Lab 4. Deploy ONNX model on Pi5

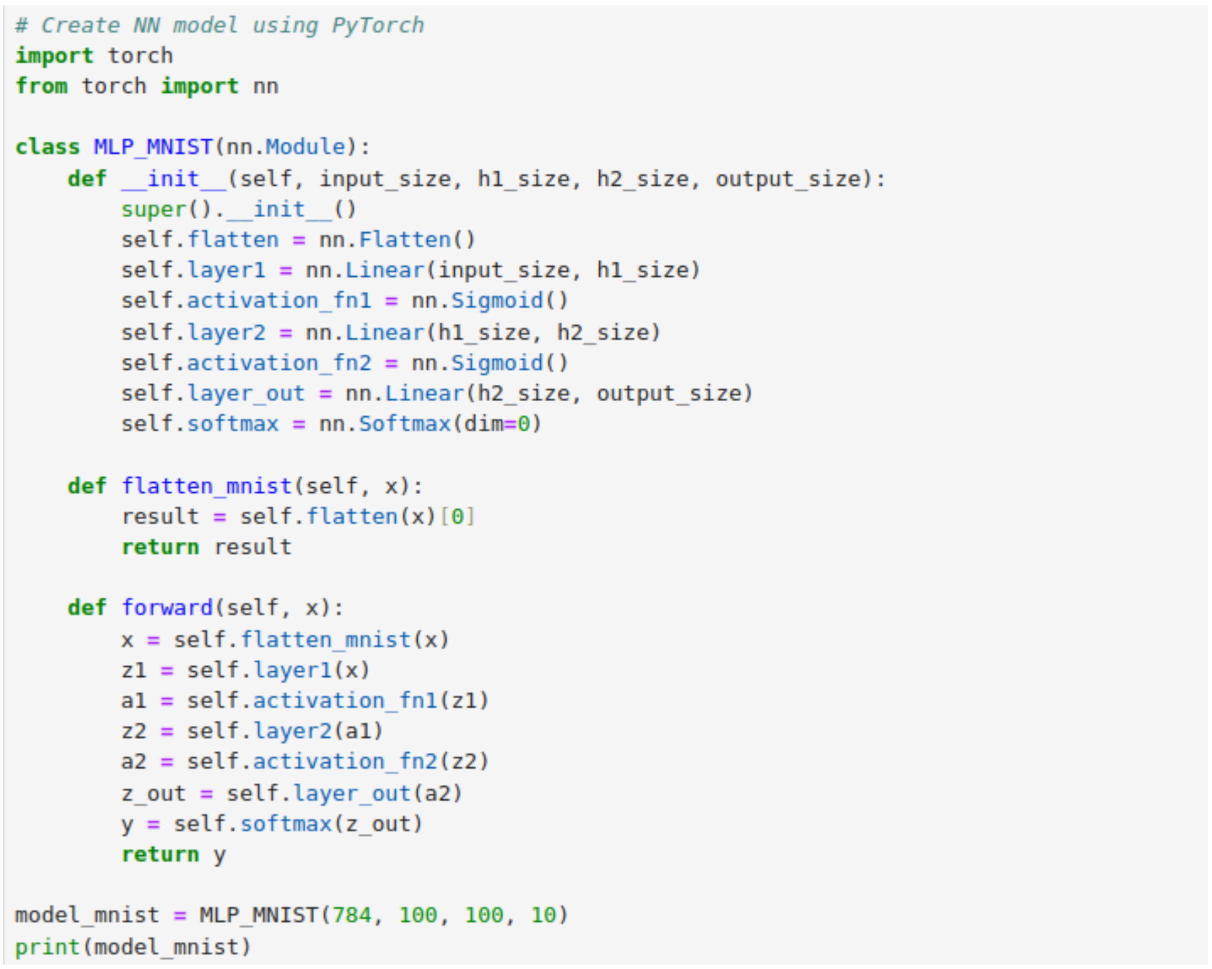
Objective

To **deploy** the **ONNX model** as the open-format model and use it on the embedded device i.e. Raspberry Pi 5. An ONNX model in this lab will be generated from the Pytorch MNIST handwritten model from the previous lab. The **OpenCV** will be used for the image acquisition in real-time MNIST handwritten classification.

Experiment 1 : Create an ANN model for MNIST handwritten digit using PyTorch

Step 1.

Create and train the ANN for MNIST handwritten digit model with PyTorch in the previous lab, by using the following .ipynb code.



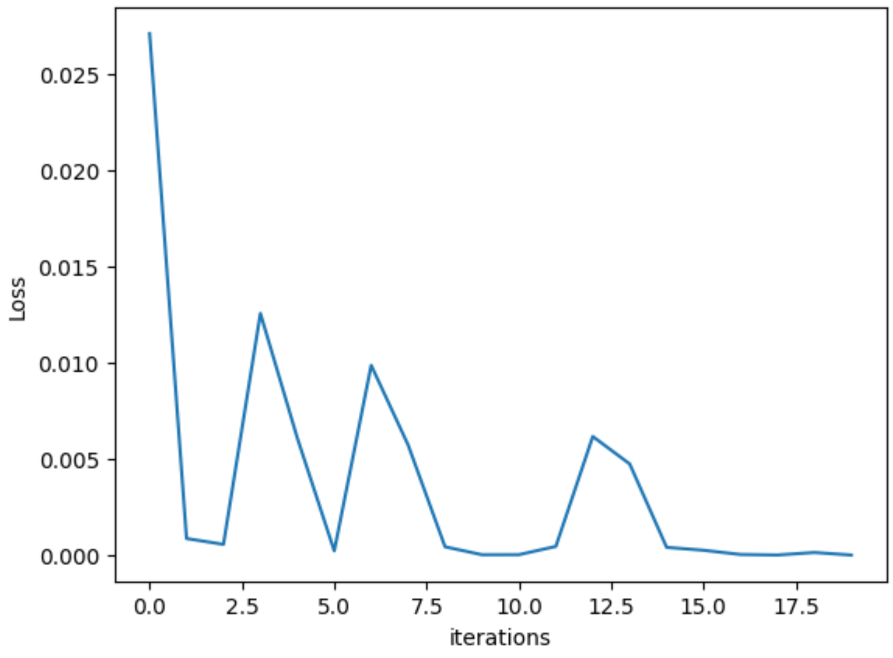
```python
# Create NN model using PyTorch
import torch
from torch import nn

class MLP_MNIST(nn.Module):
    def __init__(self, input_size, h1_size, h2_size, output_size):
        super().__init__()
        self.flatten = nn.Flatten()
        self.layer1 = nn.Linear(input_size, h1_size)
        self.activation_fn1 = nn.Sigmoid()
        self.layer2 = nn.Linear(h1_size, h2_size)
        self.activation_fn2 = nn.Sigmoid()
        self.layer_out = nn.Linear(h2_size, output_size)
        self.softmax = nn.Softmax(dim=0)

    def flatten_mnist(self, x):
        result = self.flatten(x)[0]
        return result

    def forward(self, x):
        x = self.flatten_mnist(x)
        z1 = self.layer1(x)
        a1 = self.activation_fn1(z1)
        z2 = self.layer2(a1)
        a2 = self.activation_fn2(z2)
        z_out = self.layer_out(a2)
        y = self.softmax(z_out)
        return y

model_mnist = MLP_MNIST(784, 100, 100, 10)
print(model_mnist)
```

```python
# Load the MNIST handwritten number dataset by using PyTorch
import torch
import torchvision
from torchvision.datasets import MNIST
from torch.utils.data import TensorDataset, DataLoader

batch_size = 1
# Read the MNIST dataset by using torch and torchvision
# This will download the file to PATH if there's no dataset in the PATH
train_loader = DataLoader(
    MNIST('dataset', train=True, download=True,
          transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor()])),
    batch_size=batch_size,
    shuffle=True
)

test_loader = DataLoader(
    MNIST('dataset', train=False, download=True,
          transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor()])),
    batch_size=batch_size,
    shuffle=True
)
```

```python
# Our label is integer (0-9). So, we need a function to convert it into 1x10 vector
# example 0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# example 3 -> [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
def label_conversion_for_outputlayer(y):
    if y == 0:
        ret = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    elif y == 1:
        ret = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    elif y == 2:
        ret = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
    elif y == 3:
        ret = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
    elif y == 4:
        ret = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
    elif y == 5:
        ret = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
    elif y == 6:
        ret = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
    elif y == 7:
        ret = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
    elif y == 8:
        ret = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
    elif y == 9:
        ret = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
    return torch.Tensor(ret)
```

```python
# Training
from torch.optim import SGD

# hyperparameters
learning_rate = 0.01
momentum = 0.01

# Loss function
loss_fn = nn.MSELoss(reduction='mean')
# Stochastic Gradient Descent (SGD) as optimizer in training procedure
#
# v = momentum * v - lr * grad(w)
# w = w + v
optimizer = SGD(model_mnist.parameters(), lr=learning_rate, momentum=momentum)

num_epochs = 20
# Start iterate for model training
loss_arr = []
for epoch in range(num_epochs):
    for batch_idx, train_data in enumerate(train_loader):
        x_batch, y_batch = train_data
        # 0. Reset the gradients to zero
        optimizer.zero_grad()
        # 1. Generate predictions
        predict = model_mnist(x_batch)
        # 2. Calculate loss
        y = label_conversion_for_outputlayer(y_batch)
        loss = loss_fn(predict, y)
        # 3. Compute gradients
        loss.backward()
        # 4. Update parameters using gradients
        optimizer.step()
    print(f'Epoch {epoch} Loss {loss.item():.4f}')
    loss_arr.append(loss.item())
```

```python
# Plot the epoch vs loss
import matplotlib.pyplot as plt

plt.plot(list(range(num_epochs)), loss_arr)
plt.xlabel('iterations')
plt.ylabel('Loss')
plt.show()
```



Step 2.

Test the result model with MNIST test dataset.

```
# Testing resulted model by measuring the accuracy
# accuracy = num of correction / num of total prediction

correct = 0
total = 0
for idx, test_data in enumerate(test_loader):
    x, actual = test_data
    predict = model_mnist(x)
    actual = label_conversion_for_outputlayer(actual)
    val_predict, indices_predict = torch.max(predict, 0)
    val_actual, indices_actual = torch.max(actual, 0)
    if indices_predict == indices_actual:
        correct += 1
    total += 1
print("number of correction=", correct, " out of ", total, ", accuracy=", correct/total)
```

Experiment 2 : Export the MNIST into ONNX model

Step 1.

Save the result PyTorch model on your local folder first.

```
# Save the model in PyTorch .pth file with its current state
import torch
path = 'model_mnist.pth'
torch.save(model_mnist, path)
```

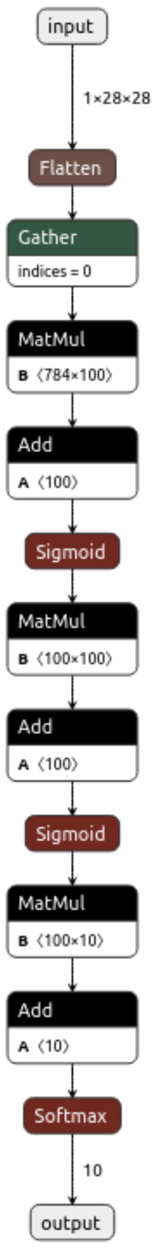
Step 2.

Export the ONNX model from the saved .pth file from step 1.

```
# Export ONNX formatted model from .pth model
import torch

path = 'model_mnist.pth'
loaded_model = torch.load(path, weights_only=False)
loaded_model.eval()

# input_size = 28*28
dummy_input = torch.rand(1, 28, 28)
input_names = ["input"]
output_names = ["output"]
torch.onnx.export(loaded_model,
                  dummy_input,
                  "model_mnist.onnx",
                  verbose=False,
                  input_names=input_names,
                  output_names=output_names,
                  export_params=True,
                  )
```

Step 3.

Visualize the architecture of our exported ONNX model using **Netron.** Try the web-version by opening the following link: https://netron.app/. Upload the .onnx file into it. **Check** if the structure of ANN model is correct according to the PyTorch model or not.



Experiment 3 : Camera interfacing with OpenCV

The **Raspberry Pi 5** will be used in this section for the camera interfacing with OpenCV.

Step 1.

Setup python virtual environment with **venv**, due to the new version of Raspbian OS by run these following commands.

```
$ cd ~

$ mkdir venv

$ cd venv

$ python -m venv ml_onnx

$ source ml_onnx/bin/activate
```

Then, create the **requirements.txt** file as this:

```
numpy
pillow
opencv-python
onnx
onnxscript
onnxruntime
```

and run PIP to install all python modules.

```
$ python -m pip install -r requirements.txt
```

Step 2.

Create a Python program (.py) by using these 2 example codes for interfacing the camera with the camera. One for the basic interface and another one for using the drop-frame technique to fix the buffer issue.

capture_basic.py

```python
import cv2
import time

def process():
    time.sleep(0.2)

#source = "/dev/v4l/by-id/usb-046d_C922_Pro_Stream_Webcam_5B3499FF-video-index0"
source = 0
stream = cv2.VideoCapture(source)

print("start capture")
while stream.isOpened():
    # read frame
    ret, frame = stream.read()

    # some blocking process
    process()

    # output / display
    cv2.imshow("output", frame)

    # check keyboard for exit program
    key = cv2.waitKey(5) & 0xFF
    if key == ord('q'):
        break

stream.release()
print("capture is over")
```

capture_dropframe.py

```
import cv2
import time
import threading
import queue

def process():
    time.sleep(0.2)

source = "/dev/v4l/by-id/usb-046d_C922_Pro_Stream_Webcam_5B3499FF-video-index0"
source = 0
stream = cv2.VideoCapture(source)
q = queue.Queue()
lock = threading.Lock()

def task_inputframe():
    while stream.isOpened():
        # read frame
        ret, frame = stream.read()
        if not ret:
            break

        # put frame to queue
        q.put(frame)
    stream.release()


def task_outputframe():
    while stream.isOpened():
        # drop old frames and read the recent frame
        frame_cnt = q.qsize()
        print("num of frame in Q=", frame_cnt)
        if frame_cnt != 0:
            for i in range(frame_cnt):
                frame = q.get()

            # some process
            process()

            # output / display
            cv2.imshow("output", frame)

            #check keyboard for exit program
            key = cv2.waitKey(5) & 0xFF
            if key == ord('q'):
                break
    stream.release()

t1 = threading.Thread(target=task_inputframe)
t2 = threading.Thread(target=task_outputframe)

t1.start()
t2.start()
t1.join()
t2.join()
print("capture is over")
```

Experiment 4 : Deploy ONNX MNIST model on the Raspberry Pi 5

Create a Python program by using this example code (capture_mnist.py) for the MNIST real-time classification on the raspberry pi 5. ONNX runtime (python version) will be used to run our ONNX model from experiment 2.

The ONNX can be used in the python with **onnxruntime** module as shown here:

```
import onnxruntime

# Load the ONNX model
session = onnxruntime.InferenceSession('model_mnist.onnx')
```

```
# Prediction
predict = session.run(None, {'input': data_input})
```

capture_mnist.py

```
import cv2
import time
import threading
import queue
import numpy as np
import onnxruntime

def preprocess(frame_in):
    # Thresholding to remove background
    tmp = cv2.cvtColor(frame_in, cv2.COLOR_BGR2GRAY)
    _, mask = cv2.threshold(tmp, 100, 255, cv2.THRESH_BINARY)
    frame_in = cv2.bitwise_or(frame_in, frame_in, mask=mask)

    # Increase the contrast of input image
    ycrcb_image = cv2.cvtColor(frame_in, cv2.COLOR_BGR2YCrCb)
    y_channel, cr_channel, cb_channel = cv2.split(ycrcb_image)
    y_channel_stretched = cv2.normalize(y_channel, None, 0, 255, cv2.NORM_MINMAX)
    contrast_stretched_ycrcb = cv2.merge([y_channel_stretched, cr_channel, cb_channel])
    frame_in = cv2.cvtColor(contrast_stretched_ycrcb, cv2.COLOR_YCrCb2BGR)

    # Resize to 28x28, Invert and Normalize the pixel values
    frame_in = cv2.bitwise_not(frame_in)
    frame_out = cv2.resize(frame_in, (28, 28), interpolation=cv2.INTER_AREA)
    frame_out = frame_out / 256
    return frame_out

def task_inputframe():
    while stream.isOpened():
        # read frame
        ret, frame = stream.read()
        if not ret:
            break

        # put frame to queue
        q.put(frame)
    stream.release()
```

```
def task_processframe():
    # Load the ONNX model
    session = onnxruntime.InferenceSession('model_mnist.onnx')

    while stream.isOpened():
        # drop old frames and read the recent frame
        frame_cnt = q.qsize()
        if frame_cnt != 0:
            for i in range(frame_cnt):
                frame = q.get()

            # preprocess
            frame_out = preprocess(frame)
            # Add extra dimension to numpy image due to onnx model (input size = 1x28x28)
            # and select only 1 color channel
            data_input = frame_out[:,:,0]
            data_input = data_input.astype(np.float32)
            data_input = np.expand_dims(data_input, axis=0)

            # run the MNIST ONNX model
            predict = session.run(None, {'input': data_input})
            indices_predict = np.argmax(predict)
            print("predicted number=", int(indices_predict))

            # output / display
            font = cv2.FONT_HERSHEY_SIMPLEX
            h, w, _ = frame.shape
            c= (0, 255, 255)
            cv2.putText(frame, str(indices_predict), (0, h - 40), font, 10, c, 4, cv2.LINE_AA)
            cv2.imshow("input", frame)
            cv2.imshow("output", frame_out)

            #check keyboard for exit program
            key = cv2.waitKey(5) & 0xFF
            if key == ord('q'):
                break
    stream.release()
```

```
#source = "/dev/v4l/by-id/usb-046d_C922_Pro_Stream_Webcam_5B3499FF-video-index0"
source = 0
stream = cv2.VideoCapture(source)

q = queue.Queue()
lock = threading.Lock()

# create threads and start
t1 = threading.Thread(target=task_inputframe)
t2 = threading.Thread(target=task_processframe)

t1.start()
t2.start()
t1.join()
t2.join()
print("capture is over")
```