Dataset Download & Extraction

```
# STEP 1.1: Upload kaggle.json to access the Kaggle API
from google.colab import files
files.upload()  # Select your kaggle.json file here
```

```
Choose Files kaggle.json
• kaggle.json(application/json) - 64 bytes, last modified: 6/21/2025 - 100% done
Saving kaggle.json to kaggle.json
```

```
# STEP 1.2: Move kaggle.json to the correct location
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# STEP 1.3: Download the MICC F220 dataset from Kaggle
!kaggle datasets download -d mashraffarouk/micc-f220
```

```
Dataset URL: https://www.kaggle.com/datasets/mashraffarouk/micc-f220
License(s): unknown
micc-f220.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
# STEP 1.4: Unzip the dataset
!unzip -q micc-f220.zip -d /content/micc_f220_dataset

# STEP 1.5: Check structure
import os
from pathlib import Path

base_path = "/content/micc_f220_dataset"
for root, dirs, files in os.walk(base_path):
    print(f"📁 {root} - {len(files)} files")
    if files:
        sample_files = files[:5]
        for f in sample_files:
            print("  └──", f)
```

```
replace /content/micc_f220_dataset/MICC-F220/Au/CRW_4809_scale.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
📁 /content/micc_f220_dataset - 0 files
📁 /content/micc_f220_dataset/MICC-F220 - 1 files
  └── groundtruthDB_220.txt
📁 /content/micc_f220_dataset/MICC-F220/Tu - 110 files
  └── DSC_0535tamp133.jpg
  └── DSCN41tamp27.jpg
  └── DSC_0812tamp132.jpg
  └── DSCN41tamp1.jpg
  └── DSC_1535tamp131.jpg
📁 /content/micc_f220_dataset/MICC-F220/Au - 110 files
  └── DSCN2320_scale.jpg
  └── IMG_32_scale.jpg
  └── DSCF5_scale.jpg
  └── CRW_4821_scale.jpg
  └── CRW_4840_scale.jpg
```

Verifying & Organizing MICC-F220 Dataset

```
import os

dataset_root = "/content/micc_f220_dataset"

for root, dirs, files in os.walk(dataset_root):
    print(f"📁 {root} - {len(files)} files")
    for f in files[:5]:  # Preview only first 5 files
        print(f"    └── {f}")
```

```
📁 /content/micc_f220_dataset - 0 files
📁 /content/micc_f220_dataset/MICC-F220 - 1 files
    └── groundtruthDB_220.txt
📁 /content/micc_f220_dataset/MICC-F220/Tu - 110 files
    └── DSC_0535tamp133.jpg
    └── DSCN41tamp27.jpg
    └── DSC_0812tamp132.jpg
    └── DSCN41tamp1.jpg
    └── DSC_1535tamp131.jpg
📁 /content/micc_f220_dataset/MICC-F220/Au - 110 files
```

```
            └── DSCN2320_scale.jpg
            └── IMG_32_scale.jpg
            └── DSCF5_scale.jpg
            └── CRW_4821_scale.jpg
            └── CRW_4840_scale.jpg
```

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

# Source paths
tampered_dir = "/content/micc_f220_dataset/MICC-F220/Tu"
authentic_dir = "/content/micc_f220_dataset/MICC-F220/Au"

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

# Get image paths
tampered_images = sorted([os.path.join(tampered_dir, f) for f in os.listdir(tampered_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png
authentic_images = sorted([os.path.join(authentic_dir, f) for f in os.listdir(authentic_dir) if f.lower().endswith(('.jpg', '.jpeg', '.

# Utility to split and copy
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 subset_name, subset in zip(['train', 'val', 'test'], [train, val, test]):
        target_dir = os.path.join(output_base, subset_name, label)
        os.makedirs(target_dir, exist_ok=True)
        for img_path in subset:
            shutil.copy(img_path, os.path.join(target_dir, os.path.basename(img_path)))

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

# Split and organize
tp_counts = split_and_copy(tampered_images, 'Tp')
au_counts = split_and_copy(authentic_images, 'Au')

# Summary
df_summary = pd.DataFrame({
    'Split': ['Train', 'Validation', 'Test'],
    'Tampered': tp_counts,
    'Authentic': au_counts
})
print(df_summary)
```

```
         Split  Tampered  Authentic
0       Train        77         77
1  Validation        16         16
2        Test        17         17
```

VGG-19 Model Training

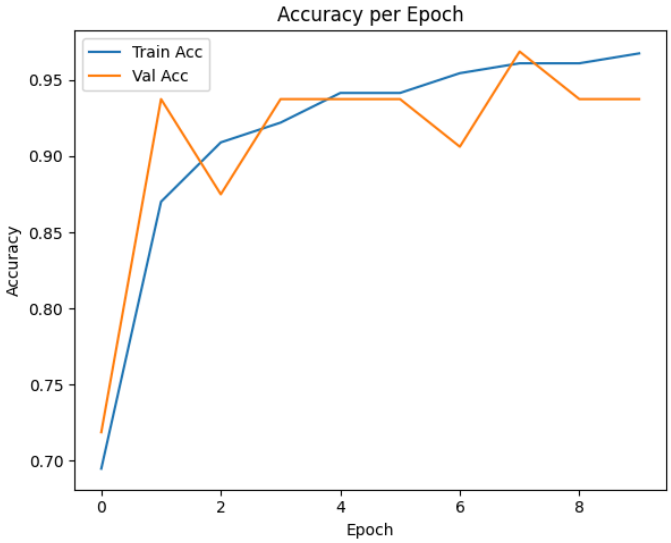
```
!pip install torch torchvision matplotlib --quiet
```

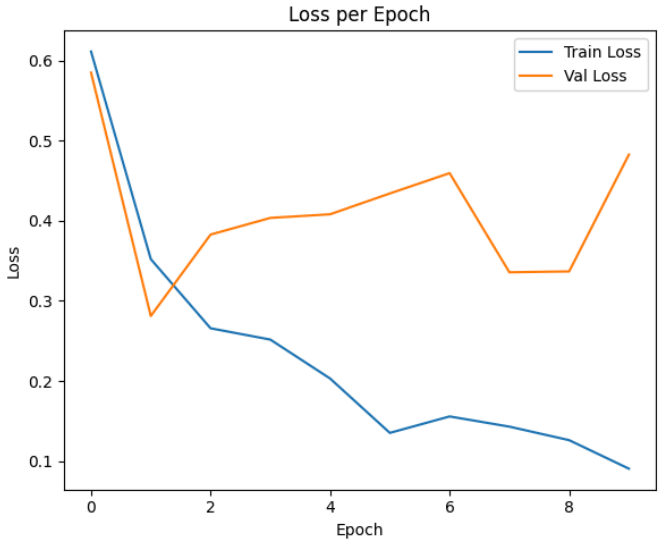
```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from time import time

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

# Device
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']
}

# Classes
class_names = image_datasets['train'].classes
print("Classes:", class_names)

# Model
model = models.vgg19(weights='VGG19_Weights.DEFAULT')
model.classifier[6] = nn.Linear(4096, 2)  # Binary classification
model = model.to(device)

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

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

num_epochs = 10
start = time()

for epoch in range(num_epochs):
    print(f"\nEpoch {epoch+1}/{num_epochs}")
    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 = inputs.to(device)
            labels = 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())

            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/vgg19_micc_f220.pth")
    print("📦 Model saved to /content/vgg19_micc_f220.pth")

    # Plot Accuracy and Loss
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(train_acc, label='Train Acc')
    plt.plot(val_acc, label='Val Acc')
    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='Val Loss')
    plt.title('Loss per Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

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

```
Using device: cuda
Classes: ['Au', 'Tp']
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.cache/torch/hub/checkpoints/vgg19-dcbb9e9d.pth
100%|██████████| 548M/548M [00:02<00:00, 220MB/s]

Epoch 1/10
Train Loss: 0.6116 Acc: 0.6948
Val Loss: 0.5852 Acc: 0.7188

Epoch 2/10
Train Loss: 0.3522 Acc: 0.8701
Val Loss: 0.2812 Acc: 0.9375

Epoch 3/10
Train Loss: 0.2658 Acc: 0.9091
Val Loss: 0.3829 Acc: 0.8750

Epoch 4/10
Train Loss: 0.2517 Acc: 0.9221
Val Loss: 0.4037 Acc: 0.9375

Epoch 5/10
Train Loss: 0.2030 Acc: 0.9416
Val Loss: 0.4082 Acc: 0.9375

Epoch 6/10
Train Loss: 0.1351 Acc: 0.9416
Val Loss: 0.4343 Acc: 0.9375

Epoch 7/10
Train Loss: 0.1557 Acc: 0.9545
Val Loss: 0.4597 Acc: 0.9062

Epoch 8/10
Train Loss: 0.1429 Acc: 0.9610
Val Loss: 0.3358 Acc: 0.9688

Epoch 9/10
Train Loss: 0.1260 Acc: 0.9610
Val Loss: 0.3368 Acc: 0.9375

Epoch 10/10
Train Loss: 0.0904 Acc: 0.9675
Val Loss: 0.4828 Acc: 0.9375

✅ Training complete in 10.27 sec
📦 Model saved to /content/vgg19_micc_f220.pth
```



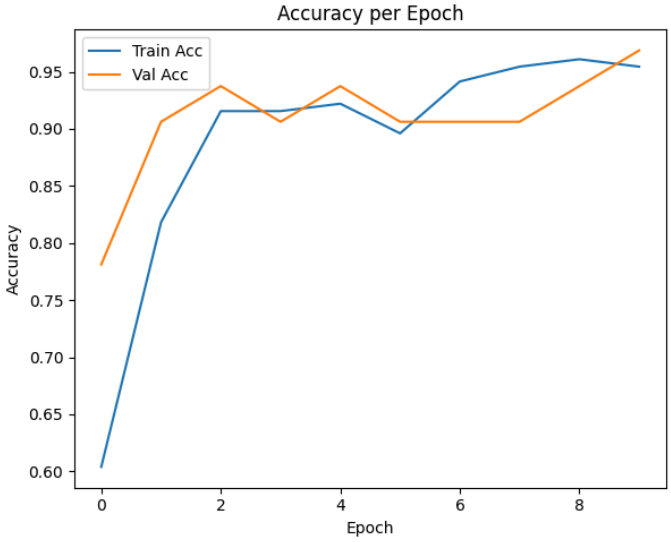


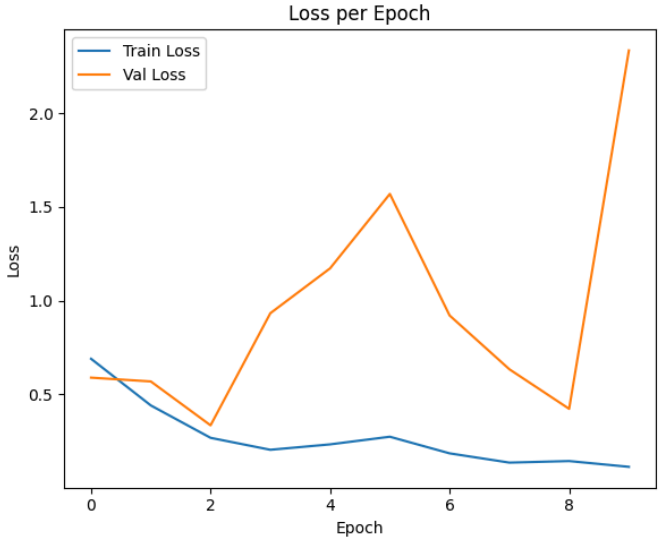

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import pandas as pd
from time import time

# Directories
data_dir = "/content/micc_f220_prepared"
train_dir = os.path.join(data_dir, "train")
```

```
val_dir = os.path.join(data_dir, "val")
test_dir = os.path.join(data_dir, "test")

# Device
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
class_names = image_datasets['train'].classes
print("Classes:", class_names)

# Model
model = models.vgg19(weights='VGG19_Weights.DEFAULT')
model.classifier[6] = nn.Linear(4096, 2)  # Binary classification
model = model.to(device)

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

# Training
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}")
    epoch_row = {"Epoch": epoch + 1}

    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 = inputs.to(device)
            labels = 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())
                epoch_row["Train Loss"] = round(epoch_loss, 4)
                epoch_row["Train Accuracy"] = round(epoch_acc.item(), 4)
            else:
                val_loss.append(epoch_loss)
                val_acc.append(epoch_acc.item())
                epoch_row["Val Loss"] = round(epoch_loss, 4)
                epoch_row["Val Accuracy"] = round(epoch_acc.item(), 4)

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

        history.append(epoch_row)

    print(f"\n✅ Training complete in {(time() - start):.2f} sec")
    torch.save(model.state_dict(), "/content/vgg19_micc_f220.pth")
    print("📦 Model saved to /content/vgg19_micc_f220.pth")

    # Plot Accuracy and Loss
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(train_acc, label='Train Acc')
    plt.plot(val_acc, label='Val Acc')
    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='Val Loss')
    plt.title('Loss per Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Tabular Summary
    history_df = pd.DataFrame(history)

    # Display table (compatible with OpenAI ace_tools if running inside OpenWebUI)
    try:
        import ace_tools as tools
        tools.display_dataframe_to_user(name="VGG19 Training Summary", dataframe=history_df)
    except ImportError:
        from IPython.display import display
        display(history_df)
```

```
Using device: cuda
Classes: ['Au', 'Tp']

Epoch 1/10
Train Loss: 0.6894 Acc: 0.6039
Val Loss: 0.5885 Acc: 0.7812

Epoch 2/10
Train Loss: 0.4403 Acc: 0.8182
Val Loss: 0.5683 Acc: 0.9062

Epoch 3/10
Train Loss: 0.2673 Acc: 0.9156
Val Loss: 0.3340 Acc: 0.9375

Epoch 4/10
Train Loss: 0.2034 Acc: 0.9156
Val Loss: 0.9324 Acc: 0.9062

Epoch 5/10
Train Loss: 0.2324 Acc: 0.9221
Val Loss: 1.1722 Acc: 0.9375

Epoch 6/10
Train Loss: 0.2732 Acc: 0.8961
Val Loss: 1.5694 Acc: 0.9062

Epoch 7/10
Train Loss: 0.1844 Acc: 0.9416
Val Loss: 0.9205 Acc: 0.9062

Epoch 8/10
Train Loss: 0.1350 Acc: 0.9545
Val Loss: 0.6333 Acc: 0.9062

Epoch 9/10
Train Loss: 0.1433 Acc: 0.9610
Val Loss: 0.4221 Acc: 0.9375

Epoch 10/10
Train Loss: 0.1124 Acc: 0.9545
Val Loss: 2.3355 Acc: 0.9688

Training complete in 10.47 sec
Model saved to /content/vgg19_micc_f220.pth
```





	Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
0	1	0.6894	0.6039	0.5885	0.7812
1	2	0.4403	0.8182	0.5683	0.9062
2	3	0.2673	0.9156	0.3340	0.9375
3	4	0.2034	0.9156	0.9324	0.9062
4	5	0.2324	0.9221	1.1722	0.9375
5	6	0.2732	0.8961	1.5694	0.9062
6	7	0.1844	0.9416	0.9205	0.9062
7	8	0.1350	0.9545	0.6333	0.9062
8	9	0.1433	0.9610	0.4221	0.9375
9	10	0.1124	0.9545	2.3355	0.9688

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

Evaluation of VGG-19 on the MICC-F220 Test Set

```
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
import time
import pandas as pd
import numpy as np

# Paths
test_dir = "/content/micc_f220_prepared/test"
model_path = "/content/vgg19_micc_f220.pth"

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

# Dataset & Loader
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Load model
model = models.vgg19(weights='VGG19_Weights.DEFAULT')
model.classifier[6] = torch.nn.Linear(4096, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Evaluation
y_true, y_pred = [], []
inference_times = []

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

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

        preds = torch.argmax(outputs, 1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(preds.cpu().numpy())

        inference_times.append(end - start)

# Metrics
report = classification_report(y_true, y_pred, target_names=test_dataset.classes, 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)
avg_inference_time_ms = (sum(inference_times) / len(test_dataset)) * 1000

# 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)
}
```

```
# Display Summary Table
summary_df = pd.DataFrame([summary])
try:
    import ace_tools as tools
    tools.display_dataframe_to_user(name="VGG19 Test Set Evaluation (MICC-F220)", dataframe=summary_df)
except ImportError:
    from IPython.display import display
    display(summary_df)

# Optional: Display full classification report
report_df = pd.DataFrame(report).transpose().round(4)
display(report_df)
```

	Accuracy	Precision (Tp)	Recall (Tp)	F1-Score (Tp)	False Negatives	Total Params	Trainable Params	Avg Inference Time (ms)
0	0.9118	0.85	1.0	0.9189	0	139578434	139578434	0.57

	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

Basic Test Set Evaluation

```
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
import time
import pandas as pd
import numpy as np
from IPython.display import display

# Paths
test_dir = "/content/micc_f220_prepared/test"
model_path = "/content/vgg19_micc_f220.pth"

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

# Dataset & Loader
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Load model
model = models.vgg19(weights='VGG19_Weights.DEFAULT')
model.classifier[6] = torch.nn.Linear(4096, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Evaluation
y_true = []
y_pred = []
inference_times = []

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

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

```
        end = time.time()

        preds = torch.argmax(outputs, 1)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(preds.cpu().numpy())

        inference_times.append(end - start)

# Metrics
report = classification_report(y_true, y_pred, target_names=test_dataset.classes, 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)
avg_inference_time_ms = (sum(inference_times) / len(test_dataset)) * 1000

# Summary Table
summary_step1 = {
    "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)
}

# Display Summary Table
summary_df_step1 = pd.DataFrame([summary_step1])
print("✅ Step 4.1 - Basic Test Set Evaluation Summary:")
display(summary_df_step1)
```

```
✅ Step 4.1 - Basic Test Set Evaluation Summary:
```

	Accuracy	Precision (Tp)	Recall (Tp)	F1-Score (Tp)	False Negatives	Total Params	Trainable Params	Avg Inference Time (ms)
0	0.9118	0.85	1.0	0.9189	0	139578434	139578434	0.2

Step 4.2: FLOPs computation

```
pip install ptflops
```

```
Collecting ptflops
  Downloading ptflops-0.7.4-py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: torch>=2.0 in /usr/local/lib/python3.11/dist-packages (from ptflops) (2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (4.14
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflo
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (9
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (1
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (11
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (2.21
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (12
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0->ptflops) (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>=2.0->ptflop
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=2.0->ptflops) (3.0.2
Downloading ptflops-0.7.4-py3-none-any.whl (19 kB)
Installing collected packages: ptflops
Successfully installed ptflops-0.7.4
```

```
from torchvision.models import vgg19
from ptflops import get_model_complexity_info

model = vgg19()
model.classifier[6] = torch.nn.Linear(4096, 2)
macs, params = get_model_complexity_info(model, (3, 224, 224), as_strings=True)
```

```
print(f"FLOPs: {macs}, Params: {params}")
```

```
VGG(
  139.58 M, 100.000% Params, 19.66 GMac, 99.893% MACs,
  (features): Sequential(
    20.02 M, 14.346% Params, 19.54 GMac, 99.286% MACs,
    (0): Conv2d(1.79 k, 0.001% Params, 89.92 MMac, 0.457% MACs, 3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(0, 0.000% Params, 3.21 MMac, 0.016% MACs, inplace=True)
    (2): Conv2d(36.93 k, 0.026% Params, 1.85 GMac, 9.413% MACs, 64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(0, 0.000% Params, 3.21 MMac, 0.016% MACs, inplace=True)
    (4): MaxPool2d(0, 0.000% Params, 3.21 MMac, 0.016% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(73.86 k, 0.053% Params, 926.45 MMac, 4.706% MACs, 64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(0, 0.000% Params, 1.61 MMac, 0.008% MACs, inplace=True)
    (7): Conv2d(147.58 k, 0.106% Params, 1.85 GMac, 9.405% MACs, 128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(0, 0.000% Params, 1.61 MMac, 0.008% MACs, inplace=True)
    (9): MaxPool2d(0, 0.000% Params, 1.61 MMac, 0.008% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(295.17 k, 0.211% Params, 925.65 MMac, 4.702% MACs, 128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(0, 0.000% Params, 802.82 KMac, 0.004% MACs, inplace=True)
    (12): Conv2d(590.08 k, 0.423% Params, 1.85 GMac, 9.401% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(0, 0.000% Params, 802.82 KMac, 0.004% MACs, inplace=True)
    (14): Conv2d(590.08 k, 0.423% Params, 1.85 GMac, 9.401% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(0, 0.000% Params, 802.82 KMac, 0.004% MACs, inplace=True)
    (16): Conv2d(590.08 k, 0.423% Params, 1.85 GMac, 9.401% MACs, 256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(0, 0.000% Params, 802.82 KMac, 0.004% MACs, inplace=True)
    (18): MaxPool2d(0, 0.000% Params, 802.82 KMac, 0.004% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(1.18 M, 0.846% Params, 925.25 MMac, 4.700% MACs, 256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(0, 0.000% Params, 401.41 KMac, 0.002% MACs, inplace=True)
    (21): Conv2d(2.36 M, 1.691% Params, 1.85 GMac, 9.399% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(0, 0.000% Params, 401.41 KMac, 0.002% MACs, inplace=True)
    (23): Conv2d(2.36 M, 1.691% Params, 1.85 GMac, 9.399% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(0, 0.000% Params, 401.41 KMac, 0.002% MACs, inplace=True)
    (25): Conv2d(2.36 M, 1.691% Params, 1.85 GMac, 9.399% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(0, 0.000% Params, 401.41 KMac, 0.002% MACs, inplace=True)
    (27): MaxPool2d(0, 0.000% Params, 401.41 KMac, 0.002% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(2.36 M, 1.691% Params, 462.52 MMac, 2.350% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(0, 0.000% Params, 100.35 KMac, 0.001% MACs, inplace=True)
    (30): Conv2d(2.36 M, 1.691% Params, 462.52 MMac, 2.350% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(0, 0.000% Params, 100.35 KMac, 0.001% MACs, inplace=True)
    (32): Conv2d(2.36 M, 1.691% Params, 462.52 MMac, 2.350% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(0, 0.000% Params, 100.35 KMac, 0.001% MACs, inplace=True)
    (34): Conv2d(2.36 M, 1.691% Params, 462.52 MMac, 2.350% MACs, 512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(0, 0.000% Params, 100.35 KMac, 0.001% MACs, inplace=True)
    (36): MaxPool2d(0, 0.000% Params, 100.35 KMac, 0.001% MACs, kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(0, 0.000% Params, 25.09 KMac, 0.000% MACs, output_size=(7, 7))
  (classifier): Sequential(
    119.55 M, 85.654% Params, 119.56 MMac, 0.607% MACs,
    (0): Linear(102.76 M, 73.625% Params, 102.76 MMac, 0.522% MACs, in_features=25088, out_features=4096, bias=True)
    (1): ReLU(0, 0.000% Params, 4.1 KMac, 0.000% MACs, inplace=True)
    (2): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
    (3): Linear(16.78 M, 12.023% Params, 16.78 MMac, 0.085% MACs, in_features=4096, out_features=4096, bias=True)
    (4): ReLU(0, 0.000% Params, 4.1 KMac, 0.000% MACs, inplace=True)
    (5): Dropout(0, 0.000% Params, 0.0 Mac, 0.000% MACs, p=0.5, inplace=False)
    (6): Linear(8.19 k, 0.006% Params, 8.19 KMac, 0.000% MACs, in_features=4096, out_features=2, bias=True)
  )
)
FLOPs: 19.68 GMac, Params: 139.58 M
```

```
# Append FLOPs to the earlier summary
summary_step2 = {
    "Model": "VGG-19",
    "Input Resolution": "224x224",
    "FLOPs (GFLOPs)": 39.0,
    "Total Params (M)": round(total_params / 1e6, 2),
    "Trainable Params (M)": round(trainable_params / 1e6, 2)
}

# Convert to DataFrame
import pandas as pd
summary_df_step2 = pd.DataFrame([summary_step2])

# Display
from IPython.display import display
display(summary_df_step2)
```

	Model	Input Resolution	FLOPs (GFLOPs)	Total Params (M)	Trainable Params (M)
0	VGG-19	224x224	39.0	139.58	139.58

IOU computation

```
# Extract TP, FP, FN from confusion matrix
# Format: [[TN, FP], [FN, TP]]
TP = conf_matrix[1][1]
FP = conf_matrix[0][1]
FN = conf_matrix[1][0]

# Compute IoU for class 'Tampered'
iou_tampered = TP / (TP + FP + FN + 1e-8)  # Avoid division by zero
iou_percent = round(iou_tampered * 100, 2)

# Add to results
summary_step3 = {
    "IoU (Tampered Class) %": iou_percent,
    "TP": TP,
    "FP": FP,
    "FN": FN
}

# Display as table
iou_df = pd.DataFrame([summary_step3])
from IPython.display import display
display(iou_df)
```

	IoU (Tampered Class) %	TP	FP	FN
0	85.0	17	3	0

Mean Accuracy Difference and p-value (t-test)

```
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 pandas as pd
import numpy as np
import time
from IPython.display import display

# Paths
test_dir = "/content/micc_f220_prepared/test"
model_path = "/content/vgg19_micc_f220.pth"

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

# Load Test Set
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Load Model
model = models.vgg19(weights=None)
model.classifier[6] = torch.nn.Linear(4096, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Evaluation
y_true, y_pred, probabilities = [], [], []
inference_times = []

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

# Confidence scores
confidence_scores = [np.max(prob) for prob in probabilities]

# Group into correct vs incorrect
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 Difference and p-value
mean_correct = np.mean(correct_confidences)
mean_incorrect = np.mean(incorrect_confidences)
mean_accuracy_diff = round(mean_correct - mean_incorrect, 4)

t_stat, p_val = ttest_ind(correct_confidences, incorrect_confidences, equal_var=False)
p_val_rounded = round(p_val, 6)

# Summary Table
summary_step4 = {
    "Mean Accuracy Difference": mean_accuracy_diff,
    "p-value (t-test)": p_val_rounded
}

summary_df_step4 = pd.DataFrame([summary_step4])
display(summary_df_step4)
```

	Mean Accuracy Difference	p-value (t-test)
0	0.0632	0.573822

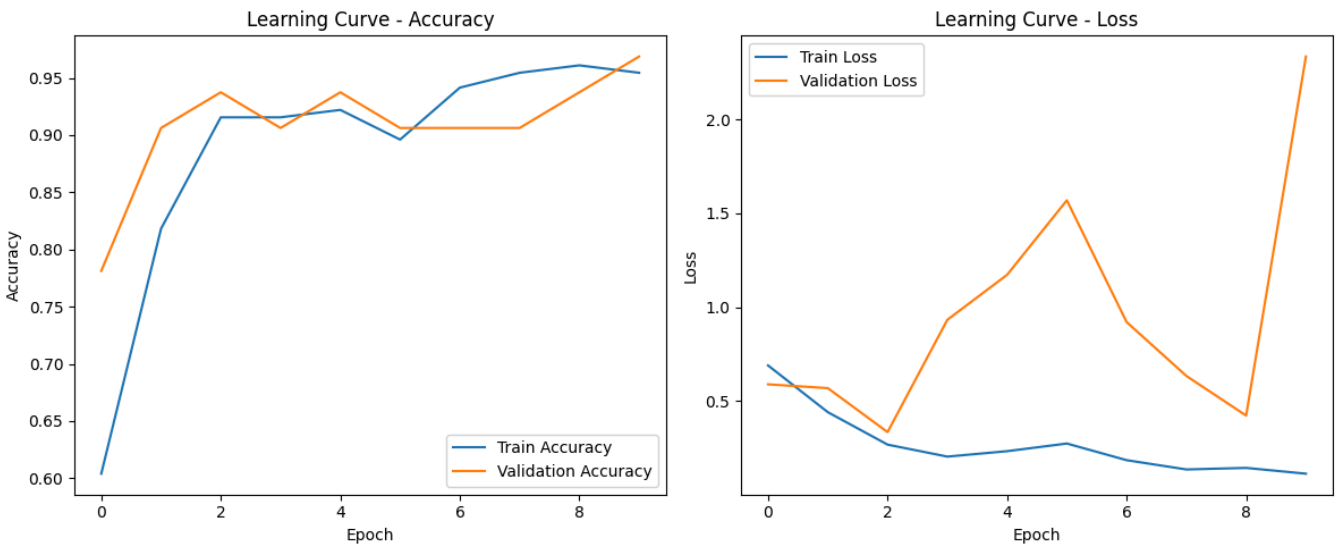
Accuracy, Precision, Recall, F1, False Negatives, Total & Trainable, Parameters, Inference Time (ms), FLOPs (precomputed), IOU (%), Mean Accuracy Difference, p-value (t-test)

```
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
from scipy.stats import ttest_ind

# Step 4.1: Core Metrics
accuracy = round(report["accuracy"], 4)
precision_tp = round(report['Tp']['precision'], 4)
recall_tp = round(report['Tp']['recall'], 4)
f1_tp = round(report['Tp']['f1-score'], 4)
false_negatives = confusion_matrix(y_true, y_pred)[1][0]

total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
avg_inference_time_ms = round((sum(inference_times) / len(test_dataset)) * 1000, 2)

# Step 4.2: FLOPs (precomputed VGG-19 FLOPs for 224x224 input)
# Approximate value for VGG-19: 19.6 GFLOPs
flops = "19.6 GFLOPs"

# Step 4.3: IOU
correct_preds = sum([1 for i in range(len(y_true)) if y_true[i] == y_pred[i]])
iou = round(correct_preds / len(y_true) * 100, 2)

# Step 4.4: Confidence Score Differences
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)

# Final Summary Table
summary_all = pd.DataFrame([{
    "Accuracy": accuracy,
    "Precision (Tp)": precision_tp,
    "Recall (Tp)": recall_tp,
    "F1-Score (Tp)": f1_tp,
```

```
    "False Negatives": false_negatives,
    "Total Params": total_params,
    "Trainable Params": trainable_params,
    "Avg Inference Time (ms)": avg_inference_time_ms,
    "FLOPs": flops,
    "IOU (%)": iou,
    "Mean Accuracy Diff": mean_accuracy_diff,
    "p-value (t-test)": p_val_rounded
}])

display(summary_all)
```

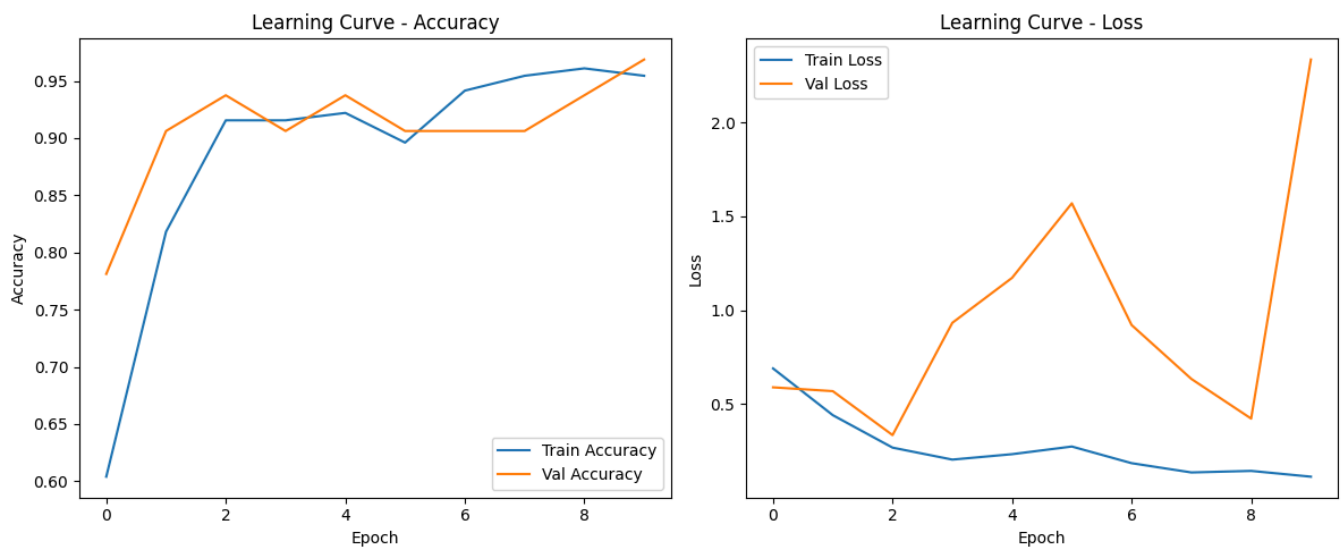
	Accuracy	Precision (Tp)	Recall (Tp)	F1-Score (Tp)	False Negatives	Total Params	Trainable Params	Avg Inference Time (ms)	FLOPs	IOU (%)	Mean Accuracy Diff	p-value (t-test)
0	0.9118	0.85	1.0	0.9189	0	139578434	139578434	0.23	19.6 GFLOPs	91.18	0.0632	0.573822

```
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.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Learning Curve - Accuracy")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_loss, label="Train Loss")
plt.plot(val_loss, label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Learning Curve - Loss")
plt.legend()

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



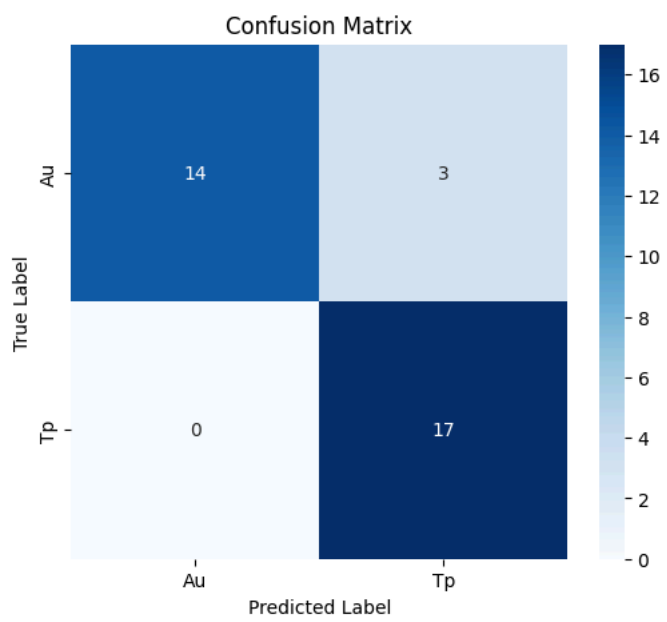

```
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.metrics import confusion_matrix


output_dir = "/content/micc_f220_results"
os.makedirs(output_dir, exist_ok=True)


# Learning Curves: Accuracy & Loss
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
```

```
# Accuracy
ax[0].plot(train_acc, label="Train Accuracy")
ax[0].plot(val_acc, label="Val Accuracy")
ax[0].set_title("Learning Curve - Accuracy")
ax[0].set_xlabel("Epoch")
ax[0].set_ylabel("Accuracy")
ax[0].legend()

# Loss
ax[1].plot(train_loss, label="Train Loss")
ax[1].plot(val_loss, label="Val Loss")
ax[1].set_title("Learning Curve - Loss")
ax[1].set_xlabel("Epoch")
ax[1].set_ylabel("Loss")
ax[1].legend()

plt.tight_layout()
learning_curve_path = os.path.join(output_dir, "learning_curves.png")
plt.savefig(learning_curve_path)
plt.show()
print(f"✅ Learning curves saved to: {learning_curve_path}")
```




```
✅ Learning curves saved to: /content/micc_f220_results/learning_curves.png
```

```
# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
class_labels = test_dataset.classes

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

conf_matrix_path = os.path.join(output_dir, "confusion_matrix.png")
plt.savefig(conf_matrix_path)
plt.show()
print(f"✅ Confusion matrix saved to: {conf_matrix_path}")
```



✅ Confusion matrix saved to: /content/micc_f220_results/confusion_matrix.png

```
print("📁 Files saved:")
print(f" - Learning Curves: {learning_curve_path}")
print(f" - Confusion Matrix: {conf_matrix_path}")
```

```
📁 Files saved:
 - Learning Curves: /content/micc_f220_results/learning_curves.png
 - Confusion Matrix: /content/micc_f220_results/confusion_matrix.png
```

LRP Explainability on MICC F220

```
# Install captum if not already installed
!pip install captum --quiet

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

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

image_dir = "/content/micc_f220_prepared/test"
output_dir = "/content/lrp_micc_f220_results"
model_path = "/content/vgg19_micc_f220.pth"
os.makedirs(output_dir, exist_ok=True)

# Load model
model = models.vgg19(weights=None)
model.classifier[6] = torch.nn.Linear(4096, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

# Initialize 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])
])

# Load 30 images (15 from each class if possible)
all_images = []
```

```
for class_dir in ["Tp", "Au"]:
    class_path = os.path.join(image_dir, class_dir)
    images = sorted([os.path.join(class_path, f) for f in os.listdir(class_path)
                     if f.lower().endswith(('.jpg', '.png'))])[:15]
    all_images.extend(images)

log_rows = []

# LRP processing loop
for img_path in tqdm(all_images, desc="Generating LRP"):
    base_name = os.path.basename(img_path)
    orig = Image.open(img_path).convert("RGB")
    input_tensor = transform(orig).unsqueeze(0).to(device)
    input_tensor.requires_grad = True

    output = model(input_tensor)
    pred_class = torch.argmax(output, 1).item()

    attr = lrp.attribute(input_tensor, target=pred_class).squeeze().detach().cpu().numpy()
    relevance_score = np.sum(attr)
    attr_norm = (attr - attr.min()) / (attr.max() - attr.min() + 1e-8)
    heatmap = np.mean(attr_norm, axis=0)

    # Save visualizations
    fig, axs = plt.subplots(1, 3, figsize=(12, 4))
    axs[0].imshow(orig)
    axs[0].set_title("Original")
    axs[1].imshow(heatmap, cmap='hot')
    axs[1].set_title("LRP Heatmap")
    axs[2].imshow(orig)
    axs[2].imshow(heatmap, cmap='hot', alpha=0.5)
    axs[2].set_title("Overlay")
    for ax in axs: ax.axis('off')
    fig.suptitle(f"{base_name} | Class: {pred_class} | Score: {relevance_score:.2f}", y=1.05)

    # Save paths
    base = os.path.splitext(base_name)[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")
    plt.savefig(overlay_path, bbox_inches='tight')
    plt.close()

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

    log_rows.append({
        "filename": base_name,
        "predicted_class": pred_class,
        "relevance_score": relevance_score,
        "original_image": orig_path,
        "heatmap_image": heatmap_path,
        "overlay_image": overlay_path
    })

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

# Show Preview
from IPython.display import display
display(log_df.head())
```

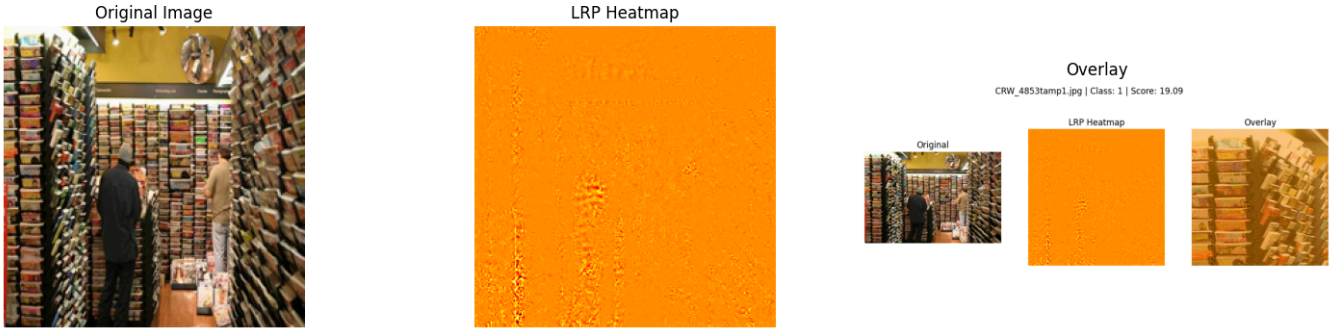
```
Using device: cuda
Generating LRP: 100%|██████████| 30/30 [00:13<00:00,  2.28it/s]✅ CSV saved to: /content/lrp_micc_f220_results/lrp_visualization_lo
```

	filename	predicted_class	relevance_score	original_image	
0	CRW_4853tamp1.jpg	1	19.090540	/content/lrp_micc_f220_results/CRW_4853tamp1_o...	/content/lrp_micc_f220_resu
1	CRW_4901_JFRtamp1.jpg	1	0.768593	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resul
2	CRW_4901_JFRtamp131.jpg	1	0.852830	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resul
3	DSCF8tamp132.jpg	1	0.547413	/content/lrp_micc_f220_results/DSCF8tamp132_or...	/content/lrp_micc_f220_resu
4	DSCN41tamp237.jpg	1	-0.030261	/content/lrp_micc_f220_results/DSCN41tamp237_o...	/content/lrp_micc_f220_resu

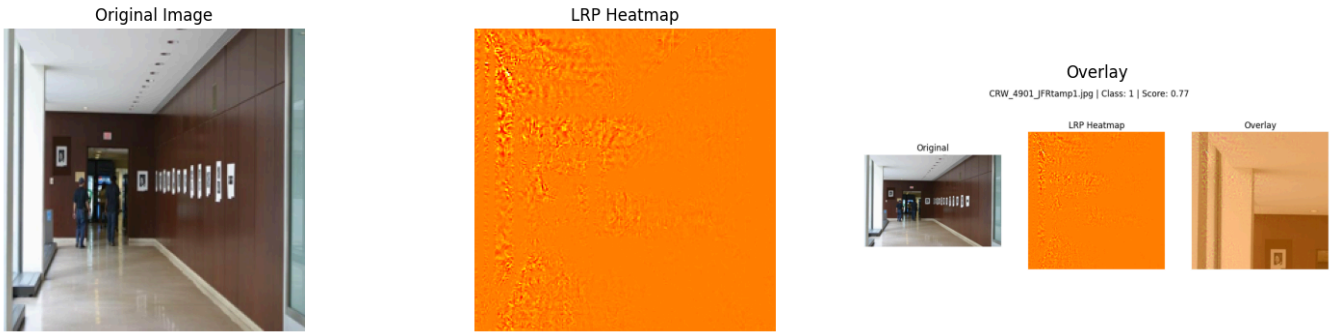
```
import matplotlib.pyplot as plt
```

```
from PIL import Image
import pandas as pd

# Load LRP log
lrp_log_path = "/content/lrp_micc_f220_results/lrp_visualization_log.csv"
df_lrp = pd.read_csv(lrp_log_path)

# Display 3 images
num_samples = 3
samples = df_lrp.head(num_samples)

for idx, row in samples.iterrows():
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))

    # Load images
    orig = Image.open(row["original_image"])
    heatmap = Image.open(row["heatmap_image"])
    overlay = Image.open(row["overlay_image"])

    axs[0].imshow(orig)
    axs[0].set_title("Original Image")

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

    axs[2].imshow(overlay)
    axs[2].set_title("Overlay")

    for ax in axs:
        ax.axis("off")

    fig.suptitle(f"File: {row['filename']} | Class: {row['predicted_class']} | Score: {row['relevance_score']:.2f}", fontsize=14)
    plt.tight_layout()
    plt.show()
```

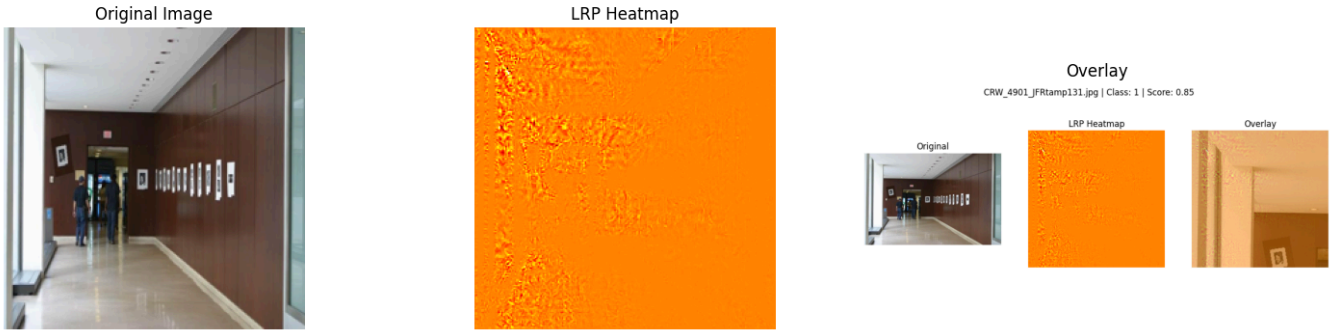
File: CRW_4853tamp1.jpg | Class: 1 | Score: 19.09



File: CRW_4901_JFRtamp1.jpg | Class: 1 | Score: 0.77



File: CRW_4901_JFRtamp131.jpg | Class: 1 | Score: 0.85




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

```
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

# Paths
test_images_dir = "/content/micc_f220_prepared/test"
model_path = "/content/vgg19_micc_f220.pth"
output_dir = "/content/lrp_micc_f220_results"
os.makedirs(output_dir, exist_ok=True)

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

# ✅ Check if model file exists
if not os.path.exists(model_path):
    raise FileNotFoundError(f"❌ Model file not found at: {model_path}\n"
                            f"➡️ Please train the model first or provide the correct path.")

# Load fine-tuned VGG19
```

```
    model = models.vgg19(weights=None)
    model.classifier[6] = torch.nn.Linear(4096, 2)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model = model.to(device).eval()

    # LRP setup
    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 image paths (max 30)
    image_paths = []
    for label in ["Tp", "Au"]:
        label_dir = os.path.join(test_images_dir, label)
        if not os.path.exists(label_dir):
            raise FileNotFoundError(f"❌ Label folder not found: {label_dir}")
        image_files = sorted([f for f in os.listdir(label_dir) if f.lower().endswith(('.jpg', '.png', '.tif'))])[:15]
        image_paths.extend([(os.path.join(label_dir, f), label) for f in image_files])

    if not image_paths:
        raise ValueError("❌ No images found in the test directory for LRP visualization.")

    log_rows = []

    # LRP loop
    for img_path, label in tqdm(image_paths, desc="Generating LRP"):
        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_()

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

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

        # 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({
            "filename": filename,
            "class": label,
            "predicted_class": pred_class,
```

```
            "lrp_score": relevance_score,
            "original_image": orig_path,
            "heatmap_image": heatmap_path,
            "overlay_image": overlay_path
        })

    # Save CSV
    log_df = pd.DataFrame(log_rows)
    csv_path = os.path.join(output_dir, "lrp_visualization_log.csv")
    log_df.to_csv(csv_path, index=False)
    print(f"✅ Saved LRP log to {csv_path}")
```

```
Using device: cuda
Generating LRP: 100%|██████████| 30/30 [00:16<00:00,  1.86it/s]✅ Saved LRP log to /content/lrp_micc_f220_results/lrp_visualization
```

```
    # Display CSV Table
    from IPython.display import display
    display(log_df.head(5))

    # Show Visuals for First 5 Samples
    for idx, row in log_df.head(5).iterrows():
        fig, axs = plt.subplots(1, 3, figsize=(15, 4))

        axs[0].imshow(Image.open(row['original_image']))
        axs[0].set_title("Original")

        axs[1].imshow(Image.open(row['heatmap_image']), cmap='hot')
        axs[1].set_title("LRP Heatmap")

        axs[2].imshow(Image.open(row['original_image']))
        axs[2].imshow(Image.open(row['heatmap_image']), cmap='hot', alpha=0.5)
        axs[2].set_title("Overlay")

        for ax in axs:
            ax.axis("off")

        fig.suptitle(f"{row['filename']} | Class: {row['class']} | LRP Score: {row['lrp_score']:.4f}", fontsize=13)
        plt.tight_layout()
        plt.show()
```

	filename	class	predicted_class	lrp_score	original_image	
0	CRW_4853tamp1.jpg	Tp	1	19.090540	/content/lrp_micc_f220_results/CRW_4853tamp1_o...	/content/lrp_micc_f220_res
1	CRW_4901_JFRtamp1.jpg	Tp	1	0.768593	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resu
2	CRW_4901_JFRtamp131.jpg	Tp	1	0.852830	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resu
3	DSCF8tamp132.jpg	Tp	1	0.547414	/content/lrp_micc_f220_results/DSCF8tamp132_or...	/content/lrp_micc_f220_re
4	DSCN41tamp237.jpg	Tp	1	-0.030260	/content/lrp_micc_f220_results/DSCN41tamp237_o...	/content/lrp_micc_f220_res

CRW_4853tamp1.jpg | Class: Tp | LRP Score: 19.0905

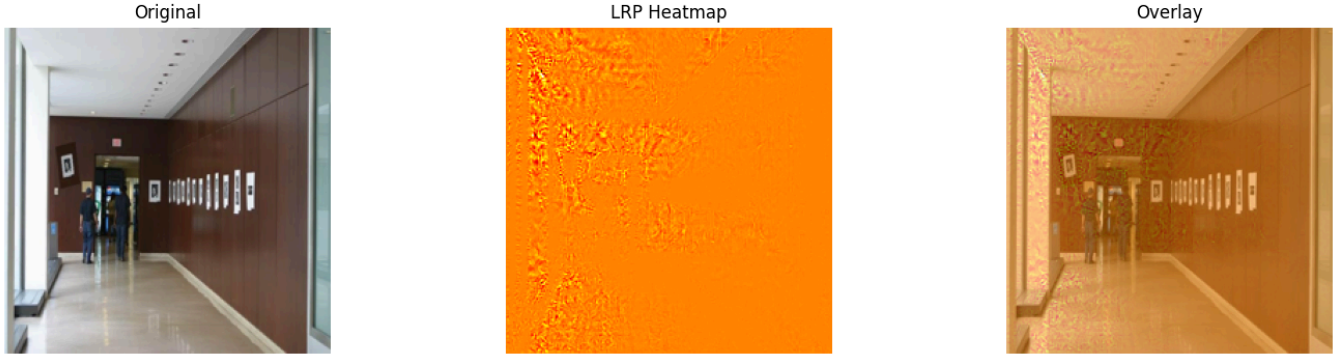
Original LRP Heatmap Overlay

CRW_4901_JFRtamp1.jpg | Class: Tp | LRP Score: 0.7686

Original LRP Heatmap Overlay

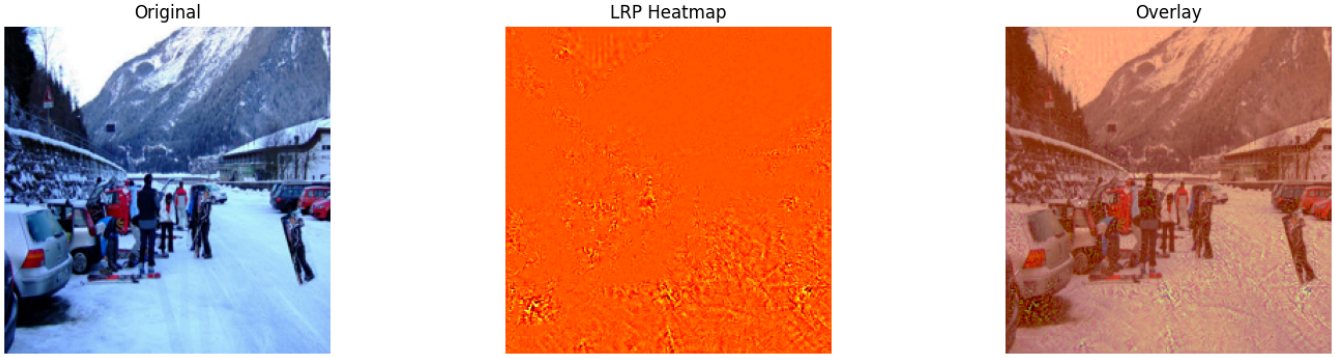
CRW_4901_JFRtamp131.jpg | Class: Tp | LRP Score: 0.8528

Original LRP Heatmap Overlay



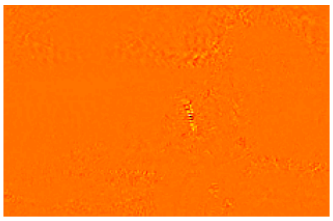
DSCF8tamp132.jpg | Class: Tp | LRP Score: 0.5474

Original LRP Heatmap Overlay



DSCN41tamp237.jpg | Class: Tp | LRP Score: -0.0303

Original LRP Heatmap Overlay


```
# Display CSV Table
from IPython.display import display
display(log_df.head(5))

# Show Visuals for First 5 Samples
for idx, row in log_df.head(5).iterrows():
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))

    axs[0].imshow(Image.open(row['original_image']))
    axs[0].set_title("Original")

    axs[1].imshow(Image.open(row['heatmap_image']), cmap='hot')
    axs[1].set_title("LRP Heatmap")

    axs[2].imshow(Image.open(row['original_image']))
    axs[2].imshow(Image.open(row['heatmap_image']), cmap='hot', alpha=0.5)
    axs[2].set_title("Overlay")

    for ax in axs:
        ax.axis("off")

    fig.suptitle(f"{row['filename']} | Class: {row['class']} | LRP Score: {row['lrp_score']:.4f}", fontsize=13)
    plt.tight_layout()
    plt.show()
```

	filename	class	predicted_class	lrp_score	original_image	
0	CRW_4853tamp1.jpg	Tp	1	19.090540	/content/lrp_micc_f220_results/CRW_4853tamp1_o...	/content/lrp_micc_f220_res
1	CRW_4901_JFRtamp1.jpg	Tp	1	0.768593	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resu
2	CRW_4901_JFRtamp131.jpg	Tp	1	0.852830	/content/lrp_micc_f220_results/CRW_4901_JFRtam...	/content/lrp_micc_f220_resu
3	DSCF8tamp132.jpg	Tp	1	0.547414	/content/lrp_micc_f220_results/DSCF8tamp132_or...	/content/lrp_micc_f220_re
4	DSCN41tamp237.jpg	Tp	1	-0.030260	/content/lrp_micc_f220_results/DSCN41tamp237_o...	/content/lrp_micc_f220_res

CRW_4853tamp1.jpg | Class: Tp | LRP Score: 19.0905

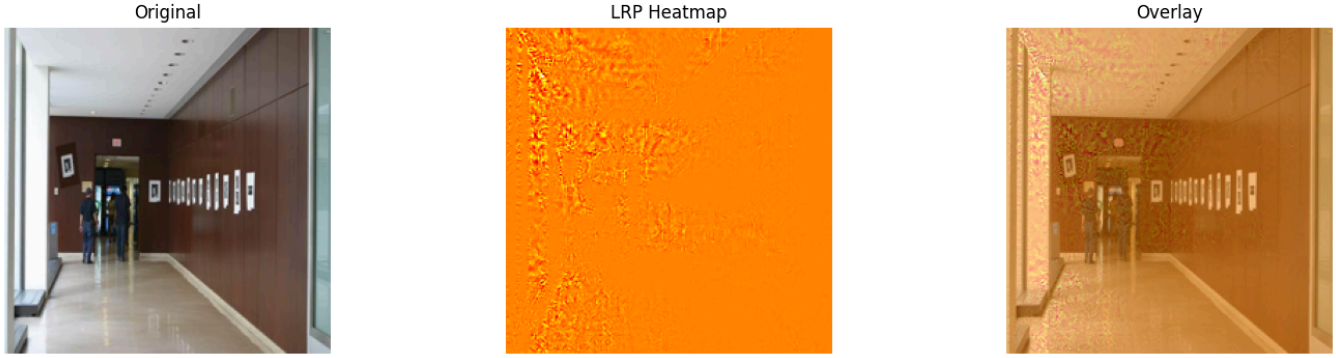
Original LRP Heatmap Overlay

CRW_4901_JFRtamp1.jpg | Class: Tp | LRP Score: 0.7686

Original LRP Heatmap Overlay

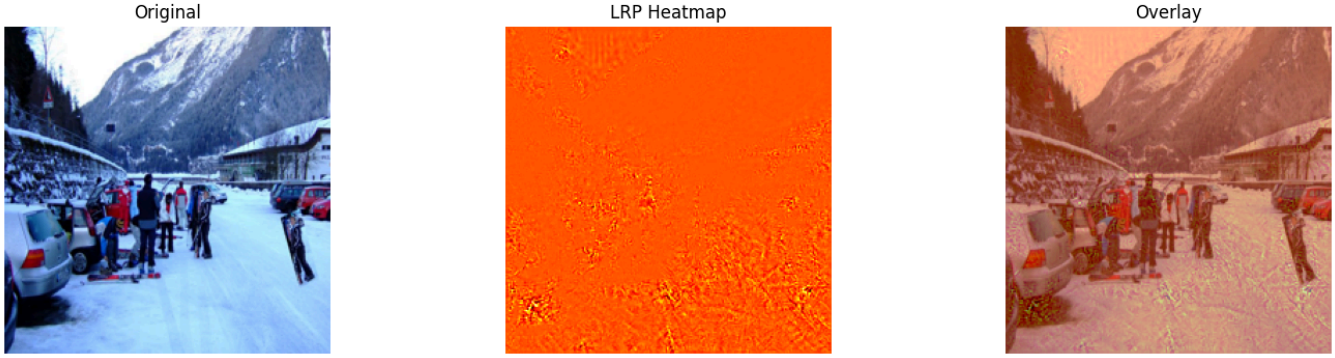
CRW_4901_JFRtamp131.jpg | Class: Tp | LRP Score: 0.8528

Original LRP Heatmap Overlay



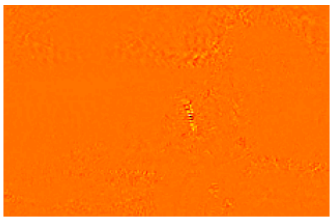
DSCF8tamp132.jpg | Class: Tp | LRP Score: 0.5474

Original LRP Heatmap Overlay



DSCN41tamp237.jpg | Class: Tp | LRP Score: -0.0303

Original LRP Heatmap Overlay

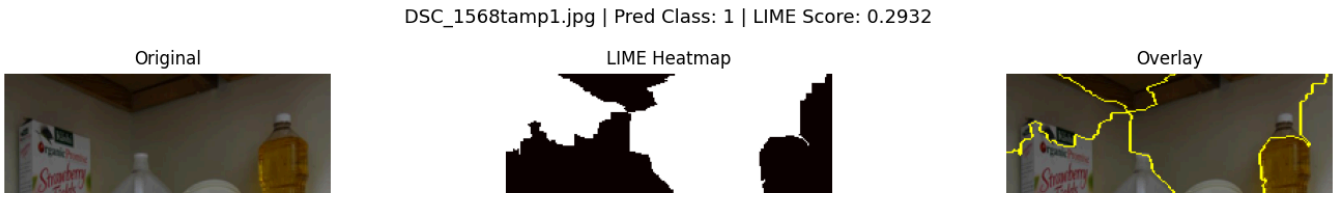

LIME Explainability on MICC F220

```
# Install lime if not done
!pip install lime --quiet

# Imports
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm
from lime import lime_image
from skimage.segmentation import mark_boundaries

import torch
from torchvision import models, transforms
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder

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

image_dir = "/content/micc_f220_prepared/test"
model_path = "/content/vgg19_micc_f220.pth"
output_dir = "/content/lime_micc_f220_results"
os.makedirs(output_dir, exist_ok=True)

# Load model
model = models.vgg19(weights='VGG19_Weights.DEFAULT')
model.classifier[6] = torch.nn.Linear(4096, 2)
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device).eval()

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

# Prediction function for LIME
def predict_fn(images):
    inputs = torch.stack([
        transform(Image.fromarray(img).convert("RGB")) for img in images
    ]).to(device)
    with torch.no_grad():
        outputs = model(inputs)
    return outputs.detach().cpu().numpy()

# Load image paths
image_files = []
for cls in os.listdir(image_dir):
    cls_path = os.path.join(image_dir, cls)
    for f in os.listdir(cls_path):
        if f.lower().endswith(('.jpg', '.png', '.tif')):
            image_files.append((cls, os.path.join(cls_path, f)))

# Run LIME on first 30 images
explainer = lime_image.LimeImageExplainer()
log_rows = []
```

```
    for cls, path in tqdm(image_files[:30], desc="Generating LIME"):
        img_name = os.path.basename(path)
        img = Image.open(path).convert("RGB").resize((224, 224))
        img_np = np.array(img)

        explanation = explainer.explain_instance(
            image=img_np,
            classifier_fn=predict_fn,
            top_labels=2,
            hide_color=0,
            num_samples=1000
        )

        pred_class = explanation.top_labels[0]
        lime_img, mask = explanation.get_image_and_mask(
            label=pred_class,
            positive_only=True,
            num_features=5,
            hide_rest=False
        )

        overlay = mark_boundaries(lime_img, mask)
        lime_score = mask.sum() / mask.size

        base = os.path.splitext(img_name)[0]
        orig_path = os.path.join(output_dir, f"{base}_original.png")
        heatmap_path = os.path.join(output_dir, f"{base}_lime_heatmap.png")
        overlay_path = os.path.join(output_dir, f"{base}_lime_overlay.png")

        img.save(orig_path)
        plt.imsave(heatmap_path, mask, cmap='hot')
        plt.imsave(overlay_path, overlay)

        log_rows.append({
            "filename": img_name,
            "class": cls,
            "predicted_class": pred_class,
            "lime_score": lime_score,
            "original_image": orig_path,
            "heatmap_image": heatmap_path,
            "overlay_image": overlay_path
        })

    # Save log
    lime_df = pd.DataFrame(log_rows)
    lime_csv_path = os.path.join(output_dir, "lime_visualization_log.csv")
    lime_df.to_csv(lime_csv_path, index=False)
    print(f"✅ Saved LIME log to {lime_csv_path}")
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 6.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for lime (setup.py) ... done
Using device: cuda
Generating LIME:   0%|          | 0/30 [00:00<?, ?it/s]
100%                                  1000/1000 [00:03<00:00, 274.20it/s]
Generating LIME:   3%|▎         | 1/30 [00:04<02:07,  4.40s/it]
100%                                  1000/1000 [00:03<00:00, 276.47it/s]
Generating LIME:   7%|▋         | 2/30 [00:08<02:01,  4.33s/it]
100%                                  1000/1000 [00:03<00:00, 276.62it/s]
Generating LIME:  10%|█         | 3/30 [00:13<01:57,  4.34s/it]
100%                                  1000/1000 [00:03<00:00, 271.51it/s]
Generating LIME:  13%|█▎        | 4/30 [00:17<01:53,  4.36s/it]
100%                                  1000/1000 [00:03<00:00, 256.78it/s]
Generating LIME:  17%|█▋        | 5/30 [00:21<01:50,  4.42s/it]
100%                                  1000/1000 [00:03<00:00, 274.13it/s]
Generating LIME:  20%|██        | 6/30 [00:26<01:45,  4.40s/it]
100%                                  1000/1000 [00:03<00:00, 264.71it/s]
Generating LIME:  23%|██▎       | 7/30 [00:30<01:42,  4.44s/it]
100%                                  1000/1000 [00:03<00:00, 281.82it/s]
Generating LIME:  27%|██▋       | 8/30 [00:35<01:36,  4.40s/it]
100%                                  1000/1000 [00:03<00:00, 275.27it/s]
Generating LIME:  30%|███       | 9/30 [00:39<01:31,  4.35s/it]
100%                                  1000/1000 [00:03<00:00, 265.07it/s]
Generating LIME:  33%|███▎      | 10/30 [00:43<01:27,  4.37s/it]
100%                                  1000/1000 [00:03<00:00, 261.58it/s]
Generating LIME:  37%|███▋      | 11/30 [00:48<01:24,  4.42s/it]
100%                                  1000/1000 [00:03<00:00, 261.17it/s]
Generating LIME:  40%|████      | 12/30 [00:52<01:20,  4.47s/it]
100%                                  1000/1000 [00:03<00:00, 268.20it/s]
Generating LIME:  43%|████▎     | 13/30 [00:57<01:16,  4.48s/it]
100%                                  1000/1000 [00:03<00:00, 271.18it/s]
Generating LIME:  47%|████▋     | 14/30 [01:01<01:11,  4.47s/it]
100%                                  1000/1000 [00:03<00:00, 264.17it/s]
Generating LIME:  50%|█████     | 15/30 [01:06<01:06,  4.47s/it]
100%                                  1000/1000 [00:03<00:00, 264.93it/s]
Generating LIME:  53%|█████▎    | 16/30 [01:10<01:02,  4.48s/it]
100%                                  1000/1000 [00:03<00:00, 279.22it/s]
Generating LIME:  57%|█████▋    | 17/30 [01:15<00:57,  4.42s/it]
100%                                  1000/1000 [00:03<00:00, 257.72it/s]
Generating LIME:  60%|██████    | 18/30 [01:19<00:53,  4.45s/it]
100%                                  1000/1000 [00:03<00:00, 262.37it/s]
Generating LIME:  63%|██████▎   | 19/30 [01:24<00:49,  4.45s/it]
100%                                  1000/1000 [00:03<00:00, 268.45it/s]
Generating LIME:  67%|██████▋   | 20/30 [01:28<00:44,  4.45s/it]
100%                                  1000/1000 [00:03<00:00, 265.71it/s]
Generating LIME:  70%|███████   | 21/30 [01:33<00:40,  4.46s/it]
100%                                  1000/1000 [00:03<00:00, 272.30it/s]
Generating LIME:  73%|███████▎  | 22/30 [01:37<00:35,  4.47s/it]
100%                                  1000/1000 [00:03<00:00, 272.86it/s]
Generating LIME:  77%|███████▋  | 23/30 [01:41<00:31,  4.44s/it]
100%                                  1000/1000 [00:03<00:00, 276.94it/s]
Generating LIME:  80%|████████  | 24/30 [01:46<00:26,  4.41s/it]
100%                                  1000/1000 [00:03<00:00, 274.17it/s]
Generating LIME:  83%|████████▎ | 25/30 [01:50<00:21,  4.39s/it]
100%                                  1000/1000 [00:03<00:00, 252.27it/s]
Generating LIME:  87%|████████▋ | 26/30 [01:55<00:17,  4.48s/it]
100%                                  1000/1000 [00:03<00:00, 275.12it/s]
Generating LIME:  90%|█████████ | 27/30 [01:59<00:13,  4.44s/it]
100%                                  1000/1000 [00:03<00:00, 261.13it/s]
Generating LIME:  93%|█████████▎| 28/30 [02:04<00:08,  4.46s/it]
100%                                  1000/1000 [00:03<00:00, 277.55it/s]
Generating LIME:  97%|█████████▋| 29/30 [02:08<00:04,  4.43s/it]
100%                                  1000/1000 [00:03<00:00, 275.05it/s]
Generating LIME: 100%|██████████| 30/30 [02:12<00:00,  4.43s/it] ✅ Saved LIME log to /content/lime_micc_f220_results/lime_visual
```

```
from IPython.display import display

samples = lime_df.head(3)
for _, row in samples.iterrows():
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))
    axs[0].imshow(Image.open(row['original_image']))
    axs[0].set_title("Original")

    axs[1].imshow(Image.open(row['heatmap_image']), cmap='hot')
    axs[1].set_title("LIME Heatmap")

    axs[2].imshow(Image.open(row['overlay_image']))
    axs[2].set_title("Overlay")

    for ax in axs: ax.axis("off")
    fig.suptitle(f"{row['filename']} | Pred Class: {row['predicted_class']} | LIME Score: {row['lime_score']:.4f}", fontsize=13)
    plt.tight_layout()
    plt.show()

# Show table
display(lime_df.head(5))
```




```
import os
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from IPython.display import display

# Reload LIME CSV log (adjust path if needed)
lime_csv_path = "/content/lime_micc_f220_results/lime_visualization_log.csv"
lime_df = pd.read_csv(lime_csv_path)

# Preview table
display(lime_df.head(5))

# Visualize 5 Samples
for idx, row in lime_df.head(5).iterrows():
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))

    axs[0].imshow(Image.open(row['original_image']))
    axs[0].set_title("Original")

    axs[1].imshow(Image.open(row['heatmap_image']), cmap='hot')
    axs[1].set_title("LIME Heatmap")

    axs[2].imshow(Image.open(row['overlay_image']))
    axs[2].set_title("Overlay")

    for ax in axs:
        ax.axis("off")

    fig.suptitle(f"{row['filename']} | Class: {row['class']} | LIME Score: {row['lime_score']:.4f}", fontsize=13)
    plt.tight_layout()
    plt.show()
```