Training_and_eval

October 17, 2025

[1]:
```
import torch
import sys
import platform

print("="*60)
print("SYSTEM INFORMATION")
print("="*60)
print(f"Python Version: {sys.version}")
print(f"PyTorch Version: {torch.__version__}")
print(f"Platform: {platform.platform()}")
print(f"\nGPU Available: {torch.cuda.is_available()}")

if torch.cuda.is_available():
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
    print(f"GPU Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:
↪.2f} GB")
    print(f"CUDA Version: {torch.version.cuda}")
else:
    print("  WARNING: No GPU detected! Training will be VERY slow on CPU.")
    print("Consider using a machine with GPU or Google Colab.")
```

```
============================================================
SYSTEM INFORMATION
============================================================
Python Version: 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0]
PyTorch Version: 2.2.2+cu121
Platform: Linux-6.16.3-76061603-generic-x86_64-with-glibc2.35

GPU Available: True
GPU Name: NVIDIA GeForce RTX 4060 Laptop GPU
GPU Memory: 8.18 GB
CUDA Version: 12.1
```

[2]:
```
import os
import numpy as np
import pandas as pd
from pathlib import Path
from PIL import Image
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings('ignore')

# PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

# Models
import timm
from facenet_pytorch import InceptionResnetV1

# Metrics
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    roc_auc_score,
    confusion_matrix,
    classification_report,
    roc_curve
)

print("  All libraries imported successfully!")
```

```
 All libraries imported successfully!
```

[3]:
```
# Set your dataset path
DATASET_PATH = '/home/rishabh/Zentej/Data/mixed_training_data'  # Change this␣
 ↪if your path is different

print(f"Checking dataset at: {DATASET_PATH}")

if not os.path.exists(DATASET_PATH):
    print(f"  ERROR: Dataset not found at {DATASET_PATH}")
    print("Please update DATASET_PATH variable with correct path")
else:
    print("  Dataset folder found!")

    # Check structure
    fake_dir = os.path.join(DATASET_PATH, 'fake')
    real_dir = os.path.join(DATASET_PATH, 'real')
```

```
    if os.path.exists(fake_dir):
        fake_count = len([f for f in os.listdir(fake_dir) if f.endswith(('.
↪jpg', '.png', '.jpeg'))])
        print(f"  - Fake images: {fake_count}")
    else:
        print(f"   Warning: 'fake' folder not found")

    if os.path.exists(real_dir):
        real_count = len([f for f in os.listdir(real_dir) if f.endswith(('.
↪jpg', '.png', '.jpeg'))])
        print(f"  - Real images: {real_count}")
    else:
        print(f"   Warning: 'real' folder not found")

    total = fake_count + real_count
    print(f"\n Total images: {total}")
    print(f"   Fake: {fake_count} ({fake_count/total*100:.1f}%)")
    print(f"   Real: {real_count} ({real_count/total*100:.1f}%)")
```

```
Checking dataset at: /home/rishabh/Zentej/Data/mixed_training_data
 Dataset folder found!
  - Fake images: 50000
  - Real images: 50000

 Total images: 100000
   Fake: 50000 (50.0%)
   Real: 50000 (50.0%)
```

[4]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from facenet_pytorch import InceptionResnetV1
import timm

class TwoModelEnsemble(nn.Module):
    def __init__(self, num_classes=2, dropout=0.3):
        super().__init__()

        print("Loading models...")

        # EfficientNet via timm (direct)
        self.efficientnet = timm.create_model('efficientnet_b0',␣
↪pretrained=True, num_classes=0)
        print("  EfficientNet-B0 loaded")

        # FaceNet via direct import (not torch.hub)
        print("Downloading FaceNet weights...")
```

```python
        self.facenet = InceptionResnetV1(pretrained='vggface2')
        print("  FaceNet loaded")

        # Fusion layers
        self.fusion = nn.Sequential(
            nn.Linear(1792, 1024),  # 1280 + 512
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(dropout)
        )

        self.classifier = nn.Linear(512, num_classes)

        self.confidence_head = nn.Sequential(
            nn.Linear(512, 128),
            nn.ReLU(),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

        print("  Dual-model ensemble ready!")

    def forward(self, x, return_attention=False):
        # Extract features
        eff_feat = self.efficientnet(x)
        face_feat = self.facenet(x)

        # Concatenate
        combined = torch.cat([eff_feat, face_feat], dim=1)

        # Fusion
        fused = self.fusion(combined)

        # Classification
        logits = self.classifier(fused)
        confidence = self.confidence_head(fused)

        if return_attention:
            return logits, confidence, fused

        return logits, confidence

print("  Two-Model Ensemble defined!")
```

```
Two-Model Ensemble defined!
```

[5]:

```python
class DeepfakeDataset(Dataset):
    """Custom dataset for fake/real folders"""
    def __init__(self, root_dir, transform=None, split='train', train_ratio=0.
↪8, seed=42):
        self.root_dir = Path(root_dir)
        self.transform = transform

        self.images = []
        self.labels = []

        # Load real images (label=0)
        real_dir = self.root_dir / 'real'
        if real_dir.exists():
            real_images = list(real_dir.glob('*.jpg')) + list(real_dir.glob('*.
↪png')) + list(real_dir.glob('*.jpeg'))
            self.images.extend(real_images)
            self.labels.extend([0] * len(real_images))

        # Load fake images (label=1)
        fake_dir = self.root_dir / 'fake'
        if fake_dir.exists():
            fake_images = list(fake_dir.glob('*.jpg')) + list(fake_dir.glob('*.
↪png')) + list(fake_dir.glob('*.jpeg'))
            self.images.extend(fake_images)
            self.labels.extend([1] * len(fake_images))

        # Split dataset
        indices = np.arange(len(self.images))
        np.random.seed(seed)
        np.random.shuffle(indices)

        split_idx = int(len(indices) * train_ratio)

        if split == 'train':
            indices = indices[:split_idx]
        else:
            indices = indices[split_idx:]

        self.images = [self.images[i] for i in indices]
        self.labels = [self.labels[i] for i in indices]

        print(f"\n{split.upper()} Dataset:")
        print(f"  Total: {len(self.images)}")
        print(f"  Real: {self.labels.count(0)}")
        print(f"  Fake: {self.labels.count(1)}")

    def __len__(self):
```

```
        return len(self.images)

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

        try:
            image = Image.open(img_path).convert('RGB')

            if self.transform:
                image = self.transform(image)

            return image, label
        except Exception as e:
            print(f"Error loading {img_path}: {e}")
            # Return a black image if error
            if self.transform:
                return torch.zeros(3, 224, 224), label
            return Image.new('RGB', (224, 224)), label

print("  Dataset class defined!")
```

```
 Dataset class defined!
```

[6]:
```
def get_train_transforms():
    """Heavy augmentation for better generalization"""
    return transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.RandomCrop(224),
        transforms.RandomHorizontalFlip(p=0.5),

        # Color augmentation (helps with different lighting)
        transforms.ColorJitter(
            brightness=0.4,
            contrast=0.4,
            saturation=0.4,
            hue=0.2
        ),

        # Geometric augmentation
        transforms.RandomRotation(20),
        transforms.RandomAffine(
            degrees=0,
            translate=(0.1, 0.1),
            scale=(0.9, 1.1)
        ),

```

```
        # Quality augmentation (simulates compression)
        transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
        transforms.RandomGrayscale(p=0.2),

        # Standard normalization
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),

        # Random erasing (helps prevent overfitting)
        transforms.RandomErasing(p=0.4, scale=(0.02, 0.20))
    ])

def get_val_transforms():
    """Validation transforms"""
    return transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

print("  Transforms defined!")
```

```
 Transforms defined!
```

[7]:
```
# Create sample dataset
sample_dataset = DeepfakeDataset(
    DATASET_PATH,
    transform=get_val_transforms(),
    split='train'
)

# Visualize
def show_samples(dataset, num_samples=8):
    fig, axes = plt.subplots(2, 4, figsize=(16, 8))
    axes = axes.ravel()

    indices = np.random.choice(len(dataset), num_samples, replace=False)

    for i, idx in enumerate(indices):
        img, label = dataset[idx]

        # Denormalize
        img_show = img.permute(1, 2, 0).numpy()
        img_show = img_show * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]
        img_show = np.clip(img_show, 0, 1)

        axes[i].imshow(img_show)
```

```
        axes[i].set_title(f"{'FAKE' if label == 1 else 'REAL'}",
                          color='red' if label == 1 else 'green',
                          fontsize=14, fontweight='bold')
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()

print("Showing sample images from dataset:")
show_samples(sample_dataset, num_samples=8)
```

```
TRAIN Dataset:
  Total: 80000
  Real: 39931
  Fake: 40069
Showing sample images from dataset:
```

[8]:
```
# Configuration
CONFIG = {
    'data_dir': DATASET_PATH,
    'batch_size': 8,
    'num_epochs': 25,              # ← CHANGED (was 50)
    'learning_rate': 2e-4,         # ← CHANGED (was 1e-4)
    'device': 'cuda' if torch.cuda.is_available() else 'cpu',
    'save_dir': './models_v2',     # ← CHANGED (was './models')
    'num_workers': 2,
    'train_ratio': 0.8,
```

```python
    'seed': 42
}
print("="*60)
print("TRAINING CONFIGURATION")
print("="*60)
for key, value in CONFIG.items():
    print(f"{key:20s}: {value}")
print("="*60)

# Create save directory
os.makedirs(CONFIG['save_dir'], exist_ok=True)
```

```
============================================================
TRAINING CONFIGURATION
============================================================
data_dir            : /home/rishabh/Zentej/Data/mixed_training_data
batch_size          : 8
num_epochs          : 25
learning_rate       : 0.0002
device              : cuda
save_dir            : ./models_v2
num_workers         : 2
train_ratio         : 0.8
seed                : 42
============================================================
```

[9]:

```python
print("\n Building model...")

# Create model
device = torch.device(CONFIG['device'])
model = TwoModelEnsemble(num_classes=2)
model = model.to(device)

# Count parameters
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" Model created!")
print(f"   Total parameters: {total_params:,}")
print(f"   Trainable parameters: {trainable_params:,}")
print(f"   Model size: ~{total_params * 4 / 1e6:.1f} MB")

# Create datasets
print("\n Creating datasets...")

train_dataset = DeepfakeDataset(
    CONFIG['data_dir'],
```

```python
    transform=get_train_transforms(),
    split='train',
    train_ratio=CONFIG['train_ratio'],
    seed=CONFIG['seed']
)

val_dataset = DeepfakeDataset(
    CONFIG['data_dir'],
    transform=get_val_transforms(),
    split='val',
    train_ratio=CONFIG['train_ratio'],
    seed=CONFIG['seed']
)

# Create dataloaders
train_loader = DataLoader(
    train_dataset,
    batch_size=CONFIG['batch_size'],
    shuffle=True,
    num_workers=CONFIG['num_workers'],
    pin_memory=True if CONFIG['device'] == 'cuda' else False
)

val_loader = DataLoader(
    val_dataset,
    batch_size=CONFIG['batch_size'],
    shuffle=False,
    num_workers=CONFIG['num_workers'],
    pin_memory=True if CONFIG['device'] == 'cuda' else False
)

print(f" Data loaders created!")
print(f"   Train batches: {len(train_loader)}")
print(f"   Val batches: {len(val_loader)}")
```

```
 Building model…
Loading models…
 EfficientNet-B0 loaded
Downloading FaceNet weights…
 FaceNet loaded
 Dual-model ensemble ready!
 Model created!
   Total parameters: 34,345,526
   Trainable parameters: 34,345,526
   Model size: ~137.4 MB
```

```
 Creating datasets…

TRAIN Dataset:
  Total: 80000
  Real: 39931
  Fake: 40069

VAL Dataset:
  Total: 20000
  Real: 10069
  Fake: 9931
 Data loaders created!
   Train batches: 10000
   Val batches: 2500
```

[10]:

```
# ============================================================================
# Cell 11: Define Training Functions
# ============================================================================

class FocalLoss(nn.Module):
    """Focal loss for handling imbalance"""
    def __init__(self, alpha=0.25, gamma=2.0):
        super().__init__()
        self.alpha = alpha
        self.gamma = gamma

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

# Loss functions
focal_loss = FocalLoss()
bce_loss = nn.BCELoss()

# Optimizer - UPDATED for TwoModelEnsemble structure
optimizer = torch.optim.AdamW([
    {'params': model.efficientnet.parameters(), 'lr': 1e-5},  # Pre-trained␣
 ↪backbone
    {'params': model.facenet.parameters(), 'lr': 1e-5},       # Pre-trained␣
 ↪backbone
    {'params': model.fusion.parameters(), 'lr': 1e-4},        # New layers
    {'params': model.classifier.parameters(), 'lr': 1e-4},    # New layers
    {'params': model.confidence_head.parameters(), 'lr': 1e-4}  # New layers
], weight_decay=0.01)
```

```python
# Scheduler
scheduler = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(
    optimizer, T_0=10, T_mult=2
)

print("  Training components ready!")
```

```
 Training components ready!
```

[11]:
```python
print("\n" + "="*60)
print("  STARTING TRAINING")
print("="*60 + "\n")

# Training history
history = {
    'train_loss': [],
    'val_loss': [],
    'train_acc': [],
    'val_acc': [],
    'val_f1': [],
    'val_auc': []
}

best_auc = 0.0
best_epoch = 0

# Training loop
for epoch in range(CONFIG['num_epochs']):
    print(f"\nEpoch {epoch+1}/{CONFIG['num_epochs']}")
    print("-" * 60)

    # ========== TRAINING PHASE ==========
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_total = 0

    pbar = tqdm(train_loader, desc='Training', leave=False)
    for images, labels in pbar:
        images, labels = images.to(device), labels.to(device)

        # Forward
        logits, confidence = model(images)

        # Losses
        cls_loss = focal_loss(logits, labels)
        probs = F.softmax(logits, dim=1)
```

```
            correct_probs = probs[range(len(labels)), labels]
            conf_loss = bce_loss(confidence.squeeze(), correct_probs.detach())

            total_loss = cls_loss + 0.1 * conf_loss

            # Backward
            optimizer.zero_grad()
            total_loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()

            # Metrics
            preds = torch.argmax(logits, dim=1)
            train_correct += (preds == labels).sum().item()
            train_total += labels.size(0)
            train_loss += total_loss.item()

            pbar.set_postfix({
                'loss': f'{total_loss.item():.4f}',
                'acc': f'{100*train_correct/train_total:.2f}%'
            })

        avg_train_loss = train_loss / len(train_loader)
        train_acc = train_correct / train_total

        # ========== VALIDATION PHASE ==========
        model.eval()
        val_loss = 0.0
        all_preds = []
        all_labels = []
        all_probs = []

        with torch.no_grad():
            pbar = tqdm(val_loader, desc='Validation', leave=False)
            for images, labels in pbar:
                images, labels = images.to(device), labels.to(device)

                logits, confidence = model(images)
                loss = F.cross_entropy(logits, labels)

                probs = F.softmax(logits, dim=1)
                preds = torch.argmax(probs, dim=1)

                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
                all_probs.extend(probs[:, 1].cpu().numpy())
```

```python
            val_loss += loss.item()

    avg_val_loss = val_loss / len(val_loader)

    # Calculate metrics
    val_acc = accuracy_score(all_labels, all_preds)
    val_f1 = f1_score(all_labels, all_preds)
    val_auc = roc_auc_score(all_labels, all_probs)

    # Update history
    history['train_loss'].append(avg_train_loss)
    history['val_loss'].append(avg_val_loss)
    history['train_acc'].append(train_acc)
    history['val_acc'].append(val_acc)
    history['val_f1'].append(val_f1)
    history['val_auc'].append(val_auc)

    # Print results
    print(f"Train Loss: {avg_train_loss:.4f} | Train Acc: {train_acc*100:.2f}%")
    print(f"Val Loss: {avg_val_loss:.4f} | Val Acc: {val_acc*100:.2f}%")
    print(f"Val F1: {val_f1:.4f} | Val AUC: {val_auc:.4f}")

    # Save best model
    if val_auc > best_auc:
        best_auc = val_auc
        best_epoch = epoch + 1

        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'val_auc': val_auc,
            'val_acc': val_acc,
            'val_f1': val_f1,
            'history': history
        }, os.path.join(CONFIG['save_dir'], 'best_model.pth'))

        print(f"  NEW BEST MODEL! (AUC: {val_auc:.4f})")

    scheduler.step()

print("\n" + "="*60)
print("  TRAINING COMPLETE!")
print("="*60)
print(f"Best Epoch: {best_epoch}")
print(f"Best AUC: {best_auc:.4f}")
```

```
============================================================
  STARTING TRAINING
============================================================


Epoch 1/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.1013 | Train Acc: 67.05%
Val Loss: 0.4981 | Val Acc: 72.57%
Val F1: 0.7264 | Val AUC: 0.8346
  NEW BEST MODEL! (AUC: 0.8346)

Epoch 2/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0917 | Train Acc: 71.69%
Val Loss: 0.4500 | Val Acc: 76.45%
Val F1: 0.7912 | Val AUC: 0.8635
  NEW BEST MODEL! (AUC: 0.8635)

Epoch 3/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0863 | Train Acc: 73.71%
Val Loss: 0.4142 | Val Acc: 76.50%
Val F1: 0.7890 | Val AUC: 0.8623

Epoch 4/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0829 | Train Acc: 74.86%
Val Loss: 0.3974 | Val Acc: 77.79%
Val F1: 0.8034 | Val AUC: 0.8732
  NEW BEST MODEL! (AUC: 0.8732)
```

```
Epoch 5/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0799 | Train Acc: 75.90%
Val Loss: 0.3998 | Val Acc: 76.78%
Val F1: 0.7907 | Val AUC: 0.8664

Epoch 6/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0773 | Train Acc: 76.59%
Val Loss: 0.3784 | Val Acc: 77.78%
Val F1: 0.8011 | Val AUC: 0.8752
 NEW BEST MODEL! (AUC: 0.8752)

Epoch 7/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0748 | Train Acc: 76.95%
Val Loss: 0.3703 | Val Acc: 78.32%
Val F1: 0.8082 | Val AUC: 0.8756
 NEW BEST MODEL! (AUC: 0.8756)

Epoch 8/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0728 | Train Acc: 77.49%
Val Loss: 0.3646 | Val Acc: 78.80%
Val F1: 0.8168 | Val AUC: 0.8766
 NEW BEST MODEL! (AUC: 0.8766)

Epoch 9/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
```

```
Train Loss: 0.0723 | Train Acc: 77.77%
Val Loss: 0.3671 | Val Acc: 78.61%
Val F1: 0.8142 | Val AUC: 0.8758

Epoch 10/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0717 | Train Acc: 77.94%
Val Loss: 0.3647 | Val Acc: 78.78%
Val F1: 0.8164 | Val AUC: 0.8753

Epoch 11/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0785 | Train Acc: 76.42%
Val Loss: 0.3818 | Val Acc: 78.38%
Val F1: 0.8159 | Val AUC: 0.8708

Epoch 12/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0782 | Train Acc: 76.44%
Val Loss: 0.3787 | Val Acc: 77.92%
Val F1: 0.8051 | Val AUC: 0.8743

Epoch 13/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0769 | Train Acc: 76.76%
Val Loss: 0.3694 | Val Acc: 78.86%
Val F1: 0.8207 | Val AUC: 0.8736

Epoch 14/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
```

```
Train Loss: 0.0752 | Train Acc: 77.17%
Val Loss: 0.3685 | Val Acc: 78.27%
Val F1: 0.8079 | Val AUC: 0.8759

Epoch 15/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0741 | Train Acc: 77.49%
Val Loss: 0.3738 | Val Acc: 78.48%
Val F1: 0.8140 | Val AUC: 0.8737

Epoch 16/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0739 | Train Acc: 77.52%
Val Loss: 0.3734 | Val Acc: 78.44%
Val F1: 0.8106 | Val AUC: 0.8744

Epoch 17/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0726 | Train Acc: 77.77%
Val Loss: 0.3616 | Val Acc: 79.03%
Val F1: 0.8221 | Val AUC: 0.8735

Epoch 18/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
Train Loss: 0.0712 | Train Acc: 78.11%
Val Loss: 0.3692 | Val Acc: 78.64%
Val F1: 0.8143 | Val AUC: 0.8755

Epoch 19/25
------------------------------------------------------------
Training:   0%|          | 0/10000 [00:00<?, ?it/s]
Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
```

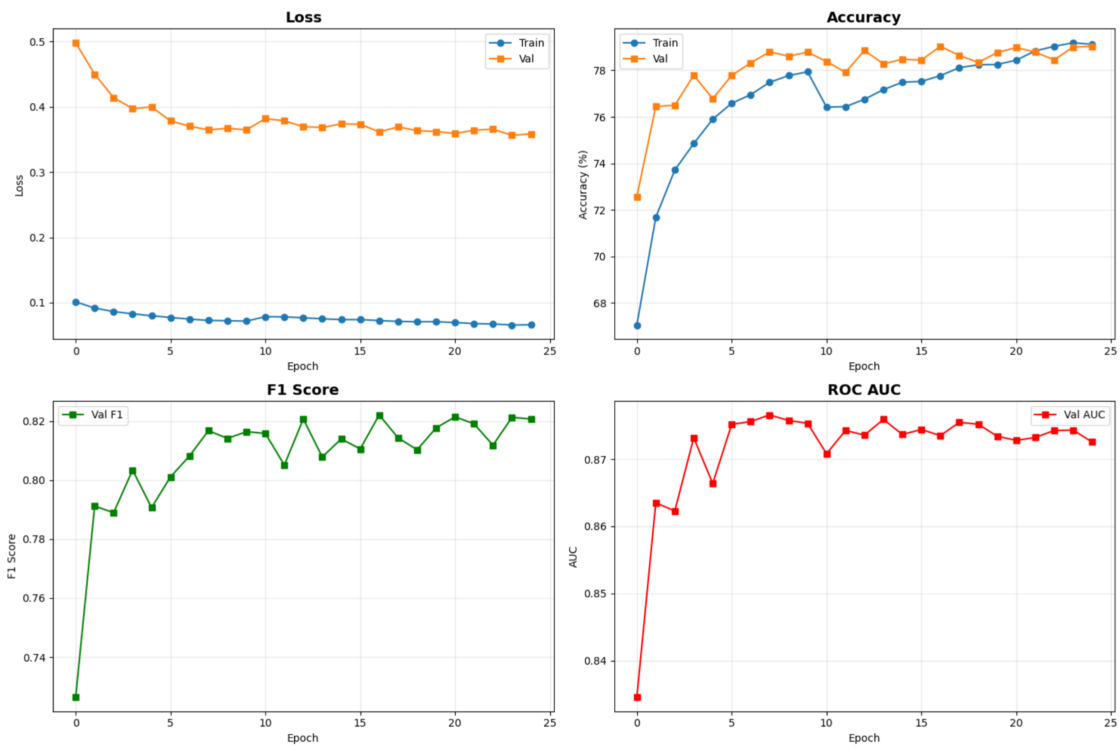
```
Train Loss: 0.0706 | Train Acc: 78.24%
Val Loss: 0.3636 | Val Acc: 78.34%
Val F1: 0.8103 | Val AUC: 0.8752

Epoch 20/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0708 | Train Acc: 78.25%
Val Loss: 0.3621 | Val Acc: 78.77%
Val F1: 0.8177 | Val AUC: 0.8734

Epoch 21/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0695 | Train Acc: 78.44%
Val Loss: 0.3592 | Val Acc: 78.98%
Val F1: 0.8215 | Val AUC: 0.8728

Epoch 22/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0679 | Train Acc: 78.83%
Val Loss: 0.3642 | Val Acc: 78.79%
Val F1: 0.8191 | Val AUC: 0.8732

Epoch 23/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0673 | Train Acc: 79.03%
Val Loss: 0.3658 | Val Acc: 78.45%
Val F1: 0.8119 | Val AUC: 0.8743

Epoch 24/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]
```

```
Train Loss: 0.0657 | Train Acc: 79.18%
Val Loss: 0.3563 | Val Acc: 79.01%
Val F1: 0.8213 | Val AUC: 0.8743

Epoch 25/25
------------------------------------------------------------

Training:   0%|          | 0/10000 [00:00<?, ?it/s]

Validation:   0%|          | 0/2500 [00:00<?, ?it/s]

Train Loss: 0.0661 | Train Acc: 79.12%
Val Loss: 0.3586 | Val Acc: 79.03%
Val F1: 0.8207 | Val AUC: 0.8725

============================================================
 TRAINING COMPLETE!
============================================================
Best Epoch: 8
Best AUC: 0.8766
```

[12]:

```python
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Loss
axes[0, 0].plot(history['train_loss'], label='Train', marker='o')
axes[0, 0].plot(history['val_loss'], label='Val', marker='s')
axes[0, 0].set_title('Loss', fontsize=14, fontweight='bold')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Loss')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Accuracy
axes[0, 1].plot([x*100 for x in history['train_acc']], label='Train',␣
 ↪marker='o')
axes[0, 1].plot([x*100 for x in history['val_acc']], label='Val', marker='s')
axes[0, 1].set_title('Accuracy', fontsize=14, fontweight='bold')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Accuracy (%)')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# F1 Score
axes[1, 0].plot(history['val_f1'], label='Val F1', marker='s', color='green')
axes[1, 0].set_title('F1 Score', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].set_ylabel('F1 Score')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)
```

```python
# AUC
axes[1, 1].plot(history['val_auc'], label='Val AUC', marker='s', color='red')
axes[1, 1].set_title('ROC AUC', fontsize=14, fontweight='bold')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].set_ylabel('AUC')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(os.path.join(CONFIG['save_dir'], 'training_history.png'), dpi=150)
plt.show()

print("  Training history plotted!")
```



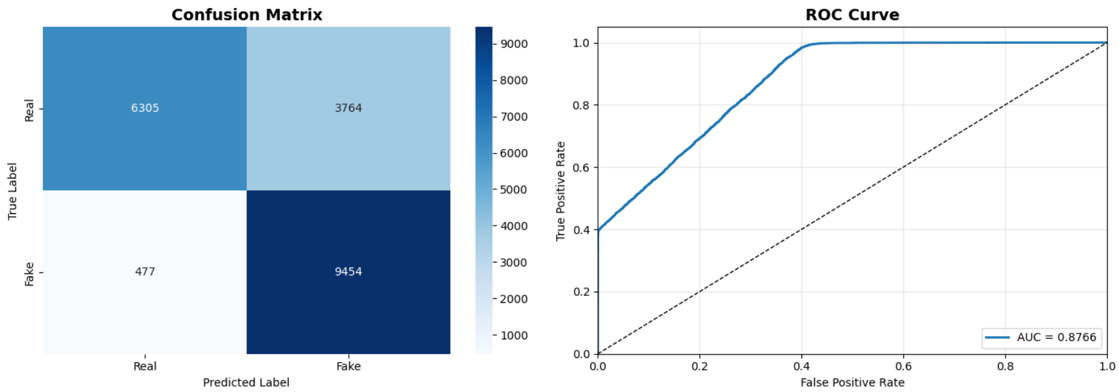

```
Training history plotted!
```

[13]:
```python
# Load best model
print("Loading best model for evaluation...")
checkpoint = torch.load(os.path.join(CONFIG['save_dir'], 'best_model.pth'))
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()
```

```python
print(f"\nBest Model from Epoch {checkpoint['epoch']+1}:")
print(f"  AUC: {checkpoint['val_auc']:.4f}")
print(f"  Accuracy: {checkpoint['val_acc']*100:.2f}%")
print(f"  F1: {checkpoint['val_f1']:.4f}")

# Detailed evaluation
print("\nRunning final evaluation...")

all_preds = []
all_labels = []
all_probs = []

with torch.no_grad():
    for images, labels in tqdm(val_loader, desc='Evaluating'):
        images = images.to(device)

        logits, _ = model(images)
        probs = F.softmax(logits, dim=1)
        preds = torch.argmax(probs, dim=1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.numpy())
        all_probs.extend(probs[:, 1].cpu().numpy())

# Metrics
final_acc = accuracy_score(all_labels, all_preds)
final_f1 = f1_score(all_labels, all_preds)
final_auc = roc_auc_score(all_labels, all_probs)

print("\n" + "="*60)
print("FINAL METRICS")
print("="*60)
print(f"Accuracy:  {final_acc*100:.2f}%")
print(f"F1 Score:  {final_f1:.4f}")
print(f"ROC AUC:   {final_auc:.4f}")
print("="*60)

# Confusion Matrix
cm = confusion_matrix(all_labels, all_preds)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0],
            xticklabels=['Real', 'Fake'],
            yticklabels=['Real', 'Fake'])
axes[0].set_title('Confusion Matrix', fontsize=14, fontweight='bold')
```

```python
axes[0].set_ylabel('True Label')
axes[0].set_xlabel('Predicted Label')

# ROC Curve
fpr, tpr, _ = roc_curve(all_labels, all_probs)
axes[1].plot(fpr, tpr, linewidth=2, label=f'AUC = {final_auc:.4f}')
axes[1].plot([0, 1], [0, 1], 'k--', linewidth=1)
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.05])
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('ROC Curve', fontsize=14, fontweight='bold')
axes[1].legend(loc="lower right")
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(os.path.join(CONFIG['save_dir'], 'final_evaluation.png'), dpi=150)
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(all_labels, all_preds,
                            target_names=['Real', 'Fake'],
                            digits=4))
```

```
Loading best model for evaluation…

Best Model from Epoch 8:
  AUC: 0.8766
  Accuracy: 78.80%
  F1: 0.8168

Running final evaluation…

Evaluating:   0%|          | 0/2500 [00:00<?, ?it/s]


============================================================
FINAL METRICS
============================================================
Accuracy:  78.80%
F1 Score:  0.8168
ROC AUC:   0.8766
============================================================
```




```
Classification Report:
              precision    recall  f1-score   support

        Real     0.9297    0.6262    0.7483     10069
        Fake     0.7152    0.9520    0.8168      9931

    accuracy                         0.7880     20000
   macro avg     0.8225    0.7891    0.7826     20000
weighted avg     0.8232    0.7880    0.7823     20000
```

[15]:

```
# Save in deployment-ready format
torch.save({
    'model_state_dict': model.state_dict(),
    'model_config': {
        'num_classes': 2,
        'input_size': 224,
        'model_type': 'SuperDeepfakeDetector'
    },
    'metrics': {
        'accuracy': final_acc,
        'f1_score': final_f1,
        'auc': final_auc
    },
    'training_config': CONFIG
}, os.path.join(CONFIG['save_dir'], 'super_deepfake_model.pth'))

print(f"\n Final model saved to: {CONFIG['save_dir']}/super_deepfake_model.
 ↪pth")
print("\n ALL DONE! Your model is ready for deployment!")
```

```
Final model saved to: ./models_v2/super_deepfake_model.pth

ALL DONE! Your model is ready for deployment!
```

[ ]:

```

```