

```
1 # ===== PROFESSIONAL PYTORCH TRAINING PIPELINE (C)
2
3
4
5 #           INSTALL IMPORTS
6 !pip install kaggle timm scikit-learn --quiet
7 import os, zipfile, numpy as np, matplotlib.pyplot
8 import torch, torch.nn as nn, torch.optim as optim
9 from torch.utils.data import DataLoader, Subset
10 from torchvision import datasets, transforms, models
11 from sklearn.model_selection import StratifiedKFold
12 from sklearn.metrics import f1_score, roc_auc_score
13 from sklearn.utils.class_weight import compute_class_weight
14 from torch.cuda.amp import GradScaler, autocast
15 from tqdm import tqdm
16 import timm
17
18
19
20
21
22
23 # DATA DOWNLOAD
24 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
25 print("Device:", device)
26 from google.colab import files
27 files.upload() # upload kaggle.json
28
29 !mkdir -p ~/.kaggle
30 !cp kaggle.json ~/.kaggle/
31 !chmod 600 ~/.kaggle/kaggle.json
32
33 !kaggle datasets download -d shashwatwork/knee-osteoarthritis-dataset-with-severity
34 !unzip -q knee-osteoarthritis-dataset-with-severity.zip
35
36
37
38
39 #           TRANSFORMS
40
41 DATA_DIR = "dataset"
```

```
41 DATA_DIR = 'dataset'
42 IMG_SIZE = 160
43 BATCH_SIZE = 32
44 EPOCHS = 10
45 N_SPLITS = 3
46
47 train_transform = transforms.Compose([
48     transforms.Resize((IMG_SIZE, IMG_SIZE)),
49     transforms.RandomHorizontalFlip(),
50     transforms.RandomRotation(10),
51     transforms.ToTensor(),
52     transforms.Normalize([0.485, 0.456, 0.406],[0.
      of imagenet
53 ])
54
55 val_transform = transforms.Compose([
56     transforms.Resize((IMG_SIZE, IMG_SIZE)),
57     transforms.ToTensor(),
58     transforms.Normalize([0.485, 0.456, 0.406],[0.
59 ])
60
61
62
63 # target=It extracts only the labels. use for cl
64 full_dataset = datasets.ImageFolder(DATA_DIR, tr
65 targets = full_dataset.targets
66 num_classes = len(full_dataset.classes)
67
68
69
70
71 # CLASS WEIGHTS & CROSS VALIDATION
72
73
74 # skf not balance the dataset.only preserves dst
75 # wc=N/(K-nc),,rare cls msclsifid→los×6.95,maj c]
76 # enumerate() just attaches a counter to a loop.
77 # skf: ensures that Each fold has same % of KL 0
78
79
80 class_weights = compute_class_weight('balanced',
81                                         classes=np.
82                                         y=targets)
83 class_weights = torch.tensor(class_weights).float()
```

```
84
85
86     skf = StratifiedKFold(n_splits=N_SPLITS, shuffle=True)
87     fold_results = []
88
89     for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
90         print(f"\n===== FOLD {fold+1} =====")
91
92         train_subset = Subset(full_dataset, train_idx)
93         val_subset = Subset(datasets.ImageFolder(DATASET_PATH,
94                                         transform=val_transform), val_idx)
95
96         train_loader = DataLoader(train_subset, batch_size=16,
97                                   shuffle=True, num_workers=2)
98         val_loader = DataLoader(val_subset, batch_size=16,
99                               num_workers=2)
100
101
102
103
104     # mdl.clsfir[0] = dropoutlayer, mdl.clsfir[1] is the linear layer
105     # orgnl clasfir has 1k class, we have 5, so repa
106
107     # ===== MODEL =====
108     model = models.mobilenet_v2(weights="IMAGENET")
109     for param in model.features[:-4].parameters():
110         param.requires_grad = False
111
112     in_features = model.classifier[1].in_features
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     # smoothing= 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
```

```
165     model.eval()
166     val_loss = 0
167     all_preds, all_probs, all_labels = [], []
168
169     with torch.no_grad():
170         for images, labels in val_loader:
171             images, labels = images.to(device)
172
173             with autocast():
174                 outputs = model(images)
175                 loss = criterion(outputs, labels)
176
177                 val_loss += loss.item()
178                 probs = torch.softmax(outputs, dim=1)
179                 preds = torch.argmax(probs, dim=1)
180
181                 all_preds.extend(preds.cpu().numpy())
182                 all_probs.extend(probs.cpu().numpy())
183                 all_labels.extend(labels.cpu().numpy())
184
185             val_loss /= len(val_loader)
186             scheduler.step(val_loss)
187
188             f1 = f1_score(all_labels, all_preds, average='macro')
189             try:
190                 auc = roc_auc_score(all_labels, np.array(all_probs),
191                                     multi_class='ovr')
192             except:
193                 auc = 0
194
195             print(f"Epoch {epoch+1}: Val Loss={val_loss:.4f}")
196
197             if val_loss < best_val_loss:
198                 best_val_loss = val_loss
199                 torch.save(model.state_dict(), f"best_{epoch}.pt")
200
201             fold_results.append((f1, auc))
202
203
204
205
```

```
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:.4f}")
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_
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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/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]
/tmpp/ipython-input-110302322.py:173: FutureWarning: `torch.cuda.amp.autocast(a
with autocast():
```

```
while 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]
```

```
[000] [██████] 204/204 [00:17<00:00, 11.30it/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.GradScal
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
```

1 Start coding or generate with AI.

```
/tmp/ipython-input-110302322.py:173: FutureWarning: torch.cuda.amp.autocast(a
 1 # ===== Cell: 2 :KNEE KL PREDICTION (AUTO-FIXED V
 2
 3 import torch
 4 import matplotlib.pyplot as plt
 5 from google.colab import files
 6 from PIL import Image
 7 from torchvision import transforms, models
 8
 9 # ----- SETTINGS -----
10 IMG_SIZE = 160
11 MODEL_PATH = "best_fold_0.pth"
12
13 device = torch.device("cuda" if torch.cuda.is_ava
14
15 # ----- LOAD CHECKPOINT FIRST -----
16 checkpoint = torch.load(MODEL_PATH, map_location=
17
18 # Automatically detect number of classes from sav
19 num_classes = checkpoint["classifier.1.weight"].s
20
21 print("Detected number of classes in model:", num_
22
23 # ----- REBUILD MODEL WITH CORRECT SIZE -----
24 model = models.mobilenet_v2(weights=None)
25
26 in_features = model.classifier[1].in_features
27 model.classifier = torch.nn.Sequential(
28     torch.nn.Dropout(0.4),
29     torch.nn.Linear(in_features, num_classes)
30 )
31
32 # Load weights
33 model.load_state_dict(checkpoint)
34 model = model.to(device)
35 model.eval()
36
37 print("Model loaded successfully.")
38
39 # ----- UPLOAD IMAGE -----
40 uploaded = files.upload()
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
848
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
958
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1103
1104
1105
1105
1106
1107
1107
1108
1109
1109
1110
1111
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1760
1761
1761
1762
1762
1763
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1770
1771
1771
1772
1772
1773
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1780
1781
1781
1782
1782
1783
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1790
1791
1791
1792
1792
1793
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1800
1801
1801
1802
1802
1803
1803
1804
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1810
1811
1811
1812
1812
1813
1813
1814
1814
1815
1815
181
```

```
41 img_path = firstUploaded.keys()[0]
42
43 # ----- TRANSFORM -----
44 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 img_tensor = transform(img).unsqueeze(0).to(device)
53
54 # ----- PREDICT -----
55 with torch.no_grad():
56     outputs = model(img_tensor)
57     probabilities = torch.softmax(outputs, dim=1)
58     pred_class = torch.argmax(probabilities, dim=
59
59 # Create labels dynamically
60 class_names = [f"KL{i}" for i in range(num_classes)]
61 pred_label = class_names[pred_class]
62
63 # ----- OUTPUT -----
64 print("\n👉 Predicted Knee Osteoarthritis Grade:")
65 print("Confidence:", probabilities.cpu().numpy())
66
67 # ----- SHOW IMAGE -----
68 plt.imshow(img)
69 plt.title(f"Predicted: {pred_label}")
70 plt.axis("off")
71 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 ]
```

Predicted: KL3



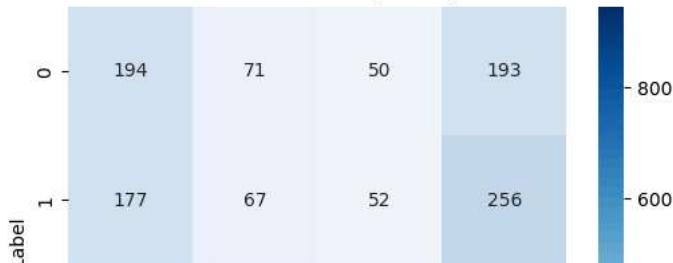
```
1 # ===== FINAL CLEAN METRICS CELL =====
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.metrics import confusion_matrix, cla
7 from sklearn.preprocessing import label_binarize
8
9 # ----- SAFELY REBUILD ARRAYS -----
10 y_true = np.array(all_labels)
11 y_prob = np.array(all_probs)
12
13 # Recompute predictions correctly
14 y_pred = np.argmax(y_prob, axis=1)
15
16 print("Final Verified Shapes:")
17 print("y_true:", y_true.shape)
18 print("y_pred:", y_pred.shape)
19 print("y_prob:", y_prob.shape)
```

```
20
21 # =====
22 # [1] 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 # [2] CLASSIFICATION REPORT
35 # =====
36 print("\n==== Classification Report ====\n")
37 print(classification_report(y_true, y_pred, digits=3))
38
39 # =====
40 # [3] ROC CURVE (MULTI-CLASS)
41 # =====
42 num_classes = y_prob.shape[1]
43 y_true_bin = label_binarize(y_true, classes=np.arange(num_classes))
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_prob[:, i])
49     roc_auc = auc(fpr, tpr)
50     plt.plot(fpr, tpr, label=f"KL {i} (AUC = {roc_auc:.2f})")
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)
```

Confusion Matrix (KL 0-4)



```
1 # XAi  
2  
3 # XAi  
4  
5 # ===== FINAL CLEAN XAI CELL =====  
6  
7 !pip install grad-cam --quiet  
8  
9 import torch  
10 import numpy as np  
11 import matplotlib.pyplot as plt  
12 import cv2  
13 from PIL import Image  
14 from torchvision import transforms, models  
15 from google.colab import files  
16 from pytorch_grad_cam import GradCAM, GradCAMPlus  
17 from pytorch_grad_cam.utils.model_targets import  
18 from pytorch_grad_cam.utils.image import show_carr  
19  
20 device = torch.device("cuda" if torch.cuda.is_ava  
21  
22 # ----- LOAD BEST MODEL -----  
23 num_classes = 4 # KL 0-4  
24 model = models.mobilenet_v2(weights=None)  
25  
26 in_features = model.classifier[1].in_features  
27 model.classifier = torch.nn.Sequential(  
28     torch.nn.Dropout(0.4),  
29     torch.nn.Linear(in_features, num_classes)  
30 )
```

```
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).tc
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
```

```
74 }
75
76 results = {}
77
78 for name, cam in cam_methods.items():
79     grayscale_cam = cam(input_tensor=input_tensor
80     visualization = show_cam_on_image(rgb_img, gr
81     results[name] = visualization
82
83 # ----- DISPLAY -----
84 plt.figure(figsize=(20,6))
85
86 plt.subplot(1,5,1)
87 plt.imshow(rgb_img)
88 plt.title("Original Knee")
89 plt.axis("off")
90
91 for i,(name,vis) in enumerate(results.items()):
92     plt.subplot(1,5,i+2)
93     plt.imshow(vis)
94     plt.title(name)
95     plt.axis("off")
96
97 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:

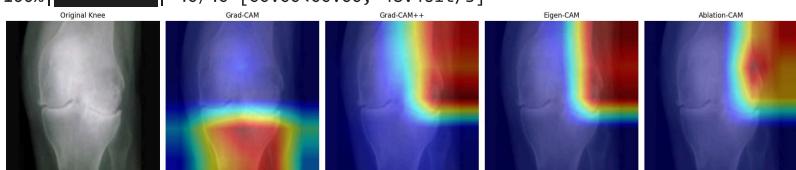
14/02/2026 - 100% done

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]



```
1 # ===== CLEAN FULL XAI METRICS C
2
3 import torch
4 import torch.nn.functional as F
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from sklearn.metrics import auc
9
10 device = torch.device("cuda" if torch.cuda.is_av
11
12 def normalize_cam(cam):
13     cam = cam - cam.min()
14     cam = cam / (cam.max() + 1e-8)
15     return cam
16
17 def compute_entropy(cam):
18     cam = cam / (cam.sum() + 1e-8)
19     return -np.sum(cam * np.log(cam + 1e-8))
20
21 def compute_sparsity(cam):
22     return np.sum(cam < 0.1) / cam.size
23
24 def compute_intensity(cam):
25     return np.mean(cam)
26
27 def confidence_drop(model, input_tensor, cam_map
28     mask = torch.tensor(cam_map).unsqueeze(0).un
29     mask = F.interpolate(mask, size=(IMG_SIZE, I
30
31     with torch.no_grad():
32         original_out = model(input_tensor)
33         original_conf = F.softmax(original_out,
34
35         masked_input = input_tensor * mask
36         masked_out = model(masked_input)
37         masked_conf = F.softmax(masked_out, dim=
38
39     return original_conf - masked_conf
40
41 def insertion_deletion_auc(model, input_tensor,
42     cam_flat = cam_map.flatten()
```

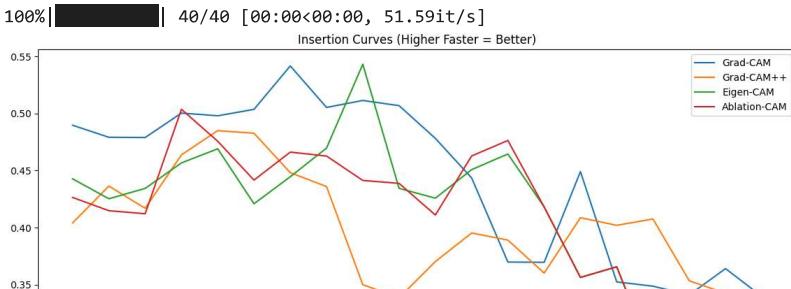
```
43     sorted_idx = np.argsort(-cam_flat)
44
45     scores_insert, scores_delete = [], []
46
47     for i in range(steps):
48         percent = int(len(sorted_idx)*(i+1)/step)
49         mask = np.zeros_like(cam_flat)
50         mask[sorted_idx[:percent]] = 1
51         mask = mask.reshape(cam_map.shape)
52
53         mask_tensor = torch.tensor(mask).unsqueeze(0)
54         mask_tensor = F.interpolate(mask_tensor,
55
56         with torch.no_grad():
57             ins_out = model(input_tensor * mask)
58             del_out = model(input_tensor * (1-ma
59
60             ins_score = F.softmax(ins_out, dim=1)[0],
61             del_score = F.softmax(del_out, dim=1)[0,
62
63             scores_insert.append(ins_score)
64             scores_delete.append(del_score)
65
66             x_axis = np.linspace(0,1,steps)
67             ins_auc = auc(x_axis, scores_insert)
68             del_auc = auc(x_axis, scores_delete)
69
70             return ins_auc, del_auc, scores_insert, scor
71
72 # ===== METRIC COMPUTATION =====
73
74 results_table = []
75
76 plt.figure(figsize=(14,6))
77
78 for name, cam in cam_methods.items():
79     grayscale_cam = cam(input_tensor=input_tenso
80     grayscale_cam = normalize_cam(grayscale_cam)
81
82     entropy = compute_entropy(grayscale_cam)
83     sparsity = compute_sparsity(grayscale_cam)
84     intensity = compute_intensity(grayscale_cam)
85     conf_drop = confidence_drop(model, input_ten
```

```
86
87     ins_auc, del_auc, ins_curve, del_curve = ins
88         model, input_tensor, grayscale_cam)
89
90     results_table.append([
91         name, ins_auc, del_auc,
92         entropy, sparsity,
93         intensity, conf_drop
94     ])
95
96     plt.plot(ins_curve, label=f"{name}")
97
98 plt.title("Insertion Curves (Higher Faster = Better")
99 plt.legend()
100 plt.show()
101
102 plt.figure(figsize=(14,6))
103
104 for name, cam in cam_methods.items():
105     grayscale_cam = normalize_cam(cam(input_tens
106     _, _, _, del_curve = insertion_deletion_auc(
107         plt.plot(del_curve, label=f"{name}") )
108
109 plt.title("Deletion Curves (Lower Faster = Better")
110 plt.legend()
111 plt.show()
112
113 # ===== RESULTS TABLE =====
114
115 columns = [
116     "Method",
117     "Insertion AUC ↑",
118     "Deletion AUC ↓",
119     "Entropy ↓",
120     "Sparsity ↑",
121     "Intensity",
122     "Confidence Drop ↓"
123 ]
124
125 df_results = pd.DataFrame(results_table, columns
126
127 print("\n===== PROFESSIONAL XAI METRICS COMPARIS
```


Next steps:

[Generate code with df_results](#)

[New interactive sheet](#)



```

1  # ===== PYTORCH MODEL COMPARISON (USING YOUR EXIS
2
3  import torch
4  import torch.nn as nn
5  import torch.optim as optim
6  from torchvision import models
7  from sklearn.metrics import f1_score
8  import numpy as np
9  import pandas as pd
10
11 device = torch.device("cuda" if torch.cuda.is_ava
12
13 NUM_CLASSES = len(full_dataset.classes)
14 EPOCHS = 5
15
16 results = {}
17
18 models_dict = {
19     "ResNet18": models.resnet18(weights="IMAGENET
20     "DenseNet121": models.densenet121(weights="IM
21     "EfficientNetB0": models.efficientnet_b0(weig
22 }
23
24 for name, model in models_dict.items():
25     print(f"\n🚀 Training {name}")
26
27     # ----- Replace Final Layer -----
28     if "resnet" in name.lower():
29         model.fc = nn.Linear(model.fc.in_features,
30     elif "densenet" in name.lower():
31         model.classifier = nn.Linear(model.classifi
32     elif "efficientnet" in name.lower():
33         model.classifier[1] = nn.Linear(model.clas
34
35     model.to(device)
36
37     # Train Model
38     for epoch in range(EPOCHS):
39         train_loss = 0.0
40         train_f1 = 0.0
41
42         for batch_idx, (inputs, targets) in enumerate(t
43             outputs = model(inputs)
44             loss = criterion(outputs, targets)
45
46             optimizer.zero_grad()
47             loss.backward()
48             optimizer.step()
49
50             train_loss += loss.item()
51             _, predicted = outputs.max(1)
52             train_f1 += f1_score(targets.cpu().numpy(), p
53
54         avg_train_loss = train_loss / len(t
55         avg_train_f1 = train_f1 / len(t
56
57         print(f"Epoch [{epoch+1}/{EPOCHS}], Loss: {avg
58
59     # Evaluate Model
60     model.eval()
61
62     total_loss = 0.0
63     total_f1 = 0.0
64
65     with torch.no_grad():
66         for batch_idx, (inputs, targets) in enumerate(t
67             outputs = model(inputs)
68             loss = criterion(outputs, targets)
69
70             total_loss += loss.item()
71             _, predicted = outputs.max(1)
72             total_f1 += f1_score(targets.cpu().numpy(), p
73
74     avg_val_loss = total_loss / len(t
75     avg_val_f1 = total_f1 / len(t
76
77     print(f"Validation Loss: {avg_val_loss}, F1 Score: {avg_val_f1}")
78
79     # Save Model
80     if avg_val_f1 > max(results["f1"]):
81         max_results = results["f1"]
82         max_results["model"] = name
83         max_results["f1"] = avg_val_f1
84         max_results["loss"] = avg_val_loss
85
86     # Save Results
87     results[name] = max_results
88
89 # Print Results
90 print(results)
91
92 # Export Results to CSV
93 df = pd.DataFrame(results)
94 df.to_csv("model_comparison.csv")
95
96 # Print Summary
97 print(f"\nModel Comparison Summary:\n{df}")
98
99 # Print Best Model
100 print(f"\nThe best performing model is {max_results['model']} with an average F1 score of {max_results['f1']} and a validation loss of {max_results['loss']}.\n")
101
102 # Print Model Architecture
103 print(f"\nModel Architecture:\n{model}")
104
105 # Print Model Summary
106 print(f"\nModel Summary:\n{model.summary()}")
107
108 # Print Model Parameters
109 print(f"\nModel Parameters:\n{model.parameters()}")
110
111 # Print Model State Dictionary
112 print(f"\nModel State Dictionary:\n{model.state_dict()}")
113
114 # Print Model Training Data
115 print(f"\nModel Training Data:\n{train_loader}")
116
117 # Print Model Validation Data
118 print(f"\nModel Validation Data:\n{val_loader}")
119
120 # Print Model Optimizer
121 print(f"\nModel Optimizer:\n{optimizer}")
122
123 # Print Model Criterion
124 print(f"\nModel Criterion:\n{criterion}")
125
126 # Print Model Device
127 print(f"\nModel Device:\n{device}")
128
129 # Print Model In Features
130 print(f"\nModel In Features:\n{model.in_features}")
131
132 # Print Model Out Features
133 print(f"\nModel Out Features:\n{model.out_features}")
134
135 # Print Model Class Name
136 print(f"\nModel Class Name:\n{model.__class__.__name__}")
137
138 # Print Model Class Description
139 print(f"\nModel Class Description:\n{model.__class__.__doc__}")
140
141 # Print Model Class Base
142 print(f"\nModel Class Base:\n{model.__class__.__base__}")
143
144 # Print Model Class Bases
145 print(f"\nModel Class Bases:\n{model.__class__.bases}")
146
147 # Print Model Class Doc
148 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
149
150 # Print Model Class Name
151 print(f"\nModel Class Name:\n{model.__class__.__name__}")
152
153 # Print Model Class Doc
154 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
155
156 # Print Model Class Bases
157 print(f"\nModel Class Bases:\n{model.__class__.bases}")
158
159 # Print Model Class Base
160 print(f"\nModel Class Base:\n{model.__class__.__base__}")
161
162 # Print Model Class Doc
163 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
164
165 # Print Model Class Name
166 print(f"\nModel Class Name:\n{model.__class__.__name__}")
167
168 # Print Model Class Doc
169 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
170
171 # Print Model Class Bases
172 print(f"\nModel Class Bases:\n{model.__class__.bases}")
173
174 # Print Model Class Base
175 print(f"\nModel Class Base:\n{model.__class__.__base__}")
176
177 # Print Model Class Doc
178 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
179
180 # Print Model Class Name
181 print(f"\nModel Class Name:\n{model.__class__.__name__}")
182
183 # Print Model Class Doc
184 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
185
186 # Print Model Class Bases
187 print(f"\nModel Class Bases:\n{model.__class__.bases}")
188
189 # Print Model Class Base
190 print(f"\nModel Class Base:\n{model.__class__.__base__}")
191
192 # Print Model Class Doc
193 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
194
195 # Print Model Class Name
196 print(f"\nModel Class Name:\n{model.__class__.__name__}")
197
198 # Print Model Class Doc
199 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
200
201 # Print Model Class Bases
202 print(f"\nModel Class Bases:\n{model.__class__.bases}")
203
204 # Print Model Class Base
205 print(f"\nModel Class Base:\n{model.__class__.__base__}")
206
207 # Print Model Class Doc
208 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
209
210 # Print Model Class Name
211 print(f"\nModel Class Name:\n{model.__class__.__name__}")
212
213 # Print Model Class Doc
214 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
215
216 # Print Model Class Bases
217 print(f"\nModel Class Bases:\n{model.__class__.bases}")
218
219 # Print Model Class Base
220 print(f"\nModel Class Base:\n{model.__class__.__base__}")
221
222 # Print Model Class Doc
223 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
224
225 # Print Model Class Name
226 print(f"\nModel Class Name:\n{model.__class__.__name__}")
227
228 # Print Model Class Doc
229 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
230
231 # Print Model Class Bases
232 print(f"\nModel Class Bases:\n{model.__class__.bases}")
233
234 # Print Model Class Base
235 print(f"\nModel Class Base:\n{model.__class__.__base__}")
236
237 # Print Model Class Doc
238 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
239
240 # Print Model Class Name
241 print(f"\nModel Class Name:\n{model.__class__.__name__}")
242
243 # Print Model Class Doc
244 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
245
246 # Print Model Class Bases
247 print(f"\nModel Class Bases:\n{model.__class__.bases}")
248
249 # Print Model Class Base
250 print(f"\nModel Class Base:\n{model.__class__.__base__}")
251
252 # Print Model Class Doc
253 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
254
255 # Print Model Class Name
256 print(f"\nModel Class Name:\n{model.__class__.__name__}")
257
258 # Print Model Class Doc
259 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
260
261 # Print Model Class Bases
262 print(f"\nModel Class Bases:\n{model.__class__.bases}")
263
264 # Print Model Class Base
265 print(f"\nModel Class Base:\n{model.__class__.__base__}")
266
267 # Print Model Class Doc
268 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
269
270 # Print Model Class Name
271 print(f"\nModel Class Name:\n{model.__class__.__name__}")
272
273 # Print Model Class Doc
274 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
275
276 # Print Model Class Bases
277 print(f"\nModel Class Bases:\n{model.__class__.bases}")
278
279 # Print Model Class Base
280 print(f"\nModel Class Base:\n{model.__class__.__base__}")
281
282 # Print Model Class Doc
283 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
284
285 # Print Model Class Name
286 print(f"\nModel Class Name:\n{model.__class__.__name__}")
287
288 # Print Model Class Doc
289 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
290
291 # Print Model Class Bases
292 print(f"\nModel Class Bases:\n{model.__class__.bases}")
293
294 # Print Model Class Base
295 print(f"\nModel Class Base:\n{model.__class__.__base__}")
296
297 # Print Model Class Doc
298 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
299
300 # Print Model Class Name
301 print(f"\nModel Class Name:\n{model.__class__.__name__}")
302
303 # Print Model Class Doc
304 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
305
306 # Print Model Class Bases
307 print(f"\nModel Class Bases:\n{model.__class__.bases}")
308
309 # Print Model Class Base
310 print(f"\nModel Class Base:\n{model.__class__.__base__}")
311
312 # Print Model Class Doc
313 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
314
315 # Print Model Class Name
316 print(f"\nModel Class Name:\n{model.__class__.__name__}")
317
318 # Print Model Class Doc
319 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
320
321 # Print Model Class Bases
322 print(f"\nModel Class Bases:\n{model.__class__.bases}")
323
324 # Print Model Class Base
325 print(f"\nModel Class Base:\n{model.__class__.__base__}")
326
327 # Print Model Class Doc
328 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
329
330 # Print Model Class Name
331 print(f"\nModel Class Name:\n{model.__class__.__name__}")
332
333 # Print Model Class Doc
334 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
335
336 # Print Model Class Bases
337 print(f"\nModel Class Bases:\n{model.__class__.bases}")
338
339 # Print Model Class Base
340 print(f"\nModel Class Base:\n{model.__class__.__base__}")
341
342 # Print Model Class Doc
343 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
344
345 # Print Model Class Name
346 print(f"\nModel Class Name:\n{model.__class__.__name__}")
347
348 # Print Model Class Doc
349 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
350
351 # Print Model Class Bases
352 print(f"\nModel Class Bases:\n{model.__class__.bases}")
353
354 # Print Model Class Base
355 print(f"\nModel Class Base:\n{model.__class__.__base__}")
356
357 # Print Model Class Doc
358 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
359
360 # Print Model Class Name
361 print(f"\nModel Class Name:\n{model.__class__.__name__}")
362
363 # Print Model Class Doc
364 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
365
366 # Print Model Class Bases
367 print(f"\nModel Class Bases:\n{model.__class__.bases}")
368
369 # Print Model Class Base
370 print(f"\nModel Class Base:\n{model.__class__.__base__}")
371
372 # Print Model Class Doc
373 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
374
375 # Print Model Class Name
376 print(f"\nModel Class Name:\n{model.__class__.__name__}")
377
378 # Print Model Class Doc
379 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
380
381 # Print Model Class Bases
382 print(f"\nModel Class Bases:\n{model.__class__.bases}")
383
384 # Print Model Class Base
385 print(f"\nModel Class Base:\n{model.__class__.__base__}")
386
387 # Print Model Class Doc
388 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
389
390 # Print Model Class Name
391 print(f"\nModel Class Name:\n{model.__class__.__name__}")
392
393 # Print Model Class Doc
394 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
395
396 # Print Model Class Bases
397 print(f"\nModel Class Bases:\n{model.__class__.bases}")
398
399 # Print Model Class Base
400 print(f"\nModel Class Base:\n{model.__class__.__base__}")
401
402 # Print Model Class Doc
403 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
404
405 # Print Model Class Name
406 print(f"\nModel Class Name:\n{model.__class__.__name__}")
407
408 # Print Model Class Doc
409 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
410
411 # Print Model Class Bases
412 print(f"\nModel Class Bases:\n{model.__class__.bases}")
413
414 # Print Model Class Base
415 print(f"\nModel Class Base:\n{model.__class__.__base__}")
416
417 # Print Model Class Doc
418 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
419
420 # Print Model Class Name
421 print(f"\nModel Class Name:\n{model.__class__.__name__}")
422
423 # Print Model Class Doc
424 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
425
426 # Print Model Class Bases
427 print(f"\nModel Class Bases:\n{model.__class__.bases}")
428
429 # Print Model Class Base
430 print(f"\nModel Class Base:\n{model.__class__.__base__}")
431
432 # Print Model Class Doc
433 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
434
435 # Print Model Class Name
436 print(f"\nModel Class Name:\n{model.__class__.__name__}")
437
438 # Print Model Class Doc
439 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
440
441 # Print Model Class Bases
442 print(f"\nModel Class Bases:\n{model.__class__.bases}")
443
444 # Print Model Class Base
445 print(f"\nModel Class Base:\n{model.__class__.__base__}")
446
447 # Print Model Class Doc
448 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
449
450 # Print Model Class Name
451 print(f"\nModel Class Name:\n{model.__class__.__name__}")
452
453 # Print Model Class Doc
454 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
455
456 # Print Model Class Bases
457 print(f"\nModel Class Bases:\n{model.__class__.bases}")
458
459 # Print Model Class Base
460 print(f"\nModel Class Base:\n{model.__class__.__base__}")
461
462 # Print Model Class Doc
463 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
464
465 # Print Model Class Name
466 print(f"\nModel Class Name:\n{model.__class__.__name__}")
467
468 # Print Model Class Doc
469 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
470
471 # Print Model Class Bases
472 print(f"\nModel Class Bases:\n{model.__class__.bases}")
473
474 # Print Model Class Base
475 print(f"\nModel Class Base:\n{model.__class__.__base__}")
476
477 # Print Model Class Doc
478 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
479
480 # Print Model Class Name
481 print(f"\nModel Class Name:\n{model.__class__.__name__}")
482
483 # Print Model Class Doc
484 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
485
486 # Print Model Class Bases
487 print(f"\nModel Class Bases:\n{model.__class__.bases}")
488
489 # Print Model Class Base
490 print(f"\nModel Class Base:\n{model.__class__.__base__}")
491
492 # Print Model Class Doc
493 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
494
495 # Print Model Class Name
496 print(f"\nModel Class Name:\n{model.__class__.__name__}")
497
498 # Print Model Class Doc
499 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
500
501 # Print Model Class Bases
502 print(f"\nModel Class Bases:\n{model.__class__.bases}")
503
504 # Print Model Class Base
505 print(f"\nModel Class Base:\n{model.__class__.__base__}")
506
507 # Print Model Class Doc
508 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
509
510 # Print Model Class Name
511 print(f"\nModel Class Name:\n{model.__class__.__name__}")
512
513 # Print Model Class Doc
514 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
515
516 # Print Model Class Bases
517 print(f"\nModel Class Bases:\n{model.__class__.bases}")
518
519 # Print Model Class Base
520 print(f"\nModel Class Base:\n{model.__class__.__base__}")
521
522 # Print Model Class Doc
523 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
524
525 # Print Model Class Name
526 print(f"\nModel Class Name:\n{model.__class__.__name__}")
527
528 # Print Model Class Doc
529 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
530
531 # Print Model Class Bases
532 print(f"\nModel Class Bases:\n{model.__class__.bases}")
533
534 # Print Model Class Base
535 print(f"\nModel Class Base:\n{model.__class__.__base__}")
536
537 # Print Model Class Doc
538 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
539
540 # Print Model Class Name
541 print(f"\nModel Class Name:\n{model.__class__.__name__}")
542
543 # Print Model Class Doc
544 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
545
546 # Print Model Class Bases
547 print(f"\nModel Class Bases:\n{model.__class__.bases}")
548
549 # Print Model Class Base
550 print(f"\nModel Class Base:\n{model.__class__.__base__}")
551
552 # Print Model Class Doc
553 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
554
555 # Print Model Class Name
556 print(f"\nModel Class Name:\n{model.__class__.__name__}")
557
558 # Print Model Class Doc
559 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
560
561 # Print Model Class Bases
562 print(f"\nModel Class Bases:\n{model.__class__.bases}")
563
564 # Print Model Class Base
565 print(f"\nModel Class Base:\n{model.__class__.__base__}")
566
567 # Print Model Class Doc
568 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
569
570 # Print Model Class Name
571 print(f"\nModel Class Name:\n{model.__class__.__name__}")
572
573 # Print Model Class Doc
574 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
575
576 # Print Model Class Bases
577 print(f"\nModel Class Bases:\n{model.__class__.bases}")
578
579 # Print Model Class Base
580 print(f"\nModel Class Base:\n{model.__class__.__base__}")
581
582 # Print Model Class Doc
583 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
584
585 # Print Model Class Name
586 print(f"\nModel Class Name:\n{model.__class__.__name__}")
587
588 # Print Model Class Doc
589 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
590
591 # Print Model Class Bases
592 print(f"\nModel Class Bases:\n{model.__class__.bases}")
593
594 # Print Model Class Base
595 print(f"\nModel Class Base:\n{model.__class__.__base__}")
596
597 # Print Model Class Doc
598 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
599
600 # Print Model Class Name
601 print(f"\nModel Class Name:\n{model.__class__.__name__}")
602
603 # Print Model Class Doc
604 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
605
606 # Print Model Class Bases
607 print(f"\nModel Class Bases:\n{model.__class__.bases}")
608
609 # Print Model Class Base
610 print(f"\nModel Class Base:\n{model.__class__.__base__}")
611
612 # Print Model Class Doc
613 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
614
615 # Print Model Class Name
616 print(f"\nModel Class Name:\n{model.__class__.__name__}")
617
618 # Print Model Class Doc
619 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
620
621 # Print Model Class Bases
622 print(f"\nModel Class Bases:\n{model.__class__.bases}")
623
624 # Print Model Class Base
625 print(f"\nModel Class Base:\n{model.__class__.__base__}")
626
627 # Print Model Class Doc
628 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
629
630 # Print Model Class Name
631 print(f"\nModel Class Name:\n{model.__class__.__name__}")
632
633 # Print Model Class Doc
634 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
635
636 # Print Model Class Bases
637 print(f"\nModel Class Bases:\n{model.__class__.bases}")
638
639 # Print Model Class Base
640 print(f"\nModel Class Base:\n{model.__class__.__base__}")
641
642 # Print Model Class Doc
643 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
644
645 # Print Model Class Name
646 print(f"\nModel Class Name:\n{model.__class__.__name__}")
647
648 # Print Model Class Doc
649 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
650
651 # Print Model Class Bases
652 print(f"\nModel Class Bases:\n{model.__class__.bases}")
653
654 # Print Model Class Base
655 print(f"\nModel Class Base:\n{model.__class__.__base__}")
656
657 # Print Model Class Doc
658 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
659
660 # Print Model Class Name
661 print(f"\nModel Class Name:\n{model.__class__.__name__}")
662
663 # Print Model Class Doc
664 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
665
666 # Print Model Class Bases
667 print(f"\nModel Class Bases:\n{model.__class__.bases}")
668
669 # Print Model Class Base
670 print(f"\nModel Class Base:\n{model.__class__.__base__}")
671
672 # Print Model Class Doc
673 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
674
675 # Print Model Class Name
676 print(f"\nModel Class Name:\n{model.__class__.__name__}")
677
678 # Print Model Class Doc
679 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
680
681 # Print Model Class Bases
682 print(f"\nModel Class Bases:\n{model.__class__.bases}")
683
684 # Print Model Class Base
685 print(f"\nModel Class Base:\n{model.__class__.__base__}")
686
687 # Print Model Class Doc
688 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
689
690 # Print Model Class Name
691 print(f"\nModel Class Name:\n{model.__class__.__name__}")
692
693 # Print Model Class Doc
694 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
695
696 # Print Model Class Bases
697 print(f"\nModel Class Bases:\n{model.__class__.bases}")
698
699 # Print Model Class Base
700 print(f"\nModel Class Base:\n{model.__class__.__base__}")
701
702 # Print Model Class Doc
703 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
704
705 # Print Model Class Name
706 print(f"\nModel Class Name:\n{model.__class__.__name__}")
707
708 # Print Model Class Doc
709 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
710
711 # Print Model Class Bases
712 print(f"\nModel Class Bases:\n{model.__class__.bases}")
713
714 # Print Model Class Base
715 print(f"\nModel Class Base:\n{model.__class__.__base__}")
716
717 # Print Model Class Doc
718 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
719
720 # Print Model Class Name
721 print(f"\nModel Class Name:\n{model.__class__.__name__}")
722
723 # Print Model Class Doc
724 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
725
726 # Print Model Class Bases
727 print(f"\nModel Class Bases:\n{model.__class__.bases}")
728
729 # Print Model Class Base
730 print(f"\nModel Class Base:\n{model.__class__.__base__}")
731
732 # Print Model Class Doc
733 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
734
735 # Print Model Class Name
736 print(f"\nModel Class Name:\n{model.__class__.__name__}")
737
738 # Print Model Class Doc
739 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
740
741 # Print Model Class Bases
742 print(f"\nModel Class Bases:\n{model.__class__.bases}")
743
744 # Print Model Class Base
745 print(f"\nModel Class Base:\n{model.__class__.__base__}")
746
747 # Print Model Class Doc
748 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
749
750 # Print Model Class Name
751 print(f"\nModel Class Name:\n{model.__class__.__name__}")
752
753 # Print Model Class Doc
754 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
755
756 # Print Model Class Bases
757 print(f"\nModel Class Bases:\n{model.__class__.bases}")
758
759 # Print Model Class Base
760 print(f"\nModel Class Base:\n{model.__class__.__base__}")
761
762 # Print Model Class Doc
763 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
764
765 # Print Model Class Name
766 print(f"\nModel Class Name:\n{model.__class__.__name__}")
767
768 # Print Model Class Doc
769 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
770
771 # Print Model Class Bases
772 print(f"\nModel Class Bases:\n{model.__class__.bases}")
773
774 # Print Model Class Base
775 print(f"\nModel Class Base:\n{model.__class__.__base__}")
776
777 # Print Model Class Doc
778 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
779
780 # Print Model Class Name
781 print(f"\nModel Class Name:\n{model.__class__.__name__}")
782
783 # Print Model Class Doc
784 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
785
786 # Print Model Class Bases
787 print(f"\nModel Class Bases:\n{model.__class__.bases}")
788
789 # Print Model Class Base
790 print(f"\nModel Class Base:\n{model.__class__.__base__}")
791
792 # Print Model Class Doc
793 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
794
795 # Print Model Class Name
796 print(f"\nModel Class Name:\n{model.__class__.__name__}")
797
798 # Print Model Class Doc
799 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
800
801 # Print Model Class Bases
802 print(f"\nModel Class Bases:\n{model.__class__.bases}")
803
804 # Print Model Class Base
805 print(f"\nModel Class Base:\n{model.__class__.__base__}")
806
807 # Print Model Class Doc
808 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
809
810 # Print Model Class Name
811 print(f"\nModel Class Name:\n{model.__class__.__name__}")
812
813 # Print Model Class Doc
814 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
815
816 # Print Model Class Bases
817 print(f"\nModel Class Bases:\n{model.__class__.bases}")
818
819 # Print Model Class Base
820 print(f"\nModel Class Base:\n{model.__class__.__base__}")
821
822 # Print Model Class Doc
823 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
824
825 # Print Model Class Name
826 print(f"\nModel Class Name:\n{model.__class__.__name__}")
827
828 # Print Model Class Doc
829 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
830
831 # Print Model Class Bases
832 print(f"\nModel Class Bases:\n{model.__class__.bases}")
833
834 # Print Model Class Base
835 print(f"\nModel Class Base:\n{model.__class__.__base__}")
836
837 # Print Model Class Doc
838 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
839
840 # Print Model Class Name
841 print(f"\nModel Class Name:\n{model.__class__.__name__}")
842
843 # Print Model Class Doc
844 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
845
846 # Print Model Class Bases
847 print(f"\nModel Class Bases:\n{model.__class__.bases}")
848
849 # Print Model Class Base
850 print(f"\nModel Class Base:\n{model.__class__.__base__}")
851
852 # Print Model Class Doc
853 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
854
855 # Print Model Class Name
856 print(f"\nModel Class Name:\n{model.__class__.__name__}")
857
858 # Print Model Class Doc
859 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
860
861 # Print Model Class Bases
862 print(f"\nModel Class Bases:\n{model.__class__.bases}")
863
864 # Print Model Class Base
865 print(f"\nModel Class Base:\n{model.__class__.__base__}")
866
867 # Print Model Class Doc
868 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
869
870 # Print Model Class Name
871 print(f"\nModel Class Name:\n{model.__class__.__name__}")
872
873 # Print Model Class Doc
874 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
875
876 # Print Model Class Bases
877 print(f"\nModel Class Bases:\n{model.__class__.bases}")
878
879 # Print Model Class Base
880 print(f"\nModel Class Base:\n{model.__class__.__base__}")
881
882 # Print Model Class Doc
883 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
884
885 # Print Model Class Name
886 print(f"\nModel Class Name:\n{model.__class__.__name__}")
887
888 # Print Model Class Doc
889 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
890
891 # Print Model Class Bases
892 print(f"\nModel Class Bases:\n{model.__class__.bases}")
893
894 # Print Model Class Base
895 print(f"\nModel Class Base:\n{model.__class__.__base__}")
896
897 # Print Model Class Doc
898 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
899
900 # Print Model Class Name
901 print(f"\nModel Class Name:\n{model.__class__.__name__}")
902
903 # Print Model Class Doc
904 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
905
906 # Print Model Class Bases
907 print(f"\nModel Class Bases:\n{model.__class__.bases}")
908
909 # Print Model Class Base
910 print(f"\nModel Class Base:\n{model.__class__.__base__}")
911
912 # Print Model Class Doc
913 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
914
915 # Print Model Class Name
916 print(f"\nModel Class Name:\n{model.__class__.__name__}")
917
918 # Print Model Class Doc
919 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
920
921 # Print Model Class Bases
922 print(f"\nModel Class Bases:\n{model.__class__.bases}")
923
924 # Print Model Class Base
925 print(f"\nModel Class Base:\n{model.__class__.__base__}")
926
927 # Print Model Class Doc
928 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
929
930 # Print Model Class Name
931 print(f"\nModel Class Name:\n{model.__class__.__name__}")
932
933 # Print Model Class Doc
934 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
935
936 # Print Model Class Bases
937 print(f"\nModel Class Bases:\n{model.__class__.bases}")
938
939 # Print Model Class Base
940 print(f"\nModel Class Base:\n{model.__class__.__base__}")
941
942 # Print Model Class Doc
943 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
944
945 # Print Model Class Name
946 print(f"\nModel Class Name:\n{model.__class__.__name__}")
947
948 # Print Model Class Doc
949 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
950
951 # Print Model Class Bases
952 print(f"\nModel Class Bases:\n{model.__class__.bases}")
953
954 # Print Model Class Base
955 print(f"\nModel Class Base:\n{model.__class__.__base__}")
956
957 # Print Model Class Doc
958 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
959
960 # Print Model Class Name
961 print(f"\nModel Class Name:\n{model.__class__.__name__}")
962
963 # Print Model Class Doc
964 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
965
966 # Print Model Class Bases
967 print(f"\nModel Class Bases:\n{model.__class__.bases}")
968
969 # Print Model Class Base
970 print(f"\nModel Class Base:\n{model.__class__.__base__}")
971
972 # Print Model Class Doc
973 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
974
975 # Print Model Class Name
976 print(f"\nModel Class Name:\n{model.__class__.__name__}")
977
978 # Print Model Class Doc
979 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
980
981 # Print Model Class Bases
982 print(f"\nModel Class Bases:\n{model.__class__.bases}")
983
984 # Print Model Class Base
985 print(f"\nModel Class Base:\n{model.__class__.__base__}")
986
987 # Print Model Class Doc
988 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
989
990 # Print Model Class Name
991 print(f"\nModel Class Name:\n{model.__class__.__name__}")
992
993 # Print Model Class Doc
994 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
995
996 # Print Model Class Bases
997 print(f"\nModel Class Bases:\n{model.__class__.bases}")
998
999 # Print Model Class Base
1000 print(f"\nModel Class Base:\n{model.__class__.__base__}")
1001
1002 # Print Model Class Doc
1003 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1004
1005 # Print Model Class Name
1006 print(f"\nModel Class Name:\n{model.__class__.__name__}")
1007
1008 # Print Model Class Doc
1009 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1010
1011 # Print Model Class Bases
1012 print(f"\nModel Class Bases:\n{model.__class__.bases}")
1013
1014 # Print Model Class Base
1015 print(f"\nModel Class Base:\n{model.__class__.__base__}")
1016
1017 # Print Model Class Doc
1018 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1019
1020 # Print Model Class Name
1021 print(f"\nModel Class Name:\n{model.__class__.__name__}")
1022
1023 # Print Model Class Doc
1024 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1025
1026 # Print Model Class Bases
1027 print(f"\nModel Class Bases:\n{model.__class__.bases}")
1028
1029 # Print Model Class Base
1030 print(f"\nModel Class Base:\n{model.__class__.__base__}")
1031
1032 # Print Model Class Doc
1033 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1034
1035 # Print Model Class Name
1036 print(f"\nModel Class Name:\n{model.__class__.__name__}")
1037
1038 # Print Model Class Doc
1039 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1040
1041 # Print Model Class Bases
1042 print(f"\nModel Class Bases:\n{model.__class__.bases}")
1043
1044 # Print Model Class Base
1045 print(f"\nModel Class Base:\n{model.__class__.__base__}")
1046
1047 # Print Model Class Doc
1048 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1049
1050 # Print Model Class Name
1051 print(f"\nModel Class Name:\n{model.__class__.__name__}")
1052
1053 # Print Model Class Doc
1054 print(f"\nModel Class Doc:\n{model.__class__.__doc__}")
1055
1056 # Print Model Class Bases
1057 print(f"\nModel Class Bases:\n{model.__class__.bases}")
1058
1059 # Print Model Class Base
1060 print(f"\nModel Class Base:\n{model.__class__.__base__}")
1061
1062 # Print Model Class Doc
1063 print(f"\nModel Class Doc:\n
```

```
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": f1}
73
74     print(f"{name} Accuracy: {acc:.4f}")
75     print(f"{name} F1 Score: {f1:.4f}")
76
```

```
77 # ----- Results Table -----
78 results_df = pd.DataFrame(results).T
79 print("\n📊 Model Comparison:")
80 print(results_df)
81
82 results_df.plot(kind="bar", figsize=(8,5))
83 plt.title("Model Comparison")
84 plt.ylim(0,1)
85 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

```
1 from google.colab import drive  
2 drive.mount('/content/drive')  
3  
4  
5 import torch  
6 from datetime import datetime  
7  
8 CHECKPOINT_PATH = "/content/drive/MyDrive/knee_OA_  
9  
10 checkpoint = {  
11     "model architecture": "mobilenet v2",
```