FashionMNIST

October 4, 2023

1 Deep Learning

1.0.1 Musel Tabares

1.0.2 A00830710

Importamos librerias

[ ]:
```
#para utilizar tensores etc
import torch
#para el modelo
from torch import nn
#para importar datasets
import torchvision
#para transformar imagenes
import torchvision.transforms as transforms
#para visualizaciones
import matplotlib.pyplot as plt
#ver a detalle el modelo
from torchsummary import summary
# barra de progreso
from tqdm.auto import tqdm

#importamos funciones
from utils import *
```

```
c:\Users\musel\anaconda3\envs\pytorch\lib\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1.1 FashionMNIST

Importamos datos

[ ]:
```
trainset = torchvision.datasets.FashionMNIST(root='./data',␣
↪train=True,download=True, transform=transforms.ToTensor())
testset = torchvision.datasets.FashionMNIST(root='./data',␣
↪train=False,download=True, transform=transforms.ToTensor())
```

```
classes = trainset.classes
```

Observamos la dimension de las imagenes

```
[ ]: # desplegamos primer imagen
image, label = trainset[0]
image.shape, label
```

```
[ ]: (torch.Size([1, 28, 28]), 9)
```

Observamos cantidad de datos en train y test

```
[ ]: len(trainset.data), len(trainset.targets), len(testset.data), len(testset.
  ↪targets)
```

```
[ ]: (60000, 60000, 10000, 10000)
```

Creamos batches de los datos

```
[ ]: batch_size = 32
trainloader = torch.utils.data.DataLoader(trainset,␣
  ↪batch_size=batch_size,shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset,␣
  ↪batch_size=batch_size,shuffle=False, num_workers=2)
```

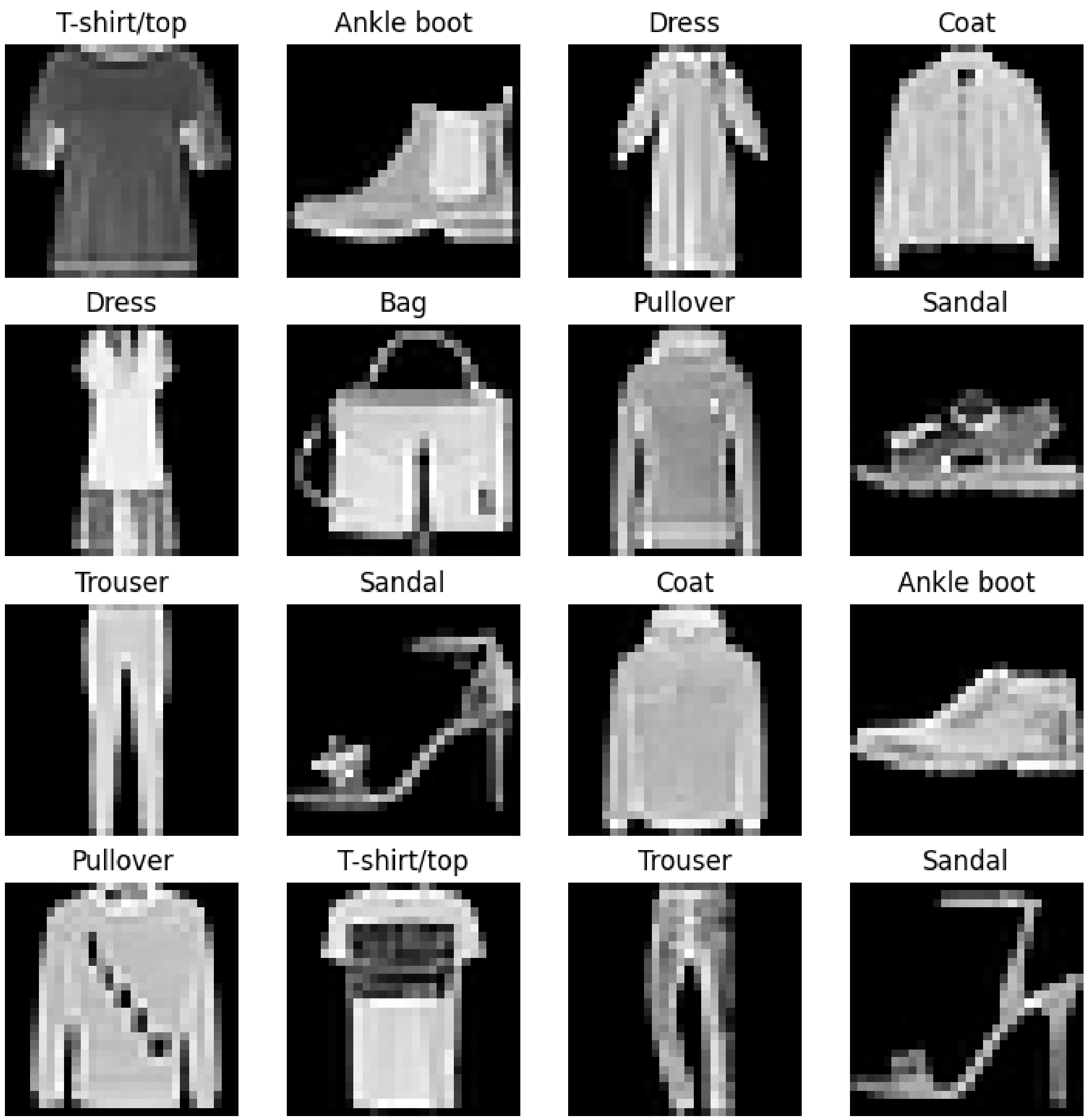
Observamos cuantos batches se crearon

```
[ ]: print(f"Length of train dataloader: {len(trainloader)} batches of {batch_size}")
print(f"Length of test dataloader: {len(testloader)} batches of {batch_size}")
```

```
Length of train dataloader: 1875 batches of 32
Length of test dataloader: 313 batches of 32
```

visualizamos 16 imagenes de manera aleatoria

```
[ ]: plot_sample_images(trainset, classes, 4, 4)
```



Creamos modelo

[ ]:
```
class conv(nn.Module):

    def __init__(self):
        super().__init__()
        self.block_1 = nn.Sequential(
            nn.Conv2d(1,32, kernel_size=(3,3), stride=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2)),
            nn.Conv2d(32,64, kernel_size=(3,3), stride=1),
            nn.ReLU(),
```

```python
            nn.MaxPool2d(kernel_size=(2, 2)),
            nn.Conv2d(64,256, kernel_size=(3,3), stride=1),
            nn.ReLU(),

        )
        self.block_2 = nn.Sequential(
            nn.Flatten(),
            nn.Linear(2304, 64),
            nn.ReLU(),
            nn.Linear(64, 10),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.block_1(x)
        x = self.block_2(x)
        return x
```

instanciamos el modelo

[ ]:
```python
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model_0 = conv().to(device)
summary(model_0, (1,28,28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 26, 26]             320
              ReLU-2           [-1, 32, 26, 26]               0
         MaxPool2d-3           [-1, 32, 13, 13]               0
            Conv2d-4           [-1, 64, 11, 11]          18,496
              ReLU-5           [-1, 64, 11, 11]               0
         MaxPool2d-6             [-1, 64, 5, 5]               0
            Conv2d-7            [-1, 256, 3, 3]         147,712
              ReLU-8            [-1, 256, 3, 3]               0
           Flatten-9                 [-1, 2304]               0
           Linear-10                   [-1, 64]         147,520
             ReLU-11                   [-1, 64]               0
           Linear-12                   [-1, 10]             650
          Sigmoid-13                   [-1, 10]               0
================================================================
Total params: 314,698
Trainable params: 314,698
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.56
Params size (MB): 1.20
```

```
Estimated Total Size (MB): 1.76
----------------------------------------------------------------
```

definimos funcion de perdida y optimizador

[ ]:
```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_0.parameters())
```

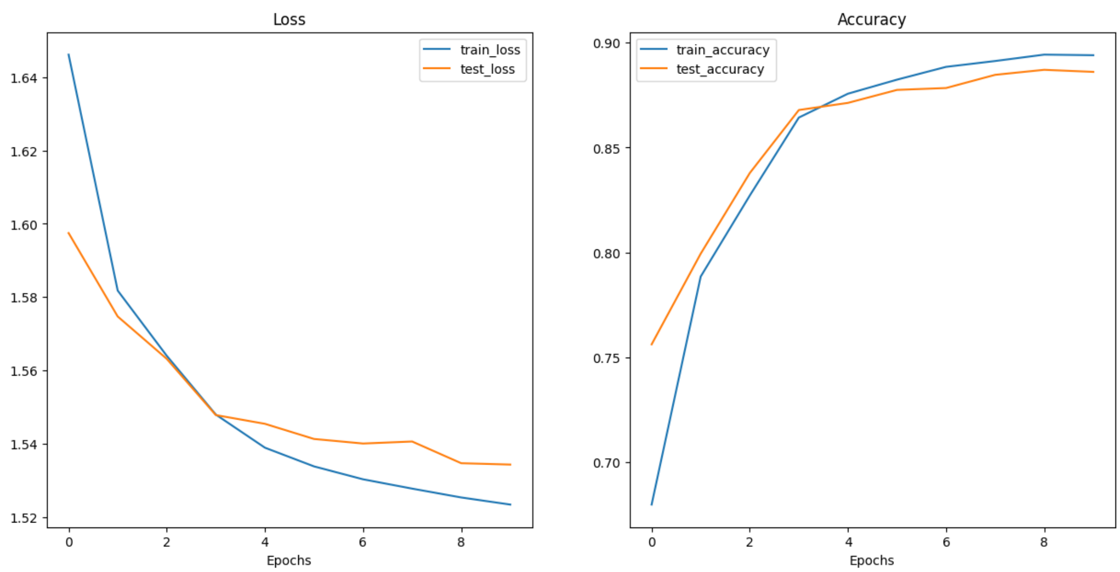
corremos modelo

[ ]:
```
NUM_EPOCHS = 10
model_0_results = train(model=model_0,
                        train_dataloader=trainloader,
                        test_dataloader=testloader,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS,
                        device=device)
```

```
 10%|          | 1/10 [00:08<01:20,  9.00s/it]

Epoch: 1 | train_loss: 1.6461 | train_acc: 0.6798 | test_loss: 1.5975 |
test_acc: 0.7562

 20%|          | 2/10 [00:17<01:10,  8.82s/it]

Epoch: 2 | train_loss: 1.5818 | train_acc: 0.7885 | test_loss: 1.5747 |
test_acc: 0.7994

 30%|          | 3/10 [00:26<01:01,  8.85s/it]

Epoch: 3 | train_loss: 1.5640 | train_acc: 0.8271 | test_loss: 1.5632 |
test_acc: 0.8379

 40%|          | 4/10 [00:35<00:52,  8.81s/it]

Epoch: 4 | train_loss: 1.5480 | train_acc: 0.8643 | test_loss: 1.5479 |
test_acc: 0.8679

 50%|          | 5/10 [00:44<00:43,  8.79s/it]

Epoch: 5 | train_loss: 1.5390 | train_acc: 0.8757 | test_loss: 1.5455 |
test_acc: 0.8713

 60%|          | 6/10 [00:53<00:36,  9.04s/it]

Epoch: 6 | train_loss: 1.5339 | train_acc: 0.8824 | test_loss: 1.5413 |
test_acc: 0.8775

 70%|          | 7/10 [01:02<00:27,  9.04s/it]

Epoch: 7 | train_loss: 1.5303 | train_acc: 0.8885 | test_loss: 1.5401 |
test_acc: 0.8784

 80%|          | 8/10 [01:11<00:18,  9.07s/it]
```

```
Epoch: 8 | train_loss: 1.5278 | train_acc: 0.8913 | test_loss: 1.5406 |
test_acc: 0.8847

 90%|      | 9/10 [01:20<00:08,  8.93s/it]

Epoch: 9 | train_loss: 1.5254 | train_acc: 0.8943 | test_loss: 1.5347 |
test_acc: 0.8871

100%|      | 10/10 [01:29<00:00,  8.92s/it]

Epoch: 10 | train_loss: 1.5235 | train_acc: 0.8940 | test_loss: 1.5344 |
test_acc: 0.8861
```

[ ]:
```
plot_loss_curves(model_0_results)
```



obtenemos una muestra de los datos del test set

[ ]:
```
import random

test_samples = []
test_labels = []

for sample, label in random.sample(list(testset), k=9):
    test_samples.append(sample)
    test_labels.append(label)
```

hacemos predicciones con la muestra que tomamos

[ ]:
```
pred_classes= make_predictions(model=model_0, data=test_samples, device=device)
```

[ ]:
```
classes[pred_classes[0]]
```

[ ]: 'Coat'

visualizamos las predicciones

[ ]:
```
plot_predictions(test_samples, test_labels, classes, pred_classes)
```