TP3-TOURE-Boubacar-M2

February 12, 2023

1 TP3 - Deep Learning : Méthodologie, Expérimentations et Régularisation

Auteur : TOURE Boubacar (boubacar.toure@etud.univ-angers.fr), étudiant en Master 2 Informatique à la faculté des sciences d'Angers

Professeur Référent : Sylvain Lamprier (sylvain.lamprier@univ-angers.fr)

Supports adaptés de Nicolas Baskiotis (nicolas.baskiotis@sorbonne-univeriste.fr) et Benjamin Piwowarski (benjamin.piwowarski@sorbonne-universite.fr) – MLIA/ISIR, Sorbonne Université

2 Import du projet

[2]:
```
import matplotlib.pyplot as plt
import torch.nn as nn
import shutil
import torch
import time
import os

from torch.utils.tensorboard import SummaryWriter
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
from torchvision import transforms

from tqdm.autonotebook import tqdm
from tqdm import tqdm
```

3 Déclaration des fonctions

[3]:
```
def save_state(fichier,epoch,model,optim):
    state = {'epoch' : epoch, 'model_state': model.state_dict(), 'optim_state':␣
 ↪optim.state_dict()}
    torch.save(state,fichier)

def load_state(fichier,model,optim):
    epoch = 0
```

```
    if os.path.isfile(fichier):
        state = torch.load(fichier)
        model.load_state_dict(state['model_state'])
        optim.load_state_dict(state['optim_state'])
        epoch = state['epoch']
    return epoch

def unnormalize(img, mean, std):
    if img.dim()==2 or ((img.dim()==3) and (img.size()[0]==1)):
        return img*std[0]+mean[0]
    return img * img.new(std).view(3, 1, 1) + img.new(mean).view(3, 1, 1)


def train(model, train_loader, validation_loader, loss_fn, optimizer, epochs,␣
 ↪typeTrain=""):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    loss_fn = loss_fn.to(device)

    # Clear any logs from previous runs
    if os.path.exists(f"/tmp/logs/deepLearning/{model.name}-train"):
        shutil.rmtree(f"/tmp/logs/deepLearning/{model.name}-train")

    # On créé un writer avec la date du modèle pour s'y retrouver
    TB_PATH = f"/tmp/logs/deepLearning"
    MODEL_PATH = "/tmp/models/"
    os.makedirs(MODEL_PATH, exist_ok=True)
    check_file = f"{MODEL_PATH}/{model.name}-train-{typeTrain}.pth"

    summary = SummaryWriter(f"{TB_PATH}/{model.name}-train")

    train_losses = []
    validation_losses = []
    accuraciesOfValidation = []
    accuraciesOfTrain = []

    start_epoch = load_state(check_file, model, optimizer)

    start_time = time.time()
    for epoch in range(start_epoch, epochs+1):
        # Entraînement sur les données d'entraînement
        model.train()
        train_loss = 0.0
        correct = 0
        total = 0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)
```

```
            optimizer.zero_grad()
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()
            loss = loss_fn(outputs, targets)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        accuracy = correct / total
        accuraciesOfTrain.append(accuracy)
        summary.add_scalar("Accuracy_Of_Train", accuracy, epoch)
        summary.add_scalar("Loss_Of_Train", train_loss/len(train_loader), epoch)
        train_losses.append(train_loss/len(train_loader))

        if epoch % 10 == 0:
            # Evaluation sur les données de test
            save_state(check_file, epoch, model, optimizer)
            model.eval()
            with torch.no_grad():
                validation_loss = 0
                correct = 0
                total = 0
                for inputs, labels in validation_loader:
                    inputs, labels = inputs.to(device), labels.to(device)
                    outputs = model(inputs)
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()
                    validation_loss += loss_fn(outputs, labels).item()

            accuracy = correct / total
            accuraciesOfValidation.append(accuracy)
            summary.add_scalar("Accuracy_Of_Validation", accuracy, epoch)
            summary.add_scalar("Loss_Of_Validation", validation_loss/
↪len(validation_loader), epoch)
            validation_losses.append(validation_loss/len(validation_loader))

            print("\tEpoch {}, Validation Loss: {:.4f}, Validation Accuracy: {:.
↪4f}".format(epoch, validation_loss, accuracy))
        else:
            print("Epoch {}, Train Loss: {:.4f}, Train Accuracy: {:.4f}".
↪format(epoch, train_loss/len(train_loader), accuracy))
    end_time = time.time()

    trainTime = end_time - start_time
```

```
    trainModelComplexity = sum(p.numel() for p in model.parameters() if p.
 ↪requires_grad)

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(train_losses)
    plt.title('Train Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfTrain)
    plt.title('Train Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(validation_losses)
    plt.title('Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfValidation)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()

    # Load the TensorBoard notebook extension
    %load_ext tensorboard
    %tensorboard --logdir {TB_PATH}/{model.name}-train

    return trainTime, trainModelComplexity, validation_loss, accuracy

def set_dropout_rate(m, rate):
    if type(m) == nn.Dropout:
        m.p = rate

def train_dropout(model, train_loader, validation_loader, loss_fn, optimizer,␣
 ↪epochs, dropout_rate, typeTrain=""):
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
loss_fn = loss_fn.to(device)

# Clear any logs from previous runs
if os.path.exists(f"/tmp/logs/deepLearning/{model.name}-train_dropout"):
    shutil.rmtree(f"/tmp/logs/deepLearning/{model.name}-train_dropout")

# On créé un writer avec la date du modèle pour s'y retrouver
TB_PATH = f"/tmp/logs/deepLearning"
MODEL_PATH = "/tmp/models/"
os.makedirs(MODEL_PATH, exist_ok=True)
check_file = f"{MODEL_PATH}/{model.name}-train_dropout-{typeTrain}.pth"

summary = SummaryWriter(f"{TB_PATH}/{model.name}-train_dropout")

train_losses = []
validation_losses = []
accuraciesOfValidation = []
accuraciesOfTrain = []

start_epoch = load_state(check_file, model, optimizer)

start_time = time.time()
for epoch in range(start_epoch, epochs+1):
    # Entraînement sur les données d'entraînement
    model.train()
    model.apply(lambda x: set_dropout_rate(x, dropout_rate))
    train_loss = 0.0
    correct = 0
    total = 0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += targets.size(0)
        correct += (predicted == targets).sum().item()
        loss = loss_fn(outputs, targets)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    accuracy = correct / total
    accuraciesOfTrain.append(accuracy)
    summary.add_scalar("Accuracy_Of_Train", accuracy, epoch)
    summary.add_scalar("Loss_Of_Train", train_loss/len(train_loader), epoch)
```

```
        train_losses.append(train_loss/len(train_loader))

        if epoch % 10 == 0:
            # Evaluation sur les données de test
            model.eval()
            model.apply(lambda x: set_dropout_rate(x, 0))
            with torch.no_grad():
                validation_loss = 0
                correct = 0
                total = 0
                for inputs, labels in validation_loader:
                    inputs, labels = inputs.to(device), labels.to(device)
                    outputs = model(inputs)
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0)
                    correct += (predicted == labels).sum().item()
                    validation_loss += loss_fn(outputs, labels).item()

            accuracy = correct / total
            accuraciesOfValidation.append(accuracy)
            summary.add_scalar("Accuracy", accuracy, epoch)
            summary.add_scalar("Loss_Of_Validation", validation_loss/
↪len(validation_loader), epoch)
            validation_losses.append(validation_loss/len(validation_loader))

            print("\tEpoch {}, Validation Loss: {:.4f}, Validation Accuracy: {:.
↪4f}".format(epoch, validation_loss, accuracy))
        else:
            print("Epoch {}, Train Loss: {:.4f}, Train Accuracy: {:.4f}".
↪format(epoch, train_loss/len(train_loader), accuracy))
    end_time = time.time()
    trainTime = end_time - start_time

    trainModelComplexity = sum(p.numel() for p in model.parameters() if p.
↪requires_grad)

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(train_losses)
    plt.title('Train Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfTrain)
    plt.title('Train Accuracy')
```

```
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(validation_losses)
    plt.title('Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfValidation)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()

    # Load the TensorBoard notebook extension
    %load_ext tensorboard
    %tensorboard --logdir {TB_PATH}/{model.name}-train_dropout

    return trainTime, trainModelComplexity, validation_loss, accuracy

def train_batchnorm(model, train_loader, validation_loader, loss_fn, optimizer,␣
 ↪epochs, typeTrain=""):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    loss_fn = loss_fn.to(device)

    # Clear any logs from previous runs
    if os.path.exists(f"/tmp/logs/deepLearning/{model.name}-train_batchnorm"):
        shutil.rmtree(f"/tmp/logs/deepLearning/{model.name}-train_batchnorm")

    # On créé un writer avec la date du modèle pour s'y retrouver
    TB_PATH = f"/tmp/logs/deepLearning"
    MODEL_PATH = "/tmp/models/"
    os.makedirs(MODEL_PATH, exist_ok=True)
    check_file = f"{MODEL_PATH}/{model.name}-train_batchnorm-{typeTrain}.pth"

    summary = SummaryWriter(f"{TB_PATH}/{model.name}-train_batchnorm")

    train_losses = []
    validation_losses = []
```

```
accuraciesOfValidation = []
accuraciesOfTrain = []

start_epoch = load_state(check_file, model, optimizer)

start_time = time.time()
for epoch in range(start_epoch, epochs+1):
    # Entraînement sur les données d'entraînement
    model.train()
    train_loss = 0.0
    correct = 0
    total = 0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += targets.size(0)
        correct += (predicted == targets).sum().item()
        loss = loss_fn(outputs, targets)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    accuracy = correct / total
    accuraciesOfTrain.append(accuracy)
    summary.add_scalar("Accuracy_Of_Train", accuracy, epoch)
    summary.add_scalar("Loss_Of_Train", train_loss/len(train_loader), epoch)
    train_losses.append(train_loss/len(train_loader))

    if epoch % 10 == 0:
        # Evaluation sur les données de test
        model.eval()
        with torch.no_grad():
            validation_loss = 0
            correct = 0
            total = 0
            for inputs, labels in validation_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
                validation_loss += loss_fn(outputs, labels).item()

        accuracy = correct / total
        accuraciesOfValidation.append(accuracy)
```

```
            summary.add_scalar("Accuracy", accuracy, epoch)
            summary.add_scalar("Loss_Of_Validation", validation_loss/
↪len(validation_loader), epoch)
            validation_losses.append(validation_loss/len(validation_loader))

            print("\tEpoch {}, Validation Loss: {:.4f}, Validation Accuracy: {:.
↪4f}".format(epoch, validation_loss, accuracy))
        else:
            print("Epoch {}, Train Loss: {:.4f}, Train Accuracy: {:.4f}".
↪format(epoch, train_loss/len(train_loader), accuracy))
    end_time = time.time()
    trainTime = end_time - start_time

    trainModelComplexity = sum(p.numel() for p in model.parameters() if p.
↪requires_grad)

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(train_losses)
    plt.title('Train Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfTrain)
    plt.title('Train Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()

    plt.figure(figsize=(20, 8))
    plt.subplot(121)
    plt.plot(validation_losses)
    plt.title('Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()

    plt.subplot(122)
    plt.plot(accuraciesOfValidation)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show()
```

```
    # Load the TensorBoard notebook extension
    %load_ext tensorboard
    %tensorboard --logdir {TB_PATH}/{model.name}-train_batchnorm

    return trainTime, trainModelComplexity, validation_loss, accuracy


# requiert que les modules soient enregistrés dans une liste model.hidden_layers
def addWeightsHisto(writer, model, epoch):
    ix = 0
    for module in model.hidden_layers:
        if isinstance(module, nn.Linear):
            writer.add_histogram(f'Linear/{ix}/weight', module.weight, epoch)
            ix += 1
```

4 Déclaration des class

[4]:
```
class LinearMultiClass(nn.Module):
    def __init__(self, in_size, out_size, hidden_layers, final_activation=None,␣
 ↪activation=nn.Tanh()):
        super(LinearMultiClass, self).__init__()
        self.name = "Linear_Multi_Class"
        self.in_size = in_size
        self.hidden_layers = nn.ModuleList()
        self.output_layer = nn.Linear(hidden_layers[-1], out_size)
        self.final_activation = final_activation
        self.activation = activation

        for i, h in enumerate(hidden_layers):
            self.hidden_layers.append(nn.Linear(in_size if i == 0 else␣
 ↪hidden_layers[i-1], h))

    def forward(self, x):
        x = x.view(-1, self.in_size)
        for i, layer in enumerate(self.hidden_layers):
            x = layer(x)
            x = self.activation(x)

        x = self.output_layer(x)
        if self.final_activation is not None:
            x = self.final_activation(x)
        return x

class LinearMultiClassWithDropout(nn.Module):
```

```
    def __init__(self, in_size, out_size, hidden_layers, dropout_rate=0.5,␣
 ↪final_activation=None, activation=nn.Tanh()):
        super(LinearMultiClassWithDropout, self).__init__()
        self.name = "Linear_Multi_Class_With_Dropout"
        self.in_size = in_size
        self.hidden_layers = nn.ModuleList()
        self.dropout_layers = nn.ModuleList()
        self.dropout_rate = dropout_rate
        self.output_layer = nn.Linear(hidden_layers[-1], out_size)
        self.final_activation = final_activation
        self.activation = activation

        for i, h in enumerate(hidden_layers):
            self.hidden_layers.append(nn.Linear(in_size if i == 0 else␣
 ↪hidden_layers[i-1], h))
            self.dropout_layers.append(nn.Dropout(p=self.dropout_rate))

    def forward(self, x):
        x = x.view(-1, self.in_size)
        for i, layer in enumerate(self.hidden_layers):
            x = layer(x)
            x = self.activation(x)
            x = self.dropout_layers[i](x)

        x = self.output_layer(x)
        if self.final_activation is not None:
            x = self.final_activation(x)
        return x

class LinearMultiClassWithBatchNorm(nn.Module):
    def __init__(self, in_size, out_size, hidden_layers, final_activation=None,␣
 ↪activation=nn.Tanh()):
        super(LinearMultiClassWithBatchNorm, self).__init__()
        self.name = "Linear_Multi_Class_With_BatchNorm"
        self.in_size = in_size
        self.hidden_layers = nn.ModuleList()
        self.batch_norm_layers = nn.ModuleList()
        self.output_layer = nn.Linear(hidden_layers[-1], out_size)
        self.final_activation = final_activation
        self.activation = activation

        for i, h in enumerate(hidden_layers):
            self.hidden_layers.append(nn.Linear(in_size if i == 0 else␣
 ↪hidden_layers[i-1], h))
            self.batch_norm_layers.append(nn.BatchNorm1d(h))

    def forward(self, x):
```

```
        x = x.view(-1, self.in_size)
        for i, layer in enumerate(self.hidden_layers):
            x = layer(x)
            x = self.batch_norm_layers[i](x)
            x = self.activation(x)

        x = self.output_layer(x)
        if self.final_activation is not None:
            x = self.final_activation(x)
        return x
```

5 Déclaration des données du projet

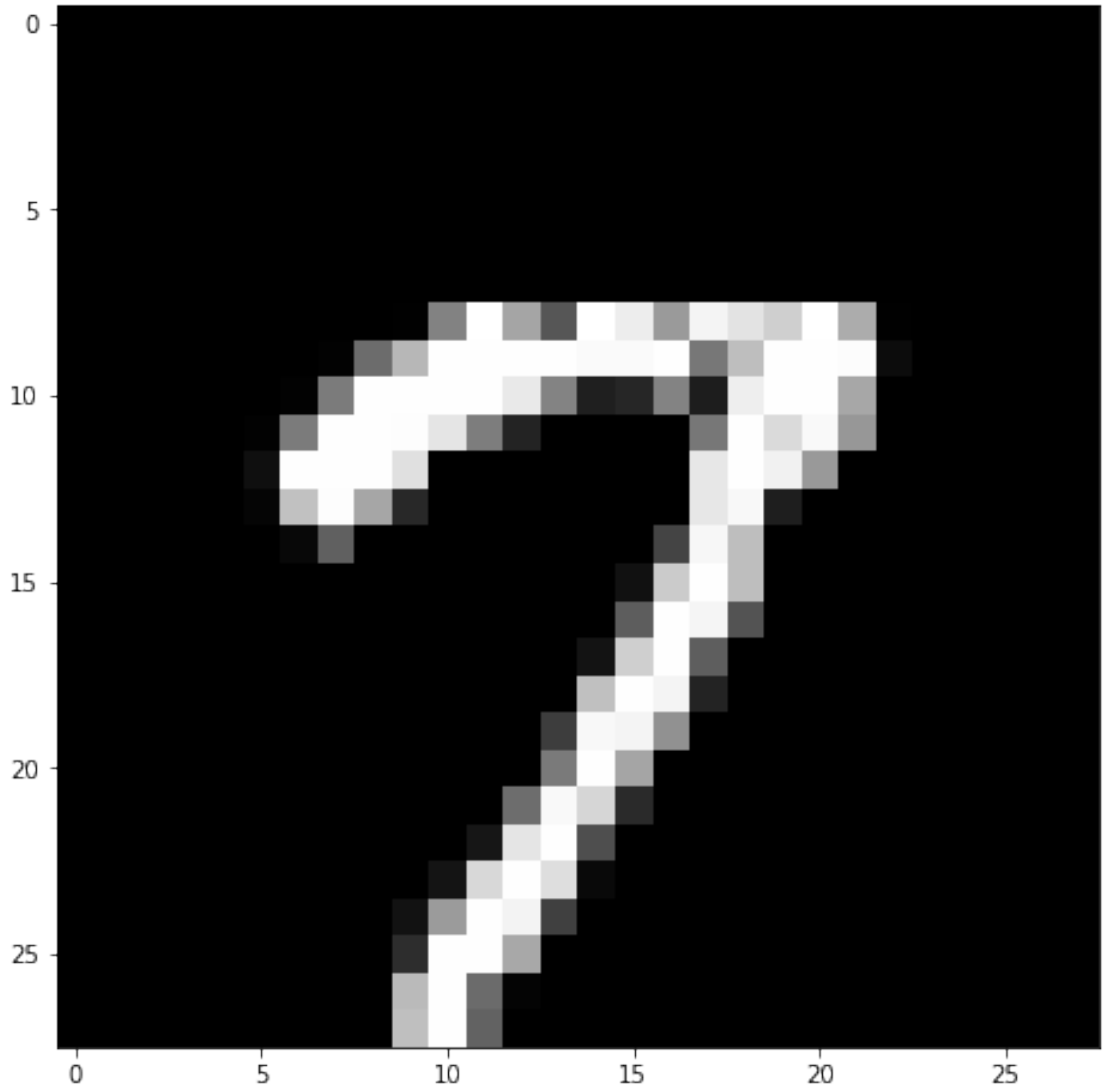
[5]:
```
#Transformations à appliquer sur le dataset (transformation des images en␣
 ↪tenseurs et normalization pour obtenir des valeurs entre -1 et 1)
mean = [0.5]
std = [0.5]
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

# Téléchargement des données (via le dataset specifique MNIST de pytorch)
mnist_train = MNIST('./data', train=True, transform=transform, download=True)
mnist_test = MNIST('./data', train=False, transform=transform, download=True)

batch_size = 64
train_loader = DataLoader(mnist_train, batch_size=batch_size, shuffle=True)
validation_loader = DataLoader(mnist_test, batch_size=batch_size)

TB_PATH = f"/tmp/logs/deepLearning"
epoch = 100

in_size = 784
out_size = 10
hidden_layers = [100, 100, 100]
```

6 Récuperation d'une image de notre batch de données

[6]:
```
imgs,labs = next(iter(train_loader))

# dimension of images (flattened)reload_ext tensorboard
HEIGHT,WIDTH = imgs.shape[2], imgs.shape[3] # taille de l'image
INPUT_DIM = HEIGHT * WIDTH
```

```
#Visualisation de la première image
img = unnormalize(imgs[0], mean, std) # pour retrouver l'image d'origine (avant
↪normalisation)
fig = plt.figure(figsize=(8, 8))

plt.imshow(img.squeeze(),cmap='Greys_r')
```

[6]: <matplotlib.image.AxesImage at 0x7fe597c82820>



7 Construction et Entrainement des modèles

7.1 Modèle LinearMultiClass

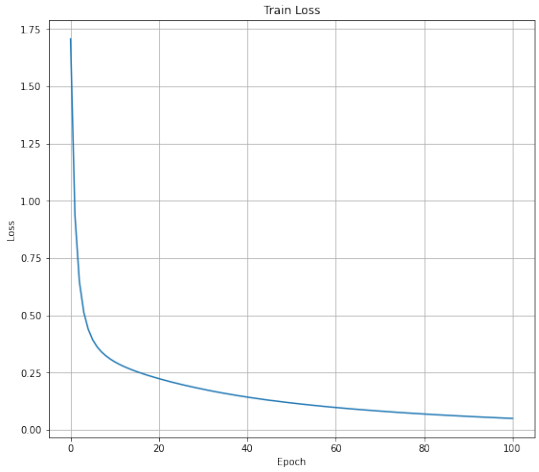
7.1.1 Avec activation = nn.Tanh et Optimiseur = Adam

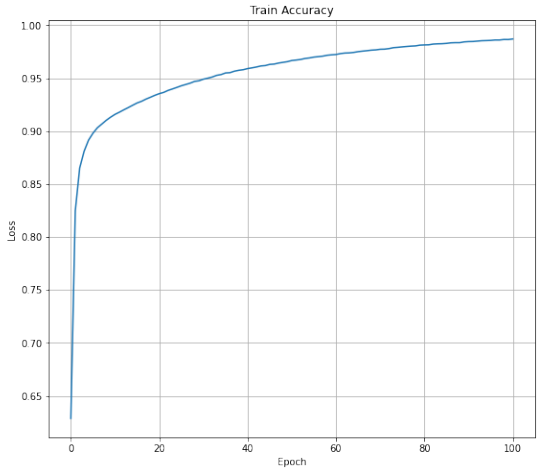
[7]:
```
# Définir le modèle, la fonction de coût, et l'optimiseur
model = LinearMultiClass(in_size=784, out_size=10, hidden_layers=[256, 128],␣
 ↪activation=nn.Tanh())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=1e-5)

# Entraîner le modèle
model1ExecutionTimeTanhAdam, model1ComplexityTanhAdam,␣
 ↪model1ValidationLossTanhAdam, model1AccuracyTanhAdam = train(model,␣
 ↪train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="Tanh-Adam")
```

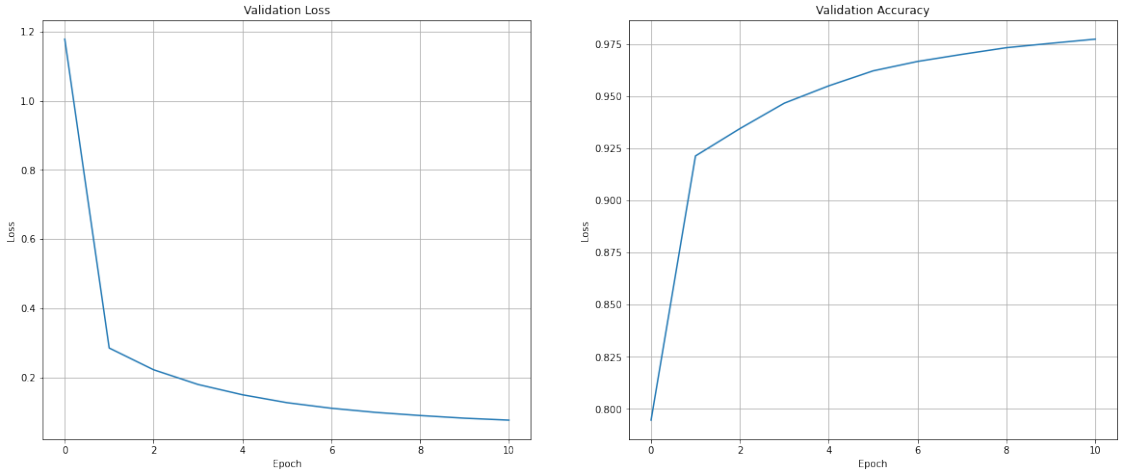
```
        Epoch 0, Validation Loss: 184.9282, Validation Accuracy: 0.7946
Epoch 1, Train Loss: 0.9363, Train Accuracy: 0.8247
Epoch 2, Train Loss: 0.6473, Train Accuracy: 0.8652
Epoch 3, Train Loss: 0.5131, Train Accuracy: 0.8809
Epoch 4, Train Loss: 0.4396, Train Accuracy: 0.8914
Epoch 5, Train Loss: 0.3942, Train Accuracy: 0.8980
Epoch 6, Train Loss: 0.3634, Train Accuracy: 0.9032
Epoch 7, Train Loss: 0.3410, Train Accuracy: 0.9066
Epoch 8, Train Loss: 0.3237, Train Accuracy: 0.9102
Epoch 9, Train Loss: 0.3094, Train Accuracy: 0.9132
        Epoch 10, Validation Loss: 44.7343, Validation Accuracy: 0.9213
Epoch 11, Train Loss: 0.2871, Train Accuracy: 0.9179
Epoch 12, Train Loss: 0.2779, Train Accuracy: 0.9202
Epoch 13, Train Loss: 0.2696, Train Accuracy: 0.9223
Epoch 14, Train Loss: 0.2620, Train Accuracy: 0.9244
Epoch 15, Train Loss: 0.2547, Train Accuracy: 0.9266
Epoch 16, Train Loss: 0.2482, Train Accuracy: 0.9282
Epoch 17, Train Loss: 0.2416, Train Accuracy: 0.9302
Epoch 18, Train Loss: 0.2357, Train Accuracy: 0.9320
Epoch 19, Train Loss: 0.2299, Train Accuracy: 0.9338
        Epoch 20, Validation Loss: 34.8654, Validation Accuracy: 0.9344
Epoch 21, Train Loss: 0.2189, Train Accuracy: 0.9366
Epoch 22, Train Loss: 0.2137, Train Accuracy: 0.9385
Epoch 23, Train Loss: 0.2085, Train Accuracy: 0.9398
Epoch 24, Train Loss: 0.2039, Train Accuracy: 0.9413
Epoch 25, Train Loss: 0.1991, Train Accuracy: 0.9428
Epoch 26, Train Loss: 0.1947, Train Accuracy: 0.9440
Epoch 27, Train Loss: 0.1904, Train Accuracy: 0.9453
Epoch 28, Train Loss: 0.1861, Train Accuracy: 0.9469
Epoch 29, Train Loss: 0.1820, Train Accuracy: 0.9475
        Epoch 30, Validation Loss: 28.1942, Validation Accuracy: 0.9466
```

```
Epoch 31, Train Loss: 0.1740, Train Accuracy: 0.9500
Epoch 32, Train Loss: 0.1703, Train Accuracy: 0.9511
Epoch 33, Train Loss: 0.1665, Train Accuracy: 0.9527
Epoch 34, Train Loss: 0.1630, Train Accuracy: 0.9534
Epoch 35, Train Loss: 0.1595, Train Accuracy: 0.9549
Epoch 36, Train Loss: 0.1563, Train Accuracy: 0.9552
Epoch 37, Train Loss: 0.1530, Train Accuracy: 0.9566
Epoch 38, Train Loss: 0.1497, Train Accuracy: 0.9574
Epoch 39, Train Loss: 0.1469, Train Accuracy: 0.9580
        Epoch 40, Validation Loss: 23.5236, Validation Accuracy: 0.9549
Epoch 41, Train Loss: 0.1409, Train Accuracy: 0.9598
Epoch 42, Train Loss: 0.1381, Train Accuracy: 0.9606
Epoch 43, Train Loss: 0.1354, Train Accuracy: 0.9616
Epoch 44, Train Loss: 0.1325, Train Accuracy: 0.9619
Epoch 45, Train Loss: 0.1299, Train Accuracy: 0.9630
Epoch 46, Train Loss: 0.1276, Train Accuracy: 0.9633
Epoch 47, Train Loss: 0.1253, Train Accuracy: 0.9643
Epoch 48, Train Loss: 0.1227, Train Accuracy: 0.9649
Epoch 49, Train Loss: 0.1204, Train Accuracy: 0.9656
        Epoch 50, Validation Loss: 19.9363, Validation Accuracy: 0.9621
Epoch 51, Train Loss: 0.1160, Train Accuracy: 0.9671
Epoch 52, Train Loss: 0.1136, Train Accuracy: 0.9677
Epoch 53, Train Loss: 0.1117, Train Accuracy: 0.9686
Epoch 54, Train Loss: 0.1096, Train Accuracy: 0.9691
Epoch 55, Train Loss: 0.1075, Train Accuracy: 0.9699
Epoch 56, Train Loss: 0.1057, Train Accuracy: 0.9703
Epoch 57, Train Loss: 0.1037, Train Accuracy: 0.9708
Epoch 58, Train Loss: 0.1018, Train Accuracy: 0.9716
Epoch 59, Train Loss: 0.0999, Train Accuracy: 0.9720
        Epoch 60, Validation Loss: 17.4019, Validation Accuracy: 0.9666
Epoch 61, Train Loss: 0.0965, Train Accuracy: 0.9732
Epoch 62, Train Loss: 0.0948, Train Accuracy: 0.9737
Epoch 63, Train Loss: 0.0932, Train Accuracy: 0.9739
Epoch 64, Train Loss: 0.0915, Train Accuracy: 0.9743
Epoch 65, Train Loss: 0.0899, Train Accuracy: 0.9750
Epoch 66, Train Loss: 0.0884, Train Accuracy: 0.9755
Epoch 67, Train Loss: 0.0869, Train Accuracy: 0.9759
Epoch 68, Train Loss: 0.0855, Train Accuracy: 0.9765
Epoch 69, Train Loss: 0.0840, Train Accuracy: 0.9767
        Epoch 70, Validation Loss: 15.5233, Validation Accuracy: 0.9700
Epoch 71, Train Loss: 0.0813, Train Accuracy: 0.9774
Epoch 72, Train Loss: 0.0798, Train Accuracy: 0.9779
Epoch 73, Train Loss: 0.0785, Train Accuracy: 0.9788
Epoch 74, Train Loss: 0.0771, Train Accuracy: 0.9791
Epoch 75, Train Loss: 0.0758, Train Accuracy: 0.9796
Epoch 76, Train Loss: 0.0746, Train Accuracy: 0.9799
Epoch 77, Train Loss: 0.0734, Train Accuracy: 0.9803
Epoch 78, Train Loss: 0.0722, Train Accuracy: 0.9805
```

```
Epoch 79, Train Loss: 0.0710, Train Accuracy: 0.9812
       Epoch 80, Validation Loss: 14.1010, Validation Accuracy: 0.9732
Epoch 81, Train Loss: 0.0688, Train Accuracy: 0.9815
Epoch 82, Train Loss: 0.0675, Train Accuracy: 0.9822
Epoch 83, Train Loss: 0.0664, Train Accuracy: 0.9825
Epoch 84, Train Loss: 0.0655, Train Accuracy: 0.9826
Epoch 85, Train Loss: 0.0644, Train Accuracy: 0.9829
Epoch 86, Train Loss: 0.0634, Train Accuracy: 0.9834
Epoch 87, Train Loss: 0.0624, Train Accuracy: 0.9836
Epoch 88, Train Loss: 0.0613, Train Accuracy: 0.9836
Epoch 89, Train Loss: 0.0605, Train Accuracy: 0.9843
       Epoch 90, Validation Loss: 12.8877, Validation Accuracy: 0.9753
Epoch 91, Train Loss: 0.0584, Train Accuracy: 0.9847
Epoch 92, Train Loss: 0.0576, Train Accuracy: 0.9850
Epoch 93, Train Loss: 0.0566, Train Accuracy: 0.9854
Epoch 94, Train Loss: 0.0557, Train Accuracy: 0.9855
Epoch 95, Train Loss: 0.0549, Train Accuracy: 0.9857
Epoch 96, Train Loss: 0.0540, Train Accuracy: 0.9861
Epoch 97, Train Loss: 0.0531, Train Accuracy: 0.9860
Epoch 98, Train Loss: 0.0522, Train Accuracy: 0.9866
Epoch 99, Train Loss: 0.0515, Train Accuracy: 0.9865
       Epoch 100, Validation Loss: 12.0054, Validation Accuracy: 0.9773
```








```
<IPython.core.display.HTML object>
```

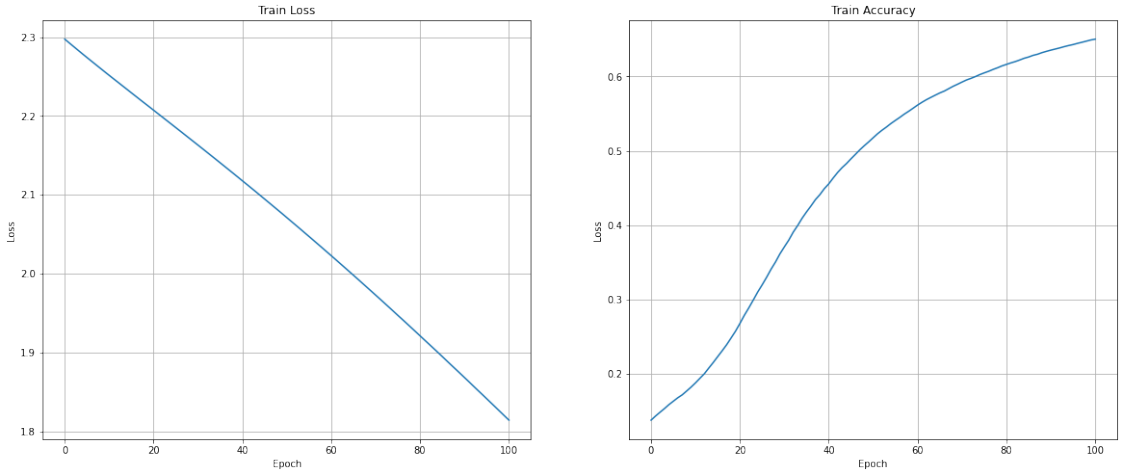
7.1.2 Avec activation = nn.Tanh et Optimiseur = SGD

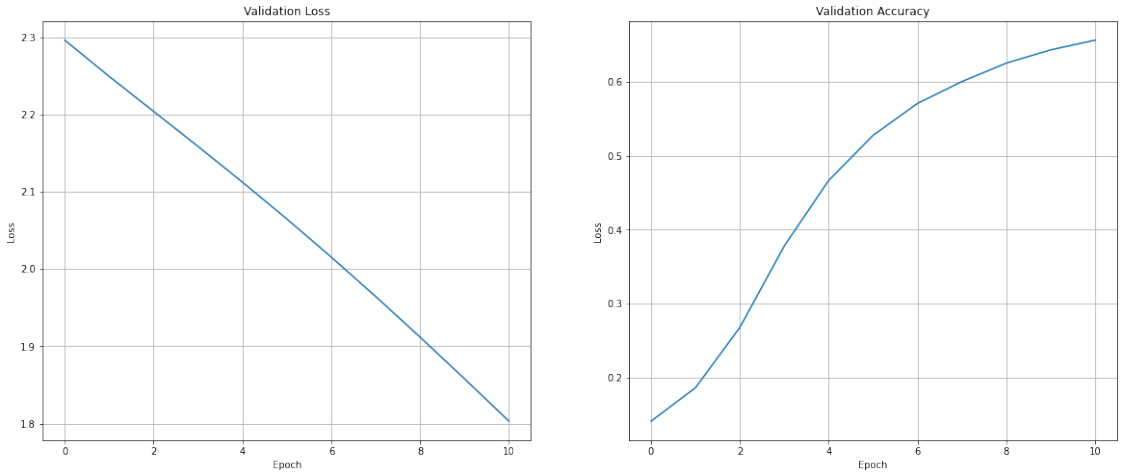
[8]:
```
# Définir le modèle, la fonction de coût, et l'optimiseur
model = LinearMultiClass(in_size=784, out_size=10, hidden_layers=[256, 128],␣
 ↪activation=nn.Tanh())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model.parameters(),lr=1e-5)

# Entraîner le modèle
model1ExecutionTimeTanhSGD, model1ComplexityTanhSGD,␣
 ↪model1ValidationLossTanhSGD, model1AccuracyTanhSGD = train(model,␣
 ↪train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="Tanh-SGD")
```

```
        Epoch 0, Validation Loss: 360.4930, Validation Accuracy: 0.1414
Epoch 1, Train Loss: 2.2926, Train Accuracy: 0.1433
Epoch 2, Train Loss: 2.2879, Train Accuracy: 0.1483
Epoch 3, Train Loss: 2.2833, Train Accuracy: 0.1532
Epoch 4, Train Loss: 2.2786, Train Accuracy: 0.1585
Epoch 5, Train Loss: 2.2740, Train Accuracy: 0.1632
Epoch 6, Train Loss: 2.2695, Train Accuracy: 0.1678
Epoch 7, Train Loss: 2.2649, Train Accuracy: 0.1718
Epoch 8, Train Loss: 2.2604, Train Accuracy: 0.1769
Epoch 9, Train Loss: 2.2559, Train Accuracy: 0.1822
        Epoch 10, Validation Loss: 353.1221, Validation Accuracy: 0.1865
Epoch 11, Train Loss: 2.2470, Train Accuracy: 0.1941
Epoch 12, Train Loss: 2.2426, Train Accuracy: 0.2002
Epoch 13, Train Loss: 2.2381, Train Accuracy: 0.2079
Epoch 14, Train Loss: 2.2337, Train Accuracy: 0.2153
Epoch 15, Train Loss: 2.2292, Train Accuracy: 0.2232
```

```
Epoch 16, Train Loss: 2.2248, Train Accuracy: 0.2310
Epoch 17, Train Loss: 2.2204, Train Accuracy: 0.2391
Epoch 18, Train Loss: 2.2160, Train Accuracy: 0.2482
Epoch 19, Train Loss: 2.2116, Train Accuracy: 0.2573
        Epoch 20, Validation Loss: 346.0155, Validation Accuracy: 0.2681
Epoch 21, Train Loss: 2.2028, Train Accuracy: 0.2788
Epoch 22, Train Loss: 2.1984, Train Accuracy: 0.2888
Epoch 23, Train Loss: 2.1940, Train Accuracy: 0.2992
Epoch 24, Train Loss: 2.1896, Train Accuracy: 0.3100
Epoch 25, Train Loss: 2.1851, Train Accuracy: 0.3196
Epoch 26, Train Loss: 2.1807, Train Accuracy: 0.3298
Epoch 27, Train Loss: 2.1763, Train Accuracy: 0.3406
Epoch 28, Train Loss: 2.1718, Train Accuracy: 0.3503
Epoch 29, Train Loss: 2.1674, Train Accuracy: 0.3613
        Epoch 30, Validation Loss: 338.9022, Validation Accuracy: 0.3781
Epoch 31, Train Loss: 2.1585, Train Accuracy: 0.3799
Epoch 32, Train Loss: 2.1540, Train Accuracy: 0.3906
Epoch 33, Train Loss: 2.1495, Train Accuracy: 0.3995
Epoch 34, Train Loss: 2.1450, Train Accuracy: 0.4093
Epoch 35, Train Loss: 2.1404, Train Accuracy: 0.4178
Epoch 36, Train Loss: 2.1359, Train Accuracy: 0.4258
Epoch 37, Train Loss: 2.1314, Train Accuracy: 0.4343
Epoch 38, Train Loss: 2.1268, Train Accuracy: 0.4410
Epoch 39, Train Loss: 2.1223, Train Accuracy: 0.4490
        Epoch 40, Validation Loss: 331.6416, Validation Accuracy: 0.4666
Epoch 41, Train Loss: 2.1131, Train Accuracy: 0.4634
Epoch 42, Train Loss: 2.1085, Train Accuracy: 0.4708
Epoch 43, Train Loss: 2.1039, Train Accuracy: 0.4773
Epoch 44, Train Loss: 2.0992, Train Accuracy: 0.4829
Epoch 45, Train Loss: 2.0945, Train Accuracy: 0.4892
Epoch 46, Train Loss: 2.0899, Train Accuracy: 0.4952
Epoch 47, Train Loss: 2.0852, Train Accuracy: 0.5014
Epoch 48, Train Loss: 2.0805, Train Accuracy: 0.5069
Epoch 49, Train Loss: 2.0758, Train Accuracy: 0.5121
        Epoch 50, Validation Loss: 324.1569, Validation Accuracy: 0.5275
Epoch 51, Train Loss: 2.0662, Train Accuracy: 0.5230
Epoch 52, Train Loss: 2.0615, Train Accuracy: 0.5277
Epoch 53, Train Loss: 2.0567, Train Accuracy: 0.5321
Epoch 54, Train Loss: 2.0518, Train Accuracy: 0.5367
Epoch 55, Train Loss: 2.0470, Train Accuracy: 0.5409
Epoch 56, Train Loss: 2.0422, Train Accuracy: 0.5450
Epoch 57, Train Loss: 2.0373, Train Accuracy: 0.5493
Epoch 58, Train Loss: 2.0325, Train Accuracy: 0.5532
Epoch 59, Train Loss: 2.0276, Train Accuracy: 0.5573
        Epoch 60, Validation Loss: 316.4122, Validation Accuracy: 0.5706
Epoch 61, Train Loss: 2.0178, Train Accuracy: 0.5653
Epoch 62, Train Loss: 2.0128, Train Accuracy: 0.5688
Epoch 63, Train Loss: 2.0078, Train Accuracy: 0.5719
```

```
Epoch 64, Train Loss: 2.0029, Train Accuracy: 0.5749
Epoch 65, Train Loss: 1.9979, Train Accuracy: 0.5778
Epoch 66, Train Loss: 1.9929, Train Accuracy: 0.5803
Epoch 67, Train Loss: 1.9878, Train Accuracy: 0.5835
Epoch 68, Train Loss: 1.9828, Train Accuracy: 0.5868
Epoch 69, Train Loss: 1.9777, Train Accuracy: 0.5895
        Epoch 70, Validation Loss: 308.4039, Validation Accuracy: 0.6001
Epoch 71, Train Loss: 1.9676, Train Accuracy: 0.5952
Epoch 72, Train Loss: 1.9625, Train Accuracy: 0.5974
Epoch 73, Train Loss: 1.9573, Train Accuracy: 0.5998
Epoch 74, Train Loss: 1.9522, Train Accuracy: 0.6025
Epoch 75, Train Loss: 1.9470, Train Accuracy: 0.6048
Epoch 76, Train Loss: 1.9419, Train Accuracy: 0.6069
Epoch 77, Train Loss: 1.9367, Train Accuracy: 0.6095
Epoch 78, Train Loss: 1.9315, Train Accuracy: 0.6117
Epoch 79, Train Loss: 1.9263, Train Accuracy: 0.6142
        Epoch 80, Validation Loss: 300.1526, Validation Accuracy: 0.6250
Epoch 81, Train Loss: 1.9158, Train Accuracy: 0.6182
Epoch 82, Train Loss: 1.9106, Train Accuracy: 0.6200
Epoch 83, Train Loss: 1.9053, Train Accuracy: 0.6222
Epoch 84, Train Loss: 1.9001, Train Accuracy: 0.6245
Epoch 85, Train Loss: 1.8948, Train Accuracy: 0.6262
Epoch 86, Train Loss: 1.8894, Train Accuracy: 0.6284
Epoch 87, Train Loss: 1.8841, Train Accuracy: 0.6299
Epoch 88, Train Loss: 1.8788, Train Accuracy: 0.6320
Epoch 89, Train Loss: 1.8735, Train Accuracy: 0.6338
        Epoch 90, Validation Loss: 291.6974, Validation Accuracy: 0.6429
Epoch 91, Train Loss: 1.8628, Train Accuracy: 0.6369
Epoch 92, Train Loss: 1.8574, Train Accuracy: 0.6384
Epoch 93, Train Loss: 1.8520, Train Accuracy: 0.6401
Epoch 94, Train Loss: 1.8467, Train Accuracy: 0.6416
Epoch 95, Train Loss: 1.8413, Train Accuracy: 0.6429
Epoch 96, Train Loss: 1.8359, Train Accuracy: 0.6445
Epoch 97, Train Loss: 1.8305, Train Accuracy: 0.6460
Epoch 98, Train Loss: 1.8250, Train Accuracy: 0.6475
Epoch 99, Train Loss: 1.8196, Train Accuracy: 0.6491
        Epoch 100, Validation Loss: 283.0912, Validation Accuracy: 0.6560
```






```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 0:15:50 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

7.1.3 Avec activation = nn.ReLU et Optimiseur = Adam

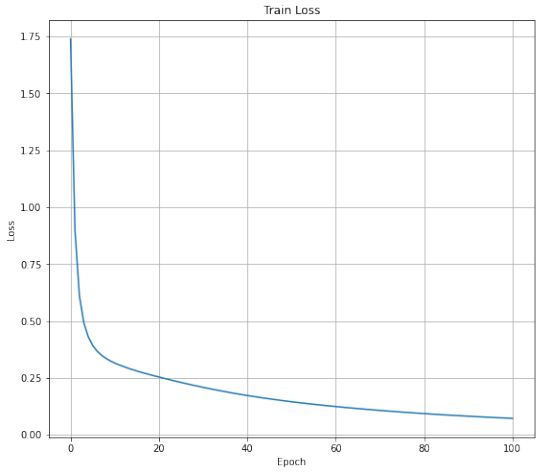
[9]:
```
# Définir le modèle, la fonction de coût, et l'optimiseur
model = LinearMultiClass(in_size=784, out_size=10, hidden_layers=[256, 128],␣
↪activation=nn.ReLU())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=1e-5)
```

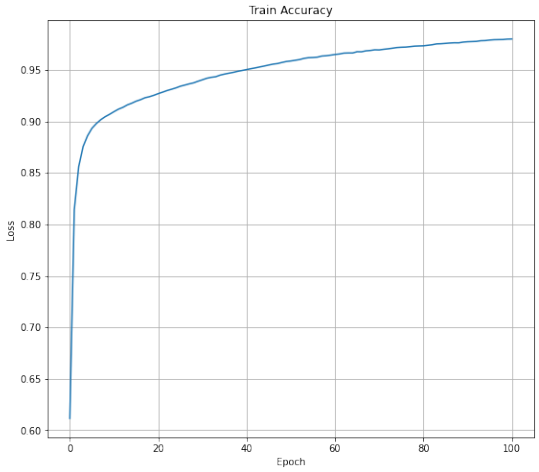
```
# Entraîner le modèle
model1ExecutionTimeTanhReLuAdam, model1ComplexityTanhReLuAdam,␣
↪model1ValidationLossTanhReLuAdam, model1AccuracyTanhReLuAdam = train(model,␣
↪train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
↪typeTrain="ReLU-Adam")
```

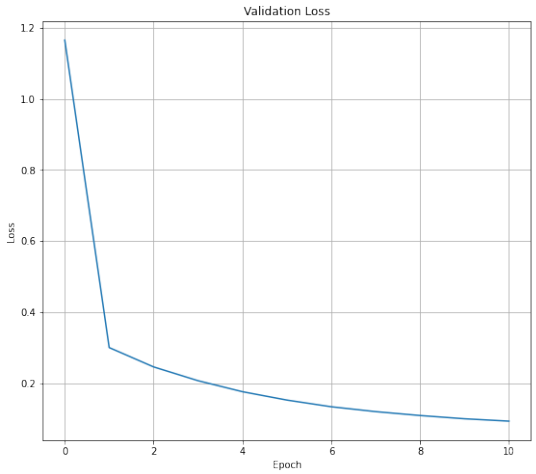
```
        Epoch 0, Validation Loss: 183.0248, Validation Accuracy: 0.7917
Epoch 1, Train Loss: 0.9010, Train Accuracy: 0.8141
Epoch 2, Train Loss: 0.6119, Train Accuracy: 0.8560
Epoch 3, Train Loss: 0.4927, Train Accuracy: 0.8755
Epoch 4, Train Loss: 0.4308, Train Accuracy: 0.8860
Epoch 5, Train Loss: 0.3931, Train Accuracy: 0.8933
Epoch 6, Train Loss: 0.3679, Train Accuracy: 0.8979
Epoch 7, Train Loss: 0.3500, Train Accuracy: 0.9017
Epoch 8, Train Loss: 0.3362, Train Accuracy: 0.9046
Epoch 9, Train Loss: 0.3248, Train Accuracy: 0.9069
        Epoch 10, Validation Loss: 47.1091, Validation Accuracy: 0.9140
Epoch 11, Train Loss: 0.3072, Train Accuracy: 0.9119
Epoch 12, Train Loss: 0.3001, Train Accuracy: 0.9136
Epoch 13, Train Loss: 0.2930, Train Accuracy: 0.9160
Epoch 14, Train Loss: 0.2866, Train Accuracy: 0.9176
Epoch 15, Train Loss: 0.2806, Train Accuracy: 0.9196
Epoch 16, Train Loss: 0.2748, Train Accuracy: 0.9211
Epoch 17, Train Loss: 0.2694, Train Accuracy: 0.9230
Epoch 18, Train Loss: 0.2641, Train Accuracy: 0.9240
Epoch 19, Train Loss: 0.2588, Train Accuracy: 0.9253
        Epoch 20, Validation Loss: 38.5710, Validation Accuracy: 0.9292
Epoch 21, Train Loss: 0.2489, Train Accuracy: 0.9284
Epoch 22, Train Loss: 0.2444, Train Accuracy: 0.9300
Epoch 23, Train Loss: 0.2393, Train Accuracy: 0.9312
Epoch 24, Train Loss: 0.2347, Train Accuracy: 0.9325
Epoch 25, Train Loss: 0.2301, Train Accuracy: 0.9342
Epoch 26, Train Loss: 0.2257, Train Accuracy: 0.9353
Epoch 27, Train Loss: 0.2214, Train Accuracy: 0.9364
Epoch 28, Train Loss: 0.2169, Train Accuracy: 0.9374
Epoch 29, Train Loss: 0.2130, Train Accuracy: 0.9390
        Epoch 30, Validation Loss: 32.4683, Validation Accuracy: 0.9405
Epoch 31, Train Loss: 0.2048, Train Accuracy: 0.9419
Epoch 32, Train Loss: 0.2007, Train Accuracy: 0.9428
Epoch 33, Train Loss: 0.1970, Train Accuracy: 0.9434
Epoch 34, Train Loss: 0.1932, Train Accuracy: 0.9449
Epoch 35, Train Loss: 0.1898, Train Accuracy: 0.9459
Epoch 36, Train Loss: 0.1861, Train Accuracy: 0.9468
Epoch 37, Train Loss: 0.1825, Train Accuracy: 0.9476
Epoch 38, Train Loss: 0.1793, Train Accuracy: 0.9486
Epoch 39, Train Loss: 0.1760, Train Accuracy: 0.9495
        Epoch 40, Validation Loss: 27.6140, Validation Accuracy: 0.9482
Epoch 41, Train Loss: 0.1699, Train Accuracy: 0.9512
```

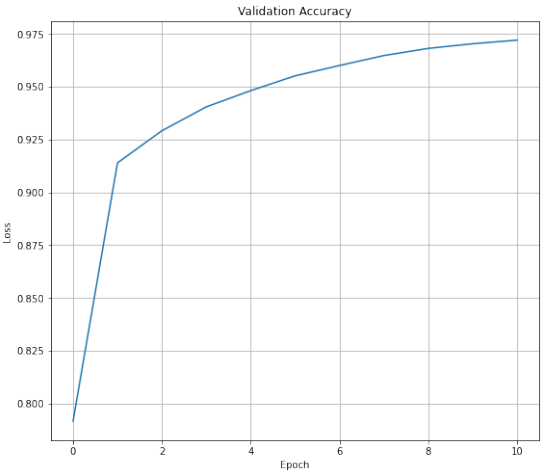
```
Epoch 42, Train Loss: 0.1670, Train Accuracy: 0.9520
Epoch 43, Train Loss: 0.1639, Train Accuracy: 0.9529
Epoch 44, Train Loss: 0.1611, Train Accuracy: 0.9538
Epoch 45, Train Loss: 0.1583, Train Accuracy: 0.9547
Epoch 46, Train Loss: 0.1555, Train Accuracy: 0.9556
Epoch 47, Train Loss: 0.1531, Train Accuracy: 0.9562
Epoch 48, Train Loss: 0.1504, Train Accuracy: 0.9573
Epoch 49, Train Loss: 0.1479, Train Accuracy: 0.9582
        Epoch 50, Validation Loss: 23.9062, Validation Accuracy: 0.9552
Epoch 51, Train Loss: 0.1431, Train Accuracy: 0.9594
Epoch 52, Train Loss: 0.1409, Train Accuracy: 0.9601
Epoch 53, Train Loss: 0.1386, Train Accuracy: 0.9612
Epoch 54, Train Loss: 0.1364, Train Accuracy: 0.9619
Epoch 55, Train Loss: 0.1343, Train Accuracy: 0.9620
Epoch 56, Train Loss: 0.1324, Train Accuracy: 0.9624
Epoch 57, Train Loss: 0.1301, Train Accuracy: 0.9635
Epoch 58, Train Loss: 0.1281, Train Accuracy: 0.9637
Epoch 59, Train Loss: 0.1263, Train Accuracy: 0.9643
        Epoch 60, Validation Loss: 20.9484, Validation Accuracy: 0.9601
Epoch 61, Train Loss: 0.1225, Train Accuracy: 0.9655
Epoch 62, Train Loss: 0.1205, Train Accuracy: 0.9663
Epoch 63, Train Loss: 0.1188, Train Accuracy: 0.9664
Epoch 64, Train Loss: 0.1172, Train Accuracy: 0.9664
Epoch 65, Train Loss: 0.1153, Train Accuracy: 0.9677
Epoch 66, Train Loss: 0.1138, Train Accuracy: 0.9676
Epoch 67, Train Loss: 0.1119, Train Accuracy: 0.9685
Epoch 68, Train Loss: 0.1106, Train Accuracy: 0.9688
Epoch 69, Train Loss: 0.1088, Train Accuracy: 0.9695
        Epoch 70, Validation Loss: 18.8004, Validation Accuracy: 0.9648
Epoch 71, Train Loss: 0.1059, Train Accuracy: 0.9700
Epoch 72, Train Loss: 0.1044, Train Accuracy: 0.9705
Epoch 73, Train Loss: 0.1029, Train Accuracy: 0.9710
Epoch 74, Train Loss: 0.1015, Train Accuracy: 0.9717
Epoch 75, Train Loss: 0.0999, Train Accuracy: 0.9720
Epoch 76, Train Loss: 0.0986, Train Accuracy: 0.9721
Epoch 77, Train Loss: 0.0973, Train Accuracy: 0.9726
Epoch 78, Train Loss: 0.0960, Train Accuracy: 0.9731
Epoch 79, Train Loss: 0.0947, Train Accuracy: 0.9732
        Epoch 80, Validation Loss: 17.0837, Validation Accuracy: 0.9682
Epoch 81, Train Loss: 0.0921, Train Accuracy: 0.9739
Epoch 82, Train Loss: 0.0910, Train Accuracy: 0.9744
Epoch 83, Train Loss: 0.0897, Train Accuracy: 0.9753
Epoch 84, Train Loss: 0.0885, Train Accuracy: 0.9754
Epoch 85, Train Loss: 0.0875, Train Accuracy: 0.9758
Epoch 86, Train Loss: 0.0863, Train Accuracy: 0.9761
Epoch 87, Train Loss: 0.0852, Train Accuracy: 0.9764
Epoch 88, Train Loss: 0.0842, Train Accuracy: 0.9763
Epoch 89, Train Loss: 0.0832, Train Accuracy: 0.9770
```

```
        Epoch 90, Validation Loss: 15.6461, Validation Accuracy: 0.9704
Epoch 91, Train Loss: 0.0810, Train Accuracy: 0.9775
Epoch 92, Train Loss: 0.0800, Train Accuracy: 0.9777
Epoch 93, Train Loss: 0.0790, Train Accuracy: 0.9784
Epoch 94, Train Loss: 0.0780, Train Accuracy: 0.9785
Epoch 95, Train Loss: 0.0770, Train Accuracy: 0.9790
Epoch 96, Train Loss: 0.0760, Train Accuracy: 0.9793
Epoch 97, Train Loss: 0.0751, Train Accuracy: 0.9794
Epoch 98, Train Loss: 0.0743, Train Accuracy: 0.9795
Epoch 99, Train Loss: 0.0734, Train Accuracy: 0.9799
        Epoch 100, Validation Loss: 14.5963, Validation Accuracy: 0.9721
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 0:32:16 ago. (Use '!kill␣
 ↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

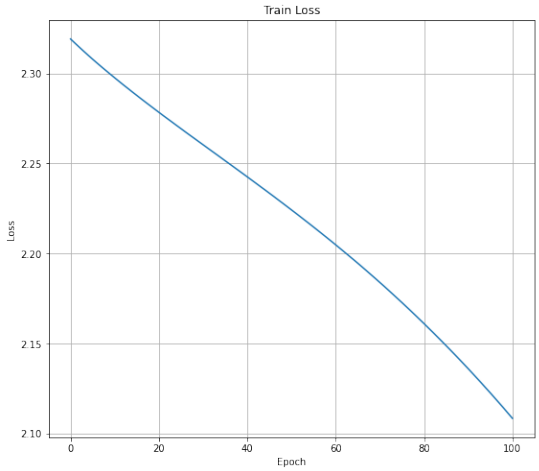
7.1.4 Avec activation = nn.ReLU et Optimiseur = SGD

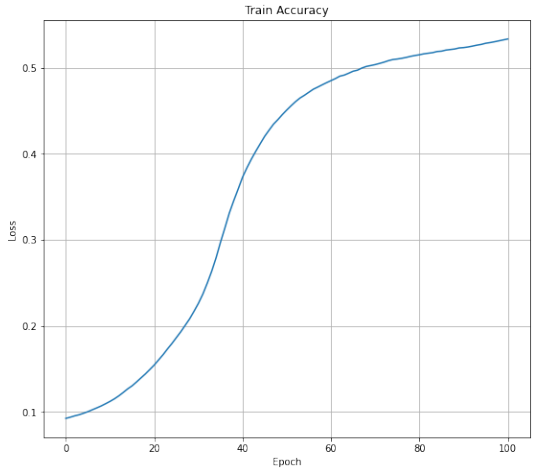
[10]:
```
# Définir le modèle, la fonction de coût, et l'optimiseur
model = LinearMultiClass(in_size=784, out_size=10, hidden_layers=[256, 128],␣
 ↪activation=nn.ReLU())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model.parameters(),lr=1e-5)

# Entraîner le modèle
model1ExecutionTimeTanhReLuSGD, model1ComplexityTanhReLuSGD,␣
 ↪model1ValidationLossTanhReLuSGD, model1AccuracyTanhReLuSGD = train(model,␣
 ↪train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="ReLU-SGD")
```

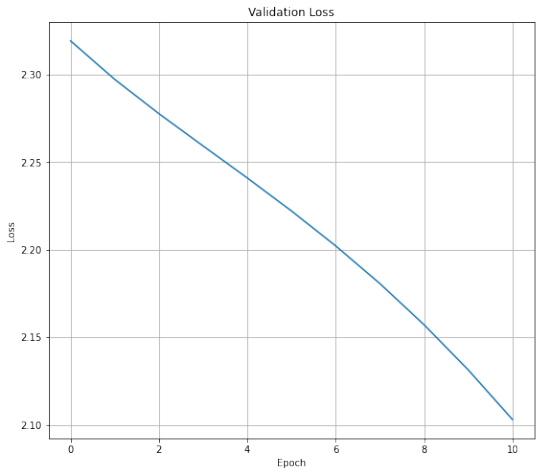
```
        Epoch 0, Validation Loss: 364.1128, Validation Accuracy: 0.0871
Epoch 1, Train Loss: 2.3168, Train Accuracy: 0.0939
Epoch 2, Train Loss: 2.3145, Train Accuracy: 0.0955
Epoch 3, Train Loss: 2.3122, Train Accuracy: 0.0967
Epoch 4, Train Loss: 2.3100, Train Accuracy: 0.0984
Epoch 5, Train Loss: 2.3079, Train Accuracy: 0.1003
Epoch 6, Train Loss: 2.3058, Train Accuracy: 0.1025
Epoch 7, Train Loss: 2.3036, Train Accuracy: 0.1047
Epoch 8, Train Loss: 2.3016, Train Accuracy: 0.1070
Epoch 9, Train Loss: 2.2995, Train Accuracy: 0.1095
        Epoch 10, Validation Loss: 360.6604, Validation Accuracy: 0.1086
Epoch 11, Train Loss: 2.2955, Train Accuracy: 0.1153
Epoch 12, Train Loss: 2.2936, Train Accuracy: 0.1188
Epoch 13, Train Loss: 2.2916, Train Accuracy: 0.1227
Epoch 14, Train Loss: 2.2897, Train Accuracy: 0.1268
Epoch 15, Train Loss: 2.2878, Train Accuracy: 0.1304
Epoch 16, Train Loss: 2.2859, Train Accuracy: 0.1348
Epoch 17, Train Loss: 2.2840, Train Accuracy: 0.1396
Epoch 18, Train Loss: 2.2821, Train Accuracy: 0.1442
Epoch 19, Train Loss: 2.2802, Train Accuracy: 0.1493
        Epoch 20, Validation Loss: 357.5888, Validation Accuracy: 0.1516
Epoch 21, Train Loss: 2.2766, Train Accuracy: 0.1605
Epoch 22, Train Loss: 2.2747, Train Accuracy: 0.1666
Epoch 23, Train Loss: 2.2729, Train Accuracy: 0.1732
Epoch 24, Train Loss: 2.2711, Train Accuracy: 0.1795
Epoch 25, Train Loss: 2.2693, Train Accuracy: 0.1863
Epoch 26, Train Loss: 2.2675, Train Accuracy: 0.1931
Epoch 27, Train Loss: 2.2657, Train Accuracy: 0.2006
Epoch 28, Train Loss: 2.2640, Train Accuracy: 0.2082
```

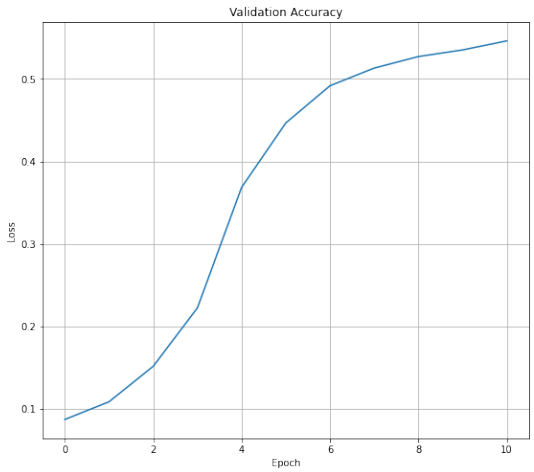
```
Epoch 29, Train Loss: 2.2622, Train Accuracy: 0.2169
        Epoch 30, Validation Loss: 354.6944, Validation Accuracy: 0.2224
Epoch 31, Train Loss: 2.2586, Train Accuracy: 0.2369
Epoch 32, Train Loss: 2.2568, Train Accuracy: 0.2497
Epoch 33, Train Loss: 2.2551, Train Accuracy: 0.2635
Epoch 34, Train Loss: 2.2533, Train Accuracy: 0.2792
Epoch 35, Train Loss: 2.2515, Train Accuracy: 0.2973
Epoch 36, Train Loss: 2.2497, Train Accuracy: 0.3140
Epoch 37, Train Loss: 2.2479, Train Accuracy: 0.3313
Epoch 38, Train Loss: 2.2462, Train Accuracy: 0.3457
Epoch 39, Train Loss: 2.2444, Train Accuracy: 0.3592
        Epoch 40, Validation Loss: 351.8265, Validation Accuracy: 0.3685
Epoch 41, Train Loss: 2.2408, Train Accuracy: 0.3840
Epoch 42, Train Loss: 2.2390, Train Accuracy: 0.3941
Epoch 43, Train Loss: 2.2372, Train Accuracy: 0.4032
Epoch 44, Train Loss: 2.2353, Train Accuracy: 0.4119
Epoch 45, Train Loss: 2.2335, Train Accuracy: 0.4204
Epoch 46, Train Loss: 2.2317, Train Accuracy: 0.4276
Epoch 47, Train Loss: 2.2298, Train Accuracy: 0.4346
Epoch 48, Train Loss: 2.2280, Train Accuracy: 0.4399
Epoch 49, Train Loss: 2.2261, Train Accuracy: 0.4457
        Epoch 50, Validation Loss: 348.8681, Validation Accuracy: 0.4466
Epoch 51, Train Loss: 2.2224, Train Accuracy: 0.4561
Epoch 52, Train Loss: 2.2205, Train Accuracy: 0.4608
Epoch 53, Train Loss: 2.2186, Train Accuracy: 0.4650
Epoch 54, Train Loss: 2.2167, Train Accuracy: 0.4682
Epoch 55, Train Loss: 2.2147, Train Accuracy: 0.4717
Epoch 56, Train Loss: 2.2128, Train Accuracy: 0.4752
Epoch 57, Train Loss: 2.2108, Train Accuracy: 0.4777
Epoch 58, Train Loss: 2.2089, Train Accuracy: 0.4803
Epoch 59, Train Loss: 2.2069, Train Accuracy: 0.4828
        Epoch 60, Validation Loss: 345.7390, Validation Accuracy: 0.4917
Epoch 61, Train Loss: 2.2029, Train Accuracy: 0.4875
Epoch 62, Train Loss: 2.2008, Train Accuracy: 0.4903
Epoch 63, Train Loss: 2.1988, Train Accuracy: 0.4916
Epoch 64, Train Loss: 2.1967, Train Accuracy: 0.4938
Epoch 65, Train Loss: 2.1947, Train Accuracy: 0.4961
Epoch 66, Train Loss: 2.1926, Train Accuracy: 0.4973
Epoch 67, Train Loss: 2.1904, Train Accuracy: 0.4998
Epoch 68, Train Loss: 2.1883, Train Accuracy: 0.5017
Epoch 69, Train Loss: 2.1862, Train Accuracy: 0.5027
        Epoch 70, Validation Loss: 342.3618, Validation Accuracy: 0.5131
Epoch 71, Train Loss: 2.1818, Train Accuracy: 0.5052
Epoch 72, Train Loss: 2.1796, Train Accuracy: 0.5067
Epoch 73, Train Loss: 2.1773, Train Accuracy: 0.5084
Epoch 74, Train Loss: 2.1751, Train Accuracy: 0.5097
Epoch 75, Train Loss: 2.1728, Train Accuracy: 0.5103
Epoch 76, Train Loss: 2.1705, Train Accuracy: 0.5111
```

```
Epoch 77, Train Loss: 2.1682, Train Accuracy: 0.5121
Epoch 78, Train Loss: 2.1659, Train Accuracy: 0.5134
Epoch 79, Train Loss: 2.1635, Train Accuracy: 0.5144
        Epoch 80, Validation Loss: 338.6699, Validation Accuracy: 0.5271
Epoch 81, Train Loss: 2.1587, Train Accuracy: 0.5163
Epoch 82, Train Loss: 2.1563, Train Accuracy: 0.5170
Epoch 83, Train Loss: 2.1538, Train Accuracy: 0.5177
Epoch 84, Train Loss: 2.1514, Train Accuracy: 0.5190
Epoch 85, Train Loss: 2.1489, Train Accuracy: 0.5194
Epoch 86, Train Loss: 2.1464, Train Accuracy: 0.5206
Epoch 87, Train Loss: 2.1438, Train Accuracy: 0.5213
Epoch 88, Train Loss: 2.1413, Train Accuracy: 0.5219
Epoch 89, Train Loss: 2.1387, Train Accuracy: 0.5232
        Epoch 90, Validation Loss: 334.6251, Validation Accuracy: 0.5351
Epoch 91, Train Loss: 2.1335, Train Accuracy: 0.5243
Epoch 92, Train Loss: 2.1308, Train Accuracy: 0.5253
Epoch 93, Train Loss: 2.1281, Train Accuracy: 0.5264
Epoch 94, Train Loss: 2.1254, Train Accuracy: 0.5273
Epoch 95, Train Loss: 2.1227, Train Accuracy: 0.5286
Epoch 96, Train Loss: 2.1199, Train Accuracy: 0.5294
Epoch 97, Train Loss: 2.1171, Train Accuracy: 0.5304
Epoch 98, Train Loss: 2.1143, Train Accuracy: 0.5314
Epoch 99, Train Loss: 2.1114, Train Accuracy: 0.5325
        Epoch 100, Validation Loss: 330.1805, Validation Accuracy: 0.5461
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 0:48:12 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

7.1.5 Comparaison entre les activations (Tanh & ReLU) et les optimiseurs (Adam & SGD)

La fonction d'activation Tanh (hyperbolique tangente) est une fonction de transfert sigmoid qui convertit une entrée linéaire en une sortie centrée autour de zéro dans l'intervalle [-1, 1]. Elle est souvent utilisée dans les réseaux de neurones de couches cachées pour normaliser les sorties des neurones.

La fonction d'activation ReLU (rectifiée linéaire) est une fonction de transfert non linéaire qui convertit toutes les entrées négatives en zéro. Il est utilisé pour résoudre le problème de décès du neurone, qui est un phénomène où les poids des neurones peuvent devenir négatifs et rester coincés à zéro.

En général, ReLU est considéré comme étant plus rapide et plus efficace pour les modèles de réseaux de neurones en raison de sa simplicité et de la vitesse de convergence plus rapide qu'il offre comparé à Tanh. Cependant, ReLU peut aussi causer des problèmes de saturation des neurones, où de nombreux neurones peuvent devenir saturés et ne pas produire de sortie. Cela peut rendre difficile la convergence du modèle. Tanh peut éviter ce problème mais il peut être plus lent à converger.

En fin de compte, le choix de la fonction d'activation dépend du modèle et des données sur lesquels vous travaillez et des performances que vous recherchez.

[11]:

```
print ("Model Linear Multi Class With Tanh (Adam): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhAdam,"\n\tComplexité:", model1ComplexityTanhAdam,␣
 ↪"\n\tValidation Loss:", model1ValidationLossTanhAdam, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhAdam, "] \n")
print ("Model Linear Multi Class With Tanh (SGD): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhSGD,"\n\tComplexité:", model1ComplexityTanhSGD,␣
 ↪"\n\tValidation Loss:", model1ValidationLossTanhSGD, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhSGD, "] \n")

if model1ValidationLossTanhAdam <= model1ValidationLossTanhSGD:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClass avec Tanh utilisant l'optimiseur (Adam) est le plus␣
 ↪performant !\n_____\n")
else:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClass avec Tanh utilisant l'optimiseur (SGD) est le plus␣
 ↪performant !\n_____\n")
```

```
Model Linear Multi Class With Tanh (Adam): [
        Temps d'exécution: 1003.714991569519
        Complexité: 235146
        Validation Loss: 12.005391817656346
        Accurancy: 0.9773 ]

Model Linear Multi Class With Tanh (SGD): [
        Temps d'exécution: 949.3833196163177
        Complexité: 235146
        Validation Loss: 283.09118032455444
        Accurancy: 0.656 ]



--------------------------------------------------
Le Modèle LinearMultiClass avec Tanh utilisant l'optimiseur (Adam) est le plus
performant !
--------------------------------------------------
```

[12]:
```
print ("Model Linear Multi Class With ReLU (Adam): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhReLuAdam, "\n\tComplexité:",␣
 ↪model1ComplexityTanhReLuAdam, "\n\tValidation Loss:",␣
 ↪model1ValidationLossTanhReLuAdam, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhReLuAdam, "] \n")
print ("Model Linear Multi Class With ReLU (SGD): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhReLuSGD, "\n\tComplexité:",␣
 ↪model1ComplexityTanhReLuSGD, "\n\tValidation Loss:",␣
 ↪model1ValidationLossTanhReLuSGD, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhReLuSGD, "] \n")
```

```
if model1ValidationLossTanhReLuAdam <= model1ValidationLossTanhReLuSGD:
    print("\n_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec ReLU utilisant l'optimiseur (Adam) est le plus␣
 ↪performant !\n_____________________________________________________\n")
else:
    print("\n_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec ReLU utilisant l'optimiseur (SGD) est le plus␣
 ↪performant !\n_____________________________________________________\n")
```

```
Model Linear Multi Class With ReLU (Adam): [
        Temps d'exécution: 986.2185385227203
        Complexité: 235146
        Validation Loss: 14.596343372249976
        Accurancy: 0.9721 ]

Model Linear Multi Class With ReLU (SGD): [
        Temps d'exécution: 955.7823257446289
        Complexité: 235146
        Validation Loss: 330.1805009841919
        Accurancy: 0.5461 ]



--------------------------------------------------
Le Modèle LinearMultiClass avec ReLU utilisant l'optimiseur (Adam) est le plus
performant !
--------------------------------------------------
```

[13]:
```
print ("Model Linear Multi Class With Tanh (SGD): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhSGD,"\n\tComplexité:", model1ComplexityTanhSGD,␣
 ↪"\n\tValidation Loss:", model1ValidationLossTanhSGD, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhSGD, "] \n")
print ("Model Linear Multi Class With ReLU (SGD): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhReLuSGD, "\n\tComplexité:",␣
 ↪model1ComplexityTanhReLuSGD, "\n\tValidation Loss:",␣
 ↪model1ValidationLossTanhReLuSGD, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhReLuSGD, "] \n")

if model1ValidationLossTanhSGD <= model1ValidationLossTanhReLuSGD:
    print("\n_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec Tanh utilisant l'optimiseur (SGD) est le plus␣
 ↪performant !\n_____________________________________________________\n")
else:
    print("\n_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec ReLU utilisant l'optimiseur (SGD) est le plus␣
 ↪performant !\n_____________________________________________________\n")
```

```
Model Linear Multi Class With Tanh (SGD): [
        Temps d'exécution: 949.3833196163177
        Complexité: 235146
        Validation Loss: 283.09118032455444
        Accurancy: 0.656 ]

Model Linear Multi Class With ReLU (SGD): [
        Temps d'exécution: 955.7823257446289
        Complexité: 235146
        Validation Loss: 330.1805009841919
        Accurancy: 0.5461 ]



--------------------------------------------------
Le Modèle LinearMultiClass avec Tanh utilisant l'optimiseur (SGD) est le plus
performant !
--------------------------------------------------
```

[14]:
```
print ("Model Linear Multi Class With Tanh (Adam): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhAdam,"\n\tComplexité:", model1ComplexityTanhAdam,␣
 ↪"\n\tValidation Loss:", model1ValidationLossTanhAdam, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhAdam, "] \n")
print ("Model Linear Multi Class With ReLU (Adam): [\n\tTemps d'exécution:",␣
 ↪model1ExecutionTimeTanhReLuAdam, "\n\tComplexité:",␣
 ↪model1ComplexityTanhReLuAdam, "\n\tValidation Loss:",␣
 ↪model1ValidationLossTanhReLuAdam, "\n\tAccurancy:",␣
 ↪model1AccuracyTanhReLuAdam, "] \n")

if model1ValidationLossTanhAdam <= model1ValidationLossTanhReLuAdam:
    print("Donc nous pouvons conclure que l'usage de la fonction d'entrainement␣
 ↪en utilisant l'activation <Tanh> est bien plus efficace que l'activation␣
 ↪<ReLU>. De plus l'optimiseur <Adam> est bien plus performant que␣
 ↪l'optimiseur <SGD> car parmi les modèles testés :")
    print("_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec Tanh utilisant l'optimiseur (Adam) est le plus␣
 ↪performant !\n_____________________________________________________\n")
else:
    print("Donc nous pouvons conclure que l'usage de la fonction d'entrainement␣
 ↪en utilisant l'activation <ReLU> est bien plus efficace que l'activation␣
 ↪<Tanh>. De plus l'optimiseur <Adam> est bien plus performant que␣
 ↪l'optimiseur <SGD> car parmi les modèles testés :")
    print("_____________________________________________________\nLe Modèle␣
 ↪LinearMultiClass avec ReLU utilisant l'optimiseur (Adam) est le plus␣
 ↪performant !\n_____________________________________________________\n")
```

```
Model Linear Multi Class With Tanh (Adam): [
        Temps d'exécution: 1003.714991569519
```

```
        Complexité: 235146
        Validation Loss: 12.005391817656346
        Accurancy: 0.9773 ]

Model Linear Multi Class With ReLU (Adam): [
        Temps d'exécution: 986.2185385227203
        Complexité: 235146
        Validation Loss: 14.596343372249976
        Accurancy: 0.9721 ]

Donc nous pouvons conclure que l'usage de la fonction d'entrainement en
utilisant l'activation <Tanh> est bien plus efficace que l'activation <ReLU>. De
plus l'optimiseur <Adam> est bien plus performant que l'optimiseur <SGD> car
parmi les modèles testés :

--------------------------------------------------
Le Modèle LinearMultiClass avec Tanh utilisant l'optimiseur (Adam) est le plus
performant !

--------------------------------------------------
```

7.1.6 Construction d'un histogramme avec les données du modèle

[ ]:
```
summary = SummaryWriter(f"{TB_PATH}/{model.name}-historigramm")
addWeightsHisto(summary, model, epoch)

# Load the TensorBoard notebook extension
%load_ext tensorboard
%tensorboard --logdir {TB_PATH}/{model.name}-historigramm
```

7.1.7 Pénalisation des couches

Une première technique pour éviter le sur-apprentissage est de régulariser chaque couche par une pénalisation sur les poids, i.e. de favoriser des poids faibles. On parle de pénalisation L1 lorsque la pénalité est de la forme $\|W\|_1$ et L2 lorsque la norme L2 est utilisée : $\|W\|_2^2$. En pratique, cela consiste à rajouter à la fonction de coût globale du réseau un terme en $\lambda Pen(W)$ pour les paramètres de chaque couche que l'on veut régulariser.

En résumé, l'utilisation de la régularisation des couches est importante pour aider à prévenir le sur-apprentissage et améliorer la performance générale du modèle sur des données non vues auparavant.

Pour la norme L2 de 10^-5 avec l'optimiseur = Adam

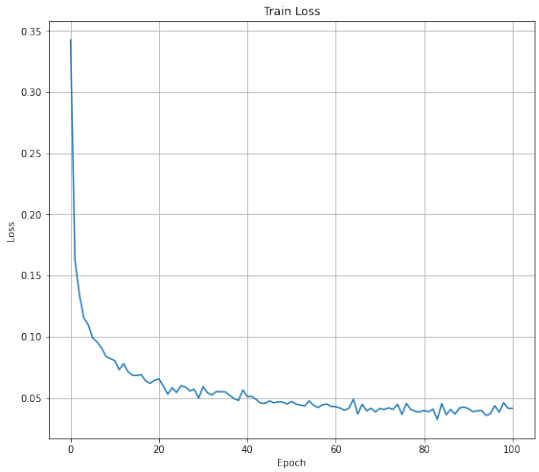
[16]:
```
print("__________________LA NORME L2 DE 10^-5 (Adam)__________________")
weight_decay = 10e-5
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
```

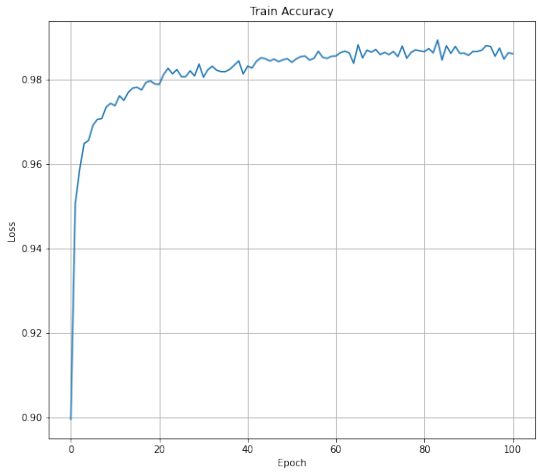
```
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-Adam-10e-5")
```

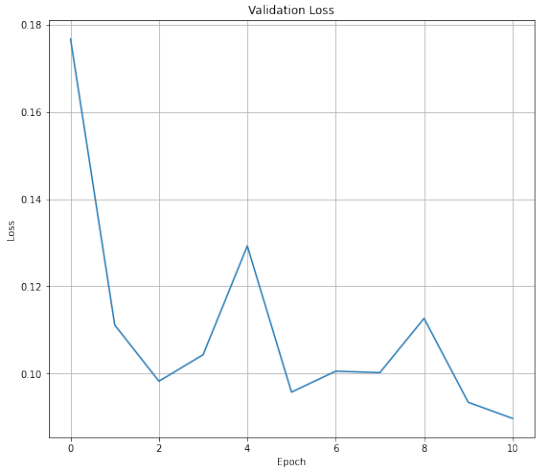
```
__________________LA NORME L2 DE 10^-5 (Adam)__________________
        Epoch 0, Validation Loss: 27.7461, Validation Accuracy: 0.9450
Epoch 1, Train Loss: 0.1627, Train Accuracy: 0.9505
Epoch 2, Train Loss: 0.1345, Train Accuracy: 0.9587
Epoch 3, Train Loss: 0.1154, Train Accuracy: 0.9648
Epoch 4, Train Loss: 0.1098, Train Accuracy: 0.9655
Epoch 5, Train Loss: 0.0991, Train Accuracy: 0.9691
Epoch 6, Train Loss: 0.0957, Train Accuracy: 0.9705
Epoch 7, Train Loss: 0.0909, Train Accuracy: 0.9707
Epoch 8, Train Loss: 0.0840, Train Accuracy: 0.9734
Epoch 9, Train Loss: 0.0821, Train Accuracy: 0.9743
        Epoch 10, Validation Loss: 17.4501, Validation Accuracy: 0.9650
Epoch 11, Train Loss: 0.0730, Train Accuracy: 0.9761
Epoch 12, Train Loss: 0.0780, Train Accuracy: 0.9750
Epoch 13, Train Loss: 0.0714, Train Accuracy: 0.9769
Epoch 14, Train Loss: 0.0685, Train Accuracy: 0.9779
Epoch 15, Train Loss: 0.0682, Train Accuracy: 0.9781
Epoch 16, Train Loss: 0.0689, Train Accuracy: 0.9775
Epoch 17, Train Loss: 0.0639, Train Accuracy: 0.9792
Epoch 18, Train Loss: 0.0618, Train Accuracy: 0.9796
Epoch 19, Train Loss: 0.0644, Train Accuracy: 0.9789
        Epoch 20, Validation Loss: 15.4255, Validation Accuracy: 0.9702
Epoch 21, Train Loss: 0.0596, Train Accuracy: 0.9811
Epoch 22, Train Loss: 0.0529, Train Accuracy: 0.9826
Epoch 23, Train Loss: 0.0582, Train Accuracy: 0.9813
Epoch 24, Train Loss: 0.0544, Train Accuracy: 0.9823
Epoch 25, Train Loss: 0.0599, Train Accuracy: 0.9806
Epoch 26, Train Loss: 0.0589, Train Accuracy: 0.9806
Epoch 27, Train Loss: 0.0556, Train Accuracy: 0.9820
Epoch 28, Train Loss: 0.0569, Train Accuracy: 0.9808
Epoch 29, Train Loss: 0.0498, Train Accuracy: 0.9836
        Epoch 30, Validation Loss: 16.3800, Validation Accuracy: 0.9701
Epoch 31, Train Loss: 0.0540, Train Accuracy: 0.9822
Epoch 32, Train Loss: 0.0523, Train Accuracy: 0.9831
Epoch 33, Train Loss: 0.0551, Train Accuracy: 0.9821
Epoch 34, Train Loss: 0.0550, Train Accuracy: 0.9818
Epoch 35, Train Loss: 0.0549, Train Accuracy: 0.9818
Epoch 36, Train Loss: 0.0521, Train Accuracy: 0.9824
Epoch 37, Train Loss: 0.0493, Train Accuracy: 0.9834
Epoch 38, Train Loss: 0.0478, Train Accuracy: 0.9844
Epoch 39, Train Loss: 0.0565, Train Accuracy: 0.9813
        Epoch 40, Validation Loss: 20.2962, Validation Accuracy: 0.9625
Epoch 41, Train Loss: 0.0512, Train Accuracy: 0.9827
Epoch 42, Train Loss: 0.0486, Train Accuracy: 0.9843
Epoch 43, Train Loss: 0.0457, Train Accuracy: 0.9851
```

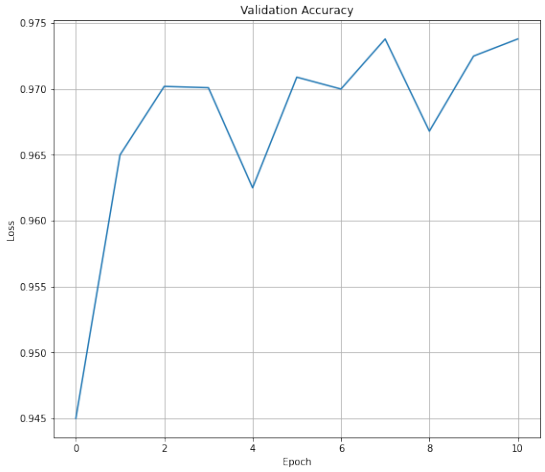
```
Epoch 44, Train Loss: 0.0453, Train Accuracy: 0.9849
Epoch 45, Train Loss: 0.0475, Train Accuracy: 0.9843
Epoch 46, Train Loss: 0.0459, Train Accuracy: 0.9848
Epoch 47, Train Loss: 0.0467, Train Accuracy: 0.9842
Epoch 48, Train Loss: 0.0466, Train Accuracy: 0.9846
Epoch 49, Train Loss: 0.0449, Train Accuracy: 0.9849
        Epoch 50, Validation Loss: 15.0327, Validation Accuracy: 0.9709
Epoch 51, Train Loss: 0.0449, Train Accuracy: 0.9848
Epoch 52, Train Loss: 0.0440, Train Accuracy: 0.9854
Epoch 53, Train Loss: 0.0434, Train Accuracy: 0.9855
Epoch 54, Train Loss: 0.0475, Train Accuracy: 0.9846
Epoch 55, Train Loss: 0.0438, Train Accuracy: 0.9850
Epoch 56, Train Loss: 0.0420, Train Accuracy: 0.9867
Epoch 57, Train Loss: 0.0443, Train Accuracy: 0.9852
Epoch 58, Train Loss: 0.0449, Train Accuracy: 0.9850
Epoch 59, Train Loss: 0.0429, Train Accuracy: 0.9855
        Epoch 60, Validation Loss: 15.7866, Validation Accuracy: 0.9700
Epoch 61, Train Loss: 0.0416, Train Accuracy: 0.9863
Epoch 62, Train Loss: 0.0399, Train Accuracy: 0.9867
Epoch 63, Train Loss: 0.0414, Train Accuracy: 0.9863
Epoch 64, Train Loss: 0.0490, Train Accuracy: 0.9838
Epoch 65, Train Loss: 0.0367, Train Accuracy: 0.9882
Epoch 66, Train Loss: 0.0448, Train Accuracy: 0.9850
Epoch 67, Train Loss: 0.0393, Train Accuracy: 0.9869
Epoch 68, Train Loss: 0.0415, Train Accuracy: 0.9864
Epoch 69, Train Loss: 0.0385, Train Accuracy: 0.9871
        Epoch 70, Validation Loss: 15.7359, Validation Accuracy: 0.9738
Epoch 71, Train Loss: 0.0405, Train Accuracy: 0.9864
Epoch 72, Train Loss: 0.0418, Train Accuracy: 0.9859
Epoch 73, Train Loss: 0.0405, Train Accuracy: 0.9866
Epoch 74, Train Loss: 0.0447, Train Accuracy: 0.9854
Epoch 75, Train Loss: 0.0364, Train Accuracy: 0.9879
Epoch 76, Train Loss: 0.0454, Train Accuracy: 0.9850
Epoch 77, Train Loss: 0.0405, Train Accuracy: 0.9864
Epoch 78, Train Loss: 0.0390, Train Accuracy: 0.9870
Epoch 79, Train Loss: 0.0384, Train Accuracy: 0.9867
        Epoch 80, Validation Loss: 17.6875, Validation Accuracy: 0.9668
Epoch 81, Train Loss: 0.0385, Train Accuracy: 0.9873
Epoch 82, Train Loss: 0.0408, Train Accuracy: 0.9863
Epoch 83, Train Loss: 0.0324, Train Accuracy: 0.9893
Epoch 84, Train Loss: 0.0454, Train Accuracy: 0.9846
Epoch 85, Train Loss: 0.0361, Train Accuracy: 0.9879
Epoch 86, Train Loss: 0.0406, Train Accuracy: 0.9861
Epoch 87, Train Loss: 0.0367, Train Accuracy: 0.9878
Epoch 88, Train Loss: 0.0416, Train Accuracy: 0.9861
Epoch 89, Train Loss: 0.0425, Train Accuracy: 0.9862
        Epoch 90, Validation Loss: 14.6654, Validation Accuracy: 0.9725
Epoch 91, Train Loss: 0.0387, Train Accuracy: 0.9866
```

```
Epoch 92, Train Loss: 0.0393, Train Accuracy: 0.9866
Epoch 93, Train Loss: 0.0396, Train Accuracy: 0.9869
Epoch 94, Train Loss: 0.0356, Train Accuracy: 0.9880
Epoch 95, Train Loss: 0.0369, Train Accuracy: 0.9878
Epoch 96, Train Loss: 0.0436, Train Accuracy: 0.9855
Epoch 97, Train Loss: 0.0383, Train Accuracy: 0.9874
Epoch 98, Train Loss: 0.0459, Train Accuracy: 0.9848
Epoch 99, Train Loss: 0.0414, Train Accuracy: 0.9863
        Epoch 100, Validation Loss: 14.0883, Validation Accuracy: 0.9738
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 1:16:28 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[16]:
```
(1036.7679529190063, 99710, 14.088333635765593, 0.9738)
```

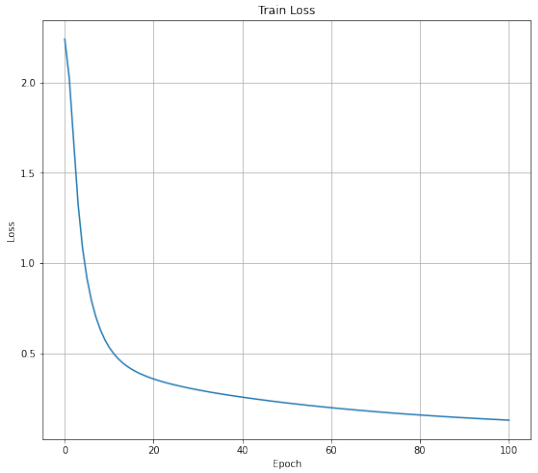
Pour la norme L2 de 10^-5 avec l'optimiseur = SGD

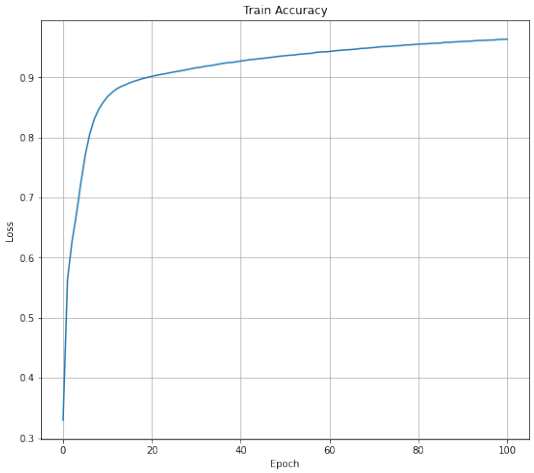
[17]:
```
print("__________________LA NORME L2 DE 10^-5 (SGD)__________________")
weight_decay = 10e-5
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-SGD-10e-5")
```

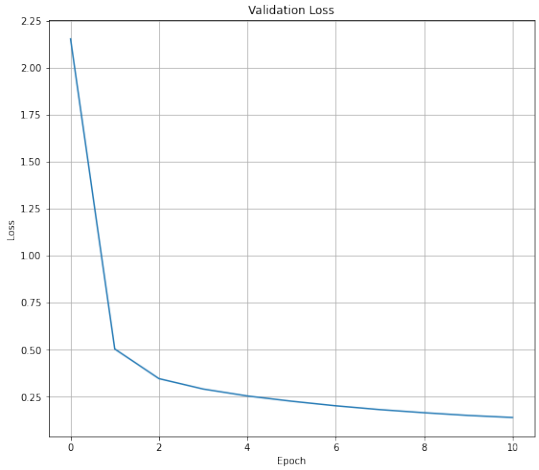
```
__________________LA NORME L2 DE 10^-5 (SGD)__________________
        Epoch 0, Validation Loss: 337.9625, Validation Accuracy: 0.5036
Epoch 1, Train Loss: 2.0262, Train Accuracy: 0.5634
Epoch 2, Train Loss: 1.6726, Train Accuracy: 0.6259
Epoch 3, Train Loss: 1.3259, Train Accuracy: 0.6718
Epoch 4, Train Loss: 1.0838, Train Accuracy: 0.7241
Epoch 5, Train Loss: 0.9158, Train Accuracy: 0.7712
Epoch 6, Train Loss: 0.7940, Train Accuracy: 0.8054
Epoch 7, Train Loss: 0.7024, Train Accuracy: 0.8299
Epoch 8, Train Loss: 0.6321, Train Accuracy: 0.8462
Epoch 9, Train Loss: 0.5774, Train Accuracy: 0.8580
        Epoch 10, Validation Loss: 79.1611, Validation Accuracy: 0.8747
Epoch 11, Train Loss: 0.4999, Train Accuracy: 0.8745
Epoch 12, Train Loss: 0.4719, Train Accuracy: 0.8803
Epoch 13, Train Loss: 0.4489, Train Accuracy: 0.8843
Epoch 14, Train Loss: 0.4299, Train Accuracy: 0.8872
Epoch 15, Train Loss: 0.4140, Train Accuracy: 0.8907
Epoch 16, Train Loss: 0.4000, Train Accuracy: 0.8933
Epoch 17, Train Loss: 0.3880, Train Accuracy: 0.8956
Epoch 18, Train Loss: 0.3774, Train Accuracy: 0.8979
Epoch 19, Train Loss: 0.3677, Train Accuracy: 0.8998
        Epoch 20, Validation Loss: 54.4194, Validation Accuracy: 0.9045
Epoch 21, Train Loss: 0.3513, Train Accuracy: 0.9032
Epoch 22, Train Loss: 0.3439, Train Accuracy: 0.9046
Epoch 23, Train Loss: 0.3371, Train Accuracy: 0.9059
Epoch 24, Train Loss: 0.3309, Train Accuracy: 0.9074
Epoch 25, Train Loss: 0.3249, Train Accuracy: 0.9088
Epoch 26, Train Loss: 0.3194, Train Accuracy: 0.9101
Epoch 27, Train Loss: 0.3139, Train Accuracy: 0.9113
Epoch 28, Train Loss: 0.3086, Train Accuracy: 0.9129
Epoch 29, Train Loss: 0.3036, Train Accuracy: 0.9144
        Epoch 30, Validation Loss: 45.7194, Validation Accuracy: 0.9179
Epoch 31, Train Loss: 0.2942, Train Accuracy: 0.9166
Epoch 32, Train Loss: 0.2898, Train Accuracy: 0.9184
```

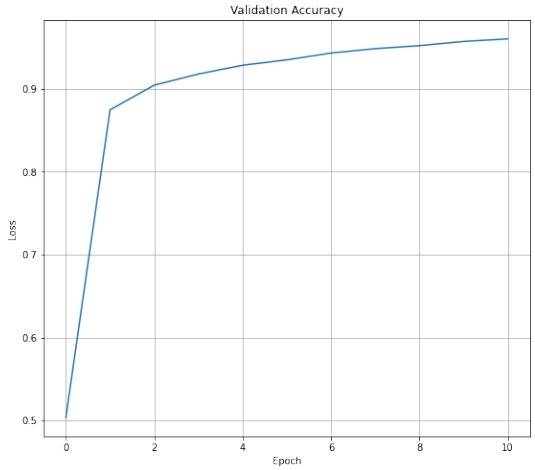
```
Epoch 33, Train Loss: 0.2853, Train Accuracy: 0.9191
Epoch 34, Train Loss: 0.2813, Train Accuracy: 0.9204
Epoch 35, Train Loss: 0.2772, Train Accuracy: 0.9217
Epoch 36, Train Loss: 0.2732, Train Accuracy: 0.9231
Epoch 37, Train Loss: 0.2694, Train Accuracy: 0.9242
Epoch 38, Train Loss: 0.2656, Train Accuracy: 0.9246
Epoch 39, Train Loss: 0.2620, Train Accuracy: 0.9257
        Epoch 40, Validation Loss: 39.9460, Validation Accuracy: 0.9285
Epoch 41, Train Loss: 0.2549, Train Accuracy: 0.9280
Epoch 42, Train Loss: 0.2515, Train Accuracy: 0.9291
Epoch 43, Train Loss: 0.2480, Train Accuracy: 0.9298
Epoch 44, Train Loss: 0.2448, Train Accuracy: 0.9306
Epoch 45, Train Loss: 0.2417, Train Accuracy: 0.9314
Epoch 46, Train Loss: 0.2385, Train Accuracy: 0.9322
Epoch 47, Train Loss: 0.2354, Train Accuracy: 0.9332
Epoch 48, Train Loss: 0.2324, Train Accuracy: 0.9341
Epoch 49, Train Loss: 0.2293, Train Accuracy: 0.9351
        Epoch 50, Validation Loss: 35.5743, Validation Accuracy: 0.9350
Epoch 51, Train Loss: 0.2237, Train Accuracy: 0.9366
Epoch 52, Train Loss: 0.2209, Train Accuracy: 0.9369
Epoch 53, Train Loss: 0.2180, Train Accuracy: 0.9381
Epoch 54, Train Loss: 0.2155, Train Accuracy: 0.9386
Epoch 55, Train Loss: 0.2126, Train Accuracy: 0.9394
Epoch 56, Train Loss: 0.2102, Train Accuracy: 0.9398
Epoch 57, Train Loss: 0.2077, Train Accuracy: 0.9414
Epoch 58, Train Loss: 0.2050, Train Accuracy: 0.9421
Epoch 59, Train Loss: 0.2027, Train Accuracy: 0.9423
        Epoch 60, Validation Loss: 31.7334, Validation Accuracy: 0.9432
Epoch 61, Train Loss: 0.1977, Train Accuracy: 0.9438
Epoch 62, Train Loss: 0.1954, Train Accuracy: 0.9446
Epoch 63, Train Loss: 0.1932, Train Accuracy: 0.9452
Epoch 64, Train Loss: 0.1911, Train Accuracy: 0.9457
Epoch 65, Train Loss: 0.1888, Train Accuracy: 0.9462
Epoch 66, Train Loss: 0.1866, Train Accuracy: 0.9468
Epoch 67, Train Loss: 0.1844, Train Accuracy: 0.9479
Epoch 68, Train Loss: 0.1824, Train Accuracy: 0.9482
Epoch 69, Train Loss: 0.1803, Train Accuracy: 0.9488
        Epoch 70, Validation Loss: 28.5230, Validation Accuracy: 0.9485
Epoch 71, Train Loss: 0.1763, Train Accuracy: 0.9503
Epoch 72, Train Loss: 0.1743, Train Accuracy: 0.9510
Epoch 73, Train Loss: 0.1723, Train Accuracy: 0.9512
Epoch 74, Train Loss: 0.1705, Train Accuracy: 0.9517
Epoch 75, Train Loss: 0.1686, Train Accuracy: 0.9523
Epoch 76, Train Loss: 0.1668, Train Accuracy: 0.9526
Epoch 77, Train Loss: 0.1650, Train Accuracy: 0.9537
Epoch 78, Train Loss: 0.1633, Train Accuracy: 0.9538
Epoch 79, Train Loss: 0.1615, Train Accuracy: 0.9544
        Epoch 80, Validation Loss: 25.9427, Validation Accuracy: 0.9521
```

```
Epoch 81, Train Loss: 0.1582, Train Accuracy: 0.9555
Epoch 82, Train Loss: 0.1567, Train Accuracy: 0.9557
Epoch 83, Train Loss: 0.1550, Train Accuracy: 0.9565
Epoch 84, Train Loss: 0.1534, Train Accuracy: 0.9565
Epoch 85, Train Loss: 0.1518, Train Accuracy: 0.9570
Epoch 86, Train Loss: 0.1503, Train Accuracy: 0.9581
Epoch 87, Train Loss: 0.1488, Train Accuracy: 0.9580
Epoch 88, Train Loss: 0.1473, Train Accuracy: 0.9587
Epoch 89, Train Loss: 0.1458, Train Accuracy: 0.9592
        Epoch 90, Validation Loss: 23.6740, Validation Accuracy: 0.9572
Epoch 91, Train Loss: 0.1433, Train Accuracy: 0.9599
Epoch 92, Train Loss: 0.1417, Train Accuracy: 0.9602
Epoch 93, Train Loss: 0.1403, Train Accuracy: 0.9610
Epoch 94, Train Loss: 0.1390, Train Accuracy: 0.9612
Epoch 95, Train Loss: 0.1377, Train Accuracy: 0.9614
Epoch 96, Train Loss: 0.1363, Train Accuracy: 0.9618
Epoch 97, Train Loss: 0.1352, Train Accuracy: 0.9619
Epoch 98, Train Loss: 0.1338, Train Accuracy: 0.9629
Epoch 99, Train Loss: 0.1326, Train Accuracy: 0.9630
        Epoch 100, Validation Loss: 21.9729, Validation Accuracy: 0.9603
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Reusing TensorBoard on port 6006 (pid 5463), started 2:26:11 ago. (Use '!kill␣
 ↪5463' to kill it.)

<IPython.core.display.HTML object>
```

[17]:
```
(4181.25968337059, 99710, 21.972865846473724, 0.9603)
```

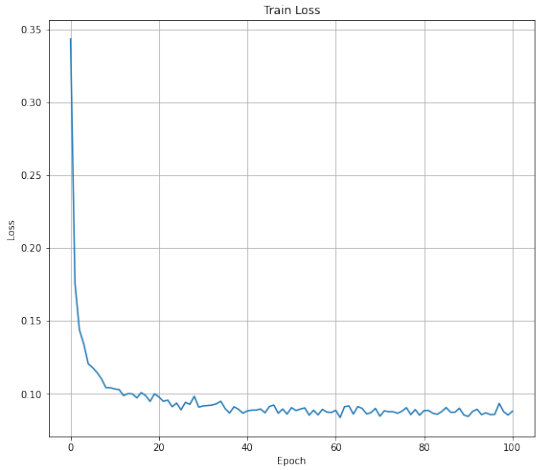
Pour la norme L2 de 10^-4 avec l'optimiseur = Adam

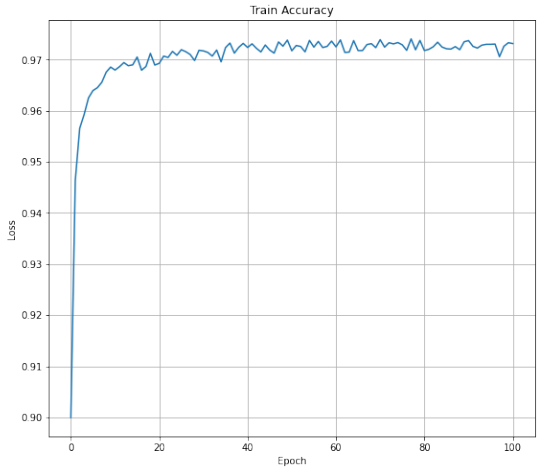
[18]:
```
print("\n_____________________LA NORME L2 DE 10^-4 (Adam)_____________________")
weight_decay = 10e-4
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-Adam-10e-4")
```

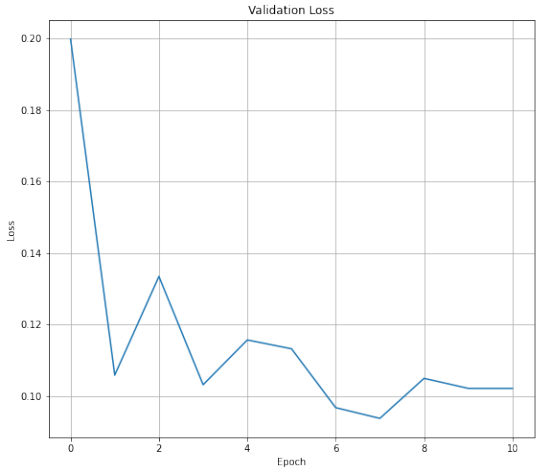
```
_____________________LA NORME L2 DE 10^-4 (Adam)_____________________
        Epoch 0, Validation Loss: 31.3756, Validation Accuracy: 0.9352
Epoch 1, Train Loss: 0.1759, Train Accuracy: 0.9464
Epoch 2, Train Loss: 0.1438, Train Accuracy: 0.9566
Epoch 3, Train Loss: 0.1340, Train Accuracy: 0.9592
Epoch 4, Train Loss: 0.1206, Train Accuracy: 0.9625
Epoch 5, Train Loss: 0.1179, Train Accuracy: 0.9639
Epoch 6, Train Loss: 0.1145, Train Accuracy: 0.9645
Epoch 7, Train Loss: 0.1102, Train Accuracy: 0.9656
Epoch 8, Train Loss: 0.1042, Train Accuracy: 0.9675
Epoch 9, Train Loss: 0.1041, Train Accuracy: 0.9685
```

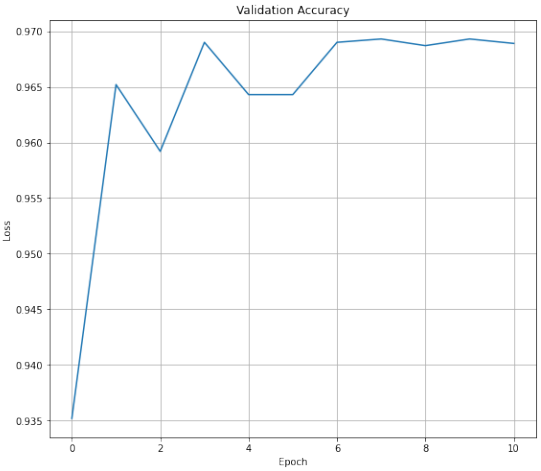
```
        Epoch 10, Validation Loss: 16.6097, Validation Accuracy: 0.9652
Epoch 11, Train Loss: 0.1028, Train Accuracy: 0.9686
Epoch 12, Train Loss: 0.0987, Train Accuracy: 0.9694
Epoch 13, Train Loss: 0.1001, Train Accuracy: 0.9688
Epoch 14, Train Loss: 0.0999, Train Accuracy: 0.9690
Epoch 15, Train Loss: 0.0972, Train Accuracy: 0.9705
Epoch 16, Train Loss: 0.1007, Train Accuracy: 0.9679
Epoch 17, Train Loss: 0.0987, Train Accuracy: 0.9687
Epoch 18, Train Loss: 0.0947, Train Accuracy: 0.9712
Epoch 19, Train Loss: 0.1000, Train Accuracy: 0.9689
        Epoch 20, Validation Loss: 20.9448, Validation Accuracy: 0.9592
Epoch 21, Train Loss: 0.0948, Train Accuracy: 0.9707
Epoch 22, Train Loss: 0.0956, Train Accuracy: 0.9704
Epoch 23, Train Loss: 0.0911, Train Accuracy: 0.9716
Epoch 24, Train Loss: 0.0936, Train Accuracy: 0.9708
Epoch 25, Train Loss: 0.0889, Train Accuracy: 0.9719
Epoch 26, Train Loss: 0.0941, Train Accuracy: 0.9716
Epoch 27, Train Loss: 0.0927, Train Accuracy: 0.9710
Epoch 28, Train Loss: 0.0983, Train Accuracy: 0.9698
Epoch 29, Train Loss: 0.0908, Train Accuracy: 0.9718
        Epoch 30, Validation Loss: 16.1870, Validation Accuracy: 0.9690
Epoch 31, Train Loss: 0.0919, Train Accuracy: 0.9714
Epoch 32, Train Loss: 0.0922, Train Accuracy: 0.9707
Epoch 33, Train Loss: 0.0931, Train Accuracy: 0.9719
Epoch 34, Train Loss: 0.0949, Train Accuracy: 0.9696
Epoch 35, Train Loss: 0.0897, Train Accuracy: 0.9723
Epoch 36, Train Loss: 0.0868, Train Accuracy: 0.9732
Epoch 37, Train Loss: 0.0911, Train Accuracy: 0.9713
Epoch 38, Train Loss: 0.0893, Train Accuracy: 0.9724
Epoch 39, Train Loss: 0.0867, Train Accuracy: 0.9732
        Epoch 40, Validation Loss: 18.1542, Validation Accuracy: 0.9643
Epoch 41, Train Loss: 0.0887, Train Accuracy: 0.9731
Epoch 42, Train Loss: 0.0888, Train Accuracy: 0.9722
Epoch 43, Train Loss: 0.0895, Train Accuracy: 0.9715
Epoch 44, Train Loss: 0.0869, Train Accuracy: 0.9728
Epoch 45, Train Loss: 0.0912, Train Accuracy: 0.9719
Epoch 46, Train Loss: 0.0922, Train Accuracy: 0.9713
Epoch 47, Train Loss: 0.0866, Train Accuracy: 0.9735
Epoch 48, Train Loss: 0.0895, Train Accuracy: 0.9726
Epoch 49, Train Loss: 0.0860, Train Accuracy: 0.9738
        Epoch 50, Validation Loss: 17.7677, Validation Accuracy: 0.9643
Epoch 51, Train Loss: 0.0885, Train Accuracy: 0.9728
Epoch 52, Train Loss: 0.0895, Train Accuracy: 0.9726
Epoch 53, Train Loss: 0.0903, Train Accuracy: 0.9715
Epoch 54, Train Loss: 0.0853, Train Accuracy: 0.9738
Epoch 55, Train Loss: 0.0887, Train Accuracy: 0.9724
Epoch 56, Train Loss: 0.0855, Train Accuracy: 0.9736
Epoch 57, Train Loss: 0.0893, Train Accuracy: 0.9724
```

```
Epoch 58, Train Loss: 0.0874, Train Accuracy: 0.9726
Epoch 59, Train Loss: 0.0871, Train Accuracy: 0.9736
        Epoch 60, Validation Loss: 15.1802, Validation Accuracy: 0.9690
Epoch 61, Train Loss: 0.0837, Train Accuracy: 0.9738
Epoch 62, Train Loss: 0.0911, Train Accuracy: 0.9714
Epoch 63, Train Loss: 0.0917, Train Accuracy: 0.9715
Epoch 64, Train Loss: 0.0860, Train Accuracy: 0.9737
Epoch 65, Train Loss: 0.0912, Train Accuracy: 0.9718
Epoch 66, Train Loss: 0.0899, Train Accuracy: 0.9718
Epoch 67, Train Loss: 0.0862, Train Accuracy: 0.9729
Epoch 68, Train Loss: 0.0871, Train Accuracy: 0.9731
Epoch 69, Train Loss: 0.0899, Train Accuracy: 0.9723
        Epoch 70, Validation Loss: 14.7100, Validation Accuracy: 0.9693
Epoch 71, Train Loss: 0.0883, Train Accuracy: 0.9724
Epoch 72, Train Loss: 0.0877, Train Accuracy: 0.9733
Epoch 73, Train Loss: 0.0877, Train Accuracy: 0.9731
Epoch 74, Train Loss: 0.0866, Train Accuracy: 0.9733
Epoch 75, Train Loss: 0.0880, Train Accuracy: 0.9729
Epoch 76, Train Loss: 0.0904, Train Accuracy: 0.9718
Epoch 77, Train Loss: 0.0857, Train Accuracy: 0.9740
Epoch 78, Train Loss: 0.0891, Train Accuracy: 0.9719
Epoch 79, Train Loss: 0.0853, Train Accuracy: 0.9738
        Epoch 80, Validation Loss: 16.4678, Validation Accuracy: 0.9687
Epoch 81, Train Loss: 0.0885, Train Accuracy: 0.9720
Epoch 82, Train Loss: 0.0866, Train Accuracy: 0.9725
Epoch 83, Train Loss: 0.0859, Train Accuracy: 0.9734
Epoch 84, Train Loss: 0.0878, Train Accuracy: 0.9725
Epoch 85, Train Loss: 0.0905, Train Accuracy: 0.9721
Epoch 86, Train Loss: 0.0873, Train Accuracy: 0.9720
Epoch 87, Train Loss: 0.0873, Train Accuracy: 0.9725
Epoch 88, Train Loss: 0.0901, Train Accuracy: 0.9719
Epoch 89, Train Loss: 0.0855, Train Accuracy: 0.9735
        Epoch 90, Validation Loss: 16.0273, Validation Accuracy: 0.9693
Epoch 91, Train Loss: 0.0879, Train Accuracy: 0.9726
Epoch 92, Train Loss: 0.0893, Train Accuracy: 0.9722
Epoch 93, Train Loss: 0.0857, Train Accuracy: 0.9728
Epoch 94, Train Loss: 0.0869, Train Accuracy: 0.9730
Epoch 95, Train Loss: 0.0857, Train Accuracy: 0.9730
Epoch 96, Train Loss: 0.0859, Train Accuracy: 0.9730
Epoch 97, Train Loss: 0.0934, Train Accuracy: 0.9706
Epoch 98, Train Loss: 0.0877, Train Accuracy: 0.9726
Epoch 99, Train Loss: 0.0854, Train Accuracy: 0.9733
        Epoch 100, Validation Loss: 16.0275, Validation Accuracy: 0.9689
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 3:36:57 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[18]:
```
(4244.096433877945, 99710, 16.027456209994853, 0.9689)
```

Pour la norme L2 de 10^-4 avec l'optimiseur = SGD

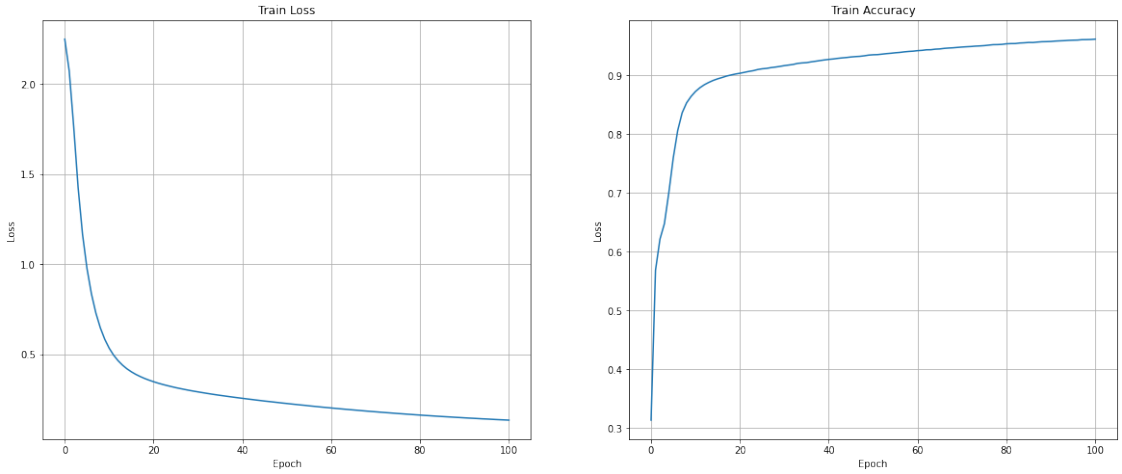
[19]:
```
print("\n__________________LA NORME L2 DE 10^-4 (SGD)__________________")
weight_decay = 10e-4
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
```

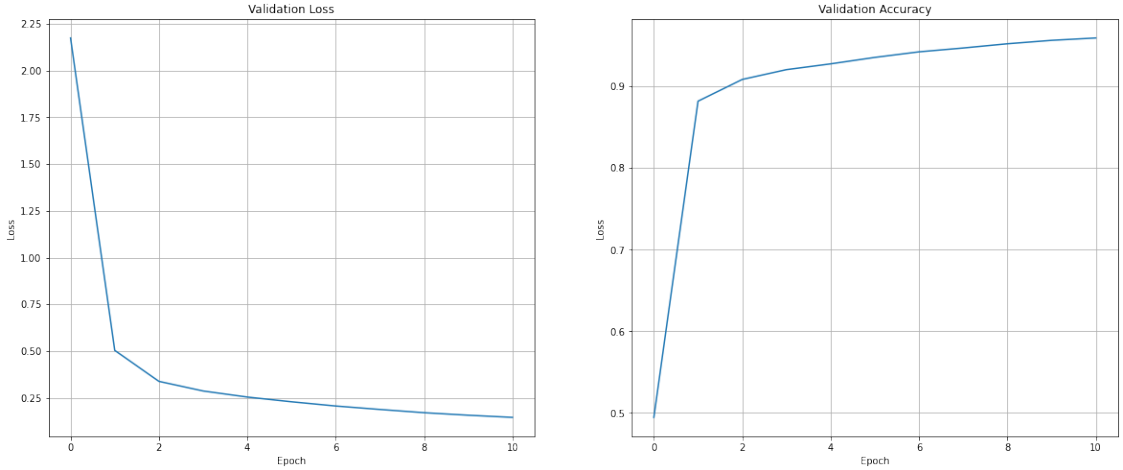
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-SGD-10e-4")
```

```
__________________LA NORME L2 DE 10^-4 (SGD)__________________
        Epoch 0, Validation Loss: 341.4649, Validation Accuracy: 0.4946
Epoch 1, Train Loss: 2.0725, Train Accuracy: 0.5686
Epoch 2, Train Loss: 1.7659, Train Accuracy: 0.6215
Epoch 3, Train Loss: 1.4272, Train Accuracy: 0.6479
Epoch 4, Train Loss: 1.1670, Train Accuracy: 0.7009
Epoch 5, Train Loss: 0.9776, Train Accuracy: 0.7601
Epoch 6, Train Loss: 0.8370, Train Accuracy: 0.8066
Epoch 7, Train Loss: 0.7307, Train Accuracy: 0.8366
Epoch 8, Train Loss: 0.6487, Train Accuracy: 0.8536
Epoch 9, Train Loss: 0.5852, Train Accuracy: 0.8644
        Epoch 10, Validation Loss: 79.0319, Validation Accuracy: 0.8818
Epoch 11, Train Loss: 0.4969, Train Accuracy: 0.8793
Epoch 12, Train Loss: 0.4663, Train Accuracy: 0.8844
Epoch 13, Train Loss: 0.4416, Train Accuracy: 0.8883
Epoch 14, Train Loss: 0.4212, Train Accuracy: 0.8917
Epoch 15, Train Loss: 0.4046, Train Accuracy: 0.8944
Epoch 16, Train Loss: 0.3903, Train Accuracy: 0.8967
Epoch 17, Train Loss: 0.3783, Train Accuracy: 0.8991
Epoch 18, Train Loss: 0.3675, Train Accuracy: 0.9010
Epoch 19, Train Loss: 0.3582, Train Accuracy: 0.9026
        Epoch 20, Validation Loss: 52.9862, Validation Accuracy: 0.9084
Epoch 21, Train Loss: 0.3420, Train Accuracy: 0.9054
Epoch 22, Train Loss: 0.3350, Train Accuracy: 0.9070
Epoch 23, Train Loss: 0.3286, Train Accuracy: 0.9083
Epoch 24, Train Loss: 0.3227, Train Accuracy: 0.9102
Epoch 25, Train Loss: 0.3169, Train Accuracy: 0.9115
Epoch 26, Train Loss: 0.3118, Train Accuracy: 0.9122
Epoch 27, Train Loss: 0.3068, Train Accuracy: 0.9135
Epoch 28, Train Loss: 0.3020, Train Accuracy: 0.9144
Epoch 29, Train Loss: 0.2976, Train Accuracy: 0.9156
        Epoch 30, Validation Loss: 44.9190, Validation Accuracy: 0.9204
Epoch 31, Train Loss: 0.2892, Train Accuracy: 0.9179
Epoch 32, Train Loss: 0.2852, Train Accuracy: 0.9189
Epoch 33, Train Loss: 0.2814, Train Accuracy: 0.9207
Epoch 34, Train Loss: 0.2776, Train Accuracy: 0.9215
Epoch 35, Train Loss: 0.2741, Train Accuracy: 0.9220
Epoch 36, Train Loss: 0.2707, Train Accuracy: 0.9232
Epoch 37, Train Loss: 0.2672, Train Accuracy: 0.9243
Epoch 38, Train Loss: 0.2639, Train Accuracy: 0.9254
Epoch 39, Train Loss: 0.2607, Train Accuracy: 0.9265
        Epoch 40, Validation Loss: 39.8246, Validation Accuracy: 0.9275
```

```
Epoch 41, Train Loss: 0.2544, Train Accuracy: 0.9282
Epoch 42, Train Loss: 0.2515, Train Accuracy: 0.9292
Epoch 43, Train Loss: 0.2484, Train Accuracy: 0.9300
Epoch 44, Train Loss: 0.2455, Train Accuracy: 0.9307
Epoch 45, Train Loss: 0.2425, Train Accuracy: 0.9316
Epoch 46, Train Loss: 0.2398, Train Accuracy: 0.9322
Epoch 47, Train Loss: 0.2371, Train Accuracy: 0.9327
Epoch 48, Train Loss: 0.2343, Train Accuracy: 0.9336
Epoch 49, Train Loss: 0.2315, Train Accuracy: 0.9348
        Epoch 50, Validation Loss: 35.8470, Validation Accuracy: 0.9354
Epoch 51, Train Loss: 0.2263, Train Accuracy: 0.9356
Epoch 52, Train Loss: 0.2237, Train Accuracy: 0.9364
Epoch 53, Train Loss: 0.2211, Train Accuracy: 0.9373
Epoch 54, Train Loss: 0.2188, Train Accuracy: 0.9379
Epoch 55, Train Loss: 0.2160, Train Accuracy: 0.9387
Epoch 56, Train Loss: 0.2136, Train Accuracy: 0.9394
Epoch 57, Train Loss: 0.2112, Train Accuracy: 0.9403
Epoch 58, Train Loss: 0.2087, Train Accuracy: 0.9410
Epoch 59, Train Loss: 0.2065, Train Accuracy: 0.9415
        Epoch 60, Validation Loss: 32.2866, Validation Accuracy: 0.9422
Epoch 61, Train Loss: 0.2018, Train Accuracy: 0.9429
Epoch 62, Train Loss: 0.1996, Train Accuracy: 0.9438
Epoch 63, Train Loss: 0.1974, Train Accuracy: 0.9439
Epoch 64, Train Loss: 0.1953, Train Accuracy: 0.9449
Epoch 65, Train Loss: 0.1932, Train Accuracy: 0.9453
Epoch 66, Train Loss: 0.1911, Train Accuracy: 0.9463
Epoch 67, Train Loss: 0.1889, Train Accuracy: 0.9468
Epoch 68, Train Loss: 0.1868, Train Accuracy: 0.9473
Epoch 69, Train Loss: 0.1849, Train Accuracy: 0.9479
        Epoch 70, Validation Loss: 29.3899, Validation Accuracy: 0.9469
Epoch 71, Train Loss: 0.1809, Train Accuracy: 0.9490
Epoch 72, Train Loss: 0.1790, Train Accuracy: 0.9494
Epoch 73, Train Loss: 0.1772, Train Accuracy: 0.9501
Epoch 74, Train Loss: 0.1754, Train Accuracy: 0.9504
Epoch 75, Train Loss: 0.1734, Train Accuracy: 0.9509
Epoch 76, Train Loss: 0.1718, Train Accuracy: 0.9517
Epoch 77, Train Loss: 0.1699, Train Accuracy: 0.9526
Epoch 78, Train Loss: 0.1683, Train Accuracy: 0.9527
Epoch 79, Train Loss: 0.1664, Train Accuracy: 0.9531
        Epoch 80, Validation Loss: 26.6688, Validation Accuracy: 0.9521
Epoch 81, Train Loss: 0.1630, Train Accuracy: 0.9544
Epoch 82, Train Loss: 0.1616, Train Accuracy: 0.9545
Epoch 83, Train Loss: 0.1599, Train Accuracy: 0.9554
Epoch 84, Train Loss: 0.1583, Train Accuracy: 0.9557
Epoch 85, Train Loss: 0.1568, Train Accuracy: 0.9563
Epoch 86, Train Loss: 0.1553, Train Accuracy: 0.9562
Epoch 87, Train Loss: 0.1540, Train Accuracy: 0.9570
Epoch 88, Train Loss: 0.1524, Train Accuracy: 0.9576
```

```
Epoch 89, Train Loss: 0.1510, Train Accuracy: 0.9577
       Epoch 90, Validation Loss: 24.5623, Validation Accuracy: 0.9563
Epoch 91, Train Loss: 0.1481, Train Accuracy: 0.9587
Epoch 92, Train Loss: 0.1469, Train Accuracy: 0.9590
Epoch 93, Train Loss: 0.1455, Train Accuracy: 0.9594
Epoch 94, Train Loss: 0.1441, Train Accuracy: 0.9599
Epoch 95, Train Loss: 0.1430, Train Accuracy: 0.9602
Epoch 96, Train Loss: 0.1416, Train Accuracy: 0.9604
Epoch 97, Train Loss: 0.1402, Train Accuracy: 0.9611
Epoch 98, Train Loss: 0.1390, Train Accuracy: 0.9612
Epoch 99, Train Loss: 0.1380, Train Accuracy: 0.9614
       Epoch 100, Validation Loss: 22.7509, Validation Accuracy: 0.9592
```






```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 4:43:29 ago. (Use '!kill␣
 ↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[19]:
```
(3990.2969510555267, 99710, 22.750947693828493, 0.9592)
```

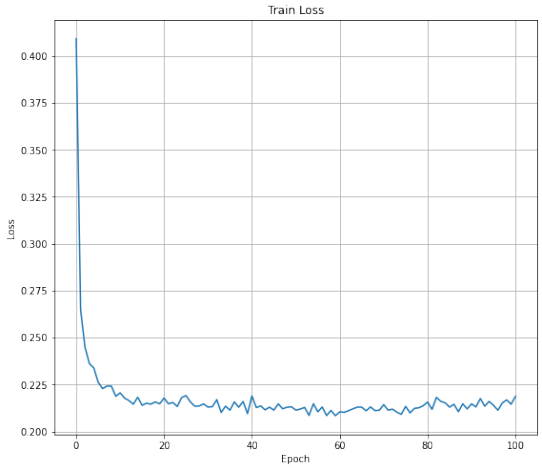
Pour la norme L2 de 10^-3 avec l'optimiseur = Adam

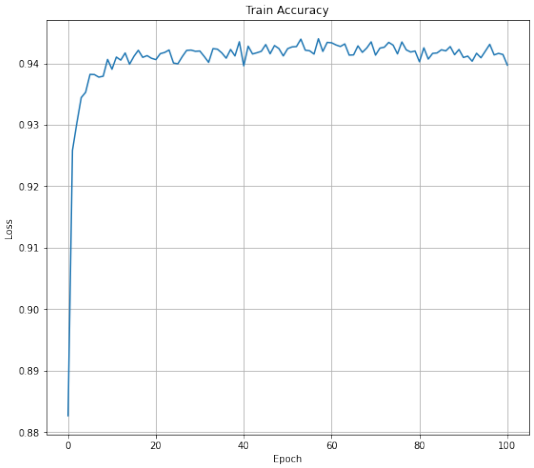
[20]:
```
print("\n__________________LA NORME L2 DE 10^-3 (Adam)__________________")
weight_decay = 10e-3
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-Adam-10e-3")
```

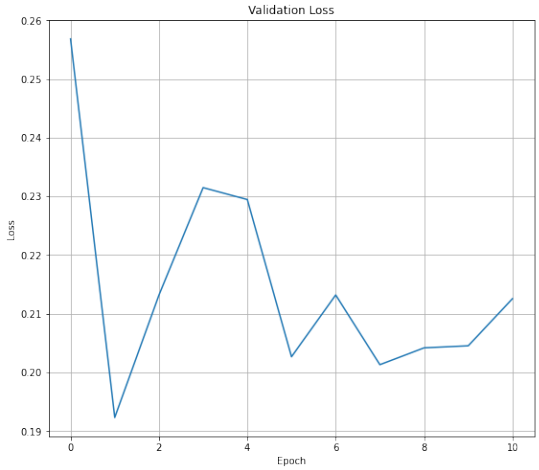
```
__________________LA NORME L2 DE 10^-3 (Adam)__________________
        Epoch 0, Validation Loss: 40.3238, Validation Accuracy: 0.9270
Epoch 1, Train Loss: 0.2649, Train Accuracy: 0.9258
Epoch 2, Train Loss: 0.2450, Train Accuracy: 0.9303
Epoch 3, Train Loss: 0.2362, Train Accuracy: 0.9344
Epoch 4, Train Loss: 0.2339, Train Accuracy: 0.9353
Epoch 5, Train Loss: 0.2263, Train Accuracy: 0.9382
Epoch 6, Train Loss: 0.2229, Train Accuracy: 0.9382
Epoch 7, Train Loss: 0.2242, Train Accuracy: 0.9378
Epoch 8, Train Loss: 0.2242, Train Accuracy: 0.9379
Epoch 9, Train Loss: 0.2188, Train Accuracy: 0.9406
        Epoch 10, Validation Loss: 30.1947, Validation Accuracy: 0.9476
Epoch 11, Train Loss: 0.2179, Train Accuracy: 0.9410
Epoch 12, Train Loss: 0.2165, Train Accuracy: 0.9405
Epoch 13, Train Loss: 0.2146, Train Accuracy: 0.9417
Epoch 14, Train Loss: 0.2183, Train Accuracy: 0.9399
Epoch 15, Train Loss: 0.2140, Train Accuracy: 0.9412
Epoch 16, Train Loss: 0.2151, Train Accuracy: 0.9421
Epoch 17, Train Loss: 0.2146, Train Accuracy: 0.9410
Epoch 18, Train Loss: 0.2158, Train Accuracy: 0.9413
Epoch 19, Train Loss: 0.2148, Train Accuracy: 0.9408
        Epoch 20, Validation Loss: 33.4680, Validation Accuracy: 0.9389
Epoch 21, Train Loss: 0.2148, Train Accuracy: 0.9416
Epoch 22, Train Loss: 0.2155, Train Accuracy: 0.9418
Epoch 23, Train Loss: 0.2134, Train Accuracy: 0.9422
Epoch 24, Train Loss: 0.2181, Train Accuracy: 0.9401
Epoch 25, Train Loss: 0.2192, Train Accuracy: 0.9399
Epoch 26, Train Loss: 0.2158, Train Accuracy: 0.9411
Epoch 27, Train Loss: 0.2135, Train Accuracy: 0.9421
Epoch 28, Train Loss: 0.2137, Train Accuracy: 0.9422
```

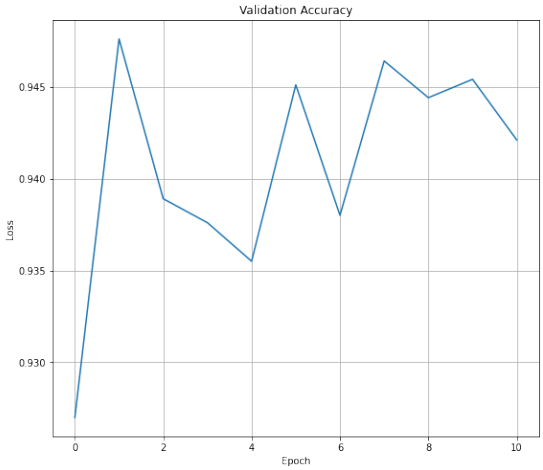
```
Epoch 29, Train Loss: 0.2148, Train Accuracy: 0.9420
        Epoch 30, Validation Loss: 36.3441, Validation Accuracy: 0.9376
Epoch 31, Train Loss: 0.2133, Train Accuracy: 0.9411
Epoch 32, Train Loss: 0.2170, Train Accuracy: 0.9402
Epoch 33, Train Loss: 0.2102, Train Accuracy: 0.9424
Epoch 34, Train Loss: 0.2135, Train Accuracy: 0.9423
Epoch 35, Train Loss: 0.2114, Train Accuracy: 0.9417
Epoch 36, Train Loss: 0.2158, Train Accuracy: 0.9408
Epoch 37, Train Loss: 0.2130, Train Accuracy: 0.9423
Epoch 38, Train Loss: 0.2160, Train Accuracy: 0.9412
Epoch 39, Train Loss: 0.2096, Train Accuracy: 0.9435
        Epoch 40, Validation Loss: 36.0263, Validation Accuracy: 0.9355
Epoch 41, Train Loss: 0.2128, Train Accuracy: 0.9428
Epoch 42, Train Loss: 0.2137, Train Accuracy: 0.9415
Epoch 43, Train Loss: 0.2116, Train Accuracy: 0.9417
Epoch 44, Train Loss: 0.2130, Train Accuracy: 0.9420
Epoch 45, Train Loss: 0.2114, Train Accuracy: 0.9431
Epoch 46, Train Loss: 0.2147, Train Accuracy: 0.9416
Epoch 47, Train Loss: 0.2122, Train Accuracy: 0.9429
Epoch 48, Train Loss: 0.2130, Train Accuracy: 0.9424
Epoch 49, Train Loss: 0.2132, Train Accuracy: 0.9412
        Epoch 50, Validation Loss: 31.8195, Validation Accuracy: 0.9451
Epoch 51, Train Loss: 0.2120, Train Accuracy: 0.9427
Epoch 52, Train Loss: 0.2129, Train Accuracy: 0.9427
Epoch 53, Train Loss: 0.2086, Train Accuracy: 0.9439
Epoch 54, Train Loss: 0.2148, Train Accuracy: 0.9422
Epoch 55, Train Loss: 0.2105, Train Accuracy: 0.9421
Epoch 56, Train Loss: 0.2131, Train Accuracy: 0.9415
Epoch 57, Train Loss: 0.2085, Train Accuracy: 0.9440
Epoch 58, Train Loss: 0.2113, Train Accuracy: 0.9419
Epoch 59, Train Loss: 0.2084, Train Accuracy: 0.9434
        Epoch 60, Validation Loss: 33.4720, Validation Accuracy: 0.9380
Epoch 61, Train Loss: 0.2103, Train Accuracy: 0.9430
Epoch 62, Train Loss: 0.2111, Train Accuracy: 0.9427
Epoch 63, Train Loss: 0.2121, Train Accuracy: 0.9432
Epoch 64, Train Loss: 0.2131, Train Accuracy: 0.9414
Epoch 65, Train Loss: 0.2130, Train Accuracy: 0.9414
Epoch 66, Train Loss: 0.2111, Train Accuracy: 0.9428
Epoch 67, Train Loss: 0.2131, Train Accuracy: 0.9418
Epoch 68, Train Loss: 0.2112, Train Accuracy: 0.9425
Epoch 69, Train Loss: 0.2113, Train Accuracy: 0.9435
        Epoch 70, Validation Loss: 31.6074, Validation Accuracy: 0.9464
Epoch 71, Train Loss: 0.2114, Train Accuracy: 0.9425
Epoch 72, Train Loss: 0.2119, Train Accuracy: 0.9426
Epoch 73, Train Loss: 0.2103, Train Accuracy: 0.9434
Epoch 74, Train Loss: 0.2092, Train Accuracy: 0.9430
Epoch 75, Train Loss: 0.2134, Train Accuracy: 0.9415
Epoch 76, Train Loss: 0.2100, Train Accuracy: 0.9435
```

```
Epoch 77, Train Loss: 0.2124, Train Accuracy: 0.9422
Epoch 78, Train Loss: 0.2127, Train Accuracy: 0.9418
Epoch 79, Train Loss: 0.2138, Train Accuracy: 0.9420
        Epoch 80, Validation Loss: 32.0578, Validation Accuracy: 0.9444
Epoch 81, Train Loss: 0.2118, Train Accuracy: 0.9425
Epoch 82, Train Loss: 0.2182, Train Accuracy: 0.9407
Epoch 83, Train Loss: 0.2161, Train Accuracy: 0.9416
Epoch 84, Train Loss: 0.2153, Train Accuracy: 0.9417
Epoch 85, Train Loss: 0.2130, Train Accuracy: 0.9422
Epoch 86, Train Loss: 0.2145, Train Accuracy: 0.9420
Epoch 87, Train Loss: 0.2106, Train Accuracy: 0.9427
Epoch 88, Train Loss: 0.2148, Train Accuracy: 0.9414
Epoch 89, Train Loss: 0.2121, Train Accuracy: 0.9423
        Epoch 90, Validation Loss: 32.1126, Validation Accuracy: 0.9454
Epoch 91, Train Loss: 0.2131, Train Accuracy: 0.9412
Epoch 92, Train Loss: 0.2176, Train Accuracy: 0.9404
Epoch 93, Train Loss: 0.2135, Train Accuracy: 0.9416
Epoch 94, Train Loss: 0.2161, Train Accuracy: 0.9409
Epoch 95, Train Loss: 0.2140, Train Accuracy: 0.9420
Epoch 96, Train Loss: 0.2114, Train Accuracy: 0.9431
Epoch 97, Train Loss: 0.2153, Train Accuracy: 0.9414
Epoch 98, Train Loss: 0.2169, Train Accuracy: 0.9416
Epoch 99, Train Loss: 0.2146, Train Accuracy: 0.9415
        Epoch 100, Validation Loss: 33.3727, Validation Accuracy: 0.9421
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 5:53:52 ago. (Use '!kill␣
 ↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[20]:
```
(4221.675123691559, 99710, 33.37267781235278, 0.9421)
```

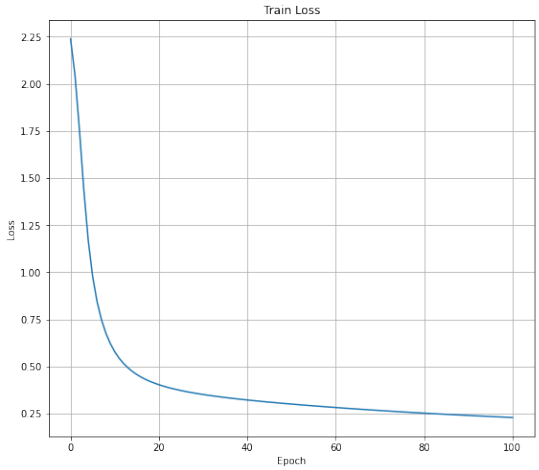
Pour la norme L2 de 10^-3 avec l'optimiseur = SGD

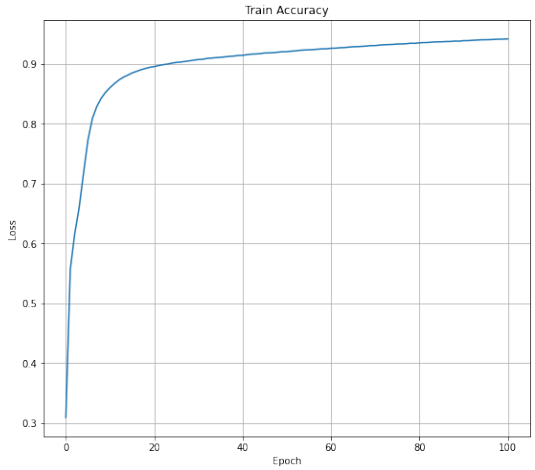
[21]:
```
print("\n__________________LA NORME L2 DE 10^-3 (SGD)__________________")
weight_decay = 10e-3
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-SGD-10e-3")
```

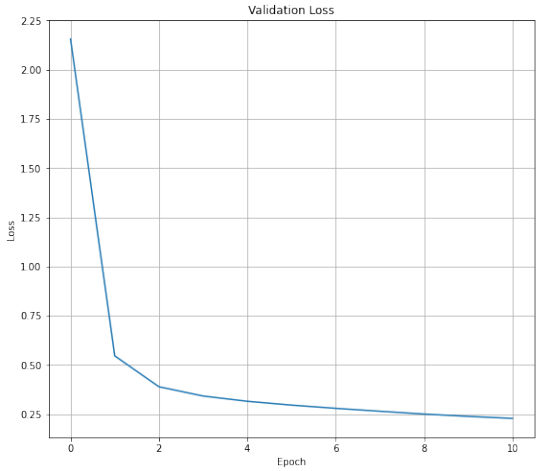
```
__________________LA NORME L2 DE 10^-3 (SGD)__________________
        Epoch 0, Validation Loss: 338.4955, Validation Accuracy: 0.5016
Epoch 1, Train Loss: 2.0473, Train Accuracy: 0.5565
Epoch 2, Train Loss: 1.7596, Train Accuracy: 0.6162
Epoch 3, Train Loss: 1.4382, Train Accuracy: 0.6593
Epoch 4, Train Loss: 1.1693, Train Accuracy: 0.7172
Epoch 5, Train Loss: 0.9782, Train Accuracy: 0.7729
Epoch 6, Train Loss: 0.8445, Train Accuracy: 0.8088
Epoch 7, Train Loss: 0.7482, Train Accuracy: 0.8287
Epoch 8, Train Loss: 0.6765, Train Accuracy: 0.8425
Epoch 9, Train Loss: 0.6217, Train Accuracy: 0.8527
```

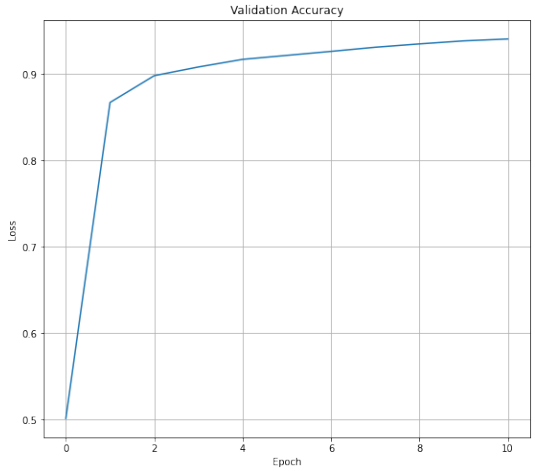
```
        Epoch 10, Validation Loss: 85.7222, Validation Accuracy: 0.8672
Epoch 11, Train Loss: 0.5442, Train Accuracy: 0.8671
Epoch 12, Train Loss: 0.5162, Train Accuracy: 0.8730
Epoch 13, Train Loss: 0.4931, Train Accuracy: 0.8777
Epoch 14, Train Loss: 0.4738, Train Accuracy: 0.8811
Epoch 15, Train Loss: 0.4576, Train Accuracy: 0.8849
Epoch 16, Train Loss: 0.4436, Train Accuracy: 0.8874
Epoch 17, Train Loss: 0.4315, Train Accuracy: 0.8901
Epoch 18, Train Loss: 0.4211, Train Accuracy: 0.8922
Epoch 19, Train Loss: 0.4119, Train Accuracy: 0.8942
        Epoch 20, Validation Loss: 61.0756, Validation Accuracy: 0.8982
Epoch 21, Train Loss: 0.3961, Train Accuracy: 0.8972
Epoch 22, Train Loss: 0.3894, Train Accuracy: 0.8986
Epoch 23, Train Loss: 0.3834, Train Accuracy: 0.8996
Epoch 24, Train Loss: 0.3779, Train Accuracy: 0.9012
Epoch 25, Train Loss: 0.3726, Train Accuracy: 0.9024
Epoch 26, Train Loss: 0.3677, Train Accuracy: 0.9030
Epoch 27, Train Loss: 0.3634, Train Accuracy: 0.9042
Epoch 28, Train Loss: 0.3593, Train Accuracy: 0.9051
Epoch 29, Train Loss: 0.3554, Train Accuracy: 0.9063
        Epoch 30, Validation Loss: 53.6965, Validation Accuracy: 0.9083
Epoch 31, Train Loss: 0.3482, Train Accuracy: 0.9077
Epoch 32, Train Loss: 0.3448, Train Accuracy: 0.9091
Epoch 33, Train Loss: 0.3417, Train Accuracy: 0.9097
Epoch 34, Train Loss: 0.3386, Train Accuracy: 0.9104
Epoch 35, Train Loss: 0.3357, Train Accuracy: 0.9109
Epoch 36, Train Loss: 0.3329, Train Accuracy: 0.9117
Epoch 37, Train Loss: 0.3303, Train Accuracy: 0.9125
Epoch 38, Train Loss: 0.3276, Train Accuracy: 0.9130
Epoch 39, Train Loss: 0.3249, Train Accuracy: 0.9140
        Epoch 40, Validation Loss: 49.5013, Validation Accuracy: 0.9171
Epoch 41, Train Loss: 0.3202, Train Accuracy: 0.9153
Epoch 42, Train Loss: 0.3180, Train Accuracy: 0.9162
Epoch 43, Train Loss: 0.3156, Train Accuracy: 0.9165
Epoch 44, Train Loss: 0.3133, Train Accuracy: 0.9171
Epoch 45, Train Loss: 0.3110, Train Accuracy: 0.9182
Epoch 46, Train Loss: 0.3092, Train Accuracy: 0.9183
Epoch 47, Train Loss: 0.3068, Train Accuracy: 0.9187
Epoch 48, Train Loss: 0.3047, Train Accuracy: 0.9195
Epoch 49, Train Loss: 0.3028, Train Accuracy: 0.9202
        Epoch 50, Validation Loss: 46.4254, Validation Accuracy: 0.9217
Epoch 51, Train Loss: 0.2989, Train Accuracy: 0.9212
Epoch 52, Train Loss: 0.2969, Train Accuracy: 0.9216
Epoch 53, Train Loss: 0.2950, Train Accuracy: 0.9226
Epoch 54, Train Loss: 0.2931, Train Accuracy: 0.9231
Epoch 55, Train Loss: 0.2913, Train Accuracy: 0.9234
Epoch 56, Train Loss: 0.2896, Train Accuracy: 0.9238
Epoch 57, Train Loss: 0.2877, Train Accuracy: 0.9244
```

```
Epoch 58, Train Loss: 0.2860, Train Accuracy: 0.9251
Epoch 59, Train Loss: 0.2842, Train Accuracy: 0.9251
       Epoch 60, Validation Loss: 43.8581, Validation Accuracy: 0.9263
Epoch 61, Train Loss: 0.2808, Train Accuracy: 0.9261
Epoch 62, Train Loss: 0.2791, Train Accuracy: 0.9269
Epoch 63, Train Loss: 0.2776, Train Accuracy: 0.9271
Epoch 64, Train Loss: 0.2759, Train Accuracy: 0.9278
Epoch 65, Train Loss: 0.2744, Train Accuracy: 0.9285
Epoch 66, Train Loss: 0.2727, Train Accuracy: 0.9286
Epoch 67, Train Loss: 0.2712, Train Accuracy: 0.9292
Epoch 68, Train Loss: 0.2696, Train Accuracy: 0.9297
Epoch 69, Train Loss: 0.2682, Train Accuracy: 0.9304
       Epoch 70, Validation Loss: 41.5334, Validation Accuracy: 0.9311
Epoch 71, Train Loss: 0.2652, Train Accuracy: 0.9313
Epoch 72, Train Loss: 0.2638, Train Accuracy: 0.9319
Epoch 73, Train Loss: 0.2623, Train Accuracy: 0.9322
Epoch 74, Train Loss: 0.2609, Train Accuracy: 0.9325
Epoch 75, Train Loss: 0.2593, Train Accuracy: 0.9331
Epoch 76, Train Loss: 0.2581, Train Accuracy: 0.9333
Epoch 77, Train Loss: 0.2567, Train Accuracy: 0.9336
Epoch 78, Train Loss: 0.2554, Train Accuracy: 0.9344
Epoch 79, Train Loss: 0.2541, Train Accuracy: 0.9343
       Epoch 80, Validation Loss: 39.3157, Validation Accuracy: 0.9350
Epoch 81, Train Loss: 0.2514, Train Accuracy: 0.9355
Epoch 82, Train Loss: 0.2502, Train Accuracy: 0.9357
Epoch 83, Train Loss: 0.2488, Train Accuracy: 0.9364
Epoch 84, Train Loss: 0.2476, Train Accuracy: 0.9367
Epoch 85, Train Loss: 0.2464, Train Accuracy: 0.9368
Epoch 86, Train Loss: 0.2451, Train Accuracy: 0.9373
Epoch 87, Train Loss: 0.2440, Train Accuracy: 0.9374
Epoch 88, Train Loss: 0.2428, Train Accuracy: 0.9381
Epoch 89, Train Loss: 0.2415, Train Accuracy: 0.9379
       Epoch 90, Validation Loss: 37.4416, Validation Accuracy: 0.9385
Epoch 91, Train Loss: 0.2392, Train Accuracy: 0.9388
Epoch 92, Train Loss: 0.2381, Train Accuracy: 0.9393
Epoch 93, Train Loss: 0.2372, Train Accuracy: 0.9397
Epoch 94, Train Loss: 0.2359, Train Accuracy: 0.9402
Epoch 95, Train Loss: 0.2348, Train Accuracy: 0.9403
Epoch 96, Train Loss: 0.2337, Train Accuracy: 0.9405
Epoch 97, Train Loss: 0.2328, Train Accuracy: 0.9408
Epoch 98, Train Loss: 0.2318, Train Accuracy: 0.9412
Epoch 99, Train Loss: 0.2306, Train Accuracy: 0.9413
       Epoch 100, Validation Loss: 35.8646, Validation Accuracy: 0.9407
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 7:00:29 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[21]:
```
(3995.626924276352, 99710, 35.864559477195144, 0.9407)
```

Pour la norme L2 de 0 avec l'optimiseur = Adam

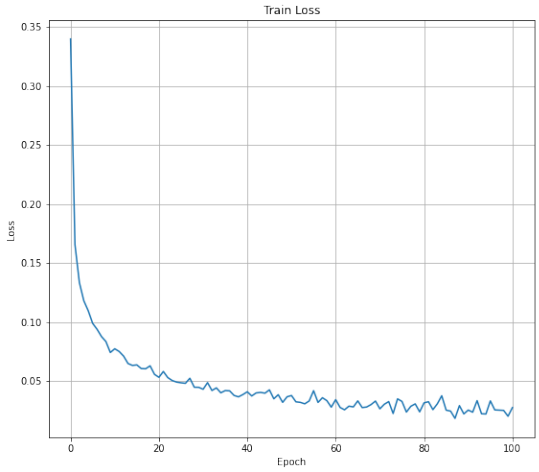
[22]:
```
print("\n__________________LA NORME L2 DE 0 (Adam)__________________")
weight_decay = 0
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
```

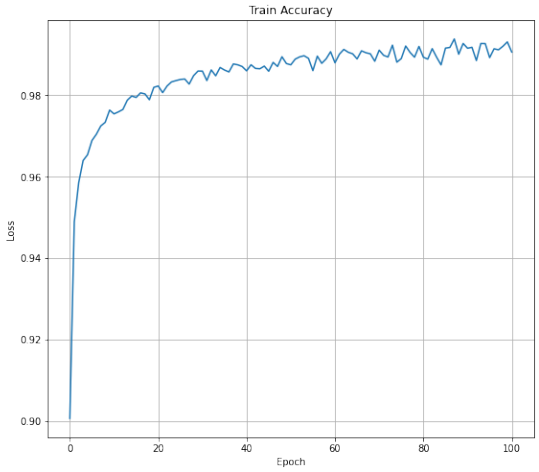
```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-Adam-10e-0")
```

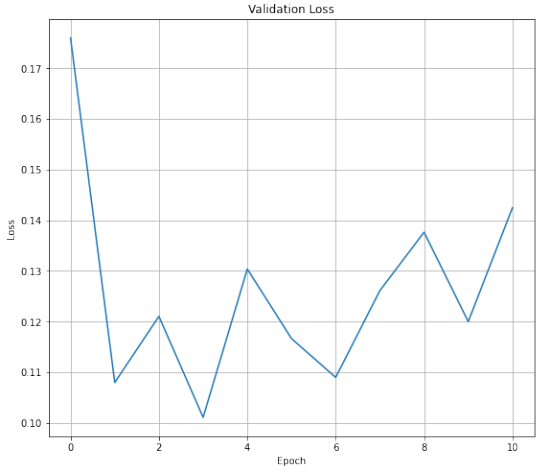
```
__________________LA NORME L2 DE 0 (Adam)__________________
        Epoch 0, Validation Loss: 27.6346, Validation Accuracy: 0.9473
Epoch 1, Train Loss: 0.1656, Train Accuracy: 0.9489
Epoch 2, Train Loss: 0.1334, Train Accuracy: 0.9586
Epoch 3, Train Loss: 0.1180, Train Accuracy: 0.9640
Epoch 4, Train Loss: 0.1096, Train Accuracy: 0.9654
Epoch 5, Train Loss: 0.0989, Train Accuracy: 0.9689
Epoch 6, Train Loss: 0.0939, Train Accuracy: 0.9705
Epoch 7, Train Loss: 0.0877, Train Accuracy: 0.9725
Epoch 8, Train Loss: 0.0835, Train Accuracy: 0.9734
Epoch 9, Train Loss: 0.0742, Train Accuracy: 0.9764
        Epoch 10, Validation Loss: 16.9506, Validation Accuracy: 0.9660
Epoch 11, Train Loss: 0.0751, Train Accuracy: 0.9759
Epoch 12, Train Loss: 0.0712, Train Accuracy: 0.9765
Epoch 13, Train Loss: 0.0649, Train Accuracy: 0.9788
Epoch 14, Train Loss: 0.0633, Train Accuracy: 0.9798
Epoch 15, Train Loss: 0.0637, Train Accuracy: 0.9795
Epoch 16, Train Loss: 0.0606, Train Accuracy: 0.9806
Epoch 17, Train Loss: 0.0604, Train Accuracy: 0.9804
Epoch 18, Train Loss: 0.0630, Train Accuracy: 0.9789
Epoch 19, Train Loss: 0.0557, Train Accuracy: 0.9820
        Epoch 20, Validation Loss: 19.0037, Validation Accuracy: 0.9643
Epoch 21, Train Loss: 0.0582, Train Accuracy: 0.9807
Epoch 22, Train Loss: 0.0530, Train Accuracy: 0.9823
Epoch 23, Train Loss: 0.0505, Train Accuracy: 0.9833
Epoch 24, Train Loss: 0.0492, Train Accuracy: 0.9836
Epoch 25, Train Loss: 0.0486, Train Accuracy: 0.9839
Epoch 26, Train Loss: 0.0481, Train Accuracy: 0.9840
Epoch 27, Train Loss: 0.0524, Train Accuracy: 0.9828
Epoch 28, Train Loss: 0.0448, Train Accuracy: 0.9848
Epoch 29, Train Loss: 0.0447, Train Accuracy: 0.9860
        Epoch 30, Validation Loss: 15.8706, Validation Accuracy: 0.9724
Epoch 31, Train Loss: 0.0487, Train Accuracy: 0.9836
Epoch 32, Train Loss: 0.0420, Train Accuracy: 0.9862
Epoch 33, Train Loss: 0.0443, Train Accuracy: 0.9848
Epoch 34, Train Loss: 0.0401, Train Accuracy: 0.9869
Epoch 35, Train Loss: 0.0420, Train Accuracy: 0.9862
Epoch 36, Train Loss: 0.0418, Train Accuracy: 0.9858
Epoch 37, Train Loss: 0.0379, Train Accuracy: 0.9877
Epoch 38, Train Loss: 0.0367, Train Accuracy: 0.9875
Epoch 39, Train Loss: 0.0386, Train Accuracy: 0.9871
        Epoch 40, Validation Loss: 20.4707, Validation Accuracy: 0.9660
```

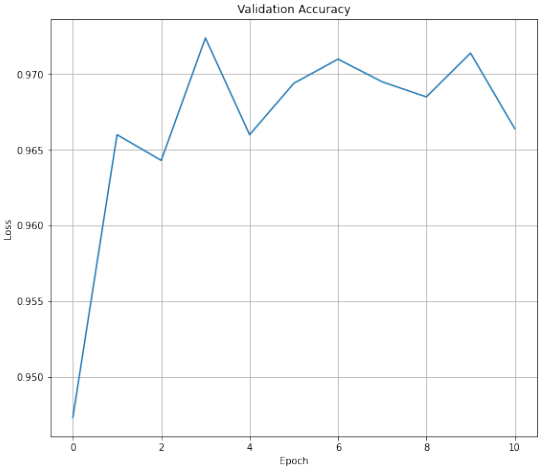
```
Epoch 41, Train Loss: 0.0374, Train Accuracy: 0.9875
Epoch 42, Train Loss: 0.0400, Train Accuracy: 0.9866
Epoch 43, Train Loss: 0.0404, Train Accuracy: 0.9865
Epoch 44, Train Loss: 0.0398, Train Accuracy: 0.9872
Epoch 45, Train Loss: 0.0426, Train Accuracy: 0.9859
Epoch 46, Train Loss: 0.0350, Train Accuracy: 0.9881
Epoch 47, Train Loss: 0.0385, Train Accuracy: 0.9871
Epoch 48, Train Loss: 0.0320, Train Accuracy: 0.9895
Epoch 49, Train Loss: 0.0367, Train Accuracy: 0.9878
        Epoch 50, Validation Loss: 18.3137, Validation Accuracy: 0.9694
Epoch 51, Train Loss: 0.0325, Train Accuracy: 0.9889
Epoch 52, Train Loss: 0.0320, Train Accuracy: 0.9895
Epoch 53, Train Loss: 0.0308, Train Accuracy: 0.9897
Epoch 54, Train Loss: 0.0332, Train Accuracy: 0.9891
Epoch 55, Train Loss: 0.0419, Train Accuracy: 0.9861
Epoch 56, Train Loss: 0.0320, Train Accuracy: 0.9896
Epoch 57, Train Loss: 0.0359, Train Accuracy: 0.9879
Epoch 58, Train Loss: 0.0335, Train Accuracy: 0.9890
Epoch 59, Train Loss: 0.0279, Train Accuracy: 0.9907
        Epoch 60, Validation Loss: 17.1101, Validation Accuracy: 0.9710
Epoch 61, Train Loss: 0.0277, Train Accuracy: 0.9901
Epoch 62, Train Loss: 0.0257, Train Accuracy: 0.9913
Epoch 63, Train Loss: 0.0288, Train Accuracy: 0.9906
Epoch 64, Train Loss: 0.0282, Train Accuracy: 0.9902
Epoch 65, Train Loss: 0.0333, Train Accuracy: 0.9889
Epoch 66, Train Loss: 0.0276, Train Accuracy: 0.9909
Epoch 67, Train Loss: 0.0281, Train Accuracy: 0.9905
Epoch 68, Train Loss: 0.0301, Train Accuracy: 0.9902
Epoch 69, Train Loss: 0.0331, Train Accuracy: 0.9884
        Epoch 70, Validation Loss: 19.8017, Validation Accuracy: 0.9695
Epoch 71, Train Loss: 0.0304, Train Accuracy: 0.9898
Epoch 72, Train Loss: 0.0327, Train Accuracy: 0.9894
Epoch 73, Train Loss: 0.0226, Train Accuracy: 0.9923
Epoch 74, Train Loss: 0.0350, Train Accuracy: 0.9881
Epoch 75, Train Loss: 0.0328, Train Accuracy: 0.9890
Epoch 76, Train Loss: 0.0238, Train Accuracy: 0.9921
Epoch 77, Train Loss: 0.0287, Train Accuracy: 0.9905
Epoch 78, Train Loss: 0.0307, Train Accuracy: 0.9894
Epoch 79, Train Loss: 0.0240, Train Accuracy: 0.9920
        Epoch 80, Validation Loss: 21.6122, Validation Accuracy: 0.9685
Epoch 81, Train Loss: 0.0326, Train Accuracy: 0.9889
Epoch 82, Train Loss: 0.0258, Train Accuracy: 0.9915
Epoch 83, Train Loss: 0.0307, Train Accuracy: 0.9894
Epoch 84, Train Loss: 0.0376, Train Accuracy: 0.9875
Epoch 85, Train Loss: 0.0254, Train Accuracy: 0.9916
Epoch 86, Train Loss: 0.0246, Train Accuracy: 0.9918
Epoch 87, Train Loss: 0.0185, Train Accuracy: 0.9939
Epoch 88, Train Loss: 0.0292, Train Accuracy: 0.9901
```

```
Epoch 89, Train Loss: 0.0222, Train Accuracy: 0.9928
       Epoch 90, Validation Loss: 18.8389, Validation Accuracy: 0.9714
Epoch 91, Train Loss: 0.0237, Train Accuracy: 0.9918
Epoch 92, Train Loss: 0.0335, Train Accuracy: 0.9886
Epoch 93, Train Loss: 0.0224, Train Accuracy: 0.9927
Epoch 94, Train Loss: 0.0221, Train Accuracy: 0.9927
Epoch 95, Train Loss: 0.0333, Train Accuracy: 0.9893
Epoch 96, Train Loss: 0.0256, Train Accuracy: 0.9914
Epoch 97, Train Loss: 0.0254, Train Accuracy: 0.9912
Epoch 98, Train Loss: 0.0252, Train Accuracy: 0.9920
Epoch 99, Train Loss: 0.0203, Train Accuracy: 0.9931
       Epoch 100, Validation Loss: 22.3710, Validation Accuracy: 0.9664
```










```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 8:10:28 ago. (Use '!kill␣
 ↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[22]:
```
(4197.075489282608, 99710, 22.37097011674632, 0.9664)
```

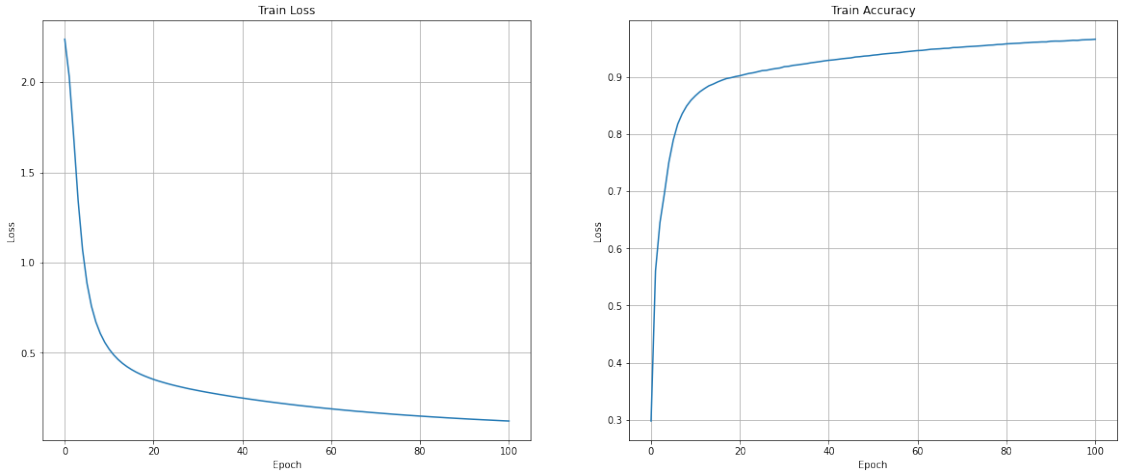
Pour la norme L2 de 0 avec l'optimiseur = SGD

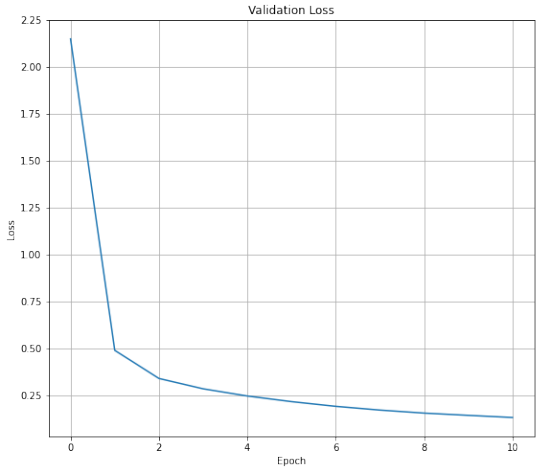
[23]:
```
print("\n__________________LA NORME L2 DE 0 (SGD)__________________")
weight_decay = 0
model = LinearMultiClass(in_size, out_size, hidden_layers, activation=nn.Tanh())
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,␣
 ↪weight_decay=weight_decay)
train(model, train_loader, validation_loader, loss_fn, optimizer, epochs=100,␣
 ↪typeTrain="L2-SGD-10e-0")
```

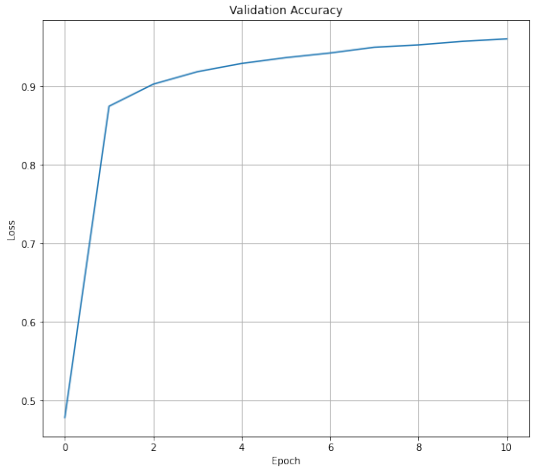
```
__________________LA NORME L2 DE 0 (SGD)__________________
        Epoch 0, Validation Loss: 337.5311, Validation Accuracy: 0.4783
Epoch 1, Train Loss: 2.0308, Train Accuracy: 0.5606
Epoch 2, Train Loss: 1.6988, Train Accuracy: 0.6451
Epoch 3, Train Loss: 1.3459, Train Accuracy: 0.6959
Epoch 4, Train Loss: 1.0729, Train Accuracy: 0.7507
Epoch 5, Train Loss: 0.8858, Train Accuracy: 0.7898
Epoch 6, Train Loss: 0.7592, Train Accuracy: 0.8177
Epoch 7, Train Loss: 0.6711, Train Accuracy: 0.8356
Epoch 8, Train Loss: 0.6073, Train Accuracy: 0.8492
Epoch 9, Train Loss: 0.5586, Train Accuracy: 0.8595
        Epoch 10, Validation Loss: 76.9736, Validation Accuracy: 0.8747
Epoch 11, Train Loss: 0.4892, Train Accuracy: 0.8743
Epoch 12, Train Loss: 0.4634, Train Accuracy: 0.8797
Epoch 13, Train Loss: 0.4419, Train Accuracy: 0.8846
Epoch 14, Train Loss: 0.4237, Train Accuracy: 0.8878
Epoch 15, Train Loss: 0.4080, Train Accuracy: 0.8915
Epoch 16, Train Loss: 0.3942, Train Accuracy: 0.8946
Epoch 17, Train Loss: 0.3821, Train Accuracy: 0.8974
Epoch 18, Train Loss: 0.3715, Train Accuracy: 0.8989
Epoch 19, Train Loss: 0.3618, Train Accuracy: 0.9009
        Epoch 20, Validation Loss: 53.4486, Validation Accuracy: 0.9028
Epoch 21, Train Loss: 0.3448, Train Accuracy: 0.9043
Epoch 22, Train Loss: 0.3375, Train Accuracy: 0.9063
Epoch 23, Train Loss: 0.3304, Train Accuracy: 0.9075
Epoch 24, Train Loss: 0.3239, Train Accuracy: 0.9093
Epoch 25, Train Loss: 0.3177, Train Accuracy: 0.9112
Epoch 26, Train Loss: 0.3119, Train Accuracy: 0.9118
Epoch 27, Train Loss: 0.3063, Train Accuracy: 0.9135
Epoch 28, Train Loss: 0.3009, Train Accuracy: 0.9148
```

```
Epoch 29, Train Loss: 0.2960, Train Accuracy: 0.9159
        Epoch 30, Validation Loss: 44.7756, Validation Accuracy: 0.9187
Epoch 31, Train Loss: 0.2863, Train Accuracy: 0.9188
Epoch 32, Train Loss: 0.2818, Train Accuracy: 0.9205
Epoch 33, Train Loss: 0.2774, Train Accuracy: 0.9214
Epoch 34, Train Loss: 0.2732, Train Accuracy: 0.9225
Epoch 35, Train Loss: 0.2689, Train Accuracy: 0.9235
Epoch 36, Train Loss: 0.2649, Train Accuracy: 0.9250
Epoch 37, Train Loss: 0.2608, Train Accuracy: 0.9260
Epoch 38, Train Loss: 0.2569, Train Accuracy: 0.9270
Epoch 39, Train Loss: 0.2530, Train Accuracy: 0.9283
        Epoch 40, Validation Loss: 38.7679, Validation Accuracy: 0.9292
Epoch 41, Train Loss: 0.2459, Train Accuracy: 0.9300
Epoch 42, Train Loss: 0.2423, Train Accuracy: 0.9310
Epoch 43, Train Loss: 0.2390, Train Accuracy: 0.9320
Epoch 44, Train Loss: 0.2356, Train Accuracy: 0.9328
Epoch 45, Train Loss: 0.2322, Train Accuracy: 0.9335
Epoch 46, Train Loss: 0.2290, Train Accuracy: 0.9351
Epoch 47, Train Loss: 0.2257, Train Accuracy: 0.9357
Epoch 48, Train Loss: 0.2226, Train Accuracy: 0.9367
Epoch 49, Train Loss: 0.2196, Train Accuracy: 0.9372
        Epoch 50, Validation Loss: 34.0623, Validation Accuracy: 0.9366
Epoch 51, Train Loss: 0.2137, Train Accuracy: 0.9391
Epoch 52, Train Loss: 0.2108, Train Accuracy: 0.9402
Epoch 53, Train Loss: 0.2078, Train Accuracy: 0.9409
Epoch 54, Train Loss: 0.2052, Train Accuracy: 0.9416
Epoch 55, Train Loss: 0.2026, Train Accuracy: 0.9423
Epoch 56, Train Loss: 0.1999, Train Accuracy: 0.9429
Epoch 57, Train Loss: 0.1973, Train Accuracy: 0.9440
Epoch 58, Train Loss: 0.1947, Train Accuracy: 0.9448
Epoch 59, Train Loss: 0.1922, Train Accuracy: 0.9456
        Epoch 60, Validation Loss: 30.1150, Validation Accuracy: 0.9425
Epoch 61, Train Loss: 0.1874, Train Accuracy: 0.9467
Epoch 62, Train Loss: 0.1849, Train Accuracy: 0.9476
Epoch 63, Train Loss: 0.1826, Train Accuracy: 0.9486
Epoch 64, Train Loss: 0.1805, Train Accuracy: 0.9491
Epoch 65, Train Loss: 0.1781, Train Accuracy: 0.9495
Epoch 66, Train Loss: 0.1757, Train Accuracy: 0.9502
Epoch 67, Train Loss: 0.1741, Train Accuracy: 0.9504
Epoch 68, Train Loss: 0.1718, Train Accuracy: 0.9517
Epoch 69, Train Loss: 0.1697, Train Accuracy: 0.9519
        Epoch 70, Validation Loss: 26.9710, Validation Accuracy: 0.9498
Epoch 71, Train Loss: 0.1658, Train Accuracy: 0.9531
Epoch 72, Train Loss: 0.1639, Train Accuracy: 0.9536
Epoch 73, Train Loss: 0.1619, Train Accuracy: 0.9540
Epoch 74, Train Loss: 0.1599, Train Accuracy: 0.9545
Epoch 75, Train Loss: 0.1581, Train Accuracy: 0.9551
Epoch 76, Train Loss: 0.1564, Train Accuracy: 0.9558
```

```
Epoch 77, Train Loss: 0.1546, Train Accuracy: 0.9562
Epoch 78, Train Loss: 0.1530, Train Accuracy: 0.9570
Epoch 79, Train Loss: 0.1513, Train Accuracy: 0.9573
        Epoch 80, Validation Loss: 24.4470, Validation Accuracy: 0.9528
Epoch 81, Train Loss: 0.1481, Train Accuracy: 0.9586
Epoch 82, Train Loss: 0.1464, Train Accuracy: 0.9590
Epoch 83, Train Loss: 0.1449, Train Accuracy: 0.9593
Epoch 84, Train Loss: 0.1434, Train Accuracy: 0.9600
Epoch 85, Train Loss: 0.1419, Train Accuracy: 0.9603
Epoch 86, Train Loss: 0.1403, Train Accuracy: 0.9609
Epoch 87, Train Loss: 0.1389, Train Accuracy: 0.9611
Epoch 88, Train Loss: 0.1375, Train Accuracy: 0.9616
Epoch 89, Train Loss: 0.1360, Train Accuracy: 0.9616
        Epoch 90, Validation Loss: 22.5742, Validation Accuracy: 0.9574
Epoch 91, Train Loss: 0.1333, Train Accuracy: 0.9630
Epoch 92, Train Loss: 0.1320, Train Accuracy: 0.9629
Epoch 93, Train Loss: 0.1308, Train Accuracy: 0.9632
Epoch 94, Train Loss: 0.1295, Train Accuracy: 0.9637
Epoch 95, Train Loss: 0.1283, Train Accuracy: 0.9644
Epoch 96, Train Loss: 0.1271, Train Accuracy: 0.9642
Epoch 97, Train Loss: 0.1258, Train Accuracy: 0.9651
Epoch 98, Train Loss: 0.1246, Train Accuracy: 0.9654
Epoch 99, Train Loss: 0.1234, Train Accuracy: 0.9656
        Epoch 100, Validation Loss: 20.7893, Validation Accuracy: 0.9605
```








```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 5463), started 9:16:27 ago. (Use '!kill␣
↪5463' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[23]:
```
(3958.3722851276398, 99710, 20.789288378786296, 0.9605)
```

Conclusion : Nous pouvons remarquer que lorsque nous utilisons l'optimiseur Adam, nous avons de nombreuses variations au niveau des courbes de pertes. Ce qui nous prouve que l'optimiseur SGD est le meilleur optimiseur lorsque nous appliquons la pénalisation des couches sur nos fonctions d'entrainement.

De plus, on ne peut pas determiner quelle est la norme la mieux adaptée de manière générale car cela dépend de nombreux facteurs tels que la taille et la qualité des données d'entraînement, la structure et la complexité du modèle, etc.

Mais en général, plus la valeur de la norme L2 est faible, plus le modèle sera pénalisé pour les poids importants. Ainsi, une valeur plus faible peut conduire à une régularisation plus forte et donc à un modèle plus simple et moins sujet au sur-apprentissage. Cependant, une régularisation trop forte peut entraîner un sous-apprentissage et une performance dégradée. D'un autre côté, une valeur plus élevée peut donner lieu à une régularisation plus faible, permettant ainsi au modèle de conserver davantage d'informations provenant des données d'entraînement. Cependant, une régularisation insuffisante peut entraîner un sur-apprentissage et une performance dégradée sur les données de validation ou de test.

7.2 Modèle LinearMultiClassWithDropout

Une autre technique très utilisée est le **Dropout** . L'idée du Dropout est proche du moyennage de modèle : en entraînant k modèles de manière indépendante, on réduit la variance du modèle. Entraîner k modèles présente un surcoût non négligeable, et l'intérêt du Dropout est de réduire la complexité mémoire/temps de calcul. Le Dropout consiste à chaque itération à *geler* certains

neurones aléatoirement dans le réseau en fixant leur sortie à zéro. Cela a pour conséquence de rendre plus robuste le réseau.

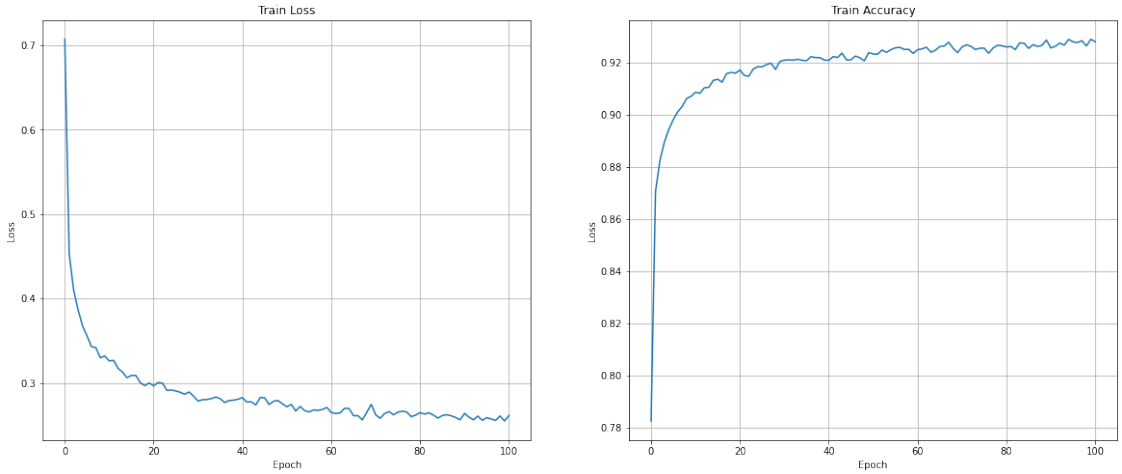
7.2.1 En utilisant l'optimiseur = Adam

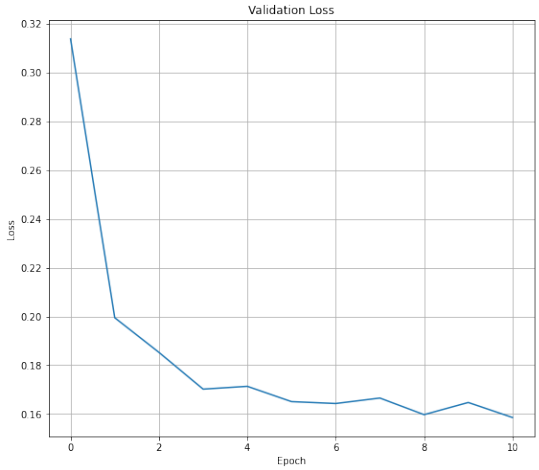
[24]:
```
dropout_rate = 0.5
model = LinearMultiClassWithDropout(in_size, out_size, hidden_layers,␣
 ↪dropout_rate)

# Define our loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Train the model Dropout
model2ExecutionTimeAdam, model2ComplexityAdam, model2ValidationLossAdam,␣
 ↪model2AccuracyAdam = train_dropout(model, train_loader, validation_loader,␣
 ↪loss_fn, optimizer, epochs=100, dropout_rate=dropout_rate,␣
 ↪typeTrain="Dropout-Adam")
```

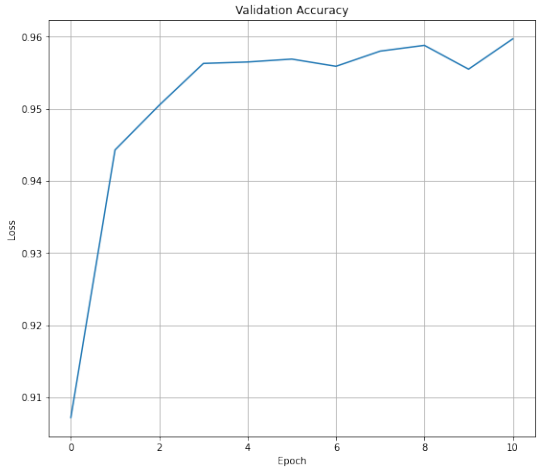
```
        Epoch 0, Validation Loss: 49.2806, Validation Accuracy: 0.9072
Epoch 1, Train Loss: 0.4526, Train Accuracy: 0.8706
Epoch 2, Train Loss: 0.4097, Train Accuracy: 0.8825
Epoch 3, Train Loss: 0.3864, Train Accuracy: 0.8895
Epoch 4, Train Loss: 0.3676, Train Accuracy: 0.8944
Epoch 5, Train Loss: 0.3558, Train Accuracy: 0.8981
Epoch 6, Train Loss: 0.3430, Train Accuracy: 0.9012
Epoch 7, Train Loss: 0.3418, Train Accuracy: 0.9032
Epoch 8, Train Loss: 0.3298, Train Accuracy: 0.9062
Epoch 9, Train Loss: 0.3319, Train Accuracy: 0.9071
        Epoch 10, Validation Loss: 31.3165, Validation Accuracy: 0.9443
Epoch 11, Train Loss: 0.3267, Train Accuracy: 0.9083
Epoch 12, Train Loss: 0.3173, Train Accuracy: 0.9104
Epoch 13, Train Loss: 0.3128, Train Accuracy: 0.9105
Epoch 14, Train Loss: 0.3061, Train Accuracy: 0.9132
Epoch 15, Train Loss: 0.3088, Train Accuracy: 0.9136
Epoch 16, Train Loss: 0.3088, Train Accuracy: 0.9125
Epoch 17, Train Loss: 0.3002, Train Accuracy: 0.9158
Epoch 18, Train Loss: 0.2967, Train Accuracy: 0.9163
Epoch 19, Train Loss: 0.3000, Train Accuracy: 0.9160
        Epoch 20, Validation Loss: 29.0858, Validation Accuracy: 0.9505
Epoch 21, Train Loss: 0.3004, Train Accuracy: 0.9151
Epoch 22, Train Loss: 0.2997, Train Accuracy: 0.9148
Epoch 23, Train Loss: 0.2911, Train Accuracy: 0.9176
Epoch 24, Train Loss: 0.2916, Train Accuracy: 0.9185
Epoch 25, Train Loss: 0.2904, Train Accuracy: 0.9184
Epoch 26, Train Loss: 0.2889, Train Accuracy: 0.9193
Epoch 27, Train Loss: 0.2865, Train Accuracy: 0.9198
Epoch 28, Train Loss: 0.2893, Train Accuracy: 0.9174
```

```
Epoch 29, Train Loss: 0.2843, Train Accuracy: 0.9204
        Epoch 30, Validation Loss: 26.7170, Validation Accuracy: 0.9563
Epoch 31, Train Loss: 0.2801, Train Accuracy: 0.9211
Epoch 32, Train Loss: 0.2803, Train Accuracy: 0.9210
Epoch 33, Train Loss: 0.2814, Train Accuracy: 0.9213
Epoch 34, Train Loss: 0.2832, Train Accuracy: 0.9209
Epoch 35, Train Loss: 0.2813, Train Accuracy: 0.9207
Epoch 36, Train Loss: 0.2768, Train Accuracy: 0.9223
Epoch 37, Train Loss: 0.2792, Train Accuracy: 0.9220
Epoch 38, Train Loss: 0.2795, Train Accuracy: 0.9219
Epoch 39, Train Loss: 0.2806, Train Accuracy: 0.9210
        Epoch 40, Validation Loss: 26.8981, Validation Accuracy: 0.9565
Epoch 41, Train Loss: 0.2773, Train Accuracy: 0.9223
Epoch 42, Train Loss: 0.2776, Train Accuracy: 0.9220
Epoch 43, Train Loss: 0.2739, Train Accuracy: 0.9237
Epoch 44, Train Loss: 0.2827, Train Accuracy: 0.9210
Epoch 45, Train Loss: 0.2824, Train Accuracy: 0.9211
Epoch 46, Train Loss: 0.2744, Train Accuracy: 0.9225
Epoch 47, Train Loss: 0.2782, Train Accuracy: 0.9219
Epoch 48, Train Loss: 0.2791, Train Accuracy: 0.9207
Epoch 49, Train Loss: 0.2752, Train Accuracy: 0.9239
        Epoch 50, Validation Loss: 25.9143, Validation Accuracy: 0.9569
Epoch 51, Train Loss: 0.2745, Train Accuracy: 0.9232
Epoch 52, Train Loss: 0.2667, Train Accuracy: 0.9248
Epoch 53, Train Loss: 0.2721, Train Accuracy: 0.9240
Epoch 54, Train Loss: 0.2672, Train Accuracy: 0.9250
Epoch 55, Train Loss: 0.2656, Train Accuracy: 0.9257
Epoch 56, Train Loss: 0.2680, Train Accuracy: 0.9259
Epoch 57, Train Loss: 0.2675, Train Accuracy: 0.9251
Epoch 58, Train Loss: 0.2685, Train Accuracy: 0.9251
Epoch 59, Train Loss: 0.2709, Train Accuracy: 0.9236
        Epoch 60, Validation Loss: 25.7919, Validation Accuracy: 0.9559
Epoch 61, Train Loss: 0.2637, Train Accuracy: 0.9253
Epoch 62, Train Loss: 0.2645, Train Accuracy: 0.9259
Epoch 63, Train Loss: 0.2699, Train Accuracy: 0.9240
Epoch 64, Train Loss: 0.2697, Train Accuracy: 0.9247
Epoch 65, Train Loss: 0.2611, Train Accuracy: 0.9262
Epoch 66, Train Loss: 0.2612, Train Accuracy: 0.9264
Epoch 67, Train Loss: 0.2563, Train Accuracy: 0.9278
Epoch 68, Train Loss: 0.2652, Train Accuracy: 0.9255
Epoch 69, Train Loss: 0.2745, Train Accuracy: 0.9238
        Epoch 70, Validation Loss: 26.1446, Validation Accuracy: 0.9580
Epoch 71, Train Loss: 0.2580, Train Accuracy: 0.9268
Epoch 72, Train Loss: 0.2637, Train Accuracy: 0.9263
Epoch 73, Train Loss: 0.2661, Train Accuracy: 0.9251
Epoch 74, Train Loss: 0.2623, Train Accuracy: 0.9255
Epoch 75, Train Loss: 0.2653, Train Accuracy: 0.9256
Epoch 76, Train Loss: 0.2665, Train Accuracy: 0.9236
```

```
Epoch 77, Train Loss: 0.2655, Train Accuracy: 0.9256
Epoch 78, Train Loss: 0.2600, Train Accuracy: 0.9267
Epoch 79, Train Loss: 0.2619, Train Accuracy: 0.9265
        Epoch 80, Validation Loss: 25.0730, Validation Accuracy: 0.9588
Epoch 81, Train Loss: 0.2630, Train Accuracy: 0.9262
Epoch 82, Train Loss: 0.2646, Train Accuracy: 0.9250
Epoch 83, Train Loss: 0.2618, Train Accuracy: 0.9276
Epoch 84, Train Loss: 0.2584, Train Accuracy: 0.9274
Epoch 85, Train Loss: 0.2613, Train Accuracy: 0.9255
Epoch 86, Train Loss: 0.2623, Train Accuracy: 0.9269
Epoch 87, Train Loss: 0.2611, Train Accuracy: 0.9262
Epoch 88, Train Loss: 0.2590, Train Accuracy: 0.9266
Epoch 89, Train Loss: 0.2562, Train Accuracy: 0.9287
        Epoch 90, Validation Loss: 25.8583, Validation Accuracy: 0.9555
Epoch 91, Train Loss: 0.2594, Train Accuracy: 0.9263
Epoch 92, Train Loss: 0.2563, Train Accuracy: 0.9275
Epoch 93, Train Loss: 0.2607, Train Accuracy: 0.9267
Epoch 94, Train Loss: 0.2556, Train Accuracy: 0.9289
Epoch 95, Train Loss: 0.2589, Train Accuracy: 0.9280
Epoch 96, Train Loss: 0.2573, Train Accuracy: 0.9277
Epoch 97, Train Loss: 0.2556, Train Accuracy: 0.9284
Epoch 98, Train Loss: 0.2608, Train Accuracy: 0.9264
Epoch 99, Train Loss: 0.2548, Train Accuracy: 0.9290
        Epoch 100, Validation Loss: 24.8911, Validation Accuracy: 0.9597
```








```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
<IPython.core.display.HTML object>
```

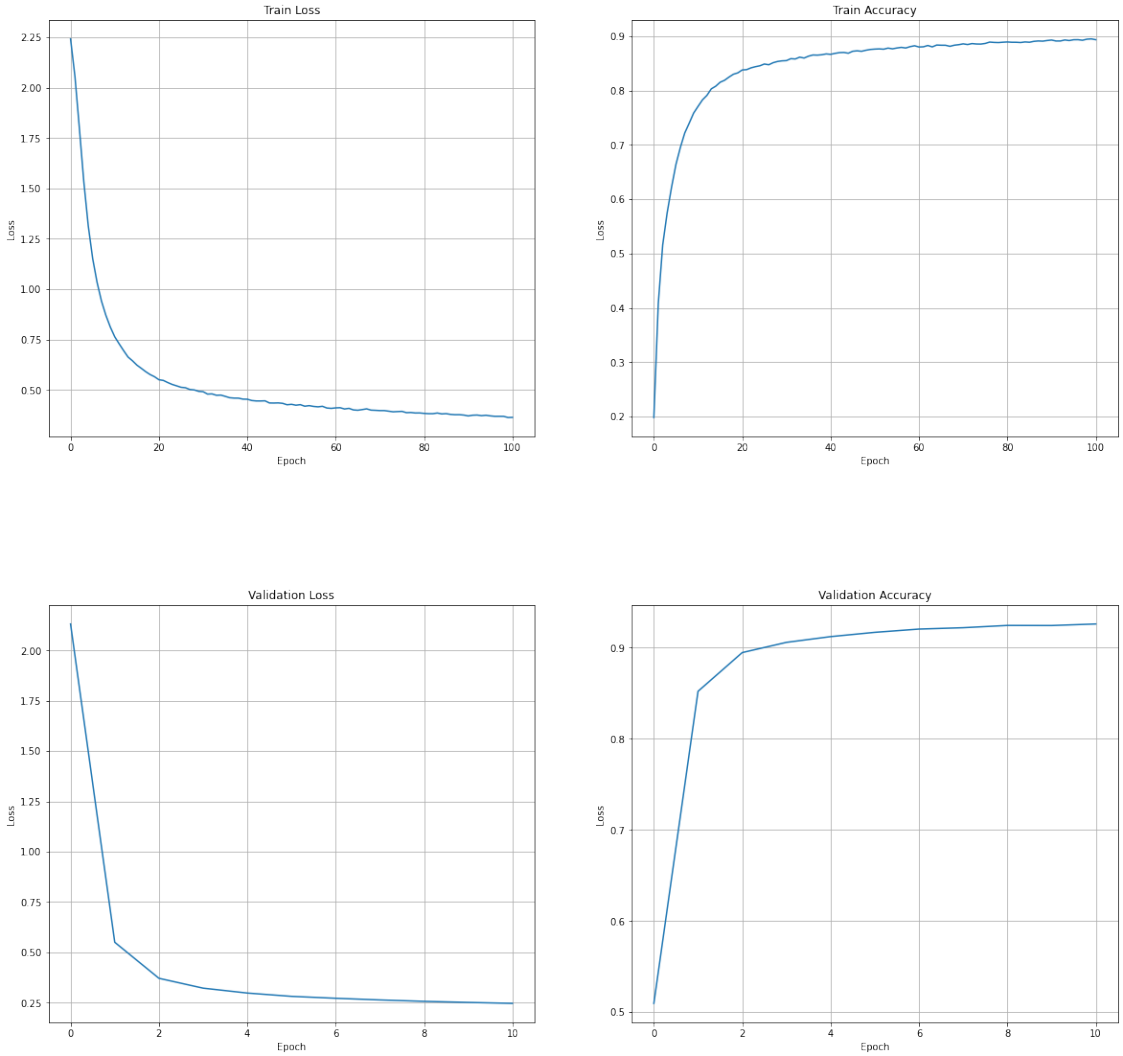
7.2.2 En utilisant l'optimiseur = SGD

[25]:
```
num_workersnum_workersdropout_rate = 0.5
model = LinearMultiClassWithDropout(in_size, out_size, hidden_layers,␣
 ↪dropout_rate)

# Define our loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

# Train the model Dropout
model2ExecutionTimeSGD, model2ComplexitySGD, model2ValidationLossSGD,␣
 ↪model2AccuracySGD = train_dropout(model, train_loader, validation_loader,␣
 ↪loss_fn, optimizer, epochs=100, dropout_rate=dropout_rate,␣
 ↪typeTrain="Dropout-SGD")
```

```
        Epoch 0, Validation Loss: 334.6284, Validation Accuracy: 0.5096
Epoch 1, Train Loss: 2.0551, Train Accuracy: 0.4081
Epoch 2, Train Loss: 1.7984, Train Accuracy: 0.5159
Epoch 3, Train Loss: 1.5331, Train Accuracy: 0.5744
Epoch 4, Train Loss: 1.3151, Train Accuracy: 0.6219
Epoch 5, Train Loss: 1.1530, Train Accuracy: 0.6642
Epoch 6, Train Loss: 1.0341, Train Accuracy: 0.6955
Epoch 7, Train Loss: 0.9408, Train Accuracy: 0.7222
Epoch 8, Train Loss: 0.8699, Train Accuracy: 0.7400
Epoch 9, Train Loss: 0.8118, Train Accuracy: 0.7585
        Epoch 10, Validation Loss: 86.3392, Validation Accuracy: 0.8522
```

```
Epoch 11, Train Loss: 0.7287, Train Accuracy: 0.7829
Epoch 12, Train Loss: 0.6954, Train Accuracy: 0.7911
Epoch 13, Train Loss: 0.6636, Train Accuracy: 0.8033
Epoch 14, Train Loss: 0.6448, Train Accuracy: 0.8081
Epoch 15, Train Loss: 0.6234, Train Accuracy: 0.8154
Epoch 16, Train Loss: 0.6073, Train Accuracy: 0.8190
Epoch 17, Train Loss: 0.5907, Train Accuracy: 0.8249
Epoch 18, Train Loss: 0.5764, Train Accuracy: 0.8301
Epoch 19, Train Loss: 0.5655, Train Accuracy: 0.8326
        Epoch 20, Validation Loss: 58.2361, Validation Accuracy: 0.8947
Epoch 21, Train Loss: 0.5472, Train Accuracy: 0.8386
Epoch 22, Train Loss: 0.5366, Train Accuracy: 0.8419
Epoch 23, Train Loss: 0.5272, Train Accuracy: 0.8440
Epoch 24, Train Loss: 0.5202, Train Accuracy: 0.8457
Epoch 25, Train Loss: 0.5130, Train Accuracy: 0.8489
Epoch 26, Train Loss: 0.5108, Train Accuracy: 0.8476
Epoch 27, Train Loss: 0.5021, Train Accuracy: 0.8514
Epoch 28, Train Loss: 0.4999, Train Accuracy: 0.8536
Epoch 29, Train Loss: 0.4929, Train Accuracy: 0.8548
        Epoch 30, Validation Loss: 50.5385, Validation Accuracy: 0.9059
Epoch 31, Train Loss: 0.4794, Train Accuracy: 0.8588
Epoch 32, Train Loss: 0.4804, Train Accuracy: 0.8582
Epoch 33, Train Loss: 0.4735, Train Accuracy: 0.8616
Epoch 34, Train Loss: 0.4744, Train Accuracy: 0.8599
Epoch 35, Train Loss: 0.4682, Train Accuracy: 0.8633
Epoch 36, Train Loss: 0.4614, Train Accuracy: 0.8656
Epoch 37, Train Loss: 0.4595, Train Accuracy: 0.8652
Epoch 38, Train Loss: 0.4592, Train Accuracy: 0.8660
Epoch 39, Train Loss: 0.4544, Train Accuracy: 0.8675
        Epoch 40, Validation Loss: 46.7193, Validation Accuracy: 0.9122
Epoch 41, Train Loss: 0.4475, Train Accuracy: 0.8685
Epoch 42, Train Loss: 0.4453, Train Accuracy: 0.8699
Epoch 43, Train Loss: 0.4452, Train Accuracy: 0.8702
Epoch 44, Train Loss: 0.4460, Train Accuracy: 0.8688
Epoch 45, Train Loss: 0.4353, Train Accuracy: 0.8724
Epoch 46, Train Loss: 0.4350, Train Accuracy: 0.8732
Epoch 47, Train Loss: 0.4356, Train Accuracy: 0.8723
Epoch 48, Train Loss: 0.4335, Train Accuracy: 0.8743
Epoch 49, Train Loss: 0.4264, Train Accuracy: 0.8757
        Epoch 50, Validation Loss: 44.1448, Validation Accuracy: 0.9169
Epoch 51, Train Loss: 0.4241, Train Accuracy: 0.8767
Epoch 52, Train Loss: 0.4269, Train Accuracy: 0.8761
Epoch 53, Train Loss: 0.4191, Train Accuracy: 0.8782
Epoch 54, Train Loss: 0.4220, Train Accuracy: 0.8768
Epoch 55, Train Loss: 0.4181, Train Accuracy: 0.8784
Epoch 56, Train Loss: 0.4161, Train Accuracy: 0.8796
Epoch 57, Train Loss: 0.4187, Train Accuracy: 0.8783
Epoch 58, Train Loss: 0.4102, Train Accuracy: 0.8809
```

```
Epoch 59, Train Loss: 0.4084, Train Accuracy: 0.8825
        Epoch 60, Validation Loss: 42.6493, Validation Accuracy: 0.9205
Epoch 61, Train Loss: 0.4117, Train Accuracy: 0.8806
Epoch 62, Train Loss: 0.4054, Train Accuracy: 0.8830
Epoch 63, Train Loss: 0.4085, Train Accuracy: 0.8804
Epoch 64, Train Loss: 0.4008, Train Accuracy: 0.8838
Epoch 65, Train Loss: 0.3991, Train Accuracy: 0.8835
Epoch 66, Train Loss: 0.4024, Train Accuracy: 0.8834
Epoch 67, Train Loss: 0.4060, Train Accuracy: 0.8814
Epoch 68, Train Loss: 0.3997, Train Accuracy: 0.8835
Epoch 69, Train Loss: 0.3987, Train Accuracy: 0.8845
        Epoch 70, Validation Loss: 41.3556, Validation Accuracy: 0.9220
Epoch 71, Train Loss: 0.3974, Train Accuracy: 0.8848
Epoch 72, Train Loss: 0.3944, Train Accuracy: 0.8865
Epoch 73, Train Loss: 0.3914, Train Accuracy: 0.8857
Epoch 74, Train Loss: 0.3925, Train Accuracy: 0.8856
Epoch 75, Train Loss: 0.3936, Train Accuracy: 0.8868
Epoch 76, Train Loss: 0.3871, Train Accuracy: 0.8893
Epoch 77, Train Loss: 0.3880, Train Accuracy: 0.8887
Epoch 78, Train Loss: 0.3855, Train Accuracy: 0.8884
Epoch 79, Train Loss: 0.3857, Train Accuracy: 0.8891
        Epoch 80, Validation Loss: 40.2727, Validation Accuracy: 0.9245
Epoch 81, Train Loss: 0.3821, Train Accuracy: 0.8890
Epoch 82, Train Loss: 0.3821, Train Accuracy: 0.8889
Epoch 83, Train Loss: 0.3854, Train Accuracy: 0.8885
Epoch 84, Train Loss: 0.3809, Train Accuracy: 0.8896
Epoch 85, Train Loss: 0.3823, Train Accuracy: 0.8890
Epoch 86, Train Loss: 0.3786, Train Accuracy: 0.8906
Epoch 87, Train Loss: 0.3770, Train Accuracy: 0.8914
Epoch 88, Train Loss: 0.3772, Train Accuracy: 0.8910
Epoch 89, Train Loss: 0.3752, Train Accuracy: 0.8922
        Epoch 90, Validation Loss: 39.4336, Validation Accuracy: 0.9244
Epoch 91, Train Loss: 0.3745, Train Accuracy: 0.8913
Epoch 92, Train Loss: 0.3753, Train Accuracy: 0.8913
Epoch 93, Train Loss: 0.3722, Train Accuracy: 0.8932
Epoch 94, Train Loss: 0.3743, Train Accuracy: 0.8921
Epoch 95, Train Loss: 0.3711, Train Accuracy: 0.8936
Epoch 96, Train Loss: 0.3690, Train Accuracy: 0.8937
Epoch 97, Train Loss: 0.3691, Train Accuracy: 0.8927
Epoch 98, Train Loss: 0.3689, Train Accuracy: 0.8946
Epoch 99, Train Loss: 0.3632, Train Accuracy: 0.8952
        Epoch 100, Validation Loss: 38.6740, Validation Accuracy: 0.9261
```




```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6007 (pid 18147), started 0:16:37 ago. (Use '!kill␣
↪18147' to kill it.)
```

```
<IPython.core.display.HTML object>
```

7.3 Modèle LinearMultiClassWithBatchNorm

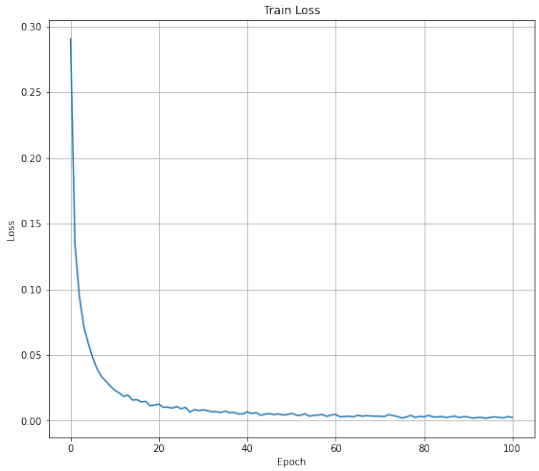
On sait que les données centrées réduites permettent un apprentissage plus rapide et stable d'un modèle ; bien qu'on puisse faire en sorte que les données en entrées soient centrées réduites, cela est plus délicat pour les couches internes d'un réseau de neurones. La technique de **BatchNorm** consiste à ajouter une couche qui a pour but de centrer/réduire les données en utilisant une moyenne/variance glissante (en inférence) et les statistiques du batch (en apprentissage).

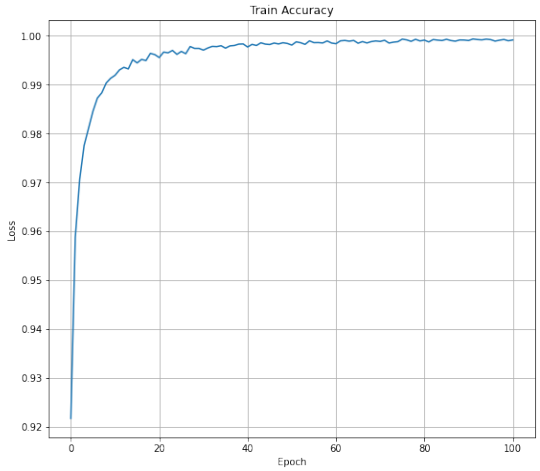
[26]:
```
model = LinearMultiClassWithBatchNorm(in_size=784, out_size=10,␣
 ↪hidden_layers=[256, 128, 64])
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
model3ExecutionTimeAdam, model3ComplexityAdam, model3ValidationLossAdam,␣
 ↪model3AccuracyAdam = train_batchnorm(model, train_loader, validation_loader,␣
 ↪loss_fn, optimizer, epochs=100, typeTrain="BatchNorm-Adam")
```

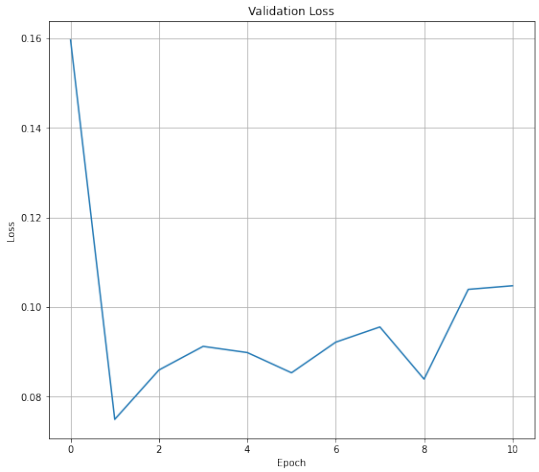
```
        Epoch 0, Validation Loss: 25.0595, Validation Accuracy: 0.9518
Epoch 1, Train Loss: 0.1343, Train Accuracy: 0.9588
Epoch 2, Train Loss: 0.0947, Train Accuracy: 0.9707
Epoch 3, Train Loss: 0.0718, Train Accuracy: 0.9776
Epoch 4, Train Loss: 0.0589, Train Accuracy: 0.9811
Epoch 5, Train Loss: 0.0481, Train Accuracy: 0.9846
Epoch 6, Train Loss: 0.0394, Train Accuracy: 0.9873
Epoch 7, Train Loss: 0.0336, Train Accuracy: 0.9883
Epoch 8, Train Loss: 0.0299, Train Accuracy: 0.9903
Epoch 9, Train Loss: 0.0263, Train Accuracy: 0.9913
        Epoch 10, Validation Loss: 11.7660, Validation Accuracy: 0.9783
Epoch 11, Train Loss: 0.0211, Train Accuracy: 0.9930
Epoch 12, Train Loss: 0.0184, Train Accuracy: 0.9936
Epoch 13, Train Loss: 0.0195, Train Accuracy: 0.9932
Epoch 14, Train Loss: 0.0157, Train Accuracy: 0.9951
Epoch 15, Train Loss: 0.0159, Train Accuracy: 0.9945
Epoch 16, Train Loss: 0.0142, Train Accuracy: 0.9952
Epoch 17, Train Loss: 0.0146, Train Accuracy: 0.9950
Epoch 18, Train Loss: 0.0113, Train Accuracy: 0.9964
Epoch 19, Train Loss: 0.0119, Train Accuracy: 0.9962
        Epoch 20, Validation Loss: 13.4952, Validation Accuracy: 0.9793
Epoch 21, Train Loss: 0.0102, Train Accuracy: 0.9967
Epoch 22, Train Loss: 0.0102, Train Accuracy: 0.9965
Epoch 23, Train Loss: 0.0095, Train Accuracy: 0.9970
Epoch 24, Train Loss: 0.0107, Train Accuracy: 0.9962
Epoch 25, Train Loss: 0.0091, Train Accuracy: 0.9968
Epoch 26, Train Loss: 0.0100, Train Accuracy: 0.9963
Epoch 27, Train Loss: 0.0065, Train Accuracy: 0.9978
Epoch 28, Train Loss: 0.0083, Train Accuracy: 0.9974
Epoch 29, Train Loss: 0.0077, Train Accuracy: 0.9974
        Epoch 30, Validation Loss: 14.3240, Validation Accuracy: 0.9793
Epoch 31, Train Loss: 0.0076, Train Accuracy: 0.9975
Epoch 32, Train Loss: 0.0067, Train Accuracy: 0.9979
Epoch 33, Train Loss: 0.0068, Train Accuracy: 0.9978
Epoch 34, Train Loss: 0.0061, Train Accuracy: 0.9980
Epoch 35, Train Loss: 0.0073, Train Accuracy: 0.9975
Epoch 36, Train Loss: 0.0060, Train Accuracy: 0.9980
Epoch 37, Train Loss: 0.0063, Train Accuracy: 0.9980
Epoch 38, Train Loss: 0.0051, Train Accuracy: 0.9983
Epoch 39, Train Loss: 0.0051, Train Accuracy: 0.9983
```

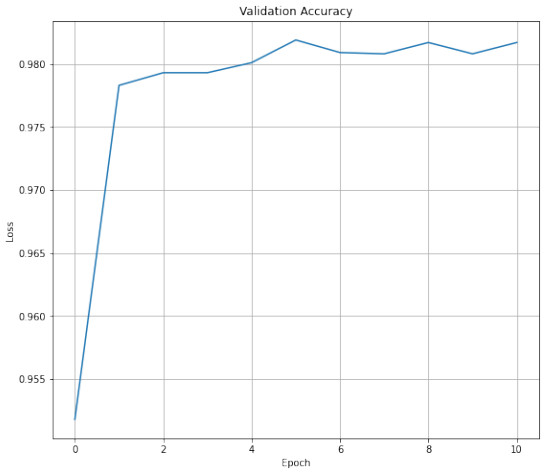
```
        Epoch 40, Validation Loss: 14.1071, Validation Accuracy: 0.9801
Epoch 41, Train Loss: 0.0054, Train Accuracy: 0.9983
Epoch 42, Train Loss: 0.0061, Train Accuracy: 0.9980
Epoch 43, Train Loss: 0.0041, Train Accuracy: 0.9986
Epoch 44, Train Loss: 0.0049, Train Accuracy: 0.9983
Epoch 45, Train Loss: 0.0054, Train Accuracy: 0.9982
Epoch 46, Train Loss: 0.0046, Train Accuracy: 0.9985
Epoch 47, Train Loss: 0.0051, Train Accuracy: 0.9983
Epoch 48, Train Loss: 0.0044, Train Accuracy: 0.9986
Epoch 49, Train Loss: 0.0046, Train Accuracy: 0.9984
        Epoch 50, Validation Loss: 13.3984, Validation Accuracy: 0.9819
Epoch 51, Train Loss: 0.0043, Train Accuracy: 0.9988
Epoch 52, Train Loss: 0.0040, Train Accuracy: 0.9986
Epoch 53, Train Loss: 0.0052, Train Accuracy: 0.9983
Epoch 54, Train Loss: 0.0034, Train Accuracy: 0.9990
Epoch 55, Train Loss: 0.0040, Train Accuracy: 0.9986
Epoch 56, Train Loss: 0.0043, Train Accuracy: 0.9986
Epoch 57, Train Loss: 0.0046, Train Accuracy: 0.9986
Epoch 58, Train Loss: 0.0032, Train Accuracy: 0.9990
Epoch 59, Train Loss: 0.0044, Train Accuracy: 0.9985
        Epoch 60, Validation Loss: 14.4704, Validation Accuracy: 0.9809
Epoch 61, Train Loss: 0.0029, Train Accuracy: 0.9990
Epoch 62, Train Loss: 0.0031, Train Accuracy: 0.9991
Epoch 63, Train Loss: 0.0033, Train Accuracy: 0.9989
Epoch 64, Train Loss: 0.0028, Train Accuracy: 0.9991
Epoch 65, Train Loss: 0.0041, Train Accuracy: 0.9985
Epoch 66, Train Loss: 0.0034, Train Accuracy: 0.9988
Epoch 67, Train Loss: 0.0038, Train Accuracy: 0.9986
Epoch 68, Train Loss: 0.0035, Train Accuracy: 0.9988
Epoch 69, Train Loss: 0.0034, Train Accuracy: 0.9990
        Epoch 70, Validation Loss: 15.0041, Validation Accuracy: 0.9808
Epoch 71, Train Loss: 0.0030, Train Accuracy: 0.9991
Epoch 72, Train Loss: 0.0045, Train Accuracy: 0.9985
Epoch 73, Train Loss: 0.0038, Train Accuracy: 0.9987
Epoch 74, Train Loss: 0.0031, Train Accuracy: 0.9988
Epoch 75, Train Loss: 0.0019, Train Accuracy: 0.9993
Epoch 76, Train Loss: 0.0028, Train Accuracy: 0.9992
Epoch 77, Train Loss: 0.0040, Train Accuracy: 0.9989
Epoch 78, Train Loss: 0.0025, Train Accuracy: 0.9993
Epoch 79, Train Loss: 0.0032, Train Accuracy: 0.9990
        Epoch 80, Validation Loss: 13.1758, Validation Accuracy: 0.9817
Epoch 81, Train Loss: 0.0040, Train Accuracy: 0.9988
Epoch 82, Train Loss: 0.0029, Train Accuracy: 0.9992
Epoch 83, Train Loss: 0.0028, Train Accuracy: 0.9991
Epoch 84, Train Loss: 0.0032, Train Accuracy: 0.9990
Epoch 85, Train Loss: 0.0023, Train Accuracy: 0.9993
Epoch 86, Train Loss: 0.0030, Train Accuracy: 0.9990
Epoch 87, Train Loss: 0.0034, Train Accuracy: 0.9989
```

```
Epoch 88, Train Loss: 0.0023, Train Accuracy: 0.9992
Epoch 89, Train Loss: 0.0032, Train Accuracy: 0.9991
       Epoch 90, Validation Loss: 16.3207, Validation Accuracy: 0.9808
Epoch 91, Train Loss: 0.0019, Train Accuracy: 0.9994
Epoch 92, Train Loss: 0.0023, Train Accuracy: 0.9992
Epoch 93, Train Loss: 0.0024, Train Accuracy: 0.9992
Epoch 94, Train Loss: 0.0017, Train Accuracy: 0.9993
Epoch 95, Train Loss: 0.0025, Train Accuracy: 0.9992
Epoch 96, Train Loss: 0.0028, Train Accuracy: 0.9989
Epoch 97, Train Loss: 0.0024, Train Accuracy: 0.9991
Epoch 98, Train Loss: 0.0021, Train Accuracy: 0.9993
Epoch 99, Train Loss: 0.0031, Train Accuracy: 0.9990
       Epoch 100, Validation Loss: 16.4474, Validation Accuracy: 0.9817
```










```
The tensorboard extension is already loaded. To reload it, use:
```

```
%reload_ext tensorboard
```

```
<IPython.core.display.HTML object>
```

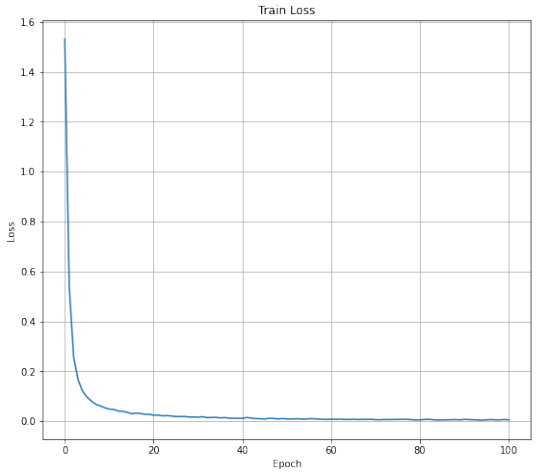
7.3.1 En utilisant l'activation = Softmax et l'optimiseur = Adam

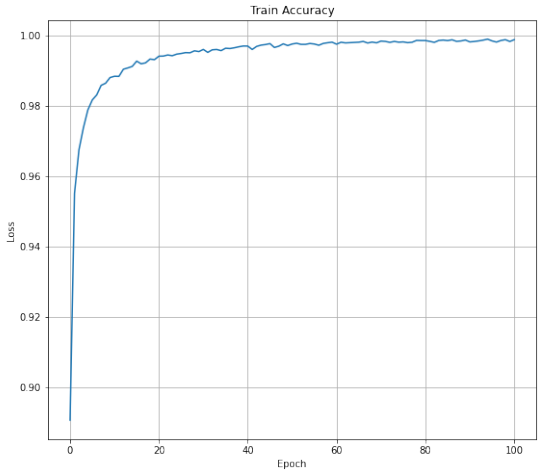
[27]:
```
model = LinearMultiClassWithBatchNorm(in_size=784, out_size=10,␣
↪hidden_layers=[256, 128, 64], activation=nn.Softmax())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
model3ExecutionTimeAdam, model3ComplexityAdam, model3ValidationLossAdam,␣
↪model3AccuracyAdam = train_batchnorm(model, train_loader, validation_loader,␣
↪loss_fn, optimizer, epochs=100, typeTrain="BatchNorm-Adam")
```

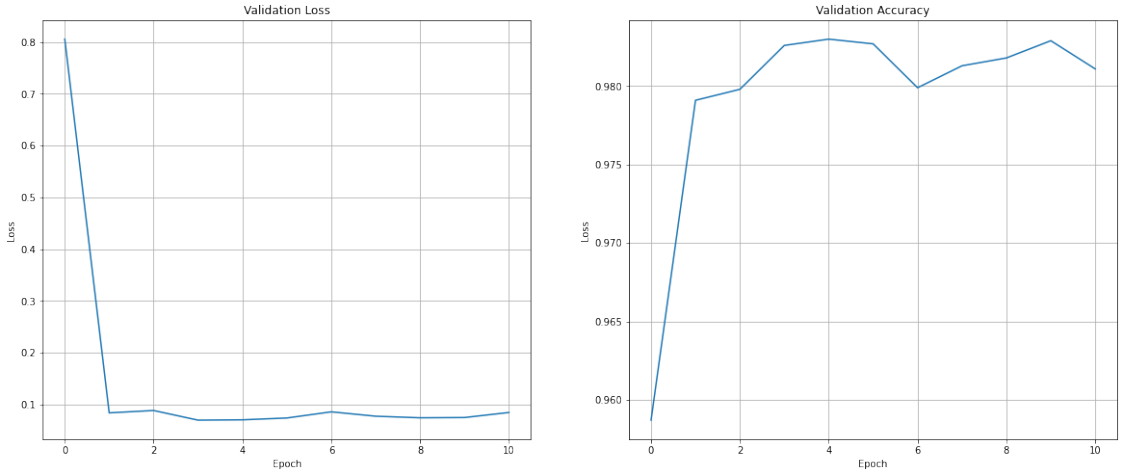
```
<ipython-input-4-8096a4ae721a>:73: UserWarning: Implicit dimension choice for
softmax has been deprecated. Change the call to include dim=X as an argument.
  x = self.activation(x)

        Epoch 0, Validation Loss: 126.4466, Validation Accuracy: 0.9587
Epoch 1, Train Loss: 0.5397, Train Accuracy: 0.9550
Epoch 2, Train Loss: 0.2564, Train Accuracy: 0.9674
Epoch 3, Train Loss: 0.1661, Train Accuracy: 0.9737
Epoch 4, Train Loss: 0.1208, Train Accuracy: 0.9787
Epoch 5, Train Loss: 0.0963, Train Accuracy: 0.9816
Epoch 6, Train Loss: 0.0804, Train Accuracy: 0.9830
Epoch 7, Train Loss: 0.0676, Train Accuracy: 0.9857
Epoch 8, Train Loss: 0.0615, Train Accuracy: 0.9863
Epoch 9, Train Loss: 0.0538, Train Accuracy: 0.9879
        Epoch 10, Validation Loss: 13.1739, Validation Accuracy: 0.9791
Epoch 11, Train Loss: 0.0475, Train Accuracy: 0.9883
Epoch 12, Train Loss: 0.0410, Train Accuracy: 0.9903
Epoch 13, Train Loss: 0.0395, Train Accuracy: 0.9907
Epoch 14, Train Loss: 0.0360, Train Accuracy: 0.9911
Epoch 15, Train Loss: 0.0301, Train Accuracy: 0.9926
Epoch 16, Train Loss: 0.0319, Train Accuracy: 0.9919
Epoch 17, Train Loss: 0.0311, Train Accuracy: 0.9921
Epoch 18, Train Loss: 0.0275, Train Accuracy: 0.9932
Epoch 19, Train Loss: 0.0274, Train Accuracy: 0.9930
        Epoch 20, Validation Loss: 13.8890, Validation Accuracy: 0.9798
Epoch 21, Train Loss: 0.0243, Train Accuracy: 0.9940
Epoch 22, Train Loss: 0.0216, Train Accuracy: 0.9944
Epoch 23, Train Loss: 0.0222, Train Accuracy: 0.9941
Epoch 24, Train Loss: 0.0211, Train Accuracy: 0.9946
Epoch 25, Train Loss: 0.0185, Train Accuracy: 0.9948
Epoch 26, Train Loss: 0.0186, Train Accuracy: 0.9950
Epoch 27, Train Loss: 0.0190, Train Accuracy: 0.9950
Epoch 28, Train Loss: 0.0168, Train Accuracy: 0.9956
Epoch 29, Train Loss: 0.0170, Train Accuracy: 0.9953
        Epoch 30, Validation Loss: 10.9673, Validation Accuracy: 0.9826
```

```
Epoch 31, Train Loss: 0.0180, Train Accuracy: 0.9951
Epoch 32, Train Loss: 0.0145, Train Accuracy: 0.9958
Epoch 33, Train Loss: 0.0149, Train Accuracy: 0.9959
Epoch 34, Train Loss: 0.0155, Train Accuracy: 0.9956
Epoch 35, Train Loss: 0.0134, Train Accuracy: 0.9963
Epoch 36, Train Loss: 0.0146, Train Accuracy: 0.9962
Epoch 37, Train Loss: 0.0124, Train Accuracy: 0.9964
Epoch 38, Train Loss: 0.0118, Train Accuracy: 0.9967
Epoch 39, Train Loss: 0.0114, Train Accuracy: 0.9969
        Epoch 40, Validation Loss: 11.0643, Validation Accuracy: 0.9830
Epoch 41, Train Loss: 0.0149, Train Accuracy: 0.9959
Epoch 42, Train Loss: 0.0124, Train Accuracy: 0.9968
Epoch 43, Train Loss: 0.0105, Train Accuracy: 0.9972
Epoch 44, Train Loss: 0.0100, Train Accuracy: 0.9973
Epoch 45, Train Loss: 0.0086, Train Accuracy: 0.9976
Epoch 46, Train Loss: 0.0118, Train Accuracy: 0.9965
Epoch 47, Train Loss: 0.0112, Train Accuracy: 0.9969
Epoch 48, Train Loss: 0.0089, Train Accuracy: 0.9975
Epoch 49, Train Loss: 0.0106, Train Accuracy: 0.9970
        Epoch 50, Validation Loss: 11.6187, Validation Accuracy: 0.9827
Epoch 51, Train Loss: 0.0087, Train Accuracy: 0.9977
Epoch 52, Train Loss: 0.0098, Train Accuracy: 0.9974
Epoch 53, Train Loss: 0.0093, Train Accuracy: 0.9974
Epoch 54, Train Loss: 0.0078, Train Accuracy: 0.9976
Epoch 55, Train Loss: 0.0102, Train Accuracy: 0.9975
Epoch 56, Train Loss: 0.0099, Train Accuracy: 0.9971
Epoch 57, Train Loss: 0.0084, Train Accuracy: 0.9977
Epoch 58, Train Loss: 0.0081, Train Accuracy: 0.9979
Epoch 59, Train Loss: 0.0068, Train Accuracy: 0.9980
        Epoch 60, Validation Loss: 13.4824, Validation Accuracy: 0.9799
Epoch 61, Train Loss: 0.0072, Train Accuracy: 0.9980
Epoch 62, Train Loss: 0.0084, Train Accuracy: 0.9978
Epoch 63, Train Loss: 0.0071, Train Accuracy: 0.9979
Epoch 64, Train Loss: 0.0070, Train Accuracy: 0.9979
Epoch 65, Train Loss: 0.0079, Train Accuracy: 0.9980
Epoch 66, Train Loss: 0.0066, Train Accuracy: 0.9982
Epoch 67, Train Loss: 0.0074, Train Accuracy: 0.9978
Epoch 68, Train Loss: 0.0072, Train Accuracy: 0.9980
Epoch 69, Train Loss: 0.0078, Train Accuracy: 0.9978
        Epoch 70, Validation Loss: 12.1587, Validation Accuracy: 0.9813
Epoch 71, Train Loss: 0.0059, Train Accuracy: 0.9982
Epoch 72, Train Loss: 0.0069, Train Accuracy: 0.9980
Epoch 73, Train Loss: 0.0065, Train Accuracy: 0.9982
Epoch 74, Train Loss: 0.0069, Train Accuracy: 0.9980
Epoch 75, Train Loss: 0.0071, Train Accuracy: 0.9981
Epoch 76, Train Loss: 0.0075, Train Accuracy: 0.9979
Epoch 77, Train Loss: 0.0076, Train Accuracy: 0.9980
Epoch 78, Train Loss: 0.0061, Train Accuracy: 0.9985
```

```
Epoch 79, Train Loss: 0.0054, Train Accuracy: 0.9985
        Epoch 80, Validation Loss: 11.6834, Validation Accuracy: 0.9818
Epoch 81, Train Loss: 0.0073, Train Accuracy: 0.9982
Epoch 82, Train Loss: 0.0076, Train Accuracy: 0.9980
Epoch 83, Train Loss: 0.0059, Train Accuracy: 0.9985
Epoch 84, Train Loss: 0.0044, Train Accuracy: 0.9986
Epoch 85, Train Loss: 0.0053, Train Accuracy: 0.9984
Epoch 86, Train Loss: 0.0053, Train Accuracy: 0.9987
Epoch 87, Train Loss: 0.0060, Train Accuracy: 0.9982
Epoch 88, Train Loss: 0.0063, Train Accuracy: 0.9983
Epoch 89, Train Loss: 0.0053, Train Accuracy: 0.9986
        Epoch 90, Validation Loss: 11.7605, Validation Accuracy: 0.9829
Epoch 91, Train Loss: 0.0065, Train Accuracy: 0.9982
Epoch 92, Train Loss: 0.0059, Train Accuracy: 0.9984
Epoch 93, Train Loss: 0.0048, Train Accuracy: 0.9986
Epoch 94, Train Loss: 0.0042, Train Accuracy: 0.9989
Epoch 95, Train Loss: 0.0056, Train Accuracy: 0.9984
Epoch 96, Train Loss: 0.0064, Train Accuracy: 0.9980
Epoch 97, Train Loss: 0.0053, Train Accuracy: 0.9985
Epoch 98, Train Loss: 0.0052, Train Accuracy: 0.9987
Epoch 99, Train Loss: 0.0070, Train Accuracy: 0.9982
        Epoch 100, Validation Loss: 13.2875, Validation Accuracy: 0.9811
```








```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6008 (pid 18952), started 0:18:19 ago. (Use '!kill␣
 ↪18952' to kill it.)
```

```
<IPython.core.display.HTML object>
```

7.3.2 En utilisant l'activation = Softmax et l'optimiseur = SGD

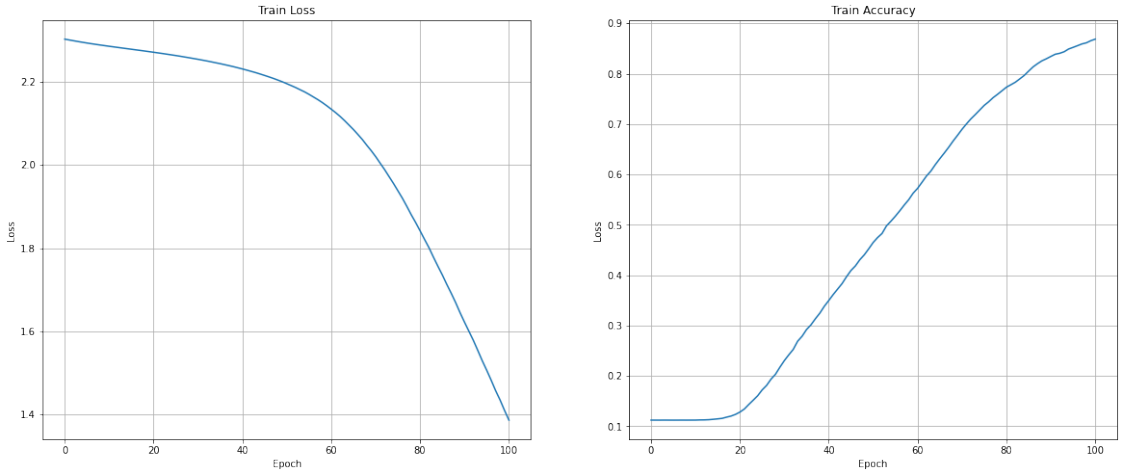
[28]:
```
model = LinearMultiClassWithBatchNorm(in_size=784, out_size=10,␣
 ↪hidden_layers=[256, 128, 64], activation=nn.Softmax())
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
model3ExecutionTimeSGD, model3ComplexitySGD, model3ValidationLossSGD,␣
 ↪model3AccuracySGD = train_batchnorm(model, train_loader, validation_loader,␣
 ↪loss_fn, optimizer, epochs=100, typeTrain="BatchNorm-SGD")
```

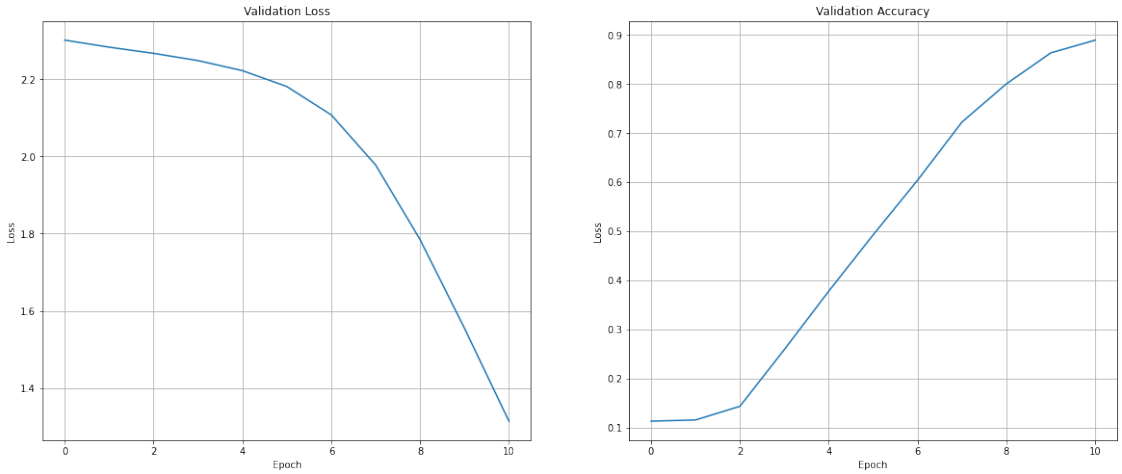
```
<ipython-input-4-8096a4ae721a>:73: UserWarning: Implicit dimension choice for
softmax has been deprecated. Change the call to include dim=X as an argument.
  x = self.activation(x)
```

```
        Epoch 0, Validation Loss: 361.3012, Validation Accuracy: 0.1137
Epoch 1, Train Loss: 2.3003, Train Accuracy: 0.1124
Epoch 2, Train Loss: 2.2984, Train Accuracy: 0.1124
Epoch 3, Train Loss: 2.2965, Train Accuracy: 0.1124
Epoch 4, Train Loss: 2.2947, Train Accuracy: 0.1124
Epoch 5, Train Loss: 2.2929, Train Accuracy: 0.1124
Epoch 6, Train Loss: 2.2913, Train Accuracy: 0.1124
Epoch 7, Train Loss: 2.2897, Train Accuracy: 0.1124
Epoch 8, Train Loss: 2.2881, Train Accuracy: 0.1124
Epoch 9, Train Loss: 2.2866, Train Accuracy: 0.1124
        Epoch 10, Validation Loss: 358.4453, Validation Accuracy: 0.1161
```

```
Epoch 11, Train Loss: 2.2837, Train Accuracy: 0.1128
Epoch 12, Train Loss: 2.2822, Train Accuracy: 0.1129
Epoch 13, Train Loss: 2.2808, Train Accuracy: 0.1133
Epoch 14, Train Loss: 2.2794, Train Accuracy: 0.1140
Epoch 15, Train Loss: 2.2780, Train Accuracy: 0.1149
Epoch 16, Train Loss: 2.2765, Train Accuracy: 0.1159
Epoch 17, Train Loss: 2.2751, Train Accuracy: 0.1183
Epoch 18, Train Loss: 2.2737, Train Accuracy: 0.1204
Epoch 19, Train Loss: 2.2722, Train Accuracy: 0.1237
        Epoch 20, Validation Loss: 355.9202, Validation Accuracy: 0.1439
Epoch 21, Train Loss: 2.2692, Train Accuracy: 0.1344
Epoch 22, Train Loss: 2.2676, Train Accuracy: 0.1435
Epoch 23, Train Loss: 2.2661, Train Accuracy: 0.1520
Epoch 24, Train Loss: 2.2644, Train Accuracy: 0.1608
Epoch 25, Train Loss: 2.2628, Train Accuracy: 0.1722
Epoch 26, Train Loss: 2.2611, Train Accuracy: 0.1812
Epoch 27, Train Loss: 2.2593, Train Accuracy: 0.1936
Epoch 28, Train Loss: 2.2575, Train Accuracy: 0.2033
Epoch 29, Train Loss: 2.2556, Train Accuracy: 0.2174
        Epoch 30, Validation Loss: 352.9756, Validation Accuracy: 0.2595
Epoch 31, Train Loss: 2.2518, Train Accuracy: 0.2419
Epoch 32, Train Loss: 2.2498, Train Accuracy: 0.2526
Epoch 33, Train Loss: 2.2477, Train Accuracy: 0.2690
Epoch 34, Train Loss: 2.2454, Train Accuracy: 0.2789
Epoch 35, Train Loss: 2.2433, Train Accuracy: 0.2922
Epoch 36, Train Loss: 2.2409, Train Accuracy: 0.3015
Epoch 37, Train Loss: 2.2385, Train Accuracy: 0.3136
Epoch 38, Train Loss: 2.2360, Train Accuracy: 0.3248
Epoch 39, Train Loss: 2.2333, Train Accuracy: 0.3383
        Epoch 40, Validation Loss: 348.8990, Validation Accuracy: 0.3783
Epoch 41, Train Loss: 2.2277, Train Accuracy: 0.3616
Epoch 42, Train Loss: 2.2247, Train Accuracy: 0.3724
Epoch 43, Train Loss: 2.2217, Train Accuracy: 0.3833
Epoch 44, Train Loss: 2.2184, Train Accuracy: 0.3970
Epoch 45, Train Loss: 2.2150, Train Accuracy: 0.4093
Epoch 46, Train Loss: 2.2114, Train Accuracy: 0.4184
Epoch 47, Train Loss: 2.2077, Train Accuracy: 0.4308
Epoch 48, Train Loss: 2.2037, Train Accuracy: 0.4404
Epoch 49, Train Loss: 2.1997, Train Accuracy: 0.4523
        Epoch 50, Validation Loss: 342.4152, Validation Accuracy: 0.4930
Epoch 51, Train Loss: 2.1905, Train Accuracy: 0.4747
Epoch 52, Train Loss: 2.1858, Train Accuracy: 0.4828
Epoch 53, Train Loss: 2.1806, Train Accuracy: 0.4979
Epoch 54, Train Loss: 2.1754, Train Accuracy: 0.5071
Epoch 55, Train Loss: 2.1695, Train Accuracy: 0.5171
Epoch 56, Train Loss: 2.1631, Train Accuracy: 0.5279
Epoch 57, Train Loss: 2.1568, Train Accuracy: 0.5393
Epoch 58, Train Loss: 2.1497, Train Accuracy: 0.5498
```

```
Epoch 59, Train Loss: 2.1420, Train Accuracy: 0.5627
        Epoch 60, Validation Loss: 330.8959, Validation Accuracy: 0.6043
Epoch 61, Train Loss: 2.1254, Train Accuracy: 0.5845
Epoch 62, Train Loss: 2.1164, Train Accuracy: 0.5968
Epoch 63, Train Loss: 2.1064, Train Accuracy: 0.6068
Epoch 64, Train Loss: 2.0956, Train Accuracy: 0.6195
Epoch 65, Train Loss: 2.0848, Train Accuracy: 0.6312
Epoch 66, Train Loss: 2.0730, Train Accuracy: 0.6423
Epoch 67, Train Loss: 2.0609, Train Accuracy: 0.6539
Epoch 68, Train Loss: 2.0472, Train Accuracy: 0.6663
Epoch 69, Train Loss: 2.0341, Train Accuracy: 0.6777
        Epoch 70, Validation Loss: 310.4759, Validation Accuracy: 0.7227
Epoch 71, Train Loss: 2.0042, Train Accuracy: 0.7002
Epoch 72, Train Loss: 1.9888, Train Accuracy: 0.7102
Epoch 73, Train Loss: 1.9720, Train Accuracy: 0.7188
Epoch 74, Train Loss: 1.9555, Train Accuracy: 0.7279
Epoch 75, Train Loss: 1.9378, Train Accuracy: 0.7370
Epoch 76, Train Loss: 1.9201, Train Accuracy: 0.7441
Epoch 77, Train Loss: 1.9006, Train Accuracy: 0.7523
Epoch 78, Train Loss: 1.8802, Train Accuracy: 0.7590
Epoch 79, Train Loss: 1.8614, Train Accuracy: 0.7660
        Epoch 80, Validation Loss: 280.1361, Validation Accuracy: 0.8005
Epoch 81, Train Loss: 1.8209, Train Accuracy: 0.7780
Epoch 82, Train Loss: 1.8009, Train Accuracy: 0.7830
Epoch 83, Train Loss: 1.7783, Train Accuracy: 0.7895
Epoch 84, Train Loss: 1.7566, Train Accuracy: 0.7964
Epoch 85, Train Loss: 1.7356, Train Accuracy: 0.8050
Epoch 86, Train Loss: 1.7130, Train Accuracy: 0.8134
Epoch 87, Train Loss: 1.6915, Train Accuracy: 0.8198
Epoch 88, Train Loss: 1.6690, Train Accuracy: 0.8255
Epoch 89, Train Loss: 1.6447, Train Accuracy: 0.8295
        Epoch 90, Validation Loss: 244.0682, Validation Accuracy: 0.8636
Epoch 91, Train Loss: 1.5999, Train Accuracy: 0.8385
Epoch 92, Train Loss: 1.5779, Train Accuracy: 0.8404
Epoch 93, Train Loss: 1.5537, Train Accuracy: 0.8435
Epoch 94, Train Loss: 1.5291, Train Accuracy: 0.8489
Epoch 95, Train Loss: 1.5063, Train Accuracy: 0.8521
Epoch 96, Train Loss: 1.4825, Train Accuracy: 0.8555
Epoch 97, Train Loss: 1.4575, Train Accuracy: 0.8590
Epoch 98, Train Loss: 1.4350, Train Accuracy: 0.8612
Epoch 99, Train Loss: 1.4101, Train Accuracy: 0.8654
        Epoch 100, Validation Loss: 206.3141, Validation Accuracy: 0.8895
```






```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6008 (pid 18952), started 0:35:32 ago. (Use '!kill␣
↪18952' to kill it.)
```

```
<IPython.core.display.HTML object>
```

8 Fonction de comparaison des modèles

Cette fonction renvoie maintenant des informations détaillées sur les deux modèles, y compris la perte, la précision, la complexité en termes de nombre de paramètres et le temps d'évaluation sur les données de validation.

Pour comparer les performances d'un modèle sur des données de validation, il est plus pratique de comparer les moyennes des pertes de validation (validation_loss) calculées sur l'ensemble des

données de validation pour chaque époque.

N.B. L'accuracy est un métrique couramment utilisé pour évaluer la performance d'un modèle de classification. C'est la proportion de prédictions correctes que le modèle fait par rapport au nombre total de prédictions. L'accuracy est généralement exprimée en pourcentage et peut être calculée en comparant les étiquettes prévues par le modèle avec les étiquettes réelles. Plus l'accuracy est élevée, meilleure est la performance du modèle.

[29]:
```
def compare_models(executionTimeOfModel1, complexityOfModel1,␣
↪validationLossOfModel1, accuracyOfModel1, executionTimeOfModel2,␣
↪complexityOfModel2, validationLossOfModel2, accuracyOfModel2):
    print("Le modèle 1 a une perte de {} sur les données de validation avec une␣
↪précision de {}% et une complexité de {} paramètres et un temps d'évaluation␣
↪de {}s".format(validationLossOfModel1, accuracyOfModel1*100,␣
↪complexityOfModel1, executionTimeOfModel1))
    print("\nLe modèle 2 a une perte de {} sur les données de validation avec␣
↪une précision de {}% et une complexité de {} paramètres et un temps␣
↪d'évaluation de {}s".format(validationLossOfModel2, accuracyOfModel2*100,␣
↪complexityOfModel2, executionTimeOfModel2))

    if validationLossOfModel1 < validationLossOfModel2:
        print("\n\tLe modèle 1 a une perte plus faible sur les données de␣
↪validation.")
        return 1
    else:
        print("\n\tLe modèle 2 a une perte plus faible sur les données de␣
↪validation.")
        return 2
```

9 Comparaison des données des modèles

9.1 Pour : LinearMultiClass VS LinearMultiClassWithDropout

[30]:
```
resultTest1 = compare_models(model1ExecutionTimeTanhAdam,␣
↪model1ComplexityTanhAdam, model1ValidationLossTanhAdam,␣
↪model1AccuracyTanhAdam, model2ExecutionTimeAdam, model2ComplexityAdam,␣
↪model2ValidationLossAdam, model2AccuracyAdam)

if resultTest1 == 1:
    print("\n_____\nLe Modèle␣
↪LinearMultiClass (Adam) est le plus performant !
↪\n_____\n")
else:
    print("\n_____\nLe Modèle␣
↪LinearMultiClassWithDropout (Adam) est le plus performant !
↪\n_____\n")
```

```
Le modèle 1 a une perte de 12.005391817656346 sur les données de validation avec
```

une précision de 97.72999999999999% et une complexité de 235146 paramètres et un temps d'évaluation de 1003.714991569519s

Le modèle 2 a une perte de 24.891086659394205 sur les données de validation avec une précision de 95.97% et une complexité de 99710 paramètres et un temps d'évaluation de 3740.377825021744s

Le modèle 1 a une perte plus faible sur les données de validation.

Le Modèle LinearMultiClass (Adam) est le plus performant !
---

[31]:
```
resultTest1 = compare_models(model1ExecutionTimeTanhSGD,␣
 ↪model1ComplexityTanhSGD, model1ValidationLossTanhSGD, model1AccuracyTanhSGD,␣
 ↪model2ExecutionTimeSGD, model2ComplexitySGD, model2ValidationLossSGD,␣
 ↪model2AccuracySGD)

if resultTest1 == 1:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClass (SGD) est le plus performant !
 ↪\n_____\n")
else:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClassWithDropout (SGD) est le plus performant !
 ↪\n_____\n")
```

Le modèle 1 a une perte de 283.09118032455444 sur les données de validation avec une précision de 65.60000000000001% et une complexité de 235146 paramètres et un temps d'évaluation de 949.3833196163177s

Le modèle 2 a une perte de 38.673973706085235 sur les données de validation avec une précision de 92.61% et une complexité de 99710 paramètres et un temps d'évaluation de 996.7878522872925s

Le modèle 2 a une perte plus faible sur les données de validation.

Le Modèle LinearMultiClassWithDropout (SGD) est le plus performant !

9.2 Pour : LinearMultiClass VS LinearMultiClassWithBatchNorm

[32]:
```
resultTest2 = compare_models(model1ExecutionTimeTanhAdam,␣
 ↪model1ComplexityTanhAdam, model1ValidationLossTanhAdam,␣
 ↪model1AccuracyTanhAdam, model3ExecutionTimeAdam, model3ComplexityAdam,␣
 ↪model3ValidationLossAdam, model3AccuracyAdam)

if resultTest2 == 1:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClass (Adam) est le plus performant !
 ↪\n_____\n")
else:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClassWithBatchNorm (Adam) est le plus performant !
 ↪\n_____\n")
```

```
Le modèle 1 a une perte de 12.005391817656346 sur les données de validation avec
une précision de 97.72999999999999% et une complexité de 235146 paramètres et un
temps d'évaluation de 1003.714991569519s

Le modèle 2 a une perte de 13.287496212164115 sur les données de validation avec
une précision de 98.11% et une complexité de 243658 paramètres et un temps
d'évaluation de 1098.6271667480469s

        Le modèle 1 a une perte plus faible sur les données de validation.


--------------------------------------------------
Le Modèle LinearMultiClass (Adam) est le plus performant !
--------------------------------------------------
```

[33]:
```
resultTest2 = compare_models(model1ExecutionTimeTanhSGD,␣
 ↪model1ComplexityTanhSGD, model1ValidationLossTanhSGD, model1AccuracyTanhSGD,␣
 ↪model3ExecutionTimeSGD, model3ComplexitySGD, model3ValidationLossSGD,␣
 ↪model3AccuracySGD)

if resultTest2 == 1:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClass (SGD) est le plus performant !
 ↪\n_____\n")
else:
    print("\n_____\nLe Modèle␣
 ↪LinearMultiClassWithBatchNorm (SGD) est le plus performant !
 ↪\n_____\n")
```

```
Le modèle 1 a une perte de 283.09118032455444 sur les données de validation avec
une précision de 65.60000000000001% et une complexité de 235146 paramètres et un
temps d'évaluation de 949.3833196163177s
```

```
Le modèle 2 a une perte de 206.31406617164612 sur les données de validation avec
une précision de 88.94999999999999% et une complexité de 243658 paramètres et un
temps d'évaluation de 1032.8414254188538s

        Le modèle 2 a une perte plus faible sur les données de validation.


--------------------------------------------------
Le Modèle LinearMultiClassWithBatchNorm (SGD) est le plus performant !
--------------------------------------------------
```

9.3 Pour : LinearMultiClassWithDropout VS LinearMultiClassWithBatchNorm

[34]:
```
resultTest3 = compare_models(model2ExecutionTimeAdam, model2ComplexityAdam,␣
 ↪model2ValidationLossAdam, model2AccuracyAdam, model3ExecutionTimeAdam,␣
 ↪model3ComplexityAdam, model3ValidationLossAdam, model3AccuracyAdam)

if resultTest3 == 1:
   ␣
 ↪print("\n_____________________________________________________________________\nLe␣
 ↪Modèle LinearMultiClassWithDropout (Adam) est le plus performant !
 ↪\n_____________________________________________________________________\n")
else:
   ␣
 ↪print("\n_____________________________________________________________________\nLe␣
 ↪Modèle LinearMultiClassWithBatchNorm (Adam) est le plus performant !
 ↪\n_____________________________________________________________________\n")
```

```
Le modèle 1 a une perte de 24.891086659394205 sur les données de validation avec
une précision de 95.97% et une complexité de 99710 paramètres et un temps
d'évaluation de 3740.377825021744s

Le modèle 2 a une perte de 13.287496212164115 sur les données de validation avec
une précision de 98.11% et une complexité de 243658 paramètres et un temps
d'évaluation de 1098.6271667480469s

        Le modèle 2 a une perte plus faible sur les données de validation.


---------------------------------------------------------------
Le Modèle LinearMultiClassWithBatchNorm (Adam) est le plus performant !
---------------------------------------------------------------
```

[35]:
```
resultTest3 = compare_models(model2ExecutionTimeSGD, model2ComplexitySGD,␣
 ↪model2ValidationLossSGD, model2AccuracySGD, model3ExecutionTimeSGD,␣
 ↪model3ComplexitySGD, model3ValidationLossSGD, model3AccuracySGD)
```

```
if resultTest3 == 1:
    ␣
  ↪print("\n_____________________________________________________________\nLe␣
  ↪Modèle LinearMultiClassWithDropout (SGD) est le plus performant !
  ↪\n_____________________________________________________________\n")
else:
    ␣
  ↪print("\n_____________________________________________________________\nLe␣
  ↪Modèle LinearMultiClassWithBatchNorm (SGD) est le plus performant !
  ↪\n_____________________________________________________________\n")
```

```
Le modèle 1 a une perte de 38.673973706085235 sur les données de validation avec
une précision de 92.61% et une complexité de 99710 paramètres et un temps
d'évaluation de 996.7878522872925s

Le modèle 2 a une perte de 206.31406617164612 sur les données de validation avec
une précision de 88.94999999999999% et une complexité de 243658 paramètres et un
temps d'évaluation de 1032.8414254188538s

        Le modèle 1 a une perte plus faible sur les données de validation.


-------------------------------------------------------------
Le Modèle LinearMultiClassWithDropout (SGD) est le plus performant !
-------------------------------------------------------------
```

10 Conclusion :

En se basant sur les données recueillis lors de nos différents tests principalement en se basant le nombre de pertes de chaque modèle. Nous pouvons donc conclure que le modèle LinearMultiClass est le plus performant des trois (3) modèles que nous avons eu à étudier avec nos jeux de données. En deuxième position LinearMultiClassWithBatchNorm et enfin LinearMultiClassWithDropout en utilisant l'optimiseur Adam. Par contre, en terme de précisions sur les données. Le modèle LinearMultiClassWithBatchNorm est le meilleur parmi les trois modèles étudiés.

Cependant, dans le cas où nous utilison l'optimiseur SGD. Nous pouvons remarquer que le modèle LinearMultiClassWithDropout est le plus performant des trois (3) modèles que nous avons eu à étudier avec nos jeux de données. En deuxième position LinearMultiClassWithBatchNorm et enfin LinearMultiClass.