Download & Extracting CoMoFoD Dataset

```
# STEP 1: Download & Extract CoMoFoD from Kaggle

from google.colab import files
import os

print("Please upload your Kaggle API JSON file (kaggle.json):")
uploaded = files.upload()  # Upload your kaggle.json when prompted

# Move kaggle.json to ~/.kaggle and set permissions
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download and unzip the CoMoFoD dataset
!kaggle datasets download tusharchauhan1898/comofod --unzip -p ./comofod

# Print the top-level folder structure to verify
print("\nCoMoFoD Dataset folder structure (showing top levels):")
for root, dirs, files in os.walk('./comofod'):
    level = root.replace('./comofod', '').count(os.sep)
    indent = ' ' * 4 * (level)
    print(f"{indent}{os.path.basename(root)}/")
    if level > 2:
        continue  # Skip deep subfolders for brevity
    subindent = ' ' * 4 * (level + 1)
    for f in files[:5]:  # Show first 5 files per folder
        print(f"{subindent}{f}")
```

```
Please upload your Kaggle API JSON file (kaggle.json):
Choose Files kaggle.json
• kaggle.json(application/json) - 64 bytes, last modified: 6/22/2025 - 100% done
Saving kaggle.json to kaggle.json
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / client 1.6.17)
Dataset URL: https://www.kaggle.com/datasets/tusharchauhan1898/comofod
License(s): unknown
Downloading comofod.zip to ./comofod
100% 2.98G/2.99G [00:18<00:00, 162MB/s]
100% 2.99G/2.99G [00:18<00:00, 173MB/s]

CoMoFoD Dataset folder structure (showing top levels):
comofod/
    CoMoFoD_small_v2/
        129_O_CA1.png
        163_O_NA3.png
        095_F_CR1.png
        038_M.png
        016_F_JC5.jpg
```

Data Preparation

```
import os
import random
import shutil
from collections import Counter
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image

# Set base paths
base_dir = 'comofod/CoMoFoD_small_v2'
split_dir = 'comofod/split'
os.makedirs(split_dir, exist_ok=True)

# Classify: 'F' in filename = Tampered (Tp), 'O' or 'M' = Authentic (Au)
def is_tp(filename): return '_F_' in filename or '_F.' in filename or filename.startswith('F_') or '_F' in filename
def is_au(filename): return '_O_' in filename or '_O.' in filename or filename.startswith('O_') or '_O' in filename or '_M' in filename

all_files = [f for f in os.listdir(base_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
tp_files = [f for f in all_files if is_tp(f)]
au_files = [f for f in all_files if is_au(f)]

print(f"Tamp.: {len(tp_files)} | Authentic: {len(au_files)} | Total: {len(all_files)}")

# Split ratios
train_pct, val_pct, test_pct = 0.7, 0.15, 0.15
random.seed(42)
```

```
splits = {'train': {}, 'val': {}, 'test': {}}
class_names = ['Tp', 'Au']

for cls, files in zip(class_names, [tp_files, au_files]):
    n_total = len(files)
    random.shuffle(files)
    n_train = int(n_total * train_pct)
    n_val = int(n_total * val_pct)
    n_test = n_total - n_train - n_val
    splits['train'][cls] = files[:n_train]
    splits['val'][cls] = files[n_train:n_train+n_val]
    splits['test'][cls] = files[n_train+n_val:]
    # Copy files to split folders
    for split in ['train', 'val', 'test']:
        target_dir = os.path.join(split_dir, split, cls)
        os.makedirs(target_dir, exist_ok=True)
        for f in splits[split][cls]:
            shutil.copy2(os.path.join(base_dir, f), os.path.join(target_dir, f))

# Print summary as a table
summary_df = pd.DataFrame({split: {cls: len(splits[split][cls]) for cls in class_names} for split in ['train', 'val', 'test']}).T
summary_df['Total'] = summary_df['Tp'] + summary_df['Au']
summary_df.columns = ['Tampered (Tp)', 'Authentic (Au)', 'Total']
print("\nDataset Split Summary:")
from IPython.display import display
display(summary_df)

# Show sample images for both classes
def show_samples(split, cls, n=3):
    img_dir = os.path.join(split_dir, split, cls)
    img_files = random.sample(os.listdir(img_dir), min(n, len(os.listdir(img_dir))))
    plt.figure(figsize=(12, 3))
    for i, f in enumerate(img_files):
        img = Image.open(os.path.join(img_dir, f))
        plt.subplot(1, n, i+1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"{split}/{cls}\n{f}")
    plt.show()

print("Sample Tampered (Tp) images from train split:")
show_samples('train', 'Tp')
print("Sample Authentic (Au) images from train split:")
show_samples('train', 'Au')
```

```
Tamp.: 5000 | Authentic: 5200 | Total: 10400
```

Dataset Split Summary:

	Tampered (Tp)	Authentic (Au)	Total
train	3500	3639	7139
val	750	780	1530
test	750	781	1531

Sample Tampered (Tp) images from train split:

Sample Authentic (Au) images from train split:

Next steps: Generate code with summary_df | View recommended plots | New interactive sheet

Dataloaders & Transforms for DenseNet-201

```
import torch
from torchvision import datasets, transforms

IMG_SIZE = 224  # Standard for DenseNet
BATCH_SIZE = 16

# Image transforms: note ImageNet normalization (DenseNet-201 is pretrained on ImageNet)
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
}

# Create datasets and dataloaders for each split
image_datasets = {x: datasets.ImageFolder(os.path.join(split_dir, x),
                                          data_transforms[x])
```

```
                  for x in ['train', 'val', 'test']}

dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=BATCH_SIZE,
                                              shuffle=True if x == 'train' else False, num_workers=2)
               for x in ['train', 'val', 'test']}

class_names = image_datasets['train'].classes
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Class Names:", class_names)
print("Device:", device)
```

```
Class Names: ['Au', 'Tp']
Device: cuda
```

Model Setup – DenseNet-201 for binary classification

```
import torchvision.models as models
import torch.nn as nn

# Load pre-trained DenseNet-201
model = models.densenet201(pretrained=True)

# Modify the classifier for 2 output classes (Tampered, Authentic)
num_ftrs = model.classifier.in_features
model.classifier = nn.Linear(num_ftrs, 2)  # Two classes: Tp, Au

# Move model to device (GPU/CPU)
model = model.to(device)

print("DenseNet-201 binary classification model ready.")
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/densenet201-c1103571.pth" to /root/.cache/torch/hub/checkpoints/densenet201-c1103
100%|██████████| 77.4M/77.4M [00:00<00:00, 188MB/s]
DenseNet-201 binary classification model ready.
```

Loss, Optimizer, Scheduler setup

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

# Loss function: CrossEntropyLoss for binary classification (with logits)
criterion = nn.CrossEntropyLoss()

# Optimizer: Adam (robust for fine-tuning)
optimizer = optim.Adam(model.parameters(), lr=1e-4)

# Scheduler: Reduce learning rate by half every 5 epochs
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

print("Loss, optimizer, and scheduler set up.")
```

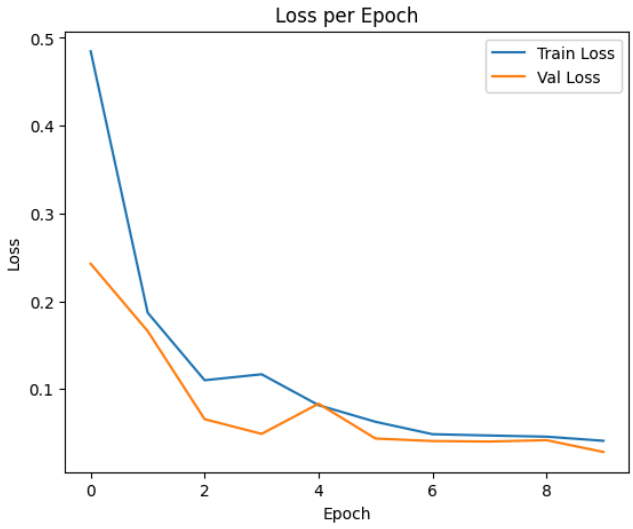
```
Loss, optimizer, and scheduler set up.
```

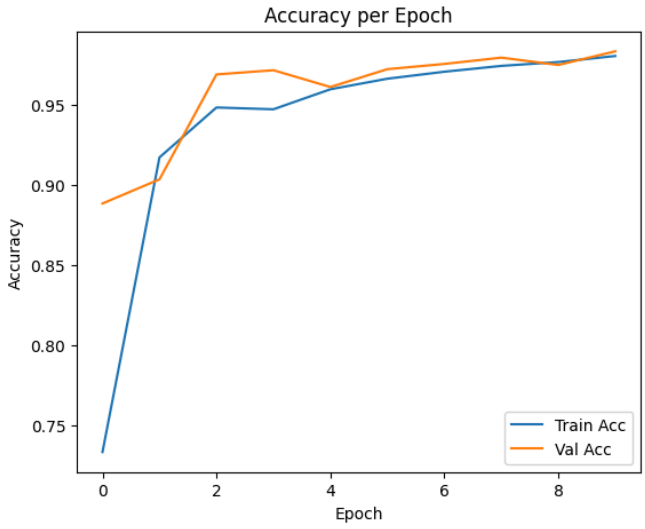
Training Loop with Learning Curves

```
import time
import copy
import matplotlib.pyplot as plt
import pandas as pd

num_epochs = 10  # Adjust as needed

train_acc_history = []
val_acc_history = []
train_loss_history = []
val_loss_history = []
history_table = []

best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0
```

```
    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch+1}/{num_epochs}")
        print("-" * 20)
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
            epoch_acc = running_corrects.double() / len(image_datasets[phase])
            print(f"{phase.capitalize()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

            if phase == 'train':
                train_loss_history.append(epoch_loss)
                train_acc_history.append(epoch_acc.item())
                scheduler.step()
            else:
                val_loss_history.append(epoch_loss)
                val_acc_history.append(epoch_acc.item())
                if epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())
        # Log to table after both phases
        history_table.append({
            "Epoch": epoch + 1,
            "Train Loss": train_loss_history[-1],
            "Train Acc": train_acc_history[-1],
            "Val Loss": val_loss_history[-1],
            "Val Acc": val_acc_history[-1]
        })

    model.load_state_dict(best_model_wts)

    # Show learning curves
    plt.figure(figsize=(14, 5))
    plt.subplot(1,2,1)
    plt.plot(train_loss_history, label='Train Loss')
    plt.plot(val_loss_history, label='Val Loss')
    plt.title('Loss per Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(train_acc_history, label='Train Acc')
    plt.plot(val_acc_history, label='Val Acc')
    plt.title('Accuracy per Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    # Show table
    history_df = pd.DataFrame(history_table)
    print("\nEpoch-wise Training/Validation History:")
    from IPython.display import display
    display(history_df)
```

```
Epoch 1/10
--------------------
Train Loss: 0.4847 Acc: 0.7334
Val Loss: 0.2427 Acc: 0.8882

Epoch 2/10
--------------------
Train Loss: 0.1871 Acc: 0.9169
Val Loss: 0.1661 Acc: 0.9033

Epoch 3/10
--------------------
Train Loss: 0.1102 Acc: 0.9480
Val Loss: 0.0658 Acc: 0.9686

Epoch 4/10
--------------------
Train Loss: 0.1169 Acc: 0.9469
Val Loss: 0.0491 Acc: 0.9712

Epoch 5/10
--------------------
Train Loss: 0.0817 Acc: 0.9594
Val Loss: 0.0834 Acc: 0.9608

Epoch 6/10
--------------------
Train Loss: 0.0628 Acc: 0.9660
Val Loss: 0.0438 Acc: 0.9719

Epoch 7/10
--------------------
Train Loss: 0.0487 Acc: 0.9703
Val Loss: 0.0409 Acc: 0.9752

Epoch 8/10
--------------------
Train Loss: 0.0472 Acc: 0.9739
Val Loss: 0.0404 Acc: 0.9791

Epoch 9/10
--------------------
Train Loss: 0.0459 Acc: 0.9763
Val Loss: 0.0419 Acc: 0.9745

Epoch 10/10
--------------------
Train Loss: 0.0412 Acc: 0.9801
Val Loss: 0.0285 Acc: 0.9830
```





Epoch-wise Training/Validation History:

	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	1	0.484668	0.733436	0.242732	0.888235
1	2	0.187107	0.916935	0.166144	0.903268
2	3	0.110167	0.948032	0.065778	0.968627
3	4	0.116928	0.946911	0.049085	0.971242
4	5	0.081749	0.959378	0.083416	0.960784
5	6	0.062811	0.965962	0.043763	0.971895
6	7	0.048715	0.970304	0.040930	0.975163

7	8	0.047217	0.973946	0.040357	0.979085
8	9	0.045881	0.976327	0.041923	0.974510
9	10	0.041218	0.980109	0.028467	0.983007

Next steps: Generate code with `history_df` View recommended plots New interactive sheet

Evaluation & Metrics Table

```
import torch
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score, jaccard_score
from scipy.stats import ttest_ind
import time

# Evaluate on the test set
model.eval()
all_preds, all_labels = [], []
with torch.no_grad():
    for inputs, labels in dataloaders['test']:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Basic Metrics
acc = accuracy_score(all_labels, all_preds)
prec = precision_score(all_labels, all_preds)
rec = recall_score(all_labels, all_preds)
f1 = f1_score(all_labels, all_preds)
cm = confusion_matrix(all_labels, all_preds)
false_negatives = cm[1,0]
true_negatives = cm[0,0]
true_positives = cm[1,1]
false_positives = cm[0,1]

# Number of parameters
param_count = sum(p.numel() for p in model.parameters())

# FLOPs (MACs) using ptflops (install if needed)
try:
    !pip install ptflops --quiet
    from ptflops import get_model_complexity_info
    macs, params = get_model_complexity_info(model, (3, 224, 224), as_strings=False, print_per_layer_stat=False)
    flops = macs
except Exception:
    flops = 'N/A (install ptflops)'

# Inference time (ms per image, averaged over 50 runs)
n_runs = 50
sample = torch.rand(1, 3, 224, 224).to(device)
start = time.time()
with torch.no_grad():
    for _ in range(n_runs):
        _ = model(sample)
elapsed = (time.time() - start) / n_runs * 1000  # ms per image

# IOU (mean per class, Jaccard index)
iou = jaccard_score(all_labels, all_preds, average=None)
mean_iou = np.mean(iou) * 100  # %

# Mean accuracy diff (between classwise accs)
cm = confusion_matrix(all_labels, all_preds)
class_accs = cm.diagonal() / cm.sum(axis=1)
mean_acc_diff = np.abs(class_accs[0] - class_accs[1])

# p-value (t-test between predicted and true labels)
tstat, pval = ttest_ind(all_labels, all_preds)

# Tabulate all results
metrics_table = pd.DataFrame({
    "Metric": [
        "Accuracy", "Precision", "Recall", "F1 Score",
        "False Negatives", "True Negatives", "True Positives", "False Positives",
        "Parameter Count", "FLOPs (MACs)",
```

```
        "Inference Time (ms)", "Mean IOU (%)", "Mean Accuracy Diff", "p-value (t-test)"
    ],
    "Value": [
        f"{acc:.4f}", f"{prec:.4f}", f"{rec:.4f}", f"{f1:.4f}",
        false_negatives, true_negatives, true_positives, false_positives,
        f"{param_count:,}", flops if isinstance(flops, str) else f"{flops:,}",
        f"{elapsed:.2f}", f"{mean_iou:.2f}", f"{mean_acc_diff:.4f}", f"{pval:.4g}"
    ]
})

print("\nFull Metrics Table:")
from IPython.display import display
display(metrics_table)

# Display confusion matrix
print("\nConfusion Matrix (Rows: True, Columns: Predicted):")
print(cm)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 118.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 98.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 52.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 42.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 3.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 99.8 MB/s eta 0:00:00
```

Full Metrics Table:

	Metric	Value
0	Accuracy	0.9850
1	Precision	0.9892
2	Recall	0.9800
3	F1 Score	0.9846
4	False Negatives	15
5	True Negatives	773
6	True Positives	735
7	False Positives	8
8	Parameter Count	18,096,770
9	FLOPs (MACs)	4,391,432,450
10	Inference Time (ms)	33.71
11	Mean IOU (%)	97.04
12	Mean Accuracy Diff	0.0098
13	p-value (t-test)	0.8003

```
Confusion Matrix (Rows: True, Columns: Predicted):
[[773   8]
 [ 15 725]]
```

Next steps: Generate code with `metrics_table` View recommended plots New interactive sheet

LRP Explainability

```
!pip install captum --quiet

from captum.attr import LayerDeepLift
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
from PIL import Image
import io, base64
from IPython.display import display, HTML

# Parameters
IMG_SIZE = 224
N_GRID = 9
N_TABLE = 5
tp_dir = os.path.join(split_dir, 'test', 'Tp')
tp_imgs = [f for f in os.listdir(tp_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
```

```
sample_files = tp_imgs[:max(N_GRID, N_TABLE)]

densenet_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

def load_tensor(img_path):
    img = Image.open(img_path).convert("RGB")
    return densenet_transform(img).unsqueeze(0).to(device)

# Use the last DenseBlock for attributions (DenseNet-201: features.denseblock4)
lrp = LayerDeepLift(model, model.features.denseblock4)

img_names, lrp_scores, lrp_heatmaps, overlays = [], [], [], []

for fname in sample_files:
    img_path = os.path.join(tp_dir, fname)
    input_tensor = load_tensor(img_path)
    input_tensor.requires_grad_()
    output = model(input_tensor)
    pred = torch.argmax(output, dim=1).item()
    # Attribution (LRP)
    attributions = lrp.attribute(input_tensor, target=pred)
    attr_map = attributions.squeeze().detach().cpu().numpy()
    attr_map = np.sum(attr_map, axis=0)  # sum over channels
    score = np.sum(np.abs(attr_map))
    lrp_scores.append(score)
    img_names.append(fname)
    lrp_heatmaps.append(attr_map)
    img_np = input_tensor.squeeze().permute(1,2,0).detach().cpu().numpy()
    img_disp = np.clip(img_np * [0.229,0.224,0.225] + [0.485,0.456,0.406], 0, 1)
    attr_norm = (attr_map - attr_map.min()) / (attr_map.max() - attr_map.min() + 1e-8)
    overlays.append((img_disp, attr_norm))

# --- 3x3 Grid Display ---
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
for i in range(min(N_GRID, len(sample_files))):
    row, col = divmod(i, 3)
    img_disp, attr_norm = overlays[i]
    axs[row, col].imshow(img_disp)
    axs[row, col].imshow(attr_norm, cmap='hot', alpha=0.4)
    axs[row, col].set_title(f"{img_names[i]}\nLRP Score: {lrp_scores[i]:.2f}", fontsize=10)
    axs[row, col].axis('off')
for i in range(len(sample_files), 9):
    row, col = divmod(i, 3)
    axs[row, col].axis('off')
plt.suptitle("LRP Overlays for Tampered Images (3x3 Grid)", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# --- Tabular Display of 5 Images ---
def fig2img(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png', bbox_inches='tight')
    buf.seek(0)
    img = Image.open(buf)
    return img

def img_to_html(img):
    buf = io.BytesIO()
    img.save(buf, format='PNG')
    buf.seek(0)
    data = base64.b64encode(buf.read()).decode('utf-8')
    return f'<img src="{data}" width="120"/>'

rows = []
for i in range(N_TABLE):
    img_disp, attr_norm = overlays[i]
    img_disp_255 = (img_disp * 255).astype(np.uint8)
    orig_pil = Image.fromarray(img_disp_255)
    # LRP heatmap
    fig1, ax1 = plt.subplots()
    ax1.imshow(lrp_heatmaps[i], cmap='seismic')
    ax1.axis('off')
    ax1.set_title('LRP Heatmap')
    lrp_heatmap_img = fig2img(fig1)
    plt.close(fig1)
    # Overlay
    fig2, ax2 = plt.subplots()
```

```
        ax2.imshow(img_disp)
        ax2.imshow(attr_norm, cmap='hot', alpha=0.4)
        ax2.axis('off')
        ax2.set_title('Overlay')
        overlay_img_pil = fig2img(fig2)
        plt.close(fig2)
        row_html = f"""
        <tr>
            <td>{img_names[i]}</td>
            <td>{img_to_html(lrp_heatmap_img)}</td>
            <td>{lrp_scores[i]:.2f}</td>
            <td>{img_to_html(overlay_img_pil)}</td>
        </tr>
        """
        rows.append(row_html)

    table_html = f"""
    <table>
        <tr>
            <th>Image Name</th>
            <th>LRP Heatmap</th>
            <th>LRP Score</th>
            <th>Overlay</th>
        </tr>
        {''.join(rows)}
    </table>
    """
    display(HTML(table_html))

    # --- Save results to CSV ---
    csv_df = pd.DataFrame({
        'Image Name': img_names[:N_TABLE],
        'LRP Score': lrp_scores[:N_TABLE]
    })
    csv_df.to_csv("lrp_results_densenet201_comofod.csv", index=False)
    print("LRP results (image name, score) saved to lrp_results_densenet201_comofod.csv")
```

LRP Overlays for Tampered Images (3x3 Grid)

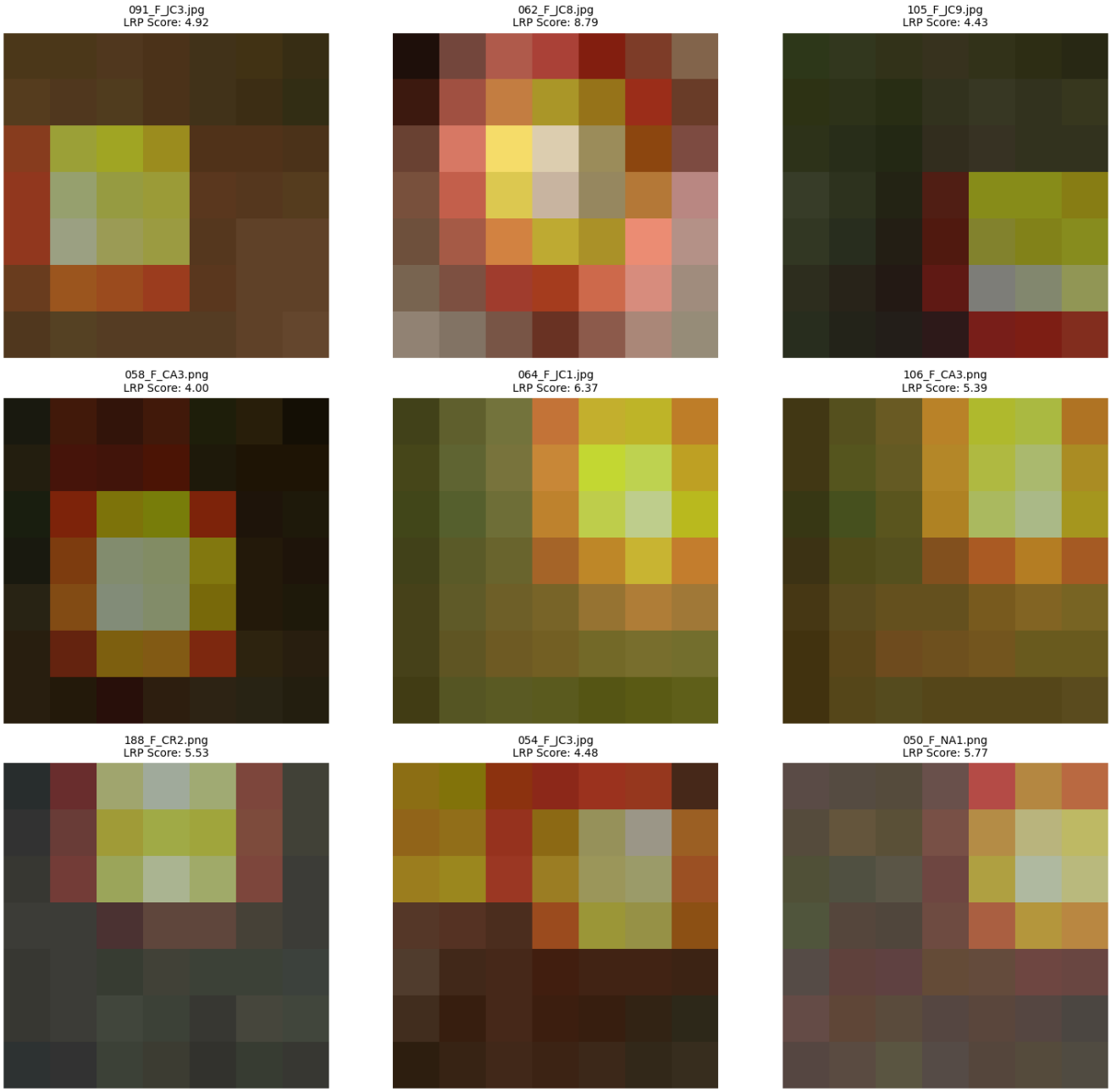
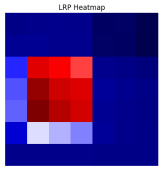
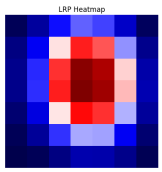
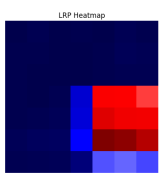
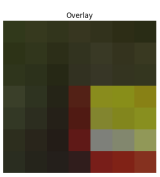
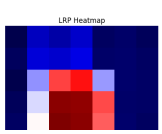
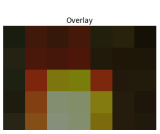
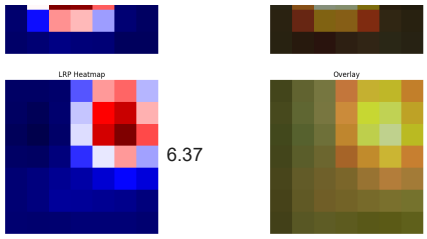


Image Name	LRP Heatmap	LRP Score	Overlay
091_F_JC3.jpg		4.92	
062_F_JC8.jpg		8.79	
105_F_JC9.jpg		4.43	
058_F_CA3.png		4.00	



064_F_JC1.jpg 6.37

```
LRP results (image name, score) saved to lrp_results_densenet201_comofod.csv
```

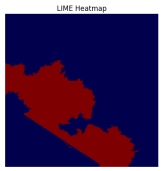
LIME Explainability

```
!pip install lime scikit-image --quiet

import numpy as np
from lime import lime_image
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt
import io, base64
from PIL import Image
from IPython.display import display, HTML

# ---- Reuse variables: N_GRID, N_TABLE, img_names, sample_files, tp_dir ----

def preprocess_for_lime(img_path):
    img = Image.open(img_path).convert('RGB').resize((IMG_SIZE, IMG_SIZE))
    img_np = np.array(img)
    return img_np

# Batch predict for LIME: DenseNet expects normalized tensors
def batch_predict(images):
    model.eval()
    images = [torch.tensor(i.transpose((2,0,1))).float() / 255.0 for i in images]
    images = torch.stack(images)
    for i in range(3):
        images[:,i,:,:] = (images[:,i,:,:] - [0.485,0.456,0.406][i]) / [0.229,0.224,0.225][i]
    images = images.to(device)
    with torch.no_grad():
        logits = model(images)
        probs = torch.softmax(logits, dim=1).cpu().numpy()
    return probs

lime_scores, lime_heatmaps, overlays_lime, origs = [], [], [], []
explainer = lime_image.LimeImageExplainer()

for fname in sample_files:
    img_path = os.path.join(tp_dir, fname)
    img_np = preprocess_for_lime(img_path)
    explanation = explainer.explain_instance(
        img_np,
        batch_predict,
        top_labels=1,
        hide_color=0,
        num_samples=1000
    )
    pred_class = explanation.top_labels[0]
    img_lime, mask_lime = explanation.get_image_and_mask(
        pred_class, positive_only=True, num_features=5, hide_rest=False
    )
    lime_score = np.sum([abs(weight) for _, weight in explanation.local_exp[pred_class]])
    lime_scores.append(lime_score)
    lime_heatmaps.append(mask_lime)
    overlays_lime.append(mark_boundaries(img_np, mask_lime))
    origs.append(img_np)

# --- 3x3 Grid Display ---
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
for i in range(min(N_GRID, len(sample_files))):
    row, col = divmod(i, 3)
    axs[row, col].imshow(overlays_lime[i])
    axs[row, col].set_title(f"{img_names[i]}\nLIME Score: {lime_scores[i]:.2f}", fontsize=10)
    axs[row, col].axis('off')
for i in range(len(sample_files), 9):
    row, col = divmod(i, 3)
```

```
        axs[row, col].axis('off')
    plt.suptitle("LIME Overlays for Tampered Images (3x3 Grid)", fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

    # --- Tabular Display of 5 Images ---
    def fig2img(fig):
        buf = io.BytesIO()
        fig.savefig(buf, format='png', bbox_inches='tight')
        buf.seek(0)
        img = Image.open(buf)
        return img

    def img_to_html(img):
        buf = io.BytesIO()
        img.save(buf, format='PNG')
        buf.seek(0)
        data = base64.b64encode(buf.read()).decode('utf-8')
        return f'<img src="{data}" width="120"/>'

    rows = []
    for i in range(N_TABLE):
        # LIME heatmap
        fig1, ax1 = plt.subplots()
        ax1.imshow(lime_heatmaps[i], cmap='seismic')
        ax1.axis('off')
        ax1.set_title('LIME Heatmap')
        lime_heatmap_img = fig2img(fig1)
        plt.close(fig1)
        # Overlay
        overlay_img_pil = Image.fromarray((overlays_lime[i]*255).astype(np.uint8))
        row_html = f"""
        <tr>
            <td>{img_names[i]}</td>
            <td>{img_to_html(lime_heatmap_img)}</td>
            <td>{lime_scores[i]:.2f}</td>
            <td>{img_to_html(overlay_img_pil)}</td>
        </tr>
        """
        rows.append(row_html)

    table_html = f"""
    <table>
        <tr>
            <th>Image Name</th>
            <th>LIME Heatmap</th>
            <th>LIME Score</th>
            <th>Overlay</th>
        </tr>
        {''.join(rows)}
    </table>
    """
    display(HTML(table_html))

    # --- Save results to CSV ---
    csv_df = pd.DataFrame({
        'Image Name': img_names[:N_TABLE],
        'LIME Score': lime_scores[:N_TABLE]
    })
    csv_df.to_csv("lime_results_densenet201_comofod.csv", index=False)
    print("LIME results (image name, score) saved to lime_results_densenet201_comofod.csv")
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 5.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
100%                                    1000/1000 [00:06<00:00, 160.94it/s]
100%                                    1000/1000 [00:06<00:00, 154.57it/s]
100%                                    1000/1000 [00:06<00:00, 160.44it/s]
100%                                    1000/1000 [00:06<00:00, 160.33it/s]
100%                                    1000/1000 [00:06<00:00, 159.32it/s]
100%                                    1000/1000 [00:06<00:00, 158.60it/s]
100%                                    1000/1000 [00:06<00:00, 159.33it/s]
100%                                    1000/1000 [00:06<00:00, 160.41it/s]
100%                                    1000/1000 [00:06<00:00, 156.68it/s]
```

LIME Overlays for Tampered Images (3x3 Grid)

Image Name	LIME Heatmap	LIME Score	Overlay
091_F_JC3.jpg		1.35