retfound

December 4, 2025

[25]:

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split, Dataset
import os
```

[26]:

```
np.random.seed(0)
```

[27]:

```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
elif torch.cuda.is_available():
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[ ]:

```
from PIL import Image
import cv2
import numpy

class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None, apply_clahe=True):
        with open(file_path, 'rb') as f:
            data = pickle.load(f)
        self.images = data['images']
        self.labels = data['labels'].reshape(-1)
        self.transform = transform
        self.apply_clahe = apply_clahe
    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image = self.images[idx]
        label = int(self.labels[idx])

        if self.apply_clahe:
            image = self.clahe_preprocess(image)

        image = Image.fromarray(image.astype('uint8'))

        if self.transform:
            image = self.transform(image)

        return image, label

    @staticmethod
    def clahe_preprocess(img):
        lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
        l, a, b = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        l2 = clahe.apply(l)
        lab2 = cv2.merge((l2, a, b))
        return cv2.cvtColor(lab2, cv2.COLOR_LAB2RGB)
```

[ ]:

```
class RetFoundDataset(Dataset):
    def __init__(self, images, labels=None, clahe=True):
        self.images = images
        self.labels = labels
        self.clahe = clahe

    def __getitem__(self, idx):
        img = self.images[idx].astype('uint8')

        # Preprocessing
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

        if self.clahe:
            c = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
            gray = c.apply(gray)

        img3 = np.stack([gray]*3, axis=-1)   # RetFound wants 3 channels
        enc = processor(img3, return_tensors="pt")  # RetFound processor

        pixel_values = enc["pixel_values"].squeeze(0)

        if self.labels is None:
            return pixel_values
        else:
            return pixel_values, int(self.labels[idx])
```

[29]:

```
dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl')
```

[30]:

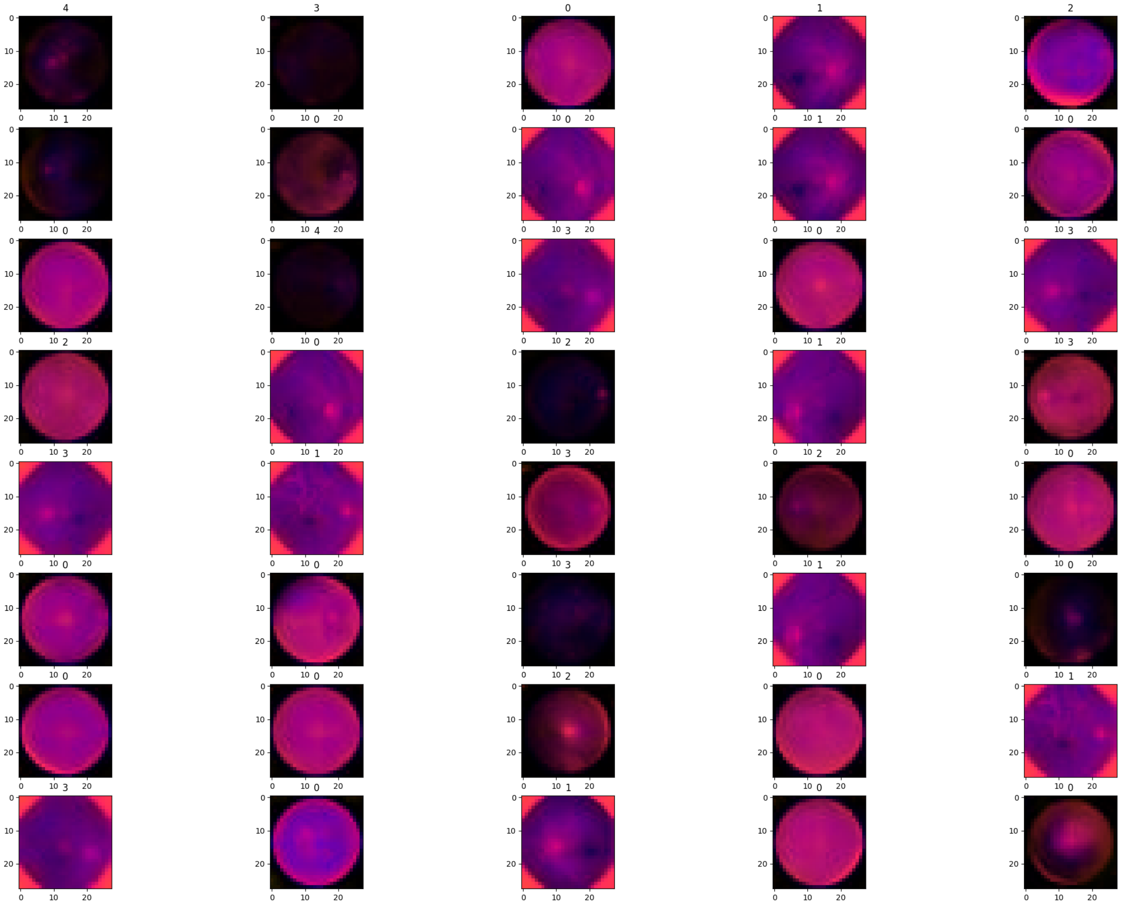
```
raw_images, raw_labels = dataset.images, dataset.labels
```

[31]:

```
from skimage import exposure
import cv2

bad_images= []

for i, img in enumerate(raw_images):

    img_norm = img / 255.0

    if img_norm.std() < 0.07:
        bad_images.append(i)

    if img_norm.mean() > 0.25:
        bad_images.append(i)


    r, g, b = img_norm[:,:,0], img_norm[:,:,1], img_norm[:,:,2]
    if r.mean() > g.mean() * 1.8 and r.mean() > b.mean() * 1.8:
        bad_images.append(i)


"""
for i, img in enumerate(raw_labels):
    if img == 4:
        bad_images.append(i)
"""




print(f"Found {len(bad_images)} bad images out of {len(raw_images)}")

mask = np.ones(len(raw_images), dtype=bool)
mask[bad_images] = False
```

```python
n_show = min(40, len(bad_images))

if n_show == 0:
    print("aucune image")

else:
    plt.figure(figsize=(28,28))
    for i,idx in enumerate(bad_images[:n_show]):
        plt.subplot(11, 5, i+1)
        plt.title(raw_labels[idx])
        plt.imshow(raw_images[idx])
    plt.show()

cleaned_images = raw_images[mask]
cleaned_labels = raw_labels[mask]

dataset.images = cleaned_images
dataset.labels = cleaned_labels
```

```
Found 55 bad images out of 1080
```



[32]:

```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[ ]:

```
"""

train_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.Grayscale(num_output_channels=1),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.25])
])

val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    #transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5],
                         [0.25, 0.25, 0.25])]
)
"""
```

[34]:

```
from sklearn.model_selection import train_test_split

labels = dataset.labels
idx = np.arange(len(dataset))
train_idx, valid_idx = train_test_split(idx, test_size=0.2, stratify=labels,␣
 ↪random_state=42)
```

[35]:

```
class TransformSubset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, idx):
        image, label = self.subset[idx]
        if self.transform:
            image = self.transform(image)
        return image, label

    def __len__(self):
        return len(self.subset)
```

[36]:

```
from torch.utils.data import WeightedRandomSampler
from sklearn.utils.class_weight import compute_class_weight
```

```
train_labels = dataset.labels[train_idx]

classes = np.unique(train_labels)

class_weights = compute_class_weight('balanced', classes=classes,␣
 ↪y=train_labels)

sample_weights = class_weights[train_labels]

sampler = WeightedRandomSampler(
    weights=sample_weights,
    num_samples=len(sample_weights),
    replacement=True
)
```

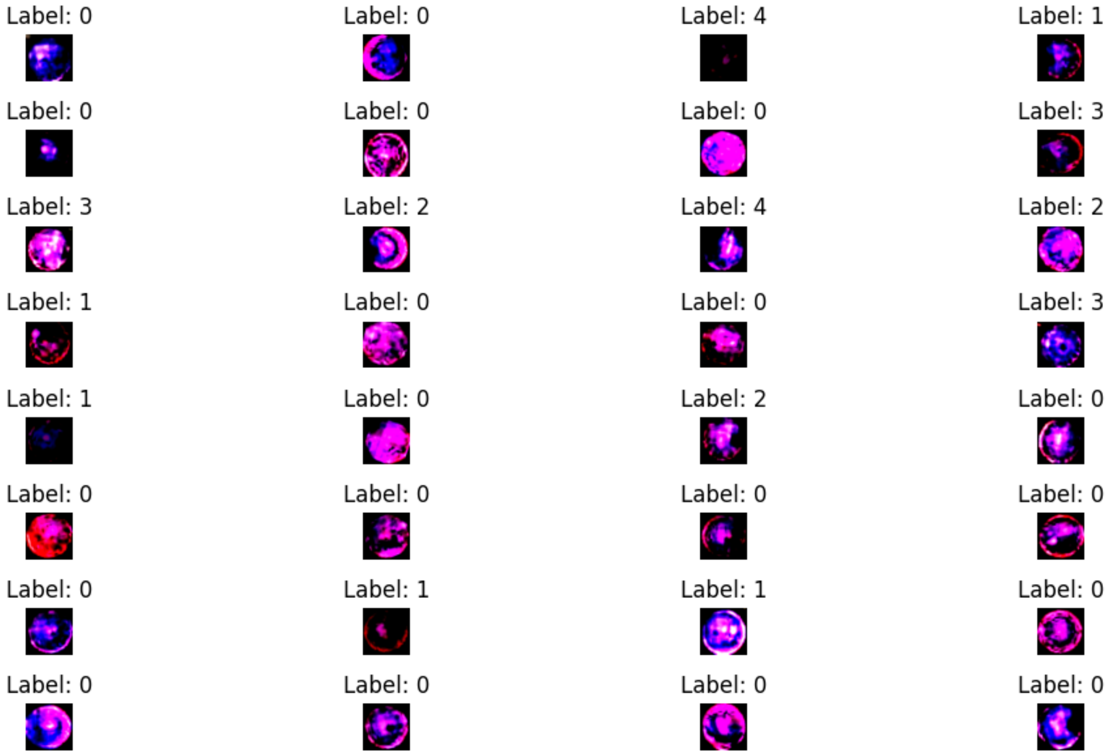
[ ]:

```
from torch.utils.data import Subset

train_dataset = Subset(dataset, train_idx)
train_data = TransformSubset(train_dataset, train_transform)

val_dataset = Subset(dataset, valid_idx)
val_data = TransformSubset(val_dataset, val_transform)
# sampler=sampler,
train_loader = DataLoader(train_data, batch_size=8,shuffle=True,␣
 ↪pin_memory=True)
val_loader = DataLoader(val_data, batch_size=8, shuffle=False, pin_memory=True)
```

[38]:

```
# Visualiser directement sans dénormaliser
images, labels = next(iter(train_loader))

fig, axes = plt.subplots(8, 4, figsize=(12, 6))
for i in range(32):
    ax = axes[i // 4, i % 4]
    # Les images sont normalisées, donc elles auront des teintes bizarres
    # mais vous verrez quand même le contenu
    img_np = images[i].cpu().permute(1, 2, 0).numpy()
    ax.imshow(img_np)
    ax.set_title(f"Label: {labels[i].item()}")
    ax.axis('off')
plt.tight_layout()
plt.show()
```

```
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/torch/utils/data/dataloader.py:692: UserWarning: 'pin_memory'
argument is set as true but not supported on MPS now, device pinned memory won't
be used.
  warnings.warn(warn_msg)
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.5764706..1.7647059].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.309804].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7960784..0.43137264].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.3882353].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.9686275..1.8901961].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.9372549].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.9843137..1.9529412].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8901961..1.0901961].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.9372549..1.8745098].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..0.9490197].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.6705883..1.7176471].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.9372549].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7490196..0.4627452].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.6392157].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.654902].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.9686275..1.8431373].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8117647..1.2784314].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7333333..1.3254902].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.827451].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.827451..1.6392157].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.6862745..0.90196085].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8431373..1.5450981].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7019608..2.0].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7803922..1.5294118].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.5921569].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0..1.827451].
```



```
[ ]: #from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights

     # model = CNNNet(num_classes=5)

     from transformers import AutoImageProcessor, AutoModelForImageClassification

     processor = AutoImageProcessor.from_pretrained("microsoft/retfound-base")
```

```
model = AutoModelForImageClassification.from_pretrained(
    "microsoft/retfound-base",
    num_labels=5,
    ignore_mismatched_sizes=True
)
```

[41]:

```
model.to(device)
```

[41]:

```
CNNNet(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Dropout(p=0.25, inplace=False)
  )
  (conv3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): AdaptiveAvgPool2d(output_size=1)
    (4): Dropout(p=0.4, inplace=False)
  )
  (fc1): Linear(in_features=64, out_features=32, bias=True)
  (dropout_fc): Dropout(p=0.4, inplace=False)
  (fc2): Linear(in_features=32, out_features=5, bias=True)
)
```

[42]:

```
weights_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)
```

[ ]:

```
from torch.optim import AdamW

criterion = nn.CrossEntropyLoss(weight=weights_tensor, label_smoothing=0.1)

optimizer = optim.AdamW(model.parameters(), lr=2e-5,
                        #weight_decay=10e-2
                        )
```

[ ]:

```
from sklearn.metrics import balanced_accuracy_score

for epoch in range(10):
    model.train()
    running = 0

    for pixel_values, labels in train_loader:
        pixel_values = pixel_values.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(pixel_values=pixel_values, labels=labels)
        loss = outputs.loss

        loss.backward()
        optimizer.step()

        running += loss.item()

    print(f"Epoch {epoch+1}, Train Loss = {running / len(train_loader):.4f}")

    # Validation
    model.eval()
    preds, gts = [], []

    with torch.no_grad():
        for pixel_values, labels in val_loader:
            pixel_values = pixel_values.to(device)
            labels = labels.to(device)

            outputs = model(pixel_values=pixel_values)
            pred = outputs.logits.argmax(dim=1)

            preds.extend(pred.cpu().numpy())
            gts.extend(labels.cpu().numpy())

    bal_acc = balanced_accuracy_score(gts, preds)

    print(f"Bal Acc = {bal_acc:.4f}")
```

```
Epoch 1, Train Loss: 1.7154, Val Loss: 1.6924
Epoch 2, Train Loss: 1.6893, Val Loss: 1.6920
Epoch 3, Train Loss: 1.6844, Val Loss: 1.6912
Epoch 4, Train Loss: 1.6673, Val Loss: 1.6900
Epoch 5, Train Loss: 1.6662, Val Loss: 1.6887
Epoch 6, Train Loss: 1.7117, Val Loss: 1.6873
Epoch 7, Train Loss: 1.7135, Val Loss: 1.6862
Epoch 8, Train Loss: 1.6149, Val Loss: 1.6853
Epoch 9, Train Loss: 1.6180, Val Loss: 1.6838
Epoch 10, Train Loss: 1.6420, Val Loss: 1.6827
Epoch 11, Train Loss: 1.6961, Val Loss: 1.6819
Epoch 12, Train Loss: 1.6467, Val Loss: 1.6815
Epoch 13, Train Loss: 1.7178, Val Loss: 1.6809
Epoch 14, Train Loss: 1.6662, Val Loss: 1.6800
Epoch 15, Train Loss: 1.7274, Val Loss: 1.6793
Epoch 16, Train Loss: 1.6679, Val Loss: 1.6788
Epoch 17, Train Loss: 1.6569, Val Loss: 1.6779
Epoch 18, Train Loss: 1.6442, Val Loss: 1.6768
Epoch 19, Train Loss: 1.7536, Val Loss: 1.6755
Epoch 20, Train Loss: 1.6030, Val Loss: 1.6749
```

[45]:

```
model.eval()

all_preds = []
all_labels = []

from sklearn.metrics import accuracy_score, recall_score,␣
 ↪balanced_accuracy_score, classification_report, confusion_matrix

with torch.no_grad():
    for images, labels in val_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())


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

[46]:

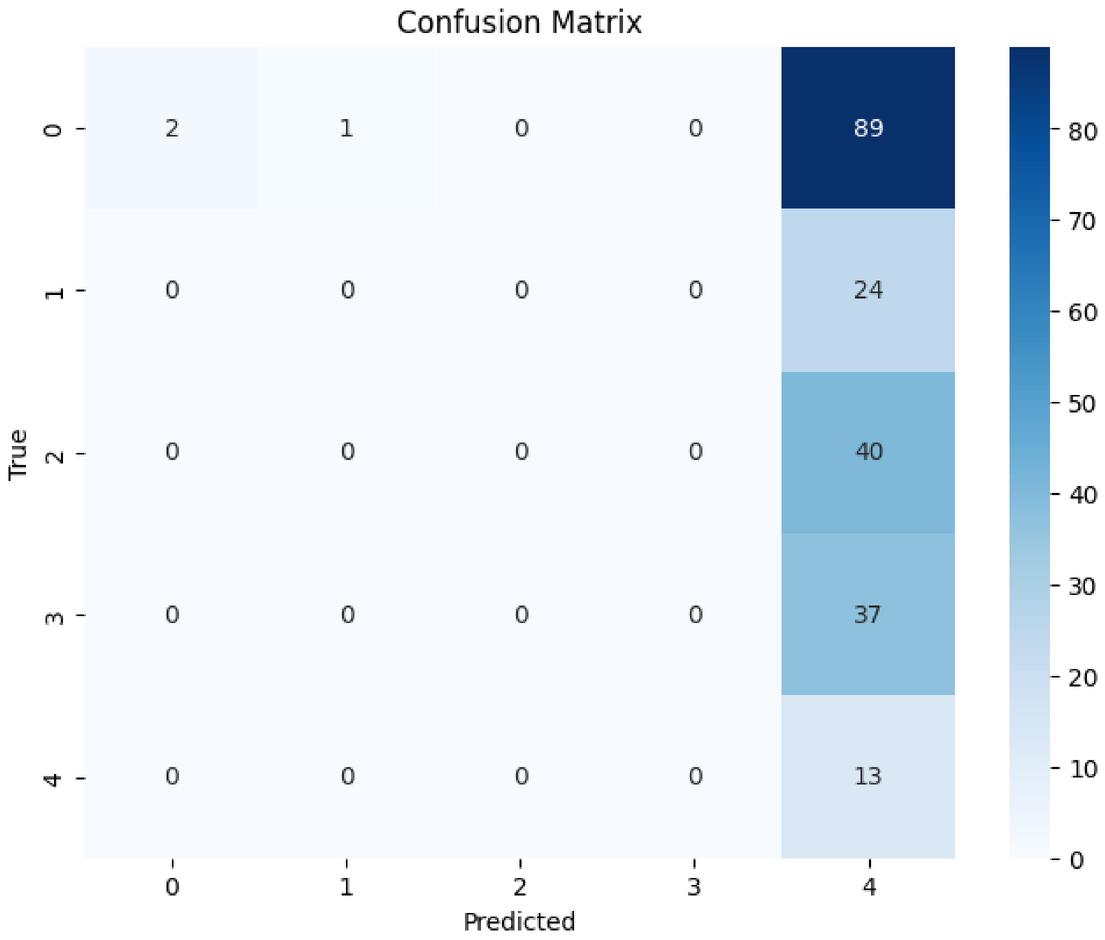
```
bal_acc = balanced_accuracy_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds, average='macro')
acc = accuracy_score(all_labels, all_preds)

print(f"Validation Balanced Accuracy: {bal_acc:.4f}")
print(f"Validation Recall: {recall:.4f}")
print(f"Validation Accuracy: {acc:.4f}")
print(classification_report(all_labels, all_preds, digits=4))
```

```
Validation Balanced Accuracy: 0.2043
Validation Recall: 0.2043
Validation Accuracy: 0.0728
              precision    recall  f1-score   support

           0     1.0000    0.0217    0.0426        92
           1     0.0000    0.0000    0.0000        24
           2     0.0000    0.0000    0.0000        40
           3     0.0000    0.0000    0.0000        37
           4     0.0640    1.0000    0.1204        13

    accuracy                         0.0728       206
   macro avg     0.2128    0.2043    0.0326       206
weighted avg     0.4506    0.0728    0.0266       206
```

```
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.
13/site-packages/sklearn/metrics/_classification.py:1731:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

[47]:

```python
import seaborn as sns

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



[48]:
```
x = model.conv1(x)
x = model.conv2(x)
x = model.conv3(x)
x = model.conv4(x)
embedding = x.view(x.size(0), -1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[48], line 1
----> 1 x = model.conv1(x)
      2 x = model.conv2(x)
      3 x = model.conv3(x)

NameError: name 'x' is not defined
```

```
[ ]: torch.save({
         "model_state_dict": model.state_dict(),
         "num_classes": 5,
     }, "efficientnet_finetuned.pth")
```

```
[ ]: checkpoint = torch.load("efficientnet_finetuned.pth", map_location=device,␣
      ↪weights_only=False)

     model = efficientnet_b0(weights=None)

     model.classifier[1] = nn.Linear(model.classifier[1].in_features,␣
      ↪checkpoint["num_classes"])

     model.load_state_dict(checkpoint["model_state_dict"])
     model.to(device)
     model.eval()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[26], line 7
      3 model = efficientnet_b0(weights=None)
      5 model.classifier[1] = nn.Linear(model.classifier[1].in_features,␣
  ↪checkpoint["num_classes"])
----> 7 model.load_state_dict(checkpoint[                  ])
      8 model.to(device)
      9 model.eval()

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
  ↪torch/nn/modules/module.py:2629, in Module.load_state_dict(self, state_dict,␣
  ↪strict, assign)
   2621         error_msgs.insert(
   2622             0,
   2623             "Missing key(s) in state_dict: {}. ".format(
   2624                 ", ".join(f'"{k}"' for k in missing_keys)
   2625             ),
   2626         )
   2628 if len(error_msgs) > 0:
-> 2629     raise RuntimeError(
   2630         "Error(s) in loading state_dict for {}:\n\t{}".format(
   2631             self.__class__.__name__, "\n\t".join(error_msgs)
   2632         )
   2633     )
   2634 return _IncompatibleKeys(missing_keys, unexpected_keys)

RuntimeError: Error(s) in loading state_dict for EfficientNet:
      Missing key(s) in state_dict: "features.0.0.weight", "features.0.1.
↪weight", "features.0.1.bias", "features.0.1.running_mean", "features.0.1.
↪running_var", "features.1.0.block.0.0.weight", "features.1.0.block.0.1.
↪weight", "features.1.0.block.0.1.bias", "features.1.0.block.0.1.running_mean" ␣
↪"features.1.0.block.0.1.running_var", "features.1.0.block.1.fc1.weight",␣
↪"features.1.0.block.1.fc1.bias", "features.1.0.block.1.fc2.weight", "features
↪1.0.block.1.fc2.bias", "features.1.0.block.2.0.weight", "features.1.0.block.2
↪1.weight", "features.1.0.block.2.1.bias", "features.1.0.block.2.1.
↪running_mean", "features.1.0.block.2.1.running_var", "features.2.0.block.0.0.
↪weight", "features.2.0.block.0.1.weight", "features.2.0.block.0.1.bias",␣
↪"features.2.0.block.0.1.running_mean", "features.2.0.block.0.1.running_var",␣
↪"features.2.0.block.1.0.weight", "features.2.0.block.1.1.weight", "features.2
↪0.block.1.1.bias", "features.2.0.block.1.1.running_mean", "features.2.0.block
↪1.1.running_var", "features.2.0.block.2.fc1.weight", "features.2.0.block.2.fc..
↪bias", "features.2.0.block.2.fc2.weight", "features.2.0.block.2.fc2.bias",␣
↪"features.2.0.block.3.0.weight", "features.2.0.block.3.1.weight", "features.2
↪0.block.3.1.bias", "features.2.0.block.3.1.running_mean", "features.2.0.block
↪3.1.running_var", "features.2.1.block.0.0.weight", "features.2.1.block.0.1.
↪weight", "features.2.1.block.0.1.bias", "features.2.1.block.0.1.running_mean" ␣
↪"features.2.1.block.0.1.running_var", "features.2.1.block.1.0.weight",␣
↪"features.2.1.block.1.1.weight", "features.2.1.block.1.1.bias", "features.2.1
↪block.1.1.running_mean", "features.2.1.block.1.1.running_var", "features.2.1.
↪block.2.fc1.weight", "features.2.1.block.2.fc1.bias", "features.2.1.block.2.
↪fc2.weight", "features.2.1.block.2.fc2.bias", "features.2.1.block.3.0.weight" ␣
↪"features.2.1.block.3.1.weight", "features.2.1.block.3.1.bias", "features.2.1
↪block.3.1.running_mean", "features.2.1.block.3.1.running_var", "features.3.0.
↪block.0.0.weight", "features.3.0.block.0.1.weight", "features.3.0.block.0.1.
↪bias", "features.3.0.block.0.1.running_mean", "features.3.0.block.0.1.
↪running_var", "features.3.0.block.1.0.weight", "features.3.0.block.1.1.
↪weight", "features.3.0.block.1.1.bias", "features.3.0.block.1.1.running_mean" ␣
↪"features.3.0.block.1.1.running_var", "features.3.0.block.2.fc1.weight",␣
↪"features.3.0.block.2.fc1.bias", "features.3.0.block.2.fc2.weight", "features
↪3.0.block.2.fc2.bias", "features.3.0.block.3.0.weight", "features.3.0.block.3
↪1.weight", "features.3.0.block.3.1.bias", "features.3.0.block.3.1.
↪running_mean", "features.3.0.block.3.1.running_var", "features.3.1.block.0.0.
↪weight", "features.3.1.block.0.1.weight", "features.3.1.block.0.1.bias",␣
↪"features.3.1.block.0.1.running_mean", "features.3.1.block.0.1.running_var",␣
↪"features.3.1.block.1.0.weight", "features.3.1.block.1.1.weight", "features.3
↪1.block.1.1.bias", "features.3.1.block.1.1.running_mean", "features.3.1.block
↪1.1.running_var", "features.3.1.block.2.fc1.weight", "features.3.1.block.2.fc..
↪bias", "features.3.1.block.2.fc2.weight", "features.3.1.block.2.fc2.bias",␣
↪"features.3.1.block.3.0.weight", "features.3.1.block.3.1.weight", "features.3
↪1.block.3.1.bias", "features.3.1.block.3.1.running_mean", "features.3.1.block
↪3.1.running_var", "features.4.0.block.0.0.weight", "features.4.0.block.0.1.
↪weight", "features.4.0.block.0.1.bias", "features.4.0.block.0.1.running_mean" ␣
↪"features.4.0.block.0.1.running_var", "features.4.0.block.1.0.weight",␣
↪"features.4.0.block.1.1.weight", "features.4.0.block.1.1.bias", "features.4.0
↪block.1.1.running_mean", "features.4.0.block.1.1.running_var", "features.4.0.
↪block.2.fc1.weight", "features.4.0.block.2.fc1.bias", "features.4.0.block.2.
↪fc2.weight", "features.4.0.block.2.fc2.bias", "features.4.0.block.3.0.weight" ␣
↪"features.4.0.block.3.1.weight", "features.4.0.block.3.1.bias", "features.4.0
↪block.3.1.running_mean", "features.4.0.block.3.1.running_var", "features.4.1.
↪block.0.0.weight", "features.4.1.block.0.1.weight", "features.4.1.block.0.1.
↪bias", "features.4.1.block.0.1.running_mean", "features.4.1.block.0.1.
↪running_var", "features.4.1.block.1.0.weight", "features.4.1.block.1.1.
↪weight", "features.4.1.block.1.1.bias", "features.4.1.block.1.1.running_mean" ␣
↪"features.4.1.block.1.1.running_var", "features.4.1.block.2.fc1.weight",␣
↪"features.4.1.block.2.fc1.bias", "features.4.1.block.2.fc2.weight", "features
↪4.1.block.2.fc2.bias", "features.4.1.block.3.0.weight", "features.4.1.block.3
↪1.weight", "features.4.1.block.3.1.bias", "features.4.1.block.3.1.
↪running_mean", "features.4.1.block.3.1.running_var", "features.4.2.block.0.0.
↪weight", "features.4.2.block.0.1.weight", "features.4.2.block.0.1.bias",␣
↪"features.4.2.block.0.1.running_mean", "features.4.2.block.0.1.running_var",␣
↪"features.4.2.block.1.0.weight", "features.4.2.block.1.1.weight", "features.4
↪2.block.1.1.bias", "features.4.2.block.1.1.running_mean", "features.4.2.block
↪1.1.running_var", "features.4.2.block.2.fc1.weight", "features.4.2.block.2.fc..
↪bias", "features.4.2.block.2.fc2.weight", "features.4.2.block.2.fc2.bias",␣
↪"features.4.2.block.3.0.weight", "features.4.2.block.3.1.weight", "features.4
↪2.block.3.1.bias", "features.4.2.block.3.1.running_mean", "features.4.2.block
↪3.1.running_var", "features.5.0.block.0.0.weight", "features.5.0.block.0.1.
↪weight", "features.5.0.block.0.1.bias", "features.5.0.block.0.1.running_mean" ␣
↪"features.5.0.block.0.1.running_var", "features.5.0.block.1.0.weight",␣
↪"features.5.0.block.1.1.weight", "features.5.0.block.1.1.bias", "features.5.0
↪block.1.1.running_mean", "features.5.0.block.1.1.running_var", "features.5.0.
↪block.2.fc1.weight", "features.5.0.block.2.fc1.bias", "features.5.0.block.2.
↪fc2.weight", "features.5.0.block.2.fc2.bias", "features.5.0.block.3.0.weight" ␣
↪"features.5.0.block.3.1.weight", "features.5.0.block.3.1.bias", "features.5.0
```

```
        Unexpected key(s) in state_dict: "conv1.0.weight", "conv1.0.bias",␣
↪"conv1.1.weight", "conv1.1.bias", "conv1.1.running_mean", "conv1.1.
↪running_var", "conv1.1.num_batches_tracked", "conv2.0.weight", "conv2.0.bias", ␣
↪"conv2.1.weight", "conv2.1.bias", "conv2.1.running_mean", "conv2.1.
↪running_var", "conv2.1.num_batches_tracked", "conv3.0.weight", "conv3.0.bias", ␣
↪"conv3.1.weight", "conv3.1.bias", "conv3.1.running_mean", "conv3.1.
↪running_var", "conv3.1.num_batches_tracked", "conv4.0.weight", "conv4.0.bias", ␣
↪"conv4.1.weight", "conv4.1.bias", "conv4.1.running_mean", "conv4.1.
↪running_var", "conv4.1.num_batches_tracked", "fc.weight", "fc.bias".
```

[ ]:

```python
test_dataset = pickle.load(open('ift-3395-6390-kaggle-2-competition-fall-2025/
 ↪test_data.pkl', 'rb'))
test_images = test_dataset['images']

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

[ ]:

```python
class TestPKLDataset(Dataset):
    def __init__(self, images, transform=None, apply_clahe=True):
        self.images = images
        self.transform = transform
        self.apply_clahe = apply_clahe

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image = self.images[idx]

        # Appliquer CLAHE comme pour le train
        if self.apply_clahe:
            image = self.clahe_preprocess(image)

        image = Image.fromarray(image.astype('uint8'))

        if self.transform:
            image = self.transform(image)
        return image

    @staticmethod
    def clahe_preprocess(img):
        lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
        l, a, b = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        l2 = clahe.apply(l)
        lab2 = cv2.merge((l2, a, b))
        return cv2.cvtColor(lab2, cv2.COLOR_LAB2RGB)
```

[ ]:

```
test_ds = TestPKLDataset(test_images, transform=test_transform,␣
 ↪apply_clahe=True)
test_loader = DataLoader(test_ds, batch_size=64, shuffle=False, pin_memory=True)
preds = []

with torch.no_grad():
    for images in test_loader:
        images = images.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.cpu().numpy())

df = pd.DataFrame({

    "ID": np.arange(1, len(preds) + 1),
    "Label": preds
})

df.to_csv("IFT3395_YAPS_MCSV52.csv", index=False)
```