

MAX78000

TensorFlow 2.3.0 Support with Keras API

September 9, 2020

Overview

This document describes the modeling of networks using **TensorFlow 2 Keras API**. The machine learning models created and trained can be ported to and executed on MAX78000. Different types of Keras models with TensorFlow are supported, including high-level sequential, functional and subclassing APIs. The following development flow has to be used:

1. Create the Keras model using supported MAX78000 TensorFlow sub-classes which reflect hardware behavior and limit operations.
2. Train the model and store the model graph and weights into a **saved_model.pb** file.
3. Use the Tensorflow to ONNX converter (**tf2onnx.convert**) to create an ONNX framework model from **saved_model.pb**.
4. Quantize the ONNX model weights and feed to MAX78000 synthesis tool (*ai8xizer*) to generate C code.
5. Compile the generated C code, and load into MAX78000 to verify.

Setup

1. Install NVIDIA GPU drivers (CUDA 10.1 or CUDA 10.2), CUDA Toolkit, CUPTI and cuDNN SDK 7.6 as described in Software requirements for TensorFlow: <https://www.tensorflow.org/install/gpu>
2. Make sure that `~/.bash_profile` includes path to CUDA and CUPTI:

```
export PATH="$PATH:/usr/local/cuda-10.1/bin"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda-10.1/extras/CUPTI/lib64"
```

3. To create a virtual environment, please refer to the section "Creating the Virtual Environment" of [1]: ../README.md, including the installation of ai8x-training and ai8x-synthesis requirements.

Supported MAX78000 TensorFlow Keras Subclasses

ai85TF.py includes a set of customized TensorFlow 2 Keras subclasses to be used by any model that is designed to run on MAX78000.

Name	Description/Keras Equivalent
Conv1D	Generic Conv1D, padding_size=0
FusedConv1D	Conv1D with activation as None, padding_size=0
FusedConv1DReLU	Conv1D with activation as 'relu', padding_size=0
FusedMaxPoolConv1D	MaxPool1D, followed by Conv1D with activation as None, padding_size=0
FusedMaxPoolConv1DReLU	MaxPool1D, followed by Conv1D with activation as 'relu', padding_size=0
FusedAvgPoolConv1D	AveragePooling1D, followed by Conv1D with activation as None, padding_size=0
FusedAvgPoolConv1DReLU	AveragePooling1D followed by Conv1D with activation as 'relu', padding_size=0
MaxPool1D	MaxPool1D
AvgPool1D	AveragePooling1D
Conv2D	Generic Conv2D, padding_size=0
FusedConv2D	Conv2D with activation as None, padding_size=0
FusedConv2DReLU	Conv2D with activation as 'relu', padding_size=0
FusedMaxPoolConv2D	MaxPool2D, followed by Conv2D with activation as None, padding_size=0
FusedMaxPoolConv2DReLU	MaxPool2D, followed by Conv2D with activation as 'relu', padding_size=0
FusedAvgPoolConv2D	AveragePooling2D, followed by Conv2D with activation as None, padding_size=0
FusedAvgPoolConv2DReLU	AveragePooling2D followed by Conv2D with activation as 'relu', padding_size=0
MaxPool2D	MaxPool2D

AvgPool2D	AveragePooling2D
Dense	Generic Dense
FusedDense	Dense with activation as None
FusedDenseReLU	Dense with activation as 'relu'

Limitations of supported operations:

Conv2D:

- Kernel sizes must be 1×1 or 3×3.
- Padding can be 0, 1, or 2 (default: padding_size = 0).
- Stride is fixed to 1. Pooling, including 1×1, can be used to achieve a stride other than 1.

Conv1D:

- Kernel sizes must be 1 through 9.
- Padding can be 0, 1, or 2 (default: padding_size = 0).
- Stride is fixed to 1. Pooling, including 1, can be used to achieve a stride other than 1.

Conv2DTranspose:

- Kernel sizes must be 3×3.
- Padding can be 0, 1, or 2 (default: padding_size = 0).
- Stride is fixed to 2

Pooling:

- Both max pooling and average pooling are available, with or without convolution.
- Pooling does not support padding.
- Pooling strides can be 1 through 16. For 2D pooling, the stride is the same for both dimensions.
- For 2D pooling, supported pooling kernel sizes are 1×1 through 16×16, including non-square kernels. 1D pooling supports kernels from 1 through 16. *Note: 1×1 kernels can be used when a convolution stride other than 1 is desired.*
- The number of input channels must not exceed 1024.
- The number of output channels must not exceed 1024.

For more details of MAX7800 hardware related limitations please check document [1].

Model Training

The following bash scripts are provided to download datasets, train the models and to convert to ONNX format:

```
train_cifar10.sh
train_kws20.sh
train_mnist.sh
train_cifar100.sh
train_fashionmnist.sh
train_rock.sh
```

Example (`train_mnist.sh`):

```
python train.py --epochs 100 --batch_size 256 --optimizer Adam --lr 0.001 --model
mnist_model --dataset mnist --save-sample 1
python -m tf2onnx.convert --saved-model export/mnist --opset 10 --output
export/mnist/saved_model.onnx
```

The script automatically downloads the corresponding dataset, processes and copies the data into `data/` if needed, and starts training. Training progress and results, including checkpoints and a sample prediction for one test data sample in HWC format will be stored in log files inside the `logs/` directory. The model graph and weights are stored as `saved_model.pb` inside the `logs/` directory, as well as in the `export/` directory.

Command-Line Arguments for Training

The following command line parameters are supported for training:

Argument	Description
--epochs	Number of training epochs (default: 100)
--batch_size	Training batch size (default: 32)
--optimizer	Optimizer type: Adam or SGD (default: Adam)
--lr	Initial learning rate (default: 0.0001). During training learning rate is adjusted according to the schedule in the model
--model	Model name
--dataset	Dataset name
--save-sample	Save input sample with specified index in <i>.npy</i> format in the <code>export/</code> folder for verification
--save-sample-per-class	Save one input sample for each class in <i>.npy</i> format in the <code>logs/</code> folder to be used for verification
--channel-first	Save sample in channel first format in multi-channel cases (default: channel last, the native format on TensorFlow), suitable to be used by the synthesis script (NCHW)
--swap	if --channel-first is selected, this option swaps order of H and W in the saved sample data. This is only needed if first layer is a conv1d (default: no swap)
--metrics	Metrics used in compiling model (default: accuracy)

Once training is complete, the model is converted to ONNX format and stored in `export/`.

Models

Model examples are located in the `models/` directory. Each model includes a TensorFlow Keras sequential model, as well as the callback function for the learning rate adjustment scheduler:

```
models/cifar10_model.py
models/cifar100_model.py
models/fashionmnist_model.py
models/kws20_model.py
models/mnist_model.py
models/rock_model.py
```

Learning Rate Scheduler

Each model script includes the *lr scheduler* callback function to be used for that model. The default schedule is *ReduceLROnPlateau* with the following parameters:

```
lr_schedule = tf.keras.callbacks.ReduceLROnPlateau(  
    monitor='val_accuracy',  
    mode='max',  
    factor=0.2,  
    patience=3,  
    verbose=1,  
    min_lr=1e-5)
```

Datasets

Dataset scripts are located in the `dataset/` directory and used to download the dataset and create a processed *.npz* dataset file (if needed) in the `data/` folder to be used by the training scripts:

```
datasets/cifar10.py  
datasets/cifar100.py  
datasets/fashionmnist.py  
datasets/kws20.py  
datasets/mnist.py  
datasets/rock.py
```

Datasets include training, validation and test images and labels. Images are in the $[-128, 127]$ range when created by the dataset scripts.

In the training script, data points are normalized to $[-0.5, 0.5]$ and fed to the network. However, in the synthesis script, samples are expected in the $[-128, 127]$ range and normalized to the $[-1, +1]$ range to mimic hardware. To accommodate $\times 2$ scaling, the first layer weights are scaled by 1/2 by default, unless `--keep-first` is selected in synthesis.

Examples

MNIST model

The MNIST model is an example of a Keras model with sequential API and it recognizes 28×28 images of handwritten digits from 0 to 9.

```
model = tf.keras.models.Sequential([  
    tf.keras.Input(shape=(28, 28)),  
    tf.keras.layers.Reshape(target_shape=(28, 28, 1)),  
    ai8xTF.FusedConv2DReLU(  
        kernel_size=(3, 3),  
        filters=16,  
        padding='same',  
        activation='relu',  
        data_format='channels_last',  
        name='conv2d_1')])
```

```

        filters=60,
        kernel_size=3,
        strides=1,
        padding_size=1,
        use_bias=False),
ai8xTF.FusedMaxPoolConv2DReLU(
    filters=60,
    kernel_size=3,
    strides=1,
    padding_size=2,
    pool_size=2,
    pool_strides=2,
    use_bias=False),
ai8xTF.FusedMaxPoolConv2DReLU(
    filters=56,
    kernel_size=3,
    strides=1,
    padding_size=1,
    pool_size=2,
    pool_strides=2,
    use_bias=False),
ai8xTF.FusedAvgPoolConv2DReLU(
    filters=12,
    kernel_size=3,
    strides=1,
    padding_size=1,
    pool_size=2,
    pool_strides=2,
    use_bias=False),
tf.keras.layers.Flatten(),
ai8xTF.FusedDense(10, wide=True, use_bias=True),
1)

```

To train the MNIST model, execute following script:

```
$ bash train_mnist.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

```

Epoch 98/100
211/211 - 2s - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.0279 - val_accuracy:
0.9908
Epoch 99/100
211/211 - 2s - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.0279 - val_accuracy:
0.9908

```

```

Epoch 100/100
211/211 - 2s - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0278 - val_accuracy:
0.9908
188/188 - 0s - loss: 0.0302 - accuracy: 0.9900
Test Accuracy: 0.9900000095367432
Confusion Matrix:
tf.Tensor(
[[619   0   1   1   0   0   1   0   2   0]
 [  0 654   0   0   0   0   0   0   0   0]
 [  0   1 569   0   0   0   0   1   0   1]
 [  0   0   2 584   0   2   0   0   1   0]
 [  1   0   0   0 572   0   0   1   0   6]
 [  1   0   0   1   0 544   2   0   3   0]
 [  2   0   1   1   1   2 572   0   1   0]
 [  0   1   3   0   1   0   0 627   0   1]
 [  1   0   2   1   1   0   0   0 580   0]
 [  0   0   0   1   7   1   0   3   1 619]], shape=(10, 10), dtype=int32)

```

At end of training, a summary of the model is reported and model graph and weights are stored as `saved_model.pb`.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
reshape (Reshape)	(None, 28, 28, 1)	0
<hr/>		
fused_conv2d_re_lu (FusedCon	(None, 28, 28, 60)	540
<hr/>		
fused_max_pool_conv2d_re_lu	(None, 16, 16, 60)	32400
<hr/>		
fused_max_pool_conv2d_re_lu_	(None, 8, 8, 56)	30240
<hr/>		
fused_avg_pool_conv2d_re_lu	(None, 4, 4, 12)	6048
<hr/>		
flatten (Flatten)	(None, 192)	0
<hr/>		
fused_dense (FusedDense)	(None, 10)	1930
=====		
Total params: 71,158		
Trainable params: 71,158		
Non-trainable params: 0		
<hr/>		

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/mnist/saved_model.pb  
export/mnist/saved_model.onnx
```

FashionMNIST model

This model demonstrates recognition of 10 28×28 fashion images: *T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot*.

The FashionMNIST model is an example of a Keras model with functional API:

```
# create a functional model  
input_layer = tf.keras.Input(shape=(28, 28))  
  
reshape = tf.keras.layers.Reshape(target_shape=(28, 28, 1))(input_layer)  
conv1 = ai8xTF.FusedConv2DReLU(  
    filters=60,  
    kernel_size=3,  
    strides=1,  
    padding_size=1)(reshape)  
  
conv2 = ai8xTF.FusedMaxPoolConv2DReLU(  
    filters=60,  
    kernel_size=3,  
    strides=1,  
    padding_size=2,  
    pool_size=2,  
    pool_strides=2)(conv1)  
  
# dropout1 = tf.keras.layers.Dropout(0.2)(conv2)  
  
conv3 = ai8xTF.FusedMaxPoolConv2DReLU(  
    filters=56,  
    kernel_size=3,  
    strides=1,  
    padding_size=1,  
    pool_size=2,  
    pool_strides=2)(conv2)  
  
conv4 = ai8xTF.FusedAvgPoolConv2DReLU(  
    filters=10,  
    kernel_size=2,  
    strides=1,  
    padding_size=0)(conv3)
```

```

        filters=12,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2)(conv3)

flat = tf.keras.layers.Flatten(input_shape=(28, 28))(conv4)

output_layer = ai8xTF.FusedDense(10, wide=True)(flat)

model = tf.keras.Model(inputs=[input_layer], outputs=[output_layer])

```

To train the FashionMNIST model, execute following script:

```
$ bash train_fashionmnist.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

```

Epoch 97/100
211/211 - 2s - loss: 0.1520 - accuracy: 0.9490 - val_loss: 0.2439 - val_accuracy:
0.9121
Epoch 98/100
211/211 - 2s - loss: 0.1517 - accuracy: 0.9492 - val_loss: 0.2441 - val_accuracy:
0.9113
Epoch 99/100
211/211 - 2s - loss: 0.1515 - accuracy: 0.9491 - val_loss: 0.2440 - val_accuracy:
0.9108
Epoch 100/100
211/211 - 2s - loss: 0.1512 - accuracy: 0.9493 - val_loss: 0.2441 - val_accuracy:
0.9107
188/188 - 0s - loss: 0.2329 - accuracy: 0.9148
Test Accuracy: 0.9148333072662354
Confusion Matrix:
tf.Tensor(
[[522   1  11  14   1   0  46   0   2   0]
 [  0 597   0  10   0   0   1   0   0   0]
 [  9   0 545   3  31   0  22   0   1   0]
 [ 15   3   2 536  17   0  13   0   1   0]
 [  0   0  24  22 546   0  33   0   2   0]
 [  0   0   0   0   0 601   1  11   4   4]
 [ 60   2  32  17  35   0 467   0   6   0]
 [  0   0   0   0   0   7   0 533   0  10]
 [  2   1   2   0   2   2   3   0 578   0]
 [  0   0   0   0   0   3   0  22   1 564]], shape=(10, 10), dtype=int32)

```

At the end of training, a summary of the model is reported and model graph and weights are stored as `saved_model.pb`.

Model: "functional_1"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 28, 28)]	0
reshape (Reshape)	(None, 28, 28, 1)	0
fused_conv2d_re_lu (FusedCon	(None, 28, 28, 60)	600
fused_max_pool_conv2d_re_lu	(None, 16, 16, 60)	32460
fused_max_pool_conv2d_re_lu_	(None, 8, 8, 56)	30296
fused_avg_pool_conv2d_re_lu	(None, 4, 4, 12)	6060
flatten (Flatten)	(None, 192)	0
fused_dense (FusedDense)	(None, 10)	1930
=====		
Total params: 71,346		
Trainable params: 71,346		
Non-trainable params: 0		

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/fashionmnist/saved_model.pb
export/fashionmnist/saved_model.onnx
```

CIFAR10 model

The CIFAR-10 dataset consists of 60000 32×32 color images in 10 classes: *plane, car, bird, cat, deer, dog, frog, horse, ship, truck*.

<https://www.cs.toronto.edu/~kriz/cifar.html>

The CIFAR10 model is an example of a Keras model with sequential API:

```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(32, 32, 3)),
    ai8xTF.FusedConv2DReLU(
        filters=60, kernel_size=3, strides=1, padding_size=1, use_bias=False),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=60,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=56,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    ai8xTF.FusedAvgPoolConv2DReLU(
        filters=12,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    tf.keras.layers.Flatten(),
    ai8xTF.FusedDense(10, wide=True, use_bias=False),
])
```

To train the CIFAR10 model execute following script:

```
$ bash train_cifar10.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

```
Epoch 98/100
704/704 - 5s - loss: 0.5455 - accuracy: 0.8174 - val_loss: 0.8520 - val_accuracy:
0.7002
Epoch 99/100
```

```

704/704 - 6s - loss: 0.5443 - accuracy: 0.8179 - val_loss: 0.8533 - val_accuracy:
0.7017
Epoch 100/100
704/704 - 6s - loss: 0.5436 - accuracy: 0.8173 - val_loss: 0.8515 - val_accuracy:
0.7040
157/157 - 0s - loss: 0.8420 - accuracy: 0.7036
Test Accuracy: 0.7035999894142151
Confusion Matrix:
tf.Tensor(
[[356  15  21  10   6   3   3   8  31  23]
 [ 13 402   5   2   1   1   3   4  17  39]
 [ 41   3 311  32  43  36  36  13   6   6]
 [ 13   1  35 243  37 126  37  21   5   5]
 [ 16   1  22  20 335  17  26  41   4   4]
 [  3   1  27  91  26 312   9  31   1   3]
 [  7   2  31  26  33  12 367   4   1   2]
 [  9   2  20  15  36  30   3 382   3   9]
 [ 41  20   2   6   1   0   4   3 416  10]
 [ 15  47   4   6   7   5   2   5  15 394]], shape=(10, 10), dtype=int32)

```

At the end of training a summary of the model is reported and model graph and weights are stored as `saved_model.pb`.

Model: "sequential"

Layer (type)	Output Shape	Param #
fused_conv2d_re_lu (FusedCon	(None, 32, 32, 60)	1620
fused_max_pool_conv2d_re_lu	(None, 16, 16, 60)	32400
fused_max_pool_conv2d_re_lu_	(None, 8, 8, 56)	30240
fused_avg_pool_conv2d_re_lu	(None, 4, 4, 12)	6048
flatten (Flatten)	(None, 192)	0
fused_dense (FusedDense)	(None, 10)	1920

Total params: 72,228
 Trainable params: 72,228
 Non-trainable params: 0

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/cifar10/saved_model.pb
export/cifar10/saved_model.onnx
```

CIFAR100 model

The CIFAR100 model classifies 100 32×32 color images from a 60,000 dataset.

<https://www.cs.toronto.edu/~kriz/cifar.html>

The CIFAR100 model is an example of a Keras model with sequential API:

```
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(32, 32, 3)),
    ai8xTF.FusedConv2DReLU(
        filters=60, kernel_size=3, strides=1, padding_size=1, use_bias=False),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=60,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=56,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    ai8xTF.FusedAvgPoolConv2DReLU(
        filters=12,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2,
        use_bias=False),
    tf.keras.layers.Flatten(),
    ai8xTF.FusedDense(10, wide=True, use_bias=False),
```

```
1)
```

To train the CIFAR100 model execute following script:

```
$ bash train_cifar100.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

```
Epoch 99/100
1407/1407 - 11s - loss: 1.1360 - accuracy: 0.7284 - val_loss: 2.8744 -
val_accuracy: 0.3305
Epoch 100/100
1407/1407 - 11s - loss: 1.1316 - accuracy: 0.7310 - val_loss: 2.8761 -
val_accuracy: 0.3283
157/157 - 0s - loss: 2.8540 - accuracy: 0.3338
Test Accuracy: 0.33379998803138733
Confusion Matrix:
tf.Tensor(
[[34  2  0 ...  0  0  0]
 [ 0 18  1 ...  0  0  0]
 [ 0  0 19 ...  1  4  0]
 ...
 [ 0  0  0 ... 11  0  0]
 [ 0  0  2 ...  1 18  0]
 [ 0  0  0 ...  1  0  8]], shape=(100, 100), dtype=int32)
```

At end of training a summary of the model is reported and model graph and weights are stored as `saved_model.pb`:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
fused_conv2d_re_lu (FusedCon	(None, 32, 32, 16)	432
fused_conv2d_re_lu_1 (FusedC	(None, 32, 32, 20)	2880
fused_conv2d_re_lu_2 (FusedC	(None, 32, 32, 20)	3600
fused_conv2d_re_lu_3 (FusedC	(None, 32, 32, 20)	3600
fused_max_pool_conv2d_re_lu	(None, 16, 16, 20)	3600
fused_conv2d_re_lu_4 (FusedC	(None, 16, 16, 20)	3600

fused_conv2d_re_lu_5	(FusedC (None, 16, 16, 44))	7920
fused_max_pool_conv2d_re_lu_	(None, 8, 8, 48)	19008
fused_conv2d_re_lu_6	(FusedC (None, 8, 8, 48))	20736
fused_max_pool_conv2d_re_lu_	(None, 4, 4, 96)	41472
fused_max_pool_conv2d_re_lu_	(None, 2, 2, 512)	49152
fused_conv2d_re_lu_7	(FusedC (None, 2, 2, 128))	65536
fused_max_pool_conv2d_re_lu_	(None, 1, 1, 128)	147456
conv2d_13	(Conv2D) (None, 1, 1, 100)	12800
flatten	(Flatten) (None, 100)	0
=====		
Total params: 381,792		
Trainable params: 381,792		
Non-trainable params: 0		

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/cifar100/saved_model.pb
export/cifar100/saved_model.onnx
```

KWS20 model

The KWS20 model uses the second version of the *Google speech commands dataset*, which consists of 35 keywords and more than 100,000 utterances.

https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz

This model demonstrates recognition of 20 keywords: 'up', 'down', 'left', 'right', 'stop', 'go', 'yes', 'no', 'on', 'off', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'zero'. The rest of keywords are placed into the "unknown" category.

The KWS20 model is an example of a Keras model with sequential API:


```

model = tf.keras.models.Sequential([
    # Need to specify the input shape if you want to show it in model summary
    tf.keras.Input(shape=(128, 128)),
    ai8xTF.FusedConv1DReLU(
        filters=100,
        kernel_size=1,
        strides=1,
        padding_size=0,
        kernel_regularizer=regularizer,
        activity_regularizer=activity_regularizer,
        use_bias=False),
    ai8xTF.FusedConv1DReLU(
        filters=100,
        kernel_size=1,
        strides=1,
        padding_size=0,
        kernel_regularizer=regularizer,
        activity_regularizer=activity_regularizer,
        use_bias=False),
    ai8xTF.FusedConv1DReLU(
        filters=50,
        kernel_size=1,
        strides=1,
        padding_size=0,
        kernel_regularizer=regularizer,
        activity_regularizer=activity_regularizer,
        use_bias=False),
    ai8xTF.FusedConv1DReLU(
        filters=16,
        kernel_size=1,
        strides=1,
        padding_size=0,
        kernel_regularizer=regularizer,
        activity_regularizer=activity_regularizer,
        use_bias=False),

    # Conversion 1D to 2D
    tf.keras.layers.Reshape(target_shape=(8, 16, 16)),
    ai8xTF.FusedConv2DReLU(
        filters=32,
        kernel_size=3,
        strides=1,
        padding_size=1,
        kernel_regularizer=regularizer,
        activity_regularizer=activity_regularizer,

```

```

        use_bias=False),
ai8xTF.FusedConv2DReLU(
    filters=64,
    kernel_size=3,
    strides=1,
    padding_size=1,
    kernel_regularizer=regularizer,
    activity_regularizer=activity_regularizer,
    use_bias=False),
ai8xTF.FusedConv2DReLU(
    filters=64,
    kernel_size=3,
    strides=1,
    padding_size=1,
    kernel_regularizer=regularizer,
    activity_regularizer=activity_regularizer,
    use_bias=False),
ai8xTF.FusedConv2DReLU(
    filters=30,
    kernel_size=3,
    strides=1,
    padding_size=1,
    kernel_regularizer=regularizer,
    activity_regularizer=activity_regularizer,
    use_bias=False),
ai8xTF.FusedConv2DReLU(
    filters=7,
    kernel_size=3,
    strides=1,
    padding_size=1,
    kernel_regularizer=regularizer,
    activity_regularizer=activity_regularizer,
    use_bias=False),
tf.keras.layers.Flatten(),
ai8xTF.FusedDense(
    21, wide=True,
    use_bias=False,
    kernel_regularizer=regularizer,
    activity_regularizer=activity_regularizer),
1)

```

To train the KWS20 model execute the following script:

```
$ bash train_kws20.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

Epoch 198/200

667/667 - 5s - loss: 0.3848 - accuracy: 0.9528 - val_loss: 0.6943 - val_accuracy: 0.8573

Epoch 199/200

667/667 - 5s - loss: 0.3847 - accuracy: 0.9522 - val_loss: 0.7012 - val_accuracy: 0.8545

Epoch 200/200

667/667 - 4s - loss: 0.3842 - accuracy: 0.9527 - val_loss: 0.6977 - val_accuracy: 0.8561

319/319 - 1s - loss: 0.7320 - accuracy: 0.8514

Test Accuracy: 0.8514375686645508

Confusion Matrix:

tf.Tensor(
[[323

0	1	0	0	17]	[2	303	0	0	1	13	0	11	1	0	1	2	0	1	0	1
1	0	8	0	34]	[1	0	304	8	0	0	13	3	0	2	1	0	1	0	0	0
0	1	3	1	23]	[1	1	7	297	0	0	0	0	2	0	5	0	2	1	6	0
0	1	14	0	21]	[4	1	0	0	350	2	0	0	0	2	0	1	0	2	0	3
8	0	0	0	9]	[4	11	0	0	3	273	0	17	1	1	0	6	0	5	0	0
0	3	1	0	46]	[0	2	8	0	0	0	364	2	0	3	2	0	1	0	0	3
0	1	0	2	15]	[2	12	0	0	0	21	2	307	0	0	2	1	0	1	0	0
0	0	7	2	30]	[2	4	0	0	0	2	0	0	298	12	9	1	1	2	15	1
0	0	2	0	12]	[24	0	1	0	0	2	2	0	9	284	1	1	0	4	5	0
0	0	0	0	7]	[1	1	2	4	0	2	1	3	7	0	320	0	0	1	2	0
0	0	7	0	33]	[1	1	1	0	1	7	1	0	0	0	0	319	3	4	0	2
3	0	1	11	18]	[0	0	0	3	0	5	1	0	1	0	0	9	289	0	0	4
3	6	1	3	32]	[0	0	0	0	0	6	0	0	6	2	3	0	1	286	1	3
0	0	0	3	48]	[1	2	2	10	1	0	0	0	9	2	2	0	3	0	321	2
1	1	3	0	26]																	

```
[ 0 0 0 0 0 0 0 2 0 0 0 0 1 1 0 0 349
0 3 0 1 7]
[ 0 6 0 0 5 0 0 0 0 2 0 0 6 2 1 1 7
332 0 1 2 30]
[ 1 0 0 0 0 2 2 0 0 0 0 2 6 0 1 5
0 349 0 0 15]
[ 0 2 5 9 0 3 0 7 3 0 6 0 2 0 3 0
0 0 319 1 28]
[ 0 1 1 0 0 1 0 1 0 0 0 7 4 1 0 3
4 0 0 358 18]
[ 20 32 16 12 9 32 10 21 15 14 17 10 36 37 17 5
10 7 18 18 2332]], shape=(21, 21), dtype=int32)
```

At end of training, a summary of the model is reported and model graph and weights are stored as `saved_model.pb`.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
fused_conv1d_re_lu (FusedCon	(None, 128, 100)	12800
fused_conv1d_re_lu_1 (FusedC	(None, 128, 100)	10000
fused_conv1d_re_lu_2 (FusedC	(None, 128, 50)	5000
fused_conv1d_re_lu_3 (FusedC	(None, 128, 16)	800
reshape (Reshape)	(None, 8, 16, 16)	0
fused_conv2d_re_lu (FusedCon	(None, 8, 16, 32)	4608
fused_conv2d_re_lu_1 (FusedC	(None, 8, 16, 64)	18432
fused_conv2d_re_lu_2 (FusedC	(None, 8, 16, 64)	36864
fused_conv2d_re_lu_3 (FusedC	(None, 8, 16, 30)	17280
fused_conv2d_re_lu_4 (FusedC	(None, 8, 16, 7)	1890
flatten (Flatten)	(None, 896)	0
fused_dense (FusedDense)	(None, 21)	18816
=====		
Total params: 126,490		

Trainable params: 126,490

Non-trainable params: 0

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/kws20/saved_model.pb
export/kws20/saved_model.onnx
```

Rock-Paper-Scissors model

This model demonstrates recognition of images of hands playing the popular “rock, paper, scissors” (RPS) game.

https://www.tensorflow.org/datasets/catalog/rock_paper_scissors

The RPS model is an example of a Keras model with sequential API:

```
IMG_SIZE = 64 # All images will be resized to 120x120
# Setup model
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3)),
    ai8xTF.FusedConv2DReLU(
        filters=15,
        kernel_size=3,
        strides=1,
        padding_size=1
    ),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=30,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2
    ),
    tf.keras.layers.Dropout(0.2),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=60,
        kernel_size=3,
        strides=1,
```

```

        padding_size=1,
        pool_size=2,
        pool_strides=2
    ),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=30,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2
    ),
    ai8xTF.FusedMaxPoolConv2DReLU(
        filters=30,
        kernel_size=3,
        strides=1,
        padding_size=1,
        pool_size=2,
        pool_strides=2
    ),
    ai8xTF.FusedConv2DReLU(
        filters=30,
        kernel_size=3,
        strides=1,
        padding_size=1
    ),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
    ai8xTF.FusedDense(3, wide=True)
1)

```

To train the RPS model, execute the following script:

```
$ bash train_rock.sh
```

Training progress, accuracy results and confusion table are reported and stored in a log file:

```

Epoch 98/100
79/79 - 0s - loss: 6.1023e-05 - accuracy: 1.0000 - val_loss: 0.3391 -
val_accuracy: 0.9140
Epoch 99/100
79/79 - 1s - loss: 5.7724e-05 - accuracy: 1.0000 - val_loss: 0.3437 -
val_accuracy: 0.9140
Epoch 100/100
79/79 - 0s - loss: 6.5221e-05 - accuracy: 1.0000 - val_loss: 0.3369 -
val_accuracy: 0.9140
6/6 - 0s - loss: 0.3191 - accuracy: 0.9140
Test Accuracy: 0.9139785170555115
Confusion Matrix:
tf.Tensor(
[[61  0  0]
 [ 8 47  8]
 [ 0  0 62]], shape=(3, 3), dtype=int32)

```

At the end of training, a summary of the model is reported and model graph and weights are stored as `saved_model.pb`.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
fused_conv2d_re_lu (FusedCon	(None, 64, 64, 15)	420
fused_max_pool_conv2d_re_lu	(None, 32, 32, 30)	4080
dropout (Dropout)	(None, 32, 32, 30)	0
fused_max_pool_conv2d_re_lu_	(None, 16, 16, 60)	16260
fused_max_pool_conv2d_re_lu_	(None, 8, 8, 30)	16230
fused_max_pool_conv2d_re_lu_	(None, 4, 4, 30)	8130
fused_conv2d_re_lu_1 (FusedC	(None, 4, 4, 30)	8130
flatten (Flatten)	(None, 480)	0
dropout_1 (Dropout)	(None, 480)	0
fused_dense (FusedDense)	(None, 3)	1443
=====		
Total params: 54,693		

```
Trainable params: 54,693
```

```
Non-trainable params: 0
```

Note: Empty classes may be included as part of subclasses in the sequential model. However, they are not needed and skipped during serialization.

Additionally, the model is converted to ONNX format:

```
export/rock/saved_model.pb  
export/rock/saved_model.onnx
```

Post-Training Model Quantization

The synthesis script quantizes the weights from the provided ONNX file internally. To run through evaluation, the script can be run with `--generate-dequantized-onnx-file` and it will regenerate a dequantized ONNX file that can be used with the ONNX runtime for evaluation.

Model Evaluation

After quantization, the dequantized model can be evaluated and compared with the unquantized model (MNIST example) :

```
$ bash evaluate_mnist.sh
```

Alternatively, the user can evaluate all of the model examples by running a bash script:

```
$ bash evaluate_ALL.sh
```

MAX78000 Synthesis

To quantize TensorFlow models and synthesize MAX78000 C source code from ONNX files, execute the following command (MNIST example) :


```
$ (ai8x-synthesis) ./ai8xize.py --verbose -L --top-level cnn --test-dir tensorflow
--prefix tf-mnist --checkpoint-file ../ai8x-
training/TensorFlow/export/mnist/saved_model.onnx --config-file ./networks/mnist-
chw-ai85-tf.yaml --sample-input ../ai8x-
training/TensorFlow/export/mnist/sampled_data.npy --device MAX78000 --compact-data -
-mexpress --embedded-code --scale 1.0 --softmax --display-checkpoint $@
```

`ai8xize` requires three input files:

1. `ai8x-training/TensorFlow/export/mnist/saved_model.onnx` - ONNX representation of the TensorFlow model
2. `ai8x-synthesis/networks/mnist-chw-ai85-tf.yaml` - YAML description of the model
3. `ai8x-training/TensorFlow/export/mnist/sampled_data.npy` - Input data sample file

Parameter	Description
<code>--keep-first</code>	Applies same scale factor of weights as specified in <code>--scale</code> to the first layer. When not specified (default), half of the scale factor is applied to the first layer. See the section <i>Datasets</i> in this document.
<code>--scale</code>	Scale factor for weight quantization (default = 1.0)
<code>--generate-dequantized-onnx-file</code>	Generates a dequantized copy of the ONNX file to be used for evaluation. See the section <i>Post-Training Model Quantization</i> in this document.

Other parameters are described in section “Network Loader (AI8Xize)” of [1].

Generated C code is stored in the `ai8x-synthesis/tensorflow/tf-mnist/` directory.

To generate MAX78000 C source code for all TensorFlow examples, execute following script:

```
$ (ai8x-synthesis) bash gen-tf-demos-max78000.sh
```

Expected shape of input data sample file

Example	TensorFlow dataset native shape	Synthesis expected shape	Command line option for training to generate expected sample file shape
mnist, fashionmnist (or cases with conv2d as input with 1 channel)	(1,28,28)	same: (1,28,28)	none
cifar10, cifar100, (or cases with conv2d as input with multiple channels like rock)	(1,32,32,3)	channel first: (3,32,32)	--channel-first
kws20 (or cases with conv1d as input)	(1,128,128)	channel first: (128,128)	-- channel-first --swap

References

[1] [ai8x-training/README.md](#)