CODE(ANGLE EMBEDDING):

import torch

import torchreid

import pennylane as qml

# Quantum setup

n\_qubits = 6

n\_layers = 32

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

@qml.qnode(dev)

def qnode(inputs, weights):

    qml.AngleEmbedding(inputs, wires=range(n\_qubits), rotation='Y')  # Changed from Amplitude to AngleEmbedding

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

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

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

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()

        # Freeze all layers

        for param in self.backbone.parameters():

            param.requires\_grad = False

        for param in self.backbone.layer4.parameters():

            param.requires\_grad = True

        self.features\_dim = 2048  # ResNet-50 final feature dim

        self.part\_dim = self.features\_dim // n\_layers

        # Projector to match quantum input size

        self.projector = torch.nn.Linear(self.part\_dim, n\_qubits)

        self.qlayers = torch.nn.ModuleList([

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

        ])

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

    def forward(self, x):

        features = self.backbone(x)

        quantum\_outputs = []

        for i in range(n\_layers):

            start\_idx = i \* self.part\_dim

            end\_idx = (i + 1) \* self.part\_dim if i < n\_layers - 1 else None

            x\_part = features[:, start\_idx:end\_idx]

            x\_part = self.projector(x\_part)  # Reduce to n\_qubits

            quantum\_outputs.append(self.qlayers[i](x\_part))

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

        x = self.fc(x)

        return x

# Main logic

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

    transform\_pipeline = [

        'random\_flip',

        'random\_rotate',

        'random\_crop',

        'color\_jitter',

        'normalize'

    ]

    datamanager = torchreid.data.VideoDataManager(

        root='',

        sources='prid2011',

        height=256,

        width=128,

        batch\_size\_train=8,

        batch\_size\_test=64,

        seq\_len=6,

        sample\_method='random',

        transforms=transform\_pipeline,

        num\_instances=4,

        workers=8

    )

    datamanager.train\_loader.num\_workers = 0

    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 = torch.optim.Adam(

        model.parameters(),

        lr=0.0001,

        weight\_decay=5e-4

    )

    scheduler = torch.optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=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\_qml\_angleembed\_fixed',

        print\_freq=1,

        test\_only=False,

        eval\_freq=1

    )

RESULTS:

EPOCH 1:

Speed: 3.0749 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 17.8%

CMC curve

Rank-1 : 9.0%

Rank-5 : 23.6%

Rank-10 : 39.3%

Rank-20 : 53.9%

EPOCH 2:

Speed: 2.9035 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 28.8%

CMC curve

Rank-1 : 13.5%

Rank-5 : 44.9%

Rank-10 : 59.6%

Rank-20 : 74.2%

EPOCH 3:

Speed: 1.2243 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 38.2%

CMC curve

Rank-1 : 24.7%

Rank-5 : 50.6%

Rank-10 : 66.3%

Rank-20 : 79.8%

EPOCH 4:

Speed: 1.2284 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 43.9%

CMC curve

Rank-1 : 29.2%

Rank-5 : 58.4%

Rank-10 : 75.3%

Rank-20 : 87.6%

EPOCH 5:

Speed: 1.2410 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 48.7%

CMC curve

Rank-1 : 31.5%

Rank-5 : 69.7%

Rank-10 : 82.0%

Rank-20 : 91.0%

EPOCH 6:

Speed: 1.0905 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 50.7%

CMC curve

Rank-1 : 36.0%

Rank-5 : 68.5%

Rank-10 : 79.8%

Rank-20 : 92.1%

EPOCH 7:

Speed: 1.2709 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 53.2%

CMC curve

Rank-1 : 39.3%

Rank-5 : 67.4%

Rank-10 : 82.0%

Rank-20 : 87.6%

EPOCH 8:

Speed: 1.8066 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 55.9%

CMC curve

Rank-1 : 42.7%

Rank-5 : 74.2%

Rank-10 : 83.1%

Rank-20 : 93.3%

EPOCH 9:

Speed: 2.3915 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 55.7%

CMC curve

Rank-1 : 43.8%

Rank-5 : 69.7%

Rank-10 : 83.1%

Rank-20 : 92.1%

EPOCH 10:

Speed: 1.1854 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 56.0%

CMC curve

Rank-1 : 42.7%

Rank-5 : 74.2%

Rank-10 : 84.3%

Rank-20 : 96.6%

EPOCH 11:

Speed: 1.1018 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 59.0%

CMC curve

Rank-1 : 48.3%

Rank-5 : 69.7%

Rank-10 : 80.9%

Rank-20 : 94.4%

EPOCH 12:

Speed: 1.1583 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.8%

CMC curve

Rank-1 : 46.1%

Rank-5 : 71.9%

Rank-10 : 80.9%

Rank-20 : 92.1%

EPOCH 13:

Speed: 1.1503 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.8%

CMC curve

Rank-1 : 43.8%

Rank-5 : 75.3%

Rank-10 : 84.3%

Rank-20 : 96.6%

EPOCH 14:

Speed: 1.1872 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 57.5%

CMC curve

Rank-1 : 46.1%

Rank-5 : 68.5%

Rank-10 : 82.0%

Rank-20 : 89.9%

EPOCH 15:

Speed: 1.2586 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 60.6%

CMC curve

Rank-1 : 50.6%

Rank-5 : 68.5%

Rank-10 : 83.1%

Rank-20 : 93.3%

EPOCH 16:

Speed: 1.1927 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 59.4%

CMC curve

Rank-1 : 48.3%

Rank-5 : 68.5%

Rank-10 : 84.3%

Rank-20 : 93.3%

EPOCH 17:

Speed: 1.2176 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.5%

CMC curve

Rank-1 : 47.2%

Rank-5 : 73.0%

Rank-10 : 83.1%

Rank-20 : 93.3%

EPOCH 18:

Speed: 1.2289 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 57.0%

CMC curve

Rank-1 : 44.9%

Rank-5 : 71.9%

Rank-10 : 84.3%

Rank-20 : 93.3%

EPOCH 19:

Speed: 1.2431 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.8%

CMC curve

Rank-1 : 48.3%

Rank-5 : 70.8%

Rank-10 : 85.4%

Rank-20 : 88.8%

EPOCH 20:

Speed: 1.5153 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 54.5%

CMC curve

Rank-1 : 42.7%

Rank-5 : 69.7%

Rank-10 : 78.7%

Rank-20 : 91.0%

EPOCH 21:

Speed: 1.2289 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 60.2%

CMC curve

Rank-1 : 50.6%

Rank-5 : 73.0%

Rank-10 : 80.9%

Rank-20 : 89.9%

EPOCH 22:

Speed: 1.3167 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.2%

CMC curve

Rank-1 : 47.2%

Rank-5 : 71.9%

Rank-10 : 79.8%

Rank-20 : 92.1%

EPOCH 23:

Speed: 1.2381 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 59.9%

CMC curve

Rank-1 : 47.2%

Rank-5 : 73.0%

Rank-10 : 83.1%

Rank-20 : 93.3%

EPOCH 24:

Speed: 1.2667 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 60.8%

CMC curve

Rank-1 : 50.6%

Rank-5 : 73.0%

Rank-10 : 84.3%

Rank-20 : 93.3%

EPOCH 25:

Speed: 1.1873 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 57.6%

CMC curve

Rank-1 : 44.9%

Rank-5 : 73.0%

Rank-10 : 79.8%

Rank-20 : 88.8%

EPOCH 26:

Speed: 1.2419 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.4%

CMC curve

Rank-1 : 46.1%

Rank-5 : 75.3%

Rank-10 : 84.3%

Rank-20 : 91.0%

EPOCH 27:

Speed: 1.2424 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 58.9%

CMC curve

Rank-1 : 49.4%

Rank-5 : 68.5%

Rank-10 : 80.9%

Rank-20 : 96.6%

EPOCH 28:

Speed: 1.1994 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 60.7%

CMC curve

Rank-1 : 50.6%

Rank-5 : 73.0%

Rank-10 : 84.3%

Rank-20 : 91.0%

EPOCH 29:

Speed: 1.2380 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 55.7%

CMC curve

Rank-1 : 42.7%

Rank-5 : 71.9%

Rank-10 : 82.0%

Rank-20 : 95.5%

EPOCH 30:

Speed: 1.1817 sec/batch

Computing distance matrix with metric=euclidean ...

Computing CMC and mAP ...

\*\* Results \*\*

mAP: 56.7%

CMC curve

Rank-1 : 46.1%

Rank-5 : 69.7%

Rank-10 : 83.1%

Rank-20 : 88.8%

Elapsed 1:04:58