Dataset Download & folder verification

```
# STEP 1: Download CoMoFoD from Kaggle and print folder structure

from google.colab import files
import os

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

# Move to Kaggle location 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 folder structure (top 2 levels)
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  # To avoid printing too many files
    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
Dataset URL: https://www.kaggle.com/datasets/tusharchauhan1898/comofod
License(s): unknown
Downloading comofod.zip to ./comofod
 97% 2.91G/2.99G [00:05<00:00, 654MB/s]
100% 2.99G/2.99G [00:05<00:00, 580MB/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
```

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

# --- Step 2: Data Preparation ---

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

# For CoMoFoD: Tp = tampered, Au = authentic
# Let's assume 'F' in filename = forged/tampered (Tp), 'O' or 'M' = original/authentic (Au)
class_map = {'Tp': lambda f: '_F_' in f or '_F.' in f or f.startswith('F_') or '_F' in f,
             'Au': lambda f: '_O_' in f or '_O.' in f or f.startswith('O_') or '_O' in f or '_M' in f or '_M.' in f or f.startswith('M')}

class_names = ['Tp', 'Au']

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 class_map['Tp'](f)]
au_files = [f for f in all_files if class_map['Au'](f)]

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

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

splits = {'train': {}, 'val': {}, 'test': {}}

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 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 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 VGG-19

```
import torch
from torchvision import datasets, transforms

IMG_SIZE = 224  # Standard for VGG-19
BATCH_SIZE = 16

# Data transforms (VGG-19 uses ImageNet stats)
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])
    ]),
}

# ImageFolder expects: split_dir/train/Tp, split_dir/train/Au, etc.
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
```

Step 4: Model Setup – VGG-19 for binary classification

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

# Load pre-trained VGG-19
model = models.vgg19(pretrained=True)

# Modify classifier for 2 output classes (Tp, Au)
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, 2)  # 2 output classes

# Move model to device (GPU if available)
model = model.to(device)

print("VGG-19 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/vgg19-dcbb9e9d.pth" to /root/.cache/torch/hub/checkpoints/vgg19-dcbb9e9d.pth
100%|██████████| 548M/548M [00:02<00:00, 217MB/s]
VGG-19 binary classification model ready.
```

Loss, Optimizer, Scheduler setup

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

# Loss function
criterion = nn.CrossEntropyLoss()

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

# Learning rate scheduler (helps reduce LR after a few epochs)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

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

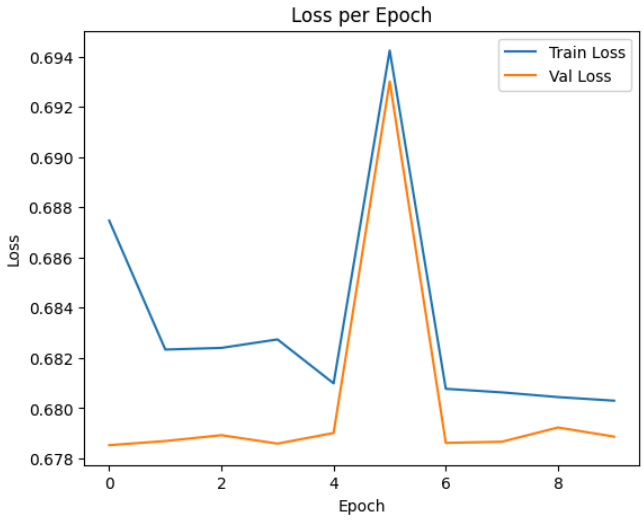
```
Loss, optimizer, and scheduler set up.
```

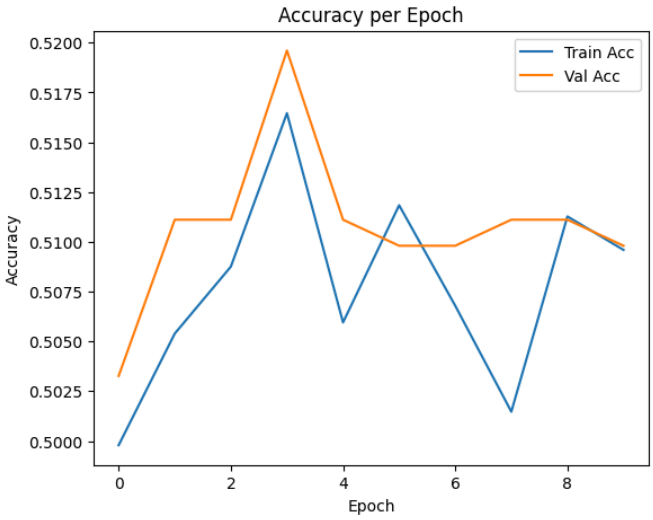
Training Loop with Learning Curves

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

num_epochs = 10  # Adjust for your experiments

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):
```

```
    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: 0.6875 Acc: 0.4998
Val Loss: 0.6785 Acc: 0.5033

Epoch 2/10
--------------------
Train Loss: 0.6823 Acc: 0.5054
Val Loss: 0.6787 Acc: 0.5111

Epoch 3/10
--------------------
Train Loss: 0.6824 Acc: 0.5088
Val Loss: 0.6789 Acc: 0.5111

Epoch 4/10
--------------------
Train Loss: 0.6827 Acc: 0.5165
Val Loss: 0.6786 Acc: 0.5196

Epoch 5/10
--------------------
Train Loss: 0.6810 Acc: 0.5060
Val Loss: 0.6790 Acc: 0.5111

Epoch 6/10
--------------------
Train Loss: 0.6942 Acc: 0.5118
Val Loss: 0.6930 Acc: 0.5098

Epoch 7/10
--------------------
Train Loss: 0.6808 Acc: 0.5068
Val Loss: 0.6786 Acc: 0.5098

Epoch 8/10
--------------------
Train Loss: 0.6806 Acc: 0.5015
Val Loss: 0.6787 Acc: 0.5111

Epoch 9/10
--------------------
Train Loss: 0.6804 Acc: 0.5113
Val Loss: 0.6792 Acc: 0.5111

Epoch 10/10
--------------------
Train Loss: 0.6803 Acc: 0.5096
Val Loss: 0.6789 Acc: 0.5098
```





Epoch-wise Training/Validation History:

	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	1	0.687471	0.499790	0.678528	0.503268
1	2	0.682335	0.505393	0.678693	0.511111
2	3	0.682402	0.508755	0.678925	0.511111
3	4	0.682739	0.516459	0.678590	0.519608
4	5	0.680990	0.505953	0.679012	0.511111
5	6	0.694244	0.511836	0.693012	0.509804
6	7	0.680773	0.506794	0.678619	0.509804

7	8	0.680630	0.501471	0.678666	0.511111
8	9	0.680443	0.511276	0.679231	0.511111
9	10	0.680299	0.509595	0.678869	0.509804

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 96.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 100.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 69.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 42.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 3.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 105.4 MB/s eta 0:00:00
```

Full Metrics Table:

	Metric	Value
0	Accuracy	0.4977
1	Precision	0.4916
2	Recall	0.7453
3	F1 Score	0.5925
4	False Negatives	191
5	True Negatives	203
6	True Positives	559
7	False Positives	578
8	Parameter Count	139,578,434
9	FLOPs (MACs)	19,684,848,642
10	Inference Time (ms)	1.78
11	Mean IOU (%)	31.49
12	Mean Accuracy Diff	0.4854
13	p-value (t-test)	1.866e-48

```
Confusion Matrix (Rows: True, Columns: Predicted):
[[203 578]
 [101 559]]
```

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

LRP Explainability

```
# Step 8: LRP Explainability for VGG-19

!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
from torchvision import transforms
import io, base64
from IPython.display import display, HTML

# Parameters
IMG_SIZE = 224
N_GRID = 9
N_TABLE = 5
```

```
    tampered_test_dir = os.path.join(split_dir, 'test', 'tp')
    tampered_imgs = [f for f in os.listdir(tampered_test_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
    sample_files = tampered_imgs[:max(N_GRID, N_TABLE)]

    vgg_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 vgg_transform(img).unsqueeze(0).to(device)

    # Use the last conv layer for LRP (for VGG-19: features[36])
    lrp = LayerDeepLift(model, model.features[36])

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

    for fname in sample_files:
        img_path = os.path.join(tampered_test_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_vgg19_comofod.csv", index=False)
    print("LRP results (image name, score) saved to lrp_results_vgg19_comofod.csv")
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 5.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 68.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 107.7 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
```

LRP Overlays for Tampered Images (3x3 Grid)

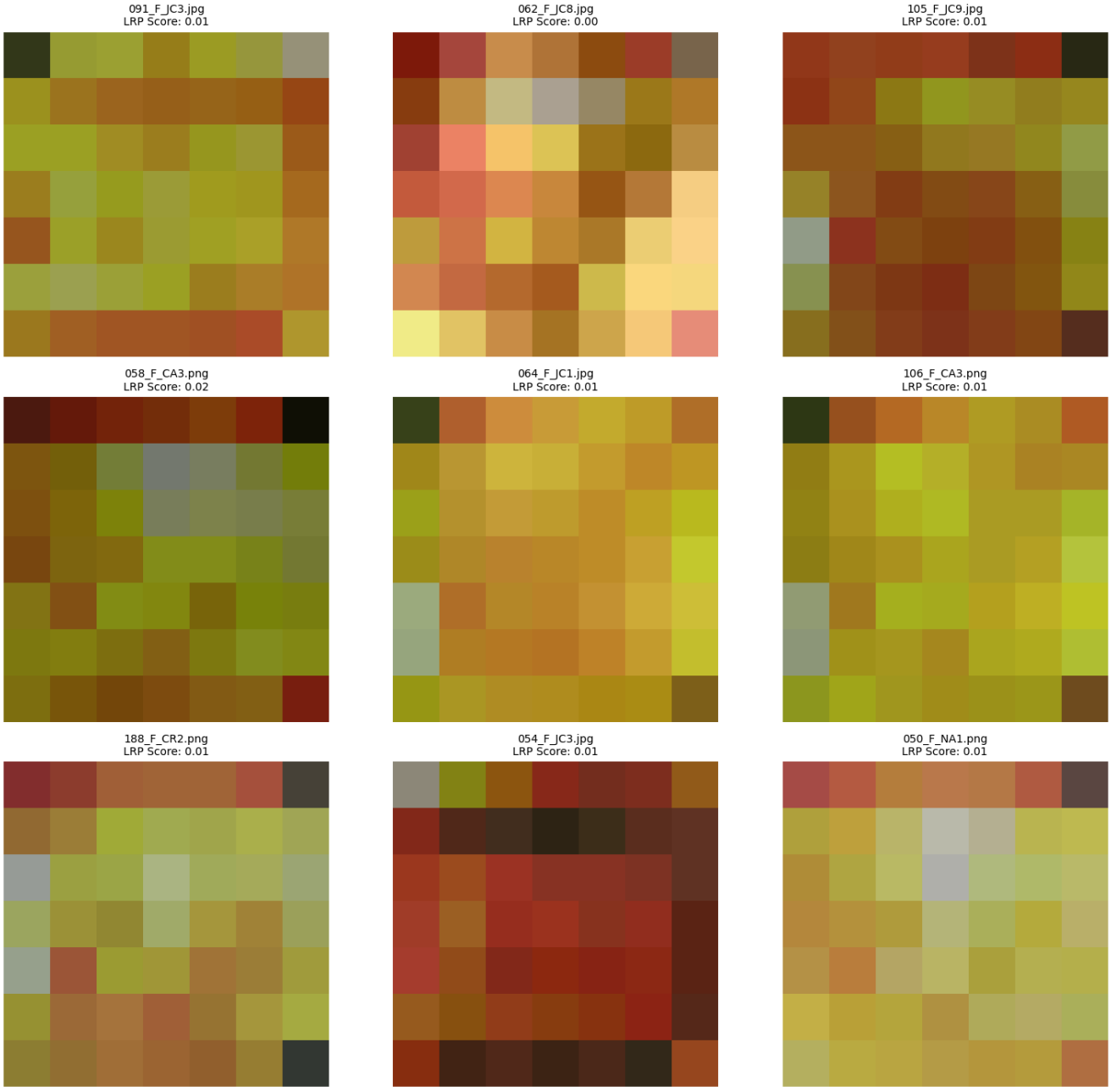
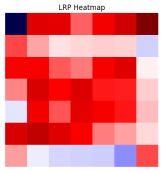
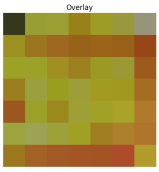
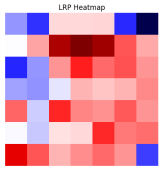
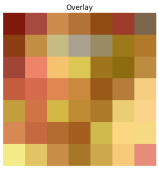
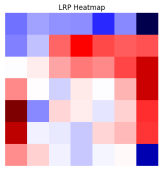
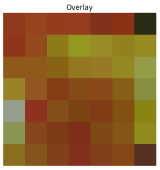
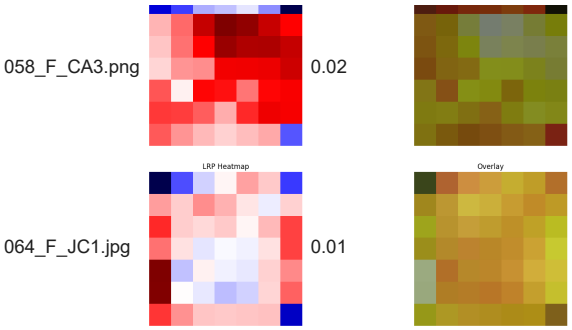


Image Name	LRP Heatmap	LRP Score	Overlay
091_F_JC3.jpg		0.01	
062_F_JC8.jpg		0.00	
105_F_JC9.jpg		0.01	




```
LRP results (image name, score) saved to lrp_results_vgg19_comofod.csv
```

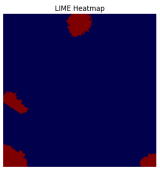
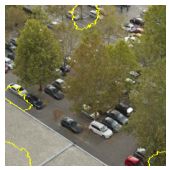
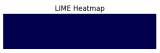
LIME Explainability

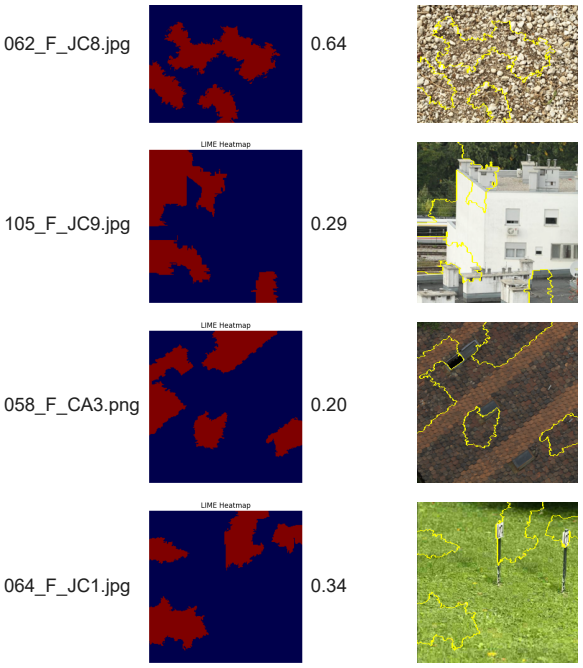
```
# Step 9: LIME Explainability for VGG-19

!pip install lime scikit-image --quiet

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

# Use same images as in LRP
N_GRID = 9
N_TABLE = 5
IMG_SIZE = 224
tampered_test_dir = os.path.join(split_dir, 'test', 'Tp')
sample_files = tampered_imgs[:max(N_GRID, N_TABLE)]

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

# LIME batch predict function (VGG-19 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

# Helpers for table
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"/>'

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

for fname in sample_files:
    img_path = os.path.join(tampered_test_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)
        img_names.append(fname)
        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 ---
    rows = []
    for i in range(N_TABLE):
        orig_pil = Image.fromarray(origs[i])
        # 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_vgg19_comofod.csv", index=False)
    print("LIME results (image name, score) saved to lime_results_vgg19_comofod.csv")
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 24.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
100%                                                  1000/1000 [00:03<00:00, 290.66it/s]
100%                                                  1000/1000 [00:03<00:00, 285.25it/s]
100%                                                  1000/1000 [00:03<00:00, 284.20it/s]
100%                                                  1000/1000 [00:03<00:00, 269.78it/s]
100%                                                  1000/1000 [00:03<00:00, 303.47it/s]
100%                                                  1000/1000 [00:03<00:00, 284.24it/s]
100%                                                  1000/1000 [00:03<00:00, 287.88it/s]
100%                                                  1000/1000 [00:03<00:00, 287.77it/s]
100%                                                  1000/1000 [00:03<00:00, 288.96it/s]
```

LIME Overlays for Tampered Images (3x3 Grid)

Image Name	LIME Heatmap	LIME Score	Overlay
091_F_JC3.jpg		0.18	
			

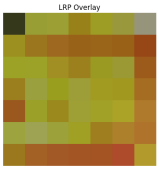
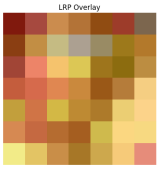
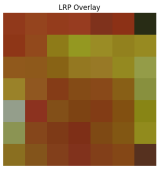
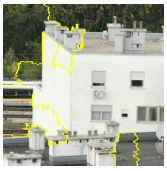
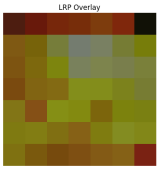
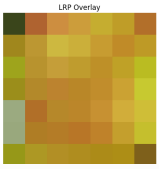



```
LIME results (image name, score) saved to lime_results_vgg19_comofod.csv
```

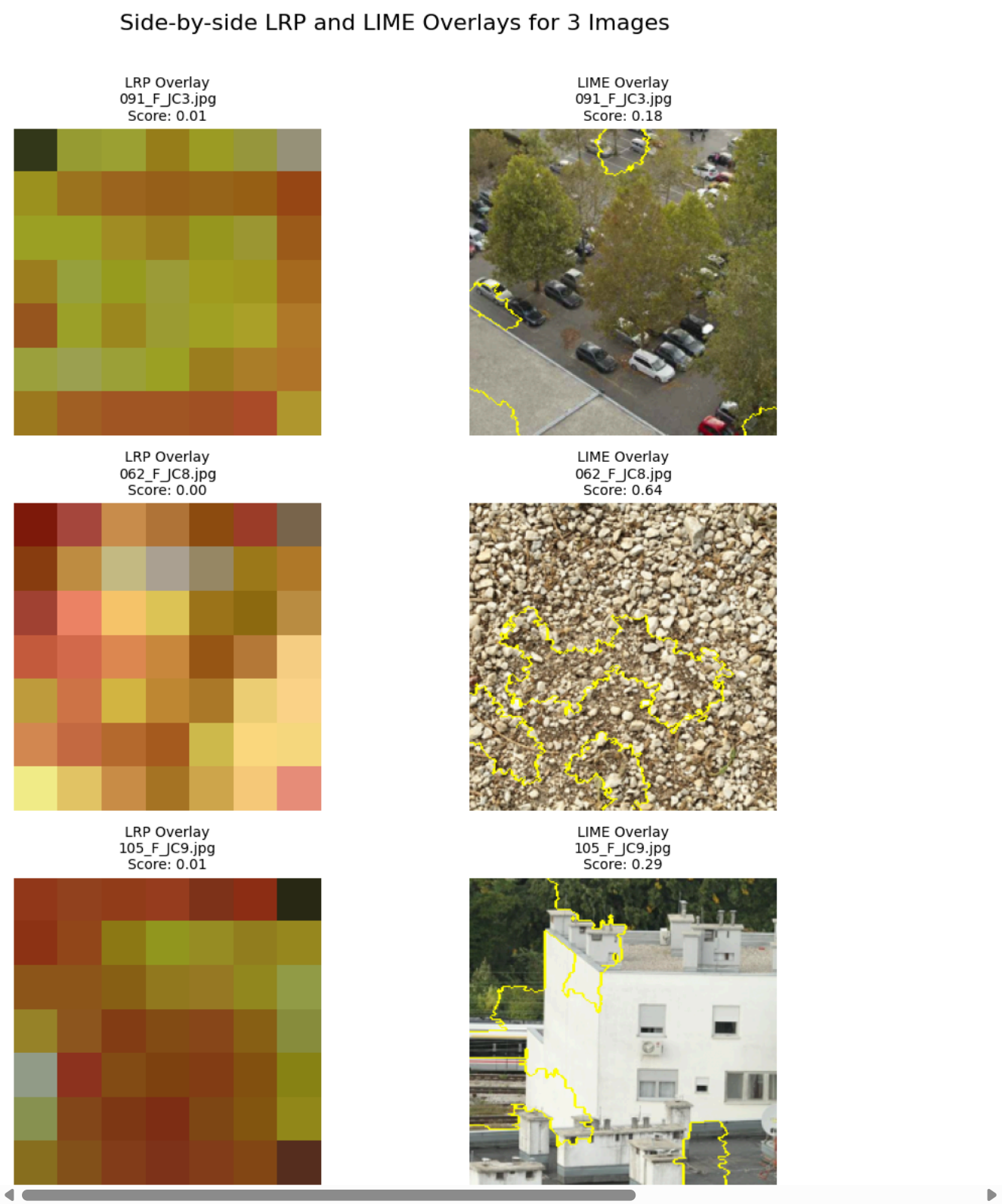
Displaying LIME & LRP overlays side-by-side

```
import io, base64
from PIL import Image
from IPython.display import display, HTML

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"/>'

N_TABLE = 5
rows = []
for i in range(N_TABLE):
    # LRP overlay
    lrp_img_disp, lrp_attr_norm = overlays[i]
    fig_lrp, ax_lrp = plt.subplots()
    ax_lrp.imshow(lrp_img_disp)
    ax_lrp.imshow(lrp_attr_norm, cmap='hot', alpha=0.4)
    ax_lrp.axis('off')
    ax_lrp.set_title('LRP Overlay')
    lrp_overlay_img = fig2img(fig_lrp)
    plt.close(fig_lrp)
    # LIME overlay
    lime_overlay_pil = Image.fromarray((overlays_lime[i]*255).astype(np.uint8))
    # Row
    row_html = f"""
    <tr>
        <td>{img_names[i]}</td>
        <td>{lrp_scores[i]:.2f}</td>
        <td>{img_to_html(lrp_overlay_img)}</td>
        <td>{lime_scores[i]:.2f}</td>
        <td>{img_to_html(lime_overlay_pil)}</td>
    </tr>
    """
    rows.append(row_html)

table_html = f"""
```

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

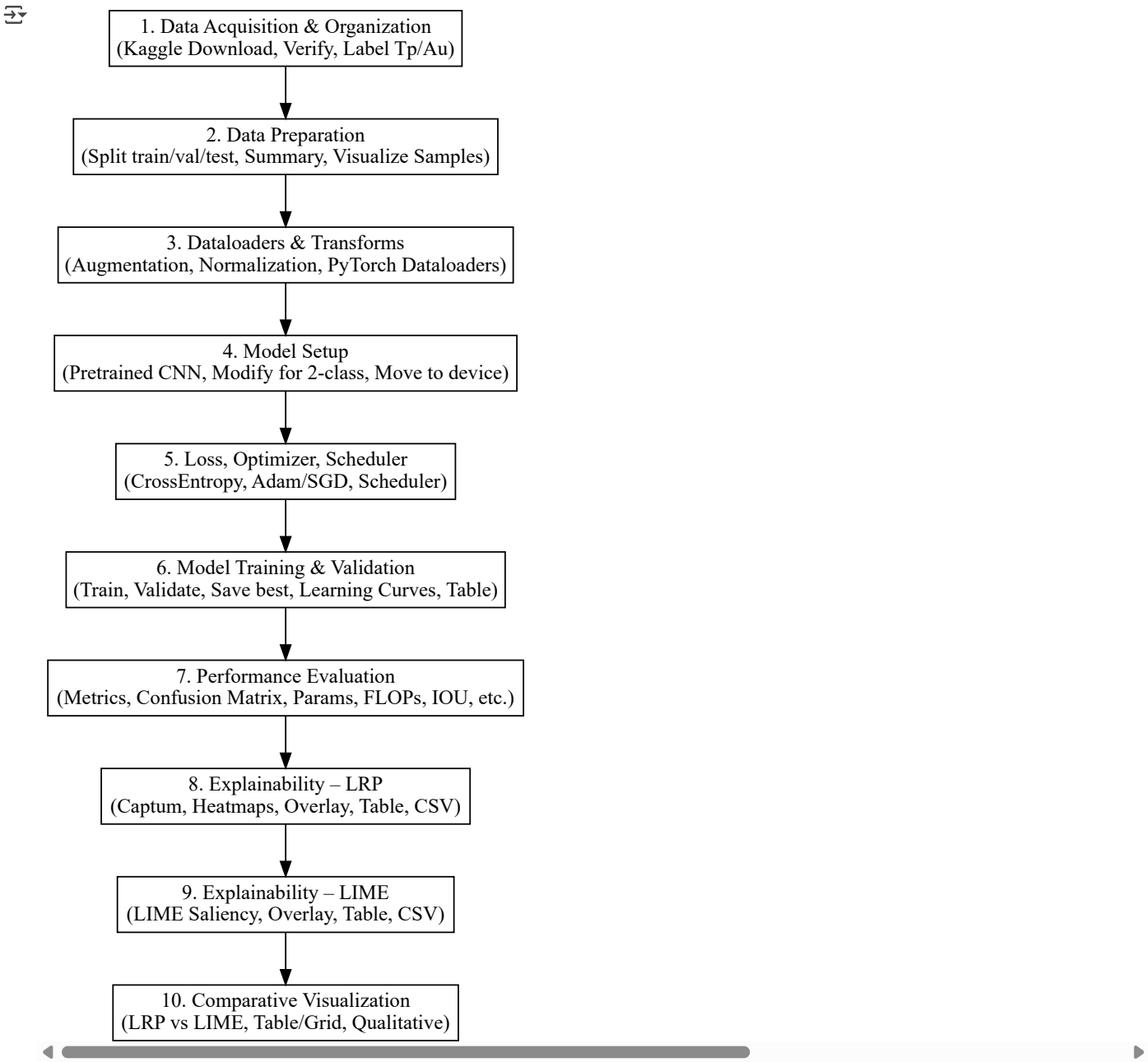
Image Name	LRP Score	LRP Overlay	LIME Score	LIME Overlay
091_F_JC3.jpg	0.01		0.18	
062_F_JC8.jpg	0.00		0.64	
105_F_JC9.jpg	0.01		0.29	
058_F_CA3.png	0.02		0.20	
064_F_JC1.jpg	0.01		0.34	

```
N_GRID = 3  # Change to 5/9 if you have more overlays

fig, axs = plt.subplots(N_GRID, 2, figsize=(10, 4*N_GRID))

for i in range(N_GRID):
    # LRP overlay
    lrp_img_disp, lrp_attr_norm = overlays[i]
    axs[i, 0].imshow(lrp_img_disp)
    axs[i, 0].imshow(lrp_attr_norm, cmap='hot', alpha=0.4)
    axs[i, 0].set_title(f"LRP Overlay\n{img_names[i]}\nScore: {lrp_scores[i]:.2f}", fontsize=10)
    axs[i, 0].axis('off')
    # LIME overlay
    axs[i, 1].imshow(overlays_lime[i])
    axs[i, 1].set_title(f"LIME Overlay\n{img_names[i]}\nScore: {lime_scores[i]:.2f}", fontsize=10)
    axs[i, 1].axis('off')

plt.suptitle(f"Side-by-side LRP and LIME Overlays for {N_GRID} Images", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```



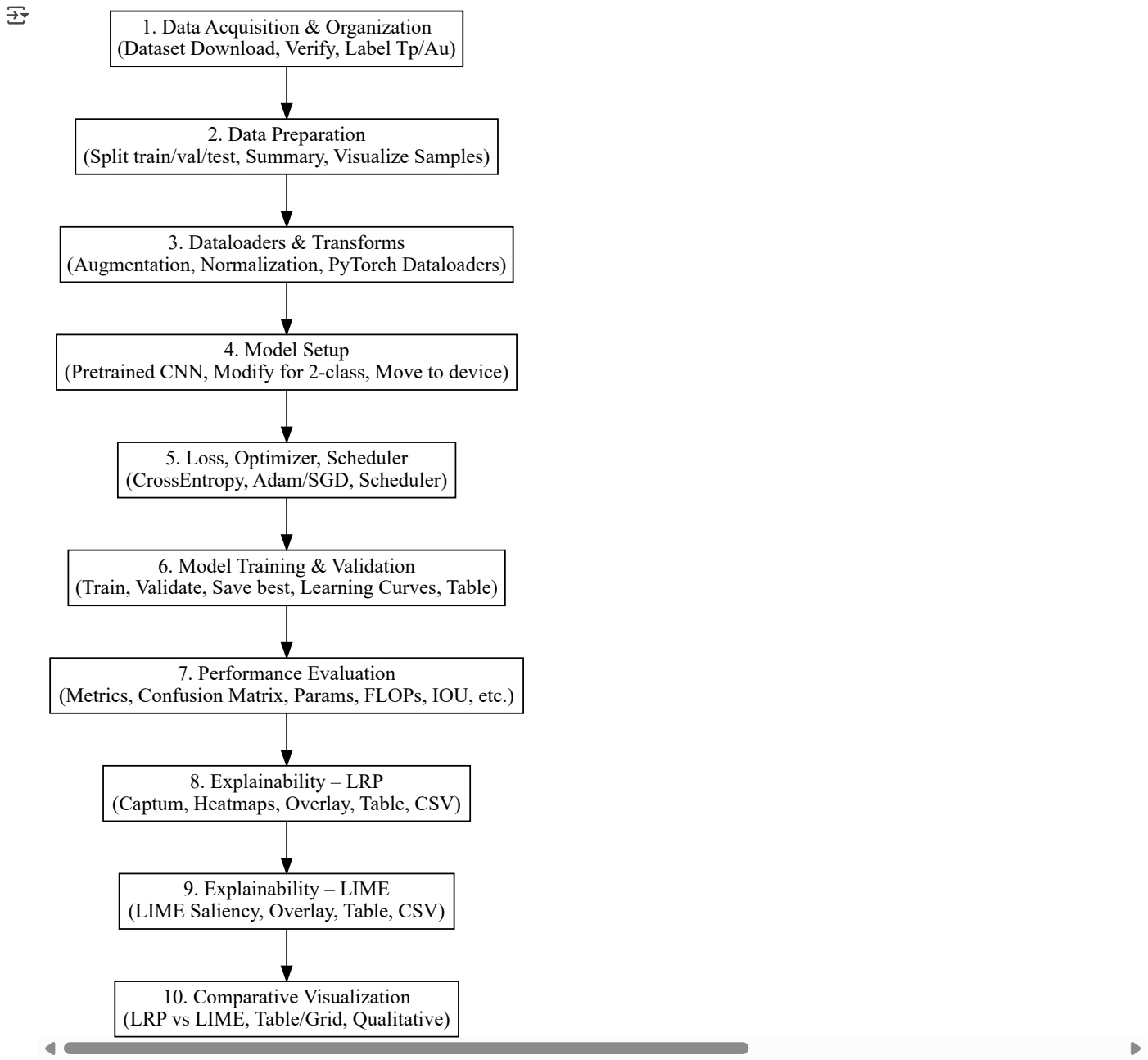

```
# 1. Install graphviz (if not already done)
!pip install graphviz --quiet

# 2. Import Digraph from graphviz
from graphviz import Digraph

# 3. Now build the diagram as before
dot = Digraph(comment='Forgery Detection & XAI Pipeline', format='png')
dot.attr(rankdir='TB', size='8,12')

# Main steps as nodes
dot.node('A', '1. Data Acquisition & Organization\n(Kaggle Download, Verify, Label Tp/Au)', shape='box')
dot.node('B', '2. Data Preparation\n(Split train/val/test, Summary, Visualize Samples)', shape='box')
dot.node('C', '3. Dataloaders & Transforms\n(Augmentation, Normalization, PyTorch Dataloaders)', shape='box')
dot.node('D', '4. Model Setup\n(Pretrained CNN, Modify for 2-class, Move to device)', shape='box')
dot.node('E', '5. Loss, Optimizer, Scheduler\n(CrossEntropy, Adam/SGD, Scheduler)', shape='box')
dot.node('F', '6. Model Training & Validation\n(Train, Validate, Save best, Learning Curves, Table)', shape='box')
dot.node('G', '7. Performance Evaluation\n(Metrics, Confusion Matrix, Params, FLOPs, IOU, etc.)', shape='box')
dot.node('H', '8. Explainability: LRP\n(Captum, Heatmaps, Overlay, Table, CSV)', shape='box')
```

```
dot.node('H', '8. Explainability – LRP\n(Captum, Heatmaps, Overlay, Table, CSV)', shape='box')
dot.node('I', '9. Explainability – LIME\n(LIME Saliency, Overlay, Table, CSV)', shape='box')
dot.node('J', '10. Comparative Visualization\n(LRP vs LIME, Table/Grid, Qualitative)', shape='box')

# Arrows between steps
dot.edges(['AB', 'BC', 'CD', 'DE', 'EF', 'FG', 'GH', 'HI', 'IJ'])

# Render (creates forgery_pipeline.png in the current directory)
dot.render('forgery_pipeline', view=True)
display(dot)
```




```
# 1. Install graphviz (if not already done)
!pip install graphviz --quiet

# 2. Import Digraph from graphviz
from graphviz import Digraph

# 3. Now build the diagram with updated label
dot = Digraph(comment='Forgery Detection & XAI Pipeline', format='png')
dot.attr(rankdir='TB', size='8,12')

# Main steps as nodes (updated wording for node A)
dot.node('A', '1. Data Acquisition & Organization\n(Dataset Download, Verify, Label Tp/Au)', shape='box')
dot.node('B', '2. Data Preparation\n(Split train/val/test, Summary, Visualize Samples)', shape='box')
dot.node('C', '3. Dataloaders & Transforms\n(Augmentation, Normalization, PyTorch Dataloaders)', shape='box')
dot.node('D', '4. Model Setup\n(Pretrained CNN, Modify for 2-class, Move to device)', shape='box')
dot.node('E', '5. Loss, Optimizer, Scheduler\n(CrossEntropy, Adam/SGD, Scheduler)', shape='box')
dot.node('F', '6. Model Training & Validation\n(Train, Validate, Save best, Learning Curves, Table)', shape='box')
dot.node('G', '7. Performance Evaluation\n(Metrics, Confusion Matrix, Params, FLOPs, IOU, etc.)', shape='box')
dot.node('H', '8. Explainability – LRP\n(Captum, Heatmaps, Overlay, Table, CSV)', shape='box')
```

```
dot.node('I', '9. Explainability – LIME\n(LIME Saliency, Overlay, Table, CSV)', shape='box')
dot.node('J', '10. Comparative Visualization\n(LRP vs LIME, Table/Grid, Qualitative)', shape='box')

# Arrows between steps
dot.edges(['AB', 'BC', 'CD', 'DE', 'EF', 'FG', 'GH', 'HI', 'IJ'])

# Render (creates forgery_pipeline.png in the current directory)
dot.render('forgery_pipeline', view=True)
display(dot)
```



Displaying the architecture (layer-by-layer)

```
import torch
from torchsummary import summary

# Make sure your model is on the correct device
model = model.to(device)

# Print full model architecture
print(model)

# Print summary: number of parameters, layer shapes, etc.
# For most pretrained CNNs, input size is (3, 224, 224)
try:
    # If torchsummary not installed:
    # !pip install torchsummary
    summary(model, input_size=(3, 224, 224), device=str(device))
except Exception as e:
    print("If torchsummary isn't installed, run: !pip install torchsummary")
```

```
print(e)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
           Conv2d-17          [-1, 256, 56, 56]         590,080
             ReLU-18          [-1, 256, 56, 56]               0
        MaxPool2d-19          [-1, 256, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       1,180,160
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
           Conv2d-24          [-1, 512, 28, 28]       2,359,808
             ReLU-25          [-1, 512, 28, 28]               0
           Conv2d-26          [-1, 512, 28, 28]       2,359,808
             ReLU-27          [-1, 512, 28, 28]               0
        MaxPool2d-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
           Conv2d-31          [-1, 512, 14, 14]       2,359,808
             ReLU-32          [-1, 512, 14, 14]               0
           Conv2d-33          [-1, 512, 14, 14]       2,359,808
             ReLU-34          [-1, 512, 14, 14]               0
           Conv2d-35          [-1, 512, 14, 14]       2,359,808
             ReLU-36          [-1, 512, 14, 14]               0
        MaxPool2d-37            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-38            [-1, 512, 7, 7]               0
           Linear-39                 [-1, 4096]     102,764,544
             ReLU-40                 [-1, 4096]               0
          Dropout-41                 [-1, 4096]               0
           Linear-42                 [-1, 4096]      16,781,312
             ReLU-43                 [-1, 4096]               0
          Dropout-44                 [-1, 4096]               0
           Linear-45                    [-1, 2]           8,194
================================================================
Total params: 139,578,434
Trainable params: 139,578,434
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 238.68
Params size (MB): 532.45
Estimated Total Size (MB): 771.70
----------------------------------------------------------------
```

Full architecture

```
print("Layer-wise names:")
for name, layer in model.named_children():
    print(f"{name}: {layer}")
```

```
Layer-wise names:
features: Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```