**Code:**

import torch

import torchreid

import torchreid.reid

import torchreid.reid.data.datasets

import torchreid.reid.data.datasets.video

from torchvision import transforms

import pennylane as qml

# ==== PARAMETERS ====

n\_qubits = 6

n\_layers = 4

# Define the quantum device

dev = qml.device("default.qubit", wires=n\_qubits)

# Define the quantum circuit using PennyLane

@qml.qnode(dev)

def qnode(inputs, weights):

    qml.AmplitudeEmbedding(inputs, wires=range(n\_qubits), pad\_with=0.0, normalize=True)

    qml.BasicEntanglerLayers(weights, wires=range(n\_qubits))

    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n\_qubits)]

# Set weight shapes using generic parameters

weight\_shapes = {"weights": (n\_layers, n\_qubits)}

# ==== Hybrid Model Definition ====

class HybridReIDModel(torch.nn.Module):

    def \_\_init\_\_(self, num\_classes):

        super(HybridReIDModel, self).\_\_init\_\_()

        self.backbone = torchreid.models.build\_model(

            name='resnet50',

            num\_classes=num\_classes,

            loss='softmax',

            pretrained=True,

        ).cuda()

        # Classical head projecting into n\_layers \* n\_qubits

        self.backbonefc = torch.nn.Linear(2048, n\_layers \* (2\*\*n\_qubits))

        # Create quantum layers dynamically

        self.qlayers = torch.nn.ModuleList([

            qml.qnn.TorchLayer(qnode, weight\_shapes) for \_ in range(n\_layers)

        ])

        # Final classification layer

        self.fc = torch.nn.Linear(n\_layers \* n\_qubits, num\_classes)

    def forward(self, x):

        features = self.backbone(x)

        features = self.backbonefc(features)

        # Split into chunks for each quantum layer

        chunks = torch.chunk(features, n\_layers, dim=1)

        quantum\_outputs = []

        for i in range(n\_layers):

            q\_out = self.qlayers[i](chunks[i])

            quantum\_outputs.append(q\_out)

        # Concatenate outputs from all quantum layers

        x = torch.cat(quantum\_outputs, dim=1)

        x = self.fc(x)

        return x

# ==== Main Training Code ====

if \_\_name\_\_ == '\_\_main\_\_':

    datamanager = torchreid.data.VideoDataManager(

        root='',

        sources='prid2011',

        height=256,

        width=128,

        batch\_size\_train=32,

        batch\_size\_test=64,

        seq\_len=15,

        sample\_method='evenly',

        transforms=['random\_flip', 'random\_crop', 'resize', 'normalize']

    )

    train\_loader = datamanager.train\_loader

    test\_loader = datamanager.test\_loader

    query\_loader = test\_loader['prid2011']['query']

    gallery\_loader = test\_loader['prid2011']['gallery']

    num\_classes = datamanager.num\_train\_pids

    model = HybridReIDModel(num\_classes).cuda()

    optimizer = torchreid.optim.build\_optimizer(

        model,

        optim='adam',

        lr=0.0003

    )

    scheduler = torchreid.optim.build\_lr\_scheduler(

        optimizer,

        lr\_scheduler='single\_step',

        stepsize=20

    )

    engine = torchreid.engine.VideoSoftmaxEngine(

        datamanager,

        model,

        optimizer,

        scheduler=scheduler,

        pooling\_method='avg',

        use\_gpu=True,

    )

    engine.run(

        max\_epoch=30,

        save\_dir='log/hybrid\_resnet50\_dynamic\_layers16',

        print\_freq=1,

        test\_only=True,

        eval\_freq=1

    )

**Results (1 to 30 epochs):**

**Epoch 5:**

Speed: 23.2713 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 27.9%

CMC curve

Rank-1 : 16.9%

Rank-5 : 38.2%

Rank-10 : 56.2%

Rank-20 : 69.7%

**Epoch 10:**

Speed: 22.8544 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 34.4%

CMC curve

Rank-1 : 21.3%

Rank-5 : 42.7%

Rank-10 : 59.6%

Rank-20 : 75.3%

**Epoch 15:**

Speed: 23.1314 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 41.2%

CMC curve

Rank-1 : 29.2%

Rank-5 : 51.7%

Rank-10 : 65.2%

Rank-20 : 77.5%

**Epoch 20:**

Speed: 23.0115 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 40.1%

CMC curve

Rank-1 : 28.1%

Rank-5 : 58.4%

Rank-10 : 64.0%

Rank-20 : 75.3%

**Epoch 25:**

Speed: 23.0796 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 35.1%

CMC curve

Rank-1 : 22.5%

Rank-5 : 49.4%

Rank-10 : 65.2%

Rank-20 : 74.2%

**Epoch 30:**

Speed: 23.0411 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 39.9%

CMC curve

Rank-1 : 27.0%

Rank-5 : 56.2%

Rank-10 : 66.3%

Rank-20 : 73.0%