SIMON FRASER UNIVERSITY

SCHOOL OF ENGINEERING SCIENCE

MULTIMEDIA LABORATORY

DFTS Version 2: User Documentation

TM-ML-2021-001

Hans Dhondea

ashivdhondea5@gmail.com

hdhondea@sfu.ca

October 20, 2021

Contents

List of Figures

List of Abbreviations

ALTeC Adaptive Linear Tensor Completion. 6

CALTeC Content Adaptive Linear Tensor Completion. 6

CI Collaborative Intelligence. 6

DFTS Deep Feature Transmission Simulator. v, 1–3, 5, 6

DNN Deep Neural Network. iv, 1

HaLRTC High accuracy Low Rank Tensor Completion. 6

ResNet18 ResNet18 is a type of residual network commonly used for computer vision tasks. 3, 4

SFU Simon Fraser University. 1

SiLRTC Simple Low Rank Tensor Completion. 6

List of Symbols

$\mathbf{c}$ Channel of shape $h \times w$ in a tensor $\mathcal{X}$. v, 3

h Height of a channel $\mathbf{c}$ in a tensor $\mathcal{X}$. v, 3

$\mathbf{x}$ Packet of shape $r_{\text{pkt}} \times w$ in DFTS. $\mathbf{x}_i^{(\mathbf{c})}$ refers to the packet with index i in channel $\mathbf{c}$. v

$r_{\mathbf{pkt}}$ Number of rows of features per packet $\mathbf{x}$ in DFTS. 3, 4, 6

w Width of a channel $\mathbf{c}$ in a tensor $\mathcal{X}$. v, 3

$\mathcal{X}$ Generic tensor of appropriate dimensions, usually $\mathcal{X} \in \mathbb{R}^{h \times w \times \mathbf{c}}$. v, 3, 4

Chapter 1

Introduction

This technical report serves as user documentation for DFTS version 2. The original DFTS, developed by Harsha at SFU's Multimedia Lab in 2018, had the objective of serving as a testbed for studies in packet-based transmission of deep features over unreliable communication channels [1].

A more sophisticated simulation framework called DFTS2, presented in [2], is now compatible with TensorFlow 2. It has the following new capabilities: additional channel models and missing feature recovery methods from the recent literature. In addition to these, it can compute additional performance metrics such as Top-5 prediction accuracy for the image classification task.

1.1 What can DFTS version 2 do?

We will use "DFTS" to refer to DFTS2 from this point onward. Briefly, DFTS can split a pre-trained deep model (DNN) into a mobile sub-model and a cloud sub-model in a collaborative intelligence system. to continue.

include figures.

running dfts from the terminal. show code snippet and example yaml file.

Broadly, there are two types of experiments possible with DFTS: single-shot or demo experiments and Monte Carlo experiments. Demonstration experiments are meant to be run on a small test set to generate deep feature tensor packet visualizations whereas Monte Carlo experiments are meant to be "full-scale' experiments run on the entire test set to compute overall statistical results. Single-shot experiments run very quickly and produce qualitative results while full-scale experiments may take hours and produce quantitative results.

1.2 Overview of this document

- Chapter 2 describes in detail each component of DFTS.
- Chapter 3 explains how to run demonstration experiments.
- Chapter 4 explains how to run Monte Carlo experiments.
- Chapter 5 concludes this technical report and provides recommendations for future work.

Chapter 2

Simulator description

2.1 Introduction

This chapter discusses in detail the different components of DFTS.

2.2 Intermediate feature tensor packetization

A feature tensor with height h, width w, and the number of channels c will be denoted $\mathcal{X} \in \mathbb{R}^{h \times w \times \mathbf{c}}$. An illustration of a feature tensor extracted from layer `add_1` of ResNet18 [3] is shown in Fig. 2-1. Layer `add_1` corresponds to the output of the shortcut connection and element-wise addition applied to the final output of the `conv2_x` layers, as denoted in [3, Table 1]. Each channel in a DFTS2 tensor is partitioned into groups of r_{pkt} rows, as indicated by the dashed green lines in Fig. 2-1. Zero padding is done on the last packet of a channel if the number of channel rows (h) is not an integer multiple of the number of packet rows (r_{pkt}). Packetized tensors can be min-max quantized [4] to a user-specified number of bits/element. While it is possible to further compress the data packets by entropy coding, this was not implemented in the current version of DFTS because the focus is on error resilience, not on compression.

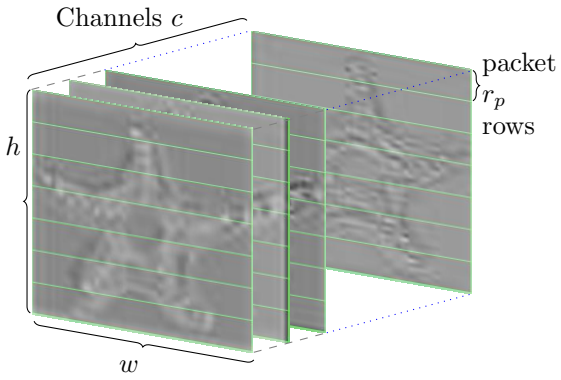

Figure 2-1: A tensor $\mathcal{X} \in \mathbb{R}^{h \times w \times c}$ from layer `add_1` of ResNet-18. Several consecutive rows in each channel form a feature data packet.

In `yaml` files, the packetization parameter may be specified as shown below

```
Transmission:
  rowsperpacket: 8
```

Packetization is implemented in `models.packetModel.py`. The user can create a packetization object with the following code snippet. The `rowsPerPacket` is the desired number of rows per packet r_{pkt} for each feature tensor channel. `deviceOut` is a deep feature tensor $\mathcal{X}$, usually output by the device sub-model.

Listing 2.1: Creating a packetization object

```
from models.packetModel import PacketModel as PacketModel
pkt_obj_model = PacketModel(rows_per_packet=rowsPerPacket,
data_tensor=deviceOut)
```

The packetized tensor data may be accessed as

Listing 2.2: Manipulating packetized tensor data

```
packetized_data = pkt_obj_model.packet_seq
unpacketized_data = pkt_obj_model.data_tensor
```

In `add_1` ResNet18 tensors, there are 64 channels, each of which is 56×56. With $r_{\text{pkt}} = 8$, we have 7 packets of 8×56 elements each. So `packetized_data` would be a 32-bit floating point `numpy` array of dimensions $7 \times 8 \times 56 \times 64$. When dealing with tensor data for a batch of k images, then `packetized_data` would be $k \times 7 \times 8 \times 56 \times 64$. `unpacketized_data` would be $k \times 56 \times 56 \times 64$.

2.3 Channel models

DFTS offers three models for the communication channel model: random loss, i.e., independent and identically distributed (iid), Gilbert-Elliott and external packet traces. Technically, the perfect channel model case, that is, no packet loss at all, is also a channel model.

In `yaml` files, the communication channel parameters may be specified as shown below.

```
Transmission:
  channel:
    GilbertChannel:
      burstLength: 1
      include: True
      lossProbability: 0.3
    RandomLossChannel:
      include: false
      lossProbability: 0
    ExternalChannel:
      include: False
      trace_dir: external_traces
```

The user can switch between channel models by editing the `include` flag. If no channel model is selected (all three `include` flags are `False`), the DFTS experiment will be run in the perfect channel model mode.

Gilbert and random loss channel models are implemented in `gbChannel.py` and `trivialChannel.py` in the `channel` directory. An object may be initialized for these two models using the following code snippet. `channel` is a dictionary created by processing the `yaml` file.

Listing 2.3: Creating a lossy channel model object

```
from runExpt.calloc import loadChannel
channel = loadChannel(channel)
```

The `channel` object is then passed to the function `transmit` which unsurprisingly transmits the packetized bitstream over a lossy channel.

Listing 2.4: Lossy channel model

```
from .simmods import *
dO, lM, rI, lI = transmit(deviceOut, channel, rowsPerPacket)
```

`deviceOut` is the device sub-model output tensor (may or may not be quantized). `d0` is the packetized (according to `rowsPerPacket`-r_{pkt}) tensor transmitted over an unreliable `channel` realization. `lM` is the loss matrix, `rI` is a list of received indices while `lI` is a list of lost indices.

We will later on discuss how the user should go about loading external packet traces.

2.3.1 Gilbert-Elliott channels

2.3.2 Packet traces

2.4 Packet loss concealment

By recovering missing packets in deep feature tensors, we are able to provide the cloud sub-model with more "complete' tensor data to exploit in order to complete the inference task. DFTS allows the user to switch on or off error concealment. The following packet loss concealment methods include

- general tensor completion methods: SiLRTC [5] and HaLRTC [5].
  These methods were developed to recover missing values in visual data. They make no assumption as to the nature of the loss (in other words, they are agnostic to the chosen packetization scheme). SiLRTC and HaLRTC adopt an iterative approach: they have to be run a number of times on the corrupted tensor to recover the missing packets. The number of iterations to be run is a parameter for the engineer to tune.
- methods which are specific to deep feature tensors in CI: ALTeC [6] and CALTeC [7].
  ALTeC was originally developed for a single row of feature data per packet scheme. DFTS provides an ALTeC modified to handle a multiple rows of feature tensor data per packet scheme.
- image inpainting-based methods, such as Navier-Stokes [8] which has been used for error concealment in CI in [9].

2.5 Simulation mode

Chapter 3

Demo experiments

3.1 Introduction

3.2 Quantization demonstration

3.3 Lossy channel demonstration

3.4 Summary

This chapter discussed the mechanics of demonstration experiments in DFTS. We now provide a list of example `yaml` files for demo experiments.

- Quantization experiment.
- Gilbert-Elliott lossy channel experiment.

Chapter 4

Monte Carlo experiments

4.1 Introduction

4.2 Quantization experiments

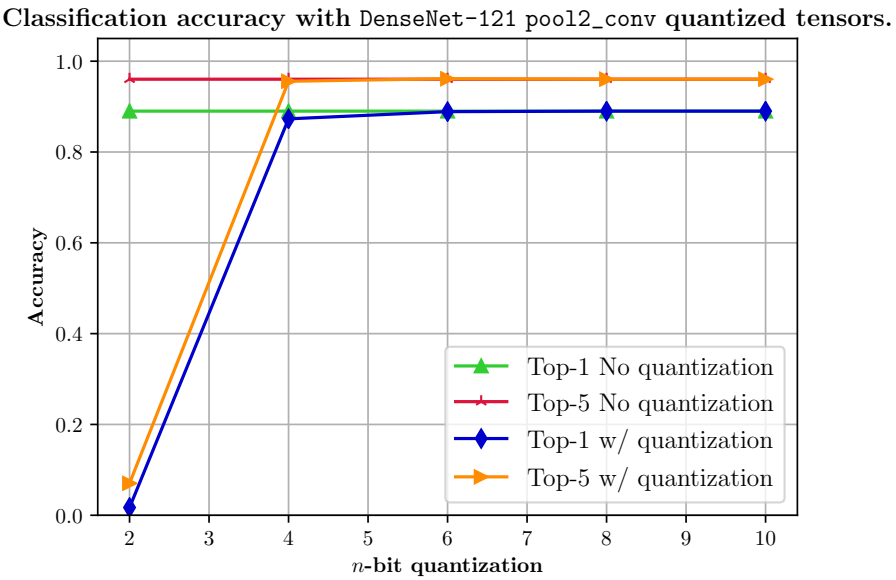

Figure 4-1: Quantization experiments with densenet.

4.3 Lossy channel experiments

As discussed in 2.3, there are two options for imperfect channel models: random loss and Gilbert-Elliott channel models. We will focus our discussion on Gilbert-Elliott channel model experiments.

4.4 Experiments with external packet traces

4.5 Summary

This chapter discussed the mechanics of Monte Carlo experiments in DFTS. We now provide a list of example `yaml` files for Monte Carlo experiments.

- Quantization experiment.
- Gilbert-Elliott lossy channel experiment.
- External packet trace experiment.

Chapter 5

Conclusions and recommendations

5.1 Conclusions

5.2 Recommendations

Bibliography

[1] H. Unnibhavi, H. Choi, S. R. Alvar, and I. V. Bajić, "DFTS: Deep feature transmission simulator." *IEEE MMSP'18* demo, 2018. https://github.com/SFU-Multimedia-Lab/DFTS. [1]

[2] A. Dhondea, R. A. Cohen, and I. V. Bajić, "DFTS2: Deep feature transmission simulator for collaborative intelligence," in *Proc. IEEE VCIP*, (Munich, Germany), 2021. to appear. [1]

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/CVF CVPR*, pp. 770–778, Jun. 2016. [3]

[4] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *Proc. IEEE ICIP'18*, pp. 3743–3747, Oct. 2018. [3]

[5] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 35, pp. 208–220, Jan. 2013. [6]

[6] L. Bragilevsky and I. V. Bajić, "Tensor completion methods for collaborative intelligence," *IEEE Access*, vol. 8, pp. 41162–41174, 2020. [6]

[7] A. Dhondea, R. A. Cohen, and I. V. Bajić, "CALTeC: Content-adaptive linear tensor completion for collaborative intelligence," in *Proc. IEEE ICIP*, (Anchorage, AK), 2021. [6]

[8] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-Stokes, fluid dynamics, and image and video inpainting," in *Proc. IEEE/CVF CVPR*, vol. 1, pp. I–355–I–362, 2001. [6]

[9] I. V. Bajić, "Latent space inpainting for loss-resilient collaborative object detection," in *Proc. IEEE ICC*, 2021. [6]