1. Transferability Analysis

The source code is of a Python API that is able to fulfil a complete workflow of:

1. training full-precision and quantized neural networks;
2. creating adversarial examples on these networks;
3. transferring adversarial examples from the network where samples are created (source) to another (target) network.

- The API is based on **Tensropack (Wu, Y. et al., 2016**) (https://github.com/tensorpack) for training and inference. Tensorpack is a training framework which is a part of **TensorFlow 1.13 (Abadi et al., 2016)** (https://www.tensorflow.org) API.

- For quantization **DoReFa-Net method (Zhou et al., 2018)** (https://arxiv.org/abs/1606.06160) is used

- For adversarial attack generation **Adversarial Robustness Toolbox** or **ART (Nicolae et al., 2019)** (https://arxiv.org/abs/1807.01069) is used.

- The API is fairly simple to use. Details on how to use the api is in "How to use the API" section in this ReadMe file.

2. Credit to the authors of the works used in this API

- Quantization is based on DoReFa-Net method as proposed in the paper: https://arxiv.org/abs/1606.06160

  Cited as:

  Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2018). DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160 [cs].

- For quantization, the library available from the authors is used. This is available at: https://github.com/tensorpack/tensorpack/tree/master/examples/DoReFa-Net

- The networks used are from the examples provided on the Tensorpack repository:

  Tensorpack cited as:

  Wu, Y. et al. (2016). Tensorpack. https://github.com/tensorpack.

- Tensorpack models are available at:
  https://github.com/tensorpack/tensorpack/tree/master/examples

- TensorFlow cited as:

  Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., . . . Zheng, X. (2016). TensorFlow: A system for large-scale machine learning, 21

- Adversarial Examples are created using Adversarial Robustness Toolbox (ART) v. 1.5.1. Official paper: https://arxiv.org/abs/1807.01069

  Cited as:

  Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I. M., & Edwards, B. (2019). Adversarial robustness toolbox v1.0.0. arXiv:1807.01069

  ART is one of the popular APIs for adversarial examples generation and supports a large number of attacks. It is open-source with large number of very well explained examples. Please visit their repository at:

  https://github.com/Trusted-AI/adversarial-robustness-toolbox

3. Replication

All data required to replicate the results are uploaded at: https://mega.nz/fm/public-links/ql8CwJxb

This includes:

1. All trained models which includes all MNIST and CIFAR10 full-precision and quantized models.
2. All analysis data which includes all transferability experiment data (adversarial accuracy of target models).
3. All generated adversarial images (image name reflects the attack hyperparameters used).

There is another ReadMe.txt file in the cloud drive which explains how to use the data.

4. How to use the API

To perform the experiments, a custom API was developed that is: (a) capable of training full-precision and quantized models; (b) using the trained models as source to create adversarial examples; and (c) performing adversarial attacks on target networks with the generated adversarial examples.

The figure below shows the usability of the API in terms of a basic workflow. As shown in the figure, given the model description (architecture) and input dataset (MNIST or CIFAR10), full-precision models can be trained. A quantized version of the network can be trained by providing the quantization hyperparameters (weight and activation bitwidths). The trained networks can be used to craft adversarial examples by specifying the attack algorithm. To study the transferability of an attack, the adversarial examples created at source can be provided as input to the target network. The adversarial inference process computes the adversarial accuracy of the target network, along with this other relevant information like the correctly and incorrectly classified samples; L_∞ and L_2 distance between the successful adversarial and clean samples; and the overall accuracy of the target (in absence of attacks) are logged. Thus, the cumulative information generated during inference is enough to observe the transferability of samples between the defined source and target network and debug (if necessary) the entire process.

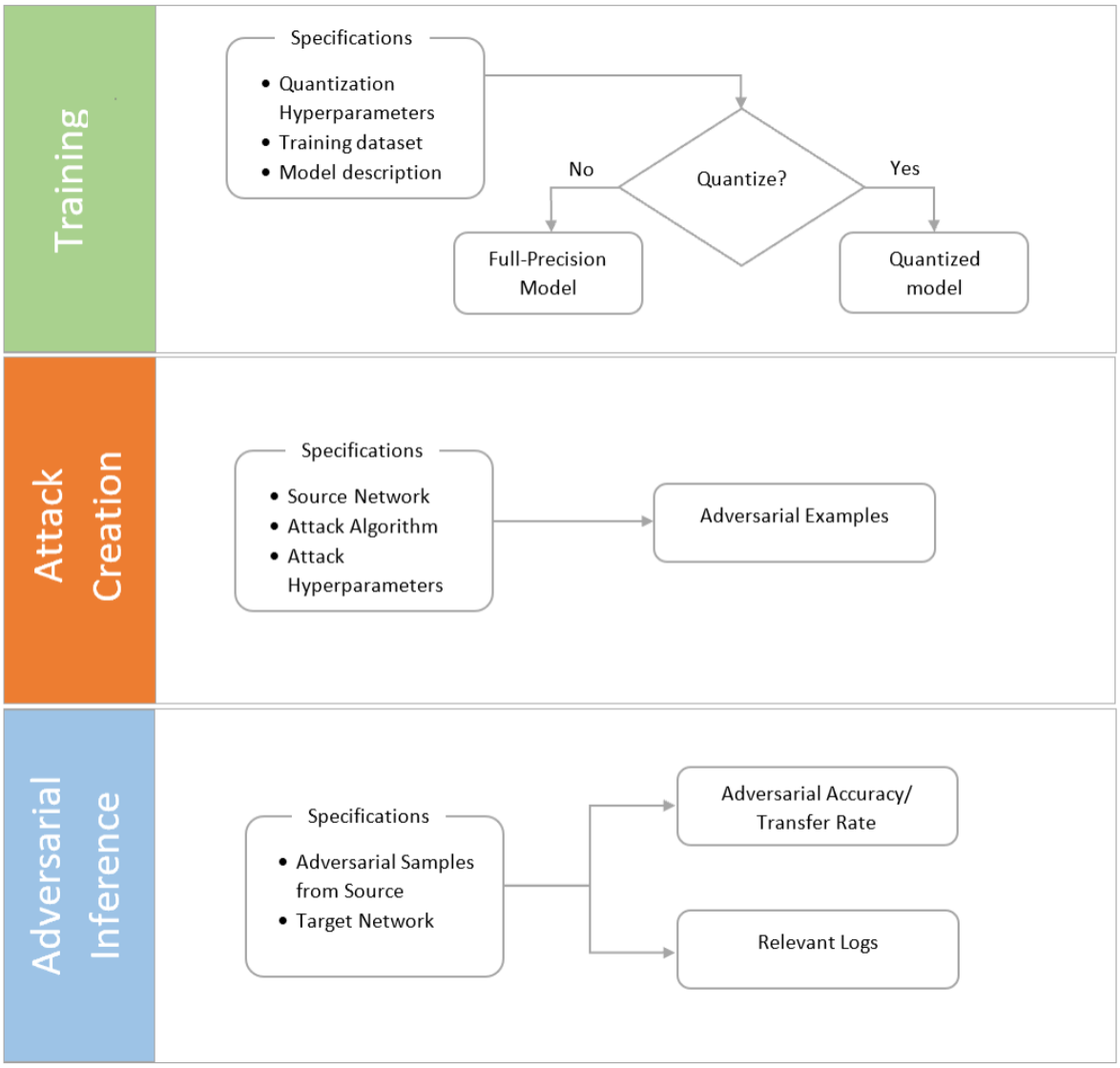

Figure 1: The custom API can be used to train full-precision and quantized networks, create adversarial examples, and transfer created adversarial examples.

The main functionality provided by the API is the ability to create adversarial examples on a network of specified bitwidth and use them to attack another (target) network. Further, the API is flexible enough such that the workflow can be expanded or automated with minimal effort. For instance, the adversarial transfers can be automated when attacks are required to be transferred from multiple sources to multiple targets by simply creating an interface to call the corresponding modules. To conduct the transferability experiments, configuration files (e.g. YAML) were used to specify details like the target models and the adversarial source. A simple parser then read the configuration file and made calls to the related modules of the API. At the end of a run, adversarial accuracy for each pair of source and target transfers was then generated.

The steps below explain how to perform training, inference, and adversarial attack creation and transfer using the API.

The API has three main modules:

1. train_net.py
2. attack_net.py (/CreateAttacks/attack_net.py)
3. inference_net.py (/Inference/inference_net.py)

These three modules can be used to train, attack, and perform inferences.

As a use case, training, performing inference, attack creation and transfer is done using YAML configuration files.

The steps below are for performing single inference/ attack transfer using the *config.yml* file and the corresponding *run_exp.py* parser. Additionally, *config_transfer_attack_multi_runs.yaml* and the corresponding parser *run_transfer_attacks_multi_runs.py* can be used to perform transfers on multiple target models at once.

For training:

In the *config.yml* file:

1. Update the "task" --> "type" to "training".
   Then in the "training-options" section:
2. Specify the dataset that the model is to be trained on in the "dataset" field
3. Specify the model to be trained in the "model" field
4. Specify the quantization bitwidth in the "precision" field.
5. To initiate the training run "*run_exp.py*" file.

For inference:

In the *config.yml* file:

1. Update the "task" --> "type" to "inference".
2. Specify the model path (path of the saved checkpoint) in the “load-model” field.
3. Name the base model in the “base-model” field, i.e., the model that was used to train, for instance, *model_a* for Mnist A model.
4. In the “images” field, specify the dataset the model was trained on, for instance MNIST.
5. Specify the precision or bitwidth of the model being inferred in the “bitwidth” within “precision”.
6. To initiate inference run “*run_exp.py*” file.

For creating adversarial examples:

In the *config.yml* file:

1. Update the "task" --> "type" to "attack"
2. In the "attack-options" section:
    i. Set the "attack-mode" to "create".

3. In the "create-attack" section:
    i. Load a saved model on which the attack is to be created in the "load-model" field. This can be a Tensorflow checkpoint.
    ii. Specify the base-model, that is the type of the model to load in the "base-model" field. For instance, model_a for Mnist A mode.
    iii. Specify the attack algorithm to use in the "algorithm" field.
    iv. Specify the dataset to create the adversarial examples in the "dataset" field. Data points will be sampled from this dataset.
    v. Specify the quantization bitwidth of the loaded model in the "precision" field.
4. To initiate the attack creation run "run_exp.py" file.

For adversarial transfer:

In the *config.yml* file:

1. Update the "task" --> "type" to "attack"
2. Then in the "attack-options" section:
    i. Set the "attack-mode" to "transfer".
3. In the "transfer-attack" section:
    i. Specify source properties in the "Source" section, this includes:
        a) Add the location of the .npz file containing adversarial examples in the "source-images" field.
        b) Specify the dataset used to create the adversarial examples in the "dataset" field.
    ii. Specify target properties in the "Target" section, this includes:

a) Load a saved model on which the attack is to be transferred (target model) in the "target-model" field. This can be a Tensorflow checkpoint.
b) Specify the base-model, that is the type of the target mode "base-model" field.
c) Specify the quantization bitwidth of the loaded model in the "precision" field.

4. To initiate the attack creation run "run_exp.py" file.

Logs are created under the “logs” folder.

5. Hyperparameter tuning

Currently, it is not possible to change the hyperparameters from the yml files.

Model training hyperparameters.

To change model training hyperparameters, please use the ***train_net.py*** file. The source is well-commented to guide where to change the parameters.

Adversarial attack algorithm hyperparameters.

To change the attack hyperparameters please use:
CreateAttacks/ create_adversarial_samples.py