```
# ===== PROFESSIONAL PYTORCH TRAINING PIPELINE (



#       INSTALL IMPORTS
!pip install kaggle timm scikit-learn --quiet
import os, zipfile, numpy as np, matplotlib.pypl
import torch, torch.nn as nn, torch.optim as opt
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms, mo
from sklearn.model_selection import StratifiedKF
from sklearn.metrics import f1_score, roc_auc_sc
from sklearn.utils.class_weight import compute_c
from torch.cuda.amp import GradScaler, autocast
from tqdm import tqdm
import timm




#  DATA DOWNLOAD
device = torch.device("cuda" if torch.cuda.is_av
print("Device:", device)
from google.colab import files
files.upload()  # upload kaggle.json

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d shashwatwork/
knee-osteoarthritis-dataset-with-severity
!unzip -q knee-osteoarthritis-dataset-with-sever



#            TRANSFORMS

DATA_DIR = "dataset"
```

```
DATA_DIR = 'dataset'
IMG_SIZE = 160
BATCH_SIZE = 32
EPOCHS = 10
N_SPLITS = 3

train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.
    of imagenet
])

val_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.
])



# target=It extracts only the labels. use for cl
full_dataset = datasets.ImageFolder(DATA_DIR, tr
targets = full_dataset.targets
num_classes = len(full_dataset.classes)




#               CLASS WEIGHTS & CROSS VALIDATION


# skf not balance the dataset.only preserves dst
# wc=N/(K·nc),,rare cls msclsifid→los×6.95,maj cl
# enumerate() just attaches a counter to a loop.
# skf: ensures that Each fold has same % of KL 0


class_weights = compute_class_weight('balanced',
                                     classes=np.
                                     y=targets)
class_weights = torch.tensor(class_weights,dtyp
```

```
84
85
86  skf = StratifiedKFold(n_splits=N_SPLITS, shuffle
87  fold_results = []
88
89  for fold, (train_idx, val_idx) in enumerate(skf.
    targets)):
90      print(f"\n========== FOLD {fold+1} =========
91
92      train_subset = Subset(full_dataset, train_id
93      val_subset = Subset(datasets.ImageFolder(DAT
        transform=val_transform), val_idx)
94
95      train_loader = DataLoader(train_subset, batc
        shuffle=True, num_workers=2)
96      val_loader = DataLoader(val_subset, batch_si
        num_workers=2)
97
98
99
100
101
102
103
104 # mdl.clsfir[0] = dropoutlayer, mdl.clsfir[1] li
105 # orgnl clasfir has 1k class, we have 5, so repa
106
107     # ================= MODEL ================
108     model = models.mobilenet_v2(weights="IMAGENE
109     for param in model.features[:-4].parameters(
110         param.requires_grad = False
111
112     in_features = model.classifier[1].in_feature
113     model.classifier = nn.Sequential(
114         nn.Dropout(0.4),
115         nn.Linear(in_features, num_classes)
116     )
117     model = model.to(device)
118
119
120
121
122
```

```
123
124     # ================ LOSS & OPTIMIZER =======
125
126 # smothing= one-hot labels (e.g., [0,0,1,0] → [0
127 # gradsclr=mix prcion trng; use float 16 and flo
128 # Weight decay = L2 regularization.
129 # ReduceLROnPlateau:reduces the lrate when valid
130 #  If metric n't improve for 2 epochs, reduce le
131
132     criterion = nn.CrossEntropyLoss(weight=class
133     optimizer = optim.AdamW(model.parameters(),
134     scheduler = optim.lr_scheduler.ReduceLROnPla
        patience=2)
135     scaler = GradScaler()
136     best_val_loss = np.inf
137
138
139
140
141
142
143 # model.train():on=drop out,batchNorm,, model.ev
144 # tqdm= progress bar
145
146     #                TRAIN LOOP
147     for epoch in range(EPOCHS):
148         model.train()
149         train_loss = 0
150
151         for images, labels in tqdm(train_loader)
152             images, labels = images.to(device),
153             optimizer.zero_grad()
154
155             with autocast():
156                 outputs = model(images)
157                 loss = criterion(outputs, labels
158
159             scaler.scale(loss).backward()
160             scaler.step(optimizer)
161             scaler.update()
162
163             train_loss += loss.item()
164
```

```
        model.eval()
        val_loss = 0
        all_preds, all_probs, all_labels = [], [
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(devic
                with autocast():
                    outputs = model(images)
                    loss = criterion(outputs, la
                val_loss += loss.item()
                probs = torch.softmax(outputs, d
                preds = torch.argmax(probs, dim=
                all_preds.extend(preds.cpu().num
                all_probs.extend(probs.cpu().num
                all_labels.extend(labels.cpu().n
        val_loss /= len(val_loader)
        scheduler.step(val_loss)
        f1 = f1_score(all_labels, all_preds, ave
        try:
            auc = roc_auc_score(all_labels, np.a
            multi_class='ovr')
        except:
            auc = 0
        print(f"Epoch {epoch+1}: Val Loss={val_l
        4f}")
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), f"bes
    fold_results.append((f1, auc))
```

```
206
207 #  FINAL RESULTS
208 print("\n===== CROSS-VALIDATION RESULTS =====")
209 for i, (f1, auc) in enumerate(fold_results):
210     print(f"Fold {i+1}: F1={f1:.4f}, AUC={auc:.4
211
212 print("\nMean F1:", np.mean([x[0] for x in fold_
213 print("Mean AUC:", np.mean([x[1] for x in fold_r
214
215 #  TEST-TIME EVALUATION (TTA)
216 print("\nRunning Test-Time Augmentation...")
217
218 tta_transform = transforms.Compose([
219     transforms.Resize((IMG_SIZE, IMG_SIZE)),
220     transforms.RandomHorizontalFlip(),
221     transforms.ToTensor(),
222     transforms.Normalize([0.485,0.456,0.406],[0.
223 ])
224
225 test_dataset = datasets.ImageFolder(DATA_DIR, tr
226 test_loader = DataLoader(test_dataset, batch_siz
227
228 model.load_state_dict(torch.load("best_fold_0.pt
229 model.eval()
230
231 all_preds = []
232 with torch.no_grad():
233     for images, _ in test_loader:
234         images = images.to(device)
235         outputs = model(images)
236         preds = torch.argmax(outputs, dim=1)
237         all_preds.extend(preds.cpu().numpy())
238
239 print("Test-Time Evaluation Completed.")
240
```

```
Device: cuda
```
Choose files kaggle.json

kaggle.json(application/json) - 73 bytes, last modified: 12/02/2026 - 100% done

```
Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/shashwatwork/knee-osteoarthritis-
License(s): Attribution 4.0 International (CC BY 4.0)
Downloading knee-osteoarthritis-dataset-with-severity.zip to /content
 55% 112M/204M [00:00<00:00, 1.17GB/s]
100% 204M/204M [00:00<00:00, 846MB/s]

========== FOLD 1 ==========
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" t
100%|██████████| 13.6M/13.6M [00:00<00:00, 116MB/s]
/tmp/ipython-input-110302322.py:135: FutureWarning: `torch.cuda.amp.GradScaler
  scaler = GradScaler()
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:31<00:00,  6.52it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 1: Val Loss=1.7113 F1=0.2080 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.52it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 2: Val Loss=1.6464 F1=0.2312 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.48it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 3: Val Loss=1.6648 F1=0.1022 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.70it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 4: Val Loss=1.7050 F1=0.1269 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.44it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 5: Val Loss=1.6811 F1=0.1185 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:19<00:00, 10.49it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 6: Val Loss=1.6411 F1=0.2011 AUC=0.5262
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:23<00:00,  8.71it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 7: Val Loss=1.6439 F1=0.1601 AUC=0.5325
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 10.94it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
```

```
  with autocast():
Epoch 8: Val Loss=1.6385 F1=0.1800 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 10.99it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 9: Val Loss=1.6491 F1=0.1555 AUC=0.5351
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.63it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
/tmp/ipython-input-110302322.py:135: FutureWarning: `torch.cuda.amp.GradScaler
  scaler = GradScaler()
Epoch 10: Val Loss=1.6393 F1=0.1767 AUC=0.5337

========== FOLD 2 ==========
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 11.19it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 1: Val Loss=1.6717 F1=0.1897 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 11.21it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 2: Val Loss=1.7003 F1=0.1691 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:19<00:00, 10.69it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 3: Val Loss=1.6074 F1=0.2981 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.44it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 4: Val Loss=1.6945 F1=0.0910 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:19<00:00, 10.72it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 5: Val Loss=1.6424 F1=0.1536 AUC=0.5367
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 10.94it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 6: Val Loss=1.6636 F1=0.0571 AUC=0.5483
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.34it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 7: Val Loss=1.6622 F1=0.0757 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.58it/s]
```

```
100%|██████████| 204/204 [00:17<00:00, 11.38it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 8: Val Loss=1.6481 F1=0.1172 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.41it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 9: Val Loss=1.6506 F1=0.1559 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.38it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
/tmp/ipython-input-110302322.py:135: FutureWarning: `torch.cuda.amp.GradScaler
  scaler = GradScaler()
Epoch 10: Val Loss=1.6460 F1=0.1631 AUC=0.0000

========== FOLD 3 ==========
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:19<00:00, 10.63it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 1: Val Loss=1.6511 F1=0.1883 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 11.03it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 2: Val Loss=1.7588 F1=0.0770 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.55it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 3: Val Loss=1.7010 F1=0.2020 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.51it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 4: Val Loss=1.6971 F1=0.0989 AUC=0.0000
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:17<00:00, 11.50it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 5: Val Loss=1.6597 F1=0.1404 AUC=0.5261
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:18<00:00, 11.31it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 6: Val Loss=1.6475 F1=0.1896 AUC=0.5311
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
  with autocast():
100%|██████████| 204/204 [00:19<00:00, 10.54it/s]
/tmp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
  with autocast():
Epoch 7: Val Loss=1.6598 F1=0.1750 AUC=0.5314
  0%|          | 0/204 [00:00<?, ?it/s]/tmp/ipython-input-110302322.py:155: Fu
```

```
Start coding or generate with AI.
```

```
/tmp/ipython-input-110302322.py:173: FutureWarning:  `torch.cuda.amp.autocast(a
```

```
# ===== Cell: 2 :KNEE KL PREDICTION (AUTO-FIXED V

import torch
import matplotlib.pyplot as plt
from google.colab import files
from PIL import Image
from torchvision import transforms, models

# -------- SETTINGS --------
IMG_SIZE = 160
MODEL_PATH = "best_fold_0.pth"

device = torch.device("cuda" if torch.cuda.is_ava

# -------- LOAD CHECKPOINT FIRST --------
checkpoint = torch.load(MODEL_PATH, map_location=

# Automatically detect number of classes from sav
num_classes = checkpoint["classifier.1.weight"].s

print("Detected number of classes in model:", num

# -------- REBUILD MODEL WITH CORRECT SIZE ------
model = models.mobilenet_v2(weights=None)

in_features = model.classifier[1].in_features
model.classifier = torch.nn.Sequential(
    torch.nn.Dropout(0.4),
    torch.nn.Linear(in_features, num_classes)
)

# Load weights
model.load_state_dict(checkpoint)
model = model.to(device)
model.eval()

print("Model loaded successfully.")

# -------- UPLOAD IMAGE --------
uploaded = files.upload()
```

```
img_path = list(uploaded.keys())[0]

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

img = Image.open(img_path).convert("RGB")
img_tensor = transform(img).unsqueeze(0).to(devic

# -------- PREDICT --------
with torch.no_grad():
    outputs = model(img_tensor)
    probabilities = torch.softmax(outputs, dim=1)
    pred_class = torch.argmax(probabilities, dim=

# Create labels dynamically
class_names = [f"KL{i}" for i in range(num_classe
pred_label = class_names[pred_class]

# -------- OUTPUT --------
print("\n🔧 Predicted Knee Osteoarthritis Grade:"
print("Confidence:", probabilities.cpu().numpy()[

# -------- SHOW IMAGE --------
plt.imshow(img)
plt.title(f"Predicted: {pred_label}")
plt.axis("off")
plt.show()
```

```
Detected number of classes in model: 4
Model loaded successfully.
```

Choose files WhatsApp I….45 AM.jpeg

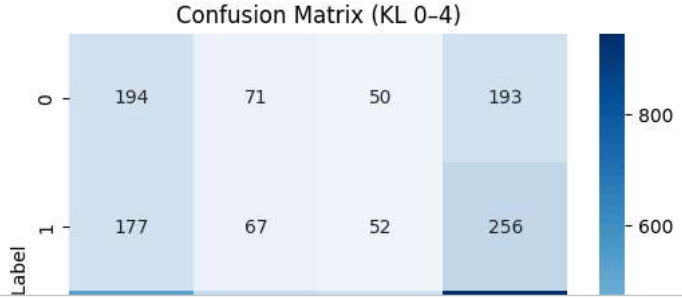
WhatsApp Image 2026-02-14 at 10.12.45 AM.jpeg(image/jpeg) - 16619 bytes, last modified: 14/02/2026 - 100% done

```
Saving WhatsApp Image 2026-02-14 at 10.12.45 AM.jpeg to WhatsApp Image 2026-02

🦴 Predicted Knee Osteoarthritis Grade: KL3
Confidence: [0.20688465 0.22698385 0.22646068 0.3396708 ]
```


```python
# ================= FINAL CLEAN METRICS CELL ====

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, cla
from sklearn.preprocessing import label_binarize

# ---------- SAFELY REBUILD ARRAYS ----------
y_true = np.array(all_labels)
y_prob = np.array(all_probs)

# Recompute predictions correctly
y_pred = np.argmax(y_prob, axis=1)

print("Final Verified Shapes:")
print("y_true:", y_true.shape)
print("y_pred:", y_pred.shape)
print("y_prob:", y_prob.shape)
```

```
20
21 # ================================
22 # 1️⃣ CONFUSION MATRIX
23 # ================================
24 cm = confusion_matrix(y_true, y_pred)
25
26 plt.figure(figsize=(6,5))
27 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues"
28 plt.title("Confusion Matrix (KL 0-4)")
29 plt.xlabel("Predicted Label")
30 plt.ylabel("True Label")
31 plt.show()
32
33 # ================================
34 # 2️⃣ CLASSIFICATION REPORT
35 # ================================
36 print("\n===== Classification Report =====\n")
37 print(classification_report(y_true, y_pred, digit
38
39 # ================================
40 # 3️⃣ ROC CURVE (MULTI-CLASS)
41 # ================================
42 num_classes = y_prob.shape[1]
43 y_true_bin = label_binarize(y_true, classes=np.ar
44
45 plt.figure(figsize=(7,6))
46
47 for i in range(num_classes):
48     fpr, tpr, _ = roc_curve(y_true_bin[:, i], y_p
49     roc_auc = auc(fpr, tpr)
50     plt.plot(fpr, tpr, label=f"KL {i} (AUC = {roc
51
52 plt.plot([0,1],[0,1],'k--')
53 plt.xlabel("False Positive Rate")
54 plt.ylabel("True Positive Rate")
55 plt.title("ROC Curve (One-vs-Rest)")
56 plt.legend()
57 plt.show()
58
59 print("\nMetrics Computation Completed Successful
```

```
Final Verified Shapes:
y_true: (3262,)
y_pred: (3262,)
y_prob: (3262, 4)
```




```
# XAi

# XAi

# ====================== FINAL CLEAN XAI CELL ====

!pip install grad-cam --quiet

import torch
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
from torchvision import transforms, models
from google.colab import files
from pytorch_grad_cam import GradCAM, GradCAMPlus
from pytorch_grad_cam.utils.model_targets import
from pytorch_grad_cam.utils.image import show_cam

device = torch.device("cuda" if torch.cuda.is_ava

# -------- LOAD BEST MODEL --------
num_classes = 4  # KL 0-4
model = models.mobilenet_v2(weights=None)

in_features = model.classifier[1].in_features
model.classifier = torch.nn.Sequential(
    torch.nn.Dropout(0.4),
    torch.nn.Linear(in_features, num_classes)
)
```

```
31
32 model.load_state_dict(torch.load("best_fold_0.pth
33 model = model.to(device)
34 model.eval()
35
36 print("Model Loaded Successfully ✅")
37
38 # -------- UPLOAD IMAGE FROM PC --------
39 uploaded = files.upload()
40 img_path = list(uploaded.keys())[0]
41
42 IMG_SIZE = 160
43
44 val_transform = transforms.Compose([
45     transforms.Resize((IMG_SIZE, IMG_SIZE)),
46     transforms.ToTensor(),
47     transforms.Normalize([0.485,0.456,0.406],
48                          [0.229,0.224,0.225])
49 ])
50
51 img = Image.open(img_path).convert("RGB")
52 input_tensor = val_transform(img).unsqueeze(0).to
53
54 rgb_img = np.array(img.resize((IMG_SIZE, IMG_SIZE
55
56 # -------- PREDICTION --------
57 with torch.no_grad():
58     output = model(input_tensor)
59     pred_class = torch.argmax(output, dim=1).item
60     confidence = torch.softmax(output, dim=1)[0,
61
62 print(f"Predicted KL Grade: {pred_class}")
63 print(f"Confidence: {confidence:.4f}")
64
65 target_layers = [model.features[-1]]
66 targets = [ClassifierOutputTarget(pred_class)]
67
68 # -------- CAM METHODS --------
69 cam_methods = {
70     "Grad-CAM": GradCAM(model=model, target_layer
71     "Grad-CAM++": GradCAMPlusPlus(model=model, ta
72     "Eigen-CAM": EigenCAM(model=model, target_lay
73     "Ablation-CAM": AblationCAM(model=model, targ
```

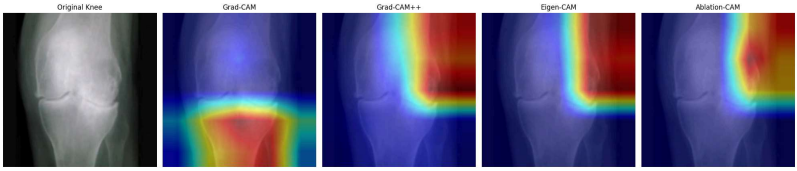
```
}

results = {}

for name, cam in cam_methods.items():
    grayscale_cam = cam(input_tensor=input_tensor
    visualization = show_cam_on_image(rgb_img, gr
    results[name] = visualization

# -------- DISPLAY --------
plt.figure(figsize=(20,6))

plt.subplot(1,5,1)
plt.imshow(rgb_img)
plt.title("Original Knee")
plt.axis("off")

for i,(name,vis) in enumerate(results.items()):
    plt.subplot(1,5,i+2)
    plt.imshow(vis)
    plt.title(name)
    plt.axis("off")

plt.tight_layout()
```

```
Model Loaded Successfully ✅
```

Choose files WhatsApp I….45 AM.jpeg

WhatsApp Image 2026-02-14 at 10.12.45 AM.jpeg(image/jpeg) - 16619 bytes, last modified: 14/02/2026 - 100% done

```
Saving WhatsApp Image 2026-02-14 at 10.12.45 AM.jpeg to WhatsApp Image 2026-02
Predicted KL Grade: 3
Confidence: 0.3397
100%|██████████| 40/40 [00:00<00:00, 48.46it/s]
```



```
# ====================== CLEAN FULL XAI METRICS C

import torch
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import auc

device = torch.device("cuda" if torch.cuda.is_av

def normalize_cam(cam):
    cam = cam - cam.min()
    cam = cam / (cam.max() + 1e-8)
    return cam

def compute_entropy(cam):
    cam = cam / (cam.sum() + 1e-8)
    return -np.sum(cam * np.log(cam + 1e-8))

def compute_sparsity(cam):
    return np.sum(cam < 0.1) / cam.size

def compute_intensity(cam):
    return np.mean(cam)

def confidence_drop(model, input_tensor, cam_map
    mask = torch.tensor(cam_map).unsqueeze(0).un
    mask = F.interpolate(mask, size=(IMG_SIZE, I

    with torch.no_grad():
        original_out = model(input_tensor)
        original_conf = F.softmax(original_out,

        masked_input = input_tensor * mask
        masked_out = model(masked_input)
        masked_conf = F.softmax(masked_out, dim=

    return original_conf - masked_conf

def insertion_deletion_auc(model, input_tensor,
    cam_flat = cam_map.flatten()
```

```
    sorted_idx = np.argsort(-cam_flat)

    scores_insert, scores_delete = [], []

    for i in range(steps):
        percent = int(len(sorted_idx)*(i+1)/step
        mask = np.zeros_like(cam_flat)
        mask[sorted_idx[:percent]] = 1
        mask = mask.reshape(cam_map.shape)

        mask_tensor = torch.tensor(mask).unsquee
        mask_tensor = F.interpolate(mask_tensor,

        with torch.no_grad():
            ins_out = model(input_tensor * mask_
            del_out = model(input_tensor * (1-ma

        ins_score = F.softmax(ins_out, dim=1)[0,
        del_score = F.softmax(del_out, dim=1)[0,

        scores_insert.append(ins_score)
        scores_delete.append(del_score)

    x_axis = np.linspace(0,1,steps)
    ins_auc = auc(x_axis, scores_insert)
    del_auc = auc(x_axis, scores_delete)

    return ins_auc, del_auc, scores_insert, scor

# ===================== METRIC COMPUTATION =====

results_table = []

plt.figure(figsize=(14,6))

for name, cam in cam_methods.items():
    grayscale_cam = cam(input_tensor=input_tenso
    grayscale_cam = normalize_cam(grayscale_cam)

    entropy = compute_entropy(grayscale_cam)
    sparsity = compute_sparsity(grayscale_cam)
    intensity = compute_intensity(grayscale_cam)
    conf_drop = confidence_drop(model, input_ten
```

```
    ins_auc, del_auc, ins_curve, del_curve = ins
        model, input_tensor, grayscale_cam)

    results_table.append([
        name, ins_auc, del_auc,
        entropy, sparsity,
        intensity, conf_drop
    ])

    plt.plot(ins_curve, label=f"{name}")

plt.title("Insertion Curves (Higher Faster = Bet
plt.legend()
plt.show()

plt.figure(figsize=(14,6))

for name, cam in cam_methods.items():
    grayscale_cam = normalize_cam(cam(input_tens
    _, _, _, del_curve = insertion_deletion_auc(
    plt.plot(del_curve, label=f"{name}")

plt.title("Deletion Curves (Lower Faster = Bette
plt.legend()
plt.show()

# ===================== RESULTS TABLE ==========

columns = [
    "Method",
    "Insertion AUC ↑",
    "Deletion AUC ↓",
    "Entropy ↓",
    "Sparsity ↑",
    "Intensity",
    "Confidence Drop ↓"
]

df_results = pd.DataFrame(results_table, columns

print("\n===== PROFESSIONAL XAI METRICS COMPARIS
```

Next steps: Generate code with `df_results` New interactive sheet

```
100%|██████████| 40/40 [00:00<00:00, 51.59it/s]
```


```python
# ===== PYTORCH MODEL COMPARISON (USING YOUR EXIS

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models
from sklearn.metrics import f1_score
import numpy as np
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_ava

NUM_CLASSES = len(full_dataset.classes)
EPOCHS = 5

results = {}

models_dict = {
    "ResNet18": models.resnet18(weights="IMAGENET
    "DenseNet121": models.densenet121(weights="IM
    "EfficientNetB0": models.efficientnet_b0(weig
}

for name, model in models_dict.items():
    print(f"\n🚀 Training {name}")

    # ----- Replace Final Layer -----
    if "resnet" in name.lower():
        model.fc = nn.Linear(model.fc.in_features
    elif "densenet" in name.lower():
        model.classifier = nn.Linear(model.classi
    elif "efficientnet" in name.lower():
        model.classifier[1] = nn.Linear(model.cla
        NUM_CLASSES)
```

```
34
35      model = model.to(device)
36
37      criterion = nn.CrossEntropyLoss()
38      optimizer = optim.Adam(model.parameters(), lr
39
40      # ----- Training -----
41      model.train()
42      for epoch in range(EPOCHS):
43          for images, labels in train_loader:
44              images = images.to(device)
45              labels = labels.to(device)
46
47              optimizer.zero_grad()
48              outputs = model(images)
49              loss = criterion(outputs, labels)
50              loss.backward()
51              optimizer.step()
52
53      # ----- Evaluation -----
54      model.eval()
55      y_true, y_pred = [], []
56
57      with torch.no_grad():
58          for images, labels in val_loader:
59              images = images.to(device)
60              outputs = model(images)
61              preds = torch.argmax(outputs, dim=1).
62
63              y_pred.extend(preds)
64              y_true.extend(labels.numpy())
65
66      y_true = np.array(y_true)
67      y_pred = np.array(y_pred)
68
69      acc = np.mean(y_true == y_pred)
70      f1 = f1_score(y_true, y_pred, average="weight
71
72      results[name] = {"Accuracy": acc, "F1 Score":
73
74      print(f"{name} Accuracy: {acc:.4f}")
75      print(f"{name} F1 Score: {f1:.4f}")
76
```

```
# ----- Results Table -----
results_df = pd.DataFrame(results).T
print("\n📊 Model Comparison:")
print(results_df)

results_df.plot(kind="bar", figsize=(8,5))
plt.title("Model Comparison")
plt.ylim(0,1)
plt.show()
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /r
100%|██████████| 44.7M/44.7M [00:00<00:00, 199MB/s]
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to
100%|██████████| 30.8M/30.8M [00:00<00:00, 122MB/s]
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f
100%|██████████| 20.5M/20.5M [00:00<00:00, 120MB/s]
```

🚀 Training ResNet18

```
from google.colab import drive
drive.mount('/content/drive')


import torch
from datetime import datetime

CHECKPOINT_PATH = "/content/drive/MyDrive/knee_OA_

checkpoint = {
    "model architecture": "mobilenet v2",
```