python demo/download_data.py \
--data_url=https://s3-us-west-2.amazonaws.com/lambdalabs-files/shakespeare.tar.gz \
--data_dir=~/demo/data/
```

1.3.5.3.2 Download Glove Embedding

```
wget http://nlp.stanford.edu/data/glove.6B.zip && unzip glove.6B.zip -d ~/demo/model/
↪glove.6B && rm glove.6B.zip
```

1.3.5.3.3 Train from scratch

```
python demo/text/text_generation.py \
--mode=train \
--model_dir=~/demo/model/word_rnn_glove_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=32 --epochs=10 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
train_args \
--learning_rate=0.002 --optimizer=adam \
--piecewise_boundaries=5 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.3.4 Evaluation

```
python demo/text/text_generation.py \
--mode=eval \
--model_dir=~/demo/model/word_rnn_glove_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=32 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt
```

1.3.5.3.5 Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/word_rnn_glove_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
--starter=king \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.3.6 Hyper-Parameter Tuning

```
python demo/text/text_generation.py \
--mode=tune \
--model_dir=~/demo/model/word_rnn_glove_shakespeare \
--network=rnn_basic \
--batch_size_per_gpu=128 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
tune_args \
--train_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--eval_dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--tune_config=source/tool/rnn_basic_shakespeare_tune_coarse.yaml
```

1.3.5.3.7 Inference Using Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/word_rnn_glove_shakespeare-
↪20190303.tar.gz | tar xvz -C ~/demo/model
```

Inference

```
python demo/text/text_generation.py \
--mode=infer \
--model_dir=~/demo/model/word_rnn_glove_shakespeare-20190303 \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
--starter=king \
--softmax_temperature=1.0 \
infer_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--callbacks=infer_basic,infer_display_text_generation
```

1.3.5.3.8 Export

```
python demo/text/text_generation.py \
--mode=export \
--model_dir=~/demo/model/word_rnn_glove_shakespeare \
--network=rnn_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
export_args \
--dataset_meta=~/demo/data/shakespeare/shakespeare_input.txt \
--export_dir=export \
--export_version=1 \
--input_ops=input_item,RNN/c0,RNN/h0,RNN/c1,RNN/h1 \
--output_ops=output_logits,output_last_state
```

1.3.5.3.9 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textgeneration \
--mount type=bind,source=/home/ubuntu/demo/model/word_rnn_glove_shakespeare/export,
↪target=/models/textgeneration \
-e MODEL_NAME=textgeneration -t tensorflow/serving:latest-gpu &


python client/text_generation_client.py \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_top_k=40000 \
```

(continues on next page)

(continued from previous page)

```
--vocab_format=txt \
--unit=word --starter=the --length=256 --softmax_temperature=1.0
```

1.3.6 Text Classification

1.3.6.1 Sequence-to-label Basic

1.3.6.1.1 Download Dataset

```
python demo/download_data.py \
--data_url=http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz \
--data_dir=~/demo/data/
```

1.3.6.1.2 Preprocess Dataset

```
python demo/text/preprocess/preprocess_aclImdb_v1.py \
--remove_punctuation=False
```

1.3.6.1.3 Build Vocabulary

```
python demo/text/preprocess/build_vocab_aclImdb_v1.py \
--input_file=~/demo/data/IMDB/train.csv \
--output_vocab=~/demo/data/IMDB/imdb_word_basic.vocab \
--unit=word \
--loader=imdb_loader
```

1.3.6.1.4 Train from scratch

```
python demo/text/text_classification.py \
--mode=train \
--model_dir=~/demo/model/seq2label_basic_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=4 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
--lr_method=linear_plus_warmup \
train_args \
--learning_rate=0.00014 --optimizer=adam \
--dataset_meta=~/demo/data/IMDB/train.csv
```

1.3.6.1.5 Evaluation

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_basic_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=1 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.1.6 Inference

```
python demo/text/text_classification.py \
--mode=infer \
--model_dir=~/demo/model/seq2label_basic_Imdb \
--network=seq2label_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
infer_args \
--callbacks=infer_basic,infer_display_text_classification \
--test_samples="This movie is awesome."#"This movie is bad."#"This movie has an␣
↪unusual taste."#"It is not clear what this movie is about."#"This is not a very␣
↪good movie."#"I saw this at the premier at TIFF and was thrilled to learn the story␣
↪is about a real friendship." \
--splitter=#
```

1.3.6.1.7 Hyper-Parameter Tuning

```
python demo/text/text_classification.py \
--mode=tune \
--model_dir=~/demo/model/seq2label_basic_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=32 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
--lr_method=linear_plus_warmup \
tune_args \
--train_dataset_meta=~/demo/data/IMDB/train.csv \
--eval_dataset_meta=~/demo/data/IMDB/test.csv \
--tune_config=source/tool/seq2label_basic_IMDB_tune_coarse.yaml
```

1.3.6.1.8 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/seq2label_basic_Imdb-
↪20190303.tar.gz | tar xvz -C ~/demo/model
```

Evaluate

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_basic_Imdb-20190303 \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=1 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.1.9 Export

```
python demo/text/text_classification.py \
--mode=export \
--model_dir=~/demo/model/seq2label_basic_Imdb \
--network=seq2label_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic \
--lr_method=linear_plus_warmup \
export_args \
--dataset_meta=~/demo/data/IMDB/train_clean.csv \
```

(continues on next page)

(continued from previous page)

```
--export_dir=export \
--export_version=1 \
--input_ops=input_text,input_mask \
--output_ops=output_probabilities
```

1.3.6.1.10 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textclassification \
--mount type=bind,source=/home/chuan/demo/model/seq2label_basic_Imdb/export,target=/
↪models/textclassification \
-e MODEL_NAME=textclassification -t tensorflow/serving:latest-gpu &

python client/text_classification_client.py \
--vocab_file=~/demo/data/IMDB/imdb_word_basic.vocab \
--vocab_format=pickle \
--vocab_top_k=40000 \
--encode_method=basic
```

1.3.6.2 Sequence-to-label Glove

- *Download Dataset*
- *Preprocess Dataset*
- *Download Glove Embedding*
- *Train from scratch*
- *Evaluation*
- *Inference*
- *Hyper-Parameter Tuning*
- *Evaluate Pre-trained model*
- *Export*
- *Serve*

1.3.6.2.1 Download Dataset

```
python demo/download_data.py \
--data_url=http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz \
--data_dir=~/demo/data/
```

1.3.6.2.2 Preprocess Dataset

```
python demo/text/preprocess/preprocess_aclImdb_v1.py \
--remove_punctuation=False
```

1.3.6.2.3 Download Glove Embedding

```
wget http://nlp.stanford.edu/data/glove.6B.zip && unzip glove.6B.zip -d ~/demo/model/
↪glove.6B && rm glove.6B.zip
```

1.3.6.2.4 Train from scratch

```
python demo/text/text_classification.py \
--mode=train \
--model_dir=~/demo/model/seq2label_glove_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=4 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
--lr_method=linear_plus_warmup \
train_args \
--learning_rate=0.002 --optimizer=adam \
--dataset_meta=~/demo/data/IMDB/train.csv
```

1.3.6.2.5 Evaluation

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_glove_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.2.6 Inference

```
python demo/text/text_classification.py \
--mode=infer \
--model_dir=~/demo/model/seq2label_glove_Imdb \
--network=seq2label_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
infer_args \
--callbacks=infer_basic,infer_display_text_classification \
```

(continues on next page)

(continued from previous page)

```
--test_samples="This movie is awesome."#"This movie is bad."#"This movie has an␣
↪unusual taste."#"It is not clear what this movie is about."#"This is not a very␣
↪good movie."#"I saw this at the premier at TIFF and was thrilled to learn the story␣
↪is about a real friendship." \
--splitter=#
```

1.3.6.2.7 Hyper-Parameter Tuning

```
python demo/text/text_classification.py \
--mode=tune \
--model_dir=~/demo/model/seq2label_glove_Imdb \
--network=seq2label_basic \
--batch_size_per_gpu=32 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--lr_method=linear_plus_warmup \
tune_args \
--train_dataset_meta=~/demo/data/IMDB/train.csv \
--eval_dataset_meta=~/demo/data/IMDB/test.csv \
--tune_config=source/tool/seq2label_glove_IMDB_tune_coarse.yaml
```

1.3.6.2.8 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/seq2label_glove_Imdb-
↪20190303.tar.gz | tar xvz -C ~/demo/model
```

Evaluate

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_glove_Imdb-20190303 \
--network=seq2label_basic \
--batch_size_per_gpu=128 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.2.9 Export

```
python demo/text/text_classification.py \
--mode=export \
--model_dir=~/demo/model/seq2label_glove_Imdb \
```

(continues on next page)

(continued from previous page)

```
--network=seq2label_basic \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic \
--lr_method=linear_plus_warmup \
export_args \
--dataset_meta=~/demo/data/IMDB/train.csv \
--export_dir=export \
--export_version=1 \
--input_ops=input_text,input_mask \
--output_ops=output_probabilities
```

1.3.6.2.10 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textclassification \
--mount type=bind,source=/home/chuan/demo/model/seq2label_glove_Imdb/export,target=/
↪models/textclassification \
-e MODEL_NAME=textclassification -t tensorflow/serving:latest-gpu &

python client/text_classification_client.py \
--vocab_file=~/demo/model/glove.6B/glove.6B.200d.txt \
--vocab_format=txt \
--vocab_top_k=40000 \
--encode_method=basic
```

1.3.6.3 Sequence-to-label BERT

- *Download Dataset*
- *Preprocess Dataset*
- *Download Pre-trained BERT model*
- *Train from scratch*
- *Evaluation*
- *Inference*
- *Hyper-Parameter Tuning*
- *Evaluate Pre-trained model*
- *Export*
- *Serve*

1.3.6.3.1 Download Dataset

```
python demo/download_data.py \
--data_url=http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz \
--data_dir=~/demo/data/
```

1.3.6.3.2 Preprocess Dataset

```
python demo/text/preprocess/preprocess_aclImdb_v1.py \
--remove_punctuation=False
```

1.3.6.3.3 Download Pre-trained BERT model

```
wget https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.
↪zip && unzip uncased_L-12_H-768_A-12.zip -d ~/demo/model && rm uncased_L-12_H-768_A-
↪12.zip
```

1.3.6.3.4 Train from scratch

```
python demo/text/text_classification.py \
--mode=train \
--model_dir=~/demo/model/seq2label_bert_Imdb \
--network=seq2label_bert \
--batch_size_per_gpu=16 --epochs=4 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
--unit=word \
--lr_method=linear_plus_warmup \
train_args \
--learning_rate=0.00002 --optimizer=custom \
--piecewise_boundaries=1 \
--piecewise_lr_decay=1.0,0.1 \
--dataset_meta=~/demo/data/IMDB/train.csv \
--pretrained_model=~/demo/model/uncased_L-12_H-768_A-12/bert_model.ckpt \
--skip_pretrained_var=classification/output_weights,classification/output_bias,global_
↪step,power,adam
```

1.3.6.3.5 Evaluation

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_bert_Imdb \
--network=seq2label_bert \
--batch_size_per_gpu=16 --epochs=1 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.3.6 Inference

```
python demo/text/text_classification.py \
--mode=infer \
--model_dir=~/demo/model/seq2label_bert_Imdb \
--network=seq2label_bert \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
--unit=word \
infer_args \
--callbacks=infer_basic,infer_display_text_classification \
--test_samples="This movie is awesome."#"This movie is bad."#"This movie has an␣
↪unusual taste."#"It is not clear what this movie is about."#"This is not a very␣
↪good movie."#"I saw this at the premier at TIFF and was thrilled to learn the story␣
↪is about a real friendship." \
--splitter=#
```

1.3.6.3.7 Hyper-Parameter Tuning

```
python demo/text/text_classification.py \
--mode=tune \
--model_dir=~/demo/model/seq2label_bert_Imdb \
--network=seq2label_bert \
--batch_size_per_gpu=16 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
--unit=word \
--lr_method=linear_plus_warmup \
tune_args \
--pretrained_model=~/demo/model/uncased_L-12_H-768_A-12/bert_model.ckpt \
--skip_pretrained_var=classification/output_weights,classification/output_bias,global_
↪step,power,adam \
--train_dataset_meta=~/demo/data/IMDB/train.csv \
--eval_dataset_meta=~/demo/data/IMDB/test.csv \
--tune_config=source/tool/seq2label_bert_IMDB_tune_coarse.yaml
```

1.3.6.3.8 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/seq2label_bert_Imdb-20190303.
↪tar.gz | tar xvz -C ~/demo/model
```

Evaluate

```
python demo/text/text_classification.py \
--mode=eval \
--model_dir=~/demo/model/seq2label_bert_Imdb-20190303 \
```

(continues on next page)

(continued from previous page)

```
--network=seq2label_bert \
--batch_size_per_gpu=16 --epochs=1 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
--unit=word \
eval_args \
--dataset_meta=~/demo/data/IMDB/test.csv
```

1.3.6.3.9 Export

```
python demo/text/text_classification.py \
--mode=export \
--model_dir=~/demo/model/seq2label_bert_Imdb \
--network=seq2label_bert \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert \
export_args \
--dataset_meta=~/demo/data/IMDB/train.csv \
--export_dir=export \
--export_version=1 \
--input_ops=input_text,input_mask \
--output_ops=output_probabilities
```

1.3.6.3.10 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_textclassification \
--mount type=bind,source=/home/chuan/demo/model/seq2label_bert_Imdb/export,target=/
↪models/textclassification \
-e MODEL_NAME=textclassification -t tensorflow/serving:latest-gpu &

python client/text_classification_client.py \
--vocab_file=~/demo/model/uncased_L-12_H-768_A-12/vocab.txt \
--vocab_format=txt \
--vocab_top_k=-1 \
--encode_method=bert
```

1.3.7 Object Detection

1.3.7.1 SSD

1.3.7.1.1 Prepare

Install cocoapi.

```
git clone https://github.com/cocodataset/cocoapi.git
cd cocoapi/PythonAPI
make install
```

Download coco dataset.

- Download train2014, val2014, val2017 data and annotations.
- Uncompress them into your local machine. We use "/mnt/data/data/mscoco" as the data path in the following examples.

1.3.7.1.2 Download VGG backbone

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/VGG_16_reduce.tar.gz | tar␣
↪xvz -C ~/demo/model
```

1.3.7.1.3 Train SSD from scratch on MSCOCO

```
python demo/image/object_detection.py \
--mode=train --model_dir=~/demo/model/ssd300_mscoco \
--network=ssd300 --augmenter=ssd_augmenter --batch_size_per_gpu=16 --epochs=100 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=300 \
--feature_net=vgg_16_reduced --feature_net_path=demo/model/VGG_16_reduce/VGG_16_
↪reduce.p \
train_args --learning_rate=0.001 --optimizer=momentum --piecewise_boundaries=60,80 \
--piecewise_lr_decay=1.0,0.1,0.01 --dataset_meta=train2014,valminusminival2014 \
--callbacks=train_basic,train_loss,train_speed,train_summary \
--skip_l2_loss_vars=l2_norm_scaler --summary_names=loss,learning_rate,class_losses,
↪bboxes_losses

python demo/image/object_detection.py \
--mode=train --model_dir=~/demo/model/ssd512_mscoco \
--network=ssd512 --augmenter=ssd_augmenter --batch_size_per_gpu=16 --epochs=100 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=512 \
--feature_net=vgg_16_reduced --feature_net_path=demo/model/VGG_16_reduce/VGG_16_
↪reduce.p \
train_args --learning_rate=0.001 --optimizer=momentum --piecewise_boundaries=60,80 \
--piecewise_lr_decay=1.0,0.1,0.01 --dataset_meta=train2014,valminusminival2014 \
```

(continues on next page)

(continued from previous page)

```
--callbacks=train_basic,train_loss,train_speed,train_summary \
--skip_l2_loss_vars=l2_norm_scaler --summary_names=loss,learning_rate,class_losses,
↪bboxes_losses
```

1.3.7.1.4 Evaluate SSD on MSCOCO

```
python demo/image/object_detection.py \
--mode=eval \
--model_dir=~/demo/model/ssd300_mscoco \
--network=ssd300 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=8 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco \
--num_classes=81 --resolution=300 --confidence_threshold=0.01 \
--feature_net=vgg_16_reduced \
eval_args --dataset_meta=val2017 --reduce_ops=False --callbacks=eval_basic,eval_speed,
↪eval_mscoco

python demo/image/object_detection.py \
--mode=eval \
--model_dir=~/demo/model/ssd512_mscoco \
--network=ssd512 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=8 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco \
--num_classes=81 --resolution=512 --confidence_threshold=0.01 \
--feature_net=vgg_16_reduced \
eval_args --dataset_meta=val2017 --reduce_ops=False --callbacks=eval_basic,eval_speed,
↪eval_mscoco
```

1.3.7.1.5 Inference

```
python demo/image/object_detection.py \
--mode=infer --model_dir=~/demo/model/ssd300_mscoco \
--network=ssd300 --augmenter=ssd_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=300 --confidence_
↪threshold=0.5 \
--feature_net=vgg_16_reduced \
infer_args --callbacks=infer_basic,infer_display_object_detection \
--test_samples=/mnt/data/data/mscoco/val2014/COCO_val2014_000000000042.jpg,/mnt/data/
↪data/mscoco/val2014/COCO_val2014_000000000073.jpg,/mnt/data/data/mscoco/val2014/
↪COCO_val2014_000000000074.jpg,/mnt/data/data/mscoco/val2014/COCO_val2014_
↪000000000133.jpg

python demo/image/object_detection.py \
--mode=infer --model_dir=~/demo/model/ssd512_mscoco \
--network=ssd512 --augmenter=ssd_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=512 --confidence_
↪threshold=0.5 \
--feature_net=vgg_16_reduced \
```

(continues on next page)

(continued from previous page)

```
infer_args --callbacks=infer_basic,infer_display_object_detection \
--test_samples=/mnt/data/data/mscoco/val2014/COCO_val2014_000000000042.jpg,/mnt/data/
↪data/mscoco/val2014/COCO_val2014_000000000073.jpg,/mnt/data/data/mscoco/val2014/
↪COCO_val2014_000000000074.jpg,/mnt/data/data/mscoco/val2014/COCO_val2014_
↪000000000133.jpg
```

1.3.7.1.6 Hyper-Parameter Tuning

```
python demo/image/object_detection.py \
--mode=tune \
--model_dir=~/demo/model/ssd300mscoco \
--network=ssd300 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=16 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=300 \
--feature_net=vgg_16_reduced --feature_net_path=demo/model/VGG_16_reduce/VGG_16_
↪reduce.p \
tune_args \
--train_callbacks=train_basic,train_loss,train_speed,train_summary \
--eval_callbacks=eval_basic,eval_speed,eval_mscoco \
--train_dataset_meta=train2017 \
--eval_dataset_meta=val2017 \
--tune_config=source/tool/ssd300_mscoco_tune_coarse.yaml \
--eval_reduce_ops=False \
--trainable_vars=SSD \
--skip_l2_loss_vars=l2_norm_scaler


python demo/image/object_detection.py \
--mode=tune \
--model_dir=~/demo/model/ssd512_mscoco \
--network=ssd512 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=16 \
--dataset_dir=/mnt/data/data/mscoco --num_classes=81 --resolution=512\
--feature_net=vgg_16_reduced --feature_net_path=demo/model/VGG_16_reduce/VGG_16_
↪reduce.p \
tune_args \
--train_callbacks=train_basic,train_loss,train_speed,train_summary \
--eval_callbacks=eval_basic,eval_speed,eval_mscoco \
--train_dataset_meta=train2017 \
--eval_dataset_meta=val2017 \
--tune_config=source/tool/ssd512_mscoco_tune_coarse.yaml \
--eval_reduce_ops=False \
--trainable_vars=SSD \
--skip_l2_loss_vars=l2_norm_scaler
```

1.3.7.1.7 Evaluate Pre-trained model

Download pre-trained models:

```
curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/ssd300_mscoco_20190105.tar.
↪gz | tar xvz -C ~/demo/model

curl https://s3-us-west-2.amazonaws.com/lambdalabs-files/ssd512_mscoco_20190105.tar.
↪gz | tar xvz -C ~/demo/model
```

Evaluate

```
python demo/image/object_detection.py \
--mode=eval \
--model_dir=~/demo/model/ssd300_mscoco_20190105 \
--network=ssd300 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=8 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco \
--num_classes=81 --resolution=300 --confidence_threshold=0.01 \
--feature_net=vgg_16_reduced \
eval_args --dataset_meta=val2017 --reduce_ops=False --callbacks=eval_basic,eval_speed,
↪eval_mscoco

python demo/image/object_detection.py \
--mode=eval \
--model_dir=~/demo/model/ssd512_mscoco_20190105 \
--network=ssd512 \
--augmenter=ssd_augmenter \
--batch_size_per_gpu=8 --epochs=1 \
--dataset_dir=/mnt/data/data/mscoco \
--num_classes=81 --resolution=512 --confidence_threshold=0.01 \
--feature_net=vgg_16_reduced \
eval_args --dataset_meta=val2017 --reduce_ops=False --callbacks=eval_basic,eval_speed,
↪eval_mscoco
```

1.3.7.1.8 Export

```
python demo/image/object_detection.py \
--mode=export \
--model_dir=~/demo/model/ssd300_mscoco_20190105 \
--network=ssd300 \
--augmenter=ssd_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
--num_classes=81 --resolution=300 \
--confidence_threshold 0.01 \
--feature_net=vgg_16_reduced \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_scores,output_labels,output_bboxes

python demo/image/object_detection.py \
--mode=export \
--model_dir=~/demo/model/ssd512_mscoco_20190105 \
--network=ssd512 \
--augmenter=ssd_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
```

(continues on next page)

(continued from previous page)

```
--num_classes=81 --resolution=512 \
--confidence_threshold 0.01 \
--feature_net=vgg_16_reduced \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_scores,output_labels,output_bboxes
```

1.3.7.1.9 Serve

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_ \
--mount type=bind,source=model_dir/export,target=/models/objectdetection \
-e MODEL_NAME=objectdetection -t tensorflow/serving:latest-gpu &

python client/image_segmenation_client.py --image_path=path_to_image
```

1.3.8 Model Serving

- *Install Docker (Ubuntu 18.04)*
- *Install Nvidia Docker*
- *Serve*

1.3.8.1 Install Docker (Ubuntu 18.04)

```
sudo apt-get update

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

sudo apt-get update

sudo apt-get install docker-ce=5:18.09.2~3-0~ubuntu-bionic

sudo groupadd docker
sudo usermod -aG docker $USER
```

1.3.8.2 Install Nvidia Docker

```
# If you have nvidia-docker 1.0 installed: we need to remove it and all existing GPU␣
↪containers
docker volume ls -q -f driver=nvidia-docker | xargs -r -I{} -n1 docker ps -q -a -f␣
↪volume={} | xargs -r docker rm -f
```

(continues on next page)

(continued from previous page)

```
sudo apt-get purge -y nvidia-docker

# Add the package repositories
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
  sudo apt-key add -
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update

# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd

# Test nvidia-smi with the latest official CUDA image
docker run --runtime=nvidia --rm nvidia/cuda:9.0-base nvidia-smi
```

1.3.8.3 Serve

The following three steps are used to serve the trained model:

- **Export**: The first step is to export the model as a ProtoBuffer file. For example, this is how to export a pre-trained resnet32 model for image classification:

```
python demo/image/image_classification.py \
--mode=export \
--model_dir=~/demo/model/cifar10-resnet32-20180824 \
--network=resnet32 \
--augmenter=cifar_augmenter \
--gpu_count=1 --batch_size_per_gpu=1 --epochs=1 \
export_args \
--export_dir=export \
--export_version=1 \
--input_ops=input_image \
--output_ops=output_classes
```

More examples can be found here: Image Segmentation , Object Detection, Style Transfer, Text Generation, Text Classification.

- **Run TF-Serving**. A typical example of serving the exported model is like this:

```
docker run --runtime=nvidia -p 8501:8501 \
--name tfserving_classification \
--mount type=bind,source=path_to_model_dir/export,target=/models/classification \
-e MODEL_NAME=classification -t tensorflow/serving:latest-gpu &
```

- **Run client**. To consume the service, we use a client. For example, for image classification we run the client with this command:

```
python client/image_classification_client.py --image_path=path_to_image
```

1.4 Write your own

CHAPTER 2

Documentation

2.1 API documentation