

DEEP LEARNING

HOLLIER Laëtitia
MANONGO Nzila
GESTEL Marie



Introduction

CONTEXTE DU PROJET

dataset

```
Dataset Flowers102
  Number of datapoints: 1020
  Root location: ./data
  split=train
  StandardTransform
  Transform: Compose(
    Resize(size=(112, 112), interpolation=bilinear, max_size=None, antialias=True)
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

8 000 Images - 1 020 importées

Import de nos données

```
# Définir les transformations pour les données
transform_couleur = transforms.Compose([
    transforms.Resize((112, 112)), # Redimensionner les images à 112x112
    transforms.ToTensor(), # Convertir les images en tenseurs PyTorch
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Normaliser avec les statistiques
])

# Définir une nouvelle transformation pour convertir les images en niveaux de gris
transform = transforms.Compose([
    transforms.Resize((112, 112)), # Redimensionner les images à 112x112
    transforms.Grayscale(), # Convertir les images en niveaux de gris
    transforms.ToTensor(), # Convertir les images en tenseurs PyTorch
])

# Télécharger et charger le jeu de données Flowers102
dataset = datasets.Flowers102(root='./data',
                               split='train',
                               transform=transform_couleur,
                               download=True)

# Définir les proportions pour l'entraînement et le test
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size

# Diviser dataset1 en ensemble d'entraînement et de test
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])
```

```
print("nombre d'échantillons d'entraînement: " + str(len(train_dataset)) + "\n" +
      "nombre d'échantillons de test: " + str(len(test_dataset)))
```

```
nombre d'échantillons d'entraînement: 816
nombre d'échantillons de test: 204
```

```
print("type de données du 1er échantillon d'entraînement: ", train_dataset[0][0].type())
print("taille du 1er échantillon d'entraînement: ", train_dataset[0][0].size())
```

```
type de données du 1er échantillon d'entraînement: torch.FloatTensor
taille du 1er échantillon d'entraînement: torch.Size([3, 112, 112])
```

Import de nos données

```
# Créer les DataLoader pour itérer sur les jeux de données
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

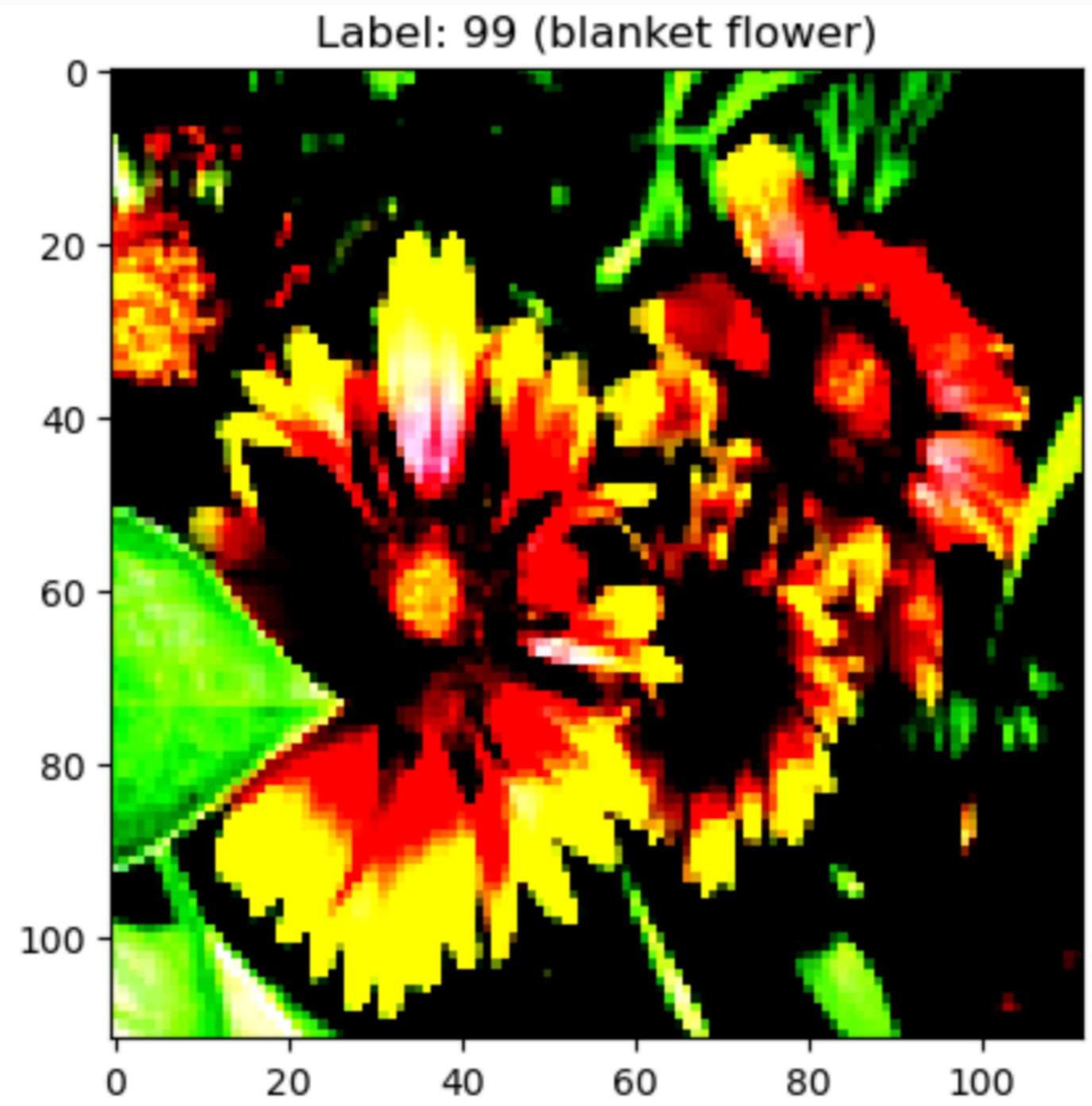
# Fonction pour afficher une image
def imshow(img, title=None):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)), cmap='gray')
    if title is not None:
        plt.title(title)
    plt.show()

# Obtenir les deux premières images de l'ensemble d'entraînement
dataiter = iter(train_loader)
images, labels = next(dataiter)

# Afficher les deux premières images avec leurs étiquettes
for i in range(2):
    imshow(images[i], title=f'Label: {labels[i].item()} ({class_names[labels[i].item()]})')

print(f'Les deux premières étiquettes dans l\'ensemble d\'entraînement sont : {labels[0].item()} ({class_names[labels[0].item()]}) et {labels[1].item()} ({class_names[labels[1].item()]})')

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Régression logistique multinomiale

```
# Création du module personnalisé pour la régression logistique
class RegressionLogistique(torch.nn.Module):

    # Définition du constructeur
    def __init__(self, n_entrees, n_sorties):
        super().__init__()
        self.lineaire = torch.nn.Linear(n_entrees, n_sorties)

    # Prédiction
    def forward(self, x):
        # y_pred = torch.sigmoid(self.lineaire(x))
        y_pred = torch.softmax(self.lineaire(x), dim=1) # Utiliser softmax pour la
        return y_pred

# Définir le modèle
n_entrees = 112 * 112 * 3 # Dimension d'entrée des images (112x112 avec 3 canaux)
n_sorties = 102 # Nombre de classes dans Flowers102
reg_log = RegressionLogistique(n_entrees, n_sorties)
```

```
# Définition de l'optimiseur
optimiseur = torch.optim.SGD(reg_log.parameters(), lr=0.001)

# Définition de la perte d'entropie croisée
critere = torch.nn.CrossEntropyLoss()

epochs = 25
Perte = []
acc = []

for epoque in range(epochs):
    reg_log.train()
    for images, etiquettes in train_loader:
        optimiseur.zero_grad()
        images = images.view(-1, 112 * 112 * 3) # Aplatir les images
        sorties = reg_log(images)
        perte = critere(sorties, etiquettes)
        perte.backward()
        optimiseur.step()

    Perte.append(perte.item())

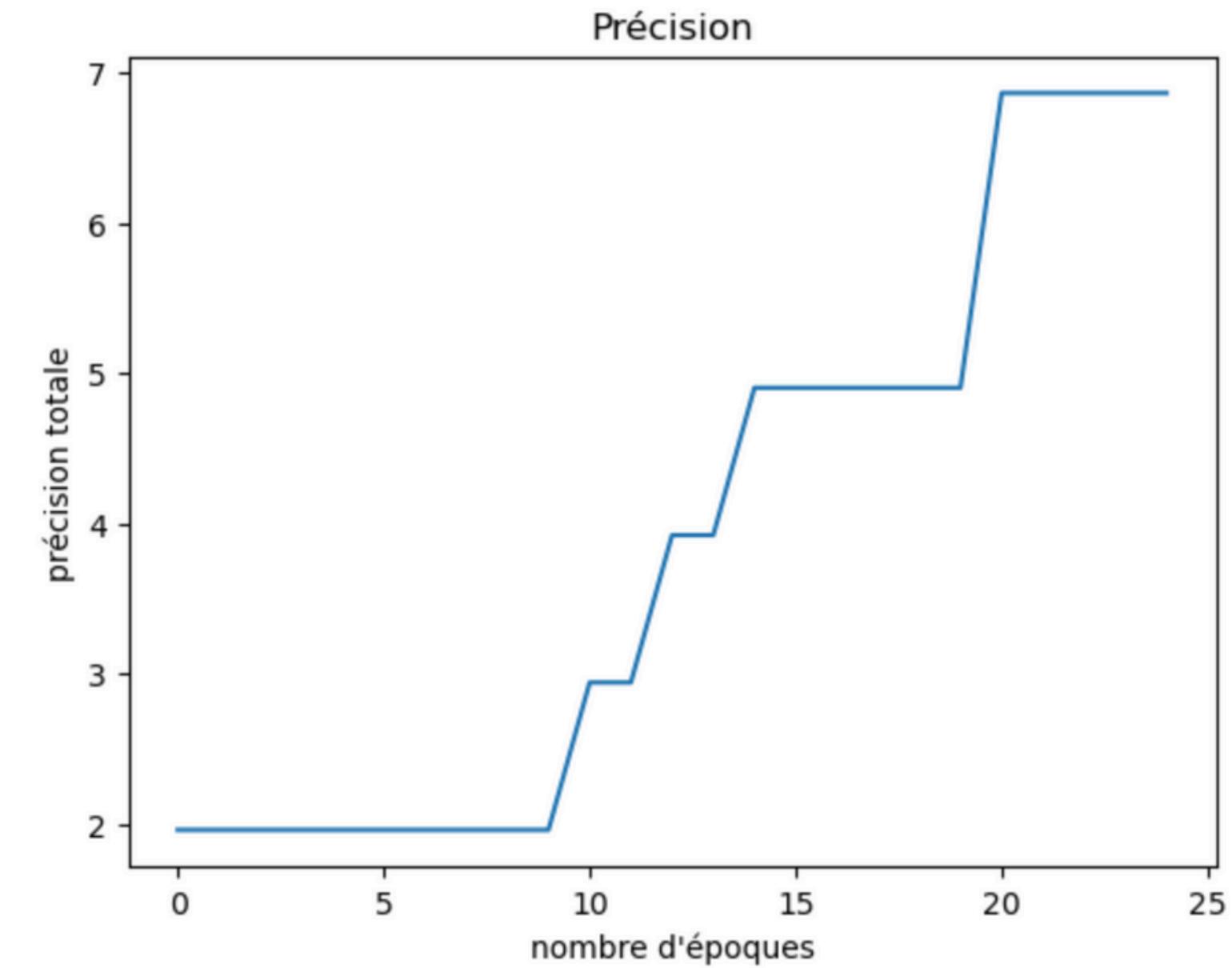
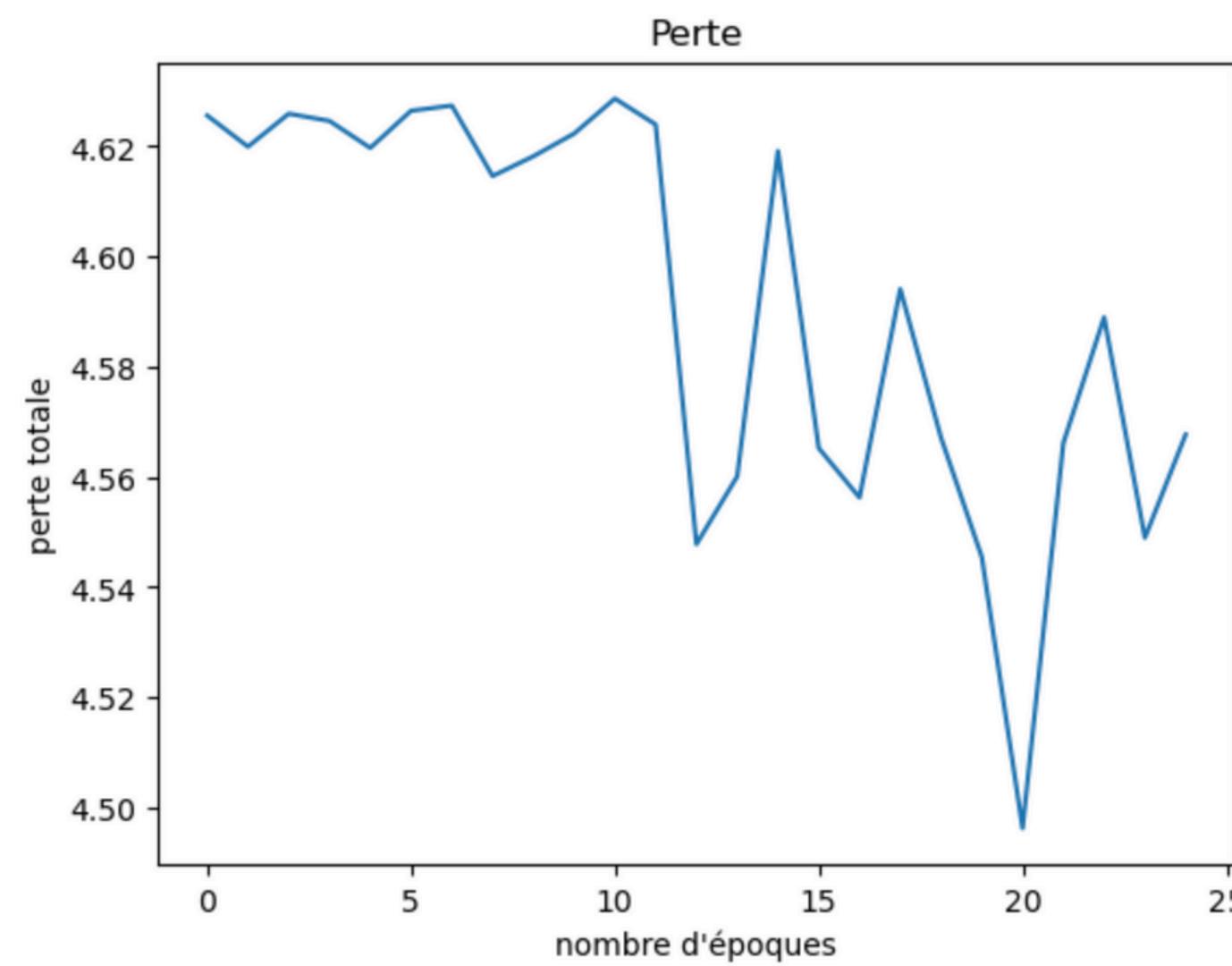
    correct = 0
    reg_log.eval()
    with torch.no_grad():
        for images, etiquettes in test_loader:
            images = images.view(-1, 112 * 112 * 3) # Aplatir les images
            sorties = reg_log(images)
            _, predit = torch.max(sorties.data, 1)
            correct += (predit == etiquettes).sum().item()

    precision = 100 * correct / len(test_dataset)
    acc.append(precision)

    if epoque % 5 == 0:
        print('Époque : {}. Perte : {}. Précision : {}'.format(epoque, perte.item(), precision))
```

Régression logistique multinomiale

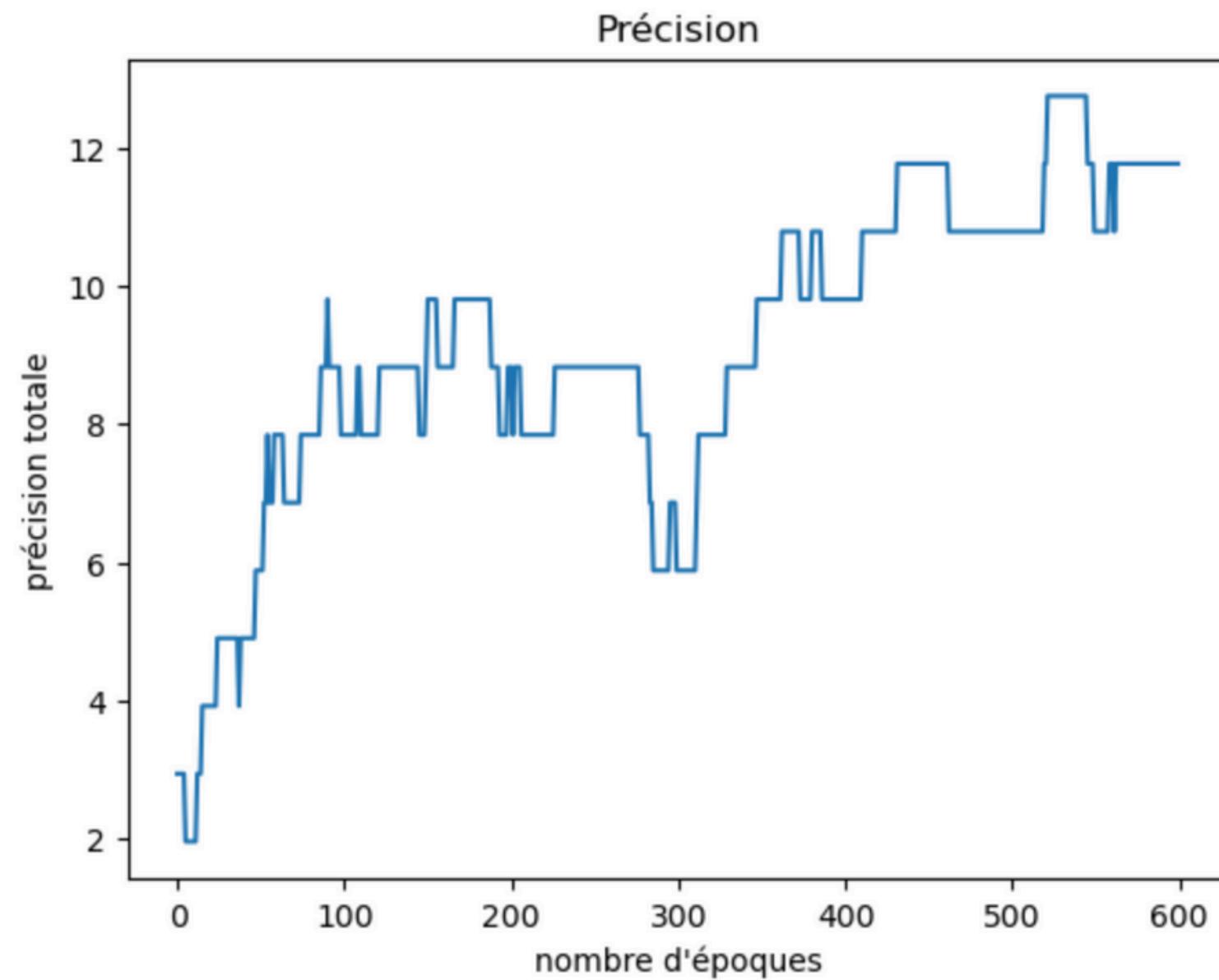
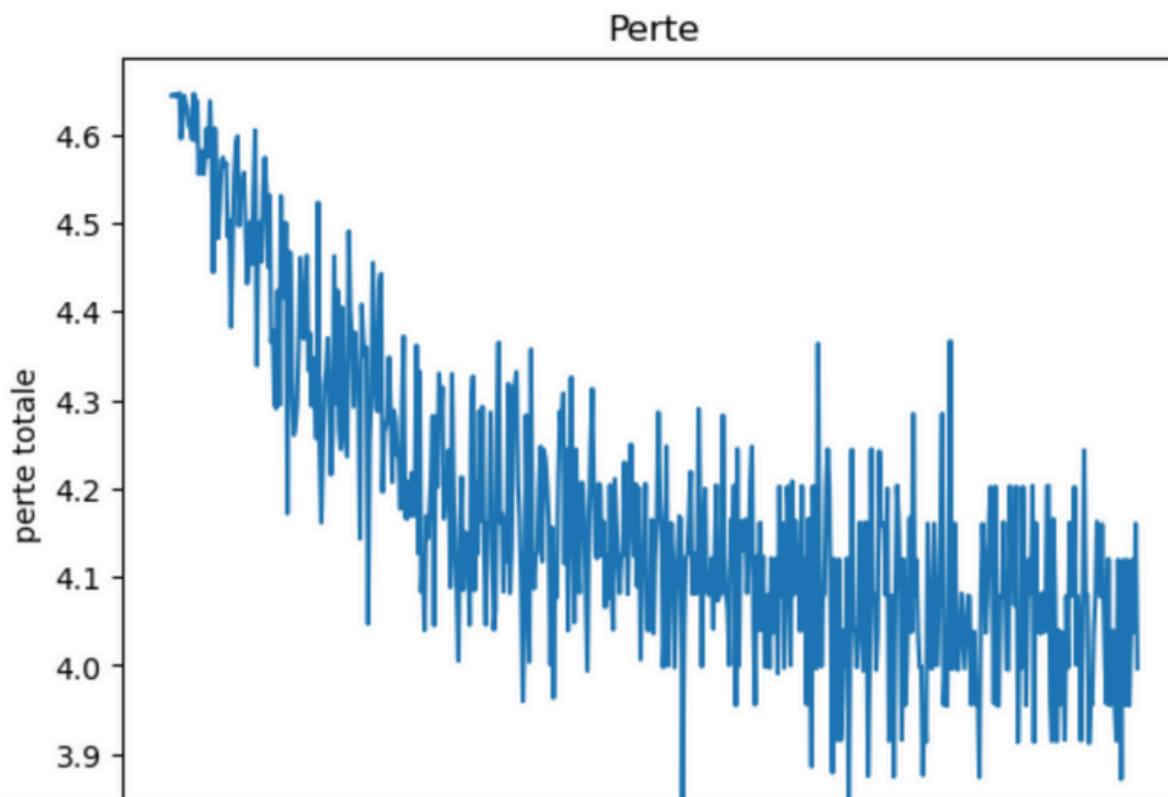
```
Époque : 0. Perte : 4.625542163848877. Précision : 1.9607843137254901
Époque : 5. Perte : 4.626407146453857. Précision : 1.9607843137254901
Époque : 10. Perte : 4.628617763519287. Précision : 2.9411764705882355
Époque : 15. Perte : 4.5651936531066895. Précision : 4.901960784313726
Époque : 20. Perte : 4.496212482452393. Précision : 6.862745098039215
```



Régression logistique multinomiale

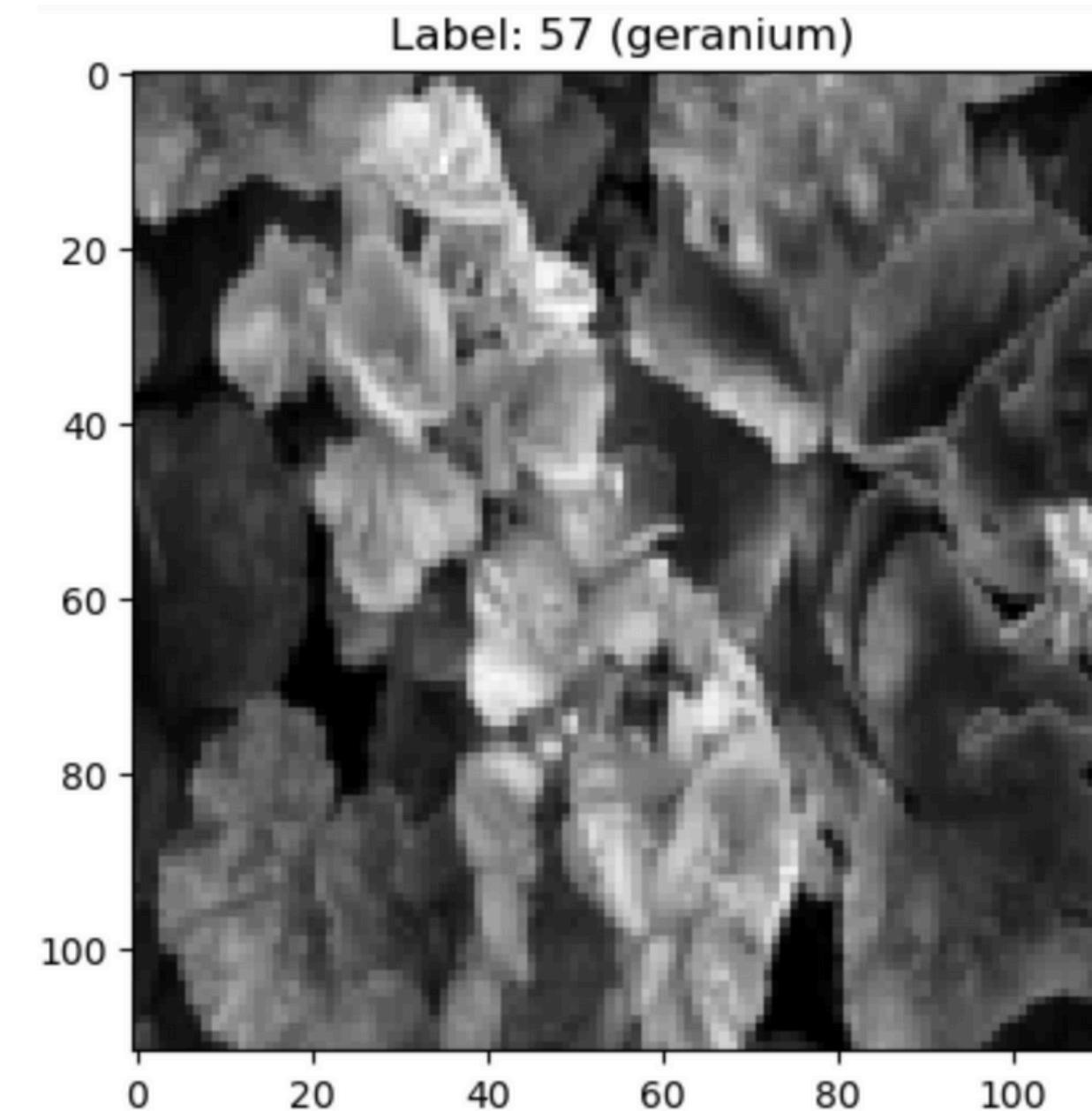
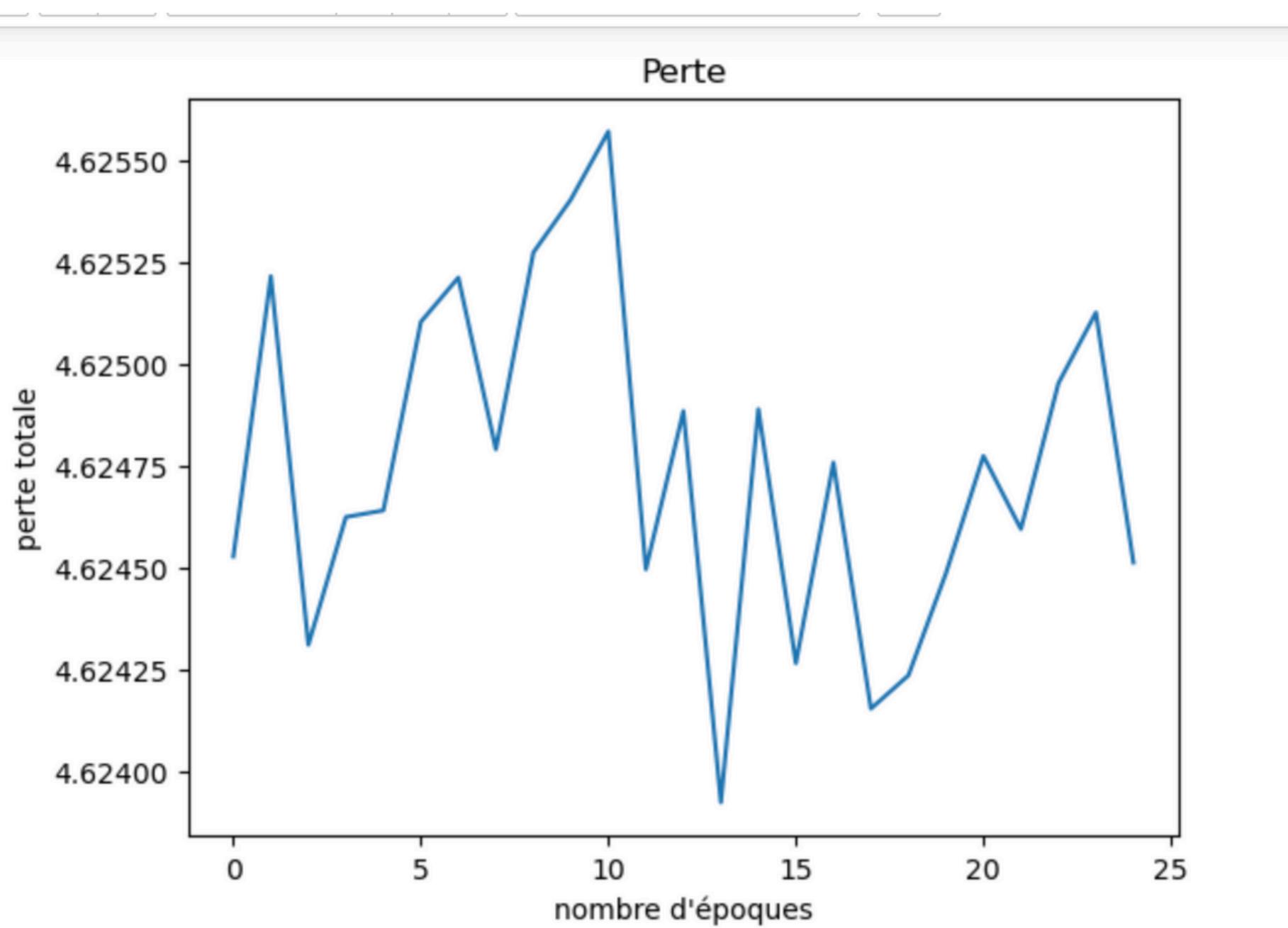
```
Époque : 0. Perte : 4.6440253257751465. Précision : 2.9411764705882355
Époque : 50. Perte : 4.453415870666504. Précision : 5.882352941176471
Époque : 100. Perte : 4.260499954223633. Précision : 7.8431372549019605
Époque : 150. Perte : 4.168687343597412. Précision : 9.803921568627452
Époque : 200. Perte : 4.040130615234375. Précision : 8.823529411764707
Époque : 250. Perte : 4.048383712768555. Précision : 8.823529411764707
Époque : 300. Perte : 4.123885631561279. Précision : 5.882352941176471
Époque : 350. Perte : 3.9543616771698. Précision : 9.803921568627452
Époque : 400. Perte : 3.996075391769409. Précision : 9.803921568627452
Époque : 450. Perte : 4.203063488006592. Précision : 11.764705882352942
Époque : 500. Perte : 3.9798154830932617. Précision : 10.784313725490197
Époque : 550. Perte : 4.039781093597412. Précision : 10.784313725490197
```

```
plt.plot(Perte)
plt.xlabel("nombre d'époques")
plt.ylabel("perte totale")
plt.title("Perte")
plt.show()
```



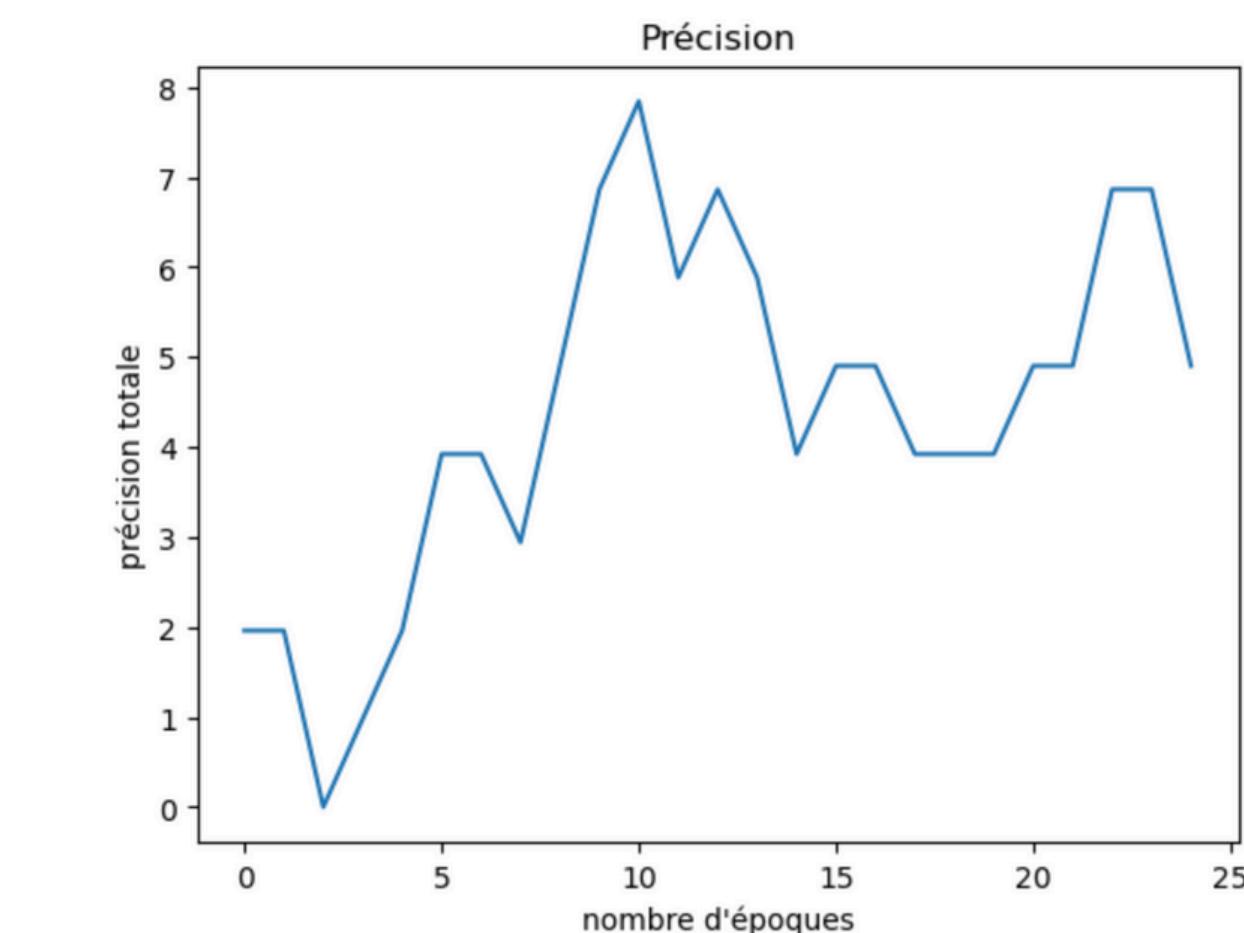
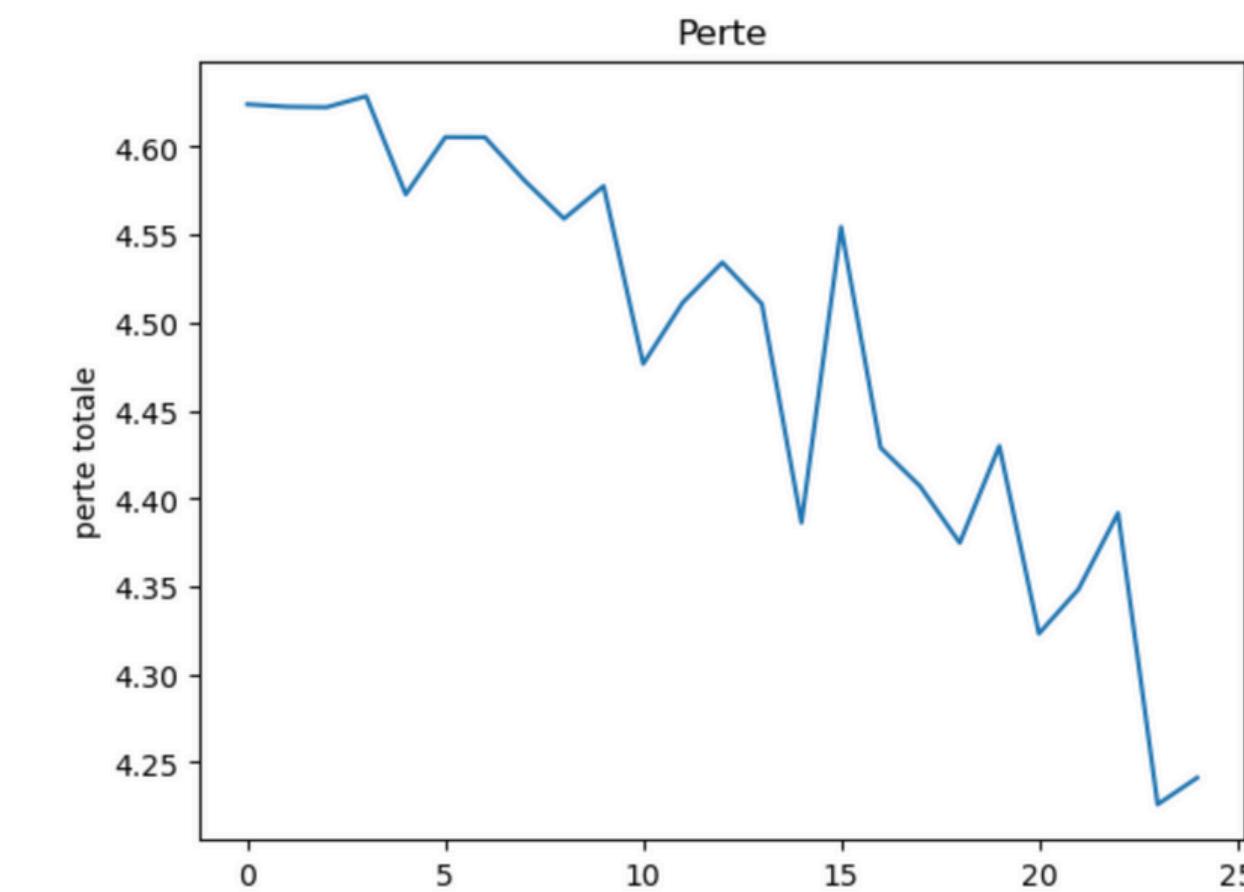
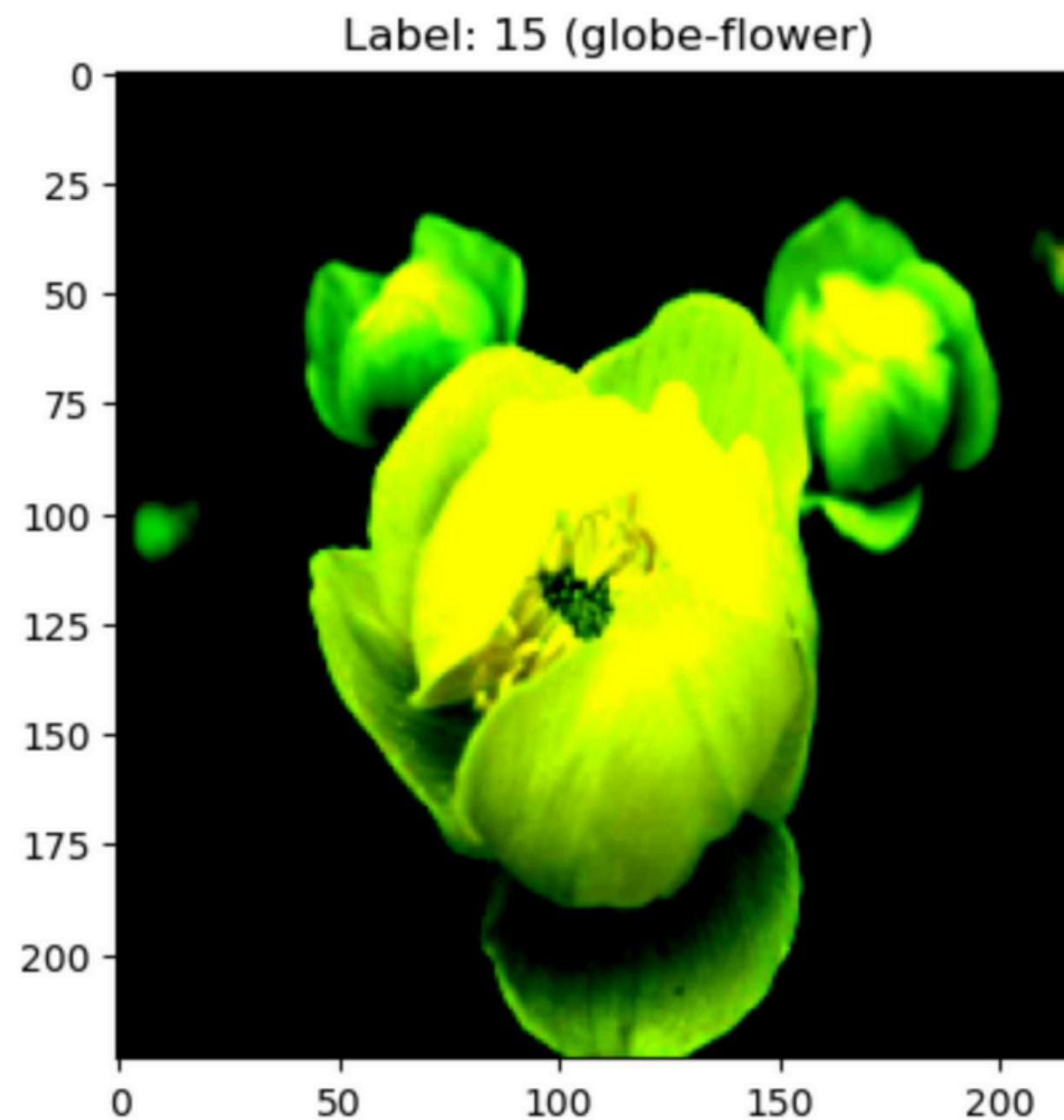
Régression logistique multinomiale

Image en gris (112*112)



Régression logistique multinomiale

Image dimension 224*224*3



CNN - Convolution

Définir le modèle CNN

Aplatir pour $112 * 112 * 3$

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=102):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.fc1 = nn.Linear(32 * 28 * 28, 512)
        self.fc2 = nn.Linear(512, num_classes)

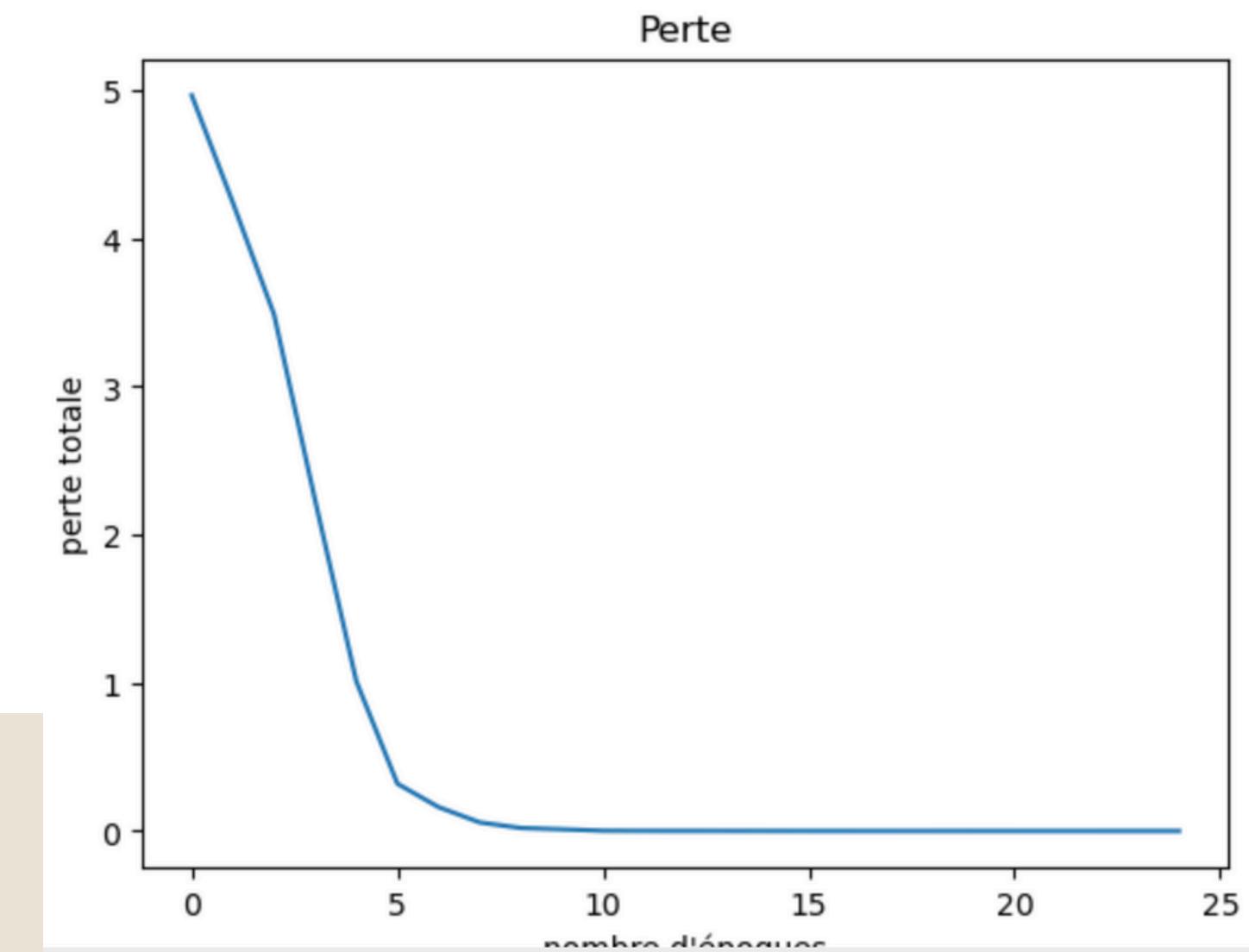
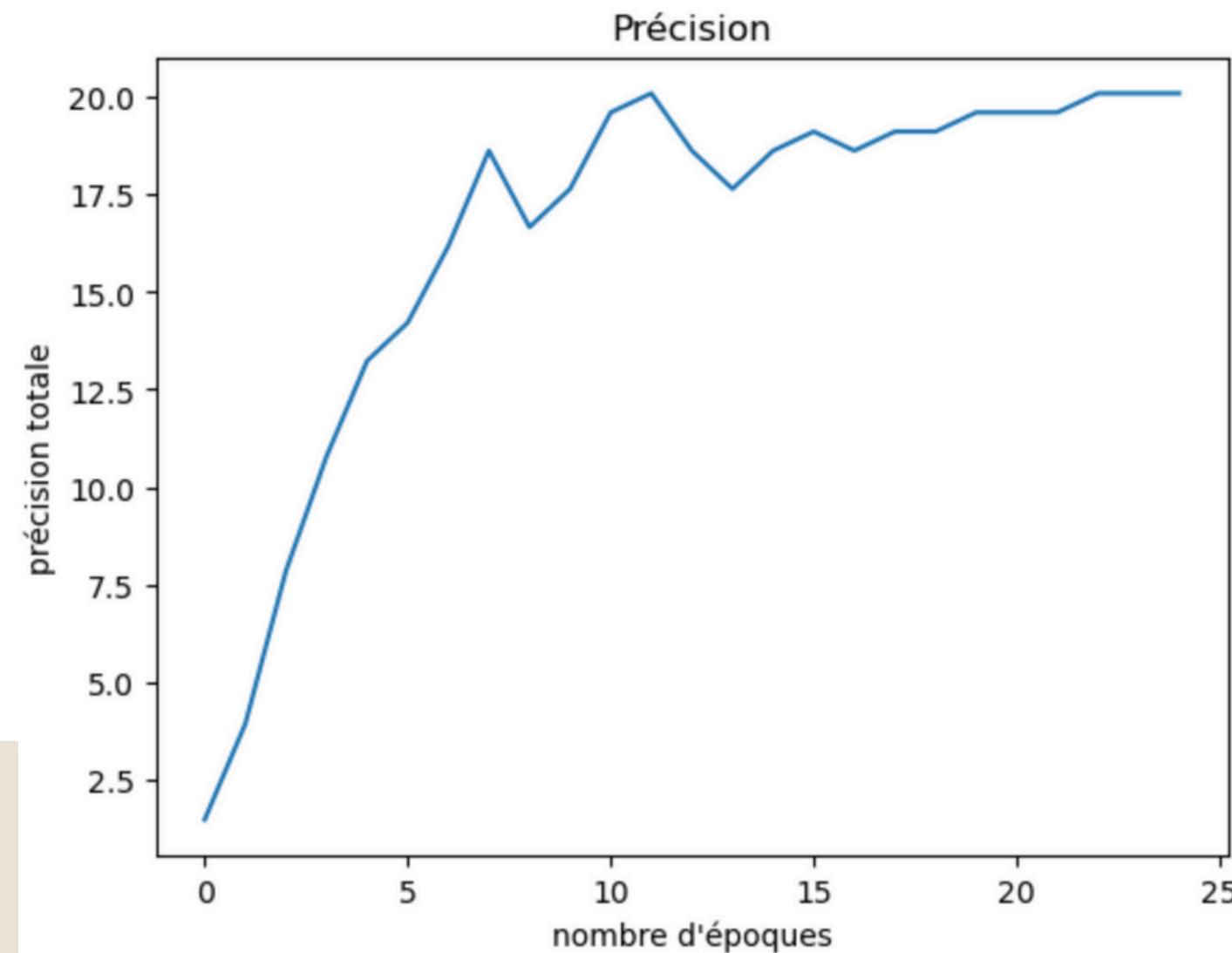
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(x.size(0), -1) # Aplatir
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x), dim=1)
        return x

model = SimpleCNN(num_classes=102)
```

```
# Définition de l'optimiseur et de la perte d'entropie croisée
optimiseur = torch.optim.Adam(model.parameters(), lr=0.001)
critere = nn.CrossEntropyLoss()
```

CNN - Convolution

```
Époque : 0, Perte : 4.964960776842558, Précision : 1.4705882352941178
Époque : 5, Perte : 0.31910361177646196, Précision : 14.215686274509803
Époque : 10, Perte : 0.0021465728014635923, Précision : 19.607843137254903
Époque : 15, Perte : 0.0005826904548135084, Précision : 19.11764705882353
Époque : 20, Perte : 0.0003404366294629514, Précision : 19.607843137254903
Nombre d'images dans le dataset original: 1020
Nombre d'images dans le train_loader: 816
Nombre d'images dans le test_loader: 204
```



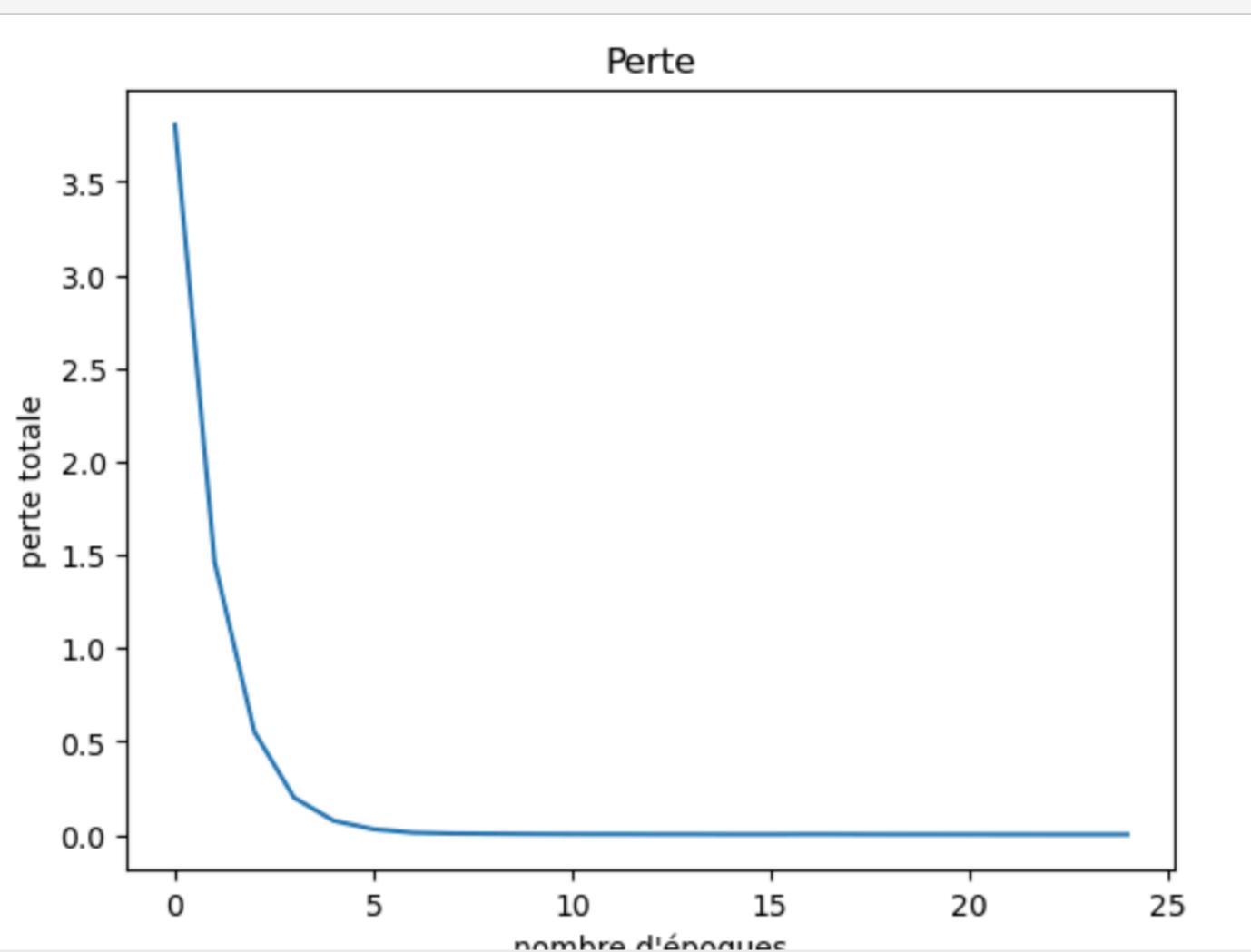
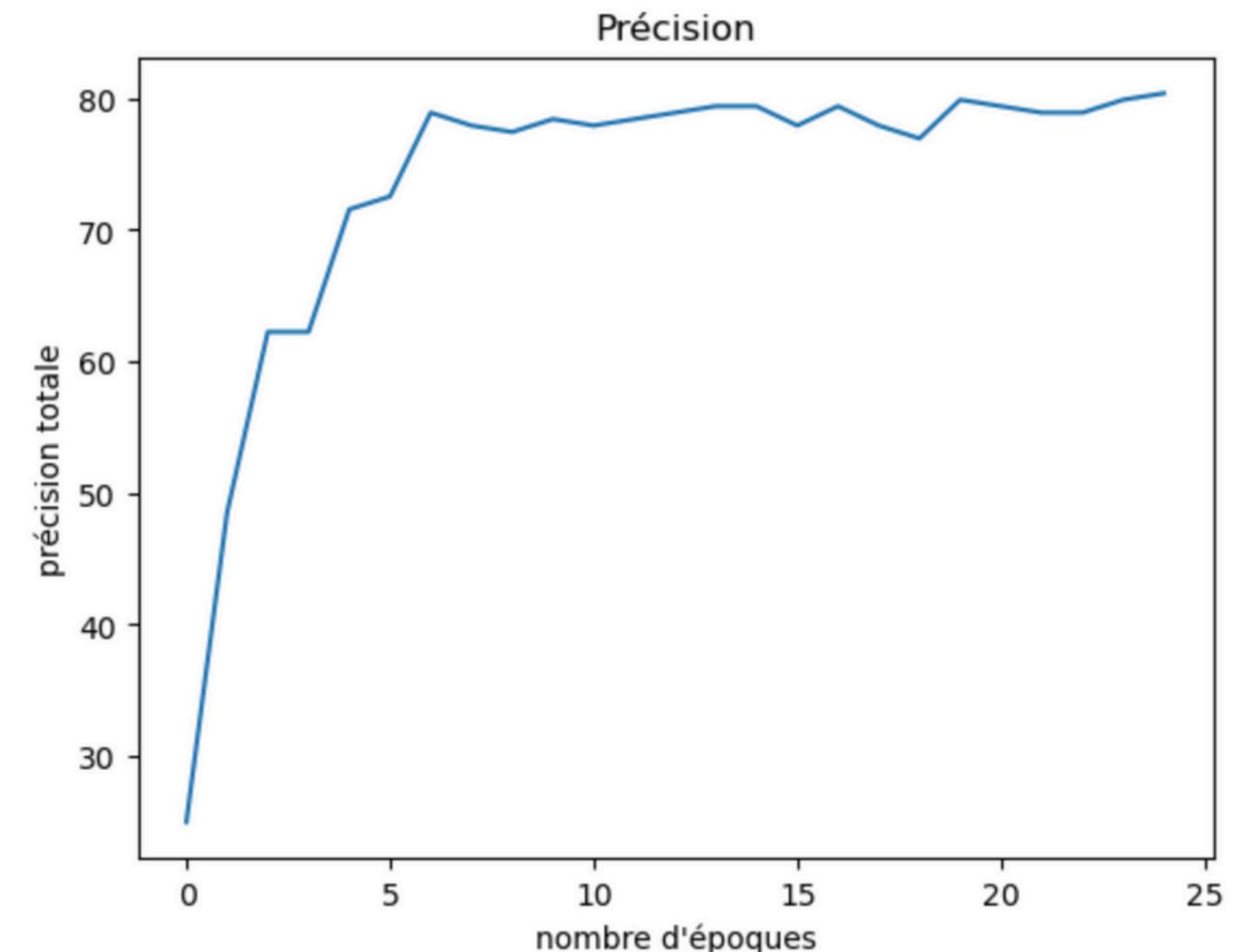
ResNet

```
# Charger le modèle pré-entraîné ResNet18
model = models.resnet18(pretrained=True)

# Remplacer la couche fully connected pour correspondre au nombre de classes
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 102)

# Définir l'optimiseur et la fonction de perte
optimiseur = optim.Adam(model.parameters(), lr=0.001)
critere = nn.CrossEntropyLoss()
```

```
Époque : 0, Perte : 3.805476555457482, Précision : 25.0
Époque : 5, Perte : 0.029064920014486864, Précision : 72.54901960784314
Époque : 10, Perte : 0.002719340013125195, Précision : 77.94117647058823
Époque : 15, Perte : 0.0015523602845720374, Précision : 77.94117647058823
Époque : 20, Perte : 0.0012268209852314054, Précision : 79.41176470588235
Nombre d'images dans le dataset original: 1020
Nombre d'images dans le train_loader: 816
Nombre d'images dans le test_loader: 204
```



Saimple

```
import torch
import torchvision.models as models

# Définir des données factices en entrée
dummy_input = torch.randn(1, 3, 112, 112) # Les dimensions doivent être identiques à l'entrée du modèle.

# Spécifier le chemin de sauvegarde du modèle ONNX
onnx_path = "flower_model.onnx"

# Exporter le modèle en format ONNX
torch.onnx.export(model, dummy_input, onnx_path)
```

Saimple + New Evaluation Evaluations List

test flower
Input: image_00002.jpg
Model: flower_model.onnx
Noise Intensity: 1e-5

Summary Dominance Relevance Images Details

Evaluation ID: unruffled_booth

Evaluation Description

Creation Date: Jun 4, 2024, 12:49:12 PM

Precision: Float Noise Type: ADDITIVE Normalized: Yes

00:00:04

The evaluation results are not available.
Browse logs for more information.
→ See Logs

3

Merci.