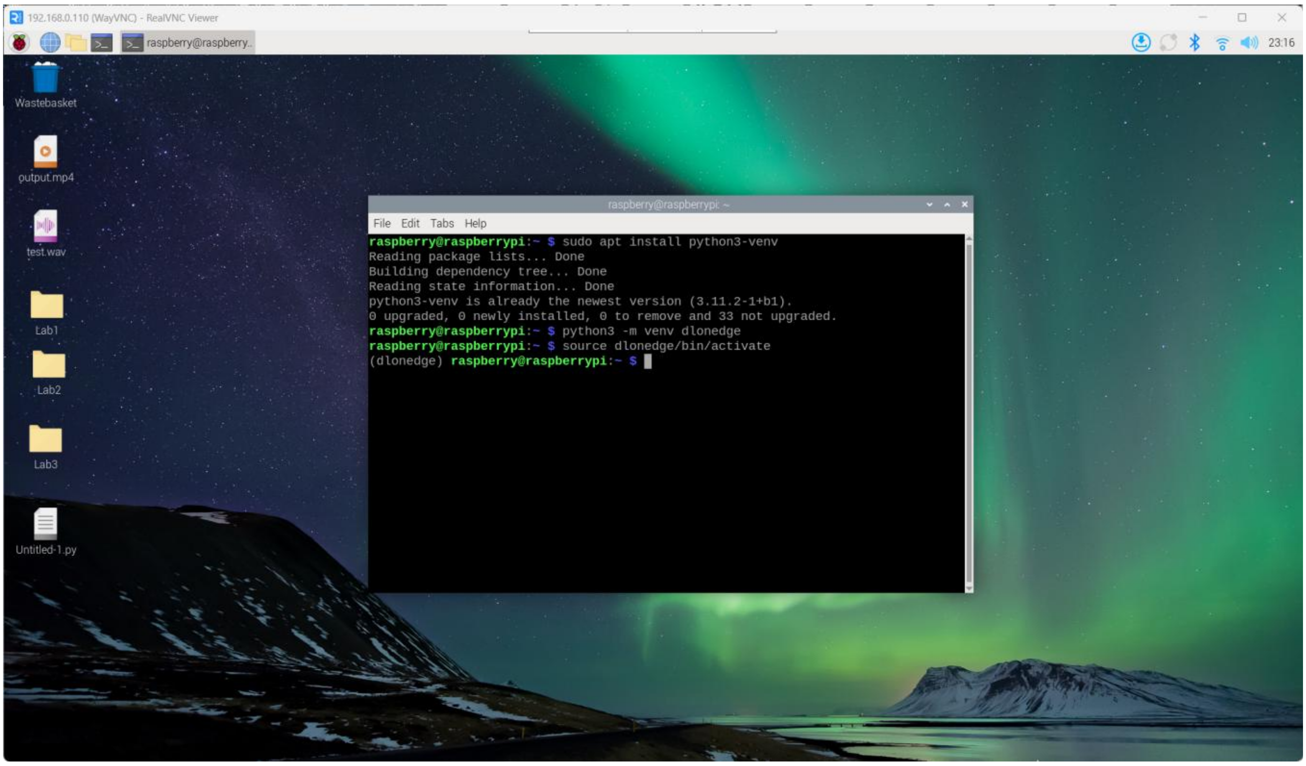
# Lab04 (DLonEdge) Screenshots

## 1. Introduction

Edge analytics with real-time processing capabilities is challenging but important and inevitable due to privacy/security concerns. However, edge devices like RaspberryPi are constrained with limited hardware resources, which at times are not sufficient to run complex deep learning models. These models require lot of computational resource and memory due to their size and complex architecture. Therefore, in such scenarios, we optimize the model such that it can run efficiently with reduced inference time critical for real-time analytics. Optimization can be achieved by combination of techniques like quantization and converting trained model into architecture specific lite model.

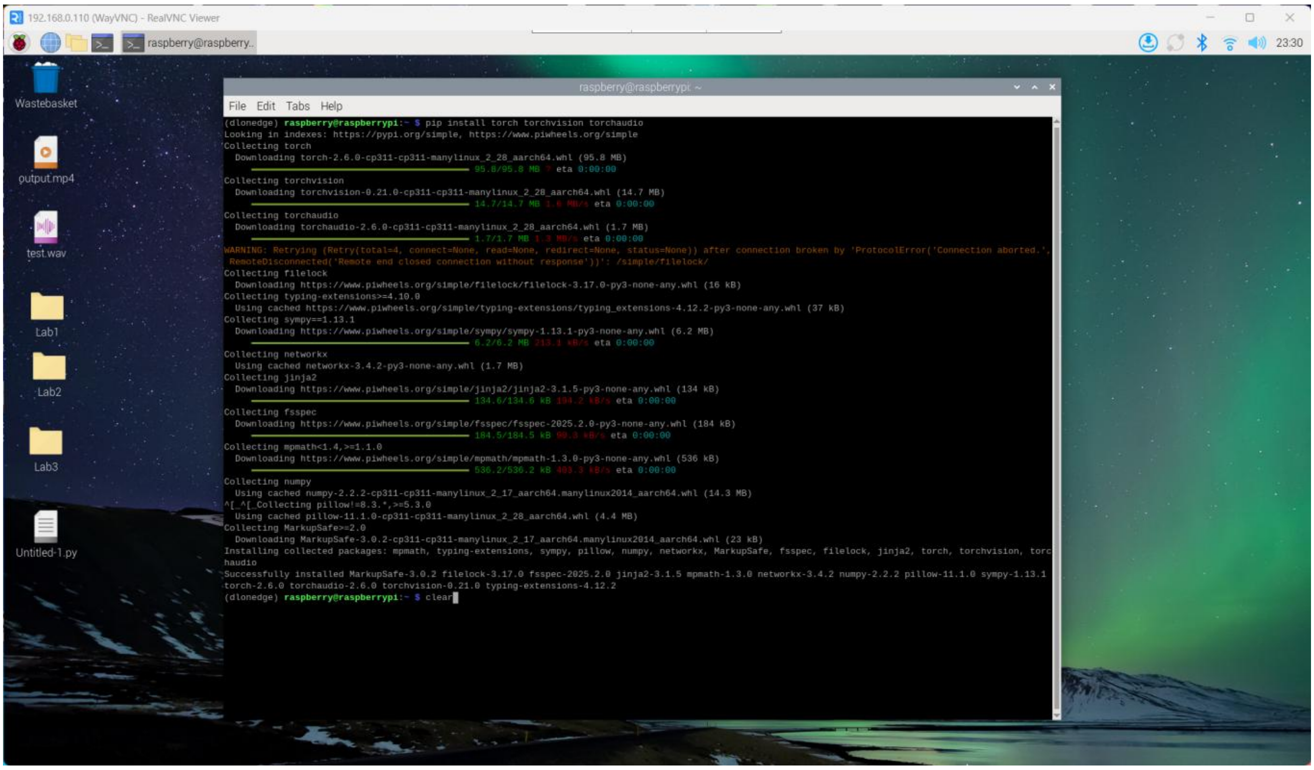
## 2. Running Deep Learning Model On RaspberryPi



```
raspberry@raspberrypi:~ $ sudo apt install python3-venv
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-venv is already the newest version (3.11.2-1+b1).
0 upgraded, 0 newly installed, 0 to remove and 33 not upgraded.
raspberry@raspberrypi:~ $ python3 -m venv dlonedge
raspberry@raspberrypi:~ $ source dlonedge/bin/activate
(dlonedge) raspberry@raspberrypi:~ $
```

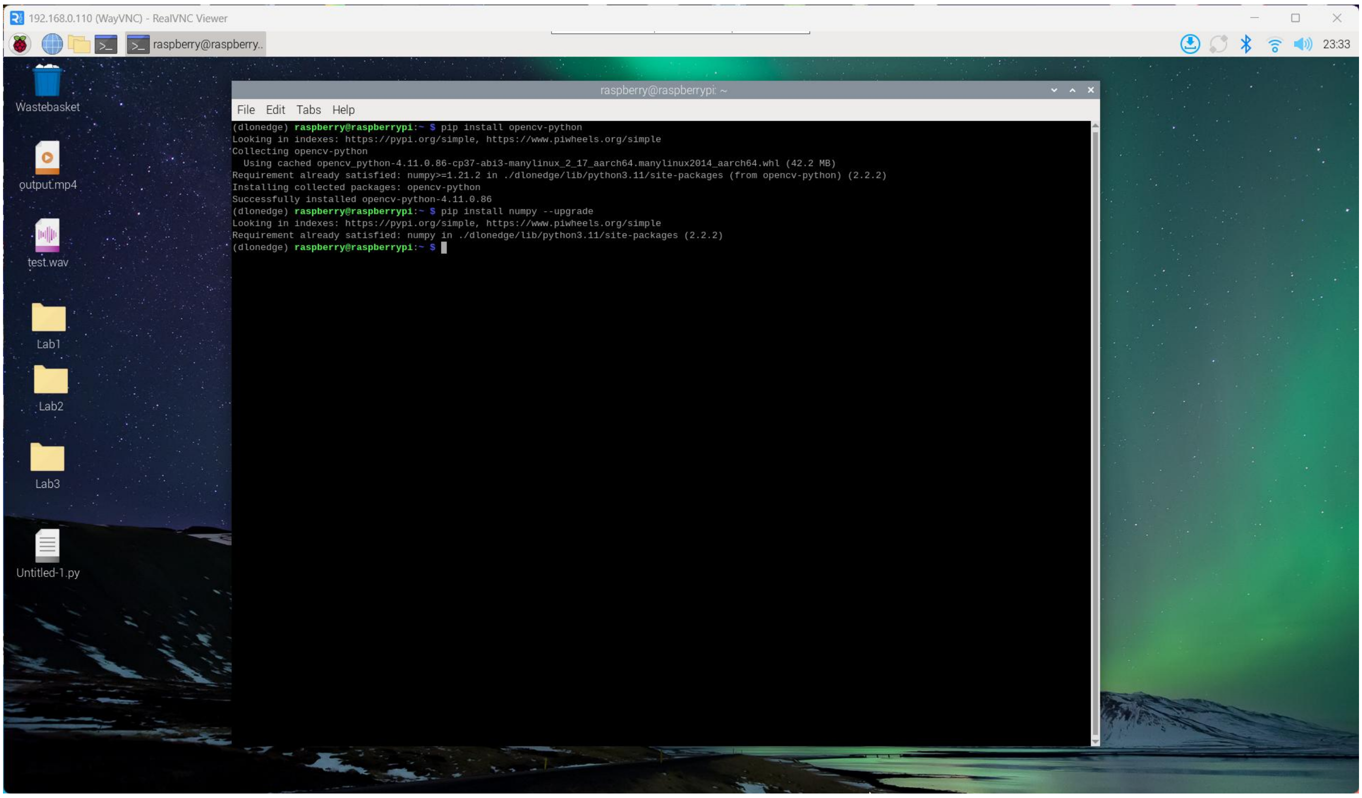
Fig. 1. Screenshot of activating a virtual environment named “dlonedge” to avoid conflicts in libraries.

# Lab04 (DLonEdge) Screenshots



```
(dlonedge) raspberry@raspberrypi:~ $ pip install torch torchvision torchaudio
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting torch
  Downloading torch-2.6.0-cp311-cp311-manylinux_2_28_aarch64.whl (95.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.8/95.8 MB ? eta 0:00:00
Collecting torchvision
  Downloading torchvision-0.21.0-cp311-cp311-manylinux_2_28_aarch64.whl (14.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.7/14.7 MB 1.6 MB/s eta 0:00:00
Collecting torchaudio
  Downloading torchaudio-2.6.0-cp311-cp311-manylinux_2_28_aarch64.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 1.3 MB/s eta 0:00:00
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProtocolError('Connection aborted.',
 RemoteDisconnected('Remote end closed connection without response'))': /simple/filelock/
Collecting filelock
  Downloading https://www.piwheels.org/simple/filelock/filelock-3.17.0-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.10.0
  Using cached https://www.piwheels.org/simple/typing-extensions/typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Collecting sympy==1.13.1
  Downloading https://www.piwheels.org/simple/sympy/sympy-1.13.1-py3-none-any.whl (6.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 213.1 kB/s eta 0:00:00
Collecting networkx
  Using cached networkx-3.4.2-py3-none-any.whl (1.7 MB)
Collecting jinja2
  Downloading https://www.piwheels.org/simple/jinja2/jinja2-3.1.5-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.6/134.6 kB 194.2 kB/s eta 0:00:00
Collecting fsspec
  Downloading https://www.piwheels.org/simple/fsspec/fsspec-2025.2.0-py3-none-any.whl (184 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 184.5/184.5 kB 90.3 kB/s eta 0:00:00
Collecting mpmath<1.4,>=1.1.0
  Downloading https://www.piwheels.org/simple/mpmath/mpmath-1.3.0-py3-none-any.whl (536 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.2/536.2 kB 403.3 kB/s eta 0:00:00
Collecting numpy
  Using cached numpy-2.2.2-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (14.3 MB)
^[_^[_Collecting pillow!=8.3.*,>=5.3.0
  Using cached pillow-11.1.0-cp311-cp311-manylinux_2_28_aarch64.whl (4.4 MB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-3.0.2-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (23 kB)
Installing collected packages: mpmath, typing-extensions, sympy, pillow, numpy, networkx, MarkupSafe, fsspec, filelock, jinja2, torch, torchvision, torc
haudio
Successfully installed MarkupSafe-3.0.2 filelock-3.17.0 fsspec-2025.2.0 jinja2-3.1.5 mpmath-1.3.0 networkx-3.4.2 numpy-2.2.2 pillow-11.1.0 sympy-1.13.1
torch-2.6.0 torchaudio-2.6.0 torchvision-0.21.0 typing-extensions-4.12.2
(dlonedge) raspberry@raspberrypi:~ $ clear
```

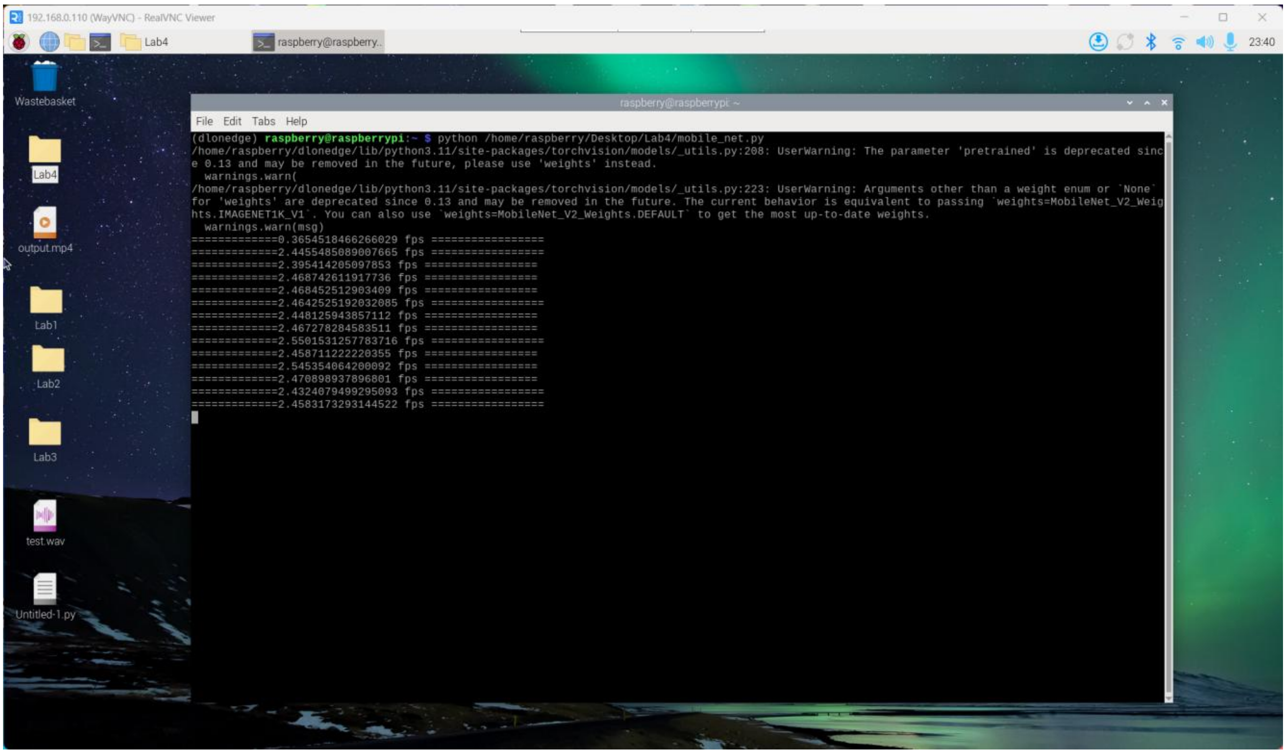
Fig. 2. Screenshot of installing PyTorch (`torch, torchvision, torchaudio`)



```
(dlonedge) raspberry@raspberrypi:~ $ pip install opencv-python
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting opencv-python
  Using cached opencv_python-4.11.0.86-cp37-abi3-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (42.2 MB)
Requirement already satisfied: numpy>=1.21.2 in ./dlonedge/lib/python3.11/site-packages (from opencv-python) (2.2.2)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.11.0.86
(dlonedge) raspberry@raspberrypi:~ $ pip install numpy --upgrade
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: numpy in ./dlonedge/lib/python3.11/site-packages (2.2.2)
(dlonedge) raspberry@raspberrypi:~ $
```

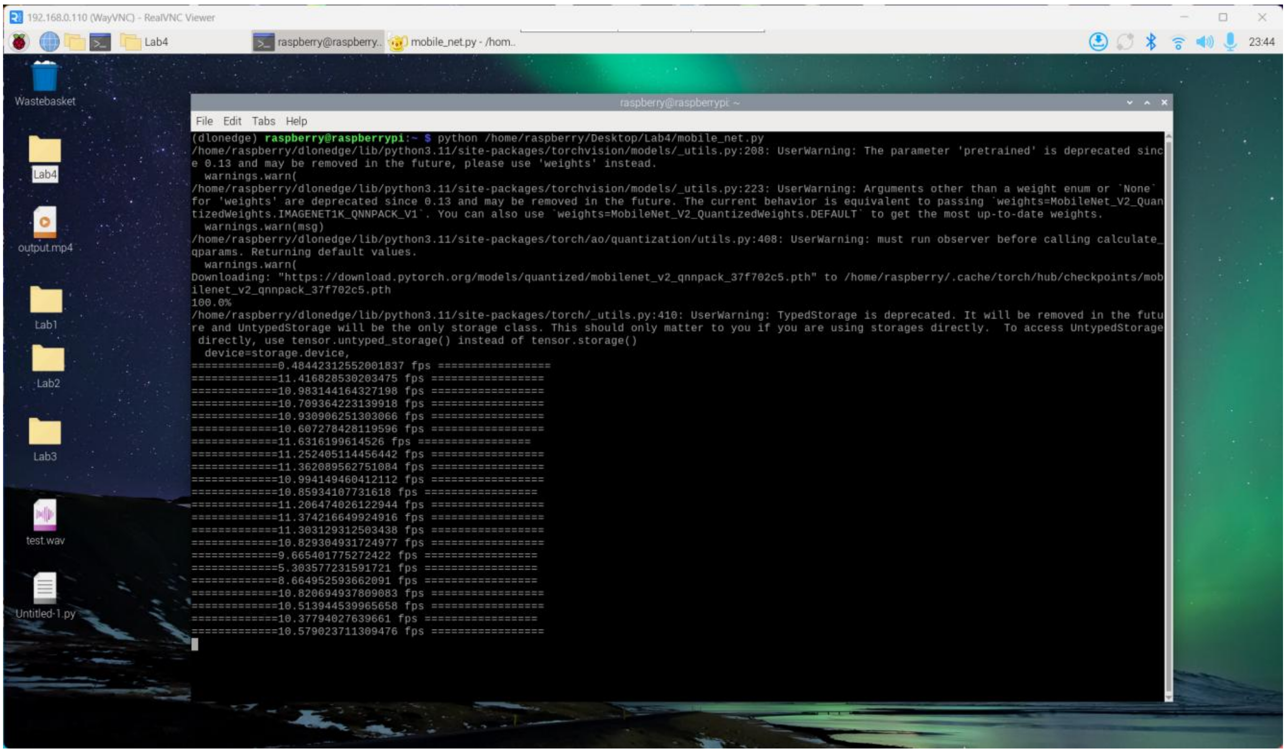
Fig. 2. Screenshot of installing OpenCV (`opencv-python`) and upgrade `numpy`

Lab04 (DLonEdge) Screenshots



```
(dlonedge) raspberry@raspberrypi:~ $ python /home/raspberry/Desktop/Lab4/mobile_net.py
/home/raspberry/dlonedge/lib/python3.11/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
e 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/home/raspberry/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 and may be removed in the future. The current behavior is equivalent to passing `weights=MobileNet_V2_Weig
hts.IMAGENET1K_V1`. You can also use `weights=MobileNet_V2_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
=============0.3654518466266029 fps =================
=============2.4455485089007665 fps =================
=============2.395414205097853 fps =================
=============2.468742611917736 fps =================
=============2.468452512903409 fps =================
=============2.4642525192032085 fps =================
=============2.448125943857112 fps =================
=============2.467278284583511 fps =================
=============2.5501531257783716 fps =================
=============2.458711222220355 fps =================
=============2.545354064200092 fps =================
=============2.470898937896801 fps =================
=============2.4324079499295093 fps =================
=============2.4583173293144522 fps =================
```

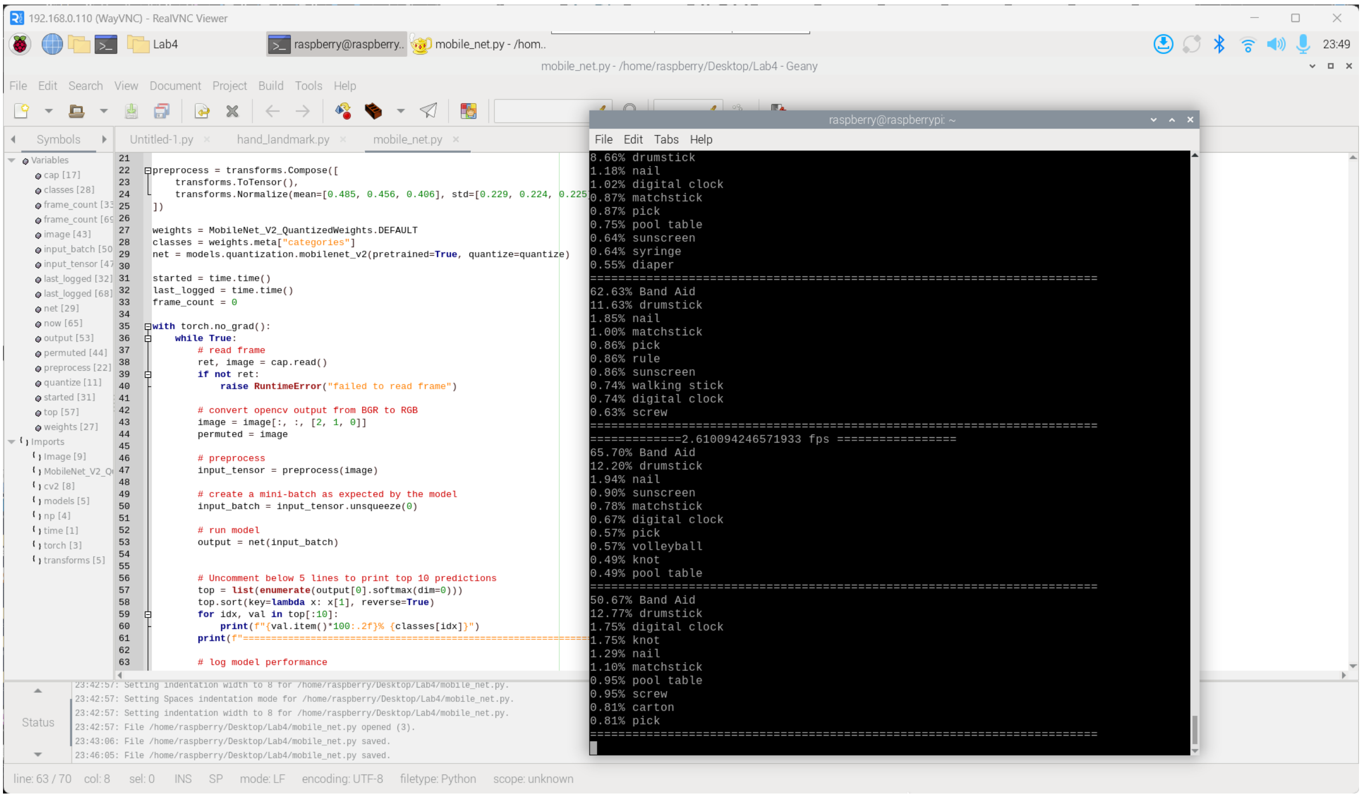
Fig. 3. Screenshot of doing model inference on local pre-trained MobileNetV2 model, with no optimization of model and could only achieve 2-3 FPS on Raspberry Pi 3B+



```
(dlonedge) raspberry@raspberrypi:~ $ python /home/raspberry/Desktop/Lab4/mobile_net.py
/home/raspberry/dlonedge/lib/python3.11/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
e 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/home/raspberry/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 and may be removed in the future. The current behavior is equivalent to passing `weights=MobileNet_V2_Quan
tizedWeights.IMAGENET1K_QNNPACK_V1`. You can also use `weights=MobileNet_V2_QuantizedWeights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
/home/raspberry/dlonedge/lib/python3.11/site-packages/torch/ao/quantization/utils.py:408: UserWarning: must run observer before calling calculate_
qparams. Returning default values.
  warnings.warn(
Downloading: "https://download.pytorch.org/models/quantized/mobilenet_v2_qnnpack_37f702c5.pth" to /home/raspberry/.cache/torch/hub/checkpoints/mob
ilenet_v2_qnnpack_37f702c5.pth
100.0%
/home/raspberry/dlonedge/lib/python3.11/site-packages/torch/_utils.py:410: UserWarning: TypedStorage is deprecated. It will be removed in the futu
re and UntypedStorage will be the only storage class. This should only matter to you if you are using storages directly.  To access UntypedStorage
 directly, use tensor.untyped_storage() instead of tensor.storage()
  device=storage.device,
=============0.48442312552001837 fps =================
=============11.416828530203475 fps =================
=============10.983144164327198 fps =================
=============10.709364223139918 fps =================
=============10.930906251303066 fps =================
=============10.607278428119596 fps =================
=============11.6316199614526 fps =================
=============11.252405114456442 fps =================
=============11.362089562751084 fps =================
=============10.994149460412112 fps =================
=============10.85934107731618 fps =================
=============11.206474026122944 fps =================
=============11.374216649924916 fps =================
=============11.303129312503438 fps =================
=============10.829304931724977 fps =================
=============9.665401775272422 fps =================
=============5.303577231591721 fps =================
=============8.664952593662091 fps =================
=============10.820694937809083 fps =================
=============10.513944539965658 fps =================
=============10.37794027639661 fps =================
=============10.579023711309476 fps =================
```

Fig. 4. Screenshot of doing model inference on pre-trained local MobileNetV2 model, with optimization of model (with quantization enabled) and could achieve 5-12 FPS on Raspberry Pi 3B+

# Lab04 (DLonEdge) Screenshots



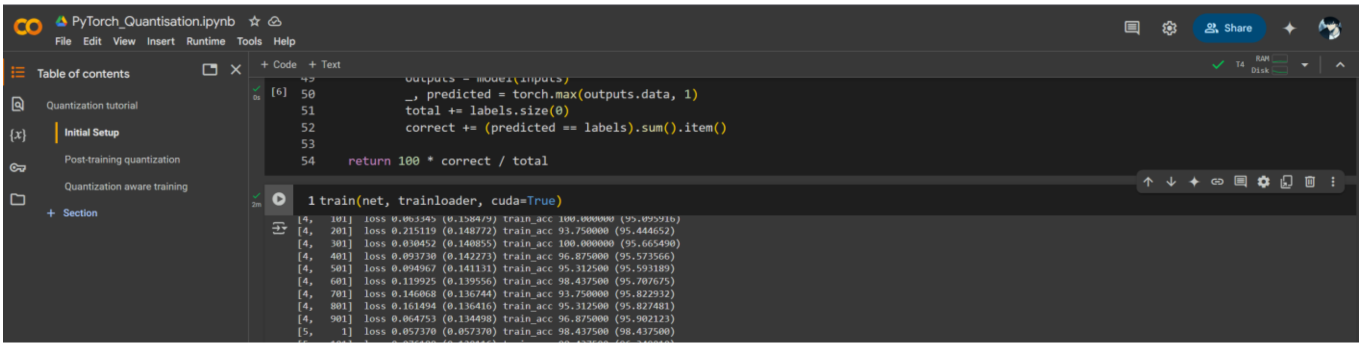
```python

preprocess = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225
])

weights = MobileNet_V2_QuantizedWeights.DEFAULT
classes = weights.meta["categories"]
net = models.quantization.mobilenet_v2(pretrained=True, quantize=quantize)

started = time.time()
last_logged = time.time()
frame_count = 0

with torch.no_grad():
    while True:
        # read frame
        ret, image = cap.read()
        if not ret:
            raise RuntimeError("failed to read frame")

        # convert opencv output from BGR to RGB
        image = image[:, :, [2, 1, 0]]
        permuted = image

        # preprocess
        input_tensor = preprocess(image)

        # create a mini-batch as expected by the model
        input_batch = input_tensor.unsqueeze(0)

        # run model
        output = net(input_batch)


        # Uncomment below 5 lines to print top 10 predictions
        top = list(enumerate(output[0].softmax(dim=0)))
        top.sort(key=lambda x: x[1], reverse=True)
        for idx, val in top[:10]:
            print(f"{val.item()*100:.2f}% {classes[idx]}")
        print(f"================================================================

        # log model performance
```

```
23:42:57: Setting indentation width to 8 for /home/raspberry/Desktop/Lab4/mobile_net.py.
23:42:57: Setting Spaces indentation mode for /home/raspberry/Desktop/Lab4/mobile_net.py.
23:42:57: Setting indentation width to 8 for /home/raspberry/Desktop/Lab4/mobile_net.py.
23:42:57: File /home/raspberry/Desktop/Lab4/mobile_net.py opened (3).
23:43:06: File /home/raspberry/Desktop/Lab4/mobile_net.py saved.
23:46:05: File /home/raspberry/Desktop/Lab4/mobile_net.py saved.
```

```
8.66% drumstick
1.18% nail
1.02% digital clock
0.87% matchstick
0.87% pick
0.75% pool table
0.64% sunscreen
0.64% syringe
0.55% diaper
==========================================================================
62.63% Band Aid
11.63% drumstick
1.85% nail
1.00% matchstick
0.86% pick
0.86% rule
0.86% sunscreen
0.74% walking stick
0.74% digital clock
0.63% screw
==========================================================================
============2.610094246571933 fps ================
65.70% Band Aid
12.20% drumstick
1.94% nail
0.90% sunscreen
0.78% matchstick
0.67% digital clock
0.57% pick
0.57% volleyball
0.49% knot
0.49% pool table
==========================================================================
50.67% Band Aid
12.77% drumstick
1.75% digital clock
1.75% knot
1.29% nail
1.10% matchstick
0.95% pool table
0.95% screw
0.81% carton
0.81% pick
==========================================================================
```

Fig. 5. Screenshot of printing the top 10 predictions made by the local pre-trained MobileNetV2 model according to streaming data sent through the webcam

## 3. Quantization using Pytorch


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
Section
+ Code + Text
T4 RAM Disk
[6]

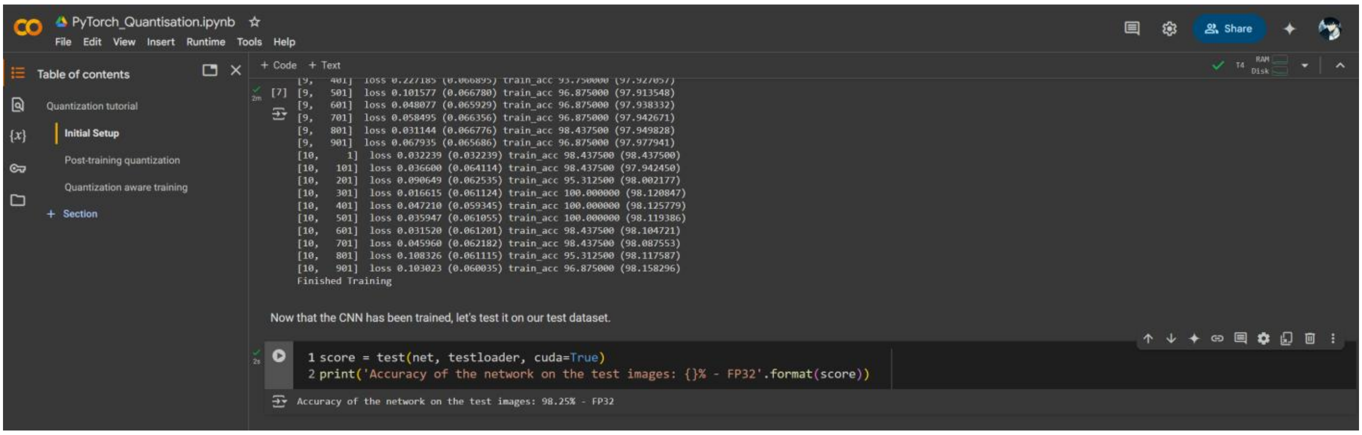
```python
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100 * correct / total
```

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

```
[4,   101] loss 0.063345 (0.158479) train_acc 100.000000 (95.095916)
[4,   201] loss 0.215119 (0.148772) train_acc 93.750000 (95.444652)
[4,   301] loss 0.030452 (0.140855) train_acc 100.000000 (95.665490)
[4,   401] loss 0.093730 (0.142273) train_acc 96.875000 (95.573566)
[4,   501] loss 0.094967 (0.141131) train_acc 95.312500 (95.593189)
[4,   601] loss 0.119925 (0.139556) train_acc 98.437500 (95.707675)
[4,   701] loss 0.146068 (0.136744) train_acc 93.750000 (95.822932)
[4,   801] loss 0.161494 (0.136416) train_acc 95.312500 (95.827481)
[4,   901] loss 0.064753 (0.134498) train_acc 96.875000 (95.902123)
[5,     1] loss 0.057370 (0.057370) train_acc 98.437500 (98.437500)
```

Fig. 6. Training of the CNN on the training dataset

# Lab04 (DLonEdge) Screenshots


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
Section
+ Code + Text
T4 RAM Disk

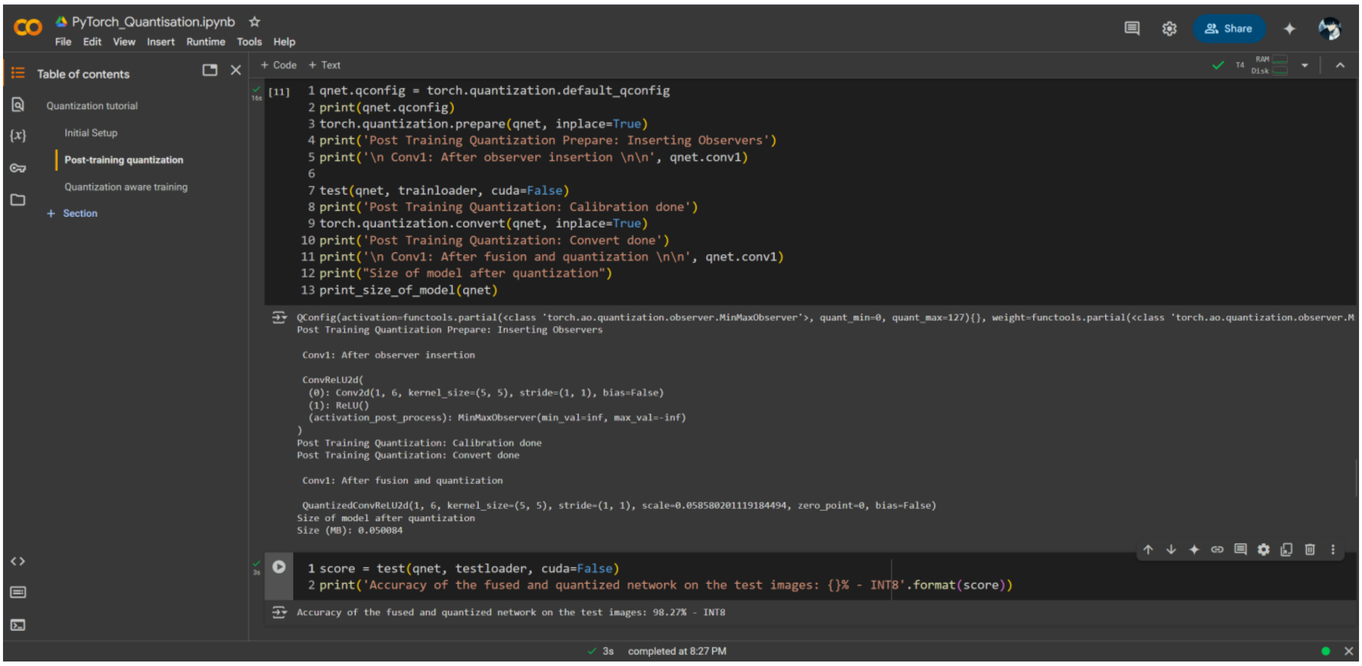
```
[9,   401]  loss 0.227185 (0.066895) train_acc 93.750000 (97.927057)
[9,   501]  loss 0.101577 (0.066780) train_acc 96.875000 (97.913548)
[9,   601]  loss 0.048077 (0.065929) train_acc 96.875000 (97.938332)
[9,   701]  loss 0.058495 (0.066356) train_acc 96.875000 (97.942671)
[9,   801]  loss 0.031144 (0.066776) train_acc 98.437500 (97.949828)
[9,   901]  loss 0.067935 (0.065686) train_acc 96.875000 (97.977941)
[10,    1]  loss 0.032239 (0.032239) train_acc 98.437500 (98.437500)
[10,  101]  loss 0.036600 (0.064114) train_acc 98.437500 (97.942450)
[10,  201]  loss 0.090649 (0.062535) train_acc 95.312500 (98.002177)
[10,  301]  loss 0.016615 (0.061124) train_acc 100.000000 (98.120847)
[10,  401]  loss 0.047210 (0.059345) train_acc 100.000000 (98.125779)
[10,  501]  loss 0.035947 (0.061055) train_acc 100.000000 (98.119386)
[10,  601]  loss 0.031520 (0.061201) train_acc 98.437500 (98.104721)
[10,  701]  loss 0.045960 (0.062182) train_acc 98.437500 (98.087553)
[10,  801]  loss 0.108326 (0.061115) train_acc 95.312500 (98.117587)
[10,  901]  loss 0.103023 (0.060035) train_acc 96.875000 (98.158296)
Finished Training
```

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

```python
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: 98.25% - FP32
```

Fig. 7. Testing the CNN on the test dataset, returning accuracy score of 98.25


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
Section
+ Code + Text
T4 RAM Disk
3s completed at 8:27 PM

```python
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)
```

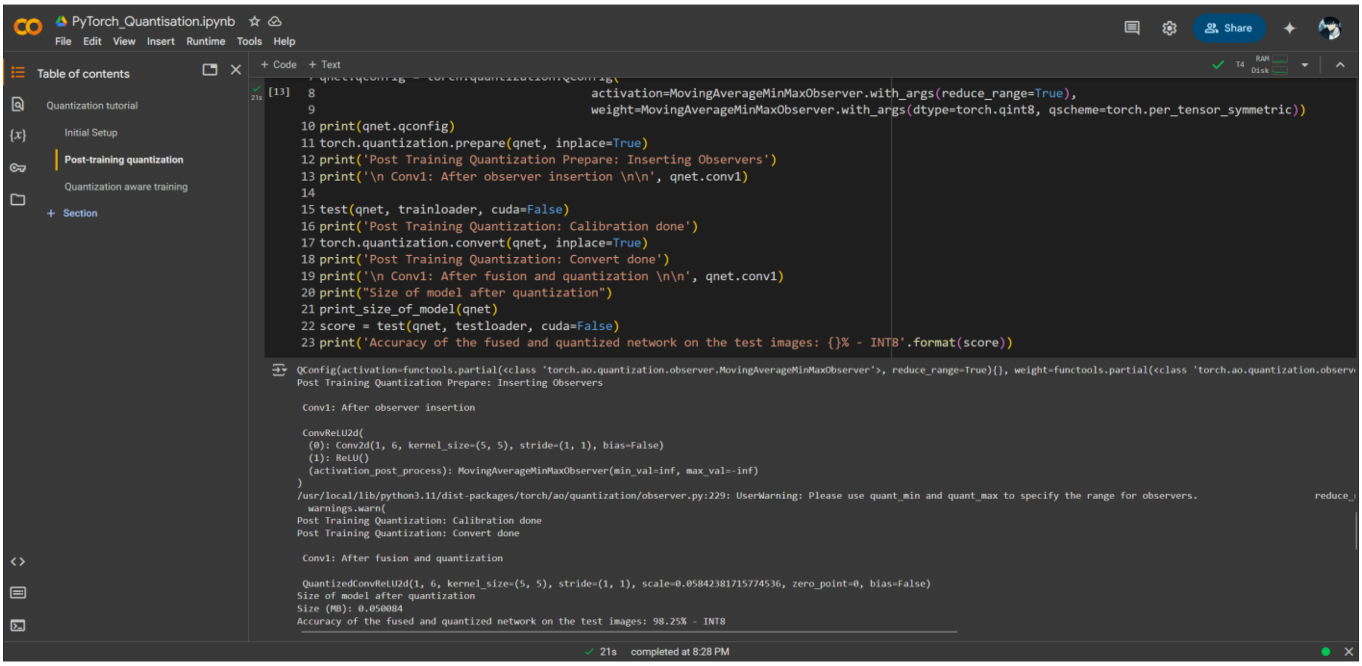
```
QConfig(activation=functools.partial(<class 'torch.ao.quantization.observer.MinMaxObserver'>, quant_min=0, quant_max=127){}, weight=functools.partial(<class 'torch.ao.quantization.observer.M
Post Training Quantization Prepare: Inserting Observers

 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=0.058580201119184494, zero_point=0, bias=False)
Size of model after quantization
Size (MB): 0.050084
```

```python
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: 98.27% - INT8
```

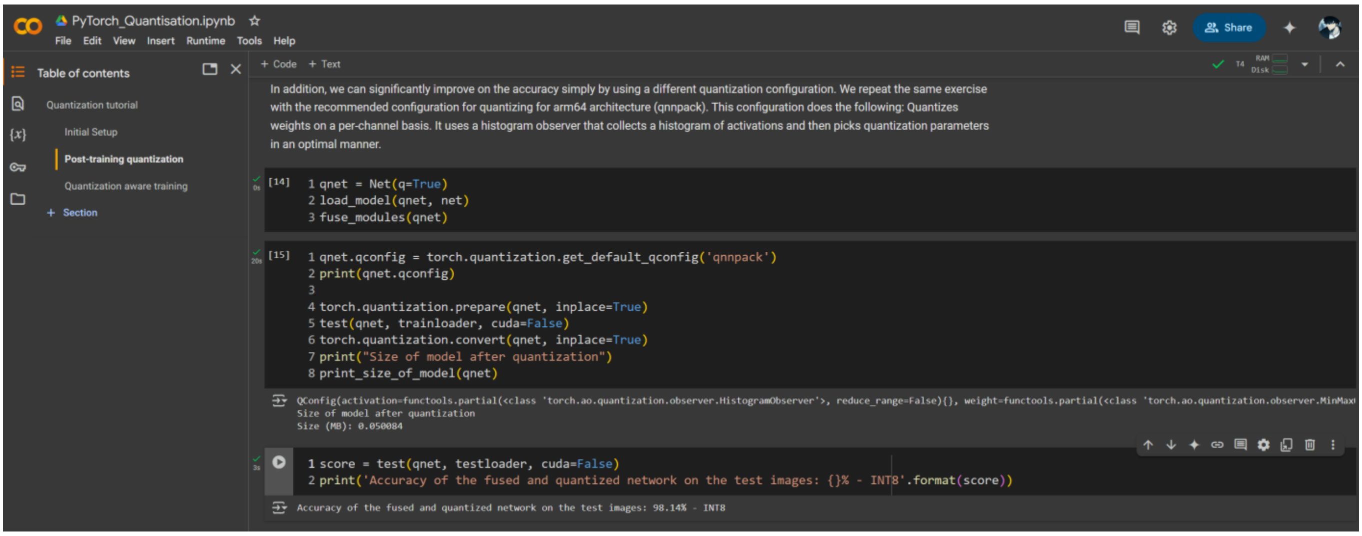
Fig. 8. Post-training quantization, with accuracy score of 98.27%

# Lab04 (DLonEdge) Screenshots


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
Section
+ Code + Text
T4 RAM Disk
[13]
21s completed at 8:28 PM

```
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.observ
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_
  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=0.05842381715774536, 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: 98.25% - INT8
```


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
Section
+ Code + Text
T4 RAM Disk
[14]
[15]

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.

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

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

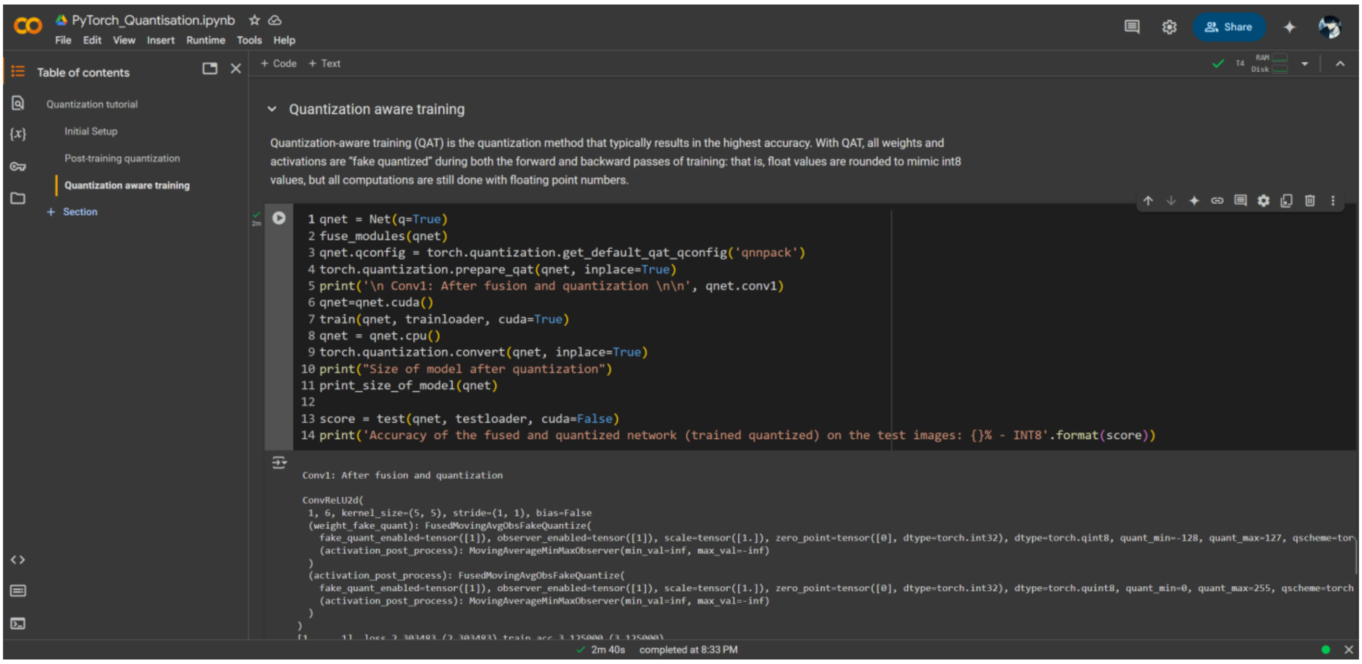
```
QConfig(activation=functools.partial(<class 'torch.ao.quantization.observer.HistogramObserver'>, reduce_range=False){}, weight=functools.partial(<class 'torch.ao.quantization.observer.MinMax
Size of model after quantization
Size (MB): 0.050084
```

```
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: 98.14% - INT8
```

Fig 9. Result of a custom quantization configuration with accuracy of 98.25%

# Lab04 (DLonEdge) Screenshots


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
+ Section
+ Code + Text
T4 RAM Disk

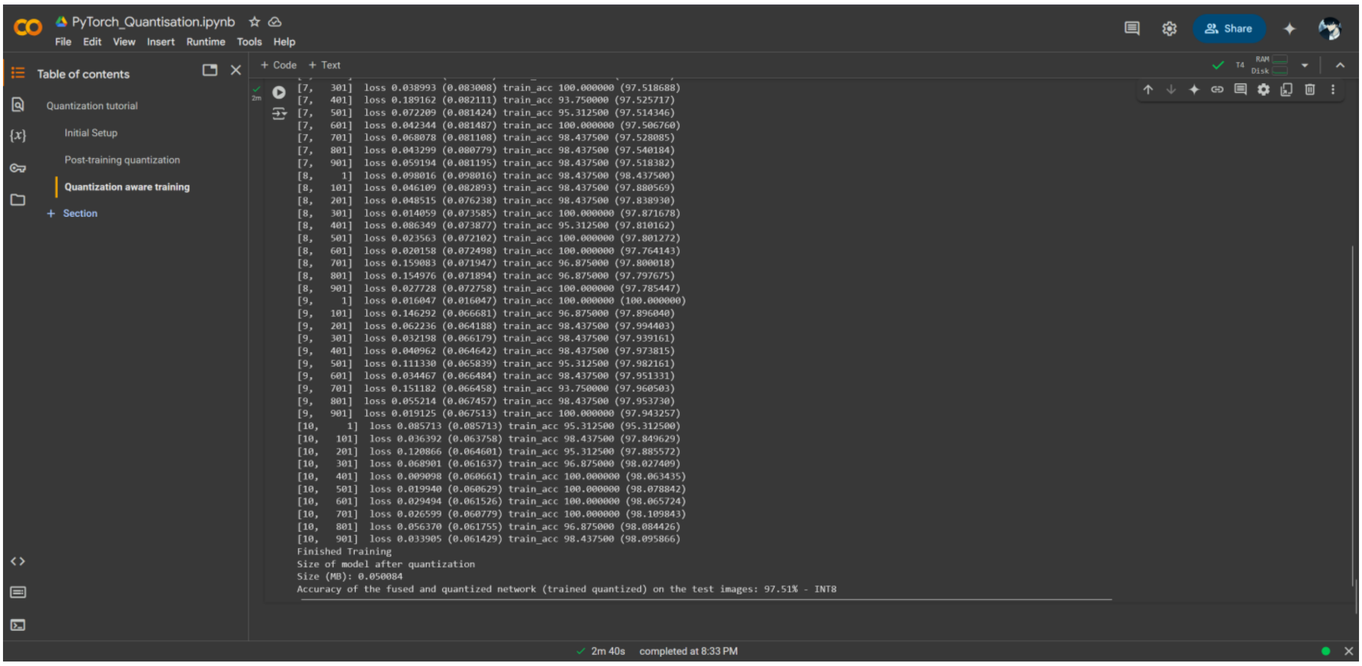
## Quantization aware training

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=tor
   (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
   (activation_post_process): MovingAverageMinMaxObserver(min_val=inf, max_val=-inf)
 )
)
[1,     1] loss 2.303483 (2.303483) train_acc 3.125000 (3.125000)
```

✓ 2m 40s completed at 8:33 PM


PyTorch_Quantisation.ipynb
File Edit View Insert Runtime Tools Help
Share
Table of contents
Quantization tutorial
Initial Setup
Post-training quantization
Quantization aware training
+ Section
+ Code + Text
T4 RAM Disk

```
[7,   301] loss 0.038993 (0.083008) train_acc 100.000000 (97.518688)
[7,   401] loss 0.189162 (0.082111) train_acc 93.750000 (97.525717)
[7,   501] loss 0.072209 (0.081424) train_acc 95.312500 (97.514346)
[7,   601] loss 0.042344 (0.081487) train_acc 100.000000 (97.506760)
[7,   701] loss 0.068078 (0.081108) train_acc 98.437500 (97.528085)
[7,   801] loss 0.043299 (0.080779) train_acc 98.437500 (97.540184)
[7,   901] loss 0.059194 (0.081195) train_acc 98.437500 (97.518382)
[8,     1] loss 0.098016 (0.098016) train_acc 98.437500 (98.437500)
[8,   101] loss 0.046109 (0.082893) train_acc 98.437500 (97.880569)
[8,   201] loss 0.048515 (0.076238) train_acc 98.437500 (97.838930)
[8,   301] loss 0.014059 (0.073585) train_acc 100.000000 (97.871678)
[8,   401] loss 0.086349 (0.073877) train_acc 95.312500 (97.810162)
[8,   501] loss 0.023563 (0.072102) train_acc 100.000000 (97.801272)
[8,   601] loss 0.020158 (0.072498) train_acc 100.000000 (97.764143)
[8,   701] loss 0.159083 (0.071947) train_acc 96.875000 (97.800018)
[8,   801] loss 0.154976 (0.071894) train_acc 96.875000 (97.797675)
[8,   901] loss 0.027728 (0.072758) train_acc 100.000000 (97.785447)
[9,     1] loss 0.016047 (0.016047) train_acc 100.000000 (100.000000)
[9,   101] loss 0.146292 (0.066681) train_acc 96.875000 (97.896040)
[9,   201] loss 0.062236 (0.064188) train_acc 98.437500 (97.994403)
[9,   301] loss 0.032198 (0.066179) train_acc 98.437500 (97.939161)
[9,   401] loss 0.040962 (0.064642) train_acc 98.437500 (97.973815)
[9,   501] loss 0.111330 (0.065839) train_acc 95.312500 (97.982161)
[9,   601] loss 0.034467 (0.066484) train_acc 98.437500 (97.951331)
[9,   701] loss 0.151182 (0.066458) train_acc 93.750000 (97.960503)
[9,   801] loss 0.055214 (0.067457) train_acc 98.437500 (97.953730)
[9,   901] loss 0.019125 (0.067513) train_acc 100.000000 (97.943257)
[10,     1] loss 0.085713 (0.085713) train_acc 95.312500 (95.312500)
[10,   101] loss 0.036392 (0.063758) train_acc 98.437500 (97.849629)
[10,   201] loss 0.120866 (0.064601) train_acc 95.312500 (97.885572)
[10,   301] loss 0.068901 (0.061637) train_acc 96.875000 (98.027409)
[10,   401] loss 0.009098 (0.060661) train_acc 100.000000 (98.063435)
[10,   501] loss 0.019940 (0.060629) train_acc 100.000000 (98.078842)
[10,   601] loss 0.029494 (0.061526) train_acc 100.000000 (98.065724)
[10,   701] loss 0.026599 (0.060779) train_acc 100.000000 (98.109843)
[10,   801] loss 0.056370 (0.061755) train_acc 96.875000 (98.084426)
[10,   901] loss 0.033905 (0.061429) train_acc 98.437500 (98.095866)
Finished Training
Size of model after quantization
Size (MB): 0.050084
Accuracy of the fused and quantized network (trained quantized) on the test images: 97.51% - INT8
```

✓ 2m 40s completed at 8:33 PM

Fig 10. Quantization Aware Training (QAT), with accuracy result of 97.51%