**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 =16

# 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: 19.2654 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 51.7%

CMC curve

Rank-1 : 36.0%

Rank-5 : 68.5%

Rank-10 : 83.1%

Rank-20 : 95.5%

**Epoch 10:**

Speed: 16.6151 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 53.3%

CMC curve

Rank-1 : 38.2%

Rank-5 : 71.9%

Rank-10 : 86.5%

Rank-20 : 93.3%

**Epoch 15:**

Speed: 18.0479 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.4%

CMC curve

Rank-1 : 43.8%

Rank-5 : 76.4%

Rank-10 : 85.4%

Rank-20 : 91.0%

**Epoch 20:**

Speed: 17.3462 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 56.3%

CMC curve

Rank-1 : 41.6%

Rank-5 : 74.2%

Rank-10 : 84.3%

Rank-20 : 93.3%

**Epoch 25:**

Speed: 17.3329 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 54.6%

CMC curve

Rank-1 : 36.0%

Rank-5 : 75.3%

Rank-10 : 85.4%

Rank-20 : 93.3%

**Epoch 30:**

Speed: 17.4510 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 52.3%

CMC curve

Rank-1 : 31.5%

Rank-5 : 76.4%

Rank-10 : 83.1%

Rank-20 : 93.3%