```
!pip install kaggle
```

Show hidden output

```
from google.colab import files
files.upload()
```

Choose Files kaggle (4).json

kaggle (4).json(application/json) - 71 bytes, last modified: 5/4/2025 - 100% done

```
Saving kaggle (4).json to kaggle (4).json
{'kaggle (4).json':
b'{"username":"dipakkhandagale","key":"163005f489516f9231ff065b93374c29"}'}
```

```
import os
os.environ['KAGGLE_USERNAME'] = 'dipakkhandagale'
os.environ['KAGGLE_KEY'] = '163005f489516f9231ff065b93374c29'
```

```
import kaggle
kaggle.api.authenticate()
```

```
!kaggle competitions download -c deepfake-detection-challenge
```

```
Downloading deepfake-detection-challenge.zip to /content
100% 4.13G/4.13G [01:03<00:00, 87.4MB/s]
100% 4.13G/4.13G [01:03<00:00, 69.8MB/s]
```

```
!unzip deepfake-detection-challenge.zip -d /content/data
```

Show hidden output

```
!pip install facenet-pytorch efficientnet_pytorch
# Ensure compatible versions of Pillow and torchvision are installed
!pip install Pillow==10.2.0 torchvision==0.17.1
```

Show hidden output

```
import os
orig = '/content/data/train sample videos'
tgt  = '/content/data/train_sample_videos'
if os.path.isdir(orig) and not os.path.isdir(tgt):
    os.rename(orig, tgt)
```

```
! pip install torch
```

Show hidden output

```
# --- Cell 4: Imports & device ---
import os
import json
from pathlib import Path
import cv2
from PIL import Image
from tqdm.auto import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as T
from torchvision import models
from facenet_pytorch import MTCNN

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

```
Using device: cuda
```

```
# --- Cell 5: Paths & params ---
VIDEO_DIR = Path('/content/data/train_sample_videos')
if not VIDEO_DIR.exists(): raise FileNotFoundError(f"Video dir not found: {VIDEO_I
possible_meta = [VIDEO_DIR / 'metadata.json', Path('/content/data/metadata.json')]
META_PATH = next((p for p in possible_meta if p.exists()), None)
if META_PATH is None: raise FileNotFoundError("metadata.json missing.")
print(f"Metadata: {META_PATH}")

FRAMES_DIR = Path('/content/frames')
NUM_FRAMES  = 16
FRAMES_DIR.mkdir(exist_ok=True)
```

```
Metadata: /content/data/train_sample_videos/metadata.json
```

```
# --- Cell 6: Extract frames ---
mtcnn  = MTCNN(image_size=224, margin=20, keep_all=False, device=DEVICE)
to_pil = T.ToPILImage()
for vid in tqdm(sorted(VIDEO_DIR.glob('*.mp4')), desc='Videos'):
    od = FRAMES_DIR / vid.stem
    od.mkdir(exist_ok=True)
    cap   = cv2.VideoCapture(str(vid))
    total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) or 1
    step  = max(total // NUM_FRAMES, 1)
    saved = 0
    for i in range(NUM_FRAMES):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i * step)
        ret, frm = cap.read()
        if not ret: break
        img = Image.fromarray(cv2.cvtColor(frm, cv2.COLOR_BGR2RGB))
        face = mtcnn(img)
        if face is None:
            w, h = img.size
            crop = img.crop(((w - h) // 2, 0, (w + h) // 2, h))
            face = T.Resize((224,224))(T.ToTensor()(crop))
        pil = to_pil(face)
        pil.save(od / f'frame_{i:04d}.jpg')
        saved += 1
    cap.release()
    if saved < NUM_FRAMES: print(f"⚠️ {vid.stem}: {saved}/{NUM_FRAMES}")
```

```
Videos:  100%                                    400/400  [1:37:23<00:00,  13.90s/it]
```

```
# --- Cell 7: Dataset & DataLoader ---
# Load metadata and strip extensions for consistency
with open(META_PATH) as f:
    raw_meta = json.load(f)
# Keys have '.mp4'; convert to stem
metadata = {Path(k).stem: raw_meta[k] for k in raw_meta}
names    = list(metadata.keys())

# Split 80/20
split     = int(0.8 * len(names))
train_keys = names[:split]
val_keys   = names[split:]
train_meta = {k: metadata[k] for k in train_keys}
val_meta   = {k: metadata[k] for k in val_keys}

class FrameDataset(Dataset):
    def __init__(self, root, meta, n_frames=NUM_FRAMES):
        self.root    = Path(root)
        self.meta    = meta
        self.names   = list(meta.keys())
        self.n_frames= n_frames
        self.tf = T.Compose([
            T.RandomHorizontalFlip(p=0.5),          # new
            T.ColorJitter(brightness=0.2,
                          contrast=0.2,
                          saturation=0.2,
                          hue=0.1),                 # new
            T.RandomRotation(degrees=5),            # new
            T.ToTensor(),
            T.Normalize([0.485,0.456,0.406],
                        [0.229,0.224,0.225])
        ])
    def __len__(self):
        return len(self.names)
    def __getitem__(self, idx):
        nm    = self.names[idx]
        label = 1.0 if self.meta[nm]['label'] == 'FAKE' else 0.0
        folder= self.root / nm
        frames_list = []
        for i in range(self.n_frames):
            fp = folder / f'frame_{i:04d}.jpg'
            if not fp.exists():
                raise FileNotFoundError(f"Missing frame: {fp}")
            img = Image.open(fp).convert('RGB')
            frames_list.append(self.tf(img))
        video_tensor = torch.stack(frames_list, dim=0)
        return video_tensor, torch.tensor(label, dtype=torch.float32)

# Create loaders
train_ds    = FrameDataset(FRAMES_DIR, train_meta)
val_ds      = FrameDataset(FRAMES_DIR, val_meta)
train_loader= DataLoader(train_ds, batch_size=4, shuffle=True,  num_workers=2, pir
val_loader  = DataLoader(val_ds,   batch_size=4, shuffle=False, num_workers=2, pir
```

```
# --- Cell 8.1: Imports & Device Setup ---
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models
from tqdm import tqdm
import os

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

```
✅ Using device: cuda
```

```
# --- Cell 8.2: Define the Model ---
class ResNet50BiLSTM(nn.Module):
    def __init__(self, hidden=256):
        super().__init__()
        # Load pretrained ResNet50 backbone
        base = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)

        # Remove the last FC layer — keep convolutional feature extractor
        self.cnn = nn.Sequential(*list(base.children())[:-1])

        # LSTM to model temporal sequence of frame features
        self.lstm = nn.LSTM(2048, hidden, batch_first=True, bidirectional=True)

        # Classification head
        self.head = nn.Sequential(
            nn.Linear(hidden * 2, 128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        B, T, C, H, W = x.shape  # (Batch, Frames, Channels, Height, Width)
        x = x.view(B * T, C, H, W)

        # CNN feature extraction per frame
        feats = self.cnn(x).view(B, T, -1)  # (B, T, 2048)

        # LSTM for temporal modeling
        lstm_out, _ = self.lstm(feats)      # (B, T, 2*hidden)

        # Take last timestep output
        last_out = lstm_out[:, -1, :]       # (B, 2*hidden)
        # Alternative: mean pooling (uncomment if preferred)
        # last_out = lstm_out.mean(dim=1)

        # Final classification
        out = self.head(last_out)           # (B, 1)
        return out
```

```
# --- Cell 8.3: Initialize Model, Optimizer, and Loss ---
model = ResNet50BiLSTM().to(DEVICE)
opt   = optim.Adam(model.parameters(), lr=1e-4)
crit  = nn.BCELoss()
epochs = 50

# Create directory to save model checkpoints
model_save_dir = '/content/models'
os.makedirs(model_save_dir, exist_ok=True)

print("✅ Model, optimizer, and loss function initialized successfully.")
```

```
✅ Model, optimizer, and loss function initialized successfully.
```

```
# --- Cell 8.4: Training & Validation Loop (Fixed) ---
for e in range(1, epochs + 1):
    model.train()
    train_loss = 0.0

    # ---------- Training ----------
    for vids, labels in tqdm(train_loader, desc=f"Epoch {e} Training"):
        vids, labels = vids.to(DEVICE), labels.to(DEVICE)
        labels = labels.view(-1, 1)  # ✅ FIX: reshape labels from [B] → [B, 1
        labels = labels.view(-1, 1)   # ✅ FIX: reshape labels from [B] → [B,

        preds = model(vids)
        loss = crit(preds, labels)

        opt.zero_grad()
        loss.backward()
        opt.step()

        train_loss += loss.item() * vids.size(0)

    train_loss /= len(train_loader.dataset)

    # ---------- Validation ----------
    model.eval()
    val_loss, correct = 0.0, 0
    with torch.no_grad():
        for vids, labels in tqdm(val_loader, desc=f"Epoch {e} Validation"):
            vids, labels = vids.to(DEVICE), labels.to(DEVICE)
            labels = labels.view(-1, 1)   # ✅ FIX again here

            preds = model(vids)
            l = crit(preds, labels)

            val_loss += l.item() * vids.size(0)
            correct += ((preds > 0.5).float() == labels).sum().item()

    val_loss /= len(val_loader.dataset)
    val_acc = correct / len(val_loader.dataset)

    print(f"Epoch {e}: Train Loss={train_loss:.4f} | Val Loss={val_loss:.4f} |

    # Save model checkpoint
    torch.save(model.state_dict(), os.path.join(model_save_dir, f"model_epoch_{
```

```
Epoch 1 Training: 100%|██████████| 80/80 [00:52<00:00,  1.52it/s]
Epoch 1 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.26it/s]
Epoch 1: Train Loss=0.5483 | Val Loss=0.4426 | Val Acc=0.8125
Epoch 2 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
Epoch 2 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.10it/s]
Epoch 2: Train Loss=0.4229 | Val Loss=0.4792 | Val Acc=0.8125
Epoch 3 Training: 100%|██████████| 80/80 [00:50<00:00,  1.58it/s]
Epoch 3 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.07it/s]
Epoch 3: Train Loss=0.3854 | Val Loss=0.4258 | Val Acc=0.8500
Epoch 4 Training: 100%|██████████| 80/80 [00:50<00:00,  1.58it/s]
Epoch 4 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.07it/s]
Epoch 4: Train Loss=0.3247 | Val Loss=0.4719 | Val Acc=0.7875
Epoch 5 Training: 100%|██████████| 80/80 [00:50<00:00,  1.58it/s]
Epoch 5 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.34it/s]
Epoch 5: Train Loss=0.2637 | Val Loss=0.5063 | Val Acc=0.8000
Epoch 6 Training: 100%|██████████| 80/80 [00:51<00:00,  1.56it/s]
Epoch 6 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.09it/s]
Epoch 6: Train Loss=0.2086 | Val Loss=0.4495 | Val Acc=0.8625
Epoch 7 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
Epoch 7 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.08it/s]
Epoch 7: Train Loss=0.2018 | Val Loss=0.8224 | Val Acc=0.7125
Epoch 8 Training: 100%|██████████| 80/80 [00:50<00:00,  1.59it/s]
Epoch 8 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.29it/s]
Epoch 8: Train Loss=0.1445 | Val Loss=0.6443 | Val Acc=0.8125
Epoch 9 Training: 100%|██████████| 80/80 [00:51<00:00,  1.56it/s]
Epoch 9 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.12it/s]
Epoch 9: Train Loss=0.1743 | Val Loss=0.5970 | Val Acc=0.7750
Epoch 10 Training: 100%|██████████| 80/80 [00:51<00:00,  1.54it/s]
Epoch 10 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.04it/s]
Epoch 10: Train Loss=0.1756 | Val Loss=0.5876 | Val Acc=0.7625
Epoch 11 Training: 100%|██████████| 80/80 [00:50<00:00,  1.59it/s]
Epoch 11 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.09it/s]
Epoch 11: Train Loss=0.1706 | Val Loss=0.5472 | Val Acc=0.8000
Epoch 12 Training: 100%|██████████| 80/80 [00:51<00:00,  1.57it/s]
Epoch 12 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.32it/s]
Epoch 12: Train Loss=0.1061 | Val Loss=0.6960 | Val Acc=0.7875
Epoch 13 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
Epoch 13 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.16it/s]
Epoch 13: Train Loss=0.1701 | Val Loss=0.5970 | Val Acc=0.8125
Epoch 14 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
Epoch 14 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.10it/s]
Epoch 14: Train Loss=0.0588 | Val Loss=0.6011 | Val Acc=0.8500
Epoch 15 Training: 100%|██████████| 80/80 [00:50<00:00,  1.59it/s]
Epoch 15 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.10it/s]
Epoch 15: Train Loss=0.0506 | Val Loss=0.7019 | Val Acc=0.8000
Epoch 16 Training: 100%|██████████| 80/80 [00:50<00:00,  1.59it/s]
Epoch 16 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.36it/s]
Epoch 16: Train Loss=0.1162 | Val Loss=0.7844 | Val Acc=0.7500
Epoch 17 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
Epoch 17 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.13it/s]
Epoch 17: Train Loss=0.1095 | Val Loss=0.8765 | Val Acc=0.7875
Epoch 18 Training: 100%|██████████| 80/80 [00:50<00:00,  1.58it/s]
Epoch 18 Validation: 100%|██████████| 20/20 [00:09<00:00,  2.03it/s]
Epoch 18: Train Loss=0.0547 | Val Loss=0.8533 | Val Acc=0.7875
Epoch 19 Training: 100%|██████████| 80/80 [00:50<00:00,  1.59it/s]
Epoch 19 Validation: 100%|██████████| 20/20 [00:08<00:00,  2.28it/s]
Epoch 19: Train Loss=0.0451 | Val Loss=0.6647 | Val Acc=0.7750
Epoch 20 Training: 100%|██████████| 80/80 [00:50<00:00,  1.57it/s]
```

Start coding or generate with AI.