cnnmodel

November 27, 2025

[119]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader, random_split
import os

np.random.seed(0)
```

[120]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
else:
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[121]:
```
class PKLDataset(Dataset):
    def __init__(self, path, transform=None):
        with open(path, "rb") as f:
            data = pickle.load(f)

        self.images = data["images"]         # shape: (N, 28, 28, 3)
        self.labels = data["labels"].reshape(-1)    # shape: (N,) instead of␣
 ↪(N,1)
        self.transform = transform

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

```python
    def __getitem__(self, idx):
        img = self.images[idx]              # numpy array (28,28,3)
        label = int(self.labels[idx])       # convert to Python int

        # Convert to tensor and permute to (C, H, W)
        img = torch.tensor(img, dtype=torch.float32).permute(2, 0, 1) / 255.0

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

        return img, label
```

[122]:

```python
import pickle

with open("ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl", "rb")␣
 ↪as f:
    data = pickle.load(f)

print(type(data))
print(len(data) if hasattr(data, "__len__") else "no len")
print(data)
```

```
<class 'dict'>
2
{'images': array([[[[ 6,  4,  0],
         [ 9,  5,  0],
         [ 8,  4,  0],
         …,
         [ 9,  6,  0],
         [ 9,  6,  0],
         [ 7,  4,  0]],

        [[11,  6,  0],
         [ 4,  4,  0],
         [ 3,  3,  0],
         …,
         [ 9,  6,  0],
         [ 6,  4,  0],
         [ 4,  2,  0]],

        [[11,  6,  0],
         [ 4,  4,  0],
         [ 3,  3,  0],
         …,
         [ 6,  4,  0],
         [ 6,  4,  0],
         [ 4,  2,  0]],
```

```
 …,

 [[ 1,  1,  0],
  [ 0,  0,  0],
  [ 0,  0,  1],
  …,
  [ 5,  4,  0],
  [ 6,  5,  0],
  [ 6,  5,  0]],

 [[ 3,  1,  1],
  [ 0,  0,  0],
  [ 0,  0,  1],
  …,
  [ 6,  5,  0],
  [ 6,  5,  0],
  [ 7,  6,  0]],

 [[10,  2,  2],
  [ 0,  0,  1],
  [ 0,  0,  1],
  …,
  [ 6,  5,  0],
  [ 7,  6,  0],
  [ 7,  6,  0]]],


[[[11,  9,  0],
  [ 9,  7,  0],
  [ 9,  7,  0],
  …,
  [ 0,  0,  1],
  [ 0,  0,  1],
  [ 0,  0,  1]],

 [[12,  9,  0],
  [11,  7,  0],
  [ 9,  6,  0],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 0,  0,  1]],

 [[ 9,  7,  0],
  [12,  8,  0],
  [10,  6,  0],
  …,
```

```
  [ 0,  0,  1],
  [ 0,  0,  1],
  [ 0,  0,  0]],

 …,

 [[ 0,  0,  3],
  [ 0,  0,  3],
  [ 0,  0,  4],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 0,  0,  0]],

 [[ 0,  0,  2],
  [ 0,  0,  2],
  [ 0,  0,  5],
  …,
  [ 0,  0,  2],
  [ 0,  0,  1],
  [ 1,  0,  0]],

 [[ 0,  0,  1],
  [ 0,  0,  2],
  [ 0,  0,  4],
  …,
  [ 0,  0,  1],
  [ 1,  0,  0],
  [ 1,  0,  0]]],


[[[18, 12,  0],
  [12,  9,  0],
  [17, 11,  0],
  …,
  [ 0,  0,  3],
  [ 5,  0,  2],
  [ 5,  0,  2]],

 [[15, 11,  0],
  [10,  9,  0],
  [10,  8,  0],
  …,
  [ 2,  0,  2],
  [ 7,  0,  1],
  [ 8,  0,  1]],

 [[17, 10,  0],
```

```
   [ 7,  6,  0],
   [ 4,  4,  0],
   …,
   [ 0,  0,  2],
   [ 5,  0,  1],
   [ 8,  0,  1]],

  …,

  [[ 2,  0,  0],
   [ 1,  0,  0],
   [ 0,  0,  0],
   …,
   [ 0,  0,  1],
   [ 0,  0,  0],
   [ 0,  0,  1]],

  [[ 2,  0,  0],
   [ 3,  1,  0],
   [ 0,  0,  0],
   …,
   [ 0,  0,  0],
   [ 0,  0,  1],
   [ 1,  0,  0]],

  [[ 3,  0,  0],
   [ 2,  0,  0],
   [ 2,  1,  0],
   …,
   [ 0,  0,  1],
   [ 1,  0,  0],
   [ 1,  0,  0]]],


 …,


 [[[56,  8, 20],
   [19,  0, 11],
   [ 0,  0,  1],
   …,
   [ 6,  3,  0],
   [11,  7,  0],
   [16,  8,  0]],

  [[19,  0, 11],
   [ 5,  0,  7],
   [ 0,  0,  3],
```

```
  …,
  [ 5,  3,  0],
  [ 5,  3,  0],
  [ 8,  4,  0]],

 [[ 0,  0,  0],
  [ 0,  0,  2],
  [ 1,  0,  6],
  …,
  [ 7,  4,  0],
  [ 5,  3,  0],
  [ 4,  2,  0]],

 …,

 [[ 4,  3,  0],
  [ 4,  2,  0],
  [ 4,  2,  0],
  …,
  [ 0,  0,  1],
  [ 1,  0,  0],
  [ 0,  0,  0]],

 [[ 1,  1,  0],
  [ 1,  1,  0],
  [ 4,  2,  0],
  …,
  [ 0,  0,  0],
  [ 3,  1,  0],
  [ 1,  1,  0]],

 [[ 1,  1,  0],
  [ 1,  1,  0],
  [ 0,  0,  0],
  …,
  [ 1,  0,  0],
  [ 3,  2,  0],
  [ 3,  3,  0]]],


[[[ 9,  6,  0],
  [ 8,  6,  0],
  [ 6,  4,  0],
  …,
  [ 3,  2,  0],
  [ 3,  2,  0],
  [ 0,  0,  0]],
```

```
 [[ 6,  6,  0],
  [ 4,  4,  0],
  [ 6,  4,  0],
  …,
  [ 3,  2,  0],
  [ 1,  0,  0],
  [ 2,  0,  0]],

 [[ 4,  4,  0],
  [ 6,  4,  0],
  [ 6,  4,  0],
  …,
  [ 1,  0,  0],
  [ 2,  0,  0],
  [ 3,  0,  0]],

 …,

 [[ 3,  3,  0],
  [ 3,  3,  0],
  [ 3,  1,  0],
  …,
  [ 0,  0,  1],
  [ 0,  0,  0],
  [ 0,  0,  0]],

 [[ 3,  3,  0],
  [ 3,  3,  0],
  [ 4,  2,  0],
  …,
  [ 2,  1,  0],
  [ 2,  1,  0],
  [ 1,  1,  0]],

 [[ 3,  3,  0],
  [ 3,  3,  0],
  [ 4,  2,  0],
  …,
  [ 2,  1,  0],
  [ 2,  1,  0],
  [ 1,  1,  0]]],


[[[ 6,  3,  0],
  [ 3,  2,  0],
  [ 2,  0,  2],
  …,
  [ 7,  6,  0],
```

```
         [ 7,  6,  0],
         [ 6,  6,  0]],

        [[ 4,  2,  0],
         [ 1,  0,  1],
         [ 2,  0,  2],
         …,
         [ 8,  5,  0],
         [ 7,  5,  0],
         [ 6,  4,  0]],

        [[ 2,  0,  0],
         [ 0,  0,  1],
         [ 0,  0,  3],
         …,
         [ 5,  3,  0],
         [ 7,  4,  0],
         [ 7,  4,  0]],

        …,

        [[ 4,  1,  0],
         [ 2,  0,  0],
         [ 1,  0,  0],
         …,
         [ 4,  2,  0],
         [ 6,  4,  0],
         [ 6,  4,  0]],

        [[ 7,  3,  0],
         [ 7,  4,  0],
         [ 6,  2,  0],
         …,
         [ 6,  4,  0],
         [ 7,  4,  0],
         [ 7,  4,  0]],

        [[11,  6,  0],
         [10,  5,  0],
         [12,  5,  0],
         …,
         [ 7,  4,  0],
         [ 7,  4,  0],
         [ 7,  5,  0]]]], shape=(1080, 28, 28, 3), dtype=uint8), 'labels':
array([[0],
       [0],
       [0],
       …,
```

```
        [2],
        [2],
        [3]], shape=(1080, 1), dtype=uint8)}
```

[123]:

```python
from sklearn.model_selection import train_test_split
from torchvision import transforms
from torch.utils.data import Subset



# Charger le dataset SANS transformation d'abord
dataset = PKLDataset(
                "ift-3395-6390-kaggle-2-competition-fall-2025/train_data.pkl",
)

loader = DataLoader(dataset, batch_size=64, shuffle=False)

d_mean = torch.zeros(3)
d_std = torch.zeros(3)
nb_samples = 0.0

for images, _ in loader:
    batch_samples = images.size(0)

    d_mean += images.mean(dim=[0,2,3]) * batch_samples
    d_std += images.std(dim=[0,2,3]) * batch_samples
    nb_samples += batch_samples

d_mean /= nb_samples
d_std /= nb_samples

d_mean = d_mean.tolist()
d_std = d_std.tolist()

print("Mean:", d_mean)
print("Std:", d_std)
```

```
Mean: [0.21014535427093506, 0.005330359563231468, 0.2285669893026352]
Std: [0.18871904909610748, 0.01642582379281521, 0.16962255537509918]
```

[124]:

```python
# Définir les transformations
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5), # La rétine n'a pas de sens gauche/
 ↪droite
    transforms.RandomVerticalFlip(p=0.5),   # Ni de haut/bas strict
    transforms.RandomRotation(180),
    transforms.RandomAdjustSharpness(sharpness_factor=2, p=1.0),
```

```
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
↪1),        # La rotation est cruciale pour l'œil
    transforms.Normalize(mean=d_mean, std=d_std),
])

transform_val = transforms.Compose([
    transforms.Normalize(mean=d_mean, std=d_std),
])
```

[125]:
```
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)
```

[126]:
```
from torch.utils.data import WeightedRandomSampler
import torch
import numpy as np

# 1. Votre Split existant (inchangé)
labels = dataset.labels
indices = np.arange(len(dataset))
train_idx, valid_idx = train_test_split(
    indices,
    test_size=0.2,
    random_state=42,
    stratify=labels
)

# 2. Préparation du Sampler (NOUVEAU BLOC)
# On récupère uniquement les labels qui sont dans le set d'entrainement
y_train = labels[train_idx].reshape(-1) # reshape pour être sûr d'avoir (N,) et␣
↪pas (N,1)

# Compter combien d'exemples il y a par classe dans le train
class_counts = np.bincount(y_train)

# Calculer le poids de chaque classe (Inverse Frequency)
# Moins la classe est fréquente, plus le poids est grand
```

```
class_weights = 1. / class_counts

# Assigner un poids à chaque ÉCHANTILLON individuel du train
# Si l'image 1 est de classe 0, elle prend le poids de la classe 0, etc.
samples_weights = class_weights[y_train]
samples_weights = torch.from_numpy(samples_weights).double()

# Créer le sampler
sampler = WeightedRandomSampler(
    weights=samples_weights,
    num_samples=len(samples_weights),
    replacement=True # CRUCIAL : permet de re-piocher les images rares␣
 ↪plusieurs fois par epoch
)

# 3. Création des Subsets et Transforms (inchangé)
train_data = Subset(dataset, train_idx)
valid_data = Subset(dataset, valid_idx)

train_data = TransformSubset(train_data, transform=transform_train)
valid_data = TransformSubset(valid_data, transform=transform_val)

# 4. DataLoaders (MODIFIÉ)
train_loader = DataLoader(
    train_data,
    batch_size=64,
    #sampler=sampler,  # <--- On ajoute le sampler ici
    shuffle=True    # <--- OBLIGATOIRE : shuffle doit être False quand on␣
 ↪utilise un sampler
)

# Le valid_loader reste classique (on ne veut pas de sampler pour la validation)
valid_loader = DataLoader(valid_data, batch_size=128, shuffle=True)
```

[127]:

```
import numpy as np

labels = dataset.labels
classes, counts = np.unique(labels, return_counts=True)

from sklearn.utils.class_weight import compute_class_weight

weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(labels),
    y=labels
)
class_weights = torch.tensor(weights, dtype=torch.float32).to(device)
```

```python
loss_fn = nn.CrossEntropyLoss(weight=class_weights)

test_loss_fn = nn.CrossEntropyLoss(weight=class_weights, reduction='sum')

# spot to save your learning curves, and potentially checkpoint your models
savedir = 'results'
if not os.path.exists(savedir):
    os.makedirs(savedir)
```

[128]:
```python
def train(model, train_loader, optimizer, epoch):
    """Perform one epoch of training."""
    model.train()

    for batch_idx, (inputs, target) in enumerate(train_loader):
        inputs, target = inputs.to(device), target.to(device)

        # 1) Reset gradients
        optimizer.zero_grad()

        # 2) Forward pass
        output = model(inputs)

        # 3) Compute loss
        loss = loss_fn(output, target)

        # 4) Backpropagation
        loss.backward()

        # 5) Update weights
        optimizer.step()

        # Logging
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch,
                batch_idx * len(inputs),
                len(train_loader.dataset),
                100. * batch_idx / len(train_loader),
                loss.item()
            ))
```

[129]:
```python
from sklearn.metrics import recall_score

def test(model, test_loader):
    """Evaluate the model by doing one pass over a dataset"""
    model.eval()
```

```python
    test_loss = 0   # total loss over test set
    correct = 0     # total number of correct test predictions
    test_size = 0   # number of test samples used
    all_preds = []  # to store all predictions
    all_targets = []# to store all targets

    with torch.no_grad():  # no backprop, faster evaluation
        for inputs, target in test_loader:
            inputs, target = inputs.to(device), target.to(device)

            # Forward pass
            output = model(inputs)

            # Accumulate loss (sum, not mean)
            loss = test_loss_fn(output, target)  # already reduction='sum'
            test_loss += loss.item()

            # Predictions
            pred = output.argmax(dim=1)  # index of highest logit

            all_preds.extend(pred.tolist())
            all_targets.extend(target.tolist()) # Target est déjà long pour␣
↪CrossEntropy

            correct += (pred == target).sum().item()

            # Keep track of sample count
            test_size += target.size(0)

    # Final metrics
    test_loss /= test_size
    accuracy = correct / test_size

    macro_recall = recall_score(
        all_targets,
        all_preds,
        average='macro',
        zero_division=0
    )

    print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, test_size, 100. * accuracy))

    return test_loss, accuracy, macro_recall, all_preds, all_targets
```

[130]:
```
class CNNNet(nn.Module):
    def __init__(self, num_classes=5):
        super().__init__()
        self.act = nn.ReLU()
        self.drop = nn.Dropout(0.5) # Dropout fort pour éviter le par cœur vu␣
↪qu'on augmente les filtres

        # Bloc 1 : Extraction de features bas niveau (bords, contrastes)
        # On passe de 3 à 64 filtres directement pour capter plus de nuances de␣
↪couleurs
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        # Bloc 2 : Extraction mi-niveau
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(128)

        # Bloc 3 : Features complexes
        self.conv5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn5 = nn.BatchNorm2d(256)

        # Classifieur
        # Après 3 max_pool (divisé par 2 trois fois): 28 -> 14 -> 7 -> 3
        self.flatten_dim = 256 * 3 * 3

        self.fc1 = nn.Linear(self.flatten_dim, 512)
        self.fc2 = nn.Linear(512, 5)

    def forward(self, x):
        # Bloc 1 (28x28)
        x = self.act(self.bn1(self.conv1(x)))
        x = self.act(self.bn2(self.conv2(x)))
        x = F.max_pool2d(x, 2) # -> 14x14

        # Bloc 2 (14x14)
        x = self.act(self.bn3(self.conv3(x)))
        x = self.act(self.bn4(self.conv4(x)))
        x = F.max_pool2d(x, 2) # -> 7x7

        # Bloc 3 (7x7)
        x = self.act(self.bn5(self.conv5(x)))
        x = F.max_pool2d(x, 2) # -> 3x3
```

```
        x = x.view(x.size(0), -1)
        x = self.drop(self.act(self.fc1(x)))
        x = self.fc2(x)
        return x
```

[131]:
```
subset = Subset(train_data, list(range(50)))
subset_loader = DataLoader(subset, batch_size=10, shuffle=False)
model = CNNNet().to(device)
lr = 0.0001
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-4)

for epoch in range(50):
    train(model, subset_loader, optimizer, epoch)
    loss, acc, _, _, _ = test(model, subset_loader)
    print(loss, acc)
```

```
Train Epoch: 0 [0/50 (0%)]      Loss: 1.604321
Test set: Average loss: 1.4815, Accuracy: 5/50 (10%)

1.481468963623047 0.1
Train Epoch: 1 [0/50 (0%)]      Loss: 1.634385
Test set: Average loss: 1.4471, Accuracy: 9/50 (18%)

1.4470633888244628 0.18
Train Epoch: 2 [0/50 (0%)]      Loss: 2.215499
Test set: Average loss: 1.4076, Accuracy: 9/50 (18%)

1.407620906829834 0.18
Train Epoch: 3 [0/50 (0%)]      Loss: 1.764416
Test set: Average loss: 1.3706, Accuracy: 11/50 (22%)

1.3706335067749023 0.22
Train Epoch: 4 [0/50 (0%)]      Loss: 1.668756
Test set: Average loss: 1.3924, Accuracy: 13/50 (26%)

1.3923808860778808 0.26
Train Epoch: 5 [0/50 (0%)]      Loss: 1.882579
Test set: Average loss: 1.3159, Accuracy: 17/50 (34%)

1.3158868980407714 0.34
Train Epoch: 6 [0/50 (0%)]      Loss: 1.499725
Test set: Average loss: 1.2686, Accuracy: 16/50 (32%)

1.268580150604248 0.32
Train Epoch: 7 [0/50 (0%)]      Loss: 1.480666
Test set: Average loss: 1.3218, Accuracy: 17/50 (34%)

1.3217594528198242 0.34
```

```
Train Epoch: 8 [0/50 (0%)]      Loss: 1.458279
Test set: Average loss: 1.2498, Accuracy: 18/50 (36%)

1.2497577095031738 0.36
Train Epoch: 9 [0/50 (0%)]      Loss: 1.316792
Test set: Average loss: 1.1872, Accuracy: 18/50 (36%)

1.1871959495544433 0.36
Train Epoch: 10 [0/50 (0%)]     Loss: 1.379931
Test set: Average loss: 1.3753, Accuracy: 13/50 (26%)

1.3752995491027833 0.26
Train Epoch: 11 [0/50 (0%)]     Loss: 1.441182
Test set: Average loss: 1.2543, Accuracy: 17/50 (34%)

1.2543045997619628 0.34
Train Epoch: 12 [0/50 (0%)]     Loss: 1.836610
Test set: Average loss: 1.1868, Accuracy: 23/50 (46%)

1.18676025390625 0.46
Train Epoch: 13 [0/50 (0%)]     Loss: 1.419449
Test set: Average loss: 1.3227, Accuracy: 17/50 (34%)

1.3227205276489258 0.34
Train Epoch: 14 [0/50 (0%)]     Loss: 1.402537
Test set: Average loss: 1.2981, Accuracy: 18/50 (36%)

1.2980594062805175 0.36
Train Epoch: 15 [0/50 (0%)]     Loss: 1.244537
Test set: Average loss: 1.2603, Accuracy: 20/50 (40%)

1.26028507232666 0.4
Train Epoch: 16 [0/50 (0%)]     Loss: 1.727993
Test set: Average loss: 1.2178, Accuracy: 19/50 (38%)

1.217839584350586 0.38
Train Epoch: 17 [0/50 (0%)]     Loss: 1.306903
Test set: Average loss: 1.1722, Accuracy: 19/50 (38%)

1.1721979522705077 0.38
Train Epoch: 18 [0/50 (0%)]     Loss: 1.784945
Test set: Average loss: 1.1551, Accuracy: 23/50 (46%)

1.1551092147827149 0.46
Train Epoch: 19 [0/50 (0%)]     Loss: 1.301293
Test set: Average loss: 1.2866, Accuracy: 15/50 (30%)

1.2866455078125 0.3
```

```
Train Epoch: 20 [0/50 (0%)]     Loss: 1.095710
Test set: Average loss: 1.2712, Accuracy: 18/50 (36%)

1.2712039756774902 0.36
Train Epoch: 21 [0/50 (0%)]     Loss: 1.797358
Test set: Average loss: 1.2129, Accuracy: 20/50 (40%)

1.212853832244873 0.4
Train Epoch: 22 [0/50 (0%)]     Loss: 1.563704
Test set: Average loss: 1.0915, Accuracy: 21/50 (42%)

1.0914714622497559 0.42
Train Epoch: 23 [0/50 (0%)]     Loss: 1.785437
Test set: Average loss: 1.3241, Accuracy: 17/50 (34%)

1.3240657424926758 0.34
Train Epoch: 24 [0/50 (0%)]     Loss: 1.165241
Test set: Average loss: 1.2238, Accuracy: 23/50 (46%)

1.2237592506408692 0.46
Train Epoch: 25 [0/50 (0%)]     Loss: 1.237355
Test set: Average loss: 1.2337, Accuracy: 21/50 (42%)

1.233691177368164 0.42
Train Epoch: 26 [0/50 (0%)]     Loss: 1.528797
Test set: Average loss: 1.0866, Accuracy: 19/50 (38%)

1.0866372680664063 0.38
Train Epoch: 27 [0/50 (0%)]     Loss: 1.492079
Test set: Average loss: 1.1735, Accuracy: 23/50 (46%)

1.173460283279419 0.46
Train Epoch: 28 [0/50 (0%)]     Loss: 1.451905
Test set: Average loss: 1.2714, Accuracy: 21/50 (42%)

1.2713822555541991 0.42
Train Epoch: 29 [0/50 (0%)]     Loss: 1.497938
Test set: Average loss: 1.1540, Accuracy: 26/50 (52%)

1.1540084075927735 0.52
Train Epoch: 30 [0/50 (0%)]     Loss: 1.743461
Test set: Average loss: 1.2104, Accuracy: 21/50 (42%)

1.2104018974304198 0.42
Train Epoch: 31 [0/50 (0%)]     Loss: 1.145347
Test set: Average loss: 1.2049, Accuracy: 19/50 (38%)

1.204909267425537 0.38
```

```
Train Epoch: 32 [0/50 (0%)]     Loss: 1.611389
Test set: Average loss: 1.2252, Accuracy: 20/50 (40%)

1.2251705741882324 0.4
Train Epoch: 33 [0/50 (0%)]     Loss: 1.205485
Test set: Average loss: 1.1379, Accuracy: 24/50 (48%)

1.137857666015625 0.48
Train Epoch: 34 [0/50 (0%)]     Loss: 1.575156
Test set: Average loss: 1.1175, Accuracy: 19/50 (38%)

1.117455654144287 0.38
Train Epoch: 35 [0/50 (0%)]     Loss: 0.950851
Test set: Average loss: 1.1537, Accuracy: 21/50 (42%)

1.1536978912353515 0.42
Train Epoch: 36 [0/50 (0%)]     Loss: 1.207170
Test set: Average loss: 1.1338, Accuracy: 24/50 (48%)

1.133829345703125 0.48
Train Epoch: 37 [0/50 (0%)]     Loss: 1.552323
Test set: Average loss: 1.0994, Accuracy: 18/50 (36%)

1.099412956237793 0.36
Train Epoch: 38 [0/50 (0%)]     Loss: 1.122940
Test set: Average loss: 1.1089, Accuracy: 23/50 (46%)

1.108892288208008 0.46
Train Epoch: 39 [0/50 (0%)]     Loss: 1.210391
Test set: Average loss: 1.1253, Accuracy: 19/50 (38%)

1.125344352722168 0.38
Train Epoch: 40 [0/50 (0%)]     Loss: 1.112084
Test set: Average loss: 1.1319, Accuracy: 21/50 (42%)

1.1318658638000487 0.42
Train Epoch: 41 [0/50 (0%)]     Loss: 1.103702
Test set: Average loss: 1.1589, Accuracy: 20/50 (40%)

1.158891487121582 0.4
Train Epoch: 42 [0/50 (0%)]     Loss: 1.115503
Test set: Average loss: 0.9937, Accuracy: 27/50 (54%)

0.9936512279510498 0.54
Train Epoch: 43 [0/50 (0%)]     Loss: 1.238985
Test set: Average loss: 1.1092, Accuracy: 24/50 (48%)

1.109208526611328 0.48
```

```
Train Epoch: 44 [0/50 (0%)]     Loss: 0.929227
Test set: Average loss: 1.0675, Accuracy: 22/50 (44%)

1.0674526977539063 0.44
Train Epoch: 45 [0/50 (0%)]     Loss: 1.145489
Test set: Average loss: 1.0961, Accuracy: 21/50 (42%)

1.0961102771759033 0.42
Train Epoch: 46 [0/50 (0%)]     Loss: 1.241889
Test set: Average loss: 1.1395, Accuracy: 23/50 (46%)

1.1395499420166015 0.46
Train Epoch: 47 [0/50 (0%)]     Loss: 0.800473
Test set: Average loss: 1.0567, Accuracy: 24/50 (48%)

1.0567286682128907 0.48
Train Epoch: 48 [0/50 (0%)]     Loss: 1.082200
Test set: Average loss: 1.1191, Accuracy: 25/50 (50%)

1.1190689849853515 0.5
Train Epoch: 49 [0/50 (0%)]     Loss: 1.031219
Test set: Average loss: 1.0538, Accuracy: 19/50 (38%)

1.0538458633422851 0.38
```

[132]:

```
# TRAINING
model = CNNNet().to(device)
lr=0.001
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=0.5, patience=4
)

results = {'name':'model', 'lr': lr, 'train_loss': [],
    'train_acc': [],
    'val_loss': [],
    'val_acc': [],
    'val_macro_recall': [],
    'final_val_preds': [],
    'final_val_targets': []
    }
savefile = os.path.join(savedir, results['name']+str(results['lr'])+'.pkl' )

for epoch in range(1, 50):
    train(model, train_loader, optimizer, epoch)

    train_loss, train_acc, _, _, _= test(model, train_loader)
```

```
    val_loss, val_acc, val_macro_recall, final_val_preds, final_val_targets  =␣
↪test(model, valid_loader)

    scheduler.step(val_acc)

    results['train_loss'].append(train_loss)
    results['train_acc'].append(train_acc)

    results['val_loss'].append(val_loss)
    results['val_acc'].append(val_acc)

    results['val_macro_recall'].append(val_macro_recall)

    if epoch == 49:
        results['final_val_preds'] = final_val_preds
        results['final_val_targets'] = final_val_targets

    with open(savefile, 'wb') as fout:
        pickle.dump(results, fout)
```

```
Train Epoch: 1 [0/864 (0%)]     Loss: 1.688974
Train Epoch: 1 [640/864 (71%)]  Loss: 1.619674
Test set: Average loss: 2.0080, Accuracy: 295/864 (34%)

Test set: Average loss: 1.6633, Accuracy: 67/216 (31%)

Train Epoch: 2 [0/864 (0%)]     Loss: 1.916270
Train Epoch: 2 [640/864 (71%)]  Loss: 1.847124
Test set: Average loss: 1.6737, Accuracy: 307/864 (36%)

Test set: Average loss: 1.5644, Accuracy: 71/216 (33%)

Train Epoch: 3 [0/864 (0%)]     Loss: 1.681325
Train Epoch: 3 [640/864 (71%)]  Loss: 1.529294
Test set: Average loss: 1.5454, Accuracy: 282/864 (33%)

Test set: Average loss: 1.5484, Accuracy: 58/216 (27%)

Train Epoch: 4 [0/864 (0%)]     Loss: 1.539677
Train Epoch: 4 [640/864 (71%)]  Loss: 1.670403
Test set: Average loss: 1.5616, Accuracy: 305/864 (35%)

Test set: Average loss: 1.5403, Accuracy: 84/216 (39%)

Train Epoch: 5 [0/864 (0%)]     Loss: 1.567348
Train Epoch: 5 [640/864 (71%)]  Loss: 1.715055
Test set: Average loss: 1.5503, Accuracy: 296/864 (34%)
```

```
Test set: Average loss: 1.5288, Accuracy: 68/216 (31%)

Train Epoch: 6 [0/864 (0%)]     Loss: 1.513247
Train Epoch: 6 [640/864 (71%)]  Loss: 1.577354
Test set: Average loss: 1.5632, Accuracy: 244/864 (28%)

Test set: Average loss: 1.5513, Accuracy: 48/216 (22%)

Train Epoch: 7 [0/864 (0%)]     Loss: 1.655100
Train Epoch: 7 [640/864 (71%)]  Loss: 1.608462
Test set: Average loss: 1.5580, Accuracy: 280/864 (32%)

Test set: Average loss: 1.5500, Accuracy: 58/216 (27%)

Train Epoch: 8 [0/864 (0%)]     Loss: 1.563547
Train Epoch: 8 [640/864 (71%)]  Loss: 1.555359
Test set: Average loss: 1.5407, Accuracy: 270/864 (31%)

Test set: Average loss: 1.5504, Accuracy: 45/216 (21%)

Train Epoch: 9 [0/864 (0%)]     Loss: 1.539650
Train Epoch: 9 [640/864 (71%)]  Loss: 1.526688
Test set: Average loss: 1.5553, Accuracy: 264/864 (31%)

Test set: Average loss: 1.5437, Accuracy: 52/216 (24%)

Train Epoch: 10 [0/864 (0%)]    Loss: 1.503703
Train Epoch: 10 [640/864 (71%)] Loss: 1.485277
Test set: Average loss: 1.5179, Accuracy: 266/864 (31%)

Test set: Average loss: 1.5468, Accuracy: 49/216 (23%)

Train Epoch: 11 [0/864 (0%)]    Loss: 1.504034
Train Epoch: 11 [640/864 (71%)] Loss: 1.531012
Test set: Average loss: 1.5362, Accuracy: 287/864 (33%)

Test set: Average loss: 1.5434, Accuracy: 57/216 (26%)

Train Epoch: 12 [0/864 (0%)]    Loss: 1.523920
Train Epoch: 12 [640/864 (71%)] Loss: 1.558376
Test set: Average loss: 1.5287, Accuracy: 276/864 (32%)

Test set: Average loss: 1.5438, Accuracy: 57/216 (26%)

Train Epoch: 13 [0/864 (0%)]    Loss: 1.518058
Train Epoch: 13 [640/864 (71%)] Loss: 1.527068
Test set: Average loss: 1.5215, Accuracy: 269/864 (31%)
```

```
Test set: Average loss: 1.5398, Accuracy: 53/216 (25%)

Train Epoch: 14 [0/864 (0%)]    Loss: 1.579265
Train Epoch: 14 [640/864 (71%)] Loss: 1.519454
Test set: Average loss: 1.5356, Accuracy: 295/864 (34%)

Test set: Average loss: 1.5294, Accuracy: 70/216 (32%)

Train Epoch: 15 [0/864 (0%)]    Loss: 1.533659
Train Epoch: 15 [640/864 (71%)] Loss: 1.499688
Test set: Average loss: 1.5222, Accuracy: 298/864 (34%)

Test set: Average loss: 1.5217, Accuracy: 66/216 (31%)

Train Epoch: 16 [0/864 (0%)]    Loss: 1.575559
Train Epoch: 16 [640/864 (71%)] Loss: 1.517337
Test set: Average loss: 1.5299, Accuracy: 267/864 (31%)

Test set: Average loss: 1.5314, Accuracy: 50/216 (23%)

Train Epoch: 17 [0/864 (0%)]    Loss: 1.549041
Train Epoch: 17 [640/864 (71%)] Loss: 1.492191
Test set: Average loss: 1.5211, Accuracy: 274/864 (32%)

Test set: Average loss: 1.5275, Accuracy: 53/216 (25%)

Train Epoch: 18 [0/864 (0%)]    Loss: 1.566365
Train Epoch: 18 [640/864 (71%)] Loss: 1.533297
Test set: Average loss: 1.5071, Accuracy: 308/864 (36%)

Test set: Average loss: 1.5191, Accuracy: 59/216 (27%)

Train Epoch: 19 [0/864 (0%)]    Loss: 1.538778
Train Epoch: 19 [640/864 (71%)] Loss: 1.559799
Test set: Average loss: 1.5144, Accuracy: 262/864 (30%)

Test set: Average loss: 1.5248, Accuracy: 57/216 (26%)

Train Epoch: 20 [0/864 (0%)]    Loss: 1.512001
Train Epoch: 20 [640/864 (71%)] Loss: 1.450939
Test set: Average loss: 1.5103, Accuracy: 271/864 (31%)

Test set: Average loss: 1.5289, Accuracy: 55/216 (25%)

Train Epoch: 21 [0/864 (0%)]    Loss: 1.493762
Train Epoch: 21 [640/864 (71%)] Loss: 1.508839
Test set: Average loss: 1.5079, Accuracy: 279/864 (32%)
```

```
Test set: Average loss: 1.5251, Accuracy: 55/216 (25%)

Train Epoch: 22 [0/864 (0%)]    Loss: 1.482474
Train Epoch: 22 [640/864 (71%)] Loss: 1.589910
Test set: Average loss: 1.5021, Accuracy: 300/864 (35%)

Test set: Average loss: 1.5197, Accuracy: 59/216 (27%)

Train Epoch: 23 [0/864 (0%)]    Loss: 1.539251
Train Epoch: 23 [640/864 (71%)] Loss: 1.590655
Test set: Average loss: 1.5051, Accuracy: 272/864 (31%)

Test set: Average loss: 1.5225, Accuracy: 55/216 (25%)

Train Epoch: 24 [0/864 (0%)]    Loss: 1.619329
Train Epoch: 24 [640/864 (71%)] Loss: 1.477285
Test set: Average loss: 1.5164, Accuracy: 284/864 (33%)

Test set: Average loss: 1.5264, Accuracy: 55/216 (25%)

Train Epoch: 25 [0/864 (0%)]    Loss: 1.557336
Train Epoch: 25 [640/864 (71%)] Loss: 1.480700
Test set: Average loss: 1.4949, Accuracy: 302/864 (35%)

Test set: Average loss: 1.5256, Accuracy: 56/216 (26%)

Train Epoch: 26 [0/864 (0%)]    Loss: 1.515304
Train Epoch: 26 [640/864 (71%)] Loss: 1.442173
Test set: Average loss: 1.5025, Accuracy: 290/864 (34%)

Test set: Average loss: 1.5252, Accuracy: 57/216 (26%)

Train Epoch: 27 [0/864 (0%)]    Loss: 1.472365
Train Epoch: 27 [640/864 (71%)] Loss: 1.440349
Test set: Average loss: 1.5031, Accuracy: 296/864 (34%)

Test set: Average loss: 1.5224, Accuracy: 56/216 (26%)

Train Epoch: 28 [0/864 (0%)]    Loss: 1.553426
Train Epoch: 28 [640/864 (71%)] Loss: 1.499764
Test set: Average loss: 1.4986, Accuracy: 312/864 (36%)

Test set: Average loss: 1.5217, Accuracy: 61/216 (28%)

Train Epoch: 29 [0/864 (0%)]    Loss: 1.462000
Train Epoch: 29 [640/864 (71%)] Loss: 1.552776
Test set: Average loss: 1.4942, Accuracy: 310/864 (36%)
```

```
Test set: Average loss: 1.5165, Accuracy: 58/216 (27%)

Train Epoch: 30 [0/864 (0%)]    Loss: 1.561329
Train Epoch: 30 [640/864 (71%)] Loss: 1.580283
Test set: Average loss: 1.5011, Accuracy: 296/864 (34%)

Test set: Average loss: 1.5174, Accuracy: 60/216 (28%)

Train Epoch: 31 [0/864 (0%)]    Loss: 1.440609
Train Epoch: 31 [640/864 (71%)] Loss: 1.521872
Test set: Average loss: 1.4741, Accuracy: 313/864 (36%)

Test set: Average loss: 1.5175, Accuracy: 60/216 (28%)

Train Epoch: 32 [0/864 (0%)]    Loss: 1.480009
Train Epoch: 32 [640/864 (71%)] Loss: 1.536060
Test set: Average loss: 1.4977, Accuracy: 317/864 (37%)

Test set: Average loss: 1.5156, Accuracy: 65/216 (30%)

Train Epoch: 33 [0/864 (0%)]    Loss: 1.502618
Train Epoch: 33 [640/864 (71%)] Loss: 1.503730
Test set: Average loss: 1.4964, Accuracy: 303/864 (35%)

Test set: Average loss: 1.5151, Accuracy: 62/216 (29%)

Train Epoch: 34 [0/864 (0%)]    Loss: 1.542801
Train Epoch: 34 [640/864 (71%)] Loss: 1.445410
Test set: Average loss: 1.4922, Accuracy: 314/864 (36%)

Test set: Average loss: 1.5162, Accuracy: 61/216 (28%)

Train Epoch: 35 [0/864 (0%)]    Loss: 1.511835
Train Epoch: 35 [640/864 (71%)] Loss: 1.516071
Test set: Average loss: 1.4975, Accuracy: 302/864 (35%)

Test set: Average loss: 1.5181, Accuracy: 60/216 (28%)

Train Epoch: 36 [0/864 (0%)]    Loss: 1.526250
Train Epoch: 36 [640/864 (71%)] Loss: 1.501496
Test set: Average loss: 1.4847, Accuracy: 313/864 (36%)

Test set: Average loss: 1.5191, Accuracy: 63/216 (29%)

Train Epoch: 37 [0/864 (0%)]    Loss: 1.474992
Train Epoch: 37 [640/864 (71%)] Loss: 1.492573
Test set: Average loss: 1.4916, Accuracy: 317/864 (37%)
```

```
Test set: Average loss: 1.5177, Accuracy: 64/216 (30%)

Train Epoch: 38 [0/864 (0%)]    Loss: 1.497055
Train Epoch: 38 [640/864 (71%)] Loss: 1.492867
Test set: Average loss: 1.4851, Accuracy: 317/864 (37%)

Test set: Average loss: 1.5160, Accuracy: 61/216 (28%)

Train Epoch: 39 [0/864 (0%)]    Loss: 1.611247
Train Epoch: 39 [640/864 (71%)] Loss: 1.445208
Test set: Average loss: 1.5114, Accuracy: 284/864 (33%)

Test set: Average loss: 1.5152, Accuracy: 60/216 (28%)

Train Epoch: 40 [0/864 (0%)]    Loss: 1.559368
Train Epoch: 40 [640/864 (71%)] Loss: 1.466190
Test set: Average loss: 1.4964, Accuracy: 303/864 (35%)

Test set: Average loss: 1.5156, Accuracy: 59/216 (27%)

Train Epoch: 41 [0/864 (0%)]    Loss: 1.450624
Train Epoch: 41 [640/864 (71%)] Loss: 1.515004
Test set: Average loss: 1.4907, Accuracy: 295/864 (34%)

Test set: Average loss: 1.5154, Accuracy: 58/216 (27%)

Train Epoch: 42 [0/864 (0%)]    Loss: 1.461328
Train Epoch: 42 [640/864 (71%)] Loss: 1.478557
Test set: Average loss: 1.4897, Accuracy: 302/864 (35%)

Test set: Average loss: 1.5142, Accuracy: 59/216 (27%)

Train Epoch: 43 [0/864 (0%)]    Loss: 1.449653
Train Epoch: 43 [640/864 (71%)] Loss: 1.506764
Test set: Average loss: 1.4851, Accuracy: 315/864 (36%)

Test set: Average loss: 1.5129, Accuracy: 60/216 (28%)

Train Epoch: 44 [0/864 (0%)]    Loss: 1.464786
Train Epoch: 44 [640/864 (71%)] Loss: 1.438298
Test set: Average loss: 1.4841, Accuracy: 316/864 (37%)

Test set: Average loss: 1.5130, Accuracy: 58/216 (27%)

Train Epoch: 45 [0/864 (0%)]    Loss: 1.485224
Train Epoch: 45 [640/864 (71%)] Loss: 1.501235
Test set: Average loss: 1.4928, Accuracy: 309/864 (36%)
```

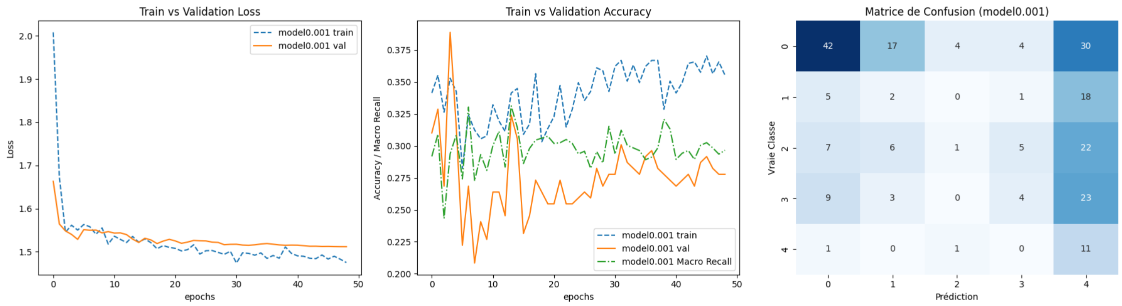
```
Test set: Average loss: 1.5124, Accuracy: 62/216 (29%)

Train Epoch: 46 [0/864 (0%)]    Loss: 1.542726
Train Epoch: 46 [640/864 (71%)] Loss: 1.487101
Test set: Average loss: 1.4833, Accuracy: 320/864 (37%)

Test set: Average loss: 1.5125, Accuracy: 63/216 (29%)

Train Epoch: 47 [0/864 (0%)]    Loss: 1.450308
Train Epoch: 47 [640/864 (71%)] Loss: 1.496252
Test set: Average loss: 1.4898, Accuracy: 308/864 (36%)

Test set: Average loss: 1.5122, Accuracy: 61/216 (28%)

Train Epoch: 48 [0/864 (0%)]    Loss: 1.505541
Train Epoch: 48 [640/864 (71%)] Loss: 1.526444
Test set: Average loss: 1.4829, Accuracy: 316/864 (37%)

Test set: Average loss: 1.5120, Accuracy: 60/216 (28%)

Train Epoch: 49 [0/864 (0%)]    Loss: 1.502518
Train Epoch: 49 [640/864 (71%)] Loss: 1.481721
Test set: Average loss: 1.4751, Accuracy: 307/864 (36%)

Test set: Average loss: 1.5119, Accuracy: 60/216 (28%)
```

[133]:

```python
import seaborn as sns
from sklearn.metrics import confusion_matrix, recall_score

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

# Liste pour stocker les résultats du dernier modèle chargé
last_results = None
last_label = None

for filename in os.listdir(savedir):
    if filename.endswith('.pkl'):
        with open(os.path.join(savedir, filename), 'rb') as fin:
            results = pickle.load(fin)
            label = filename[:-4]  # nom sans .pkl
            last_results = results
            last_label = label

            # --- Courbes de LOSS (ax1) ---
            ax1.plot(results['train_loss'], '--', label=f'{label} train')
            ax1.plot(results['val_loss'], '-',  label=f'{label} val')
```

```
            ax1.set_ylabel('Loss')
            ax1.set_xlabel('epochs')
            ax1.set_title('Train vs Validation Loss')

            ax2.plot(results['train_acc'], '--', label=f'{label} train')
            ax2.plot(results['val_acc'], '-',  label=f'{label} val')

            if 'val_macro_recall' in results:
                 ax2.plot(results['val_macro_recall'], '-.', label=f'{label}␣
 ↪Macro Recall')

            ax2.set_ylabel('Accuracy / Macro Recall')
            ax2.set_xlabel('epochs')
            ax2.set_title('Train vs Validation Accuracy')

print("Last results keys:", last_results.keys() if last_results else "No␣
 ↪results loaded")

if last_results and 'final_val_preds' in last_results:
    y_true = np.array(last_results['final_val_targets'])
    y_pred = np.array(last_results['final_val_preds'])

    cm = confusion_matrix(y_true, y_pred)

    # Affichage de la Heatmap
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax3, cbar=False)
    ax3.set_title(f'Matrice de Confusion ({last_label})')
    ax3.set_xlabel('Prédiction')
    ax3.set_ylabel('Vraie Classe')

# Légende pour ax1 et ax2
ax1.legend()
ax2.legend()

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

```
Last results keys: dict_keys(['name', 'lr', 'train_loss', 'train_acc',
'val_loss', 'val_acc', 'val_macro_recall', 'final_val_preds',
'final_val_targets'])
```


Train vs Validation Loss
model0.001 train
model0.001 val
Loss
epochs
Train vs Validation Accuracy
model0.001 train
model0.001 val
model0.001 Macro Recall
Accuracy / Macro Recall
epochs
Matrice de Confusion (model0.001)
Vraie Classe
Prédiction