Question 2 Part A

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torchvision.models import resnet50
from torch.utils.data import DataLoader

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])

transform_test = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])


train_dataset = datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform_train)
test_dataset = datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform_test)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True,
num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False,
num_workers=2)

model = resnet50(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
model.fc = nn.Linear(model.fc.in_features, 10)
model = model.to(device)


criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9,
weight_decay=5e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7,
gamma=0.1)
```

```
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct, total = 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    model.eval()
    test_correct, test_total = 0, 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = outputs.max(1)
            test_total += labels.size(0)
            test_correct += predicted.eq(labels).sum().item()

    print(f'Epoch {epoch+1}/{num_epochs}, Loss:
{running_loss/len(train_loader):.4f}, Test Acc:
{100.*test_correct/test_total:.2f}%')

torch.save(model.state_dict(), 'resnet50_cifar10.pth')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz

100%|██████████| 170M/170M [00:04<00:00, 35.3MB/s]

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified

/usr/local/lib/python3.11/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:2
23: 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=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-
0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-
0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 171MB/s]

Epoch 1/10, Loss: 0.7296, Test Acc: 80.21%
Epoch 2/10, Loss: 0.5779, Test Acc: 79.37%
Epoch 3/10, Loss: 0.5524, Test Acc: 80.59%
Epoch 4/10, Loss: 0.5364, Test Acc: 82.07%
Epoch 5/10, Loss: 0.5216, Test Acc: 82.06%
Epoch 6/10, Loss: 0.5198, Test Acc: 81.92%
Epoch 7/10, Loss: 0.5065, Test Acc: 80.95%
Epoch 8/10, Loss: 0.4966, Test Acc: 82.68%
Epoch 9/10, Loss: 0.4969, Test Acc: 82.80%
Epoch 10/10, Loss: 0.4898, Test Acc: 81.33%
```

Question 2 Part B

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243,
0.261)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243,
0.261)),
])

train_dataset = datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform_train)
test_dataset = datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform_test)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True,
num_workers=2)
```

```
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False,
num_workers=2)

conv1 = nn.Conv2d(3, 32, 3, padding=1).to(device)
conv2 = nn.Conv2d(32, 64, 3, padding=1).to(device)
conv3 = nn.Conv2d(64, 128, 3, padding=1).to(device)
pool = nn.MaxPool2d(2, 2)
fc = nn.Linear(128 * 4 * 4, 10).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(list(conv1.parameters()) +
list(conv2.parameters()) + list(conv3.parameters()) +
list(fc.parameters()), lr=0.001)

num_epochs = 20
for epoch in range(num_epochs):
    running_loss, correct, total = 0.0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        x = pool(nn.functional.relu(conv1(inputs)))
        x = pool(nn.functional.relu(conv2(x)))
        x = pool(nn.functional.relu(conv3(x)))
        x = x.view(-1, 128 * 4 * 4)
        outputs = fc(x)

        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    test_correct, test_total = 0, 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            x = pool(nn.functional.relu(conv1(inputs)))
            x = pool(nn.functional.relu(conv2(x)))
            x = pool(nn.functional.relu(conv3(x)))
            x = x.view(-1, 128 * 4 * 4)
            outputs = fc(x)
            _, predicted = outputs.max(1)
            test_total += labels.size(0)
            test_correct += predicted.eq(labels).sum().item()

    print(f'Epoch {epoch+1}/{num_epochs}, Loss:
```

```
{running_loss/len(train_loader):.4f}, Test Acc:
{100.*test_correct/test_total:.2f}%')
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1/20, Loss: 1.4003, Test Acc: 58.61%
Epoch 2/20, Loss: 1.0146, Test Acc: 67.62%
Epoch 3/20, Loss: 0.8625, Test Acc: 71.04%
Epoch 4/20, Loss: 0.7685, Test Acc: 72.12%
Epoch 5/20, Loss: 0.6982, Test Acc: 74.31%
Epoch 6/20, Loss: 0.6441, Test Acc: 75.15%
Epoch 7/20, Loss: 0.6037, Test Acc: 75.30%
Epoch 8/20, Loss: 0.5623, Test Acc: 76.18%
Epoch 9/20, Loss: 0.5296, Test Acc: 76.80%
Epoch 10/20, Loss: 0.5002, Test Acc: 76.96%
Epoch 11/20, Loss: 0.4768, Test Acc: 77.18%
Epoch 12/20, Loss: 0.4519, Test Acc: 77.39%
Epoch 13/20, Loss: 0.4286, Test Acc: 77.46%
Epoch 14/20, Loss: 0.4036, Test Acc: 77.07%
Epoch 15/20, Loss: 0.3883, Test Acc: 77.82%
Epoch 16/20, Loss: 0.3658, Test Acc: 78.11%
Epoch 17/20, Loss: 0.3496, Test Acc: 77.45%
Epoch 18/20, Loss: 0.3363, Test Acc: 77.57%
Epoch 19/20, Loss: 0.3156, Test Acc: 77.68%
Epoch 20/20, Loss: 0.3106, Test Acc: 77.48%
```

Question2 part C

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torch.nn.functional as F
from torchvision.models import resnet50

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64,
```

```
shuffle=False)

teacher_model = resnet50(pretrained=True)
teacher_model.fc = nn.Linear(teacher_model.fc.in_features, 10)
teacher_model.to(device)
teacher_model.eval()
for param in teacher_model.parameters():
    param.requires_grad = False

student_model = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(128 * 4 * 4, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
student_model.to(device)

criterion = nn.CrossEntropyLoss()
kl_criterion = nn.KLDivLoss(reduction='batchmean')
optimizer = optim.Adam(student_model.parameters(), lr=0.001)

num_epochs = 10
for epoch in range(num_epochs):
    student_model.train()
    total_loss = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()

        with torch.no_grad():
            teacher_logits = teacher_model(images)

        student_logits = student_model(images)

        ce_loss = criterion(student_logits, labels)
        kl_loss = kl_criterion(F.log_softmax(student_logits / 3,
dim=1),
                               F.softmax(teacher_logits / 3, dim=1))
        loss = 0.5 * ce_loss + 0.5 * kl_loss
```

```
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    student_model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = student_model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_acc = 100 * correct / total
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {total_loss / 
len(train_loader):.4f}, Test Accuracy: {test_acc:.2f}%')
```

```
Files already downloaded and verified
Files already downloaded and verified
Epoch 1/10, Loss: 0.7446, Test Accuracy: 63.66%
Epoch 2/10, Loss: 0.5570, Test Accuracy: 70.04%
Epoch 3/10, Loss: 0.4791, Test Accuracy: 73.67%
Epoch 4/10, Loss: 0.4271, Test Accuracy: 75.73%
Epoch 5/10, Loss: 0.3835, Test Accuracy: 76.33%
Epoch 6/10, Loss: 0.3488, Test Accuracy: 77.09%
Epoch 7/10, Loss: 0.3162, Test Accuracy: 77.39%
Epoch 8/10, Loss: 0.2863, Test Accuracy: 76.20%
Epoch 9/10, Loss: 0.2633, Test Accuracy: 77.06%
Epoch 10/10, Loss: 0.2408, Test Accuracy: 76.98%
```

Yes, accuracy increases in (c) compared to (b) because the student model learns from the teacher using Knowledge Distillation. The teacher's soft labels help the student model understand class relationships better, leading to improved learning. This makes the student model more accurate and generalizes well compared to standard training in (b).

Question 1

```
Drive already mounted at /content/drive; to attempt to forcibly 
remount, call drive.mount("/content/drive", force_remount=True).
```

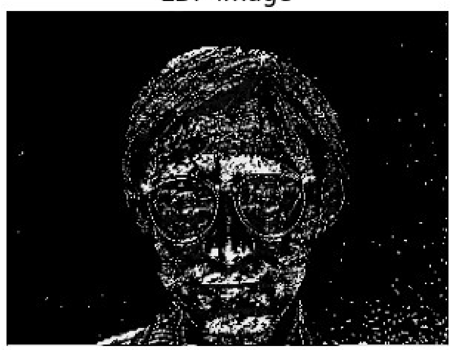
```
Wrong Classified:Actual:subject09,Predicted:subject13,Accuracy:0.0
Wrong Classified:Actual:subject15,Predicted:subject07,Accuracy:0.0
correct Classified:Actual:subject08,Predicted:subject08,Accuracy:0.33
correct Classified:Actual:subject10,Predicted:subject10,Accuracy:0.5
correct Classified:Actual:subject14,Predicted:subject14,Accuracy:0.6
Wrong Classified:Actual:subject08,Predicted:subject15,Accuracy:0.5
correct Classified:Actual:subject13,Predicted:subject13,Accuracy:0.57
Wrong Classified:Actual:subject03,Predicted:subject09,Accuracy:0.5
Wrong Classified:Actual:subject07,Predicted:subject06,Accuracy:0.44
correct Classified:Actual:subject01,Predicted:subject01,Accuracy:0.5
Wrong Classified:Actual:subject02,Predicted:subject13,Accuracy:0.45
Wrong Classified:Actual:subject04,Predicted:subject09,Accuracy:0.42
correct Classified:Actual:subject13,Predicted:subject13,Accuracy:0.46
correct Classified:Actual:subject10,Predicted:subject10,Accuracy:0.5
correct Classified:Actual:subject09,Predicted:subject09,Accuracy:0.53
correct Classified:Actual:subject11,Predicted:subject11,Accuracy:0.56
correct Classified:Actual:subject04,Predicted:subject04,Accuracy:0.59
correct Classified:Actual:subject14,Predicted:subject14,Accuracy:0.61
correct Classified:Actual:subject11,Predicted:subject11,Accuracy:0.63
Wrong Classified:Actual:subject06,Predicted:subject07,Accuracy:0.6
Wrong Classified:Actual:subject02,Predicted:subject08,Accuracy:0.57
correct Classified:Actual:subject15,Predicted:subject15,Accuracy:0.59
correct Classified:Actual:subject03,Predicted:subject03,Accuracy:0.61
correct Classified:Actual:subject12,Predicted:subject12,Accuracy:0.62
correct Classified:Actual:subject05,Predicted:subject05,Accuracy:0.64
correct Classified:Actual:subject06,Predicted:subject06,Accuracy:0.65
correct Classified:Actual:subject12,Predicted:subject12,Accuracy:0.67
correct Classified:Actual:subject07,Predicted:subject07,Accuracy:0.68
correct Classified:Actual:subject05,Predicted:subject05,Accuracy:0.69
Wrong Classified:Actual:subject01,Predicted:subject12,Accuracy:0.67


[0.00000000e+00 3.85802469e-05 3.47222222e-04 1.09310700e-03
 1.72325103e-03 1.91615226e-03 2.90637860e-03 0.00000000e+00
 4.08950617e-03 4.29526749e-03 3.66512346e-03 4.12808642e-03
 5.00257202e-03 5.10545267e-03 0.00000000e+00 5.32407407e-03
 4.96399177e-03 5.40123457e-03 5.69701646e-03 5.14403292e-03
 4.66820988e-03 0.00000000e+00 4.56532922e-03 4.28240741e-03
 3.74228395e-03 2.91923868e-03 2.88065844e-03 2.08333333e-03
 0.00000000e+00 2.31481481e-03 1.74897119e-03 1.94187243e-03
 1.90329218e-03 1.98045267e-03 2.70061728e-03 0.00000000e+00
 2.41769547e-03 2.81635802e-03 3.12500000e-03 3.11213992e-03
 3.18930041e-03 3.04783951e-03 0.00000000e+00 2.62345679e-03
 2.43055556e-03 2.63631687e-03 2.72633745e-03 2.22479424e-03
 2.31481481e-03 0.00000000e+00 2.01903292e-03 2.17335391e-03
 2.25051440e-03 2.07047325e-03 2.30195473e-03 2.62345679e-03
 0.00000000e+00 2.08333333e-03 2.30195473e-03 2.41769547e-03
```

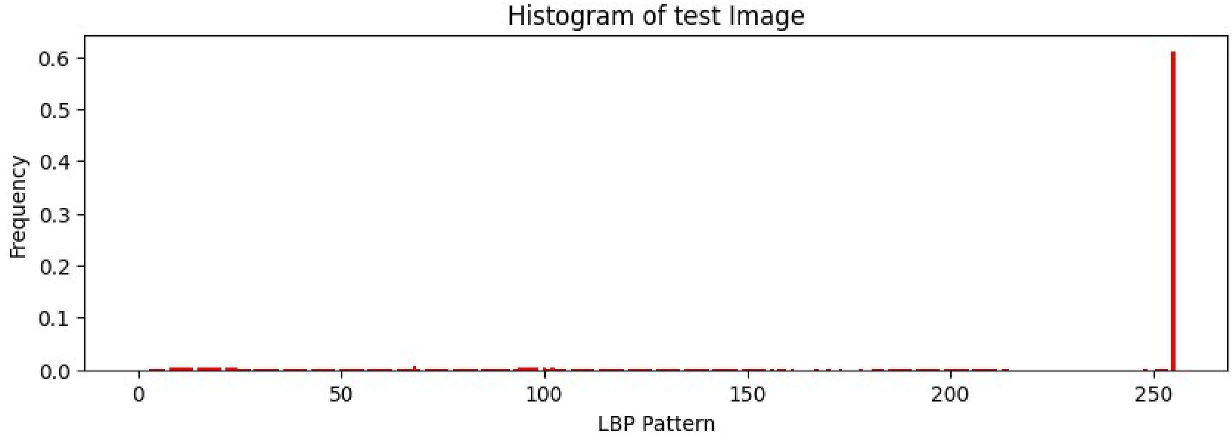
```
 2.62345679e-03 2.58487654e-03 2.22479424e-03 0.00000000e+00
 2.61059671e-03 2.91923868e-03 2.68775720e-03 2.82921811e-03
 6.84156379e-03 2.77777778e-03 0.00000000e+00 3.38220165e-03
 2.88065844e-03 2.95781893e-03 2.95781893e-03 3.11213992e-03
 2.57201646e-03 0.00000000e+00 3.00925926e-03 2.68775720e-03
 2.55915638e-03 3.08641975e-03 2.70061728e-03 2.80349794e-03
 0.00000000e+00 2.71347737e-03 2.90637860e-03 2.79063786e-03
 2.36625514e-03 2.55915638e-03 2.85493827e-03 2.77777778e-03
 0.00000000e+00 3.24074074e-03 3.51080247e-03 4.14094650e-03
 4.02520576e-03 4.14094650e-03 3.61368313e-03 0.00000000e+00
 3.61368313e-03 3.24074074e-03 3.75514403e-03 2.94495885e-03
 2.80349794e-03 3.11213992e-03 0.00000000e+00 2.81635802e-03
 2.95781893e-03 2.84207819e-03 2.44341564e-03 2.31481481e-03
 2.17335391e-03 0.00000000e+00 2.21193416e-03 2.16049383e-03
 1.74897119e-03 1.54320988e-03 1.50462963e-03 1.42746914e-03
 0.00000000e+00 1.51748971e-03 1.38888889e-03 1.19598765e-03
 1.26028807e-03 1.42746914e-03 1.14454733e-03 0.00000000e+00
 1.26028807e-03 1.15740741e-03 2.81635802e-03 1.02880658e-03
 1.01594650e-03 9.25925926e-04 0.00000000e+00 9.38786008e-04
 1.08024691e-03 9.64506173e-04 1.00308642e-03 8.48765432e-04
 1.05452675e-03 0.00000000e+00 8.61625514e-04 1.05452675e-03
 8.48765432e-04 7.71604938e-04 8.48765432e-04 8.48765432e-04
 0.00000000e+00 7.20164609e-04 7.97325103e-04 7.45884774e-04
 8.23045267e-04 6.68724280e-04 7.97325103e-04 0.00000000e+00
 7.20164609e-04 5.14403292e-04 6.55864198e-04 6.94444444e-04
 4.62962963e-04 6.94444444e-04 0.00000000e+00 5.40123457e-04
 5.91563786e-04 6.17283951e-04 5.78703704e-04 6.43004115e-04
 4.62962963e-04 0.00000000e+00 7.07304527e-04 6.17283951e-04
 5.52983539e-04 6.30144033e-04 5.52983539e-04 5.14403292e-04
 5.65843621e-04 0.00000000e+00 6.30144033e-04 5.40123457e-04
 5.65843621e-04 9.25925926e-04 7.71604938e-04 7.45884774e-04
 0.00000000e+00 8.87345679e-04 8.23045267e-04 9.38786008e-04
 1.10596708e-03 9.77366255e-04 1.35030864e-03 0.00000000e+00
 1.32458848e-03 1.44032922e-03 1.42746914e-03 1.19598765e-03
 1.13168724e-03 1.17026749e-03 0.00000000e+00 1.13168724e-03
 1.02880658e-03 9.13065844e-04 9.77366255e-04 8.74485597e-04
 9.13065844e-04 0.00000000e+00 8.48765432e-04 7.71604938e-04
 6.94444444e-04 8.87345679e-04 6.68724280e-04 7.07304527e-04
 0.00000000e+00 8.61625514e-04 6.55864198e-04 5.65843621e-04
 5.52983539e-04 5.27263374e-04 5.40123457e-04 0.00000000e+00
 5.14403292e-04 4.24382716e-04 3.72942387e-04 3.21502058e-04
 2.31481481e-04 1.92901235e-04 0.00000000e+00 2.82921811e-04
 3.47222222e-04 2.70061728e-04 2.82921811e-04 1.92901235e-04
 2.31481481e-04 0.00000000e+00 1.92901235e-04 2.31481481e-04
 2.57201646e-04 3.34362140e-04 4.11522634e-04 2.82921811e-04
 0.00000000e+00 4.88683128e-04 4.88683128e-04 4.11522634e-04
 5.27263374e-04 4.75823045e-04 5.40123457e-04 0.00000000e+00
 6.43004115e-04 5.14403292e-04 6.17283951e-04 7.33024691e-04
 1.02880658e-03 1.04166667e-03 0.00000000e+00 6.10365226e-01]
```

Original Image

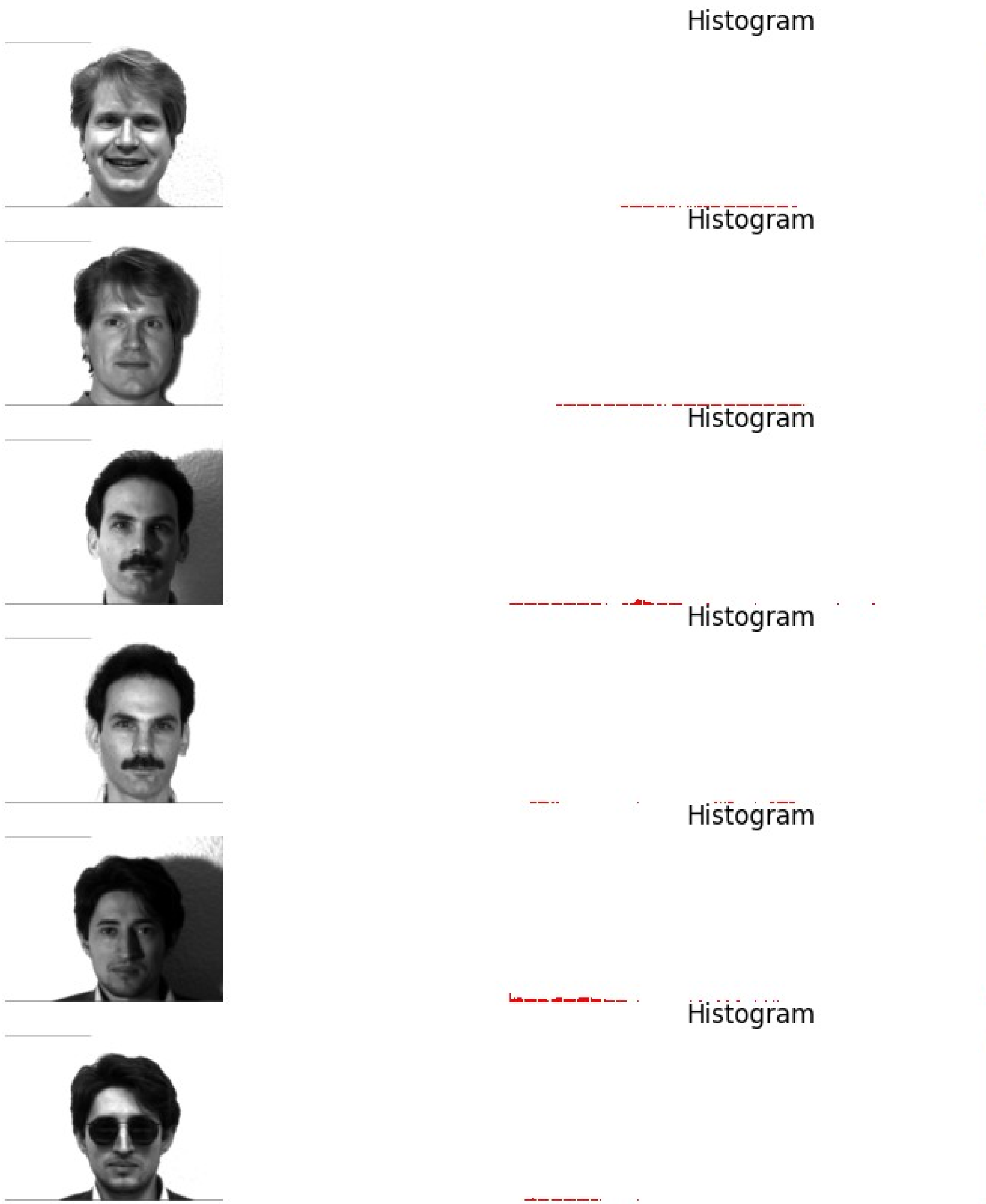
LDP Image



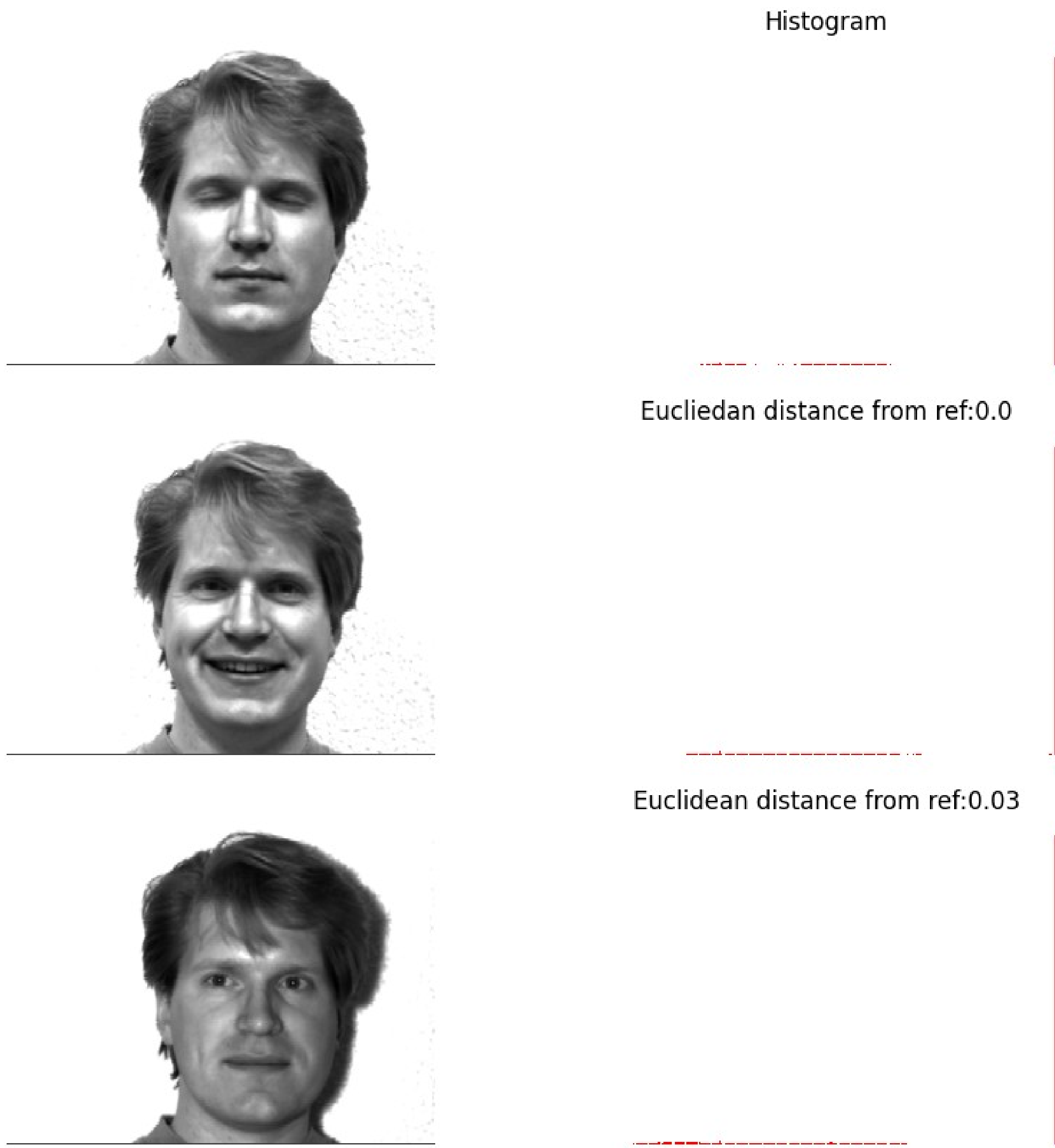
Histogram of test Image




```
Eucliedan distance forsubject01:0.033113800478886374
Eucliedan distance forsubject02:0.14910211356100095
Eucliedan distance forsubject03:0.22608752539817698
```




```
(-13.3, 268.3, 0.0, 0.5707889660493828)
```


Histogram
Eucliedan distance from ref:0.0
Euclidean distance from ref:0.03