In [1]:

```
import onnx
import torch
```

In [2]:

```
#load dataset

import torch, torchvision, copy
from torch import nn
from torchvision import datasets
from torchvision.transforms import transforms
import numpy as np
from tqdm import tqdm

transform_train = transforms.Compose([transforms.ToTensor()])
transform_test = transforms.Compose([transforms.ToTensor()])
training_data = datasets.FashionMNIST(root="/workspace/finn/notebooks/mnist_ex/data2/FashionMNIST/raw", train=True, download=
test_data = datasets.FashionMNIST(root="/workspace/finn/notebooks/mnist_ex/data2/FashionMNIST/raw", train=False, download=Fals
print("Samples in each set: train = %d, test = %s" % (len(training_data), len(test_data)))
print("Shape of one input sample: " +  str(training_data[0][0].shape))
```

```
Samples in each set: train = 60000, test = 10000
Shape of one input sample: torch.Size([1, 28, 28])
```

In [3]:

```
#divide in batches

from torch.utils.data import DataLoader, Dataset
batch_size = 32
# dataset loaders
train_dataloader = DataLoader(training_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
count = 0
for x,y in train_dataloader:
    print("Input shape for 1 batch: " + str(x.shape))
    print("Label shape for 1 batch: " + str(y.shape))
    count += 1
    if count == 1:
        break
```

```
Input shape for 1 batch: torch.Size([32, 1, 28, 28])
Label shape for 1 batch: torch.Size([32])
```

In [4]:

```
#model declaration

from torch import nn
from torch.nn import Module
import brevitas.nn as qnn
from brevitas.quant import Int8Bias as BiasQuant
from brevitas.quant import Uint8ActPerTensorFloat as ActQuant
from brevitas.quant import Int8WeightPerTensorFloat as WeighQuant
import torch.nn.functional as F

# Setting seeds for reproducibility
torch.manual_seed(0)

weight_bitx = 8
act_bitx = 8
bias_bitx = 8

model = nn.Sequential(
    #qnn.QuantIdentity(bit_width=act_bitx, return_quant_tensor=True),
    qnn.QuantConv2d(in_channels=1, out_channels=3, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2),
                                    bias=False, weight_bit_width=weight_bitx, weight_quant=WeighQuant,
                                    bias_quant=BiasQuant, return_quant_tensor=True),
    qnn.QuantReLU(bit_width=act_bitx, act_quant=ActQuant, return_quant_tensor=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    qnn.QuantConv2d(in_channels=3, out_channels=8, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2),
                                    bias=False, weight_bit_width=weight_bitx, bias_quant=BiasQuant,
                                    weight_quant=WeighQuant, return_quant_tensor=True),
    qnn.QuantReLU(bit_width=act_bitx, act_quant=ActQuant, return_quant_tensor=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    qnn.QuantConv2d(in_channels=8, out_channels=16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2),
                                    bias=False, weight_bit_width=weight_bitx, bias_quant=BiasQuant,
                                    weight_quant=WeighQuant, return_quant_tensor=True),
    qnn.QuantReLU(bit_width=act_bitx, act_quant=ActQuant, return_quant_tensor=True),
    nn.Flatten(),
    qnn.QuantLinear(in_features=16*7*7, out_features=10, bias=True, weight_bit_width=weight_bitx,
                                   weight_quant=WeighQuant, bias_quant=BiasQuant, return_quant_tensor=False)
)
```

In [5]:

```
#train and test functions
import torch
from sklearn.metrics import accuracy_score

def train(model, train_loader, optimizer, criterion):
    losses = []
    # ensure model is in training mode
    model.train()

    for i, data in enumerate(train_loader, 0):
        inputs, target = data
        optimizer.zero_grad()

        # forward pass
        output = model(inputs.float())
        loss = criterion(output, target)#.unsqueeze(1))

        # backward pass + run optimizer to update weights
        loss.backward()
        optimizer.step()

        # keep track of loss value
        losses.append(loss.data.numpy())

    return losses



def test(model, test_loader):
    # ensure model is in eval mode
    model.eval()
    y_true = []
    y_pred = []

    with torch.no_grad():
        for data in test_loader:
            inputs, target = data
            output_orig = model(inputs.float())
            # run the output through sigmoid
            #output = torch.sigmoid(output_orig)
            # compare against a threshold of 0.5 to generate 0/1
            #pred = (output.detach().numpy() > 0.5) * 1
            target = target.float()
            y_true.extend(target.tolist())
            y_pred.extend(output_orig.argmax(1).reshape(-1).tolist())

    return accuracy_score(y_true, y_pred)
```

In [6]:

```
#training settings
num_epochs = 10
lr = 0.001

def display_loss_plot(losses, title="Training loss", xlabel="Iterations", ylabel="Loss"):
    x_axis = [i for i in range(len(losses))]
    plt.plot(x_axis,losses)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()

# loss criterion and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=(0.9, 0.999))
```

In [7]:

```
#training
import numpy as np
from sklearn.metrics import accuracy_score
from tqdm import tqdm, trange

# Setting seeds for reproducibility
torch.manual_seed(0)
np.random.seed(0)

running_loss = []
running_test_acc = []
t = trange(num_epochs, desc="Training loss", leave=True)

for epoch in t:
        loss_epoch = train(model, train_dataloader, optimizer,criterion)
        test_acc = test(model, test_dataloader)
        t.set_description("Training loss = %f test accuracy = %f" % (np.mean(loss_epoch), test_acc))
        t.refresh() # to show immediately the update
        running_loss.append(loss_epoch)
        running_test_acc.append(test_acc)
```

```
Training loss = 0.239489 test accuracy = 0.898300: 100%|██████████| 10/10 [03:18<00:00, 19.82s/it]
```

In [8]:

```
#testing
test(model, test_dataloader)
```

Out[8]:

```
0.8983
```

In [9]:

```
# Save the Brevitas model to disk
torch.save(model.state_dict(), "state_dict_self-trained.pth")
```

In [10]:

```
import brevitas.onnx as bo
from brevitas.quant_tensor import QuantTensor

ready_model_filename = "model_fmnist_notebook.onnx"
input_shape = (1, 1, 28, 28)


bo.export_finn_onnx(
    model, export_path=ready_model_filename, input_shape=input_shape
)

print("Model saved to %s" % ready_model_filename)
```

```
Model saved to model_fmnist_notebook.onnx
```

In [ ]: