Downloading & Extraction CASIA_2 Dataset

```
# STEP 1: Download CASIA-2 from Kaggle in Google Colab

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 CASIA-2 dataset (about 3GB)
!kaggle datasets download divg07/casia-20-image-tampering-detection-dataset --unzip -p ./casia2

# Print the folder structure (top 2 levels)
print("\nCASIA-2 Dataset folder structure (showing top levels):")
for root, dirs, files in os.walk('./casia2'):
    level = root.replace('./casia2', '').count(os.sep)
    indent = ' ' * 4 * (level)
    print(f"{indent}{os.path.basename(root)}/")
    if level > 2:
        continue  # To avoid printing thousands of 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: 6/21/2025 - 100% done
Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/divg07/casia-20-image-tampering-detection-dataset
License(s): unknown
Downloading casia-20-image-tampering-detection-dataset.zip to ./casia2
100% 2.54G/2.56G [00:06<00:00, 309MB/s]
100% 2.56G/2.56G [00:06<00:00, 394MB/s]

CASIA-2 Dataset folder structure (showing top levels):
casia2/
    CASIA2/
        Tp/
            Tp_S_NNN_S_N_pla20052_pla20052_01952.tif
            Tp_D_NNN_S_N_nat00059_nat00059_00666.tif
            Tp_S_NNN_S_B_arc20092_arc20092_02407.tif
            Tp_D_NRN_M_N_nat10143_nat00095_12035.jpg
            Tp_S_NNN_S_N_arc10001_arc10001_20012.jpg
        CASIA 2 Groundtruth/
            Tp_S_NRN_S_N_arc00010_arc00010_01109_gt.png
            Tp_S_NNN_S_N_ind00044_ind00044_01334_gt.png
            Tp_D_NRD_S_N_ani00041_ani00040_00161_gt.png
            Tp_D_NRN_S_N_cha00035_cha00040_00355_gt.png
            Tp_S_NNN_S_N_arc20095_arc20095_02409_gt.png
        Au/
            Au_sec_30533.jpg
            Au_art_30093.jpg
            Au_pla_30586.jpg
            Au_art_30237.jpg
```

Data Preparation

```
import os
import random
import shutil
from collections import Counter

# Define paths
base_dir = 'casia2/CASIA2'
split_dir = 'casia2/split'
class_names = ['Tp', 'Au']

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

# Create split directories
for split in ['train', 'val', 'test']:
```

```
    for cls in class_names:
        os.makedirs(os.path.join(split_dir, split, cls), exist_ok=True)

# Split and copy images
split_counts = {split: Counter() for split in ['train', 'val', 'test']}

for cls in class_names:
    img_dir = os.path.join(base_dir, cls)
    img_files = [f for f in os.listdir(img_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png', '.tif'))]
    random.shuffle(img_files)
    n_total = len(img_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', img_files[:n_train]),
        ('val', img_files[n_train:n_train+n_val]),
        ('test', img_files[n_train+n_val:])
    ]

    for split, files in splits:
        for f in files:
            src = os.path.join(img_dir, f)
            dst = os.path.join(split_dir, split, cls, f)
            shutil.copy2(src, dst)
            split_counts[split][cls] += 1

# Print summary table
import pandas as pd
summary_df = pd.DataFrame(split_counts).T
summary_df.columns = ['Tampered (Tp)', 'Authentic (Au)']
summary_df['Total'] = summary_df['Tampered (Tp)'] + summary_df['Authentic (Au)']
print("\nDataset Split Summary:")
display(summary_df)

# Show some sample images
import matplotlib.pyplot as plt
from PIL import Image

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')
```

Dataset Split Summary:

	Tampered (Tp)	Authentic (Au)	Total
train	3586	5205	8791
val	768	1115	1883
test	769	1117	1886

Sample Tampered (Tp) images from train split:

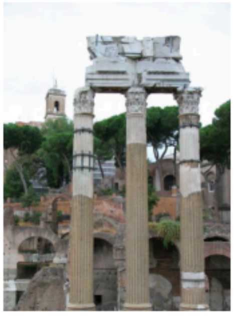
train/Tp
Tp_D_NRN_M_N_nat10116_cha00067_11374.jpg

train/Tp
Tp_S_NRN_S_N_pla20050_pla20050_0195

train/Tp
Tp_S_NRN_S_B_art10008_art10008_20049.jpg

Sample Authentic (Au) images from train split:

train/Au
Au_sec_20062.jpg



train/Au
Au_ani_20066.jpg

train/Au
Au_nat_30142.jpg

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

Dataloaders and Transforms for DenseNet-201.

```
import torch
from torchvision import datasets, transforms

IMG_SIZE = 224  # Standard input size for DenseNet-201
BATCH_SIZE = 16

# Define transforms (DenseNet-201 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 assumes structure: 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
```

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)

# Replace classifier for 2 output classes (Au, Tp)
num_ftrs = model.classifier.in_features
model.classifier = nn.Linear(num_ftrs, 2)  # 2 classes

# 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 for DenseNet-201

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

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer (Adam works well, you may also use SGD if preferred)
optimizer = optim.Adam(model.parameters(), lr=1e-4)

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

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

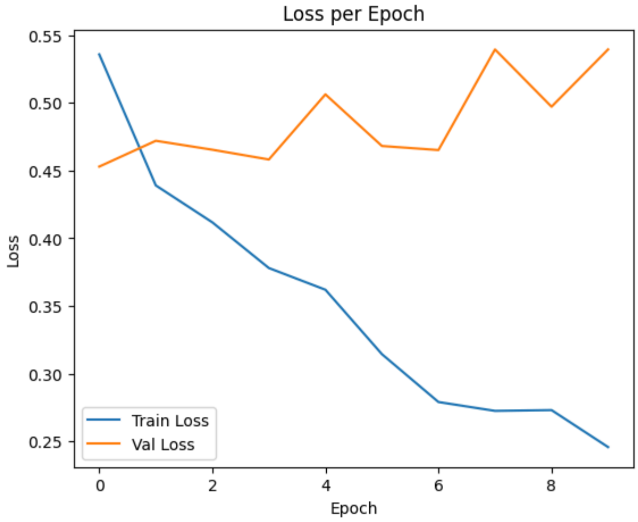
```
Loss, optimizer, and scheduler set up.
```

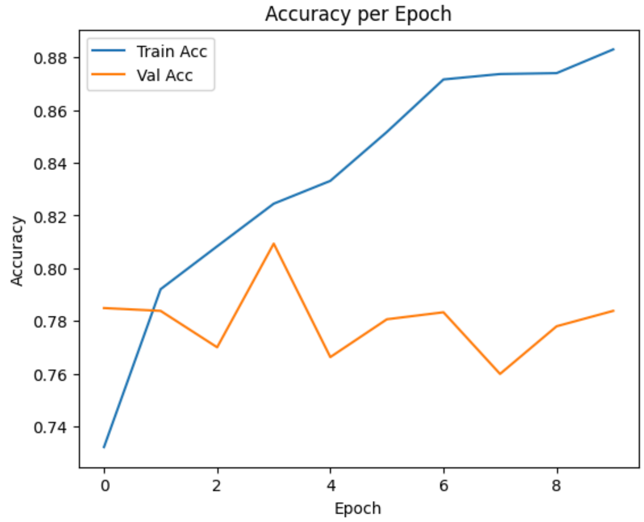
Training Loop with Learning Curves

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

num_epochs = 10  # You can increase this for longer training

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:")
    display(history_df)
```

```
Epoch 1/10
--------------------
Train Loss: 0.5359 Acc: 0.7322
Val Loss: 0.4530 Acc: 0.7849

Epoch 2/10
--------------------
Train Loss: 0.4391 Acc: 0.7921
Val Loss: 0.4720 Acc: 0.7839

Epoch 3/10
--------------------
Train Loss: 0.4118 Acc: 0.8083
Val Loss: 0.4654 Acc: 0.7700

Epoch 4/10
--------------------
Train Loss: 0.3780 Acc: 0.8245
Val Loss: 0.4582 Acc: 0.8093

Epoch 5/10
--------------------
Train Loss: 0.3620 Acc: 0.8331
Val Loss: 0.5063 Acc: 0.7663

Epoch 6/10
--------------------
Train Loss: 0.3144 Acc: 0.8517
Val Loss: 0.4682 Acc: 0.7807

Epoch 7/10
--------------------
Train Loss: 0.2790 Acc: 0.8716
Val Loss: 0.4652 Acc: 0.7833

Epoch 8/10
--------------------
Train Loss: 0.2725 Acc: 0.8736
Val Loss: 0.5395 Acc: 0.7600

Epoch 9/10
--------------------
Train Loss: 0.2731 Acc: 0.8740
Val Loss: 0.4972 Acc: 0.7780

Epoch 10/10
--------------------
Train Loss: 0.2457 Acc: 0.8829
Val Loss: 0.5395 Acc: 0.7839
```





Epoch-wise Training/Validation History:

	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	1	0.535866	0.732226	0.452959	0.784918
1	2	0.439058	0.792060	0.472034	0.783856
2	3	0.411777	0.808327	0.465428	0.770048
3	4	0.378019	0.824480	0.458209	0.809347
4	5	0.361983	0.833125	0.506331	0.766330
5	6	0.314385	0.851666	0.468171	0.780669
6	7	0.279026	0.871573	0.465189	0.783324

7	8	0.272489	0.873621	0.539504	0.759958
8	9	0.273065	0.873962	0.497204	0.778014
9	10	0.245745	0.882948	0.539524	0.783856

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

```
from IPython.display import display  # Already in your code

# Already in your code:
print("\nEpoch-wise Training/Validation History:")
display(history_df)
```

```
Epoch-wise Training/Validation History:
```

	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	1	0.535866	0.732226	0.452959	0.784918
1	2	0.439058	0.792060	0.472034	0.783856
2	3	0.411777	0.808327	0.465428	0.770048
3	4	0.378019	0.824480	0.458209	0.809347
4	5	0.361983	0.833125	0.506331	0.766330
5	6	0.314385	0.851666	0.468171	0.780669
6	7	0.279026	0.871573	0.465189	0.783324
7	8	0.272489	0.873621	0.539504	0.759958
8	9	0.273065	0.873962	0.497204	0.778014
9	10	0.245745	0.882948	0.539524	0.783856

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

```
history_df.to_csv('training_history.csv', index=False)
print("Training history saved as training_history.csv")
```

```
Training history saved as training_history.csv
```

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  # MACs ~= FLOPs for this purpose
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:")
display(metrics_table)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 129.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 99.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 61.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 42.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 3.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 106.2 MB/s eta 0:00:00
```

Full Metrics Table:

	Metric	Value
0	Accuracy	0.8054
1	Precision	0.7348
2	Recall	0.8179
3	F1 Score	0.7742
4	False Negatives	140
5	True Negatives	890
6	True Positives	629
7	False Positives	227
8	Parameter Count	18,096,770
9	FLOPs (MACs)	4,391,432,450
10	Inference Time (ms)	32.24
11	Mean IOU (%)	66.98
12	Mean Accuracy Diff	0.0212
13	p-value (t-test)	0.00422

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

LRP Explainability

```
# Step 8: LRP Explainability for DenseNet-201

!pip install captum --quiet

from captum.attr import LayerDeepLift
import matplotlib.pyplot as plt
import numpy as np
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', '.tif'))]
sample_files = tampered_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 feature block
lrp = LayerDeepLift(model, model.features[-1])

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

for fname in sample_files:
    img_path = os.path.join(tampered_test_dir, fname)
```

```
        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:image/png;base64,{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 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.csv", index=False)
    print("LRP results (image name, score) saved to lrp_results.csv")
```

```
                                          61.0/61.0 kB 2.9 MB/s eta 0:00:00
                                        1.4/1.4 MB 26.4 MB/s eta 0:00:00
                                        18.3/18.3 MB 118.9 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.
```

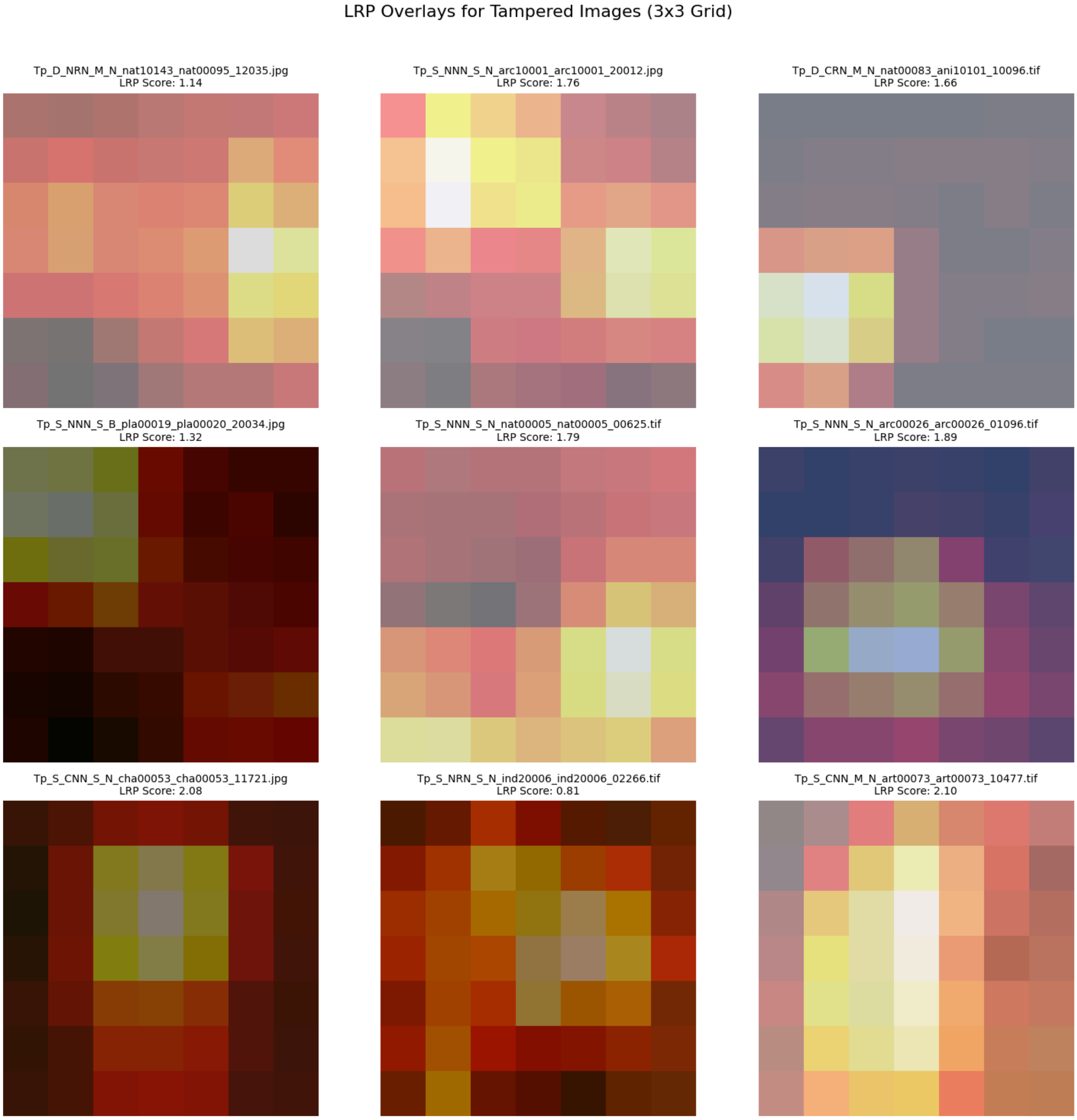
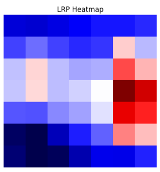
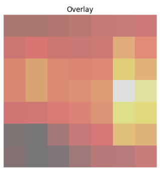
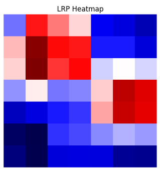
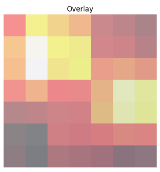
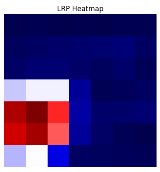
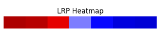
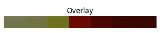
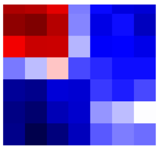
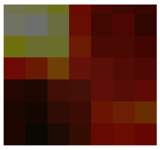
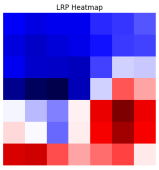
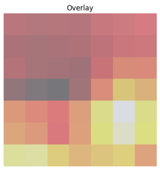


Image Name	LRP Heatmap	LRP Score	Overlay
Tp_D_NRN_M_N_nat10143_nat00095_12035.jpg		1.14	
Tp_S_NNN_S_N_arc10001_arc10001_20012.jpg		1.76	
Tp_D_CRN_M_N_nat00083_ani10101_10096.tif		1.66	
			

Tp_S_NNN_S_B_pla00019_pla00020_20034.jpg		1.32	
Tp_S_NNN_S_N_nat00005_nat00005_00625.tif		1.79	

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

LIME Explainability

```
# Step 9: LIME Explainability for DenseNet-201

!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 9 tampered images as 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 (DenseNet-201 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:image/png;base64,{data}" width="120"/>'

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

for fname in sample_files:
    img_path = os.path.join(tampered_test_dir, fname)
```