Dataset Download and Exraction

```
# STEP 1: Setup Kaggle API on Colab for automatic download

from google.colab import files
import os

print("Please upload your Kaggle API JSON file (kaggle.json):")
uploaded = files.upload()  # You'll need to select your kaggle.json

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

# Download the MICC F220 dataset
!kaggle datasets download mashraffarouk/micc-f220 --unzip -p ./micc_f220

# Verify the folder structure
import os

print("\nMICC F220 Dataset folder structure:")
for root, dirs, files in os.walk('./micc_f220'):
    level = root.replace('./micc_f220', '').count(os.sep)
    indent = ' ' * 4 * (level)
    print(f"{indent}{os.path.basename(root)}/")
    subindent = ' ' * 4 * (level + 1)
    for f in files[:5]:  # Just 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/mashraffarouk/micc-f220
License(s): unknown
Downloading micc-f220.zip to ./micc_f220
  0% 0.00/14.4M [00:00<?, ?B/s]
100% 14.4M/14.4M [00:00<00:00, 1.07GB/s]

MICC F220 Dataset folder structure:
micc_f220/
    MICC-F220/
        groundtruthDB_220.txt
        Tu/
            DSC_0535tamp133.jpg
            DSCN41tamp27.jpg
            DSC_0812tamp132.jpg
            DSCN41tamp1.jpg
            DSC_1535tamp131.jpg
        Au/
            DSCN2320_scale.jpg
            IMG_32_scale.jpg
            DSCF5_scale.jpg
            CRW_4821_scale.jpg
```

Data Preparation – Splitting and Summary

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

# Paths
base_dir = 'micc_f220/MICC-F220'
split_dir = 'micc_f220/split'
class_names = ['Tu', 'Au']

# Set 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 move/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'))]
    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 (Tu)', 'Authentic (Au)']
summary_df['Total'] = summary_df['Tampered (Tu)'] + summary_df['Authentic (Au)']
print("\nDataset Split Summary:")
display(summary_df)

# Display some sample images from each split/class
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 (Tu) images from train split:")
show_samples('train', 'Tu')
print("Sample Authentic (Au) images from train split:")
show_samples('train', 'Au')
```

Dataset Split Summary:

	Tampered (Tu)	Authentic (Au)	Total
train	77	77	154
val	16	16	32
test	17	17	34

Sample Tampered (Tu) images from train split:

train/Tu
sony_61tamp237.jpg

train/Tu
CRW_4901_JFRtamp176.jpg

train/Tu
DSC_1535tamp37.jpg

Sample Authentic (Au) images from train split:

train/Au
DSC_1573_scale.jpg

train/Au
sony_72_scale.jpg

train/Au
DSCF5_scale.jpg

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

*Model Training on MICC-F220 *

```
import torch
from torchvision import datasets, transforms

# Data transforms for training/val/test
IMG_SIZE = 224  # ResNet standard
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', 'Tu']
```

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

# Load pre-trained ResNet-50
model = models.resnet50(pretrained=True)

# Replace the classifier (fc layer) for binary classification
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)
model = model.to(device)

# Optionally freeze early layers (fine-tune only classifier & last block)
for name, param in model.named_parameters():
    if 'fc' not in name and 'layer4' not in name:
        param.requires_grad = False
```

```
/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/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 211MB/s]
```

```
import torch.optim as optim

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

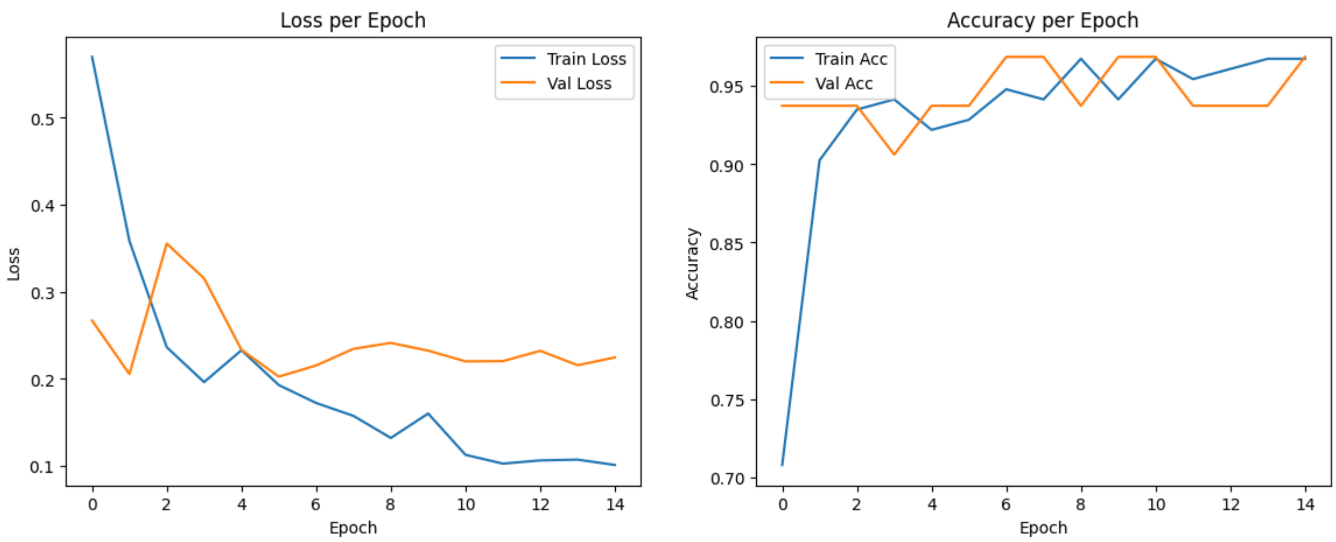
```
import time
import copy
import matplotlib.pyplot as plt

num_epochs = 15

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())
            # Save best model
            if epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

model.load_state_dict(best_model_wts)
```

```
Train Loss: 0.1964 Acc: 0.9416
Val Loss: 0.3154 Acc: 0.9062

Epoch 5/15
--------------------
Train Loss: 0.2330 Acc: 0.9221
Val Loss: 0.2335 Acc: 0.9375

Epoch 6/15
--------------------
Train Loss: 0.1930 Acc: 0.9286
Val Loss: 0.2027 Acc: 0.9375

Epoch 7/15
--------------------
Train Loss: 0.1724 Acc: 0.9481
Val Loss: 0.2155 Acc: 0.9688

Epoch 8/15
--------------------
Train Loss: 0.1576 Acc: 0.9416
Val Loss: 0.2347 Acc: 0.9688

Epoch 9/15
--------------------
Train Loss: 0.1323 Acc: 0.9675
Val Loss: 0.2414 Acc: 0.9375

Epoch 10/15
--------------------
Train Loss: 0.1603 Acc: 0.9416
Val Loss: 0.2325 Acc: 0.9688

Epoch 11/15
--------------------
Train Loss: 0.1129 Acc: 0.9675
Val Loss: 0.2203 Acc: 0.9688

Epoch 12/15
--------------------
Train Loss: 0.1028 Acc: 0.9545
Val Loss: 0.2205 Acc: 0.9375

Epoch 13/15
--------------------
Train Loss: 0.1065 Acc: 0.9610
Val Loss: 0.2323 Acc: 0.9375

Epoch 14/15
--------------------
Train Loss: 0.1074 Acc: 0.9675
Val Loss: 0.2158 Acc: 0.9375

Epoch 15/15
--------------------
Train Loss: 0.1012 Acc: 0.9675
Val Loss: 0.2247 Acc: 0.9688
<All keys matched successfully>
```

```
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: 0.8529
Test Precision: 0.7727
Test Recall: 1.0000
Test F1 Score: 0.8718
Confusion Matrix:
 [[12  5]
 [ 0 17]]
```

Tabulating and Visualizing Performance Metrics

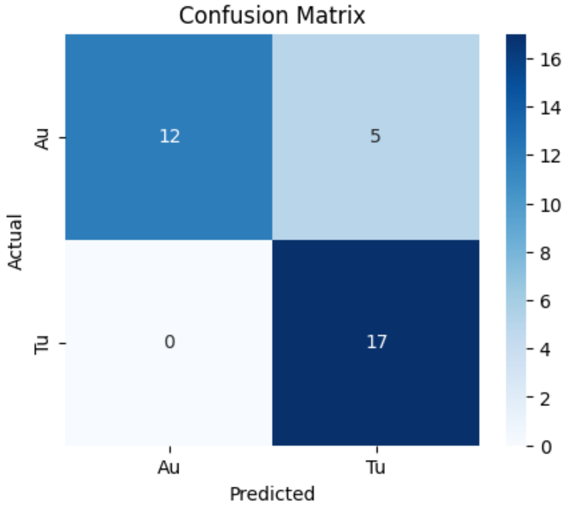
```
import torch
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score, jaccard_score
from scipy.stats import ttest_ind
import time

# Collect predictions, true labels, and probabilities for all test samples
model.eval()
all_preds = []
```

```
    all_labels = []
    all_probs = []

    with torch.no_grad():
        for inputs, labels in dataloaders['test']:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, 1)[:,1]  # probability for class 1
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            all_probs.extend(probs.cpu().numpy())

    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)
    all_probs = np.array(all_probs)

    # 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 (optional, requires ptflops)
    try:
        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)
    except Exception:
        macs = 'N/A (Install ptflops)'

    # Inference time (average per image)
    n_runs = 100
    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

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

    # Mean accuracy difference (here: absolute difference of classwise accuracy)
    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)
    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:,}", macs if isinstance(macs, str) else f"{macs:,}",
            f"{elapsed:.2f}", f"{mean_iou:.2f}", f"{mean_acc_diff:.4f}", f"{pval:.4g}"
        ]
    })
    print("\nFull Metrics Table:")
    display(metrics_table)

    # Confusion Matrix Heatmap
    import seaborn as sns
    import matplotlib.pyplot as plt

    plt.figure(figsize=(5,4))
    sns.heatmap(confusion_matrix(all_labels, all_preds), annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

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

plt.subplot(1,2,2)
plt.plot
```

Full Metrics Table:

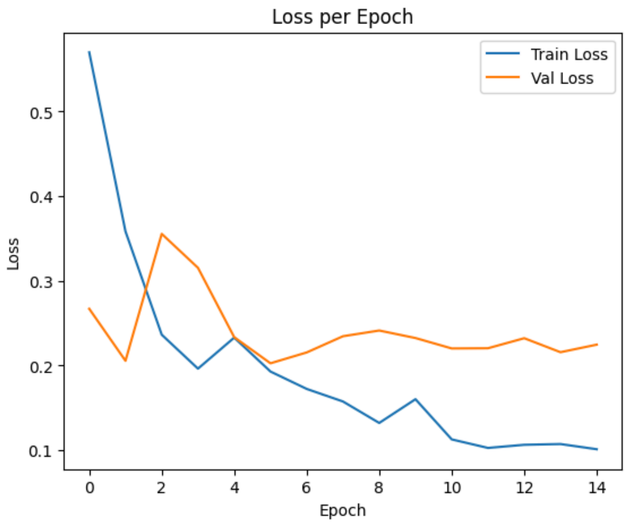
	Metric	Value
0	Accuracy	0.8529
1	Precision	0.7727
2	Recall	1.0000
3	F1 Score	0.8718
4	False Negatives	0
5	Parameter Count	23,512,130
6	FLOPs (MACs)	N/A (Install ptflops)
7	Inference Time (ms)	9.07
8	Mean IOU (%)	73.93
9	Mean Accuracy Diff	0.2941
10	p-value (t-test)	0.2263

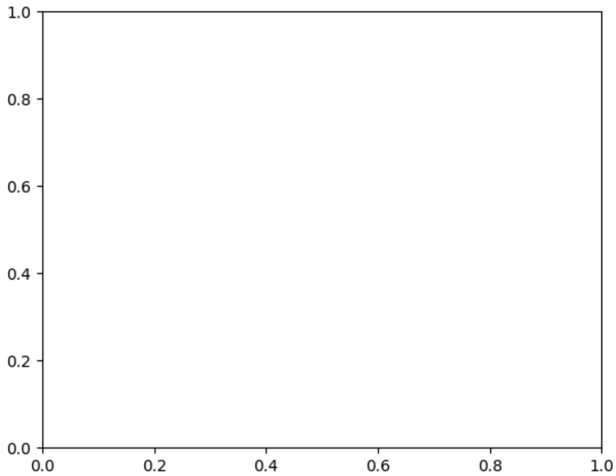



```
matplotlib.pyplot.plot
def plot(*args: float | ArrayLike | str, scalex: bool=True, scaley: bool=True, data=None, **kwargs)
-> list[Line2D]
```

/usr/local/lib/python3.11/dist-packages/matplotlib/pyplot.py

```
Plot y versus x as lines and/or markers.

Call signatures::
```





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

LRP Explainability

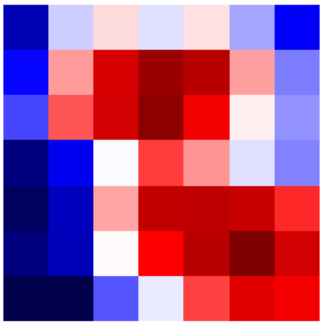
```
!pip install captum --quiet
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 6.1 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 64.4 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 112.9 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.0 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 122.2 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 99.2 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 55.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.0 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 42.7 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 18.8 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.6 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 102.3 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

N_SAMPLES = 5
model.eval()
test_imgs, test_labels = next(iter(dataloaders['test']))
test_imgs = test_imgs[:N_SAMPLES].to(device)
test_labels = test_labels[:N_SAMPLES]

deeplift = LayerDeepLift(model, model.layer4[-1])

fig, axes = plt.subplots(N_SAMPLES, 3, figsize=(12, 3*N_SAMPLES))
axes = axes if N_SAMPLES > 1 else [axes]

for i in range(N_SAMPLES):
    input_img = test_imgs[i].unsqueeze(0)
    input_img.requires_grad_()
    label = test_labels[i].item()
    output = model(input_img)
    pred = torch.argmax(output, dim=1).item()

    attributions = deeplift.attribute(input_img, target=pred)
    attr_map = attributions.squeeze().detach().cpu().numpy()
    attr_map = np.sum(attr_map, axis=0)
    score = np.sum(np.abs(attr_map))

    img_np = input_img.squeeze().permute(1,2,0).detach().cpu().numpy()  # <--- Fixed here
    img_disp = np.clip(img_np * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406], 0, 1)

    axes[i,0].imshow(img_disp)
    axes[i,0].set_title(f"Original\nTrue:{class_names[label]}, Pred:{class_names[pred]}")
    axes[i,0].axis('off')

    axes[i,1].imshow(attr_map, cmap='seismic')
    axes[i,1].set_title(f"Attribution Heatmap\nScore: {score:.2f}")
    axes[i,1].axis('off')

    attr_norm = (attr_map - attr_map.min()) / (attr_map.max() - attr_map.min() + 1e-8)
    axes[i,2].imshow(img_disp)
    axes[i,2].imshow(attr_norm, cmap='hot', alpha=0.4)
    axes[i,2].set_title("Overlay")
    axes[i,2].axis('off')

plt.tight_layout()
plt.show()
```

Original
True:Au, Pred:Au

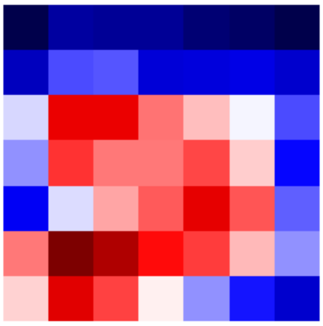
Attribution Heatmap
Score: 2.78



Overlay

Original
True:Au, Pred:Au

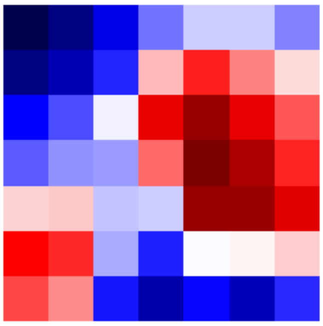
Attribution Heatmap
Score: 3.49



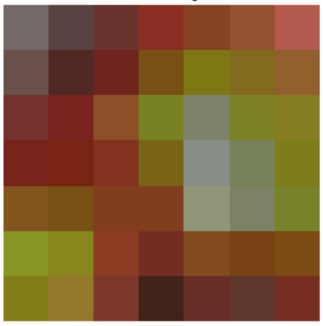
Overlay

Original
True:Au, Pred:Au

Attribution Heatmap
Score: 2.32

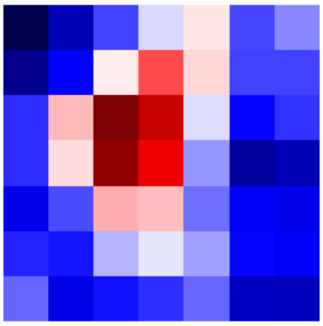


Overlay

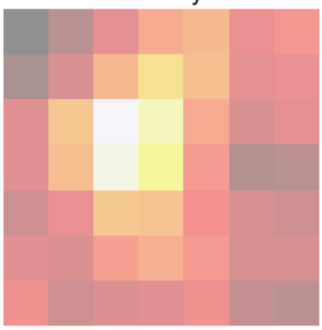


Original
True:Au, Pred:Au

Attribution Heatmap
Score: 1.10

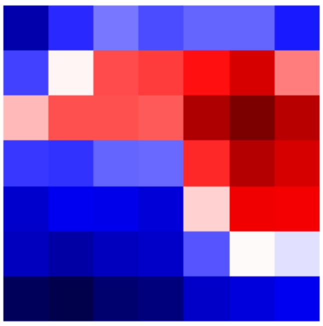


Overlay

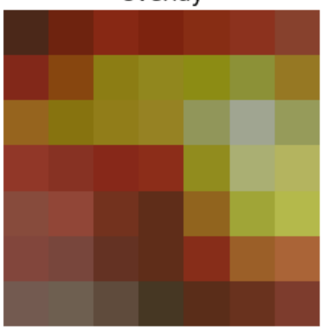


Original
True:Au, Pred:Au

Attribution Heatmap
Score: 2.21



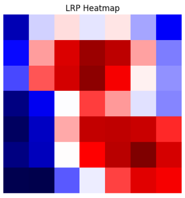
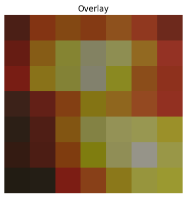
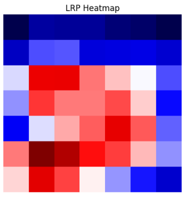
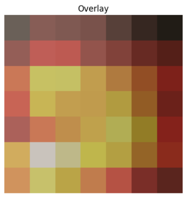
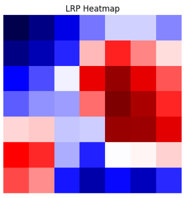
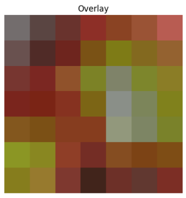
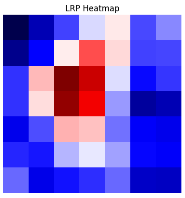
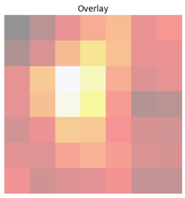
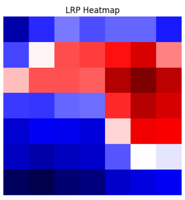
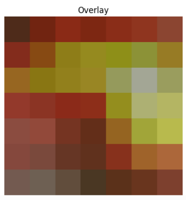
Overlay



```
import io
import base64
```

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

def fig2img(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png', bbox_inches='tight')
    buf.seek(0)
    img = Image.open(buf)
    return img

def img_to_html(img):
    buf = io.BytesIO()
    img.save(buf, format='PNG')
    buf.seek(0)
    data = base64.b64encode(buf.read()).decode('utf-8')
    return f'<img src="data:image/png;base64,{data}" width="140"/>'

# Get the filenames in order for test set (ImageFolder sorts by name)
test_folder = os.path.join(split_dir, 'test')
fnames = []
for idx in range(N_SAMPLES):
    label_folder = image_datasets['test'].imgs[idx][0].split('/')[-2]
    fname = image_datasets['test'].imgs[idx][0].split('/')[-1]
    fnames.append(fname)

# For N_SAMPLES test images
deeplift = LayerDeepLift(model, model.layer4[-1])
model.eval()
test_imgs, test_labels = next(iter(dataloaders['test']))
test_imgs = test_imgs[:N_SAMPLES].to(device)
test_labels = test_labels[:N_SAMPLES]

# Build table rows
rows = []
for i in range(N_SAMPLES):
    input_img = test_imgs[i].unsqueeze(0)
    input_img.requires_grad_()
    label = test_labels[i].item()
    output = model(input_img)
    pred = torch.argmax(output, dim=1).item()

    attributions = deeplift.attribute(input_img, target=pred)
    attr_map = attributions.squeeze().detach().cpu().numpy()
    attr_map = np.sum(attr_map, axis=0)
    score = np.sum(np.abs(attr_map))

    img_np = input_img.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)

    # Create LRP heatmap image
    fig1, ax1 = plt.subplots()
    ax1.imshow(attr_map, cmap='seismic')
    ax1.axis('off')
    ax1.set_title('LRP Heatmap')
    lrp_heatmap_img = fig2img(fig1)
    plt.close(fig1)

    # Create overlay image
    attr_norm = (attr_map - attr_map.min()) / (attr_map.max() - attr_map.min() + 1e-8)
    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 = fig2img(fig2)
    plt.close(fig2)

    row_html = f"""
    <tr>
        <td>{fnames[i]}</td>
        <td>{img_to_html(lrp_heatmap_img)}</td>
        <td>{score:.2f}</td>
        <td>{img_to_html(overlay_img)}</td>
    </tr>
    """
    rows.append(row_html)

# Create HTML table
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 the table in Colab/Notebook
    display(HTML(table_html))
```

Image Name	LRP Heatmap	LRP Score	Overlay
CRW_4815_scale.jpg		2.78	
CRW_4822_scale.jpg		3.49	
CRW_4831_scale.jpg		2.32	
CRW_4834_scale.jpg		1.10	
CRW_4842_scale.jpg		2.21	

LIME explainability on MICC F220

```
!pip install lime --quiet
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 17.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
```

```
# 1. Install LIME and required packages
!pip install lime scikit-image --quiet

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

# 3. Helper functions for images and HTML
def fig2img(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png', bbox_inches='tight')
```

```
    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="140"/>'

def preprocess_for_lime(img_tensor):
    """Convert normalized tensor image (C,H,W) to unnormalized numpy image (H,W,C), values 0-255 uint8."""
    img_np = img_tensor.permute(1,2,0).detach().cpu().numpy()
    img_np = np.clip(img_np * [0.229,0.224,0.225] + [0.485,0.456,0.406], 0, 1)
    img_np = (img_np*255).astype(np.uint8)
    return img_np

# 4. LIME explainer
explainer = lime_image.LimeImageExplainer()

# 5. Get filenames for test images
N_SAMPLES = 5
fnames = []
for idx in range(N_SAMPLES):
    fname = image_datasets['test'].imgs[idx][0].split('/')[-1]
    fnames.append(fname)

# 6. Define batch_predict function for LIME
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)
    # Normalize
    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

# 7. Prepare table rows
rows = []

model.eval()
test_imgs, test_labels = next(iter(dataloaders['test']))
test_imgs = test_imgs[:N_SAMPLES].to('cpu')  # LIME expects CPU numpy
test_labels = test_labels[:N_SAMPLES]

for i in range(N_SAMPLES):
    img_tensor = test_imgs[i].cpu()
    label = test_labels[i].item()
    img_np = preprocess_for_lime(img_tensor)

    # Run LIME
    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: sum of absolute importances for the superpixels used
    lime_score = np.sum([abs(weight) for _, weight in explanation.local_exp[pred_class]])

    # LIME heatmap (the mask only)
    fig1, ax1 = plt.subplots()
    ax1.imshow(mask_lime, cmap='seismic')
    ax1.axis('off')
    ax1.set_title('LIME Heatmap')
    lime_heatmap_img = fig2img(fig1)
    plt.close(fig1)

    # Overlay image
    overlay_img = mark_boundaries(img_np, mask_lime)
```

```
        fig2, ax2 = plt.subplots()
        ax2.imshow(overlay_img)
        ax2.axis('off')
        ax2.set_title('Overlay')
        overlay_img_pil = fig2img(fig2)
        plt.close(fig2)

        row_html = f"""
        <tr>
            <td>{fnames[i]}</td>
            <td>{img_to_html(lime_heatmap_img)}</td>
            <td>{lime_score:.2f}</td>
            <td>{img_to_html(overlay_img_pil)}</td>
        </tr>
        """
        rows.append(row_html)

    # 8. Create and display the HTML table
    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))
```

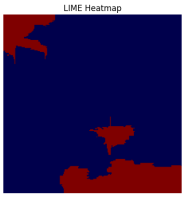
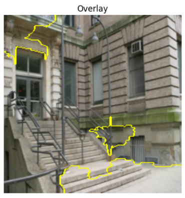
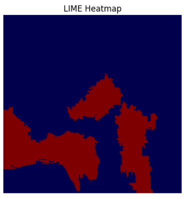
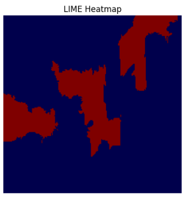
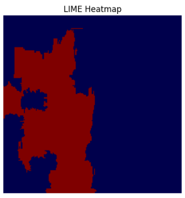
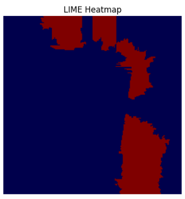
100% 1000/1000 [00:03<00:00, 263.37it/s]

100% 1000/1000 [00:03<00:00, 280.24it/s]

100% 1000/1000 [00:03<00:00, 268.25it/s]

100% 1000/1000 [00:03<00:00, 275.89it/s]

100% 1000/1000 [00:03<00:00, 272.52it/s]

Image Name	LIME Heatmap	LIME Score	Overlay
CRW_4815_scale.jpg		0.13	
CRW_4822_scale.jpg		0.06	
CRW_4831_scale.jpg		0.18	
CRW_4834_scale.jpg		0.21	
CRW_4842_scale.jpg		0.06	

```
# For vertical, non-tabular display of LIME explainability

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

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="200"/>'

def preprocess_for_lime(img_tensor):
    img_np = img_tensor.permute(1,2,0).detach().cpu().numpy()
    img_np = np.clip(img_np * [0.229,0.224,0.225] + [0.485,0.456,0.406], 0, 1)
    img_np = (img_np*255).astype(np.uint8)
```

```
        return img_np

    explainer = lime_image.LimeImageExplainer()

    N_SAMPLES = 5
    fnames = []
    for idx in range(N_SAMPLES):
        fname = image_datasets['test'].imgs[idx][0].split('/')[-1]
        fnames.append(fname)

    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

    model.eval()
    test_imgs, test_labels = next(iter(dataloaders['test']))
    test_imgs = test_imgs[:N_SAMPLES].to('cpu')
    test_labels = test_labels[:N_SAMPLES]

    for i in range(N_SAMPLES):
        img_tensor = test_imgs[i].cpu()
        label = test_labels[i].item()
        img_np = preprocess_for_lime(img_tensor)

        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 heatmap
        fig1, ax1 = plt.subplots()
        ax1.imshow(mask_lime, cmap='seismic')
        ax1.axis('off')
        ax1.set_title('LIME Heatmap')
        lime_heatmap_img = fig2img(fig1)
        plt.close(fig1)

        # Overlay image
        overlay_img = mark_boundaries(img_np, mask_lime)
        fig2, ax2 = plt.subplots()
        ax2.imshow(overlay_img)
        ax2.axis('off')
        ax2.set_title('Overlay')
        overlay_img_pil = fig2img(fig2)
        plt.close(fig2)

        # Display in vertical fashion
        display(HTML(f'<h4>Image Name: {fnames[i]}</h4>'))
        display(HTML(f'LIME Heatmap:<br>{img_to_html(lime_heatmap_img)}'))
        display(HTML(f'LIME Score: <b>{lime_score:.2f}</b>'))
        display(HTML(f'Overlay:<br>{img_to_html(overlay_img_pil)}'))
        display(HTML('<hr>'))
```