# debugcm

November 27, 2025

[14]:
```
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)
```

[15]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
else:
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[16]:
```
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)
```

```
    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
```

[17]:
```
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)}
```

[18]:
```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]
```

[19]:
```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),         # 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),
])
```

[20]:
```
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)
```

[21]:
```
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=False     # <--- 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=False)
```

[22]:
```
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)

loss_fn = nn.CrossEntropyLoss()
```

```
test_loss_fn = nn.CrossEntropyLoss(reduction='sum')

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

[23]:
```
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()
            ))
```

[24]:
```
from sklearn.metrics import recall_score

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

    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
```

[25]:
```
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
```

[26]:
```
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: 1.688330
Test set: Average loss: 1.5754, Accuracy: 23/50 (46%)

1.5753730201721192 0.46
Train Epoch: 1 [0/50 (0%)]      Loss: 1.387872
Test set: Average loss: 1.5164, Accuracy: 23/50 (46%)

1.5163601684570311 0.46
Train Epoch: 2 [0/50 (0%)]      Loss: 1.212225
Test set: Average loss: 1.4519, Accuracy: 25/50 (50%)

1.4518618392944336 0.5
Train Epoch: 3 [0/50 (0%)]      Loss: 1.252805
Test set: Average loss: 1.3523, Accuracy: 26/50 (52%)

1.3523392486572265 0.52
Train Epoch: 4 [0/50 (0%)]      Loss: 1.185224
Test set: Average loss: 1.2343, Accuracy: 27/50 (54%)

1.23432710647583 0.54
Train Epoch: 5 [0/50 (0%)]      Loss: 1.142199
Test set: Average loss: 1.1372, Accuracy: 23/50 (46%)

1.1372257614135741 0.46
Train Epoch: 6 [0/50 (0%)]      Loss: 1.492760
Test set: Average loss: 1.0926, Accuracy: 24/50 (48%)

1.092618751525879 0.48
Train Epoch: 7 [0/50 (0%)]      Loss: 1.159684
Test set: Average loss: 1.0409, Accuracy: 28/50 (56%)

1.0409052085876465 0.56
Train Epoch: 8 [0/50 (0%)]      Loss: 1.249370
Test set: Average loss: 0.9933, Accuracy: 28/50 (56%)
```

```
0.9933088207244873 0.56
Train Epoch: 9 [0/50 (0%)]      Loss: 1.146795
Test set: Average loss: 0.9777, Accuracy: 30/50 (60%)

0.9776682662963867 0.6
Train Epoch: 10 [0/50 (0%)]     Loss: 1.059865
Test set: Average loss: 0.8929, Accuracy: 33/50 (66%)

0.8928742694854737 0.66
Train Epoch: 11 [0/50 (0%)]     Loss: 1.199281
Test set: Average loss: 0.9283, Accuracy: 29/50 (58%)

0.9282955741882324 0.58
Train Epoch: 12 [0/50 (0%)]     Loss: 1.108542
Test set: Average loss: 0.9264, Accuracy: 31/50 (62%)

0.9263894748687744 0.62
Train Epoch: 13 [0/50 (0%)]     Loss: 0.853757
Test set: Average loss: 0.9116, Accuracy: 34/50 (68%)

0.9115592193603516 0.68
Train Epoch: 14 [0/50 (0%)]     Loss: 0.907382
Test set: Average loss: 0.8540, Accuracy: 35/50 (70%)

0.8539975833892822 0.7
Train Epoch: 15 [0/50 (0%)]     Loss: 0.814083
Test set: Average loss: 0.8630, Accuracy: 35/50 (70%)

0.8630134773254394 0.7
Train Epoch: 16 [0/50 (0%)]     Loss: 0.873858
Test set: Average loss: 0.8603, Accuracy: 31/50 (62%)

0.860340461730957 0.62
Train Epoch: 17 [0/50 (0%)]     Loss: 0.757625
Test set: Average loss: 0.8772, Accuracy: 34/50 (68%)

0.8771760177612304 0.68
Train Epoch: 18 [0/50 (0%)]     Loss: 0.842209
Test set: Average loss: 0.8465, Accuracy: 37/50 (74%)

0.846537799835205 0.74
Train Epoch: 19 [0/50 (0%)]     Loss: 0.797163
Test set: Average loss: 0.8708, Accuracy: 31/50 (62%)

0.8708410263061523 0.62
Train Epoch: 20 [0/50 (0%)]     Loss: 0.749020
Test set: Average loss: 0.8106, Accuracy: 35/50 (70%)
```

```
0.8106286334991455 0.7
Train Epoch: 21 [0/50 (0%)]     Loss: 0.767750
Test set: Average loss: 0.8464, Accuracy: 33/50 (66%)

0.8464185810089111 0.66
Train Epoch: 22 [0/50 (0%)]     Loss: 0.568536
Test set: Average loss: 0.7560, Accuracy: 35/50 (70%)

0.755951042175293 0.7
Train Epoch: 23 [0/50 (0%)]     Loss: 0.609723
Test set: Average loss: 0.7410, Accuracy: 33/50 (66%)

0.7410039710998535 0.66
Train Epoch: 24 [0/50 (0%)]     Loss: 0.646144
Test set: Average loss: 0.7125, Accuracy: 36/50 (72%)

0.7124612426757813 0.72
Train Epoch: 25 [0/50 (0%)]     Loss: 0.397273
Test set: Average loss: 0.6878, Accuracy: 34/50 (68%)

0.6878188228607178 0.68
Train Epoch: 26 [0/50 (0%)]     Loss: 0.426537
Test set: Average loss: 0.6674, Accuracy: 35/50 (70%)

0.6673931503295898 0.7
Train Epoch: 27 [0/50 (0%)]     Loss: 0.490698
Test set: Average loss: 0.7037, Accuracy: 39/50 (78%)

0.7037390613555908 0.78
Train Epoch: 28 [0/50 (0%)]     Loss: 0.590016
Test set: Average loss: 0.6684, Accuracy: 34/50 (68%)

0.6684346294403076 0.68
Train Epoch: 29 [0/50 (0%)]     Loss: 0.407389
Test set: Average loss: 0.7375, Accuracy: 33/50 (66%)

0.7375208377838135 0.66
Train Epoch: 30 [0/50 (0%)]     Loss: 0.566266
Test set: Average loss: 0.6601, Accuracy: 38/50 (76%)

0.6600769424438476 0.76
Train Epoch: 31 [0/50 (0%)]     Loss: 0.339873
Test set: Average loss: 0.7060, Accuracy: 33/50 (66%)

0.7059865856170654 0.66
Train Epoch: 32 [0/50 (0%)]     Loss: 0.483949
Test set: Average loss: 0.7000, Accuracy: 35/50 (70%)
```

```
0.6999807929992676 0.7
Train Epoch: 33 [0/50 (0%)]     Loss: 0.323711
Test set: Average loss: 0.7188, Accuracy: 35/50 (70%)

0.7187582111358642 0.7
Train Epoch: 34 [0/50 (0%)]     Loss: 0.302028
Test set: Average loss: 0.7363, Accuracy: 35/50 (70%)

0.7362630271911621 0.7
Train Epoch: 35 [0/50 (0%)]     Loss: 0.269341
Test set: Average loss: 0.6412, Accuracy: 34/50 (68%)

0.6412261390686035 0.68
Train Epoch: 36 [0/50 (0%)]     Loss: 0.269679
Test set: Average loss: 0.7475, Accuracy: 33/50 (66%)

0.7475107192993165 0.66
Train Epoch: 37 [0/50 (0%)]     Loss: 0.395850
Test set: Average loss: 0.7629, Accuracy: 34/50 (68%)

0.7629396915435791 0.68
Train Epoch: 38 [0/50 (0%)]     Loss: 0.366255
Test set: Average loss: 0.6724, Accuracy: 36/50 (72%)

0.6723604106903076 0.72
Train Epoch: 39 [0/50 (0%)]     Loss: 0.531890
Test set: Average loss: 0.9044, Accuracy: 34/50 (68%)

0.9044198894500732 0.68
Train Epoch: 40 [0/50 (0%)]     Loss: 0.471708
Test set: Average loss: 0.7153, Accuracy: 33/50 (66%)

0.7153428936004639 0.66
Train Epoch: 41 [0/50 (0%)]     Loss: 0.296839
Test set: Average loss: 0.6029, Accuracy: 39/50 (78%)

0.6029071426391601 0.78
Train Epoch: 42 [0/50 (0%)]     Loss: 0.250599
Test set: Average loss: 0.7204, Accuracy: 33/50 (66%)

0.7204169797897338 0.66
Train Epoch: 43 [0/50 (0%)]     Loss: 0.204442
Test set: Average loss: 0.6820, Accuracy: 35/50 (70%)

0.6820449542999267 0.7
Train Epoch: 44 [0/50 (0%)]     Loss: 0.157063
Test set: Average loss: 0.7061, Accuracy: 36/50 (72%)
```

```
0.7061007785797119 0.72
Train Epoch: 45 [0/50 (0%)]     Loss: 0.280129
Test set: Average loss: 0.5413, Accuracy: 41/50 (82%)

0.541348533630371 0.82
Train Epoch: 46 [0/50 (0%)]     Loss: 0.159924
Test set: Average loss: 0.5864, Accuracy: 36/50 (72%)

0.5864129543304444 0.72
Train Epoch: 47 [0/50 (0%)]     Loss: 0.207981
Test set: Average loss: 0.7096, Accuracy: 34/50 (68%)

0.7095625019073486 0.68
Train Epoch: 48 [0/50 (0%)]     Loss: 0.178077
Test set: Average loss: 0.7472, Accuracy: 33/50 (66%)

0.747168402671814 0.66
Train Epoch: 49 [0/50 (0%)]     Loss: 0.137306
Test set: Average loss: 0.6097, Accuracy: 36/50 (72%)

0.6096590662002563 0.72
```

[27]:
```
# 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': []
    }
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: 1.642396
Train Epoch: 1 [640/864 (71%)]  Loss: 1.689561
Test set: Average loss: 1.6843, Accuracy: 225/864 (26%)

Test set: Average loss: 1.5443, Accuracy: 75/216 (35%)

Train Epoch: 2 [0/864 (0%)]     Loss: 1.575958
Train Epoch: 2 [640/864 (71%)]  Loss: 1.489225
Test set: Average loss: 1.5467, Accuracy: 259/864 (30%)

Test set: Average loss: 1.4197, Accuracy: 85/216 (39%)

Train Epoch: 3 [0/864 (0%)]     Loss: 1.455667
Train Epoch: 3 [640/864 (71%)]  Loss: 1.645456
Test set: Average loss: 1.4980, Accuracy: 280/864 (32%)

Test set: Average loss: 1.4879, Accuracy: 68/216 (31%)

Train Epoch: 4 [0/864 (0%)]     Loss: 1.627146
Train Epoch: 4 [640/864 (71%)]  Loss: 1.333642
Test set: Average loss: 1.5157, Accuracy: 291/864 (34%)

Test set: Average loss: 1.4738, Accuracy: 77/216 (36%)

Train Epoch: 5 [0/864 (0%)]     Loss: 1.366408
Train Epoch: 5 [640/864 (71%)]  Loss: 1.620814
Test set: Average loss: 1.4954, Accuracy: 308/864 (36%)

Test set: Average loss: 1.5680, Accuracy: 52/216 (24%)

Train Epoch: 6 [0/864 (0%)]     Loss: 1.479855
Train Epoch: 6 [640/864 (71%)]  Loss: 1.524231
Test set: Average loss: 1.5308, Accuracy: 232/864 (27%)
```

```
Test set: Average loss: 1.5924, Accuracy: 69/216 (32%)

Train Epoch: 7 [0/864 (0%)]     Loss: 1.452423
Train Epoch: 7 [640/864 (71%)]  Loss: 1.479249
Test set: Average loss: 1.4871, Accuracy: 282/864 (33%)

Test set: Average loss: 1.4564, Accuracy: 68/216 (31%)

Train Epoch: 8 [0/864 (0%)]     Loss: 1.562650
Train Epoch: 8 [640/864 (71%)]  Loss: 1.366179
Test set: Average loss: 1.4463, Accuracy: 292/864 (34%)

Test set: Average loss: 1.5645, Accuracy: 57/216 (26%)

Train Epoch: 9 [0/864 (0%)]     Loss: 1.571358
Train Epoch: 9 [640/864 (71%)]  Loss: 1.383995
Test set: Average loss: 1.5485, Accuracy: 262/864 (30%)

Test set: Average loss: 1.4434, Accuracy: 83/216 (38%)

Train Epoch: 10 [0/864 (0%)]    Loss: 1.512825
Train Epoch: 10 [640/864 (71%)] Loss: 1.397849
Test set: Average loss: 1.4526, Accuracy: 310/864 (36%)

Test set: Average loss: 1.4392, Accuracy: 69/216 (32%)

Train Epoch: 11 [0/864 (0%)]    Loss: 1.504065
Train Epoch: 11 [640/864 (71%)] Loss: 1.454877
Test set: Average loss: 1.4286, Accuracy: 316/864 (37%)

Test set: Average loss: 1.4023, Accuracy: 84/216 (39%)

Train Epoch: 12 [0/864 (0%)]    Loss: 1.484661
Train Epoch: 12 [640/864 (71%)] Loss: 1.577912
Test set: Average loss: 1.4857, Accuracy: 311/864 (36%)

Test set: Average loss: 1.5601, Accuracy: 59/216 (27%)

Train Epoch: 13 [0/864 (0%)]    Loss: 1.308768
Train Epoch: 13 [640/864 (71%)] Loss: 1.430030
Test set: Average loss: 1.4538, Accuracy: 314/864 (36%)

Test set: Average loss: 1.4391, Accuracy: 68/216 (31%)

Train Epoch: 14 [0/864 (0%)]    Loss: 1.500610
Train Epoch: 14 [640/864 (71%)] Loss: 1.323643
Test set: Average loss: 1.4664, Accuracy: 317/864 (37%)
```

```
Test set: Average loss: 1.4491, Accuracy: 73/216 (34%)

Train Epoch: 15 [0/864 (0%)]    Loss: 1.505487
Train Epoch: 15 [640/864 (71%)] Loss: 1.498564
Test set: Average loss: 1.4050, Accuracy: 342/864 (40%)

Test set: Average loss: 1.4328, Accuracy: 82/216 (38%)

Train Epoch: 16 [0/864 (0%)]    Loss: 1.382088
Train Epoch: 16 [640/864 (71%)] Loss: 1.352231
Test set: Average loss: 1.3497, Accuracy: 354/864 (41%)

Test set: Average loss: 1.3772, Accuracy: 81/216 (38%)

Train Epoch: 17 [0/864 (0%)]    Loss: 1.444847
Train Epoch: 17 [640/864 (71%)] Loss: 1.479589
Test set: Average loss: 1.4247, Accuracy: 310/864 (36%)

Test set: Average loss: 1.3995, Accuracy: 72/216 (33%)

Train Epoch: 18 [0/864 (0%)]    Loss: 1.405810
Train Epoch: 18 [640/864 (71%)] Loss: 1.365805
Test set: Average loss: 1.4087, Accuracy: 328/864 (38%)

Test set: Average loss: 1.4100, Accuracy: 70/216 (32%)

Train Epoch: 19 [0/864 (0%)]    Loss: 1.353899
Train Epoch: 19 [640/864 (71%)] Loss: 1.427612
Test set: Average loss: 1.3424, Accuracy: 352/864 (41%)

Test set: Average loss: 1.4080, Accuracy: 73/216 (34%)

Train Epoch: 20 [0/864 (0%)]    Loss: 1.324723
Train Epoch: 20 [640/864 (71%)] Loss: 1.388074
Test set: Average loss: 1.3632, Accuracy: 370/864 (43%)

Test set: Average loss: 1.4279, Accuracy: 70/216 (32%)

Train Epoch: 21 [0/864 (0%)]    Loss: 1.321498
Train Epoch: 21 [640/864 (71%)] Loss: 1.404276
Test set: Average loss: 1.3650, Accuracy: 346/864 (40%)

Test set: Average loss: 1.4162, Accuracy: 73/216 (34%)

Train Epoch: 22 [0/864 (0%)]    Loss: 1.354182
Train Epoch: 22 [640/864 (71%)] Loss: 1.349026
Test set: Average loss: 1.3790, Accuracy: 340/864 (39%)
```

```
Test set: Average loss: 1.4206, Accuracy: 70/216 (32%)

Train Epoch: 23 [0/864 (0%)]    Loss: 1.438953
Train Epoch: 23 [640/864 (71%)] Loss: 1.404784
Test set: Average loss: 1.3888, Accuracy: 343/864 (40%)

Test set: Average loss: 1.3836, Accuracy: 76/216 (35%)

Train Epoch: 24 [0/864 (0%)]    Loss: 1.476820
Train Epoch: 24 [640/864 (71%)] Loss: 1.385354
Test set: Average loss: 1.3675, Accuracy: 334/864 (39%)

Test set: Average loss: 1.3429, Accuracy: 91/216 (42%)

Train Epoch: 25 [0/864 (0%)]    Loss: 1.414404
Train Epoch: 25 [640/864 (71%)] Loss: 1.376806
Test set: Average loss: 1.3799, Accuracy: 343/864 (40%)

Test set: Average loss: 1.3628, Accuracy: 85/216 (39%)

Train Epoch: 26 [0/864 (0%)]    Loss: 1.311600
Train Epoch: 26 [640/864 (71%)] Loss: 1.357656
Test set: Average loss: 1.3730, Accuracy: 361/864 (42%)

Test set: Average loss: 1.3927, Accuracy: 76/216 (35%)

Train Epoch: 27 [0/864 (0%)]    Loss: 1.414046
Train Epoch: 27 [640/864 (71%)] Loss: 1.416384
Test set: Average loss: 1.3482, Accuracy: 355/864 (41%)

Test set: Average loss: 1.4153, Accuracy: 80/216 (37%)

Train Epoch: 28 [0/864 (0%)]    Loss: 1.341959
Train Epoch: 28 [640/864 (71%)] Loss: 1.448782
Test set: Average loss: 1.3358, Accuracy: 365/864 (42%)

Test set: Average loss: 1.3772, Accuracy: 80/216 (37%)

Train Epoch: 29 [0/864 (0%)]    Loss: 1.495070
Train Epoch: 29 [640/864 (71%)] Loss: 1.516291
Test set: Average loss: 1.3247, Accuracy: 369/864 (43%)

Test set: Average loss: 1.3605, Accuracy: 82/216 (38%)

Train Epoch: 30 [0/864 (0%)]    Loss: 1.385395
Train Epoch: 30 [640/864 (71%)] Loss: 1.557401
Test set: Average loss: 1.3435, Accuracy: 358/864 (41%)
```

```
Test set: Average loss: 1.3594, Accuracy: 83/216 (38%)

Train Epoch: 31 [0/864 (0%)]    Loss: 1.446923
Train Epoch: 31 [640/864 (71%)] Loss: 1.305975
Test set: Average loss: 1.3675, Accuracy: 363/864 (42%)

Test set: Average loss: 1.3794, Accuracy: 81/216 (38%)

Train Epoch: 32 [0/864 (0%)]    Loss: 1.323347
Train Epoch: 32 [640/864 (71%)] Loss: 1.453192
Test set: Average loss: 1.3474, Accuracy: 365/864 (42%)

Test set: Average loss: 1.3761, Accuracy: 83/216 (38%)

Train Epoch: 33 [0/864 (0%)]    Loss: 1.375228
Train Epoch: 33 [640/864 (71%)] Loss: 1.427434
Test set: Average loss: 1.3464, Accuracy: 350/864 (41%)

Test set: Average loss: 1.3669, Accuracy: 85/216 (39%)

Train Epoch: 34 [0/864 (0%)]    Loss: 1.444740
Train Epoch: 34 [640/864 (71%)] Loss: 1.397272
Test set: Average loss: 1.3517, Accuracy: 368/864 (43%)

Test set: Average loss: 1.3729, Accuracy: 83/216 (38%)

Train Epoch: 35 [0/864 (0%)]    Loss: 1.380625
Train Epoch: 35 [640/864 (71%)] Loss: 1.420936
Test set: Average loss: 1.3412, Accuracy: 372/864 (43%)

Test set: Average loss: 1.3731, Accuracy: 82/216 (38%)

Train Epoch: 36 [0/864 (0%)]    Loss: 1.448111
Train Epoch: 36 [640/864 (71%)] Loss: 1.549324
Test set: Average loss: 1.3560, Accuracy: 367/864 (42%)

Test set: Average loss: 1.3865, Accuracy: 79/216 (37%)

Train Epoch: 37 [0/864 (0%)]    Loss: 1.383034
Train Epoch: 37 [640/864 (71%)] Loss: 1.386875
Test set: Average loss: 1.3549, Accuracy: 354/864 (41%)

Test set: Average loss: 1.3658, Accuracy: 82/216 (38%)

Train Epoch: 38 [0/864 (0%)]    Loss: 1.380143
Train Epoch: 38 [640/864 (71%)] Loss: 1.362363
Test set: Average loss: 1.3644, Accuracy: 347/864 (40%)
```

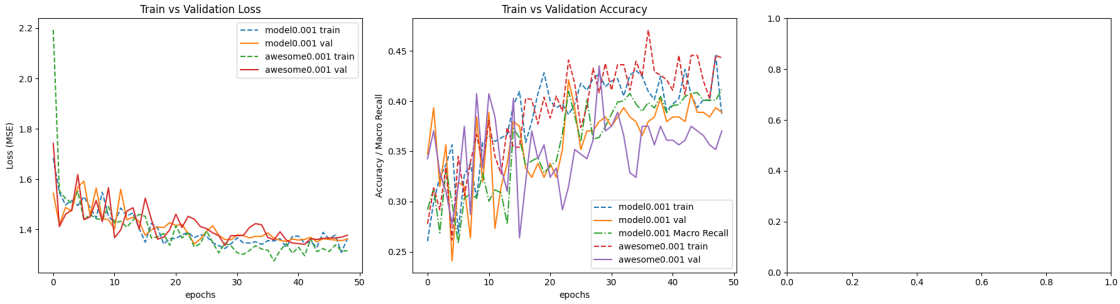
```
Test set: Average loss: 1.3574, Accuracy: 83/216 (38%)

Train Epoch: 39 [0/864 (0%)]    Loss: 1.351373
Train Epoch: 39 [640/864 (71%)] Loss: 1.429460
Test set: Average loss: 1.3293, Accuracy: 367/864 (42%)

Test set: Average loss: 1.3520, Accuracy: 87/216 (40%)

Train Epoch: 40 [0/864 (0%)]    Loss: 1.384653
Train Epoch: 40 [640/864 (71%)] Loss: 1.388518
Test set: Average loss: 1.3726, Accuracy: 337/864 (39%)

Test set: Average loss: 1.3623, Accuracy: 82/216 (38%)

Train Epoch: 41 [0/864 (0%)]    Loss: 1.333001
Train Epoch: 41 [640/864 (71%)] Loss: 1.432841
Test set: Average loss: 1.3746, Accuracy: 343/864 (40%)

Test set: Average loss: 1.3582, Accuracy: 83/216 (38%)

Train Epoch: 42 [0/864 (0%)]    Loss: 1.336418
Train Epoch: 42 [640/864 (71%)] Loss: 1.387215
Test set: Average loss: 1.3458, Accuracy: 348/864 (40%)

Test set: Average loss: 1.3611, Accuracy: 83/216 (38%)

Train Epoch: 43 [0/864 (0%)]    Loss: 1.293901
Train Epoch: 43 [640/864 (71%)] Loss: 1.370599
Test set: Average loss: 1.3525, Accuracy: 373/864 (43%)

Test set: Average loss: 1.3693, Accuracy: 82/216 (38%)

Train Epoch: 44 [0/864 (0%)]    Loss: 1.466611
Train Epoch: 44 [640/864 (71%)] Loss: 1.460432
Test set: Average loss: 1.3296, Accuracy: 351/864 (41%)

Test set: Average loss: 1.3510, Accuracy: 88/216 (41%)

Train Epoch: 45 [0/864 (0%)]    Loss: 1.367427
Train Epoch: 45 [640/864 (71%)] Loss: 1.259825
Test set: Average loss: 1.3891, Accuracy: 340/864 (39%)

Test set: Average loss: 1.3638, Accuracy: 84/216 (39%)

Train Epoch: 46 [0/864 (0%)]    Loss: 1.466855
Train Epoch: 46 [640/864 (71%)] Loss: 1.321473
Test set: Average loss: 1.3629, Accuracy: 347/864 (40%)
```

```
Test set: Average loss: 1.3608, Accuracy: 84/216 (39%)

Train Epoch: 47 [0/864 (0%)]    Loss: 1.357319
Train Epoch: 47 [640/864 (71%)] Loss: 1.382298
Test set: Average loss: 1.3783, Accuracy: 346/864 (40%)

Test set: Average loss: 1.3587, Accuracy: 83/216 (38%)

Train Epoch: 48 [0/864 (0%)]    Loss: 1.392995
Train Epoch: 48 [640/864 (71%)] Loss: 1.438382
Test set: Average loss: 1.3089, Accuracy: 385/864 (45%)

Test set: Average loss: 1.3561, Accuracy: 85/216 (39%)

Train Epoch: 49 [0/864 (0%)]    Loss: 1.334245
Train Epoch: 49 [640/864 (71%)] Loss: 1.312263
Test set: Average loss: 1.3657, Accuracy: 333/864 (39%)

Test set: Average loss: 1.3622, Accuracy: 84/216 (39%)
```

[28]:
```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 (MSE)') # Mise à jour du label Y pour MSE
            ax1.set_xlabel('epochs')
            ax1.set_title('Train vs Validation Loss')

            # --- Courbes d'ACCURACY (ax2) ---
```

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

            # AJOUT DU MACRO RECALL SI DÉFINI
            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')

# --- MATRICE DE CONFUSION (ax3) ---
# Nécessite les prédictions et targets finales (non incluses dans le .pkl␣
 ↪standard)
# SI vous avez sauvegardé all_val_preds et all_val_targets dans results:

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'])
```


Train vs Validation Loss
model0.001 train
model0.001 val
awesome0.001 train
awesome0.001 val
Loss (MSE)
epochs
Train vs Validation Accuracy
model0.001 train
model0.001 val
model0.001 Macro Recall
awesome0.001 train
awesome0.001 val
Accuracy / Macro Recall
epochs