```
# For downloading the kaagle dataset
import kagglehub

# For unzipping and getting the dataset
import os
import zipfile

# Libraries for handling dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Imports for pytorch
import torch
import torchvision.models as models
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
from tqdm.notebook import tqdm
import torch.optim as optim

import torch.nn as nn
import torch.nn.functional as F
```

```
# Download latest version
path = kagglehub.dataset_download("idealm99/face-shape-new")

train_path = path + "/face-shape-new/training_set"
test_path = path + "/face-shape-new/test_set"

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/idealm99/face-shape-new?dataset_version_number=1...
100%|██████████| 236M/236M [00:11<00:00, 21.2MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/idealm99/face-shape-new/versions/1
```

```
model = models.vgg16(pretrained=True)
# Print the model architecture
print(model)
```

Show hidden output

```
# Replace the classifier of the VGG16 model
model.classifier[6] = nn.Linear(in_features=4096, out_features=5)
# Print the modified model
print(model)
```

Show hidden output

```
# Freeze all layers except the classifier
for param in model.parameters():
    param.requires_grad = False
# Only the classifier layers will be trained
for param in model.classifier.parameters():
    param.requires_grad = True
```

```
from torchvision import transforms

train_transform = transforms.Compose([
    transforms.Resize((150, 150)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(150, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((150, 150)),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])



# Reload the datasets with new transforms
train_dataset = ImageFolder(train_path, transform=train_transform)
test_dataset = ImageFolder(test_path, transform=val_transform)

val_size = 1000
train_data, val_data = random_split(train_dataset, [len(train_dataset) - val_size, val_size])

train_dl = DataLoader(train_data, batch_size=128, shuffle=True, num_workers=2, pin_memory=True)
val_dl = DataLoader(val_data, batch_size=256, num_workers=2, pin_memory=True)


print(f"Length of Train Data : {len(train_data)}")
print(f"Length of Validation Data : {len(val_data)}")
```

```
Length of Train Data : 6953
Length of Validation Data : 1000
```

```
# Reload DataLoaders to make sure they're ready
train_dl = DataLoader(train_data, batch_size=128, shuffle=True, num_workers=2, pin_memory=True)
val_dl = DataLoader(val_data, batch_size=256, num_workers=2, pin_memory=True)


print("CUDA available:", torch.cuda.is_available())
print("Device name:", torch.cuda.get_device_name(0) if torch.cuda.is_available() else "CPU only")
```

```
CUDA available: True
Device name: Tesla T4
```

```
# Set the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Optional: Unfreeze last conv block for fine-tuning
for param in model.features[24:].parameters():
    param.requires_grad = True

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer with weight decay (L2 regularization)
optimizer = optim.SGD(filter(lambda p: p.requires_grad, model.parameters()),
                      lr=0.001, momentum=0.9, weight_decay=1e-4)

# Learning rate scheduler
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

# Early stopping params
patience = 5
best_val_loss = float('inf')
epochs_without_improvement = 0

# Training loop
for epoch in range(100):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_dl:
        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)
```

```
        correct += predicted.eq(labels).sum().item()
        total += labels.size(0)

    train_acc = 100 * correct / total
    avg_loss = running_loss / len(train_dl)
    print(f"[Epoch {epoch+1}] Train Loss: {avg_loss:.4f} | Train Acc: {train_acc:.2f}%")

    # === Validation Evaluation ===
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for inputs, labels in val_dl:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            val_correct += predicted.eq(labels).sum().item()
            val_total += labels.size(0)

    val_acc = 100 * val_correct / val_total
    avg_val_loss = val_loss / len(val_dl)
    print(f"[Epoch {epoch+1}] Val Loss: {avg_val_loss:.4f} | Val Acc: {val_acc:.2f}%")

    # === Early Stopping on Val Loss ===
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        epochs_without_improvement = 0
        print("Validation loss improved. Saving model...")
        torch.save(model.state_dict(), "best_model.pth")
    else:
        epochs_without_improvement += 1
        print(f"No improvement in val loss. ({epochs_without_improvement}/{patience})")

        if epochs_without_improvement >= patience:
            print("Early stopping triggered.")
            break

    # Step the scheduler
    scheduler.step()
```

```
[Epoch 1] Train Loss: 1.1320 | Train Acc: 54.51%
[Epoch 1] Val Loss: 1.0930 | Val Acc: 55.90%
Validation loss improved. Saving model...
[Epoch 2] Train Loss: 1.0671 | Train Acc: 57.90%
[Epoch 2] Val Loss: 1.0506 | Val Acc: 57.70%
Validation loss improved. Saving model...
[Epoch 3] Train Loss: 1.0094 | Train Acc: 60.48%
[Epoch 3] Val Loss: 1.0855 | Val Acc: 55.90%
No improvement in val loss. (1/5)
[Epoch 4] Train Loss: 0.9506 | Train Acc: 63.20%
[Epoch 4] Val Loss: 0.9320 | Val Acc: 64.50%
Validation loss improved. Saving model...
[Epoch 5] Train Loss: 0.9400 | Train Acc: 63.96%
[Epoch 5] Val Loss: 1.0077 | Val Acc: 60.10%
No improvement in val loss. (1/5)
[Epoch 6] Train Loss: 0.8535 | Train Acc: 67.41%
[Epoch 6] Val Loss: 0.8648 | Val Acc: 65.50%
Validation loss improved. Saving model...
[Epoch 7] Train Loss: 0.7744 | Train Acc: 70.34%
[Epoch 7] Val Loss: 0.8503 | Val Acc: 66.10%
Validation loss improved. Saving model...
[Epoch 8] Train Loss: 0.7771 | Train Acc: 70.34%
[Epoch 8] Val Loss: 0.8117 | Val Acc: 69.20%
Validation loss improved. Saving model...
[Epoch 9] Train Loss: 0.7535 | Train Acc: 72.33%
[Epoch 9] Val Loss: 0.8047 | Val Acc: 69.50%
Validation loss improved. Saving model...
[Epoch 10] Train Loss: 0.7446 | Train Acc: 71.65%
[Epoch 10] Val Loss: 0.8382 | Val Acc: 67.20%
No improvement in val loss. (1/5)
[Epoch 11] Train Loss: 0.6831 | Train Acc: 74.56%
[Epoch 11] Val Loss: 0.7901 | Val Acc: 68.60%
Validation loss improved. Saving model...
[Epoch 12] Train Loss: 0.6708 | Train Acc: 74.77%
[Epoch 12] Val Loss: 0.8118 | Val Acc: 68.90%
No improvement in val loss. (1/5)
```

```
[Epoch 13] Train Loss: 0.6509 | Train Acc: 75.84%
[Epoch 13] Val Loss: 0.7436 | Val Acc: 71.70%
Validation loss improved. Saving model...
[Epoch 14] Train Loss: 0.6365 | Train Acc: 76.24%
[Epoch 14] Val Loss: 0.7761 | Val Acc: 70.00%
No improvement in val loss. (1/5)
[Epoch 15] Train Loss: 0.6399 | Train Acc: 75.95%
[Epoch 15] Val Loss: 0.7596 | Val Acc: 71.00%
No improvement in val loss. (2/5)
[Epoch 16] Train Loss: 0.6329 | Train Acc: 76.00%
[Epoch 16] Val Loss: 0.7449 | Val Acc: 72.70%
No improvement in val loss. (3/5)
[Epoch 17] Train Loss: 0.6134 | Train Acc: 77.10%
[Epoch 17] Val Loss: 0.7780 | Val Acc: 69.40%
No improvement in val loss. (4/5)
[Epoch 18] Train Loss: 0.6032 | Train Acc: 78.01%
[Epoch 18] Val Loss: 0.7710 | Val Acc: 71.70%
No improvement in val loss. (5/5)
Early stopping triggered.
```

```python
# === Load best model from previous phase ===
model.load_state_dict(torch.load("best_model.pth"))
model.to(device)

# === Unfreeze more layers (conv4 block) for deeper fine-tuning ===
for param in model.features[17:].parameters():  # conv4_1 and onward
    param.requires_grad = True

# === Use Label Smoothing to improve generalization ===
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)

# === Switch to AdamW optimizer for adaptive fine-tuning ===
optimizer = torch.optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=1e-4, weight_decay=1e-4
)

# === Use ReduceLROnPlateau scheduler for adaptive LR adjustment ===
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.5, verbose=True)

# === Reinitialize early stopping tracking ===
patience = 5
best_val_loss = float('inf')
epochs_without_improvement = 0

# === Fine-Tuning Loop ===
for epoch in range(30):  # Fine-tune for a smaller number of epochs
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_dl:
        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)
        correct += predicted.eq(labels).sum().item()
        total += labels.size(0)

    train_acc = 100 * correct / total
    avg_loss = running_loss / len(train_dl)
    print(f"[FT Epoch {epoch+1}] Train Loss: {avg_loss:.4f} | Train Acc: {train_acc:.2f}%")

    # === Validation Evaluation ===
    model.eval()
    val_loss = 0.0
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for inputs, labels in val_dl:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
```

```
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            val_correct += predicted.eq(labels).sum().item()
            val_total += labels.size(0)

    val_acc = 100 * val_correct / val_total
    avg_val_loss = val_loss / len(val_dl)
    print(f"[FT Epoch {epoch+1}] Val Loss: {avg_val_loss:.4f} | Val Acc: {val_acc:.2f}%")

    # === Early Stopping on Val Loss ===
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        epochs_without_improvement = 0
        print("Validation loss improved. Saving model...")
        torch.save(model.state_dict(), "best_model.pth")
    else:
        epochs_without_improvement += 1
        print(f"No improvement in val loss. ({epochs_without_improvement}/{patience})")

        if epochs_without_improvement >= patience:
            print("Early stopping triggered.")
            break

    # Step the new scheduler based on val loss
    scheduler.step(avg_val_loss)
```

```
[FT Epoch 9] Val Loss: 0.7385 | Val Acc: 82.60%
Validation loss improved. Saving model...
[FT Epoch 10] Train Loss: 0.5534 | Train Acc: 94.38%
[FT Epoch 10] Val Loss: 0.7631 | Val Acc: 82.60%
No improvement in val loss. (1/5)
[FT Epoch 11] Train Loss: 0.5282 | Train Acc: 95.54%
[FT Epoch 11] Val Loss: 0.7426 | Val Acc: 83.20%
No improvement in val loss. (2/5)
[FT Epoch 12] Train Loss: 0.5235 | Train Acc: 95.63%
[FT Epoch 12] Val Loss: 0.6871 | Val Acc: 85.30%
Validation loss improved. Saving model...
[FT Epoch 13] Train Loss: 0.5176 | Train Acc: 95.84%
[FT Epoch 13] Val Loss: 0.6793 | Val Acc: 86.90%
Validation loss improved. Saving model...
[FT Epoch 14] Train Loss: 0.4926 | Train Acc: 97.05%
[FT Epoch 14] Val Loss: 0.6594 | Val Acc: 88.60%
Validation loss improved. Saving model...
[FT Epoch 15] Train Loss: 0.4757 | Train Acc: 97.64%
[FT Epoch 15] Val Loss: 0.6570 | Val Acc: 87.70%
Validation loss improved. Saving model...
[FT Epoch 16] Train Loss: 0.4708 | Train Acc: 97.87%
[FT Epoch 16] Val Loss: 0.6685 | Val Acc: 85.70%
No improvement in val loss. (1/5)
[FT Epoch 17] Train Loss: 0.4711 | Train Acc: 97.77%
[FT Epoch 17] Val Loss: 0.6836 | Val Acc: 86.80%
No improvement in val loss. (2/5)
```

```
[FT Epoch 27] Train Loss: 0.4178 | Train Acc: 99.63%
[FT Epoch 27] Val Loss: 0.6111 | Val Acc: 90.00%
No improvement in val loss. (2/5)
[FT Epoch 28] Train Loss: 0.4185 | Train Acc: 99.48%
[FT Epoch 28] Val Loss: 0.5845 | Val Acc: 91.40%
```

```
from google.colab import files
files.download("best_model.pth")
```

Start coding or generate with AI.