Double-click (or enter) to edit

```
# Cell 1: Install and Import Required Libraries
!pip install timm albumentations opencv-python-headless efficientnet-pytorch
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms
import timm
import cv2
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import os
import albumentations as A
from albumentations.pytorch import ToTensorV2
import warnings
warnings.filterwarnings('ignore')

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Requirement already satisfied: timm in /usr/local/lib/python3.11/dist-packages (1.0.16)
Requirement already satisfied: albumentations in /usr/local/lib/python3.11/dist-packages (2.0.8)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Collecting efficientnet-pytorch
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (from timm) (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (from timm) (0.21.0+cu124)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (from timm) (6.0.2)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.11/dist-packages (from timm) (0.33.1)
Requirement already satisfied: safetensors in /usr/local/lib/python3.11/dist-packages (from timm) (0.5.3)
Requirement already satisfied: numpy>=1.24.4 in /usr/local/lib/python3.11/dist-packages (from albumentations) (2.0.2)
Requirement already satisfied: scipy>=1.10.0 in /usr/local/lib/python3.11/dist-packages (from albumentations) (1.15.3)
Requirement already satisfied: pydantic>=2.9.2 in /usr/local/lib/python3.11/dist-packages (from albumentations) (2.11.7)
Requirement already satisfied: albucore==0.0.24 in /usr/local/lib/python3.11/dist-packages (from albumentations) (0.0.24)
Requirement already satisfied: stringzilla>=3.10.4 in /usr/local/lib/python3.11/dist-packages (from albucore==0.0.24->albur
Requirement already satisfied: simsimd>=5.9.2 in /usr/local/lib/python3.11/dist-packages (from albucore==0.0.24->albumenta
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2->alb
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2->albu
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2->
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.9.2->a
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm) (20
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm) (24
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm) (4.67.1
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch->timm) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch->timm)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch->timm)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch->timm)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch->timm)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch->timm)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch->timm)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch->timm)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch->timm)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch->timm)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch->timm)
```

```
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (2.21
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (12
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch->timm)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch->ti
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvision->timm) (1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch->timm) (3.0.2
```

```
# Cell 2: Mount Google Drive and Setup Paths
from google.colab import drive
drive.mount('/content/drive')

# Update these paths according to your drive structure
TRAIN_PATH = '/content/drive/MyDrive/Comys_Hackathon5/Task_A/train'
VAL_PATH = '/content/drive/MyDrive/Comys_Hackathon5/Task_A/val'
```

```
Mounted at /content/drive
```

```
# Cell 3: Advanced Data Augmentation Pipeline (FIXED)
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms
import timm
import cv2
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import os
import albumentations as A
from albumentations.pytorch import ToTensorV2
import warnings
warnings.filterwarnings('ignore')

class AdvancedAugmentation:
    def __init__(self, image_size=224):
        self.train_transform = A.Compose([
            A.Resize(image_size + 32, image_size + 32),
            A.RandomCrop(image_size, image_size),
            A.HorizontalFlip(p=0.5),
            A.RandomRotate90(p=0.2),
            A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=15, p=0.5),
            A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
            A.HueSaturationValue(hue_shift_limit=20, sat_shift_limit=30, val_shift_limit=20, p=0.5),
            A.GaussNoise(var_limit=(10.0, 50.0), p=0.3),
            A.MotionBlur(blur_limit=3, p=0.3),
            A.CLAHE(clip_limit=2.0, p=0.3),
            # Replaced A.Cutout with additional A.CoarseDropout for similar effect
            A.CoarseDropout(max_holes=8, max_height=16, max_width=16, p=0.3),
            A.CoarseDropout(max_holes=8, max_height=32, max_width=32, p=0.3),
            A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
            ToTensorV2()
        ])

        self.val_transform = A.Compose([
            A.Resize(image_size, image_size),
            A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
            ToTensorV2()
        ])

class FaceDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.images = []
        self.labels = []

        # Load male images (label 0)
        male_dir = os.path.join(root_dir, 'male')
```

```
            if os.path.exists(male_dir):
                for img_name in os.listdir(male_dir):
                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                        self.images.append(os.path.join(male_dir, img_name))
                        self.labels.append(0)

            # Load female images (label 1)
            female_dir = os.path.join(root_dir, 'female')
            if os.path.exists(female_dir):
                for img_name in os.listdir(female_dir):
                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                        self.images.append(os.path.join(female_dir, img_name))
                        self.labels.append(1)

            print(f"Loaded {len(self.images)} images from {root_dir}")
            print(f"Male: {self.labels.count(0)}, Female: {self.labels.count(1)}")

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

        def __getitem__(self, idx):
            img_path = self.images[idx]
            image = cv2.imread(img_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            label = self.labels[idx]

            if self.transform:
                augmented = self.transform(image=image)
                image = augmented['image']

            return image, label

    # Set device
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device: {device}")

    # Initialize augmentation
    aug = AdvancedAugmentation(image_size=224)

    # Create datasets
    train_dataset = FaceDataset(TRAIN_PATH, transform=aug.train_transform)
    val_dataset = FaceDataset(VAL_PATH, transform=aug.val_transform)

    # Create data loaders
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=2)
```

```
Using device: cpu
Loaded 1637 images from /content/drive/MyDrive/Comys_Hackathon5/Task_A/train
Male: 813, Female: 824
Loaded 731 images from /content/drive/MyDrive/Comys_Hackathon5/Task_A/val
Male: 343, Female: 388
```

```
    # Cell 4: SIMPLIFIED High-Performance Model
    class SimpleEffectiveClassifier(nn.Module):
        def __init__(self, num_classes=2):
            super(SimpleEffectiveClassifier, self).__init__()

            # Use single, proven architecture
            self.backbone = timm.create_model('efficientnet_b0', pretrained=True)
            self.backbone.classifier = nn.Sequential(
                nn.Dropout(0.2),
                nn.Linear(1280, 256),
                nn.ReLU(inplace=True),
                nn.Dropout(0.1),
                nn.Linear(256, num_classes)
            )

        def forward(self, x):
            return self.backbone(x)



    # Cell 5: Advanced Training Configuration (FIXED)
    import torch
```

```
import torch.nn as nn
import torch.optim as optim

# Loss function definitions (no model needed yet)
class FocalLoss(nn.Module):
    def __init__(self, alpha=1, gamma=2, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        ce_loss = nn.CrossEntropyLoss(reduction='none')(inputs, targets)
        pt = torch.exp(-ce_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma * ce_loss

        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        else:
            return focal_loss

class LabelSmoothingLoss(nn.Module):
    def __init__(self, classes, smoothing=0.1, dim=1):
        super(LabelSmoothingLoss, self).__init__()
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.cls = classes
        self.dim = dim

    def forward(self, pred, target):
        pred = pred.log_softmax(dim=self.dim)
        with torch.no_grad():
            true_dist = torch.zeros_like(pred)
            true_dist.fill_(self.smoothing / (self.cls - 1))
            true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        return torch.mean(torch.sum(-true_dist * pred, dim=self.dim))

class CombinedLoss(nn.Module):
    def __init__(self):
        super(CombinedLoss, self).__init__()
        self.focal_loss = FocalLoss(alpha=1, gamma=2)
        self.label_smooth_loss = LabelSmoothingLoss(classes=2, smoothing=0.1)

    def forward(self, outputs, targets):
        focal = self.focal_loss(outputs, targets)
        smooth = self.label_smooth_loss(outputs, targets)
        return 0.7 * focal + 0.3 * smooth

# Just define the loss function - optimizer will be created after model
criterion = CombinedLoss()
print("✅ Loss functions defined. Optimizer will be created after model definition.")
```

```
✅ Loss functions defined. Optimizer will be created after model definition.
```

```
2# Cell 6: BALANCED Training with Class Weight Fix + Image Validation
import torch
import torch.nn as nn
import torch.optim as optim
import timm
from torch.utils.data import DataLoader, Dataset
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
import cv2
import os
from PIL import Image
import albumentations as A
from albumentations.pytorch import ToTensorV2

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# VALIDATED DATASET CLASS - Checks images before training
class ValidatedFaceDataset(Dataset):
```

```
        def __init__(self, root_dir, transform=None):
            self.root_dir = root_dir
            self.transform = transform
            self.images = []
            self.labels = []

            # Load and validate images
            self._load_and_validate_images()

        def _load_and_validate_images(self):
            print(f"🔍 Validating images in {self.root_dir}...")
            valid_count = 0
            corrupted_count = 0

            # Load male images (label 0)
            male_dir = os.path.join(self.root_dir, 'male')
            if os.path.exists(male_dir):
                for img_name in os.listdir(male_dir):
                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                        img_path = os.path.join(male_dir, img_name)
                        if self._is_valid_image(img_path):
                            self.images.append(img_path)
                            self.labels.append(0)
                            valid_count += 1
                        else:
                            corrupted_count += 1

            # Load female images (label 1)
            female_dir = os.path.join(self.root_dir, 'female')
            if os.path.exists(female_dir):
                for img_name in os.listdir(female_dir):
                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                        img_path = os.path.join(female_dir, img_name)
                        if self._is_valid_image(img_path):
                            self.images.append(img_path)
                            self.labels.append(1)
                            valid_count += 1
                        else:
                            corrupted_count += 1

            print(f"✅ Loaded {len(self.images)} valid images from {self.root_dir}")
            print(f"   Male: {self.labels.count(0)}, Female: {self.labels.count(1)}")
            if corrupted_count > 0:
                print(f"❌ Skipped {corrupted_count} corrupted images")

        def _is_valid_image(self, img_path):
            """Check if image can be loaded properly"""
            try:
                # Try with PIL first
                with Image.open(img_path) as img:
                    img.verify()

                # Double check with actual loading
                img = Image.open(img_path).convert('RGB')
                if img.size[0] < 32 or img.size[1] < 32:  # Too small
                    return False
                return True
            except Exception:
                try:
                    # Fallback to OpenCV
                    img = cv2.imread(img_path)
                    if img is None or img.shape[0] < 32 or img.shape[1] < 32:
                        return False
                    return True
                except Exception:
                    return False

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

        def __getitem__(self, idx):
            img_path = self.images[idx]
            label = self.labels[idx]

            try:
                # Try loading with PIL first
                image = Image.open(img_path).convert('RGB')
```

```
                image = np.array(image)
            except Exception:
                try:
                    # Fallback to OpenCV
                    image = cv2.imread(img_path)
                    if image is None:
                        raise Exception("Both PIL and OpenCV failed")
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                except Exception:
                    # Last resort: return black image
                    print(f"⚠️ Failed to load {img_path}, using black image")
                    image = np.zeros((224, 224, 3), dtype=np.uint8)

            if self.transform:
                augmented = self.transform(image=image)
                image = augmented['image']

            return image, label

    # CREATE VALIDATED DATASETS
    # Get augmentation from your existing setup
    try:
        # Use existing augmentation if available
        train_dataset = ValidatedFaceDataset(TRAIN_PATH, transform=aug.train_transform)
        val_dataset = ValidatedFaceDataset(VAL_PATH, transform=aug.val_transform)
    except NameError:
        # Create simple augmentation if not available
        simple_transform = A.Compose([
            A.Resize(224, 224),
            A.HorizontalFlip(p=0.5),
            A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
            ToTensorV2()
        ])
        val_transform = A.Compose([
            A.Resize(224, 224),
            A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
            ToTensorV2()
        ])

        # Update these paths to your actual paths
        TRAIN_PATH = '/content/drive/MyDrive/Comys_Hackathon5/Task_A/train'
        VAL_PATH = '/content/drive/MyDrive/Comys_Hackathon5/Task_A/val'

        train_dataset = ValidatedFaceDataset(TRAIN_PATH, transform=simple_transform)
        val_dataset = ValidatedFaceDataset(VAL_PATH, transform=val_transform)

    # Calculate class weights to fix imbalance
    train_labels = train_dataset.labels
    class_weights = compute_class_weight('balanced', classes=np.unique(train_labels), y=train_labels)
    class_weights = torch.FloatTensor(class_weights).to(device)
    print(f"Class weights - Male: {class_weights[0]:.3f}, Female: {class_weights[1]:.3f}")

    # IMPROVED MODEL with Better Architecture
    class ImprovedGenderClassifier(nn.Module):
        def __init__(self, num_classes=2):
            super(ImprovedGenderClassifier, self).__init__()

            # Use EfficientNet-B2 for better performance
            self.backbone = timm.create_model('efficientnet_b2', pretrained=True)
            self.backbone.classifier = nn.Sequential(
                nn.Dropout(0.3),
                nn.Linear(1408, 512),
                nn.BatchNorm1d(512),
                nn.ReLU(inplace=True),
                nn.Dropout(0.2),
                nn.Linear(512, 128),
                nn.BatchNorm1d(128),
                nn.ReLU(inplace=True),
                nn.Dropout(0.1),
                nn.Linear(128, num_classes)
            )

        def forward(self, x):
            return self.backbone(x)

    model = ImprovedGenderClassifier(num_classes=2).to(device)
    print(f"Model parameters: {sum(p.numel() for p in model.parameters()):,}")
```

```
# BALANCED LOSS FUNCTION
criterion = nn.CrossEntropyLoss(weight=class_weights)  # Uses class weights!
optimizer = optim.AdamW(model.parameters(), lr=3e-4, weight_decay=1e-3)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=5, eta_min=1e-6)

# Data loaders - no multiprocessing to avoid errors
train_loader = DataLoader(train_dataset, batch_size=24, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=24, shuffle=False, num_workers=0)

def train_epoch_balanced(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    male_correct = 0
    female_correct = 0
    male_total = 0
    female_total = 0

    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()

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

        # Track by gender
        male_mask = (target == 0)
        female_mask = (target == 1)
        male_total += male_mask.sum().item()
        female_total += female_mask.sum().item()
        male_correct += (predicted[male_mask] == target[male_mask]).sum().item()
        female_correct += (predicted[female_mask] == target[female_mask]).sum().item()

        if batch_idx % 10 == 0:
            male_acc = 100 * male_correct / male_total if male_total > 0 else 0
            female_acc = 100 * female_correct / female_total if female_total > 0 else 0
            print(f'Batch {batch_idx}/{len(train_loader)}, Loss: {loss.item():.4f}, '
                  f'M_Acc: {male_acc:.1f}%, F_Acc: {female_acc:.1f}%')

    return running_loss / len(train_loader), 100. * correct / total

def validate_epoch_detailed(model, val_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    male_correct = 0
    female_correct = 0
    male_total = 0
    female_total = 0

    with torch.no_grad():
        for data, target in val_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output, target)

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

            # Track by gender
            male_mask = (target == 0)
            female_mask = (target == 1)
            male_total += male_mask.sum().item()
```

```
            female_total += female_mask.sum().item()
            male_correct += (predicted[male_mask] == target[male_mask]).sum().item()
            female_correct += (predicted[female_mask] == target[female_mask]).sum().item()

    male_acc = 100 * male_correct / male_total if male_total > 0 else 0
    female_acc = 100 * female_correct / female_total if female_total > 0 else 0

    return (running_loss / len(val_loader), 100. * correct / total,
            male_acc, female_acc)

# BALANCED TRAINING LOOP
num_epochs = 10
best_acc = 0.0
best_balanced_acc = 0.0

print("Starting BALANCED training...")
for epoch in range(num_epochs):
    print(f'\nEpoch {epoch+1}/{num_epochs}')
    print('-' * 60)

    train_loss, train_acc = train_epoch_balanced(model, train_loader, criterion, optimizer, device)
    val_loss, val_acc, male_acc, female_acc = validate_epoch_detailed(model, val_loader, criterion, device)

    scheduler.step()

    # Calculate balanced accuracy (average of male and female accuracy)
    balanced_acc = (male_acc + female_acc) / 2

    print(f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%')
    print(f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')
    print(f'Male Acc: {male_acc:.2f}%, Female Acc: {female_acc:.2f}%')
    print(f'Balanced Acc: {balanced_acc:.2f}%')
    print(f'LR: {optimizer.param_groups[0]["lr"]:.2e}')

    # Save best balanced model
    if balanced_acc > best_balanced_acc:
        best_balanced_acc = balanced_acc
        best_acc = val_acc
        torch.save(model.state_dict(), '/content/drive/MyDrive/best_balanced_gender_model.pth')
        print(f'🎯 NEW BEST BALANCED MODEL! Balanced Acc: {balanced_acc:.2f}%')

    # Success criteria
    if female_acc > 80 and male_acc > 85:
        print(f'🏆 EXCELLENT BALANCE ACHIEVED!')

    if balanced_acc > 88:
        print(f'🚀 HACKATHON-READY PERFORMANCE!')

print(f'\n🏆 Final Results:')
print(f'Best Overall Accuracy: {best_acc:.2f}%')
print(f'Best Balanced Accuracy: {best_balanced_acc:.2f}%')
```

```
Using device: cpu
🔍 Validating images in /content/drive/MyDrive/Comys_Hackathon5/Task_A/train...
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-6-1003965426.py in <cell line: 0>()
    118 try:
    119     # Use existing augmentation if available
--> 120     train_dataset = ValidatedFaceDataset(TRAIN_PATH, transform=aug.train_transform)
    121     val_dataset = ValidatedFaceDataset(VAL_PATH, transform=aug.val_transform)
    122 except NameError:

                                5 frames
/usr/local/lib/python3.11/dist-packages/PIL/JpegImagePlugin.py in load_read(self, read_bytes)
    413         so libjpeg can finish decoding
    414         """
--> 415         s = self.fp.read(read_bytes)
    416
    417         if not s and ImageFile.LOAD_TRUNCATED_IMAGES and not hasattr(self, "_ended"):

KeyboardInterrupt:
```

Start coding or generate with AI.

```
# Cell 7: COMPREHENSIVE Model Evaluation (REPLACE ENTIRE CELL 7)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import torch
import torch.nn as nn

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

# Load the best model - try multiple possible checkpoint names
model_loaded = False
checkpoint_paths = [
    '/content/drive/MyDrive/best_balanced_gender_model.pth',
    '/content/drive/MyDrive/best_gender_model.pth',
    '/content/drive/MyDrive/best_robust_model.pth'
]

for checkpoint_path in checkpoint_paths:
    try:
        checkpoint = torch.load(checkpoint_path, map_location=device)

        # Handle different checkpoint formats
        if isinstance(checkpoint, dict) and 'model_state_dict' in checkpoint:
            model.load_state_dict(checkpoint['model_state_dict'])
            print(f"✅ Loaded model from {checkpoint_path} (with state_dict)")
        else:
            model.load_state_dict(checkpoint)
            print(f"✅ Loaded model from {checkpoint_path} (direct state_dict)")

        model_loaded = True
        break
    except Exception as e:
        print(f"❌ Failed to load {checkpoint_path}: {str(e)}")
        continue

if not model_loaded:
    print("⚠️ Using current model weights (no checkpoint loaded)")

# Comprehensive evaluation function
def comprehensive_evaluation(model, val_loader, device):
    model.eval()
    all_predictions = []
    all_targets = []
    all_probabilities = []

    with torch.no_grad():
        for data, target in val_loader:
            data, target = data.to(device), target.to(device)

            # Get model outputs
            outputs = model(data)
            probabilities = torch.softmax(outputs, dim=1)
            predicted = torch.argmax(outputs, dim=1)

            # Store results
            all_predictions.extend(predicted.cpu().numpy())
            all_targets.extend(target.cpu().numpy())
            all_probabilities.extend(probabilities.cpu().numpy())

    return np.array(all_predictions), np.array(all_targets), np.array(all_probabilities)

# Run comprehensive evaluation
print("🔍 Running comprehensive model evaluation...")
predictions, targets, probabilities = comprehensive_evaluation(model, val_loader, device)

# Calculate overall accuracy
overall_accuracy = (predictions == targets).mean() * 100

# Calculate gender-specific metrics
male_mask = (targets == 0)
female_mask = (targets == 1)

male_predictions = predictions[male_mask]
male_targets = targets[male_mask]
female_predictions = predictions[female_mask]
```

```
    female_targets = targets[female_mask]

    male_accuracy = (male_predictions == male_targets).mean() * 100 if len(male_targets) > 0 else 0
    female_accuracy = (female_predictions == female_targets).mean() * 100 if len(female_targets) > 0 else 0

    # Balanced accuracy
    balanced_accuracy = (male_accuracy + female_accuracy) / 2

    print("\n" + "="*60)
    print("📊 COMPREHENSIVE EVALUATION RESULTS")
    print("="*60)
    print(f"Overall Accuracy: {overall_accuracy:.2f}%")
    print(f"Male Accuracy: {male_accuracy:.2f}%")
    print(f"Female Accuracy: {female_accuracy:.2f}%")
    print(f"Balanced Accuracy: {balanced_accuracy:.2f}%")
    print("-"*60)

    # Detailed classification report
    print("\n📈 DETAILED CLASSIFICATION REPORT:")
    try:
        report = classification_report(targets, predictions,
                                       target_names=['Male', 'Female'],
                                       digits=4, zero_division=0)
        print(report)
    except:
        print("Error generating classification report")

    # Confusion Matrix Analysis
    print("\n🎯 CONFUSION MATRIX ANALYSIS:")
    cm = confusion_matrix(targets, predictions)
    print("Raw Confusion Matrix:")
    print(f"              Predicted")
    print(f"              Male  Female")
    print(f"Actual Male   {cm[0,0]:4d}   {cm[0,1]:4d}")
    print(f"Actual Female {cm[1,0]:4d}   {cm[1,1]:4d}")

    # Calculate detailed metrics
    if len(cm.ravel()) == 4:
        tn, fp, fn, tp = cm.ravel()

        # Precision and Recall
        male_precision = tn / (tn + fn) if (tn + fn) > 0 else 0
        female_precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        male_recall = tn / (tn + fp) if (tn + fp) > 0 else 0
        female_recall = tp / (tp + fn) if (tp + fn) > 0 else 0

        print(f"\n📊 DETAILED PERFORMANCE METRICS:")
        print(f"Male   - Precision: {male_precision:.3f}, Recall: {male_recall:.3f}")
        print(f"Female - Precision: {female_precision:.3f}, Recall: {female_recall:.3f}")
        print(f"\nConfusion Matrix Components:")
        print(f"True Positives (Female correctly identified): {tp}")
        print(f"True Negatives (Male correctly identified): {tn}")
        print(f"False Positives (Male predicted as Female): {fp}")
        print(f"False Negatives (Female predicted as Male): {fn}")

    # Confidence Analysis
    print(f"\n🎯 CONFIDENCE ANALYSIS:")
    male_confidences = probabilities[male_mask, 0]  # Confidence for male predictions
    female_confidences = probabilities[female_mask, 1]  # Confidence for female predictions

    if len(male_confidences) > 0:
        print(f"Male predictions - Mean confidence: {male_confidences.mean():.3f}, Std: {male_confidences.std():.3f}")
    if len(female_confidences) > 0:
        print(f"Female predictions - Mean confidence: {female_confidences.mean():.3f}, Std: {female_confidences.std():.3f}")

    # Visualization
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

    # 1. Confusion Matrix Heatmap
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Male', 'Female'], yticklabels=['Male', 'Female'],
                ax=ax1)
    ax1.set_title(f'Confusion Matrix\nOverall Accuracy: {overall_accuracy:.2f}%')
    ax1.set_xlabel('Predicted')
    ax1.set_ylabel('Actual')

    # 2. Accuracy Comparison
```

```
categories = ['Male', 'Female', 'Overall', 'Balanced']
accuracies = [male_accuracy, female_accuracy, overall_accuracy, balanced_accuracy]
colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightyellow']

bars = ax2.bar(categories, accuracies, color=colors)
ax2.set_title('Accuracy Comparison by Category')
ax2.set_ylabel('Accuracy (%)')
ax2.set_ylim(0, 100)

# Add value labels on bars
for bar, acc in zip(bars, accuracies):
    height = bar.get_height()
    ax2.text(bar.get_x() + bar.get_width()/2., height + 1,
             f'{acc:.1f}%', ha='center', va='bottom')

# 3. Prediction Distribution
pred_counts = np.bincount(predictions, minlength=2)
labels = ['Male', 'Female']
ax3.pie(pred_counts, labels=labels, autopct='%1.1f%%', startangle=90)
ax3.set_title('Distribution of Predictions')

# 4. Confidence Distribution
if len(probabilities) > 0:
    max_probs = np.max(probabilities, axis=1)
    ax4.hist(max_probs, bins=20, alpha=0.7, color='skyblue', edgecolor='black')
    ax4.set_title('Confidence Score Distribution')
    ax4.set_xlabel('Maximum Probability')
    ax4.set_ylabel('Frequency')
    ax4.axvline(max_probs.mean(), color='red', linestyle='--',
                label=f'Mean: {max_probs.mean():.3f}')
    ax4.legend()

plt.tight_layout()
plt.show()

# Model Information
total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"\n🔧 MODEL INFORMATION:")
print(f"Total Parameters: {total_params:,}")
print(f"Trainable Parameters: {trainable_params:,}")
print(f"Model Size: {total_params * 4 / (1024**2):.2f} MB")

# Data Distribution Info
print(f"\n📊 DATASET INFORMATION:")
print(f"Validation Set Size: {len(targets)}")
print(f"Male samples: {male_mask.sum()} ({male_mask.mean()*100:.1f}%)")
print(f"Female samples: {female_mask.sum()} ({female_mask.mean()*100:.1f}%)")

# Final Assessment
print(f"\n🏆 FINAL ASSESSMENT:")
if balanced_accuracy >= 90:
    status = "🚀 EXCELLENT - Hackathon Ready!"
elif balanced_accuracy >= 80:
    status = "✅ GOOD - Competitive Performance"
elif balanced_accuracy >= 70:
    status = "⚠️ FAIR - Needs Improvement"
else:
    status = "❌ POOR - Major Issues Need Fixing"

print(f"Status: {status}")
print(f"Best Metric: Balanced Accuracy = {balanced_accuracy:.2f}%")

# Recommendations
print(f"\n💡 RECOMMENDATIONS:")
if female_accuracy < 70:
    print("- Female accuracy is low - consider data augmentation for female images")
if male_accuracy < 70:
    print("- Male accuracy is low - check for data quality issues")
if abs(male_accuracy - female_accuracy) > 15:
    print("- Large accuracy gap between genders - address class imbalance")
if overall_accuracy > 85 and balanced_accuracy < 80:
    print("- Model is biased toward majority class - use balanced loss functions")

print("\n" + "="*60)
print("EVALUATION COMPLETE")
```

```
print("="*60)
```

```
❌ Failed to load /content/drive/MyDrive/best_balanced_gender_model.pth: [Errno 2] No such file or directory: '/content/driv
❌ Failed to load /content/drive/MyDrive/best_gender_model.pth: [Errno 2] No such file or directory: '/content/drive/MyDrive
❌ Failed to load /content/drive/MyDrive/best_robust_model.pth: [Errno 2] No such file or directory: '/content/drive/MyDrive
⚠️ Using current model weights (no checkpoint loaded)
🔍 Running comprehensive model evaluation...
--------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-4-1495189986.py in <cell line: 0>()
     64 # Run comprehensive evaluation
     65 print("🔍 Running comprehensive model evaluation...")
---> 66 predictions, targets, probabilities = comprehensive_evaluation(model, val_loader, device)
     67
     68 # Calculate overall accuracy

NameError: name 'model' is not defined
```

```
# Cell 8: Test Model on Random Images from Google Drive
import torch
import numpy as np
from PIL import Image
import albumentations as A
from albumentations.pytorch import ToTensorV2
import matplotlib.pyplot as plt
import os
import cv2

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

# Make sure model is in evaluation mode
model.eval()

# Test image transform (same as validation)
test_transform = A.Compose([
    A.Resize(224, 224),
    A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ToTensorV2()
])

def predict_single_image(image_path, show_image=True):
    """
    Predict gender for a single image

    Args:
        image_path (str): Path to the image file
        show_image (bool): Whether to display the image

    Returns:
        dict: Prediction results
    """
    try:
        # Load and validate image
        if not os.path.exists(image_path):
            return {"error": f"Image not found: {image_path}"}

        # Load image with PIL
        image = Image.open(image_path).convert('RGB')
        original_image = np.array(image)

        # Apply preprocessing
        transformed = test_transform(image=original_image)
        input_tensor = transformed['image'].unsqueeze(0).to(device)

        # Make prediction
        with torch.no_grad():
            output = model(input_tensor)
            probabilities = torch.softmax(output, dim=1)
            predicted_class = torch.argmax(output, dim=1).item()
            confidence = probabilities[0][predicted_class].item() * 100

            # Get both probabilities
            male_prob = probabilities[0][0].item() * 100
            female_prob = probabilities[0][1].item() * 100

        # Determine gender
```

```
            gender = "Female" if predicted_class == 1 else "Male"

            # Display image with prediction
            if show_image:
                plt.figure(figsize=(8, 6))
                plt.imshow(original_image)
                plt.title(f'Prediction: {gender}\nConfidence: {confidence:.1f}%\n'
                          f'Male: {male_prob:.1f}% | Female: {female_prob:.1f}%',
                          fontsize=14, fontweight='bold')
                plt.axis('off')
                plt.show()

            # Return results
            result = {
                "image_path": image_path,
                "predicted_gender": gender,
                "confidence": confidence,
                "male_probability": male_prob,
                "female_probability": female_prob,
                "raw_prediction": predicted_class
            }

            return result

        except Exception as e:
            return {"error": f"Error processing image: {str(e)}"}

    def predict_multiple_images(image_paths, show_images=True):
        """
        Predict gender for multiple images

        Args:
            image_paths (list): List of image file paths
            show_images (bool): Whether to display the images

        Returns:
            list: List of prediction results
        """
        results = []

        print(f"🔍 Testing {len(image_paths)} images...")
        print("="*60)

        for i, image_path in enumerate(image_paths):
            print(f"\n📸 Testing Image {i+1}/{len(image_paths)}")
            print(f"File: {os.path.basename(image_path)}")

            result = predict_single_image(image_path, show_image=show_images)

            if "error" in result:
                print(f"❌ {result['error']}")
            else:
                print(f"🎯 Prediction: {result['predicted_gender']}")
                print(f"🎲 Confidence: {result['confidence']:.1f}%")
                print(f"📊 Probabilities - Male: {result['male_probability']:.1f}%, Female: {result['female_probability']:.1f}%")

            results.append(result)
            print("-"*40)

        return results

    def batch_test_folder(folder_path, max_images=10):
        """
        Test all images in a folder

        Args:
            folder_path (str): Path to folder containing images
            max_images (int): Maximum number of images to test

        Returns:
            list: Prediction results
        """
        image_extensions = ('.jpg', '.jpeg', '.png', '.bmp', '.tiff')
        image_files = []

        if os.path.exists(folder_path):
            for file in os.listdir(folder_path):
                if file.lower().endswith(image_extensions):
```

```
                if file.lower().endswith(image_extensions):
                    image_files.append(os.path.join(folder_path, file))

        # Limit number of images
        image_files = image_files[:max_images]

        if image_files:
            print(f"Found {len(image_files)} images in {folder_path}")
            return predict_multiple_images(image_files)
        else:
            print(f"No images found in {folder_path}")
            return []
    else:
        print(f"Folder not found: {folder_path}")
        return []

# --- USAGE EXAMPLES ---
print("🎮 GENDER CLASSIFICATION TESTING INTERFACE")
print("="*60)

# Example 1: Test a single image
print("\n📋 EXAMPLE USAGE:")
print("1. Single Image Test:")
print("   result = predict_single_image('/content/drive/MyDrive/test_image.jpg')")

print("\n2. Multiple Images Test:")
print("   image_list = [")
print("       '/content/drive/MyDrive/photo1.jpg',")
print("       '/content/drive/MyDrive/photo2.jpg',")
print("       '/content/drive/MyDrive/photo3.jpg'")
print("   ]")
print("   results = predict_multiple_images(image_list)")

print("\n3. Test All Images in Folder:")
print("   results = batch_test_folder('/content/drive/MyDrive/test_photos/', max_images=5)")

print("\n" + "="*60)
print("🚀 READY FOR TESTING!")
print("="*60)

# Quick test on validation images (optional)
print("\n🧪 QUICK TEST: Testing on random validation images...")
try:
    # Test 3 random images from validation set
    import random
    random_indices = random.sample(range(len(val_dataset)), min(3, len(val_dataset)))

    for i, idx in enumerate(random_indices):
        img_path = val_dataset.images[idx]
        true_label = "Female" if val_dataset.labels[idx] == 1 else "Male"

        print(f"\n🔬 Validation Test {i+1}:")
        print(f"True Label: {true_label}")
        result = predict_single_image(img_path, show_image=True)

        if "error" not in result:
            correct = "✅ CORRECT" if result['predicted_gender'] == true_label else "❌ WRONG"
            print(f"Result: {correct}")
except:
    print("Could not run validation test")

print(f"\n💡 TIP: Upload your test images to Google Drive and use the functions above!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-5-1899775544.py in <cell line: 0>()
     12
     13 # Make sure model is in evaluation mode
---> 14 model.eval()
     15
     16 # Test image transform (same as validation)

NameError: name 'model' is not defined
```