MACHINE LEARNING LAB - TUTORIAL 8

Juan Fernando Espinosa

303158

In this homework the objective is to replicate the model developed by the authors Yi Zheng et. al. This model is applying Convolutional Neural Networks (CNN) mixed with a Multi-Layer Perceptron (MLP) to make predictions considering a multivariate time series.

1. ARCHITECTURE OF THE MODEL TO ACCOMPLISH THE RESULT

Brief summary

> First, the input information is a multivariate time series, which means having more than one variable as input signals.
>
> In the pre-processing step it is necessary to treat each variable as univariate and perform feature learning. Normalize the dataset, apply sliding window, and cross-validation (leave-one-out).
>
> Second, the information after the pre-processing steps goes to a 2-stages CNN with the goal of extracting features.
>
> The first convolutional layer contains 8 kernels of size 5, after a subsampling process made by average pooling of size 2 and an activation function calculated by applying Sigmoid function.
>
> The second convolutional with a purpose to help the previous one in the process of extracting features but deeper ones is formed by 4 kernels of size 5, and similarly as the first one with an average pooling and an activation function.
>
> The MLP is formed by 1 hidden layer and takes as input 732 nodes, and gives an output of 4 (there are 4 predicted classes).

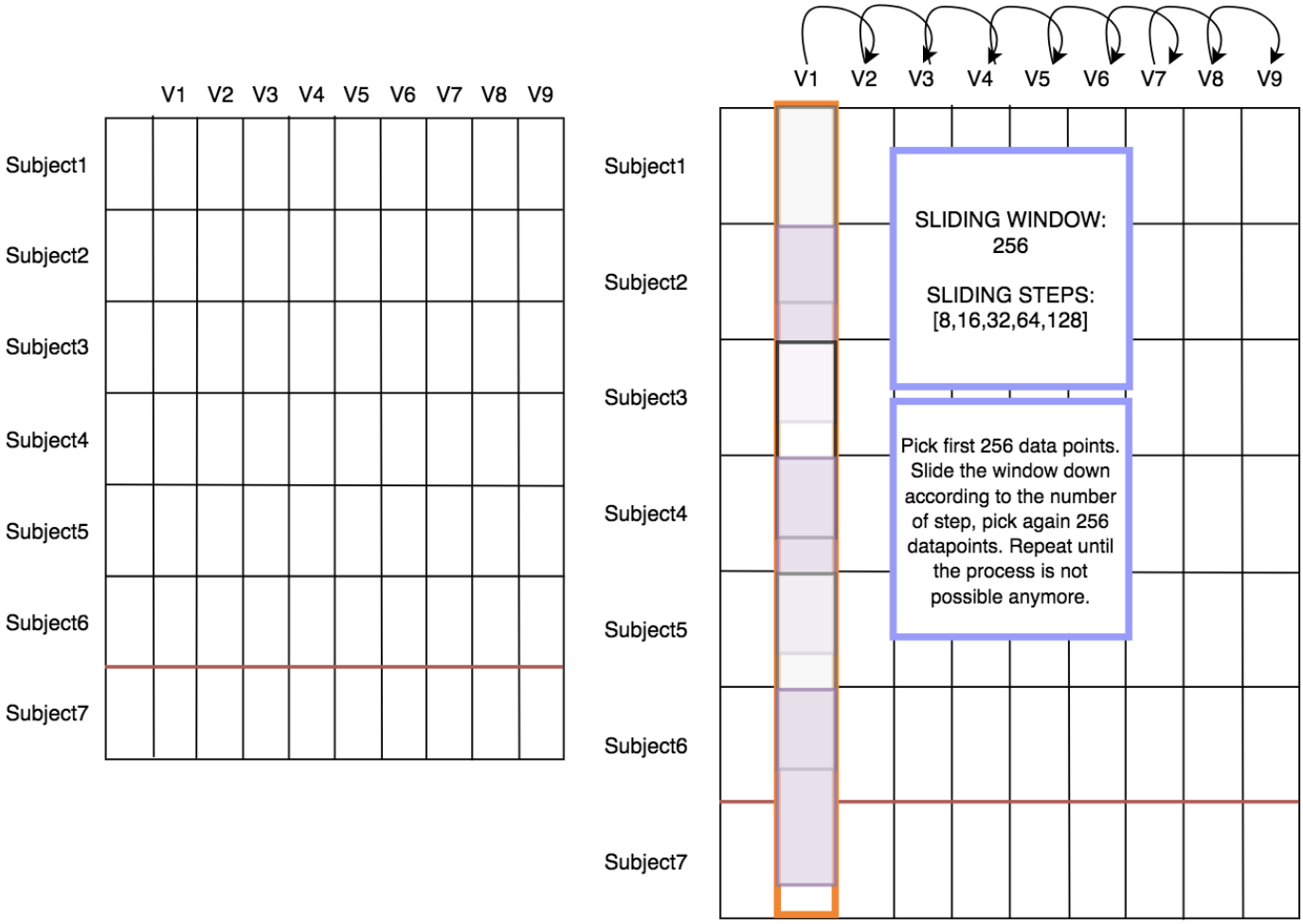
Deep explanation and visualization flow of the process of the model

> After importing the data, concatenating and splitting into train and test the result will be a matrix similar to the image below on the left.

- **Note:** The red line states that subject 7 has been taken as the test set.

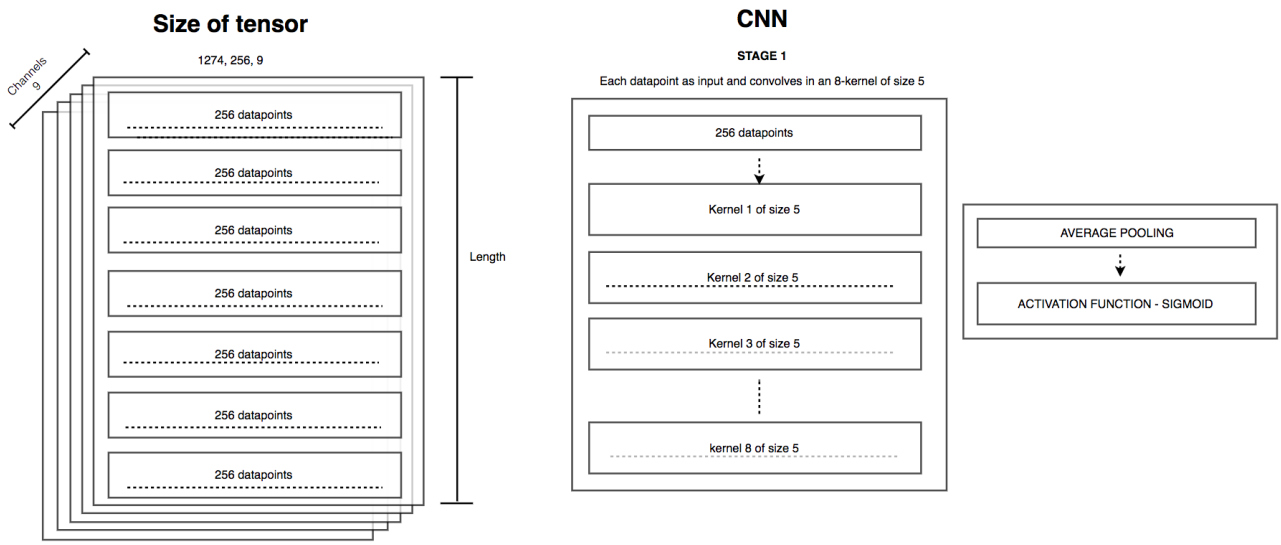
> Next step is sliding window: the box slides down considering 2 factors: **Slide window** that represent the number of datapoints to take and **sliding step** which represents the number of datapoints the windows is going to slide in order to take another group of datapoints.

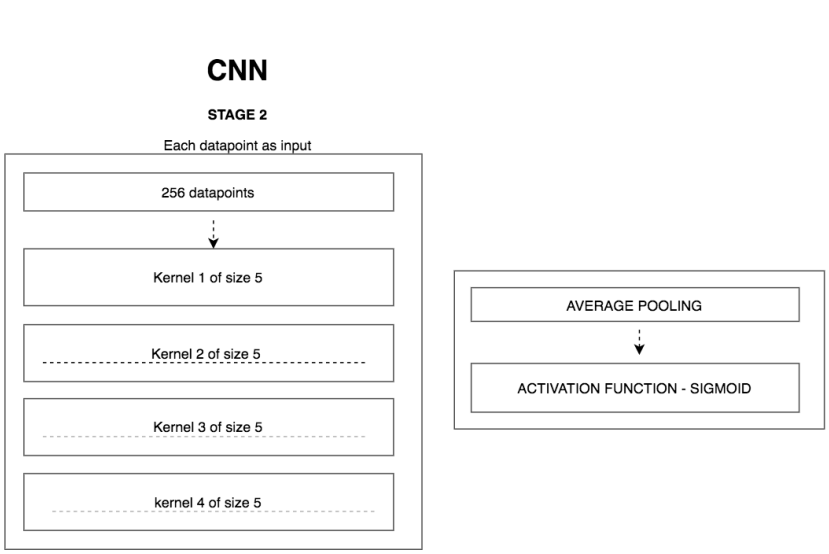
- In the image below it is represented how the sliding window works: first it picks 256 datapoints and the window slides a given number (5 options were given in the paper) to take another 256 datapoints. Since above mentioned, the multivariate time series must be treaten as a univariate one, therefore, we slide over one label only. The same process applies for all the columns of the matrix.



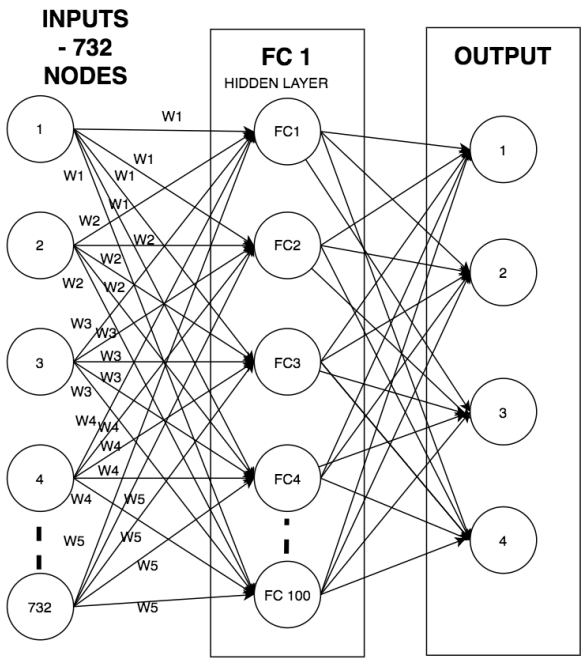
The output will be a 3D Tensor covering 9 channels (1 per column), 256 datapoints per row and the length of the tensor (L x C x R).

Each channel is then used as the input of the CNN. Each row of the channel has to go through the 8 kernels of size 5 in the first stage. Of course, bearing into account the Average Pooling and Activation Function





The output of the CNN then it is flatten to transform it into a 1d array. The length of the 1d array is now the input of the MLP and the output must be 4 according to the number of activities selected in the paper.



Backpropagation: The goal is to minimize the loss function. Since we have 4 activities CrossEntropy function is used.

To minimize the loss function it is required to get the derivative of the loss function considering the gradients of the parameters for a final update of those parameters. It is important to mention for the parameters update the SGD algorithm is used.

MODEL IMPLEMENTATION

2 PRE-PROCESS GIVEN DATASETS

```
import torch
%tensorflow_version 2.x
import tensorflow
import torchvision
from torch.utils.data import Dataset, DataLoader
from torch.utils.tensorboard import SummaryWriter
import torchvision.datasets as dataset
import torchvision.transforms as transforms
import torch.utils.data as Data
from torch.autograd import Variable
```

```
import torch.nn.functional as F
import torch.optim as optim
import torch.nn as nn
from torch.utils.tensorboard import SummaryWriter
import pandas as pd
import numpy as np
from numpy import random
%matplotlib inline
import math
import matplotlib.pyplot as plt
from google.colab import files
from google.colab import drive
```

```
TensorFlow 2.x selected.
```

```
print('PyTorch version:', torch.__version__)
print('Tensor version:',tensorflow.__version__)
%load_ext tensorboard
```

```
PyTorch version: 1.3.1
Tensor version: 2.1.0-rc1
```

```
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject101.dat"
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject102.dat"
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject103.dat"
drive.mount('/content/drive')
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject104.dat"
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject105.dat"
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject106.dat"
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject107.dat"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject101.dat'
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject102.dat'
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject103.dat'
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject104.dat'
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject105.dat'
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject106.dat'
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject107.dat'
```

> The very beginning of the pre-processing step is to upload all the subjects' information with the exception of subjects 8 and 9: one of them has left hand dominance which changes the outputs and subject 9 does not have data for the crucial metrics.

> As a next step a concatenation of all data takes place.

```
missing_values = ['-','na','Nan','nan','n/a','?']
column_names = ['Timestamp','Activity','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22',
subject_1 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject101.dat", sep=' ', na
subject_2 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject102.dat", sep=' ', na
subject_3 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject103.dat", sep=' ', na
subject_4 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject104.dat", sep=' ', na
subject_5 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject105.dat", sep=' ', na
subject_6 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject106.dat", sep=' ', na
subject_7 = pd.read_csv("/content/drive/My Drive/Colab Notebooks/LAB/tutorial 8/PAMAP2_Dataset/Protocol/subject107.dat", sep=' ', na
```

```
subject_1['55'] = 101
subject_2['55'] = 102
subject_3[54] = 103
subject_4[54] = 104
subject_5[54] = 105
subject_6[54] = 106
subject_7['55'] = 107
```

```
frames = [subject_1, subject_2, subject_3, subject_4, subject_5, subject_6]
data = pd.concat(frames)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning: Sorting because non-concatenation axis is not al
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

2.1 SPLIT TRAIN AND TEST SET

Train set

> Considering the dataset given by blocks of subjects' information to keep the order (it is a timeseries database) subject 7 is going to be chosen as the **test set**

> The columns chosen for this exercise are the following: COnsidering data of the first accelerometer (+16g) for each of the datas is recommended in the dataset information. Consequently, those 9 columns were selected.

```
data = data[data['Activity'].isin([3, 4, 12, 13])]
data = data[['Activity', '5', '6', '7', '21', '22', '23', '39', '40', '41']]
data.rename(columns={'5':'a1','6':'a2','7':'a3', '21':'b1', '22':'b2', '23':'b3', '39':'c1', '40':'c2', '41':'c3'}, inplace=True)
data.head()
```

	Activity	a1	a2	a3	b1	b2	b3	c1	c2	c3
53595	3	2.55234	9.24089	3.12614	34.3125	0.459336	9.73112	9.80611	-0.286875	-0.454091
53596	3	2.58398	9.12544	2.97305	34.3125	0.371669	9.72877	9.76538	-0.285674	-0.570285
53597	3	2.54379	8.97364	2.93448	34.3125	0.372713	9.69138	9.83693	-0.285185	-0.685323
53598	3	2.50868	8.78524	3.05005	34.3125	0.258369	9.69174	9.87495	-0.171516	-0.685638
53599	3	2.74210	8.97453	3.20686	34.3125	0.259911	9.84261	9.76958	-0.324928	-0.415640

Test set

```
test = subject_7[subject_7['Activity'].isin([3, 4, 12, 13])]
test = test[['Activity', '5', '6', '7', '21', '22', '23', '39', '40', '41']]
test.rename(columns={'5':'a1','6':'a2','7':'a3', '21':'b1', '22':'b2', '23':'b3', '39':'c1', '40':'c2', '41':'c3'}, inplace=True)
test.head()
```

	Activity	a1	a2	a3	b1	b2	b3	c1	c2	c3
51161	3	-9.35230	2.18326	1.12423	34.8125	1.11146	9.57952	9.71488	-0.461062	-2.30624
51162	3	-9.38929	2.29737	1.12330	34.8125	1.23565	9.84481	9.75490	-0.386004	-2.22912
51163	3	-9.39090	2.25910	1.08491	34.8125	1.27252	9.80678	9.71237	-0.308224	-2.42299
51164	3	-9.31104	2.29746	1.20144	34.8125	1.11236	9.91841	9.60280	-0.422382	-2.30763
51165	3	-9.38983	2.22149	1.12354	34.8125	1.10977	9.80493	9.71488	-0.461062	-2.30624

2.2 NORMALIZATION

Following the instructions of the paper, the normalization method applied is by using the mean and standard deviation.

Train set

```
def normalize(dataset):
    datanorm = (dataset - dataset.mean())/dataset.std()
    datanorm["Activity"]= dataset["Activity"]
    #datanorm["subject"]= dataset["subject"]
    return datanorm

data_train = normalize(data)
data_train.head()
```

```
# DROP MISSING VALUES

missing_values = ['-','na','Nan','nan','n/a','?']
data_train = data.dropna()
data_train.head()

y_train = pd.get_dummies(data_train['Activity']).values

X_train = data_train.drop(['Activity'], axis=1).values
X_train.shape, y_train.shape
```

```
((476713, 9), (476713, 4))
```

Test set

```
test = normalize(test)
test.head()


# DROP MISSING VALUES

missing_values = ['-','na','Nan','nan','n/a','?']
test = test.dropna()
test.head()

y_test = pd.get_dummies(test['Activity'])
y_test = np.argmax(np.array(y_test), axis=1)
X_test = test.drop(['Activity'], axis=1)
X_test.shape, y_test.shape
```

```
((87197, 9), (87197,))
```

2.3 SLIDING WINDOW

Following the above explanation, the process is applied directly to all columns.

```
def window(X,y,step_size):
    sliding_window = 256  #number of datapoints to take
    length_y = len(y)  # Length of target
    sliding_step = length_y // step_size   # Helps to identify the sliding over the variable.
    X_array = []
    y_array = []

    for i in range(0, sliding_step):
        start = i * step_size
        end = sliding_window + (i * step_size)
        if end > length_y:
            start = length_y - sliding_window
            end = length_y
        X_array.append(np.array(X[start:end]))
        a = y[start:end]
        y_array.append(np.array(a.sum() / len(a)))
    X_array = np.array(X_array)
    Y_array = np.array(y_array)
    return X_array, Y_array

# Source = [5]


x_tr, y_tr = window(X_train, y_train, 128)
x_tr = np.moveaxis(x_tr, [0, 1, 2], [0, -1, -2])
x_tr.shape, y_tr.shape
```

```
((3724, 9, 256), (3724,))
```

```
x_te, y_te = window(X_test, y_test, 128)
x_te = np.moveaxis(x_te, [0, 1, 2], [0, -1, -2])
x_te.shape, y_te.shape
```

```
((681, 9, 256), (681,))
```

3. CNN and MLP

3.1 TRANSFORMATION TO TENSORS

```
# Training
x_tr = torch.tensor(x_tr, dtype=torch.float)
y_tr = torch.tensor(y_tr, dtype=torch.long)

# Test
x_te = torch.tensor(x_te, dtype=torch.float)
y_te = torch.tensor(y_te, dtype=torch.long)

# Joining X and Y together in a dataset to ease the prediction process.
datasets_train = torch.utils.data.TensorDataset(x_tr, y_tr)
datasets_test = torch.utils.data.TensorDataset(x_te, y_te)

# Dataloaders for train and test.
train_loader = Data.DataLoader(dataset=datasets_train, batch_size=50, shuffle=False, sampler=None, batch_sampler=None)
test_loader = Data.DataLoader(dataset=datasets_test, batch_size=1, shuffle=False)
x_tr.shape, y_tr.shape, x_te.shape, y_te.shape,
```

```
(torch.Size([3724, 9, 256]),
 torch.Size([3724]),
 torch.Size([681, 9, 256]),
 torch.Size([681]))
```

3.2 CNN STRUCTURE

2 Stages:

1. first: 8 kernels of size 5.
2. second: 4 kernels of size 5.
3. Sigmoid function
4. Average pooling.

> MLP:

1. Input:
2. Output of 1 hidden layer: 100
3. input: 100
4. Final Output: 4

```
class TwoLayerNet(torch.nn.Module):
    def __init__(self):
        super(TwoLayerNet,self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv1d(9, 8, kernel_size=5),
            nn.Sigmoid(),
            nn.AvgPool1d(kernel_size=2, stride=2))
        self.conv2 = nn.Sequential(
            nn.Conv1d(8, 4, kernel_size=5),
            nn.Sigmoid(),
            nn.AvgPool1d(kernel_size=2, stride=2))

        self.fc1 = nn.Linear(244, 100)
        self.fc2 = nn.Linear(100, 4)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

# Source = [3]


model = TwoLayerNet()
print(model)
```

```
    TwoLayerNet(
      (conv1): Sequential(
        (0): Conv1d(9, 8, kernel_size=(5,), stride=(1,))
        (1): Sigmoid()
        (2): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      )
      (conv2): Sequential(
        (0): Conv1d(8, 4, kernel_size=(5,), stride=(1,))
        (1): Sigmoid()
        (2): AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
      )
      (fc1): Linear(in_features=244, out_features=100, bias=True)
      (fc2): Linear(in_features=100, out_features=4, bias=True)
    )



# Optimizer and stablishing the loss function as CrossEntropy
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
loss = torch.nn.CrossEntropyLoss()

epochs = 200
loss_array = []
accuracy_array = []
writer = SummaryWriter('./test5')
total_step = len(train_loader)
for epoch in range(epochs):
  print('---- Epoch:', epoch, ' ----')
  loss_epoch = 0.0
  losses = 0.0
  correct = 0.0
  total_size = 0.0
  accuracy = 0.0
  accuracy_epoch = 0.0
  #print("{Epoch}", epoch)
  for i, (batch_X, batch_y) in enumerate(train_loader, 0):    # for each training step

      X_temporal, y_temporal = Variable(batch_X), Variable(batch_y)

      # It is required to clear the gradients
      optimizer.zero_grad()

      #Forward pass - model prediction
      outputs = model(X_temporal.float())
      y_temporal = y_temporal.squeeze_()
      loss_size = loss(outputs, y_temporal)
      total_size += y_temporal.size(0)
      _, predicted = torch.max(outputs.data, 1)
      correct += (predicted == y_temporal).sum().item()
      accuracy = correct / total_size

      # Backpropagation
      loss_size.backward()

      optimizer.step()
      losses += loss_size.data



  loss_epoch = losses.item() / len(data)
  loss_array.append(losses.item() / len(data))
  accuracy_epoch = accuracy / len(data)
  accuracy_array.append(accuracy / len(data)*100)
  print('Loss Trainig Set:', losses.item() / 50)
  writer.add_scalar('Loss/train', losses.item(), epoch)
  writer.add_scalar('Accuracy/train', correct / total_size, epoch)




# test
  for i, (batch_X, batch_y) in enumerate(test_loader, 0):

      X_temporal, y_temporal = Variable(batch_X), Variable(batch_y)

      # It is required to clear the gradients
      optimizer.zero_grad()

      #Forward pass - model prediction
      outputs = model(X_temporal.float())
      loss_size = loss(outputs, y_temporal)
      total_size += y_temporal.size(0)
         predicted = torch.max(outputs.data, 1)
```

```
      _, predicted = torch.max(outputs.data, 1)
      correct += (predicted == y_temporal).sum().item()
      accuracy = (correct / total_size)*100

      # Backpropagation
      loss_size.backward()
      optimizer.step()
      losses += loss_size.data

  accuracy_epoch = accuracy / len(data)
  accuracy_array.append(accuracy / len(data)*100)
  print('Accuracy on Test set:', (correct / total_size) * 100, '\n')
  writer.add_scalar('Loss/test', losses.item(), epoch)
  writer.add_scalar('Accuracy/test', correct / total_size, epoch)
writer.close()
# Source: [3]
```

```
---- Epoch: 0  ----
Loss Trainig Set: 1.3528713989257812
Accuracy on Test set: 82.7922814982974

---- Epoch: 1  ----
Loss Trainig Set: 0.11173728942871093
Accuracy on Test set: 94.23382519863792

---- Epoch: 2  ----
Loss Trainig Set: 0.09959948539733887
Accuracy on Test set: 94.14301929625427

---- Epoch: 3  ----
Loss Trainig Set: 0.09494118690490723
Accuracy on Test set: 94.0295119182747

---- Epoch: 4  ----
Loss Trainig Set: 0.09487771987915039
Accuracy on Test set: 93.91600454029512

---- Epoch: 5  ----
Loss Trainig Set: 0.09714068412780762
Accuracy on Test set: 93.84790011350738

---- Epoch: 6  ----
Loss Trainig Set: 0.09885738372802734
Accuracy on Test set: 93.82519863791147

---- Epoch: 7  ----
Loss Trainig Set: 0.09796657562255859
Accuracy on Test set: 93.82519863791147

---- Epoch: 8  ----
Loss Trainig Set: 0.09387980461120606
Accuracy on Test set: 93.80249716231555

---- Epoch: 9  ----
Loss Trainig Set: 0.08743919372558594
Accuracy on Test set: 93.80249716231555

---- Epoch: 10  ----
Loss Trainig Set: 0.08075057983398437
Accuracy on Test set: 93.7116912599319

---- Epoch: 11  ----
Loss Trainig Set: 0.07605184555053711
Accuracy on Test set: 93.57548240635641

---- Epoch: 12  ----
Loss Trainig Set: 0.07430627822875976
Accuracy on Test set: 93.48467650397276

---- Epoch: 13  ----
Loss Trainig Set: 0.07471656322479248
Accuracy on Test set: 93.41657207718501

---- Epoch: 14  ----
Loss Trainig Set: 0.07543406486511231
Accuracy on Test set: 93.41657207718501

---- Epoch: 15  ----
Loss Trainig Set: 0.07359809875488281
Accuracy on Test set: 93.50737797956867

---- Epoch: 16  ----
Loss Trainig Set: 0.06725871086120605
Accuracy on Test set: 93.57548240635641

---- Epoch: 17  ----
Loss Trainig Set: 0.05697666168212891
Accuracy on Test set: 93.64358683314416

---- Epoch: 18  ----
Loss Trainig Set: 0.04482060432434082
Accuracy on Test set: 93.77979568671964

---- Epoch: 19  ----
Loss Trainig Set: 0.03295360565185547
Accuracy on Test set: 94.00681044267878

---- Epoch: 20  ----
Loss Trainig Set: 0.02265591621398926
Accuracy on Test set: 94.46083995459705

---- Epoch: 21  ----
Loss Trainig Set: 0.014677591323852539
Accuracy on Test set: 95.18728717366629

---- Epoch: 22  ----
Loss Trainig Set: 0.009527305364608765
Accuracy on Test set: 95.70942111237231
```

Accuracy on Test set: 95.70942111237231

---- Epoch: 23 ----
Loss Trainig Set: 0.006757386922836304
Accuracy on Test set: 96.29965947786606

---- Epoch: 24 ----
Loss Trainig Set: 0.0053076142072677615
Accuracy on Test set: 96.84449489216799

---- Epoch: 25 ----
Loss Trainig Set: 0.004411972165107727
Accuracy on Test set: 97.16231555051078

---- Epoch: 26 ----
Loss Trainig Set: 0.003869430124759674
Accuracy on Test set: 97.38933030646993

---- Epoch: 27 ----
Loss Trainig Set: 0.003630574941635132
Accuracy on Test set: 97.61634506242906

---- Epoch: 28 ----
Loss Trainig Set: 0.0036032018065452576
Accuracy on Test set: 97.72985244040862

---- Epoch: 29 ----
Loss Trainig Set: 0.0037097343802452087
Accuracy on Test set: 97.79795686719636

---- Epoch: 30 ----
Loss Trainig Set: 0.003904317319393158
Accuracy on Test set: 97.8660612939841

---- Epoch: 31 ----
Loss Trainig Set: 0.004162830114364624
Accuracy on Test set: 97.91146424517594

---- Epoch: 32 ----
Loss Trainig Set: 0.004474236369132995
Accuracy on Test set: 98.00227014755959

---- Epoch: 33 ----
Loss Trainig Set: 0.004836486279964447
Accuracy on Test set: 98.00227014755959

---- Epoch: 34 ----
Loss Trainig Set: 0.00525469958782196
Accuracy on Test set: 98.00227014755959

---- Epoch: 35 ----
Loss Trainig Set: 0.005740604996681213
Accuracy on Test set: 97.97956867196368

---- Epoch: 36 ----
Loss Trainig Set: 0.006313272714614868
Accuracy on Test set: 97.97956867196368

---- Epoch: 37 ----
Loss Trainig Set: 0.0070012760162353515
Accuracy on Test set: 97.97956867196368

---- Epoch: 38 ----
Loss Trainig Set: 0.00784694790840149
Accuracy on Test set: 98.02497162315551

---- Epoch: 39 ----
Loss Trainig Set: 0.008913355469703675
Accuracy on Test set: 98.02497162315551

---- Epoch: 40 ----
Loss Trainig Set: 0.01029518485069275
Accuracy on Test set: 98.02497162315551

---- Epoch: 41 ----
Loss Trainig Set: 0.012134934663772584
Accuracy on Test set: 98.02497162315551

---- Epoch: 42 ----
Loss Trainig Set: 0.014643864631652832
Accuracy on Test set: 98.04767309875142

---- Epoch: 43 ----
Loss Trainig Set: 0.01812062382698059
Accuracy on Test set: 98.11577752553916

---- Epoch: 44 ----
Loss Trainig Set: 0.022941431999206542
Accuracy on Test set: 98.20658342792281

---- Epoch: 45 ----
Loss Trainig Set: 0.029462559223175047

```
Loss Trainig Set: 0.029402959225175047
Accuracy on Test set: 98.27468785471055

---- Epoch: 46  ----
Loss Trainig Set: 0.03775681018829346
Accuracy on Test set: 98.29738933030647

---- Epoch: 47  ----
Loss Trainig Set: 0.04719242572784424
Accuracy on Test set: 98.45629965947786

---- Epoch: 48  ----
Loss Trainig Set: 0.05629948616027832
Accuracy on Test set: 98.47900113507379

---- Epoch: 49  ----
Loss Trainig Set: 0.06413933753967285
Accuracy on Test set: 98.47900113507379

---- Epoch: 50  ----
Loss Trainig Set: 0.0708885669708252
Accuracy on Test set: 98.56980703745744

---- Epoch: 51  ----
Loss Trainig Set: 0.0763835620880127
Accuracy on Test set: 98.61520998864927

---- Epoch: 52  ----
Loss Trainig Set: 0.08043878555297851
Accuracy on Test set: 98.63791146424518

---- Epoch: 53  ----
Loss Trainig Set: 0.08313610076904297
Accuracy on Test set: 98.66061293984109

---- Epoch: 54  ----
Loss Trainig Set: 0.08469927787780762
Accuracy on Test set: 98.68331441543701

---- Epoch: 55  ----
Loss Trainig Set: 0.08537615776062012
Accuracy on Test set: 98.70601589103292

---- Epoch: 56  ----
Loss Trainig Set: 0.08538357734680176
Accuracy on Test set: 98.72871736662883

---- Epoch: 57  ----
Loss Trainig Set: 0.0848919677734375
Accuracy on Test set: 98.75141884222475

---- Epoch: 58  ----
Loss Trainig Set: 0.08402725219726563
Accuracy on Test set: 98.77412031782066

---- Epoch: 59  ----
Loss Trainig Set: 0.08287957191467285
Accuracy on Test set: 98.77412031782066

---- Epoch: 60  ----
Loss Trainig Set: 0.08151206970214844
Accuracy on Test set: 98.77412031782066

---- Epoch: 61  ----
Loss Trainig Set: 0.07996836185455322
Accuracy on Test set: 98.77412031782066

---- Epoch: 62  ----
Loss Trainig Set: 0.07827825069427491
Accuracy on Test set: 98.77412031782066

---- Epoch: 63  ----
Loss Trainig Set: 0.07646236896514892
Accuracy on Test set: 98.79682179341657

---- Epoch: 64  ----
Loss Trainig Set: 0.07453476905822753
Accuracy on Test set: 98.79682179341657

---- Epoch: 65  ----
Loss Trainig Set: 0.07250589370727539
Accuracy on Test set: 98.79682179341657

---- Epoch: 66  ----
Loss Trainig Set: 0.07038358688354492
Accuracy on Test set: 98.79682179341657

---- Epoch: 67  ----
Loss Trainig Set: 0.06817521572113037
Accuracy on Test set: 98.81952326901249

---- Epoch: 68  ----
```

---- Epoch: 68 ----
Loss Trainig Set: 0.06588799953460693
Accuracy on Test set: 98.8422247446084

---- Epoch: 69 ----
Loss Trainig Set: 0.06353002071380615
Accuracy on Test set: 98.8422247446084

---- Epoch: 70 ----
Loss Trainig Set: 0.06111058235168457
Accuracy on Test set: 98.8422247446084

---- Epoch: 71 ----
Loss Trainig Set: 0.0586406135559082
Accuracy on Test set: 98.86492622020431

---- Epoch: 72 ----
Loss Trainig Set: 0.05613212585449219
Accuracy on Test set: 98.8422247446084

---- Epoch: 73 ----
Loss Trainig Set: 0.05359914302825928
Accuracy on Test set: 98.88762769580023

---- Epoch: 74 ----
Loss Trainig Set: 0.051056380271911624
Accuracy on Test set: 98.88762769580023

---- Epoch: 75 ----
Loss Trainig Set: 0.048519530296325684
Accuracy on Test set: 98.88762769580023

---- Epoch: 76 ----
Loss Trainig Set: 0.04600436210632324
Accuracy on Test set: 98.88762769580023

---- Epoch: 77 ----
Loss Trainig Set: 0.04352630615234375
Accuracy on Test set: 98.91032917139614

---- Epoch: 78 ----
Loss Trainig Set: 0.04110018730163574
Accuracy on Test set: 98.91032917139614

---- Epoch: 79 ----
Loss Trainig Set: 0.03873954296112061
Accuracy on Test set: 98.93303064699207

---- Epoch: 80 ----
Loss Trainig Set: 0.036456246376037595
Accuracy on Test set: 98.93303064699207

---- Epoch: 81 ----
Loss Trainig Set: 0.03426039218902588
Accuracy on Test set: 98.91032917139614

---- Epoch: 82 ----
Loss Trainig Set: 0.03216009616851807
Accuracy on Test set: 98.91032917139614

---- Epoch: 83 ----
Loss Trainig Set: 0.030161404609680177
Accuracy on Test set: 98.86492622020431

---- Epoch: 84 ----
Loss Trainig Set: 0.02826841115951538
Accuracy on Test set: 98.86492622020431

---- Epoch: 85 ----
Loss Trainig Set: 0.026483421325683595
Accuracy on Test set: 98.86492622020431

---- Epoch: 86 ----
Loss Trainig Set: 0.024806904792785644
Accuracy on Test set: 98.86492622020431

---- Epoch: 87 ----
Loss Trainig Set: 0.023237860202789305
Accuracy on Test set: 98.86492622020431

---- Epoch: 88 ----
Loss Trainig Set: 0.021774139404296875
Accuracy on Test set: 98.86492622020431

---- Epoch: 89 ----
Loss Trainig Set: 0.02041248798370361
Accuracy on Test set: 98.86492622020431

---- Epoch: 90 ----
Loss Trainig Set: 0.019148777723312378
Accuracy on Test set: 98.86492622020431

```
---- Epoch: 91  ----
Loss Trainig Set: 0.0179783034324646
Accuracy on Test set: 98.86492622020431

---- Epoch: 92  ----
Loss Trainig Set: 0.016896088123321534
Accuracy on Test set: 98.86492622020431

---- Epoch: 93  ----
Loss Trainig Set: 0.015896685123443603
Accuracy on Test set: 98.88762769580023

---- Epoch: 94  ----
Loss Trainig Set: 0.014974693059921265
Accuracy on Test set: 98.86492622020431

---- Epoch: 95  ----
Loss Trainig Set: 0.014124770164489747
Accuracy on Test set: 98.86492622020431

---- Epoch: 96  ----
Loss Trainig Set: 0.013341482877731323
Accuracy on Test set: 98.86492622020431

---- Epoch: 97  ----
Loss Trainig Set: 0.01261976957321167
Accuracy on Test set: 98.86492622020431

---- Epoch: 98  ----
Loss Trainig Set: 0.01195473074913025
Accuracy on Test set: 98.86492622020431

---- Epoch: 99  ----
Loss Trainig Set: 0.011341691017150879
Accuracy on Test set: 98.86492622020431

---- Epoch: 100  ----
Loss Trainig Set: 0.010776317119598389
Accuracy on Test set: 98.86492622020431

---- Epoch: 101  ----
Loss Trainig Set: 0.010254501104354859
Accuracy on Test set: 98.88762769580023

---- Epoch: 102  ----
Loss Trainig Set: 0.009772502183914185
Accuracy on Test set: 98.91032917139614

---- Epoch: 103  ----
Loss Trainig Set: 0.00932681441307068
Accuracy on Test set: 98.91032917139614

---- Epoch: 104  ----
Loss Trainig Set: 0.00891421377658844
Accuracy on Test set: 98.91032917139614

---- Epoch: 105  ----
Loss Trainig Set: 0.008531791567802429
Accuracy on Test set: 98.93303064699207

---- Epoch: 106  ----
Loss Trainig Set: 0.008176856637001038
Accuracy on Test set: 98.95573212258797

---- Epoch: 107  ----
Loss Trainig Set: 0.007846954464912414
Accuracy on Test set: 98.95573212258797

---- Epoch: 108  ----
Loss Trainig Set: 0.007539854049682617
Accuracy on Test set: 98.97843359818388

---- Epoch: 109  ----
Loss Trainig Set: 0.007253522276878357
Accuracy on Test set: 99.0011350737798

---- Epoch: 110  ----
Loss Trainig Set: 0.006986134648323059
Accuracy on Test set: 99.0011350737798

---- Epoch: 111  ----
Loss Trainig Set: 0.006736008524894714
Accuracy on Test set: 99.02383654937572

---- Epoch: 112  ----
Loss Trainig Set: 0.0065016233921051025
Accuracy on Test set: 99.02383654937572

---- Epoch: 113  ----
Loss Trainig Set: 0.006281638145446777
Accuracy on Test set: 99.02383654937572
```

Accuracy on Test set: 99.02383654937572

---- Epoch: 114 ----
Loss Trainig Set: 0.0060747754573822026
Accuracy on Test set: 99.02383654937572

---- Epoch: 115 ----
Loss Trainig Set: 0.00587993860244751
Accuracy on Test set: 99.02383654937572

---- Epoch: 116 ----
Loss Trainig Set: 0.005696080923080445
Accuracy on Test set: 99.04653802497162

---- Epoch: 117 ----
Loss Trainig Set: 0.005522279143333435
Accuracy on Test set: 99.04653802497162

---- Epoch: 118 ----
Loss Trainig Set: 0.005357692837715149
Accuracy on Test set: 99.04653802497162

---- Epoch: 119 ----
Loss Trainig Set: 0.005201565027236938
Accuracy on Test set: 99.04653802497162

---- Epoch: 120 ----
Loss Trainig Set: 0.005053222775459289
Accuracy on Test set: 99.04653802497162

---- Epoch: 121 ----
Loss Trainig Set: 0.00491199791431427
Accuracy on Test set: 99.04653802497162

---- Epoch: 122 ----
Loss Trainig Set: 0.004777364134788513
Accuracy on Test set: 99.04653802497162

---- Epoch: 123 ----
Loss Trainig Set: 0.004648785889148712
Accuracy on Test set: 99.04653802497162

---- Epoch: 124 ----
Loss Trainig Set: 0.004525805115699768
Accuracy on Test set: 99.04653802497162

---- Epoch: 125 ----
Loss Trainig Set: 0.004407954514026642
Accuracy on Test set: 99.04653802497162

---- Epoch: 126 ----
Loss Trainig Set: 0.004294915497303009
Accuracy on Test set: 99.04653802497162

---- Epoch: 127 ----
Loss Trainig Set: 0.00418625682592392
Accuracy on Test set: 99.02383654937572

---- Epoch: 128 ----
Loss Trainig Set: 0.004081691801548004
Accuracy on Test set: 99.02383654937572

---- Epoch: 129 ----
Loss Trainig Set: 0.0039809256792068485
Accuracy on Test set: 99.02383654937572

---- Epoch: 130 ----
Loss Trainig Set: 0.0038837096095085144
Accuracy on Test set: 99.04653802497162

---- Epoch: 131 ----
Loss Trainig Set: 0.0037897470593452453
Accuracy on Test set: 99.04653802497162

---- Epoch: 132 ----
Loss Trainig Set: 0.0036988303065299986
Accuracy on Test set: 99.04653802497162

---- Epoch: 133 ----
Loss Trainig Set: 0.0036107584834098815
Accuracy on Test set: 99.04653802497162

---- Epoch: 134 ----
Loss Trainig Set: 0.003525364100933075
Accuracy on Test set: 99.04653802497162

---- Epoch: 135 ----
Loss Trainig Set: 0.00344243198633194
Accuracy on Test set: 99.04653802497162

---- Epoch: 136 ----
Loss Trainig Set: 0.00336184769868850?

```
Loss Trainig Set: 0.0055018470900000507
Accuracy on Test set: 99.06923950056753

---- Epoch: 137  ----
Loss Trainig Set: 0.0032834210991859437
Accuracy on Test set: 99.09194097616346

---- Epoch: 138  ----
Loss Trainig Set: 0.0032070797681808473
Accuracy on Test set: 99.11464245175937

---- Epoch: 139  ----
Loss Trainig Set: 0.0031326383352279 66
Accuracy on Test set: 99.11464245175937

---- Epoch: 140  ----
Loss Trainig Set: 0.0030600169301033 02
Accuracy on Test set: 99.11464245175937

---- Epoch: 141  ----
Loss Trainig Set: 0.0029891058802604675
Accuracy on Test set: 99.11464245175937

---- Epoch: 142  ----
Loss Trainig Set: 0.0029198026657104494
Accuracy on Test set: 99.13734392735527

---- Epoch: 143  ----
Loss Trainig Set: 0.0028520533442497253
Accuracy on Test set: 99.13734392735527

---- Epoch: 144  ----
Loss Trainig Set: 0.0027857539057731 63
Accuracy on Test set: 99.13734392735527

---- Epoch: 145  ----
Loss Trainig Set: 0.0027208283543586733
Accuracy on Test set: 99.13734392735527

---- Epoch: 146  ----
Loss Trainig Set: 0.0026572221517562867
Accuracy on Test set: 99.13734392735527

---- Epoch: 147  ----
Loss Trainig Set: 0.0025948596000671385
Accuracy on Test set: 99.13734392735527

---- Epoch: 148  ----
Loss Trainig Set: 0.0025336796045303343
Accuracy on Test set: 99.1600454029512

---- Epoch: 149  ----
Loss Trainig Set: 0.0024736471474170684
Accuracy on Test set: 99.1827468785471

---- Epoch: 150  ----
Loss Trainig Set: 0.002414710819721222
Accuracy on Test set: 99.20544835414302

---- Epoch: 151  ----
Loss Trainig Set: 0.0023567625880241395
Accuracy on Test set: 99.20544835414302

---- Epoch: 152  ----
Loss Trainig Set: 0.0022998142242431643
Accuracy on Test set: 99.20544835414302

---- Epoch: 153  ----
Loss Trainig Set: 0.0022438125312328337
Accuracy on Test set: 99.20544835414302

---- Epoch: 154  ----
Loss Trainig Set: 0.0021886573731899 26
Accuracy on Test set: 99.22814982973892

---- Epoch: 155  ----
Loss Trainig Set: 0.0021343477070331573
Accuracy on Test set: 99.22814982973892

---- Epoch: 156  ----
Loss Trainig Set: 0.0020808764323472977
Accuracy on Test set: 99.22814982973892

---- Epoch: 157  ----
Loss Trainig Set: 0.0020278942584991454
Accuracy on Test set: 99.22814982973892

---- Epoch: 158  ----
Loss Trainig Set: 0.0019756358861923217
Accuracy on Test set: 99.27355278093076

---- Epoch: 159  ----
```

---- Epoch: 159 ----
Loss Trainig Set: 0.0019239233434200287
Accuracy on Test set: 99.29625425652667

---- Epoch: 160 ----
Loss Trainig Set: 0.0018726804852485658
Accuracy on Test set: 99.29625425652667

---- Epoch: 161 ----
Loss Trainig Set: 0.0018218421936035155
Accuracy on Test set: 99.31895573212259

---- Epoch: 162 ----
Loss Trainig Set: 0.0017713008821010 59
Accuracy on Test set: 99.3416572077185

---- Epoch: 163 ----
Loss Trainig Set: 0.0017209826409816742
Accuracy on Test set: 99.3643586833144

---- Epoch: 164 ----
Loss Trainig Set: 0.0016707997024059296
Accuracy on Test set: 99.3643586833144

---- Epoch: 165 ----
Loss Trainig Set: 0.0016206964850425 72
Accuracy on Test set: 99.3643586833144

---- Epoch: 166 ----
Loss Trainig Set: 0.0015706376731395722
Accuracy on Test set: 99.3643586833144

---- Epoch: 167 ----
Loss Trainig Set: 0.0015205772221088 41
Accuracy on Test set: 99.3643586833144

---- Epoch: 168 ----
Loss Trainig Set: 0.0014705103635787964
Accuracy on Test set: 99.3643586833144

---- Epoch: 169 ----
Loss Trainig Set: 0.0014204600453376 77
Accuracy on Test set: 99.38706015891033

---- Epoch: 170 ----
Loss Trainig Set: 0.0013704302906990052
Accuracy on Test set: 99.38706015891033

---- Epoch: 171 ----
Loss Trainig Set: 0.0013205134868621827
Accuracy on Test set: 99.38706015891033

---- Epoch: 172 ----
Loss Trainig Set: 0.0012707738578319 55
Accuracy on Test set: 99.38706015891033

---- Epoch: 173 ----
Loss Trainig Set: 0.0012213038653135 3
Accuracy on Test set: 99.38706015891033

---- Epoch: 174 ----
Loss Trainig Set: 0.0011722233146429061
Accuracy on Test set: 99.38706015891033

---- Epoch: 175 ----
Loss Trainig Set: 0.0011236266046762465
Accuracy on Test set: 99.43246311010215

---- Epoch: 176 ----
Loss Trainig Set: 0.0010756516456604003
Accuracy on Test set: 99.43246311010215

---- Epoch: 177 ----
Loss Trainig Set: 0.0010283835232257842
Accuracy on Test set: 99.43246311010215

---- Epoch: 178 ----
Loss Trainig Set: 0.0009819370508193969
Accuracy on Test set: 99.43246311010215

---- Epoch: 179 ----
Loss Trainig Set: 0.0009364219754934 31
Accuracy on Test set: 99.45516458569807

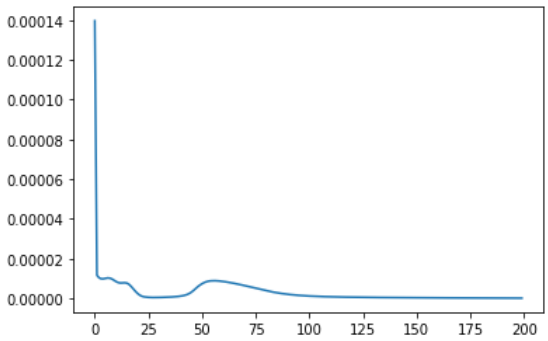
---- Epoch: 180 ----
Loss Trainig Set: 0.0008919436484575271
Accuracy on Test set: 99.45516458569807

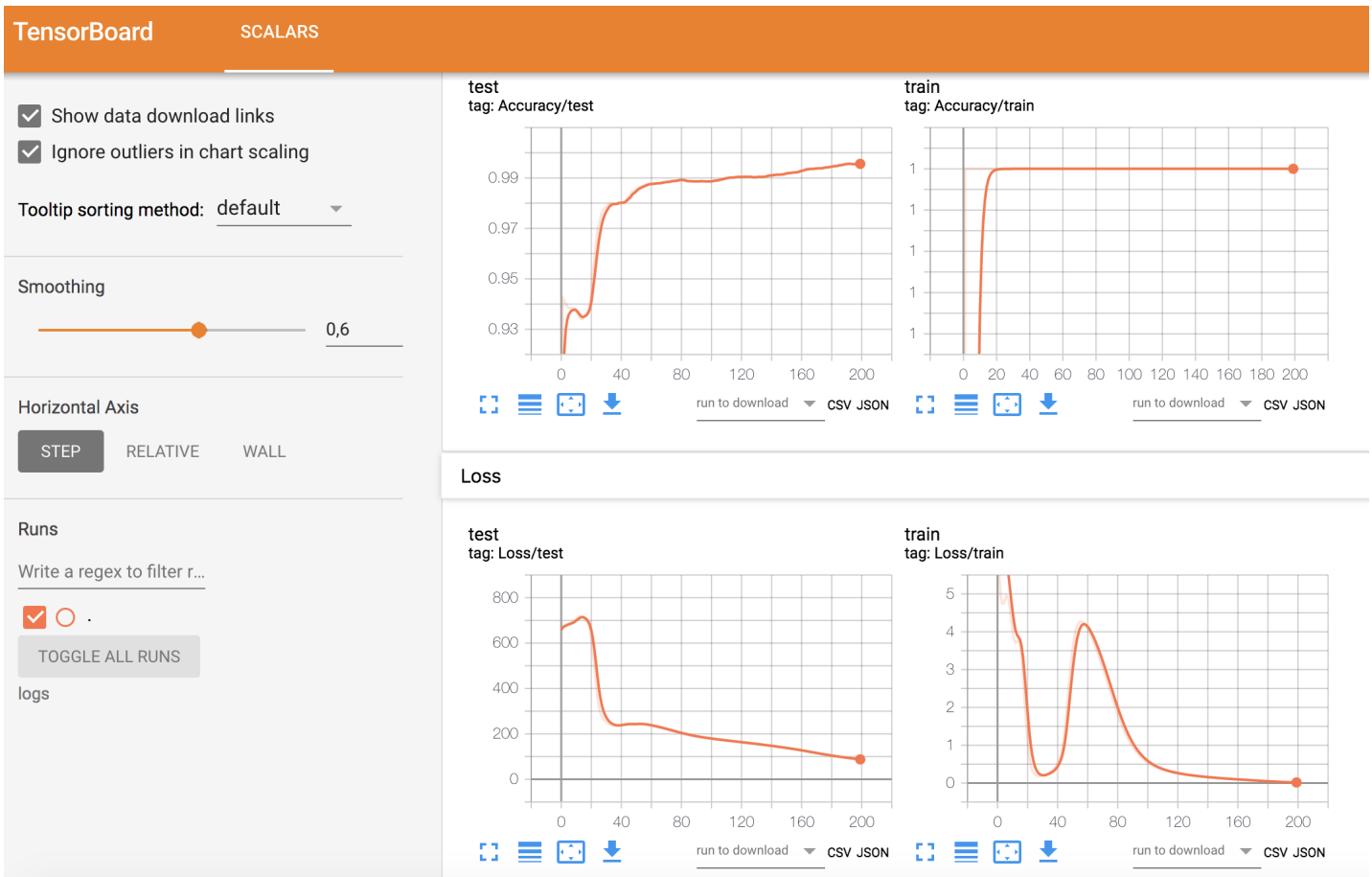
---- Epoch: 181 ----
Loss Trainig Set: 0.0008485665917396545
Accuracy on Test set: 99.47786606129398

```
---- Epoch: 182  ----
Loss Trainig Set: 0.0008063439279794693
Accuracy on Test set: 99.47786606129398

---- Epoch: 183  ----
Loss Trainig Set: 0.0007653781026601791
Accuracy on Test set: 99.47786606129398

---- Epoch: 184  ----
Loss Trainig Set: 0.0007256978005170822
Accuracy on Test set: 99.50056753688989

---- Epoch: 185  ----
Loss Trainig Set: 0.0006873390823602676
Accuracy on Test set: 99.52326901248581

---- Epoch: 186  ----
Loss Trainig Set: 0.0006503272801637649
Accuracy on Test set: 99.52326901248581

---- Epoch: 187  ----
Loss Trainig Set: 0.0006147176027297974
Accuracy on Test set: 99.52326901248581

---- Epoch: 188  ----
Loss Trainig Set: 0.0005804760009050369
Accuracy on Test set: 99.54597048808172

---- Epoch: 189  ----
Loss Trainig Set: 0.0005476345866918564
Accuracy on Test set: 99.56867196367763

---- Epoch: 190  ----
Loss Trainig Set: 0.0005161765962839127
Accuracy on Test set: 99.56867196367763

---- Epoch: 191  ----
Loss Trainig Set: 0.0004861053824424743 6
Accuracy on Test set: 99.56867196367763

---- Epoch: 192  ----
Loss Trainig Set: 0.0004573987424373626 5
Accuracy on Test set: 99.56867196367763

---- Epoch: 193  ----
Loss Trainig Set: 0.0004300390928983688 5
Accuracy on Test set: 99.54597048808172

---- Epoch: 194  ----
Loss Trainig Set: 0.0004040033370256424
Accuracy on Test set: 99.54597048808172

---- Epoch: 195  ----
Loss Trainig Set: 0.0003792398795485496
Accuracy on Test set: 99.54597048808172

---- Epoch: 196  ----
Loss Trainig Set: 0.0003557554632425308 4
Accuracy on Test set: 99.54597048808172

---- Epoch: 197  ----
Loss Trainig Set: 0.0003334928303956985 6
Accuracy on Test set: 99.54597048808172

---- Epoch: 198  ----
Loss Trainig Set: 0.0003124228306114674
Accuracy on Test set: 99.56867196367763

---- Epoch: 199  ----
Loss Trainig Set: 0.0002924678102135658
Accuracy on Test set: 99.56867196367763
```

```
plt.plot(loss_array)
print(loss_array)
plt.show()
```

```
[0.0001397346131602423, 1.1541057728245698e-05, 1.0287375114890625e-05, 9.806231424262138e-06, 9.799676078129753e-06, 1.0033411
```





4. CONCLUSIONS

> After 200 epochs the accuracy obtained in the test set is 99%. It is really high but it happens because of the number of iterations. If the number of iterations is reduced to 40, the dropping in accuracy is of around 8%.

> The distribution of train and test seems to be not good and cross validation is fundamental for this exercises to avoid overfitting.

> By using Adam for learning rate the algorithm runs faster and reaches a good aaccuracy.

> Pytorch directly allows you to use minibatches. Therefore, if at the end minibatches are still used could be an opportunity to test results with different type of optimizer.

> It was not possible to replicate the results of the paper due to a not-well-stablished used of the columns. In this exercise, 9 columns were used. It generates a good accuracy percentage.

5. BIBLIOGRAPHY:

[1] Ackermann, N. (2018, September 14). Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences. Retrieved from https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

[2] Brownlee, J. (2019, August 5). How to Develop Convolutional Neural Network Models for Time Series Forecasting. Retrieved from https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/.

[3] Convolutional Neural Networks Tutorial in PyTorch. (2018, June 16). Retrieved from https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/.

[4] torch.utils.tensorboard¶. (n.d.). Retrieved from https://pytorch.org/docs/stable/tensorboard.html.

[5] Window Sliding Technique. (2019, October 25). Retrieved from https://www.geeksforgeeks.org/window-sliding-technique/.

[6] Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014, June). Time series classification using multi-channels deep convolutional neural networks. In International Conference on Web-Age Information Management (pp. 298-310). Springer, Cham.