```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import zipfile

zip_path = "/content/drive/MyDrive/celeba.zip"  # <- make sure this is the correct path
extract_path = "/content/celeba_data"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

```
!ls "/content/celeba_data/celeba"
```

```
identity_CelebA.txt   list_bbox_celeba.txt             list_landmarks_celeba.txt
img_align_celeba.zip  list_eval_partition.txt
list_attr_celeba.txt  list_landmarks_align_celeba.txt
```

```
import zipfile
img_zip_path = "/content/celeba_data/celeba/img_align_celeba.zip"
img_extract_path = "/content/celeba_data/celeba/img_align_celeba"

with zipfile.ZipFile(img_zip_path, 'r') as zip_ref:
    zip_ref.extractall(img_extract_path)
```

```
!ls "/content/celeba_data/celeba/img_align_celeba/img_align_celeba" | head
```

```
000001.jpg
000002.jpg
000003.jpg
000004.jpg
000005.jpg
000006.jpg
000007.jpg
000008.jpg
000009.jpg
000010.jpg
```

```
from torch.utils.data import Dataset
from PIL import Image
import os

class CelebAMustacheDataset(Dataset):
    def __init__(self, attr_path, img_dir, transform=None):
        self.img_dir = img_dir
        self.transform = transform
        self.img_labels = []

        # Get set of actual image filenames
        available_imgs = set(os.listdir(img_dir))

        with open(attr_path, 'r') as f:
            lines = f.readlines()[2:]  # Skip header lines
            for line in lines:
                parts = line.strip().split()
                img_name = parts[0]
                if img_name not in available_imgs:
                    continue
                mustache_label = int(parts[22])  # 23rd column is Mustache
                label = 1 if mustache_label == 1 else 0
                self.img_labels.append((img_name, label))

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_name, label = self.img_labels[idx]
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, label
```

```
img_dir = "/content/celeba_data/celeba/img_align_celeba/img_align_celeba"
```

```
# Set correct paths
attr_path = "/content/celeba_data/celeba/list_attr_celeba.txt"
img_dir = "/content/celeba_data/celeba/img_align_celeba/img_align_celeba"

# Define transforms
from torchvision import transforms
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

# Create the dataset
dataset = CelebAMustacheDataset(attr_path, img_dir, transform=transform)

# Check length
print(f"Total images in dataset: {len(dataset)}")
```

```
Total images in dataset: 202599
```

```
from torch.utils.data import DataLoader, random_split

# Split into train and test sets
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

print(f"Train size: {len(train_dataset)} | Test size: {len(test_dataset)}")
```

```
Train size: 162079 | Test size: 40520
```

```
import torch
import torch.nn as nn

class MustacheCNN(nn.Module):
    def __init__(self):
        super(MustacheCNN, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 16 * 16, 100),
            nn.ReLU(),
            nn.Linear(100, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.conv(x)
        x = self.fc(x)
        return x
```

```
import torch.optim as optim

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = MustacheCNN().to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(5):  # Start with 5 epochs
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = labels.float().to(device).unsqueeze(1)

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

        running_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {running_loss / len(train_loader):.4f}")
```

```
Epoch 1, Loss: 0.3695
Epoch 2, Loss: 0.2388
Epoch 3, Loss: 0.2213
Epoch 4, Loss: 0.2134
Epoch 5, Loss: 0.2094
```