DETECTION OF FAKE AND REAL AADHAAR CARDS USING CLIP VLM

NAME: PRADITA G

Aim: To create a model that can differentiate between real and tampered Aadhaar cards within its dataset which is done by feeding data to the model which helps it classify the features and differentiate accordingly

PROCEDURE:

STEP 1:

INPUT AQUSITION: The data set for this classification purpose was created using a base cards template which is Aadhaar like preserving the original structure the logo the slogan their positioning and the font and other styles.

To this base template the other features like name, photo dob, and gender is added.

By tampering the real dataset generated using PIL a Fake dataset for Aadhaar dataset is created

Examples of a few Tampering's applied:

- Name tampering.
- DOB tampering.
- Gender tampering.
- Photo tampering.
- Logo tampering.
- QR tampering.
- Slogan tampering.
- Xerox/photocopy like tampering
- Noise added.
- Blurring

Progress: Data set creation accomplished.

STEP 2:

CLEANING AND ANALYZING THE DATA:

- The data is made free of noise and made better for training.
- We make sure the data is void of white spaces or unnecessary things that can affect the learning capacity of the machine.
- We plot a few graphs or understand the trend of the data before we start training the model to yield better results.

Progress: Data analysis done, and cleaning done data prepared for training

STEP 3:

SPLITTING THE DATASET:

- Although this step sounds insignificant splitting the data into parts for training and validation helps for the model to learn better and helps model understanding and evaluate if the model Is an overfitting one or underfitting one from the results of testing with the validation set
- The best way is to split is 80-20 that is 80 percent for training and 20 percent for testing.
- Here I have placed the real photos under a real folder and fake under a fake folder the training and validation have their own pair of real and fake folder with respective images.

Splitting the data set done

STEP-4:

TRAINING THE MODEL:

- The last final and the most crucial step
- We will be using the CLIP a pretrained Vision language model belonging to OPENAI.
- The CLIP pre processer will load the image and process them correctly.
- So, we freeze the layers belonging to CLIP so that during training nothing many changes.
- Only my custom classifier head will be trained every time.
- We have also used focal loss in this case.
- We have used AdamW as the optimizer which is lightly better than Adam.
- We have 15 epochs on whole.
- In each batch we will get CLIP image embeddings
- Pass those embeddings through a custom classifier.
- Compute the focal loss.
- ‘Backpropagate and update the classifier weights.
- Tracks and prints the training loss and accuracy.
- Then we have a validation loop which will switch to evaluation mode and computes the accuracy on validation no weight updates.
- Stores the predicts and labels for later predictions.
- Then at last we will plot the confusion matrix to understand the scenario on what basis the loss and accuracy is being told

Progress: Done!! Training of the model is done and validation of the model's accuracy and confusion as metric is used to check the actual accuracy of the model.

Picture of the Aadhaar template:

CODE SNIPPET FOR REAL CARD GENERATION:

real_adhaar_samples.py > ...

```python
from PIL import Image, ImageDraw, ImageFont
import random
import os

# Paths
template_path = r"D:\empty1.png"
output_folder = "generated_aadhaar_cards"


# Photo folders
male_photo_folder = r"males"
female_photo_folder = r"females"

# Font file paths
font_hindi_path = "Noto_Sans_Devanagari/static/NotoSansDevanagari-Regular.ttf"
font_english_path = "arial.ttf"
font_dob_path = "arial.ttf"
font_aadhaar_path = "courbd.ttf"


base_font_size = 22

os.makedirs(output_folder, exist_ok=True)


name_mapping = [
    ("Amit Sharma", "अमित शर्मा", "Male"),
    ("Ravi Patel", "रवि पटेल", "Male"),
    ("Sita Verma", "सीता वर्मा", "Female"),
    ("Rahul Gupta", "राहुल गुप्ता", "Male"),
    ("Pooja Reddy", "पूजा रेड्डी", "Female"),
    ("Vijay Mishra", "विजय मिश्रा", "Male"),
    ("Neha Yadav", "नेहा यादव", "Female"),
    ("Anjali Singh", "अंजलि सिंह", "Female"),
    ("Deepak Kumar", "दीपक कुमार", "Male"),
    ("Kavita Joshi", "कविता जोशी", "Female")
]

def random_aadhaar_number():
    """Generate random Aadhaar number in XXXX XXXX XXXX format."""
    return " ".join(["".join([str(random.randint(0, 9)) for _ in range(4)]) for _ in range(3)])

def random_dob():
    """Generate random date of birth in DD/MM/YYYY format."""
    return f"{random.randint(1, 28):02d}/{random.randint(1, 12):02d}/{random.randint(1965, 2000)}"

def get_random_photo(gender):
    """Get a random photo from male or female folder and resize."""
    folder = male_photo_folder if gender == "Male" else female_photo_folder
    photo_list = os.listdir(folder)
    random_photo = random.choice(photo_list)
    photo = Image.open(os.path.join(folder, random_photo)).convert("RGB")
    photo = photo.resize((192, 210), Image.LANCZOS)  # Resize to Aadhaar photo dimensions
    return photo
```

```python
for i in range(1, 101):

    card = Image.open(template_path).copy()
    draw = ImageDraw.Draw(card)

    w, h = card.size
    scale = w / 600


    font_hindi_scaled = ImageFont.truetype(font_hindi_path, int(base_font_size * scale))
    font_english_scaled = ImageFont.truetype(font_english_path, int(base_font_size * scale))
    font_dob_scaled = ImageFont.truetype(font_dob_path, int((base_font_size - 3) * scale))
    font_aadhaar_scaled = ImageFont.truetype(font_aadhaar_path, int((base_font_size + 6) * scale))  # Aadhaar number bigger

    name_english, name_hindi, gender = random.choice(name_mapping)
    aadhaar_number = random_aadhaar_number()
    dob = random_dob()


    user_photo = get_random_photo(gender)
    photo_left = int(w * 0.055)
    photo_top = int(h * 0.27)
    photo_width = 192
    photo_height = 210
    card.paste(user_photo, (photo_left, photo_top))

    x_right_of_photo = photo_left + photo_width + 30
    y_start = photo_top + 5
    line_spacing = int(h * 0.085)


    qr_left = int(w * 0.8)


    aadhaar_text_bbox = draw.textbbox((0, 0), aadhaar_number, font=font_aadhaar_scaled)
    aadhaar_text_width = aadhaar_text_bbox[2] - aadhaar_text_bbox[0]
    aadhaar_text_height = aadhaar_text_bbox[3] - aadhaar_text_bbox[1]


    available_space_left = photo_left + photo_width + 10
    available_space_right = qr_left - 10
    available_space_width = available_space_right - available_space_left

    aadhaar_x = available_space_left + (available_space_width - aadhaar_text_width) // 2 - 5
    aadhaar_y = int(h * 0.75)

    draw.text((x_right_of_photo, y_start), name_hindi, font=font_hindi_scaled, fill=(0, 0, 0))
    draw.text((x_right_of_photo, y_start + line_spacing), name_english, font=font_english_scaled, fill=(0, 0, 0))
    draw.text((x_right_of_photo, y_start + 2 * line_spacing), f"DOB: {dob}", font=font_dob_scaled, fill=(0, 0, 0))
    draw.text((x_right_of_photo, y_start + 3 * line_spacing), f"Gender: {gender}", font=font_english_scaled, fill=(0, 0, 0))
    draw.text((aadhaar_x, aadhaar_y), aadhaar_number, font=font_aadhaar_scaled, fill=(30, 30, 30))


    output_path = os.path.join(output_folder, f"aadhaar_card_{i}.png")
    card.save(output_path)
    print(f" Saved {output_path}")
```

EXAMPLE OF REAL CARDS:

EXPLANATION:

- This Python script generates **100 fake Aadhaar card images** using a given Aadhaar card template.
- It randomly selects names (in English and Hindi) and gender from a predefined list, generates a random Aadhaar number in the XXXX XXXX XXXX format, and creates a random date of birth (DD/MM/YYYY).
- For each card, it picks a photo from separate male and female photo folders, resizes it to fit the Aadhaar layout, and pastes it onto the template.
- It then writes the name, date of birth, gender, and Aadhaar number onto the card using appropriate fonts (for Hindi and English text)
- This script is useful for creating sample datasets or testing Aadhaar-related software.

CODE SNIPPET FOR FAKE CARDS CREATION BY TAMPERING OF REAL DATASET:

```
fake_adhaar_dataset.py > ...
from PIL import Image, ImageDraw, ImageFont, ImageFilter, ImageOps
import random
import os


template_path = r"D:\empty1.png"
output_folder = "tampered_aadhaar_cards"

male_photo_folder = r"males"
female_photo_folder = r"females"


font_hindi_path = "Noto_Sans_Devanagari/static/NotoSansDevanagari-Regular.ttf"
font_english_path = "arial.ttf"
font_dob_path = "arial.ttf"
font_aadhaar_path = "courbd.ttf"


base_font_size = 22

os.makedirs(output_folder, exist_ok=True)

name_mapping = [
    ("Amit Sharma", "अमित शर्मा", "Male"),
    ("Ravi Patel", "रवि पटेल", "Male"),
    ("Sita Verma", "सीता वर्मा", "Female"),
    ("Rahul Gupta", "राहुल गुप्ता", "Male"),
    ("Pooja Reddy", "पूजा रेड्डी", "Female"),
    ("Vijay Mishra", "विजय मिश्रा", "Male"),
    ("Neha Yadav", "नेहा यादव", "Female"),
    ("Anjali Singh", "अंजलि सिंह", "Female"),
    ("Deepak Kumar", "दीपक कुमार", "Male"),
    ("Kavita Joshi", "कविता जोशी", "Female")
]


fake_name_mapping = [
    ("Arjun Mehta", "अर्जुन मेहता", "Male"),
    ("Sneha Kapoor", "स्नेहा कपूर", "Female"),
    ("Karan Desai", "करण देसाई", "Male"),
    ("Riya Sharma", "रिया शर्मा", "Female"),
    ("Manish Jain", "मनीष जैन", "Male"),
    ("Priya Nair", "प्रिया नायर", "Female"),
    ("Suresh Rao", "सुरेश राव", "Male"),
    ("Divya Menon", "दिव्या मेनन", "Female"),
    ("Nikhil Bansal", "निखिल बंसल", "Male"),
    ("Aarti Chawla", "आरती चावला", "Female")
]


def random_aadhaar_number():
    """Generate random Aadhaar number."""
    return " ".join(["".join([str(random.randint(0, 9)) for _ in range(4)]) for _ in range(3)])

def random_dob():
    """Generate random date of birth."""
    return f"{random.randint(1, 28):02d}/{random.randint(1, 12):02d}/{random.randint(1965, 2000)}"
```

```python
def change_color_band(card):
    """Change color band to random color."""
    draw = ImageDraw.Draw(card)
    band_area = (0, int(card.height * 0.15), card.width, int(card.height * 0.22))
    random_color = tuple(random.choices(range(256), k=3))
    draw.rectangle(band_area, fill=random_color)
    return card

def apply_blur(card):
    """Apply blur effect."""
    return card.filter(ImageFilter.GaussianBlur(radius=2))

def apply_xerox_effect(card):
    """Apply xerox (black & white and blur) effect only to base card, not text."""
    base_card = card.convert("L")  # Convert to grayscale
    base_card = base_card.filter(ImageFilter.GaussianBlur(radius=1))
    return base_card.convert("RGB")  # Convert back to RGB for drawing text later

# Generate tampered Aadhaar cards
for i in range(1, 101):
    # Open template
    card = Image.open(template_path).copy()
    draw = ImageDraw.Draw(card)

    # Adjust font size relative to template size
    w, h = card.size
    scale = w / 600
    font_hindi_scaled = ImageFont.truetype(font_hindi_path, int(base_font_size * scale))
    font_english_scaled = ImageFont.truetype(font_english_path, int(base_font_size * scale))
    font_dob_scaled = ImageFont.truetype(font_dob_path, int((base_font_size - 3) * scale))
    font_aadhaar_scaled = ImageFont.truetype(font_aadhaar_path, int((base_font_size + 6) * scale)

    # Pick original details
    name_english, name_hindi, gender = random.choice(name_mapping)
    aadhaar_number = random_aadhaar_number()
    dob = random_dob()
    user_photo = get_random_photo(gender)

    # Insert photo
    photo_left, photo_top = int(w * 0.055), int(h * 0.27)
    card.paste(user_photo, (photo_left, photo_top))

    # Apply tampering based on batch
    batch = (i - 1) // 10 + 1

    if batch == 1:  # Name tampering
        fake_name_english, fake_name_hindi, _ = tamper_name(gender)
        name_english = fake_name_english
        name_hindi = fake_name_hindi
    elif batch == 2:  # Xerox simulation
        card = apply_xerox_effect(card)
    elif batch == 3:  # Blur effect
        card = apply_blur(card)
    elif batch == 4:  # Gender tampering
        gender = tamper_gender(gender)
```

```python
def random_dob():
    """Generate random date of birth."""
    return f"{random.randint(1, 28):02d}/{random.randint(1, 12):02d}/{random.randint(1965, 2000)}"

def get_random_photo(gender):
    """Get random photo based on gender."""
    folder = male_photo_folder if gender == "Male" else female_photo_folder
    photo_list = os.listdir(folder)
    random_photo = random.choice(photo_list)
    photo = Image.open(os.path.join(folder, random_photo)).convert("RGB")
    photo = photo.resize((192, 210), Image.LANCZOS)
    return photo

def tamper_name(original_gender):
    """Replace original name with a random fake name (gender matched)."""
    filtered_fake_names = [f for f in fake_name_mapping if f[2] == original_gender]
    return random.choice(filtered_fake_names)

def tamper_gender(original_gender):
    """Flip gender."""
    return "Female" if original_gender == "Male" else "Male"

def tamper_logo(card):
    """Remove logo from Aadhaar card."""
    draw = ImageDraw.Draw(card)
    logo_area = (int(card.width * 0.05), int(card.height * 0.05), int(card.width * 0.25), int(card.height * 0.15))
    draw.rectangle(logo_area, fill=(255, 255, 255))
    return card

def tamper_slogan(card):
    """Remove slogan from Aadhaar card."""
    draw = ImageDraw.Draw(card)
    slogan_area = (int(card.width * 0.3), int(card.height * 0.9), int(card.width * 0.7), int(card.height * 0.95))
    draw.rectangle(slogan_area, fill=(255, 255, 255))
    return card

def tamper_qr(card):
    """Remove or distort QR code."""
    draw = ImageDraw.Draw(card)
    qr_area = (int(card.width * 0.7), int(card.height * 0.05), int(card.width * 0.95), int(card.height * 0.25))
    if random.choice([True, False]):
        # Blank out QR
        draw.rectangle(qr_area, fill=(255, 255, 255))
    else:
        # Add noise
        for _ in range(1000):
            x = random.randint(qr_area[0], qr_area[2])
            y = random.randint(qr_area[1], qr_area[3])
            draw.point((x, y), fill=(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)))
    return card
```

EXAMPLE OF FAKE CARDS:

EXPLANATION OF THE CODE SNIPPET:

- This Python script generates **100 tampered Aadhaar card images** for testing and simulation purposes using the **Pillow (PIL)** library.
- It starts with a base template and overlays randomized user details like name (in English and Hindi), gender, date of birth, Aadhaar number, and a photograph selected based on gender.
- The script applies different tampering techniques across batches such as **name and gender mismatches, QR code distortion, logo/slogan removal, color band changes, blur effects, and xerox-like black-and-white simulation** to create realistic variations.
- Each card is saved in a specified output folder with sequential filenames for easy organization. This tool is ideal for creating synthetic datasets for training tamper-detection systems or testing document verification algorithms.

SPLITTING OF THE DATASET:

splitting.py > ...

```python
import os
import shutil
import random


original_real_dir = "generated_aadhaar_cards"
original_fake_dir = "tampered_aadhaar_cards"
base_dir = "dataset"


for split in ["train", "validation"]:
    for cls in ["real", "fake"]:
        split_dir = os.path.join(base_dir, split, cls)
        os.makedirs(split_dir, exist_ok=True)

train_ratio = 0.8

def split_and_copy(src_dir, dst_train_dir, dst_val_dir):
    images = os.listdir(src_dir)
    random.shuffle(images)

    train_count = int(len(images) * train_ratio)


    for img in images[:train_count]:
        src_path = os.path.join(src_dir, img)
        dst_path = os.path.join(dst_train_dir, img)
        shutil.copy2(src_path, dst_path)


    for img in images[train_count:]:
        src_path = os.path.join(src_dir, img)
        dst_path = os.path.join(dst_val_dir, img)
        shutil.copy2(src_path, dst_path)

split_and_copy(
    original_real_dir,
    os.path.join(base_dir, "train", "real"),
    os.path.join(base_dir, "validation", "real")
)


split_and_copy(
    original_fake_dir,
    os.path.join(base_dir, "train", "fake"),
    os.path.join(base_dir, "validation", "fake")
)

print(" Done splitting dataset!")
```

Here we have successfully split the dataset into training and validation.

80-20

TRAINING CODE SNIPPET:

trainings.py > ...

```python
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets
from transformers import CLIPProcessor, CLIPModel
from tqdm import tqdm
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Paths and settings
train_dir = "dataset/train"
val_dir = "dataset/validation"
batch_size = 8
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f" Using device: {device}")

# Load pretrained CLIP
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to(device)
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Freeze all CLIP layers first (then we can unfreeze later for fine-tuning)
for param in clip_model.parameters():
    param.requires_grad = False

# Custom classifier head
embed_dim = clip_model.config.projection_dim
classifier = nn.Sequential(
    nn.Linear(embed_dim, 256),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(256, 2)  # 2 classes: real, fake
).to(device)

# Dataset using CLIP preprocessing
class CLIPDataset(torch.utils.data.Dataset):
    def __init__(self, root_dir, processor):
        self.dataset = datasets.ImageFolder(root=root_dir)
        self.processor = processor

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

    def __getitem__(self, idx):
        image, label = self.dataset[idx]
        image = self.processor(images=image, return_tensors="pt")["pixel_values"].squeeze(0)
        return image, label
```

trainings.py > ...

```python

train_dataset = CLIPDataset(train_dir, clip_processor)
val_dataset = CLIPDataset(val_dir, clip_processor)

# Fix class imbalance
labels = [label for _, label in train_dataset.dataset.samples]
class_counts = torch.bincount(torch.tensor(labels))
class_weights = 1. / class_counts.float()
sample_weights = [class_weights[label] for label in labels]
sampler = torch.utils.data.WeightedRandomSampler(sample_weights, len(sample_weights))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, sampler=sampler)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

print(f" Class weights: {class_weights}")

# Focal Loss to handle imbalance
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
        self.ce = nn.CrossEntropyLoss(reduction='none')

    def forward(self, logits, targets):
        ce_loss = self.ce(logits, targets)
        pt = torch.exp(-ce_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma * ce_loss
        return focal_loss.mean() if self.reduction == 'mean' else focal_loss.sum()

criterion = FocalLoss()
optimizer = optim.AdamW(classifier.parameters(), lr=2e-4)

# Training loop
epochs = 15
best_val_acc = 0

for epoch in range(epochs):
    classifier.train()
    running_loss, correct, total = 0.0, 0, 0

    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}"):
        images, labels = images.to(device), labels.to(device)

        with torch.no_grad():  # Freeze CLIP encoder during this phase
            image_features = clip_model.get_image_features(pixel_values=images)

        outputs = classifier(image_features)
        loss = criterion(outputs, labels)
```

trainings.py > ...

```python
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

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

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

    # Validation
    classifier.eval()
    val_correct, val_total = 0, 0
    all_preds, all_labels = [], []

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            image_features = clip_model.get_image_features(pixel_values=images)
            outputs = classifier(image_features)
            _, predicted = torch.max(outputs, 1)

            val_correct += (predicted == labels).sum().item()
            val_total += labels.size(0)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    val_acc = 100 * val_correct / val_total
    print(f" Validation Acc: {val_acc:.2f}%")

    # Save best model
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save({
            'clip_model_state_dict': clip_model.state_dict(),
            'classifier_state_dict': classifier.state_dict()
        }, "best_model.pth")
        print(" Best model saved!")

# Classification report
print("\n Final Classification Report:")
print(classification_report(all_labels, all_preds, target_names=train_dataset.dataset.classes))

# Confusion matrix
cm = confusion_matrix(all_labels, all_preds)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=train_dataset.dataset.classes,
```

```python
            xticklabels=train_dataset.dataset.classes,
            yticklabels=train_dataset.dataset.classes)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

OUTPUT:

```
(.venv) PS D:\codings\machine_learning\open_cv> python trainings.py
 Using device: cpu
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior in v4.52, even
to use a slow processor with `use_fast=False`.
 Class weights: tensor([0.0101, 0.0093])
Epoch 1/15: 100%|██████████| 26/26 [00:26<00:00,  1.02s/it]
 Epoch [1/15], Loss: 0.1746, Train Acc: 56.04%
 Validation Acc: 56.78%
 Best model saved!
Epoch 2/15: 100%|██████████| 26/26 [00:26<00:00,  1.02s/it]
 Epoch [2/15], Loss: 0.1649, Train Acc: 59.90%
 Validation Acc: 65.25%
 Best model saved!
Epoch 3/15: 100%|██████████| 26/26 [00:25<00:00,  1.02it/s]
 Epoch [3/15], Loss: 0.1586, Train Acc: 65.70%
 Validation Acc: 48.31%
Epoch 4/15: 100%|██████████| 26/26 [00:26<00:00,  1.02s/it]
 Epoch [4/15], Loss: 0.1543, Train Acc: 64.25%
 Validation Acc: 57.63%
Epoch 5/15: 100%|██████████| 26/26 [00:20<00:00,  1.27it/s]
 Epoch [5/15], Loss: 0.1565, Train Acc: 67.15%
 Validation Acc: 73.73%
 Best model saved!
Epoch 6/15: 100%|██████████| 26/26 [00:27<00:00,  1.06s/it]
 Epoch [6/15], Loss: 0.1484, Train Acc: 69.08%
 Validation Acc: 58.47%
Epoch 7/15: 100%|██████████| 26/26 [00:22<00:00,  1.16it/s]
 Epoch [7/15], Loss: 0.1457, Train Acc: 70.05%
 Validation Acc: 83.05%
 Best model saved!
Epoch 8/15: 100%|██████████| 26/26 [00:21<00:00,  1.21it/s]
 Epoch [8/15], Loss: 0.1323, Train Acc: 83.09%
 Validation Acc: 90.68%
 Best model saved!
Epoch 9/15: 100%|██████████| 26/26 [00:20<00:00,  1.26it/s]
 Epoch [9/15], Loss: 0.1269, Train Acc: 83.57%
 Validation Acc: 85.59%
Epoch 10/15: 100%|█████████| 26/26 [00:21<00:00,  1.24it/s]
 Epoch [10/15], Loss: 0.1287, Train Acc: 77.29%
 Validation Acc: 58.47%
Epoch 11/15: 100%|█████████| 26/26 [00:19<00:00,  1.31it/s]
 Epoch [11/15], Loss: 0.1309, Train Acc: 78.74%
 Validation Acc: 89.83%
Epoch 12/15: 100%|█████████| 26/26 [00:17<00:00,  1.49it/s]
 Epoch [12/15], Loss: 0.1227, Train Acc: 77.78%
 Validation Acc: 79.66%
Epoch 13/15: 100%|█████████| 26/26 [00:17<00:00,  1.51it/s]
 Epoch [13/15], Loss: 0.1141, Train Acc: 79.23%
 Validation Acc: 92.37%
 Best model saved!
Epoch 14/15: 100%|█████████| 26/26 [00:16<00:00,  1.58it/s]
 Epoch [14/15], Loss: 0.1085, Train Acc: 83.09%
 Validation Acc: 83.05%
Epoch 15/15: 100%|█████████| 26/26 [00:18<00:00,  1.39it/s]
 Epoch [15/15], Loss: 0.1043, Train Acc: 88.41%
 Validation Acc: 88.98%
```

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

        fake       0.92      0.83      0.87        54
        real       0.87      0.94      0.90        64

    accuracy                           0.89       118
              precision    recall  f1-score   support

        fake       0.92      0.83      0.87        54
        real       0.87      0.94      0.90        64

    accuracy                           0.89       118

        fake       0.92      0.83      0.87        54
        real       0.87      0.94      0.90        64

    accuracy                           0.89       118

    accuracy                           0.89       118
    accuracy                           0.89       118
   macro avg       0.89      0.89      0.89       118
weighted avg       0.89      0.89      0.89       118
```

FINAL ACCURRACY OF THE MODEL: 90%