Table of Contents

Lab 5 – Neo Lok Jun

Note: Mac terminal and GUI are shown in the screenshots because this lab is done with the use of SSH to the RPi.

```
[lokjun@Neos-MacBook-Air ~ % ssh -X Grp22Pi@192.168.236.178
[Grp22Pi@192.168.236.178's password:
Linux Grp22Pi 6.6.51+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.51-1+rpt3 (2024-10-0
8) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jan 23 05:05:20 2025
[Grp22Pi@Grp22Pi:~ $ git clone https://github.com/drfuzzi/INF2009_DLonEdge
```

Figure 1: SSH Access via MAC

Venv and installations screenshots are not included, the following screenshots will show that venv is used (dlonedge).

```
(dlonedge) Grp22Pi@Grp22Pi:~/INF2009_DLonEdge $ python Codes/mobile_net.py
```

Part 1

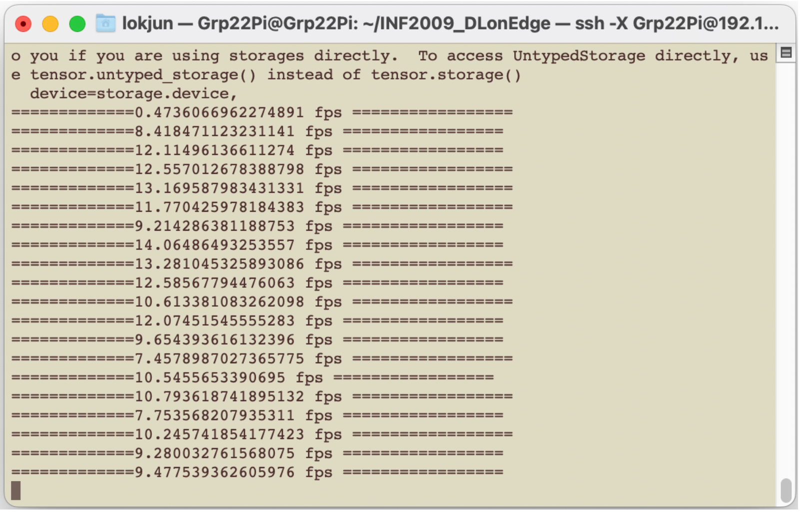
Running mobile_net.py without any optimization

```
lokjun — Grp22Pi@Grp22Pi: ~/INF2009_DLonEdge — ssh -X Grp22Pi@192.1...
100.0%
=============0.1794699157866154 fps ================
=============2.860855457012521 fps ================
=============2.8989404728142083 fps ================
=============2.8775740666628553 fps ================
=============2.8478216490276873 fps ================
=============2.9182334626289332 fps ================
=============2.841926099918692 fps ================
=============2.8962700853465386 fps ================
=============2.8872717076370855 fps ================
=============2.830979679259886 fps ================
=============2.8921806454460075 fps ================
=============2.92479144763894 fps ================
=============2.869524763421616 fps ================
=============2.9245086602830446 fps ================
=============2.8319704534942995 fps ================
=============2.8999239234219294 fps ================
=============2.873501124014261 fps ================
=============2.8705905112666277 fps ================
=============2.8918144050273753 fps ================
=============2.8664236769208338 fps ================
=============2.8808253096800116 fps ================
=============2.8181794563292235 fps ================
=============2.877147042837848 fps ================
```

Figure 2: Achieved 2-3 fps

Part 2

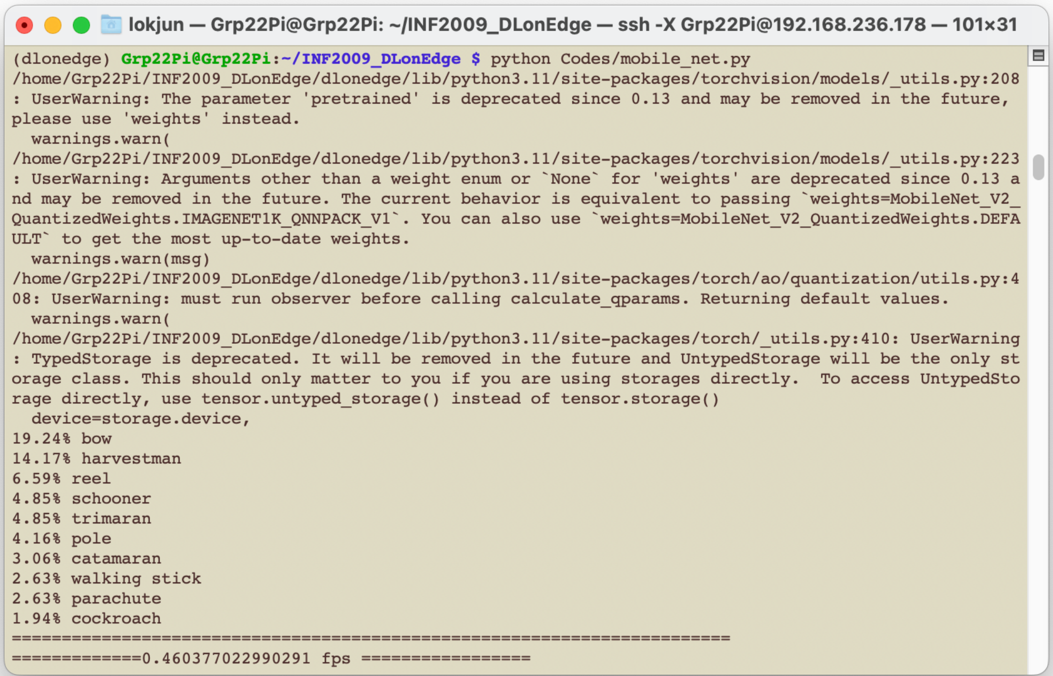
Running with quantized = True




```
o you if you are using storages directly.  To access UntypedStorage directly, us
e tensor.untyped_storage() instead of tensor.storage()
  device=storage.device,
=============0.4736066962274891 fps =================
=============8.418471123231141 fps =================
=============12.11496136611274 fps =================
=============12.557012678388798 fps =================
=============13.169587983431331 fps =================
=============11.770425978184383 fps =================
=============9.214286381188753 fps =================
=============14.06486493253557 fps =================
=============13.281045325893086 fps =================
=============12.58567794476063 fps =================
=============10.613381083262098 fps =================
=============12.07451545555283 fps =================
=============9.654393616132396 fps =================
=============7.4578987027365775 fps =================
=============10.5455653390695 fps =================
=============10.793618741895132 fps =================
=============7.753568207935311 fps =================
=============10.245741854177423 fps =================
=============9.280032761568075 fps =================
=============9.477539362605976 fps =================
```

Figure 3: Achieved ~10 fps

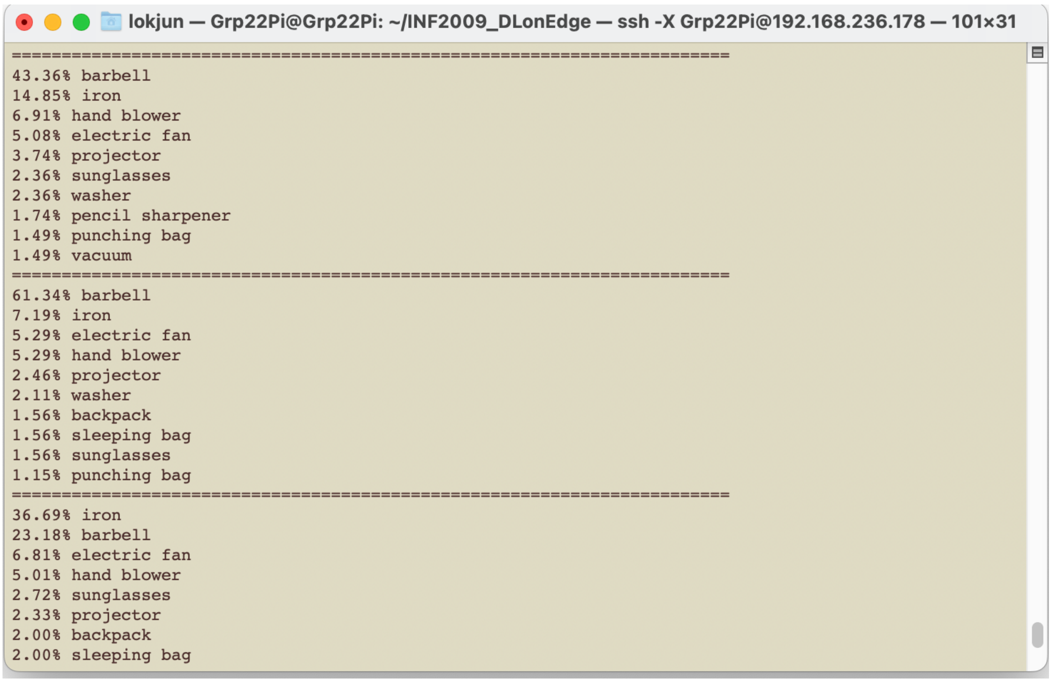
Part 3




```
(dlonedge) Grp22Pi@Grp22Pi:~/INF2009_DLonEdge $ python Codes/mobile_net.py
/home/Grp22Pi/INF2009_DLonEdge/dlonedge/lib/python3.11/site-packages/torchvision/models/_utils.py:208
: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future,
please use 'weights' instead.
  warnings.warn(
/home/Grp22Pi/INF2009_DLonEdge/dlonedge/lib/python3.11/site-packages/torchvision/models/_utils.py:223
: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 a
nd may be removed in the future. The current behavior is equivalent to passing `weights=MobileNet_V2_
QuantizedWeights.IMAGENET1K_QNNPACK_V1`. You can also use `weights=MobileNet_V2_QuantizedWeights.DEFA
ULT` to get the most up-to-date weights.
  warnings.warn(msg)
/home/Grp22Pi/INF2009_DLonEdge/dlonedge/lib/python3.11/site-packages/torch/ao/quantization/utils.py:4
08: UserWarning: must run observer before calling calculate_qparams. Returning default values.
  warnings.warn(
/home/Grp22Pi/INF2009_DLonEdge/dlonedge/lib/python3.11/site-packages/torch/_utils.py:410: UserWarning
: TypedStorage is deprecated. It will be removed in the future and UntypedStorage will be the only st
orage class. This should only matter to you if you are using storages directly.  To access UntypedSto
rage directly, use tensor.untyped_storage() instead of tensor.storage()
  device=storage.device,
19.24% bow
14.17% harvestman
6.59% reel
4.85% schooner
4.85% trimaran
4.16% pole
3.06% catamaran
2.63% walking stick
2.63% parachute
1.94% cockroach
=========================================================================
=============0.460377022990291 fps =================
```

Figure 4: Running with Top 10 predictions

Tried running it in front of an electric fan, but “electric fan” is only at top 3-4 spot of predictions.



lokjun — Grp22Pi@Grp22Pi: ~/INF2009_DLonEdge — ssh -X Grp22Pi@192.168.236.178 — 101×31

```
========================================================================
43.36% barbell
14.85% iron
6.91% hand blower
5.08% electric fan
3.74% projector
2.36% sunglasses
2.36% washer
1.74% pencil sharpener
1.49% punching bag
1.49% vacuum
========================================================================
61.34% barbell
7.19% iron
5.29% electric fan
5.29% hand blower
2.46% projector
2.11% washer
1.56% backpack
1.56% sleeping bag
1.56% sunglasses
1.15% punching bag
========================================================================
36.69% iron
23.18% barbell
6.81% electric fan
5.01% hand blower
2.72% sunglasses
2.33% projector
2.00% backpack
2.00% sleeping bag
```

Figure 5: Camera placed in front of electric fan

Tried other things like my phone or my hands, but it couldn't predict them correctly...

Part 3 - Quantization using Pytorch

Colab URL Here:
https://colab.research.google.com/drive/181T5M7JznMMubQdnQlT9NCR0XtV2rLgF?usp=sharing

Quantization tutorial

This tutorial shows how to do post-training static quantization, as well as illustrating two more advanced techniques - per-channel quantization and quantization-aware training - to further improve the model's accuracy. The task is to classify MNIST digits with a simple LeNet architecture.

Thsi is a mimialistic tutorial to show you a starting point for quantisation in PyTorch. For theory and more in-depth explanations, Please check out: [Quantizing deep convolutional networks for efficient inference: A whitepaper](#) .

The tutorial is heavily adapted from: https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html

Initial Setup

Before beginning the assignment, we import the MNIST dataset, and train a simple convolutional neural network (CNN) to classify it.

```
!pip3 install torch==1.5.0 torchvision==1.6.0
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import os
from torch.utils.data import DataLoader
import torch.quantization
from torch.quantization import QuantStub, DeQuantStub
```

```
ERROR: Could not find a version that satisfies the requirement torch==1.5.0 (from versions: 1.13.0, 1.13.1, 2.0.0, 2.0.1, 2.1.0, 2.1.1, 2.1.2, 2.2.0, 2.2.1, 2.2.2, 2.3.0, 2.3.1, 2.4.0, 2.4.1, 2.5.0, 2.5.1, 2.6.0)
ERROR: No matching distribution found for torch==1.5.0
```

Load training and test data from the MNIST dataset and apply a normalizing transformation.

```
[2] transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5,), (0.5,))])

    trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                          download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                              shuffle=True, num_workers=16, pin_memory=True)

    testset = torchvision.datasets.MNIST(root='./data', train=False,
                                         download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                             shuffle=False, num_workers=16, pin_memory=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:11<00:00, 895kB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 134kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found
```

✓ 2m 39s completed at 2:22 PM

Define some helper functions and classes that help us to track the statistics and accuracy with respect to the train/test data.

```python
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self, name, fmt=':f'):
        self.name = name
        self.fmt = fmt
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)

def accuracy(output, target):
    """ Computes the top 1 accuracy """
    with torch.no_grad():
        batch_size = target.size(0)

        _, pred = output.topk(1, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))

        res = []
        correct_one = correct[:1].view(-1).float().sum(0, keepdim=True)
        return correct_one.mul_(100.0 / batch_size).item()

def print_size_of_model(model):
    """ Prints the real size of the model """
    torch.save(model.state_dict(), "temp.p")
    print('Size (MB):', os.path.getsize("temp.p")/1e6)
    os.remove('temp.p')

def load_model(quantized_model, model):
    """ Loads in the weights into an object meant for quantization """
    state_dict = model.state_dict()
    model = model.to('cpu')
    quantized_model.load_state_dict(state_dict)

def fuse_modules(model):
    """ Fuse together convolutions/linear layers and ReLU """
    torch.quantization.fuse_modules(model, [['conv1', 'relu1'],
                                            ['conv2', 'relu2'],
                                            ['fc1', 'relu3'],
                                            ['fc2', 'relu4']], inplace=True)
```

Define a simple CNN that classifies MNIST images.

[4]

```python
class Net(nn.Module):
    def __init__(self, q = False):
        # By turning on Q we can turn on/off the quantization
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, bias=False)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5, bias=False)
```

```python
class Net(nn.Module):
    def __init__(self, q = False):
        # By turning on Q we can turn on/off the quantization
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, bias=False)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5, bias=False)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256, 120, bias=False)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(120, 84, bias=False)
        self.relu4 = nn.ReLU()
        self.fc3 = nn.Linear(84, 10, bias=False)
        self.q = q
        if q:
          self.quant = QuantStub()
          self.dequant = DeQuantStub()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if self.q:
          x = self.quant(x)
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.pool2(x)
        # Be careful to use reshape here instead of view
        x = x.reshape(x.shape[0], -1)
        x = self.fc1(x)
        x = self.relu3(x)
        x = self.fc2(x)
        x = self.relu4(x)
        x = self.fc3(x)
        if self.q:
          x = self.dequant(x)
        return x
```

[5]
```python
net = Net(q=False).cuda()
print_size_of_model(net)
```

```
Size (MB): 0.179057
```

Train this CNN on the training dataset (this may take a few moments).

Train this CNN on the training dataset (this may take a few moments).

```python
def train(model: nn.Module, dataloader: DataLoader, cuda=False, q=False):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    model.train()
    for epoch in range(10):  # loop over the dataset multiple times

        running_loss = AverageMeter('loss')
        acc = AverageMeter('train_acc')
        for i, data in enumerate(dataloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            if cuda:
              inputs = inputs.cuda()
              labels = labels.cuda()

            # zero the parameter gradients
            optimizer.zero_grad()

            if epoch>=3 and q:
              model.apply(torch.quantization.disable_observer)

            # forward + backward + optimize
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss.update(loss.item(), outputs.shape[0])
            acc.update(accuracy(outputs, labels), outputs.shape[0])
            if i % 100 == 0:    # print every 100 mini-batches
                print('[%d, %5d] ' %
                    (epoch + 1, i + 1), running_loss, acc)
    print('Finished Training')


def test(model: nn.Module, dataloader: DataLoader, cuda=False) -> float:
    correct = 0
    total = 0
    model.eval()
    with torch.no_grad():
        for data in dataloader:
            inputs, labels = data

            if cuda:
              inputs = inputs.cuda()
              labels = labels.cuda()

            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100 * correct / total
```

[7]

```python
train(net, trainloader, cuda=True)
```

```
[5,   301]  loss 0.124549 (0.118892) train_acc 98.437500 (96.340324)
[5,   401]  loss 0.117333 (0.118303) train_acc 96.875000 (96.364557)
[5,   501]  loss 0.106485 (0.115376) train_acc 95.312500 (96.422779)
[5,   601]  loss 0.048786 (0.112470) train_acc 98.437500 (96.498024)
[5,   701]  loss 0.137447 (0.112786) train_acc 95.312500 (96.525053)
[5,   801]  loss 0.046125 (0.112084) train_acc 98.437500 (96.553137)
[5,   901]  loss 0.044387 (0.109823) train_acc 98.437500 (96.614872)
[6,     1]  loss 0.037059 (0.037059) train_acc 100.000000 (100.000000)
[6,   101]  loss 0.050844 (0.106316) train_acc 98.437500 (96.844059)
[6,   201]  loss 0.031111 (0.102250) train_acc 98.437500 (96.828358)
```

Now that the CNN has been trained, let's test it on our test dataset.

[8]
```
score = test(net, testloader, cuda=True)
print('Accuracy of the network on the test images: {}% - FP32'.format(score))
```

```
Accuracy of the network on the test images: 98.19% - FP32
```

Post-training quantization

Define a new quantized network architeture, where we also define the quantization and dequantization stubs that will be important at the start and at the end.

Next, we'll "fuse modules"; this can both make the model faster by saving on memory access while also improving numerical accuracy. While this can be used with any model, this is especially common with quantized models.

[9]
```
qnet = Net(q=True)
load_model(qnet, net)
fuse_modules(qnet)
```

In general, we have the following process (Post Training Quantization):

1. Prepare: we insert some observers to the model to observe the statistics of a Tensor, for example, min/max values of the Tensor
2. Calibration: We run the model with some representative sample data, this will allow the observers to record the Tensor statistics
3. Convert: Based on the calibrated model, we can figure out the quantization parameters for the mapping function and convert the floating point operators to quantized operators

```
qnet.qconfig = torch.quantization.default_qconfig
print(qnet.qconfig)
torch.quantization.prepare(qnet, inplace=True)
print('Post Training Quantization Prepare: Inserting Observers')
print('\n Conv1: After observer insertion \n\n', qnet.conv1)

test(qnet, trainloader, cuda=False)
print('Post Training Quantization: Calibration done')
torch.quantization.convert(qnet, inplace=True)
print('Post Training Quantization: Convert done')
print('\n Conv1: After fusion and quantization \n\n', qnet.conv1)
print("Size of model after quantization")
print_size_of_model(qnet)
```

```
QConfig(activation=functools.partial(<class 'torch.ao.quantization.observer.MinMaxObserver'>, quant_min=0, quant_max=
Post Training Quantization Prepare: Inserting Observers

 Conv1: After observer insertion

 ConvReLU2d(
  (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), bias=False)
  (1): ReLU()
  (activation_post_process): MinMaxObserver(min_val=inf, max_val=-inf)
)
Post Training Quantization: Calibration done
Post Training Quantization: Convert done

 Conv1: After fusion and quantization

 QuantizedConvReLU2d(1, 6, kernel_size=(5, 5), stride=(1, 1), scale=0.05678582563996315, zero_point=0, bias=False)
Size of model after quantization
Size (MB): 0.050084
```

[11]
```
score = test(qnet, testloader, cuda=False)
print('Accuracy of the fused and quantized network on the test images: {}% - INT8'.format(score))
```

```
Accuracy of the fused and quantized network on the test images: 98.1% - INT8
```

We can also define a cusom quantization configuration, where we replace the default observers and instead of quantising with respect to

```
Accuracy of the fused and quantized network on the test images: 98.1% - INT8
```

We can also define a cusom quantization configuration, where we replace the default observers and instead of quantising with respect to max/min we can take an average of the observed max/min, hopefully for a better generalization performance.

```
[12] from torch.quantization.observer import MovingAverageMinMaxObserver

qnet = Net(q=True)
load_model(qnet, net)
fuse_modules(qnet)

qnet.qconfig = torch.quantization.QConfig(
                                     activation=MovingAverageMinMaxObserver.with_args(reduce_range=True),
                                     weight=MovingAverageMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_tensor_symmetric))
print(qnet.qconfig)
torch.quantization.prepare(qnet, inplace=True)
print('Post Training Quantization Prepare: Inserting Observers')
print('\n Conv1: After observer insertion \n\n', qnet.conv1)

test(qnet, trainloader, cuda=False)
print('Post Training Quantization: Calibration done')
torch.quantization.convert(qnet, inplace=True)
print('Post Training Quantization: Convert done')
print('\n Conv1: After fusion and quantization \n\n', qnet.conv1)
print("Size of model after quantization")
print_size_of_model(qnet)
score = test(qnet, testloader, cuda=False)
print('Accuracy of the fused and quantized network on the test images: {}% - INT8'.format(score))
```

```
QConfig(activation=functools.partial(<class 'torch.ao.quantization.observer.MovingAverageMinMaxObserver'>, reduce_range=True){}, weight=functools.partial(<class 'torch.ao.quantization.observer.MovingAverageMinMaxObserver'>, dtype=torch.qint8, qscheme=torch.per_tensor_symmetric){})
Post Training Quantization Prepare: Inserting Observers

 Conv1: After observer insertion

 ConvReLU2d(
  (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), bias=False)
  (1): ReLU()
  (activation_post_process): MovingAverageMinMaxObserver(min_val=inf, max_val=-inf)
)
/usr/local/lib/python3.11/dist-packages/torch/ao/quantization/observer.py:229: UserWarning: Please use quant_min and quant_max to specify the range for observers.                    reduce_range will be deprecated in a future release of PyTorch.
  warnings.warn(
Post Training Quantization: Calibration done
Post Training Quantization: Convert done

 Conv1: After fusion and quantization

 QuantizedConvReLU2d(1, 6, kernel_size=(5, 5), stride=(1, 1), scale=0.05639106035232544, zero_point=0, bias=False)
Size of model after quantization
Size (MB): 0.050084
Accuracy of the fused and quantized network on the test images: 98.14% - INT8
```

In addition, we can significantly improve on the accuracy simply by using a different quantization configuration. We repeat the same exercise with the recommended configuration for quantizing for arm64 architecture (qnnpack). This configuration does the following: Quantizes weights on a per-channel basis. It uses a histogram observer that collects a histogram of activations and then picks quantization parameters in an optimal manner.

+ Code + Text

```
[13] qnet = Net(q=True)
load_model(qnet, net)
fuse_modules(qnet)
```

```
[14] qnet.qconfig = torch.quantization.get_default_qconfig('qnnpack')
print(qnet.qconfig)

torch.quantization.prepare(qnet, inplace=True)
test(qnet, trainloader, cuda=False)
torch.quantization.convert(qnet, inplace=True)
```

✓ 2m 39s completed at 2:22 PM


```
[14] print(qnet.qconfig)

torch.quantization.prepare(qnet, inplace=True)
test(qnet, trainloader, cuda=False)
torch.quantization.convert(qnet, inplace=True)
```


```
print("Size of model after quantization")
print_size_of_model(qnet)
```

```
QConfig(activation=functools.partial(<class 'torch.ao.quantization.observer.HistogramObserver'>, reduce_range=False){}, weight=functools.partial(<class 'torch.ao.quantization.observer.MinMaxObserver'>, dtype=torch.qint8, qscheme=torch.per_tensor_symmetric){})
Size of model after quantization
Size (MB): 0.050084
```

```
[15] score = test(qnet, testloader, cuda=False)
print('Accuracy of the fused and quantized network on the test images: {}% - INT8'.format(score))
```

```
Accuracy of the fused and quantized network on the test images: 97.76% - INT8
```

Quantization aware training

Quantization-aware training (QAT) is the quantization method that typically results in the highest accuracy. With QAT, all weights and activations are "fake quantized" during both the forward and backward passes of training: that is, float values are rounded to mimic int8 values, but all computations are still done with floating point numbers.

```
qnet = Net(q=True)
fuse_modules(qnet)
qnet.qconfig = torch.quantization.get_default_qat_qconfig('qnnpack')
torch.quantization.prepare_qat(qnet, inplace=True)
print('\n Conv1: After fusion and quantization \n\n', qnet.conv1)
qnet=qnet.cuda()
train(qnet, trainloader, cuda=True)
qnet = qnet.cpu()
torch.quantization.convert(qnet, inplace=True)
print("Size of model after quantization")
print_size_of_model(qnet)

score = test(qnet, testloader, cuda=False)
print('Accuracy of the fused and quantized network (trained quantized) on the test images: {}% - INT8'.format(score))
```

```
 Conv1: After fusion and quantization

 ConvReLU2d(
  1, 6, kernel_size=(5, 5), stride=(1, 1), bias=False
  (weight_fake_quant): FusedMovingAvgObsFakeQuantize(
    fake_quant_enabled=tensor([1]), observer_enabled=tensor([1]), scale=tensor([1.]), zero_point=tensor([0], dtype=torch.int32), dtype=torch.qint8, quant_min=-128, quant_max=127, qscheme=torch.per_tensor_symmetric, reduce_range=False
    (activation_post_process): MovingAverageMinMaxObserver(min_val=inf, max_val=-inf)
  )
  (activation_post_process): FusedMovingAvgObsFakeQuantize(
    fake_quant_enabled=tensor([1]), observer_enabled=tensor([1]), scale=tensor([1.]), zero_point=tensor([0], dtype=torch.int32), dtype=torch.quint8, quant_min=0, quant_max=255, qscheme=torch.per_tensor_affine, reduce_range=False
    (activation_post_process): MovingAverageMinMaxObserver(min_val=inf, max_val=-inf)
  )
)
[1,     1]  loss 2.301067 (2.301067) train_acc 17.187500 (17.187500)
[1,   101]  loss 2.299710 (2.300782) train_acc 23.437500 (21.101485)
[1,   201]  loss 2.289709 (2.297257) train_acc 35.937500 (25.746269)
[1,   301]  loss 2.283399 (2.293284) train_acc 42.187500 (31.047550)
[1,   401]  loss 2.263690 (2.288294) train_acc 65.625000 (36.697319)
[1,   501]  loss 2.241506 (2.281470) train_acc 59.375000 (41.473303)
[1,   601]  loss 2.201498 (2.271854) train_acc 64.062500 (45.265703)
[1,   701]  loss 2.123823 (2.256817) train_acc 76.562500 (48.691601)
[1,   801]  loss 1.948928 (2.231720) train_acc 79.687500 (51.587859)
[1,   901]  loss 1.659243 (2.188986) train_acc 85.937500 (54.293840)
[2,     1]  loss 1.584189 (1.584189) train_acc 73.437500 (73.437500)
[2,   101]  loss 1.144466 (1.339404) train_acc 73.437500 (79.084158)
[2,   201]  loss 0.645077 (1.114868) train_acc 87.500000 (80.503731)
[2,   301]  loss 0.595823 (0.944724) train_acc 79.687500 (81.753530)
[2,   401]  loss 0.378817 (0.823271) train_acc 93.750000 (83.065773)
[2,   501]  loss 0.356126 (0.737998) train_acc 85.937500 (84.100549)
[2,   601]  loss 0.329685 (0.670119) train_acc 89.062500 (85.121152)
[2,   701]  loss 0.282937 (0.622260) train_acc 93.750000 (85.808220)
[2,   801]  loss 0.288164 (0.579900) train_acc 87.500000 (86.522706)
[2,   901]  loss 0.380598 (0.547013) train_acc 90.625000 (87.059517)
```

✓ 2m 39s completed at 2:22 PM

```
[10,   901]  loss 0.043493 (0.059836) train_acc 98.437500 (98.153094)
Finished Training
Size of model after quantization
Size (MB): 0.050084
Accuracy of the fused and quantized network (trained quantized) on the test images: 98.03% - INT8
```