Downloading & Extracting 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 your kaggle.json here

# Move the file to the proper place
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download and unzip the CASIA-2 dataset (may take some time, ~3GB)
!kaggle datasets download divg07/casia-20-image-tampering-detection-dataset --unzip -p ./casia2

# Verify the folder structure
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
 98% 2.52G/2.56G [00:03<00:00, 676MB/s]
100% 2.56G/2.56G [00:03<00:00, 824MB/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 – Split into train, val, test

```
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:

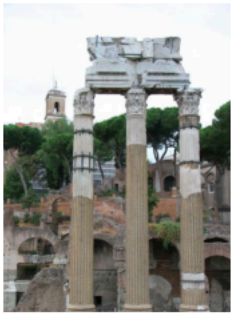
train/Tp
Tp_D_NRN_M_N_nat10116_cha00067_11374.jpg

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

VGG-19 Model Training

```
import torch
from torchvision import datasets, transforms

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

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])
    ]),
}

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)
```

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

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

model = models.vgg19_bn(pretrained=True)
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, 2)  # 2 classes: Tp, Au
model = model.to(device)
```

```
/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_bn-c79401a0.pth" to /root/.cache/torch/hub/checkpoints/vgg19_bn-c79401a0.ptl
100%|██████████| 548M/548M [00:03<00:00, 191MB/s]
```

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
```

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

num_epochs = 10

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

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())

model.load_state_dict(best_model_wts)
```

```
Epoch 1/10
--------------------
Train Loss: 0.5662 Acc: 0.7113
Val Loss: 0.5370 Acc: 0.7302

Epoch 2/10
--------------------
Train Loss: 0.4905 Acc: 0.7657
Val Loss: 0.4845 Acc: 0.7754

Epoch 3/10
--------------------
Train Loss: 0.4543 Acc: 0.7910
Val Loss: 0.5258 Acc: 0.7148

Epoch 4/10
--------------------
Train Loss: 0.4354 Acc: 0.7997
Val Loss: 0.5116 Acc: 0.7552

Epoch 5/10
--------------------
Train Loss: 0.4114 Acc: 0.8139
Val Loss: 0.4551 Acc: 0.7881

Epoch 6/10
--------------------
Train Loss: 0.3587 Acc: 0.8346
Val Loss: 0.4237 Acc: 0.8062

Epoch 7/10
--------------------
Train Loss: 0.3339 Acc: 0.8437
Val Loss: 0.4588 Acc: 0.7918

Epoch 8/10
--------------------
Train Loss: 0.3207 Acc: 0.8492
Val Loss: 0.4319 Acc: 0.8136

Epoch 9/10
--------------------
Train Loss: 0.3094 Acc: 0.8591
Val Loss: 0.4480 Acc: 0.8083

Epoch 10/10
--------------------
Train Loss: 0.2959 Acc: 0.8601
Val Loss: 0.5610 Acc: 0.7897
<All keys matched successfully>
```

```
import pandas as pd

history_table = []



# Log train/val for 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]
})



# Convert to DataFrame and show
history_df = pd.DataFrame(history_table)
print("\nEpoch-wise Training/Validation History:")
display(history_df)
```

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

	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	10	0.295886	0.860084	0.56097	0.789697

```
import pandas as pd
history_table = []

for epoch in range(num_epochs):
    # ... your existing loop code ...
    for phase in ['train', 'val']:
        # ... phase logic ...
        # [After computing epoch_loss and epoch_acc]
        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 as DataFrame
history_df = pd.DataFrame(history_table)
print("\nEpoch-wise Training/Validation History:")
display(history_df)
```

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

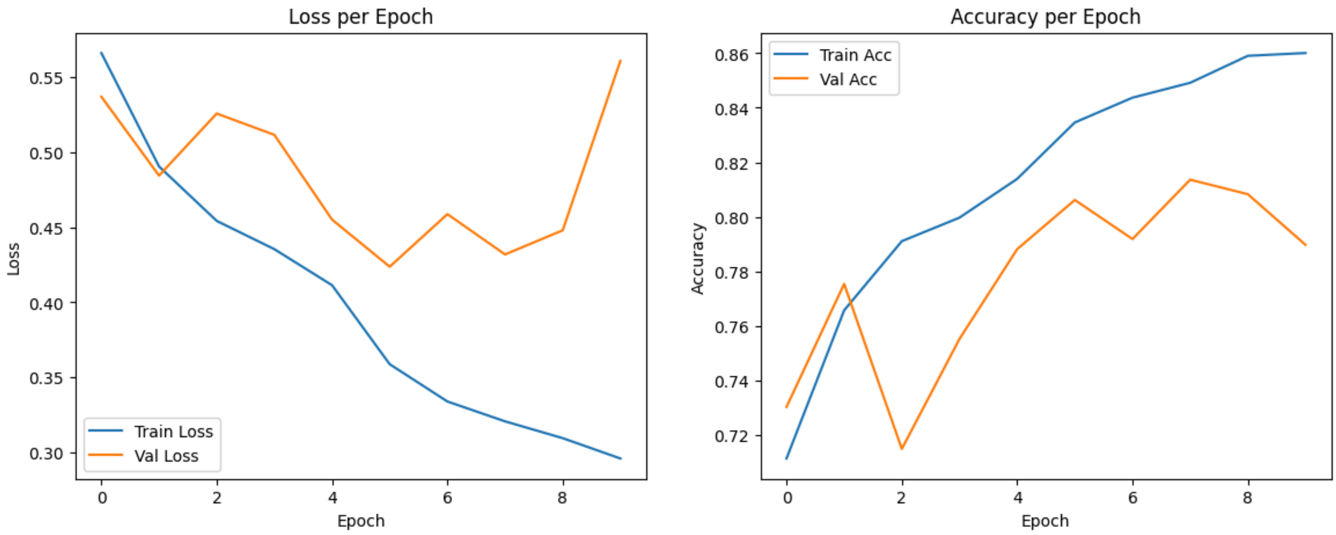
	Epoch	Train Loss	Train Acc	Val Loss	Val Acc
0	1	0.56097	0.789697	0.56097	0.789697
1	2	0.56097	0.789697	0.56097	0.789697
2	3	0.56097	0.789697	0.56097	0.789697
3	4	0.56097	0.789697	0.56097	0.789697
4	5	0.56097	0.789697	0.56097	0.789697
5	6	0.56097	0.789697	0.56097	0.789697
6	7	0.56097	0.789697	0.56097	0.789697
7	8	0.56097	0.789697	0.56097	0.789697
8	9	0.56097	0.789697	0.56097	0.789697
9	10	0.56097	0.789697	0.56097	0.789697

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

```
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()
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

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())

test_acc = accuracy_score(all_labels, all_preds)
test_prec = precision_score(all_labels, all_preds)
test_rec = recall_score(all_labels, all_preds)
test_f1 = f1_score(all_labels, all_preds)
cm = confusion_matrix(all_labels, all_preds)

print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Precision: {test_prec:.4f}")
print(f"Test Recall: {test_rec:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")
print("Confusion Matrix:\n", cm)
```

```
Test Accuracy: 0.8086
Test Precision: 0.7213
Test Recall: 0.8648
Test F1 Score: 0.7865
Confusion Matrix:
 [[860 257]
 [104 665]]
```

```
import pandas as pd

metrics_table = pd.DataFrame({
    "Metric": [
        "Test Accuracy",
        "Test Precision",
        "Test Recall",
        "Test F1 Score",
        "False Negatives",
        "True Negatives",
        "True Positives",
        "False Positives"
    ],
    "Value": [
        f"{test_acc:.4f}",
        f"{test_prec:.4f}",
        f"{test_rec:.4f}",
        f"{test_f1:.4f}",
        cm[1,0],    # False Negatives
        cm[0,0],    # True Negatives
```

```
        cm[1,1],     # True Positives
        cm[0,1]      # False Positives
    ]
})

print("\nTest Set Metrics Table:")
display(metrics_table)
```

Test Set Metrics Table:

	Metric	Value
0	Test Accuracy	0.8086
1	Test Precision	0.7213
2	Test Recall	0.8648
3	Test F1 Score	0.7865
4	False Negatives	104
5	True Negatives	860
6	True Positives	665
7	False Positives	257

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

```
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

# If not already defined:
# all_labels = ground truth labels (list)
# all_preds = predicted labels (list)

# Main 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]

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

# FLOPs calculation (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, IMG_SIZE, IMG_SIZE), as_strings=False, print_per_layer_stat=False)
    flops = macs  # MACs ~= FLOPs for this purpose
except Exception:
    flops = 'N/A (install ptflops)'

# Inference time (average per image)
n_runs = 50
sample = torch.rand(1, 3, IMG_SIZE, IMG_SIZE).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 difference (abs diff between classwise acc)
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", "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, 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 3.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 123.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 98.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 59.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 43.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 108.3 MB/s eta 0:00:00

Full Metrics Table:
```

	Metric	Value
0	Accuracy	0.8086
1	Precision	0.7213
2	Recall	0.8648
3	F1 Score	0.7865
4	False Negatives	104
5	Parameter Count	139,589,442
6	FLOPs (MACs)	19,714,552,834
7	Inference Time (ms)	2.84
8	Mean IOU (%)	67.62
9	Mean Accuracy Diff	0.0948
10	p-value (t-test)	5.278e-07

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

LRP explainability

```
!pip install captum --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 25.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 114.5 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 sou
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
```

```
from captum.attr import LayerDeepLift
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os

# Choose sample images: 9 from tampered test set
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)]

# Helper to preprocess image to tensor
from PIL import Image
from torchvision import transforms

vgg19_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 vgg19_transform(img).unsqueeze(0).to(device)

    # LRP via LayerDeepLift on the last conv layer
    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)
        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)

        # Overlay
        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 ---
    import io, base64
    from PIL import Image
    from IPython.display import display, HTML

    def arr_to_img(arr):
        arr = (arr * 255).astype(np.uint8)
        return Image.fromarray(arr)

    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):
        # Original
        img_disp, attr_norm = overlays[i]
        img_disp_255 = (img_disp * 255).astype(np.uint8)
        orig_pil = Image.fromarray(img_disp_255)
        # 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.csv", index=False)
    print("LRP results (image name, score) saved to lrp_results.csv")
```

LRP Overlays for Tampered Images (3x3 Grid)

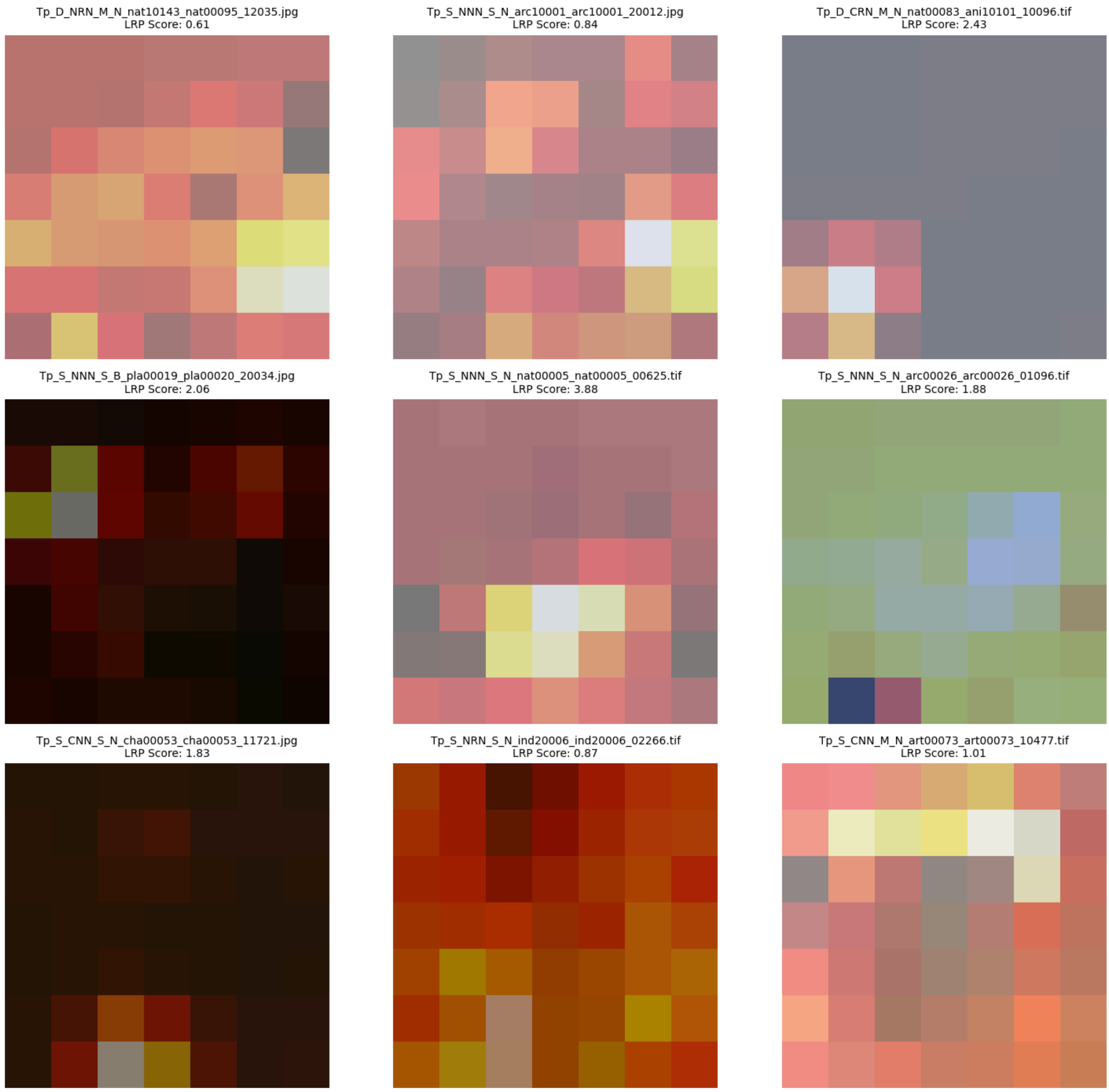
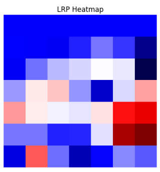
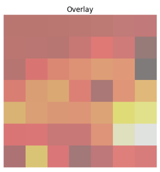
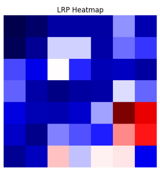
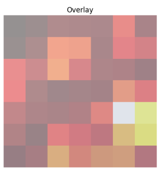
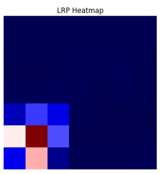
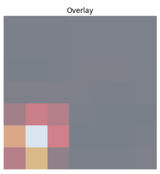
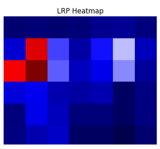
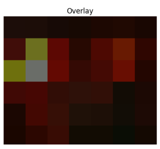
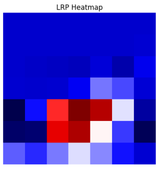


Image Name	LRP Heatmap	LRP Score	Overlay
Tp_D_NRN_M_N_nat10143_nat00095_12035.jpg		0.61	
Tp_S_NNN_S_N_arc10001_arc10001_20012.jpg		0.84	
Tp_D_CRN_M_N_nat00083_ani10101_10096.tif		2.43	
Tp_S_NNN_S_B_pla00019_pla00020_20034.jpg		2.06	


Tp_S_NNN_S_N_nat00005_nat00005_00625.tif



3.88


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

Explainability with LIME

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

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 6.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
```

```
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
import torch

# Select images: 9 tampered test images
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)]

# Transform for LIME and PIL open
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 requires a prediction function accepting (batch of images as numpy arrays)
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

# For display
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"/>'

# Run LIME, save results for grid/table
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)
```

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