Dataset Download & Extraction

```
# Step 1.1: Install Kaggle API (if not already installed)
!pip install -q kaggle

# Step 1.2: Upload your Kaggle JSON key
from google.colab import files
files.upload()  # Upload kaggle.json here

# Step 1.3: Move kaggle.json to the correct location
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Step 1.4: Download the MICC-F220 dataset from Kaggle
!kaggle datasets download -d mashraffarouk/micc-f220

# Step 1.5: Extract the downloaded dataset
!unzip -q micc-f220.zip -d micc_f220_dataset

# Step 1.6: Verify folder structure
import os

dataset_path = "/content/micc_f220_dataset/MICC-F220"
print("✅ MICC-F220 Dataset Directory Structure:")

for root, dirs, files in os.walk(dataset_path):
    level = root.replace(dataset_path, '').count(os.sep)
    indent = ' ' * 4 * level
    print(f"{indent}📁 {os.path.basename(root)}")
    subindent = ' ' * 4 * (level + 1)
    for f in files[:5]:  # Display only first 5 files for brevity
        print(f"{subindent}└── {f}")
```

```
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/mashraffarouk/micc-f220
License(s): unknown
Downloading micc-f220.zip to /content
  0% 0.00/14.4M [00:00<?, ?B/s]
100% 14.4M/14.4M [00:00<00:00, 517MB/s]
✅ MICC-F220 Dataset Directory Structure:
📁 MICC-F220
    └── groundtruthDB_220.txt
    📁 Tu
        └── sony_61tamp237.jpg
        └── DSCN45tamp133.jpg
        └── DSCF8tamp237.jpg
        └── sony_61tamp131.jpg
        └── DSC_0812tamp176.jpg
    📁 Au
        └── DSCN2326_scale.jpg
        └── DSC_1580_scale.jpg
        └── DSC_1540_scale.jpg
        └── CRW_4815_scale.jpg
        └── DSC_1542_scale.jpg
```

Data Preparation and Splitting (Train/Val/Test)

```
import os
import shutil
from sklearn.model_selection import train_test_split
import pandas as pd

# Original image directories
tampered_dir = "/content/micc_f220_dataset/MICC-F220/Tu"
authentic_dir = "/content/micc_f220_dataset/MICC-F220/Au"

# Destination base directory
output_base = "/content/micc_f220_prepared"
os.makedirs(output_base, exist_ok=True)

# Utility function to get valid image paths
def get_images(path):
    return sorted([
        os.path.join(path, f)
        for f in os.listdir(path)
        if f.lower().endswith(('.jpg', '.jpeg', '.png'))
```

```
    ])

# Gather image paths
tampered_images = get_images(tampered_dir)
authentic_images = get_images(authentic_dir)

# Function to split and copy images
def split_and_copy(images, label):
    train, test_val = train_test_split(images, test_size=0.3, random_state=42)
    val, test = train_test_split(test_val, test_size=0.5, random_state=42)

    for split, subset in zip(['train', 'val', 'test'], [train, val, test]):
        dest_dir = os.path.join(output_base, split, label)
        os.makedirs(dest_dir, exist_ok=True)
        for img in subset:
            shutil.copy(img, os.path.join(dest_dir, os.path.basename(img)))

    return len(train), len(val), len(test)

# Execute split and copy
tp_counts = split_and_copy(tampered_images, 'Tp')
au_counts = split_and_copy(authentic_images, 'Au')

# Summary table
df_summary = pd.DataFrame({
    'Split': ['Train', 'Validation', 'Test'],
    'Tampered (Tp)': tp_counts,
    'Authentic (Au)': au_counts
})
print("✅ MICC-F220 Dataset Split Summary:")
display(df_summary)
```

```
✅ MICC-F220 Dataset Split Summary:
```

	Split	Tampered (Tp)	Authentic (Au)
0	Train	77	77
1	Validation	16	16
2	Test	17	17

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

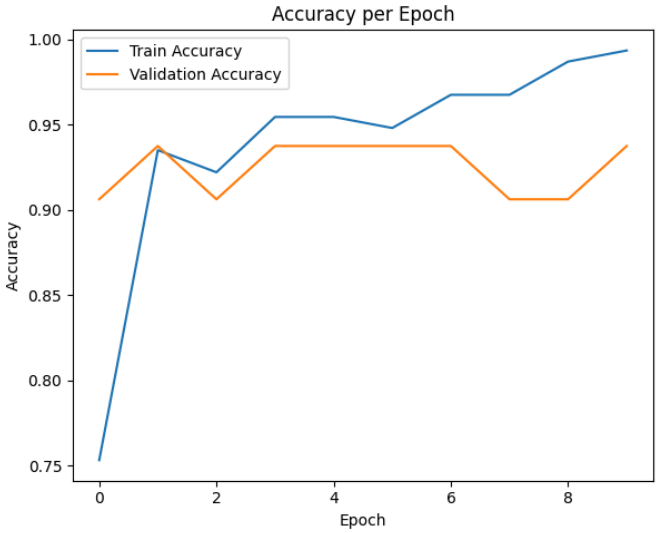
Model Training with DenseNet-201

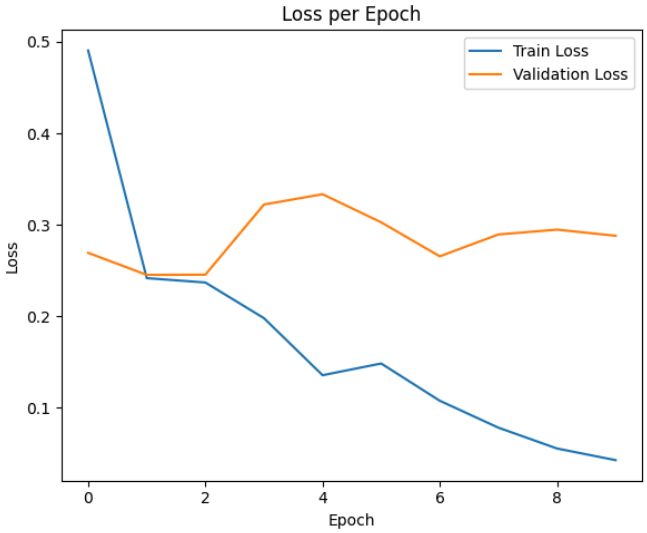
```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import pandas as pd
from time import time
from IPython.display import display

# Paths
data_dir = "/content/micc_f220_prepared"
train_dir = os.path.join(data_dir, "train")
val_dir = os.path.join(data_dir, "val")

# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Transforms
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
```

```
}

# Load Datasets
image_datasets = {
    'train': datasets.ImageFolder(train_dir, data_transforms['train']),
    'val': datasets.ImageFolder(val_dir, data_transforms['val'])
}
dataloaders = {
    x: DataLoader(image_datasets[x], batch_size=16, shuffle=True, num_workers=2)
    for x in ['train', 'val']
}
class_names = image_datasets['train'].classes
print("Classes:", class_names)

# Load DenseNet-201 and modify final layer
model = models.densenet201(weights='IMAGENET1K_V1')
model.classifier = nn.Linear(model.classifier.in_features, 2)  # Binary classification
model = model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

# Training loop
train_loss, val_loss = [], []
train_acc, val_acc = [], []
history = []

num_epochs = 10
start = time()

for epoch in range(num_epochs):
    print(f"\nEpoch {epoch+1}/{num_epochs}")
    for phase in ['train', 'val']:
        model.train() if phase == '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)
                loss = criterion(outputs, labels)
                preds = torch.argmax(outputs, 1)

                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])

        if phase == 'train':
            train_loss.append(epoch_loss)
            train_acc.append(epoch_acc.item())
        else:
            val_loss.append(epoch_loss)
            val_acc.append(epoch_acc.item())

        history.append({
            "epoch": epoch + 1,
            "phase": phase,
            "loss": epoch_loss,
            "accuracy": epoch_acc.item()
        })

        print(f"{phase.capitalize()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

print(f"\n✅ Training complete in {(time() - start):.2f} sec")
torch.save(model.state_dict(), "/content/densenet201_micc_f220.pth")
print("📦 Model saved to /content/densenet201_micc_f220.pth")

# Learning Curves
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_acc, label='Train Accuracy')
```

```
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Accuracy per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_loss, label='Train Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.savefig("/content/densenet201_learning_curves.png")
plt.show()

# Save training history
history_df = pd.DataFrame(history)
display(history_df)
```

```
Using device: cpu
Classes: ['Au', 'Tp']
Downloading: "https://download.pytorch.org/models/densenet201-c1103571.pth" to /root/.cache/torch/hub/checkpoints/densenet201-c11
100%|██████████| 77.4M/77.4M [00:00<00:00, 109MB/s]

Epoch 1/10
Train Loss: 0.4905 Acc: 0.7532
Val Loss: 0.2695 Acc: 0.9062

Epoch 2/10
Train Loss: 0.2418 Acc: 0.9351
Val Loss: 0.2454 Acc: 0.9375

Epoch 3/10
Train Loss: 0.2371 Acc: 0.9221
Val Loss: 0.2456 Acc: 0.9062

Epoch 4/10
Train Loss: 0.1981 Acc: 0.9545
Val Loss: 0.3222 Acc: 0.9375

Epoch 5/10
Train Loss: 0.1357 Acc: 0.9545
Val Loss: 0.3335 Acc: 0.9375

Epoch 6/10
Train Loss: 0.1486 Acc: 0.9481
Val Loss: 0.3028 Acc: 0.9375

Epoch 7/10
Train Loss: 0.1078 Acc: 0.9675
Val Loss: 0.2656 Acc: 0.9375

Epoch 8/10
Train Loss: 0.0784 Acc: 0.9675
Val Loss: 0.2896 Acc: 0.9062

Epoch 9/10
Train Loss: 0.0557 Acc: 0.9870
Val Loss: 0.2948 Acc: 0.9062

Epoch 10/10
Train Loss: 0.0430 Acc: 0.9935
Val Loss: 0.2881 Acc: 0.9375

✅ Training complete in 1970.30 sec
📦 Model saved to /content/densenet201_micc_f220.pth
```





	epoch	phase	loss	accuracy
0	1	train	0.490537	0.753247
1	1	val	0.269457	0.906250
2	2	train	0.241823	0.935065
3	2	val	0.245393	0.937500
4	3	train	0.237053	0.922078
5	3	val	0.245589	0.906250
6	4	train	0.198086	0.954545
7	4	val	0.322244	0.937500
8	5	train	0.135745	0.954545
9	5	val	0.333496	0.937500

10	6	train	0.148560	0.948052
11	6	val	0.302812	0.937500
12	7	train	0.107846	0.967532
13	7	val	0.265636	0.937500
14	8	train	0.078448	0.967532
15	8	val	0.289557	0.906250
16	9	train	0.055683	0.987013
17	9	val	0.294801	0.906250
18	10	train	0.042954	0.993506
19	10	val	0.288082	0.937500

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

Test Evaluation and Performance Metrics for DenseNet-201

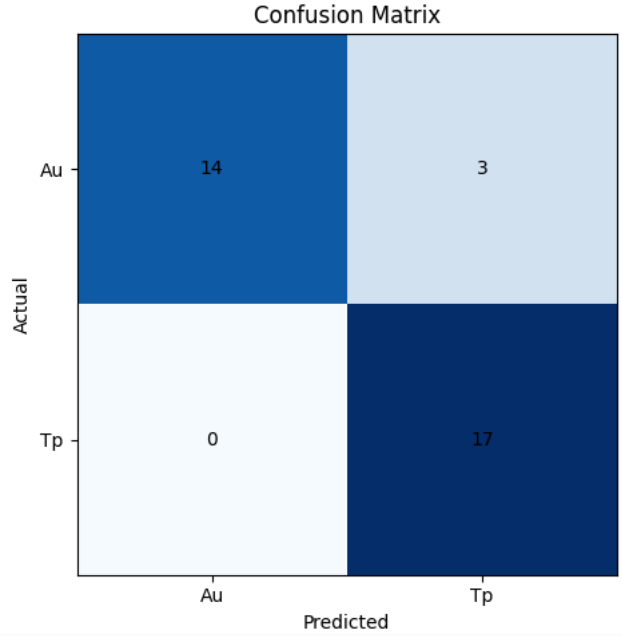
```
import torch
import torch.nn.functional as F
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
from sklearn.metrics import classification_report, confusion_matrix
from scipy.stats import ttest_ind
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display

# ✅ Paths
test_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"

# ✅ Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ✅ Transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# ✅ Test Dataset and Loader
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
class_names = test_dataset.classes

# ✅ Load Model
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# ✅ Inference
y_true, y_pred = [], []
inference_times = []
probabilities = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        start = time.time()
        outputs = model(inputs)
        end = time.time()

        probs = F.softmax(outputs, dim=1)
        preds = torch.argmax(probs, 1)

        y_true.extend(labels.cpu().numpy())
```

```
            y_pred.extend(preds.cpu().numpy())
            probabilities.extend(probs.cpu().numpy())

            inference_times.append(end - start)

    # ✅ Classification Metrics
    report = classification_report(y_true, y_pred, target_names=class_names, output_dict=True)
    conf_matrix = confusion_matrix(y_true, y_pred)
    false_negatives = conf_matrix[1][0]  # Tampered predicted as Authentic

    # ✅ Params
    total_params = sum(p.numel() for p in model.parameters())
    trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)

    # ✅ Inference Time
    avg_inference_time_ms = (sum(inference_times) / len(test_dataset)) * 1000

    # ✅ FLOPs (precomputed for DenseNet-201 224x224)
    flops = "4.3 GFLOPs"

    # ✅ IOU (confusion-matrix based approximation)
    intersection = np.diag(conf_matrix)
    ground_truth_set = conf_matrix.sum(axis=1)
    predicted_set = conf_matrix.sum(axis=0)
    union = ground_truth_set + predicted_set - intersection
    iou = np.mean(intersection / (union + 1e-6)) * 100

    # ✅ Confidence score t-test
    confidence_scores = [np.max(prob) for prob in probabilities]
    correct_confidences = [score for i, score in enumerate(confidence_scores) if y_pred[i] == y_true[i]]
    incorrect_confidences = [score for i, score in enumerate(confidence_scores) if y_pred[i] != y_true[i]]

    mean_accuracy_diff = round(np.mean(correct_confidences) - np.mean(incorrect_confidences), 4)
    t_stat, p_val = ttest_ind(correct_confidences, incorrect_confidences, equal_var=False)
    p_val_rounded = round(p_val, 6)

    # ✅ Summary Table
    summary = {
        "Accuracy": round(report["accuracy"], 4),
        "Precision (Tp)": round(report['Tp']['precision'], 4),
        "Recall (Tp)": round(report['Tp']['recall'], 4),
        "F1-Score (Tp)": round(report['Tp']['f1-score'], 4),
        "False Negatives": false_negatives,
        "Total Params": total_params,
        "Trainable Params": trainable_params,
        "Avg Inference Time (ms)": round(avg_inference_time_ms, 2),
        "FLOPs": flops,
        "Mean IOU (%)": round(iou, 2),
        "Mean Accuracy Diff": mean_accuracy_diff,
        "p-value (t-test)": p_val_rounded
    }

    summary_df = pd.DataFrame([summary])
    display(summary_df)

    # ✅ Full Classification Report
    report_df = pd.DataFrame(report).transpose().round(4)
    display(report_df)

    # ✅ Confusion Matrix Plot
    plt.figure(figsize=(6, 5))
    plt.imshow(conf_matrix, cmap='Blues')
    plt.title("Confusion Matrix")
    plt.xticks([0, 1], class_names)
    plt.yticks([0, 1], class_names)
    for i in range(2):
        for j in range(2):
            plt.text(j, i, conf_matrix[i, j], ha="center", va="center", color="black")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()
```

```
Using device: cpu
```

	Accuracy	Precision (Tp)	Recall (Tp)	F1-Score (Tp)	False Negatives	Total Params	Trainable Params	Avg Inference Time (ms)	FLOPs	Mean IOU (%)	Mean Accuracy Diff	p-value (t-test)
0	0.9118	0.85	1.0	0.9189	0	18096770	18096770	327.9	4.3 GFLOPs	83.68	0.0285	0.684494

	precision	recall	f1-score	support
Au	1.0000	0.8235	0.9032	17.0000
Tp	0.8500	1.0000	0.9189	17.0000
accuracy	0.9118	0.9118	0.9118	0.9118
macro avg	0.9250	0.9118	0.9111	34.0000
weighted avg	0.9250	0.9118	0.9111	34.0000



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

LRP Explainability using DenseNet-201 on MICC-F220

```
!pip install captum
```

```
Requirement already satisfied: captum in /usr/local/lib/python3.11/dist-packages (0.8.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from captum) (3.10.0)
Requirement already satisfied: numpy<2.0 in /usr/local/lib/python3.11/dist-packages (from captum) (1.26.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from captum) (24.2)
Requirement already satisfied: torch>=1.10 in /usr/local/lib/python3.11/dist-packages (from captum) (2.6.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from captum) (4.67.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (4.14
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->capt
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (9
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (11
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2.21
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (12
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.10->captu
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (3.2.3)
```

```
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (2.9.0.post
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->captum)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.10->captum) (3.0.2
```

```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
import pandas as pd
from tqdm import tqdm
from IPython.display import display, HTML

# ✅ Paths
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"
output_dir = "/content/lrp_densenet201_results"
os.makedirs(output_dir, exist_ok=True)

# ✅ Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ✅ Load DenseNet-201
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# ✅ Set up LRP
lrp = LRP(model)

# ✅ Image transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# ✅ Get 30 test image paths (15 Tp + 15 Au)
image_paths = []
for label in ["Tp", "Au"]:
    label_dir = os.path.join(test_images_dir, label)
    image_files = sorted([f for f in os.listdir(label_dir) if f.lower().endswith(('.jpg', '.png'))])[:15]
    image_paths.extend([(os.path.join(label_dir, f), label) for f in image_files])

log_rows = []

# ✅ Process images
for img_path, label in tqdm(image_paths, desc="Generating LRP Visuals"):
    filename = os.path.basename(img_path)
    orig = Image.open(img_path).convert("RGB")
    input_tensor = transform(orig).unsqueeze(0).to(device)
    input_tensor.requires_grad_()

    # Predict
    output = model(input_tensor)
    pred_class = output.argmax(dim=1).item()

    # LRP attribution
    attr = lrp.attribute(input_tensor, target=pred_class)
    attr = attr.squeeze().detach().cpu().numpy()
    relevance_score = float(np.sum(attr))

    # Normalize and convert to heatmap
    attr_norm = (attr - attr.min()) / (attr.max() - attr.min() + 1e-8)
    heatmap = np.mean(attr_norm, axis=0)

    # Paths to save images
    base = os.path.splitext(filename)[0]
    orig_path = os.path.join(output_dir, f"{base}_orig.png")
    heatmap_path = os.path.join(output_dir, f"{base}_lrp_heatmap.png")
    overlay_path = os.path.join(output_dir, f"{base}_lrp_overlay.png")

    # Save images
    orig.resize((224, 224)).save(orig_path)
    plt.imsave(heatmap_path, heatmap, cmap='hot')
```

```
        # Create and save overlay
        fig, axs = plt.subplots(1, 3, figsize=(15, 4))
        axs[0].imshow(orig.resize((224, 224)))
        axs[0].set_title("Original")

        axs[1].imshow(heatmap, cmap='hot')
        axs[1].set_title("LRP Heatmap")

        axs[2].imshow(orig.resize((224, 224)))
        axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
        axs[2].set_title("Overlay")

        for ax in axs: ax.axis('off')
        plt.tight_layout()
        plt.savefig(overlay_path)
        plt.close()

        # ✅ Log
        log_rows.append({
            "Image": filename,
            "Class": label,
            "Predicted Class": pred_class,
            "LRP Score": round(relevance_score, 4),
            "Original Image": orig_path,
            "LRP Heatmap": heatmap_path,
            "Overlay": overlay_path
        })

# ✅ Save to CSV
log_df = pd.DataFrame(log_rows)
log_csv = os.path.join(output_dir, "lrp_densenet201_log.csv")
log_df.to_csv(log_csv, index=False)
print(f"✅ LRP visual logs saved to: {log_csv}")

# ✅ Display 30 Samples in Table
def make_image_tag(img_path):
    return f'<img src="{img_path}" width="160"/>'

display_df = log_df.copy()
for col in ["Original Image", "LRP Heatmap", "Overlay"]:
    display_df[col] = display_df[col].apply(make_image_tag)

display(HTML(display_df.to_html(escape=False)))
```

```
Using device: cpu
Generating LRP Visuals: 100%|██████████| 30/30 [01:51<00:00,  3.71s/it] ✅ LRP visual logs saved to: /content/lrp_densenet201_res
```

	Image	Class	Predicted Class	LRP Score	Original Image	LRP Heatmap	Overlay
0	CRW_4853tamp1.jpg	Tp	1	-0.9621			
1	CRW_4901_JFRtamp1.jpg	Tp	1	0.7055			
2	CRW_4901_JFRtamp131.jpg	Tp	1	0.1164			
3	DSCF8tamp132.jpg	Tp	1	-3.9791			
4	DSCN41tamp237.jpg	Tp	1	-0.7869			
5	DSCN45tamp132.jpg	Tp	1	1.4386			
6	DSCN45tamp25.jpg	Tp	1	10.1930			
7	DSC_0535tamp237.jpg	Tp	1	-3.4815			
8	DSC_0812tamp134.jpg	Tp	1	3.9640			
9	DSC_0812tamp27.jpg	Tp	1	9.5410			
10	DSC_1535tamp1.jpg	Tp	1	1.4618			
11	DSC_1535tamp132.jpg	Tp	1	1.3952			
12	DSC_1535tamp27.jpg	Tp	1	1.6614			
13	DSC_1540tamp1.jpg	Tp	1	2.9181			
14	DSC_1540tamp131.jpg	Tp	1	-1.0758			
15	CRW_4809_scale.jpg	Au	0	1.2279			
16	CRW_4825_scale.jpg	Au	0	5.1514			
17	CRW_4827_scale.jpg	Au	0	1.3051			
18	CRW_4841_scale.jpg	Au	0	1.3791			
19	DSCF18_scale.jpg	Au	0	3.5179			
20	DSCF5_scale.jpg	Au	0	13.0736			
21	DSCN2320_scale.jpg	Au	0	0.6283			
22	DSCN2329_scale.jpg	Au	0	0.3386			
23	DSCN49_scale.jpg	Au	0	-6.5786			
24	DSC_1535_scale.jpg	Au	1	-63.6523			
25	DSC_1539_scale.jpg	Au	1	-0.0099			
26	DSC_1542_scale.jpg	Au	0	-2.1664			
27	DSC_1561_scale.jpg	Au	0	-3.2605			
28	DSC_1568_scale.jpg	Au	1	0.9960			
29	DSC_1570_scale.jpg	Au	0	1.0867			

```
!pip install captum
```

```
Requirement already satisfied: captum in /usr/local/lib/python3.11/dist-packages (0.8.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from captum) (3.10.0)
Requirement already satisfied: numpy<2.0 in /usr/local/lib/python3.11/dist-packages (from captum) (1.26.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from captum) (24.2)
Requirement already satisfied: torch>=1.10 in /usr/local/lib/python3.11/dist-packages (from captum) (2.6.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from captum) (4.67.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (4.14
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->capt
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (9
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (11
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
```

```
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2.21
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (12
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.10->captu
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (2.9.0.post
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->captum)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.10->captum) (3.0.2
```

```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
import pandas as pd
from tqdm import tqdm
from IPython.display import display, HTML

# Paths
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"
output_dir = "/content/lrp_densenet201_results"
os.makedirs(output_dir, exist_ok=True)

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("✅ Using device:", device)

# Load trained DenseNet-201 model
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Set up LRP
lrp = LRP(model)

# Image transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# Collect sample images (5 total: 3 Tp, 2 Au)
image_paths = []
for label, count in zip(["Tp", "Au"], [3, 2]):
    label_dir = os.path.join(test_images_dir, label)
    images = sorted([f for f in os.listdir(label_dir) if f.lower().endswith(('.jpg', '.png'))])[:count]
    image_paths.extend([(os.path.join(label_dir, f), label) for f in images])

# Storage for results
log_rows = []

# LRP loop
for img_path, label in tqdm(image_paths, desc="LRP Visualization"):
    filename = os.path.basename(img_path)
    orig = Image.open(img_path).convert("RGB")
    input_tensor = transform(orig).unsqueeze(0).to(device)
    input_tensor.requires_grad_()

    # Predict class
    output = model(input_tensor)
    pred_class = output.argmax(dim=1).item()

    # Get LRP attributions
    attr = lrp.attribute(input_tensor, target=pred_class)
    attr = attr.squeeze().detach().cpu().numpy()
    relevance_score = np.sum(attr)
```

```
        # Normalize for heatmap
        attr_norm = (attr - attr.min()) / (attr.max() - attr.min() + 1e-8)
        heatmap = np.mean(attr_norm, axis=0)

        # Save images
        base = os.path.splitext(filename)[0]
        orig_path = os.path.join(output_dir, f"{base}_orig.png")
        heatmap_path = os.path.join(output_dir, f"{base}_lrp_heatmap.png")
        overlay_path = os.path.join(output_dir, f"{base}_lrp_overlay.png")

        resized_orig = orig.resize((224, 224))
        resized_orig.save(orig_path)
        plt.imsave(heatmap_path, heatmap, cmap='hot')

        # Overlay
        fig, axs = plt.subplots(1, 3, figsize=(15, 4))
        axs[0].imshow(resized_orig)
        axs[0].set_title("Original")

        axs[1].imshow(heatmap, cmap='hot')
        axs[1].set_title("LRP Heatmap")

        axs[2].imshow(resized_orig)
        axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
        axs[2].set_title("Overlay")

        for ax in axs: ax.axis('off')
        plt.tight_layout()
        plt.savefig(overlay_path)
        plt.close()

        # Log
        log_rows.append({
            "Image Name": filename,
            "Class": label,
            "Predicted": "Tp" if pred_class == 1 else "Au",
            "LRP Score": round(relevance_score, 4),
            "Original Image": f'<img src="{orig_path}" width="100">',
            "LRP Heatmap": f'<img src="{heatmap_path}" width="100">',
            "Overlay": f'<img src="{overlay_path}" width="100">'
        })

    # Convert to DataFrame
    log_df = pd.DataFrame(log_rows)
    display(HTML(log_df.to_html(escape=False)))
```

```
✅ Using device: cpu
LRP Visualization: 100%|██████████| 5/5 [00:36<00:00,  7.37s/it]
```

	Image Name	Class	Predicted	LRP Score	Original Image	LRP Heatmap	Overlay
0	CRW_4853tamp1.jpg	Tp	Tp	-0.9621			
1	CRW_4901_JFRtamp1.jpg	Tp	Tp	0.7055			
2	CRW_4901_JFRtamp131.jpg	Tp	Tp	0.1164			
3	CRW_4809_scale.jpg	Au	Au	1.2279			
4	CRW_4825_scale.jpg	Au	Au	5.1514			

```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
import pandas as pd
from tqdm import tqdm
from IPython.display import display, HTML

# === Paths ===
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"
output_dir = "/content/lrp_densenet201_results"
os.makedirs(output_dir, exist_ok=True)

# === Device ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("✅ Using device:", device)
```

```
# === Load Fine-Tuned DenseNet-201 ===
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# === Captum: LRP ===
lrp = LRP(model)

# === Transforms ===
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# === Sample Image Selection ===
image_paths = []
for label, count in zip(["Tp", "Au"], [3, 2]):
    label_dir = os.path.join(test_images_dir, label)
    images = sorted([f for f in os.listdir(label_dir) if f.lower().endswith(('.jpg', '.png'))])[:count]
    image_paths.extend([(os.path.join(label_dir, f), label) for f in images])

# === LRP Attribution Loop ===
log_rows = []
for img_path, label in tqdm(image_paths, desc="Generating LRP Explanations"):
    filename = os.path.basename(img_path)
    orig = Image.open(img_path).convert("RGB")
    input_tensor = transform(orig).unsqueeze(0).to(device)
    input_tensor.requires_grad_()

    # Predict
    output = model(input_tensor)
    pred_class = output.argmax(dim=1).item()

    # LRP
    attr = lrp.attribute(input_tensor, target=pred_class)
    attr = attr.squeeze().detach().cpu().numpy()
    relevance_score = np.sum(attr)

    # Normalize Heatmap
    attr_norm = (attr - attr.min()) / (attr.max() - attr.min() + 1e-8)
    heatmap = np.mean(attr_norm, axis=0)

    # Save paths
    base = os.path.splitext(filename)[0]
    orig_path = os.path.join(output_dir, f"{base}_orig.png")
    heatmap_path = os.path.join(output_dir, f"{base}_lrp_heatmap.png")
    overlay_path = os.path.join(output_dir, f"{base}_lrp_overlay.png")

    # Save images
    resized = orig.resize((224, 224))
    resized.save(orig_path)
    plt.imsave(heatmap_path, heatmap, cmap='hot')

    # Overlay
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))
    axs[0].imshow(resized)
    axs[0].set_title("Original")
    axs[1].imshow(heatmap, cmap='hot')
    axs[1].set_title("LRP Heatmap")
    axs[2].imshow(resized)
    axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
    axs[2].set_title("Overlay")
    for ax in axs: ax.axis('off')
    plt.tight_layout()
    plt.savefig(overlay_path)
    plt.close()

    # Log row
    log_rows.append({
        "Image Name": filename,
        "LRP Score": round(relevance_score, 4),
        "LRP Heatmap": f'<img src="{heatmap_path}" width="120">',
        "Overlay": f'<img src="{overlay_path}" width="120">'
    })

# === Display Table ===
log_df = pd.DataFrame(log_rows)
display(HTML(log_df.to_html(escape=False)))
```

```
✅ Using device: cpu
Generating LRP Explanations: 100%|██████████| 5/5 [00:18<00:00,  3.64s/it]
```

	Image Name	LRP Score	LRP Heatmap	Overlay
0	CRW_4853tamp1.jpg	-0.9621		
1	CRW_4901_JFRtamp1.jpg	0.7055		
2	CRW_4901_JFRtamp131.jpg	0.1164		
3	CRW_4809_scale.jpg	1.2279		
4	CRW_4825_scale.jpg	5.1514		

```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
from tqdm import tqdm

# Paths
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"
output_dir = "/content/lrp_densenet201_samples"
os.makedirs(output_dir, exist_ok=True)

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("✅ Using device:", device)

# Load DenseNet-201 model
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Captum LRP
lrp = LRP(model)

# Transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# Select 5 images (3 tampered, 2 authentic)
image_paths = []
for label, count in zip(["Tp", "Au"], [3, 2]):
    folder = os.path.join(test_images_dir, label)
    files = sorted([f for f in os.listdir(folder) if f.lower().endswith(('.jpg', '.png'))])[:count]
    image_paths.extend([(os.path.join(folder, f), label) for f in files])

# Process each image
for img_path, label in tqdm(image_paths, desc="Processing Images"):
    filename = os.path.basename(img_path)
    image = Image.open(img_path).convert("RGB").resize((224, 224))
    input_tensor = transform(image).unsqueeze(0).to(device)
    input_tensor.requires_grad_()

    # Prediction
    output = model(input_tensor)
    pred_class = output.argmax(dim=1).item()

    # LRP Attribution
    attr = lrp.attribute(input_tensor, target=pred_class)
    attr = attr.squeeze().detach().cpu().numpy()
    relevance_score = np.sum(attr)

    # Normalize & Average across channels
    attr_norm = (attr - attr.min()) / (attr.max() - attr.min() + 1e-8)
    heatmap = np.mean(attr_norm, axis=0)

    # Save paths
    base = os.path.splitext(filename)[0]
    orig_path = os.path.join(output_dir, f"{base}_original.png")
    heatmap_path = os.path.join(output_dir, f"{base}_heatmap.png")
    overlay_path = os.path.join(output_dir, f"{base}_overlay.png")
```

```
    # Save original
    image.save(orig_path)

    # Save heatmap
    plt.imsave(heatmap_path, heatmap, cmap='hot')

    # Save overlay
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))
    axs[0].imshow(image)
    axs[0].set_title("Original")

    axs[1].imshow(heatmap, cmap='hot')
    axs[1].set_title("LRP Heatmap")

    axs[2].imshow(image)
    axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
    axs[2].set_title("Overlay")

    for ax in axs: ax.axis('off')
    plt.tight_layout()
    plt.savefig(overlay_path)
    plt.close()

    print(f"✅ {filename} | LRP Score: {round(relevance_score, 4)}")
```

```
✅ Using device: cpu
Processing Images:  20%|██          | 1/5 [00:04<00:17,  4.36s/it]✅ CRW_4853tamp1.jpg | LRP Score: 1.7145999670028687
Processing Images:  40%|████        | 2/5 [00:07<00:10,  3.59s/it]✅ CRW_4901_JFRtamp1.jpg | LRP Score: -3.086199998855591
Processing Images:  60%|██████      | 3/5 [00:11<00:07,  3.93s/it]✅ CRW_4901_JFRtamp131.jpg | LRP Score: 1.4218000173568726
Processing Images:  80%|████████    | 4/5 [00:14<00:03,  3.60s/it]✅ CRW_4809_scale.jpg | LRP Score: -0.16750000417232513
Processing Images: 100%|██████████| 5/5 [00:18<00:00,  3.65s/it]✅ CRW_4825_scale.jpg | LRP Score: 15.140700340270996
```

```
!pip install captum
```

```
Requirement already satisfied: captum in /usr/local/lib/python3.11/dist-packages (0.8.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from captum) (3.10.0)
Requirement already satisfied: numpy<2.0 in /usr/local/lib/python3.11/dist-packages (from captum) (1.26.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from captum) (24.2)
Requirement already satisfied: torch>=1.10 in /usr/local/lib/python3.11/dist-packages (from captum) (2.6.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from captum) (4.67.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (4.14
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->capt
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (9
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (11
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (2.21
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (12
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.10->captum) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=1.10->captu
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->captum) (2.9.0.post
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->captum)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.10->captum) (3.0.2
```

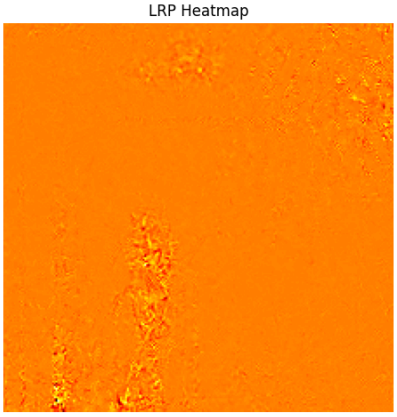
```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
from tqdm import tqdm

# --- Paths ---
```

```
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"

# Output dir
output_dir = "/content/lrp_densenet201_outputs"
os.makedirs(output_dir, exist_ok=True)

# --- Device Setup ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# --- Load DenseNet-201 model ---
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# --- LRP Setup ---
lrp = LRP(model)

# --- Image Preprocessing ---
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# --- Select sample images (5+) ---
image_paths = []
for label in ["Tp", "Au"]:
    sub_dir = os.path.join(test_images_dir, label)
    images = sorted([f for f in os.listdir(sub_dir) if f.lower().endswith(('.jpg', '.png'))])[:3]
    image_paths.extend([(os.path.join(sub_dir, f), label) for f in images])

# --- Loop through and explain ---
for img_path, label in tqdm(image_paths, desc="Generating LRP Explanations"):
    filename = os.path.basename(img_path)
    orig = Image.open(img_path).convert("RGB")
    resized = orig.resize((224, 224))
    input_tensor = transform(resized).unsqueeze(0).to(device)
    input_tensor.requires_grad_()

    # Predict class
    output = model(input_tensor)
    pred_class = output.argmax(dim=1).item()

    # Get LRP attributions
    attributions = lrp.attribute(input_tensor, target=pred_class).squeeze().detach().cpu().numpy()
    relevance_score = np.sum(attributions)

    # Normalize and average across RGB channels
    attr_norm = (attributions - attributions.min()) / (attributions.max() - attributions.min() + 1e-8)
    heatmap = np.mean(attr_norm, axis=0)

    # --- Visualization ---
    fig, axs = plt.subplots(1, 3, figsize=(16, 5))

    axs[0].imshow(resized)
    axs[0].set_title(f"Original: {filename}")

    axs[1].imshow(heatmap, cmap='hot')
    axs[1].set_title("LRP Heatmap")

    axs[2].imshow(resized)
    axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
    axs[2].set_title(f"Overlay\nLRP Score: {relevance_score:.2f}")

    for ax in axs:
        ax.axis("off")
    plt.tight_layout()
    plt.show()
```

```
Using device: cpu
Generating LRP Explanations:   0%|          | 0/6 [00:00<?, ?it/s]
```

Original: CRW_4853tamp1.jpg

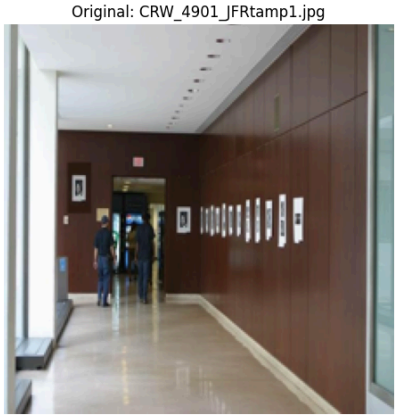
LRP Heatmap



Overlay
LRP Score: 1.71

```
Generating LRP Explanations:  17%|██          | 1/6 [00:03<00:19,  3.85s/it]
```

Original: CRW_4901_JFRtamp1.jpg

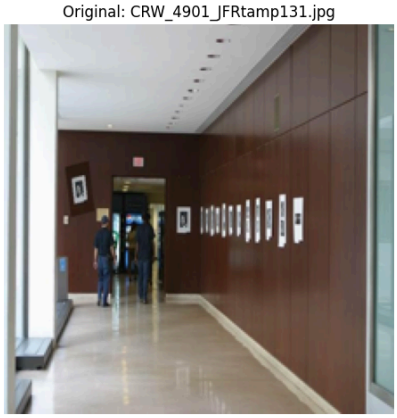


LRP Heatmap

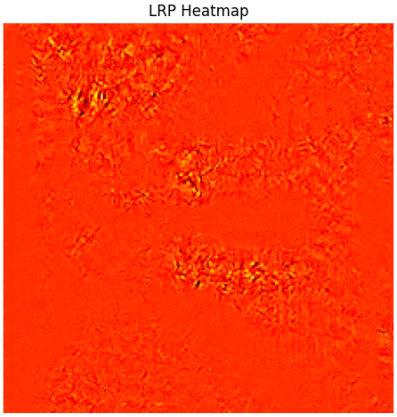
Overlay
LRP Score: -3.09

```
Generating LRP Explanations:  33%|████        | 2/6 [00:07<00:16,  4.01s/it]
```

Original: CRW_4901_JFRtamp131.jpg



LRP Heatmap

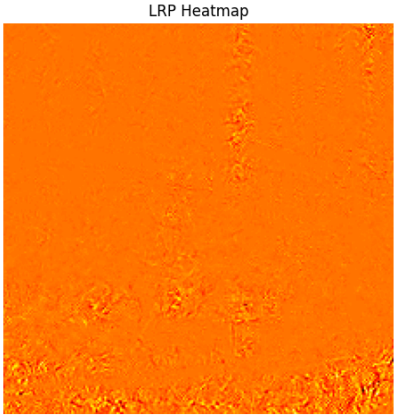


Overlay
LRP Score: 1.42

```
Generating LRP Explanations:  50%|██████      | 3/6 [00:11<00:11,  4.00s/it]
```

Original: CRW_4809_scale.jpg

LRP Heatmap



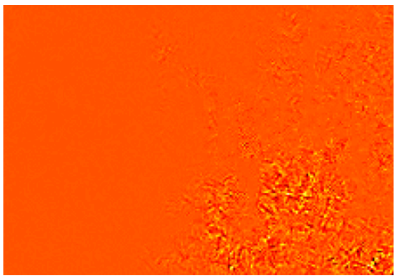
Overlay
LRP Score: -0.17

```
Generating LRP Explanations:  67%|████████    | 4/6 [00:15<00:07,  3.69s/it]
```

Original: CRW_4825_scale.jpg

LRP Heatmap

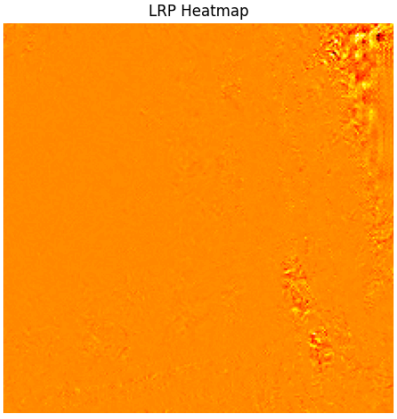
Overlay
LRP Score: 15.14


```
Generating LRP Explanations:  83%|████████▎ | 5/6 [00:18<00:03,  3.53s/it]
```

Original: CRW_4827_scale.jpg

LRP Heatmap



Overlay
LRP Score: -2.75

```
Generating LRP Explanations: 100%|██████████| 6/6 [00:22<00:00,  3.79s/it]
```

LIME Code for DenseNet-201 on MICC-F220

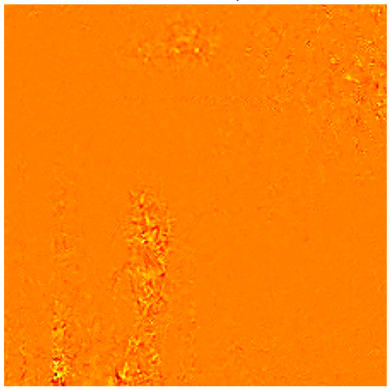
```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import models, transforms
from captum.attr import LRP
from PIL import Image
from tqdm import tqdm

# --- Paths ---
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/densenet201_micc_f220.pth"

# Output dir
output_dir = "/content/lrp_densenet201_outputs"
os.makedirs(output_dir, exist_ok=True)

# --- Device Setup ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# --- Load DenseNet-201 model ---
model = models.densenet201(weights=None)
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# --- LRP Setup ---
lrp = LRP(model)

# --- Image Preprocessing ---
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# --- Select sample images (5+) ---
```

```
    # --- Select sample images (3+3) ---
    image_paths = []
    for label in ["Tp", "Au"]:
        sub_dir = os.path.join(test_images_dir, label)
        images = sorted([f for f in os.listdir(sub_dir) if f.lower().endswith(('.jpg', '.png'))])[:3]
        image_paths.extend([(os.path.join(sub_dir, f), label) for f in images])

    # --- Loop through and explain ---
    for img_path, label in tqdm(image_paths, desc="Generating LRP Explanations"):
        filename = os.path.basename(img_path)
        orig = Image.open(img_path).convert("RGB")
        resized = orig.resize((224, 224))
        input_tensor = transform(resized).unsqueeze(0).to(device)
        input_tensor.requires_grad_()

        # Predict class
        output = model(input_tensor)
        pred_class = output.argmax(dim=1).item()

        # Get LRP attributions
        attributions = lrp.attribute(input_tensor, target=pred_class).squeeze().detach().cpu().numpy()
        relevance_score = np.sum(attributions)

        # Normalize and average across RGB channels
        attr_norm = (attributions - attributions.min()) / (attributions.max() - attributions.min() + 1e-8)
        heatmap = np.mean(attr_norm, axis=0)

        # --- Visualization ---
        fig, axs = plt.subplots(1, 3, figsize=(16, 5))

        axs[0].imshow(resized)
        axs[0].set_title(f"Original: {filename}")

        axs[1].imshow(heatmap, cmap='hot')
        axs[1].set_title("LRP Heatmap")

        axs[2].imshow(resized)
        axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
        axs[2].set_title(f"Overlay\nLRP Score: {relevance_score:.2f}")

        for ax in axs:
            ax.axis("off")
        plt.tight_layout()
        plt.show()
```

```
Using device: cpu
Generating LRP Explanations:   0%|          | 0/6 [00:00<?, ?it/s]
```

Original: CRW_4853tamp1.jpg

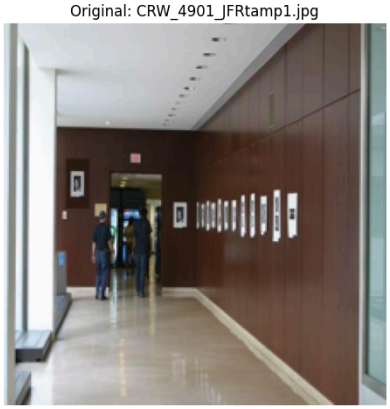
LRP Heatmap



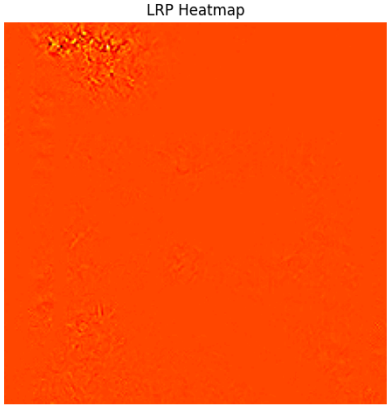
Overlay
LRP Score: 1.71

```
Generating LRP Explanations:  17%|██        | 1/6 [00:11<00:57, 11.45s/it]
```

Original: CRW_4901_JFRtamp1.jpg



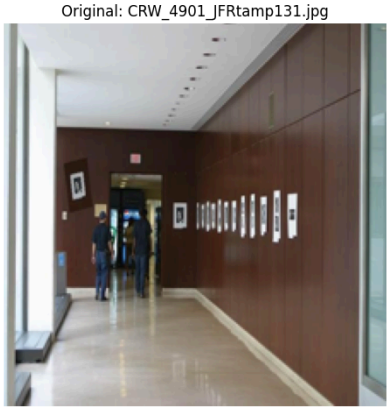
LRP Heatmap



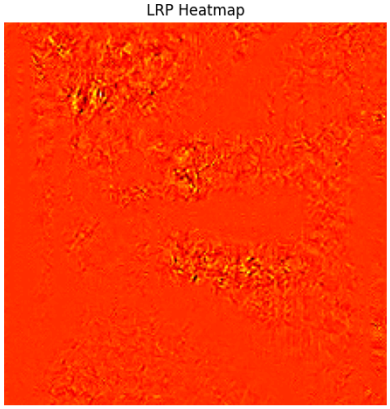
Overlay
LRP Score: -3.09

```
Generating LRP Explanations:  33%|████      | 2/6 [00:17<00:33,  8.36s/it]
```

Original: CRW_4901_JFRtamp131.jpg



LRP Heatmap

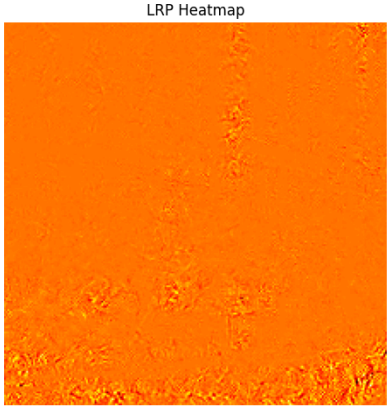


Overlay
LRP Score: 1.42

```
Generating LRP Explanations:  50%|██████    | 3/6 [00:26<00:26,  8.70s/it]
```

Original: CRW_4809_scale.jpg

LRP Heatmap

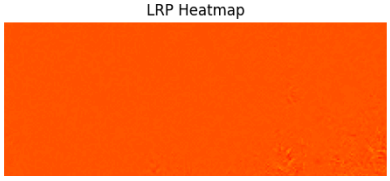


Overlay
LRP Score: -0.17

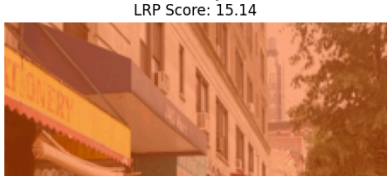
```
Generating LRP Explanations:  67%|████████  | 4/6 [00:46<00:25, 12.97s/it]
```

Original: CRW_4825_scale.jpg

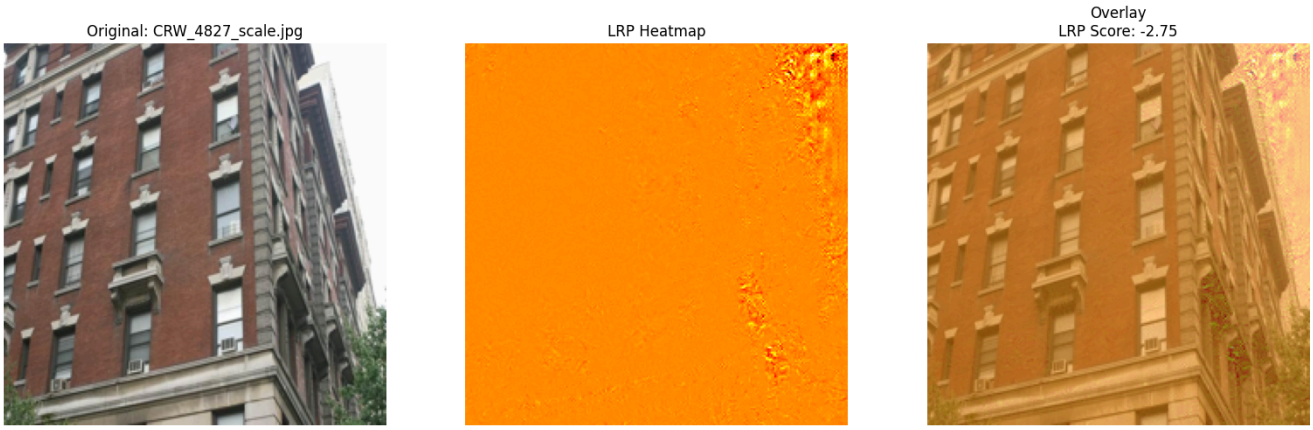
LRP Heatmap



Overlay
LRP Score: 15.14



```
Generating LRP Explanations:  83%|████████▎ | 5/6 [00:50<00:09,  9.67s/it]
```




```
Generating LRP Explanations: 100%|██████████| 6/6 [00:53<00:00,  8.89s/it]
```

```
!pip install lime
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 4.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from lime) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from lime) (1.15.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.11/dist-packages (from lime) (1.6.1)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.11/dist-packages (from lime) (0.25.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (3.5)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (11.2.1)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (2
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (2025
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18->lime) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18->lime) (3.6
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (2.9.0.post0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283834 sha256=cb973de77d0af2c3cf9ba013e30e267efde7024afd448511
  Stored in directory: /root/.cache/pip/wheels/85/fa/a3/9c2d44c9f3cd77cf4e533b58900b2bf4487f2a17e8ec212a3d
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1
```

```
import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import models, transforms
```