todebug22

December 4, 2025

[1]:
```
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
from pyclass import TransformSubset, CNNnet
```

[2]:
```
np.random.seed(0)
```

[3]:
```
class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None, apply_clahe=False):
        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)
```

[4]:
```
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
```

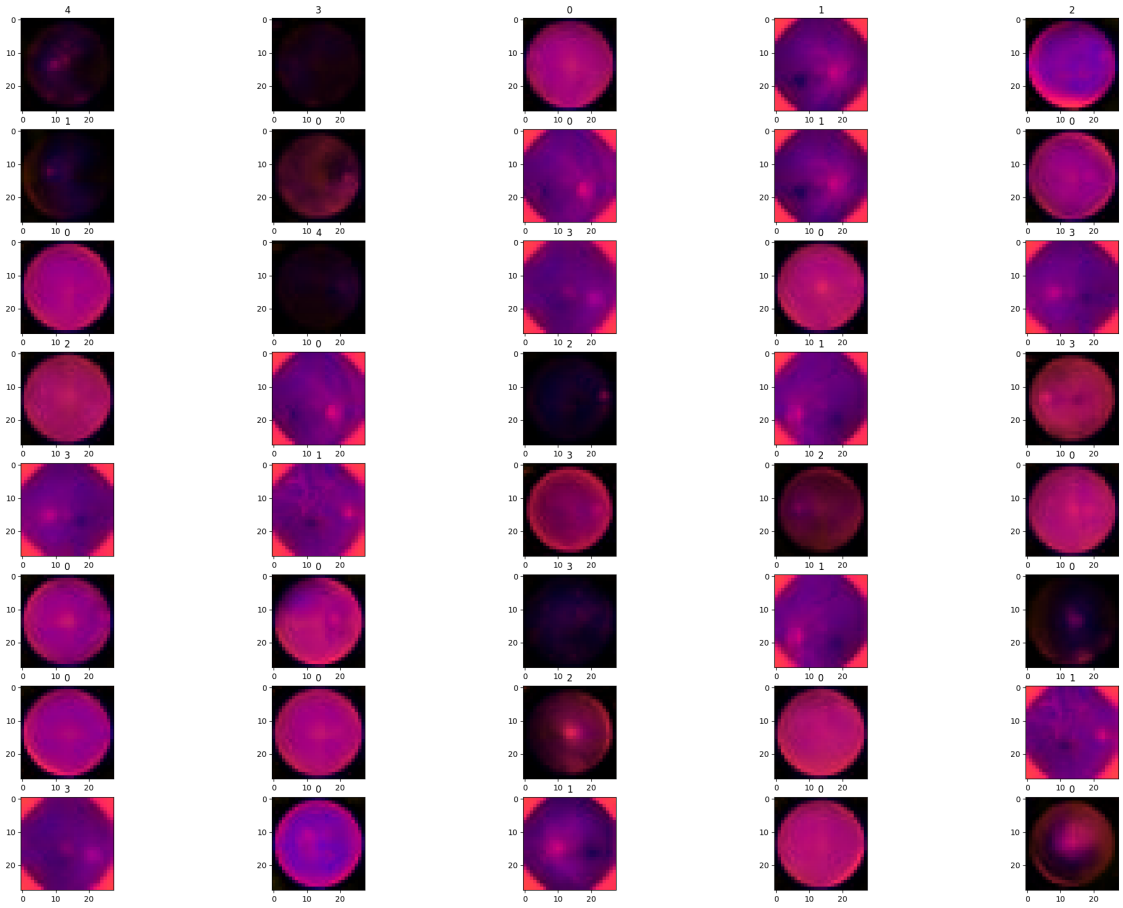
[5]:
```
dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
↪pkl')
```

[6]:
```
raw_images, raw_labels = dataset.images, dataset.labels
```

[7]:
```
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

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
```



[8]:
```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[9]:
```
IR_MEAN = [0.15, 0.15, 0.15] # Exemple si vos canaux sont sombres
IR_STD = [0.10, 0.10, 0.10]   # Exemple

train_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    #transforms.ToTensor(),
    transforms.Normalize(mean=IR_MEAN, std=IR_STD)
])

val_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    #transforms.ToTensor(),
    transforms.Normalize(mean=IR_MEAN, std=IR_STD)
])
```

[10]:
```
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)
```

[11]:
```
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)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True,␣
 ↪pin_memory=True)
val_loader   = DataLoader(val_dataset, batch_size=8, shuffle=False,␣
 ↪pin_memory=True)
```

[12]:
```
from PIL import Image
import torch
import torchvision.transforms as T
"""
# Remplacez train_loader par le loader que vous utilisez
loader = train_loader

batch = next(iter(loader))
print("Batch type:", type(batch))
if isinstance(batch, (list, tuple)):
    images, labels = batch
    print("Images container type:", type(images))
    # Si images est un tensor B,C,H,W
    if torch.is_tensor(images):
        print("images is a tensor with shape:", images.shape)
    else:
        # peut être list/ndarray d'objets
        print("First element type:", type(images[0]))
else:
    print("Batch structure unexpected:", batch)

    """
```

[12]:
```
'\n# Remplacez train_loader par le loader que vous utilisez\nloader =
train_loader\n\nbatch = next(iter(loader))\nprint("Batch type:",
type(batch))\nif isinstance(batch, (list, tuple)):\n    images, labels = batch\n
print("Images container type:", type(images))\n    # Si images est un tensor
```

```
B,C,H,W\n    if torch.is_tensor(images):\n        print("images is a tensor with
shape:", images.shape)\n    else:\n        # peut être list/ndarray d\'objets\n
print("First element type:", type(images[0]))\nelse:\n    print("Batch structure
unexpected:", batch)\n\n    '
```

[13]:
```
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)
weights_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)
```

[14]:
```
model = CNNnet().to(device)
```

[15]:
```
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
                        )
```

[16]:
```
from sklearn.metrics import accuracy_score, recall_score,␣
 ↪balanced_accuracy_score, classification_report, confusion_matrix

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

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

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)

        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 images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(pixel_values=pixel_values)
            _, pred = torch.max(outputs, 1)

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

    all_preds = np.array(preds)
    all_labels = np.array(gts)

    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"Bal Acc = {bal_acc:.4f}")
    print("Val Recall = {recall:.4f}")
    print("Val Acc: {acc.4f}")

print(classification_report(all_labels, all_preds, digits=4))
```

```
/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)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 7
      4 model.train()
      5 running = 0
----> 7 for images, labels in train_loader:
      8     pixel_values = pixel_values.to(device)
      9     labels = labels.to(device)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/dataloader.py:732, in _BaseDataLoaderIter.__next__(self)
    729 if self._sampler_iter is None:
    730     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    731     self._reset()  # type: ignore[call-arg]
--> 732 data = self._next_data()
```

```
    733 self._num_yielded += 1
    734 if (
    735     self._dataset_kind == _DatasetKind.Iterable
    736     and self._IterableDataset_len_called is not None
    737     and self._num_yielded > self._IterableDataset_len_called
    738 ):

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/dataloader.py:788, in _SingleProcessDataLoaderIter.
 ↪_next_data(self)
    786 def _next_data(self):
    787     index = self._next_index()  # may raise StopIteration
--> 788     data = self._dataset_fetcher.fetch(index)  # may raise StopIteratio
    789     if self._pin_memory:
    790         data = _utils.pin_memory.pin_memory(data, self.
 ↪_pin_memory_device)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/_utils/fetch.py:55, in _MapDatasetFetcher.fetch(self,␣
 ↪possibly_batched_index)
     53 else:
     54     data = self.dataset[possibly_batched_index]
---> 55 return self.collate_fn(data)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/_utils/collate.py:398, in default_collate(batch)
    337 def default_collate(batch):
    338     r"""
    339     Take in a batch of data and put the elements within the batch into ␣
 ↪tensor with an additional outer dimension - batch size.
    340
   (…)    396         >>> default_collate(batch)  # Handle `CustomType`␣
 ↪automatically
    397     """
--> 398     return collate(batch, collate_fn_map=default_collate_fn_map)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/_utils/collate.py:212, in collate(batch, collate_fn_map)
    208 transposed = list(zip(*batch))  # It may be accessed twice, so we use a␣
 ↪list.
    210 if isinstance(elem, tuple):
    211     return [
--> 212         collate(samples, collate_fn_map=collate_fn_map)
    213         for samples in transposed
    214     ]  # Backwards compatibility.
    215 else:
    216     try:
```

```
File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/_utils/collate.py:240, in collate(batch, collate_fn_map)
    232         except TypeError:
    233             # The sequence type may not support `copy()` /␣
 ↪`__setitem__(index, item)`
    234             # or `__init__(iterable)` (e.g., `range`).
    235             return [
    236                 collate(samples, collate_fn_map=collate_fn_map)
    237                 for samples in transposed
    238             ]
--> 240 raise TypeError(default_collate_err_msg_format.format(elem_type))

TypeError: default_collate: batch must contain tensors, numpy arrays, numbers,␣
 ↪dicts or lists; found <class 'PIL.Image.Image'>
```

[ ]:
```
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))
```

[ ]:
```
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()
```

[ ]:
```
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()
```

[ ]:
```
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])
])
```

[ ]:
```
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)
```