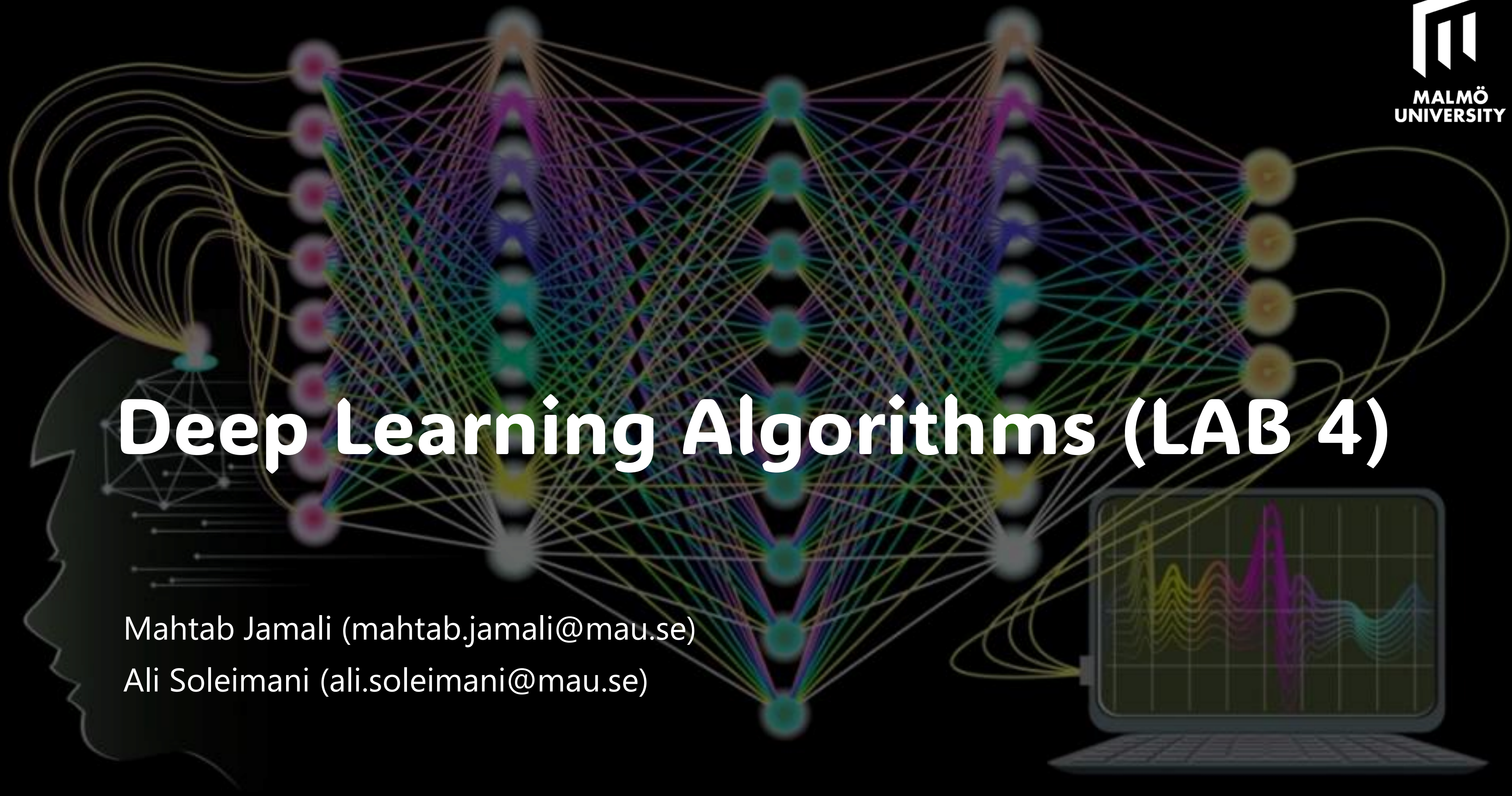


MALMÖ UNIVERSITY

# Deep Learning Algorithms (LAB 4)

Mahtab Jamali (mahtab.jamali@mau.se)

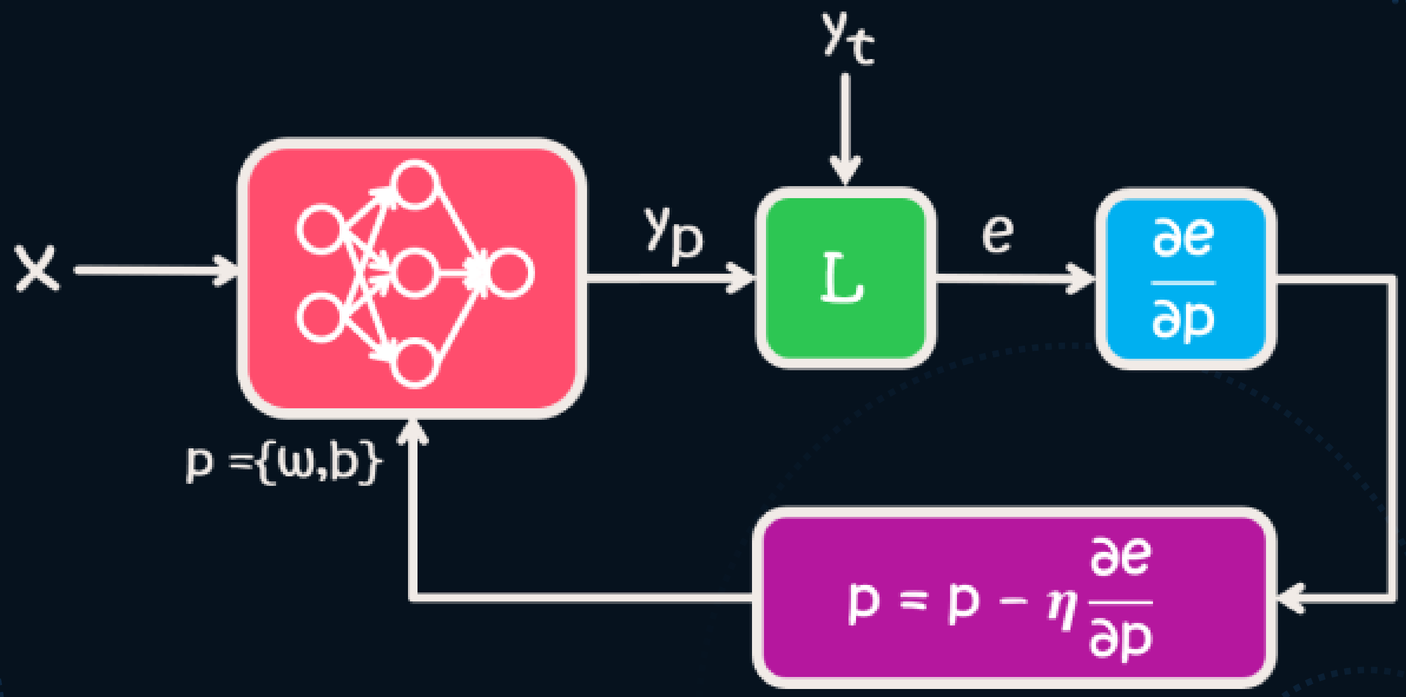
Ali Soleimani (ali.soleimani@mau.se)

# Key Points

- In this LAB, with work with Pytorch library (https://pytorch.org/)

- For implementation, to minimize errors during training the model, it is recommended to use Google Colab. https://colab.research.google.com/

- You are expected to do 2 projects: one involving a neural network (NN) with a CSV dataset, and the other involving a convolutional neural network (CNN) with an image dataset. Each project consists of 15 points (totaling 30 points). Completion of both projects is mandatory to pass this lab. For Project Number 1, testing the model on the test dataset is optional, and if you skip this step, you will receive 10 points. Regarding Project Number 2, calculating the final accuracy and saving the model are optional, and if you skip them, you will receive 10 credits. (Keep in mind that to pass all labs, you generally need 100 points.)


$y_t$
$x$
$y_p$
L
$e$
$\frac{\partial e}{\partial p}$
$p = \{\omega, b\}$
$p = p - \eta \frac{\partial e}{\partial p}$

# First Implementation:

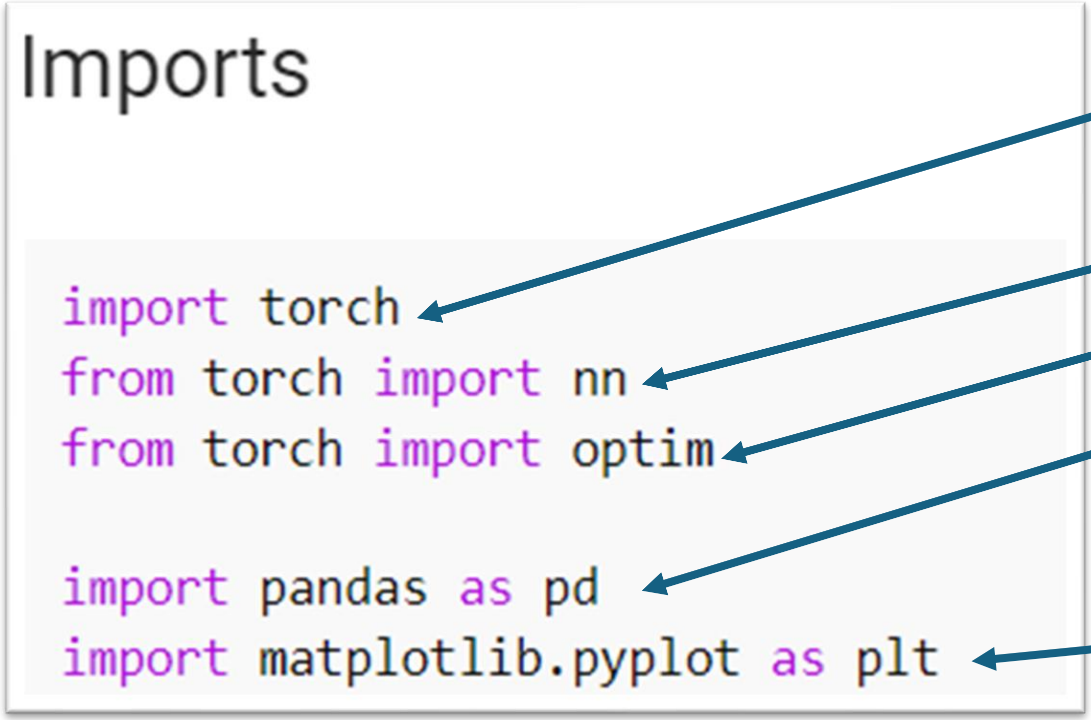
# Training a Neural Network

# Training a Neural Network

- In this implementation, we read a CSV dataset via Google Drive, then build and train a neural network model , and finally test it on test dataset and calculate loss values.
- (The dataset has been uploaded on Canvas, but can use your custom dataset)

MALMÖ UNIVERSITY

# Importing important libraries



## Imports

```
import torch
from torch import nn
from torch import optim

import pandas as pd
import matplotlib.pyplot as plt
```

Pytorch library

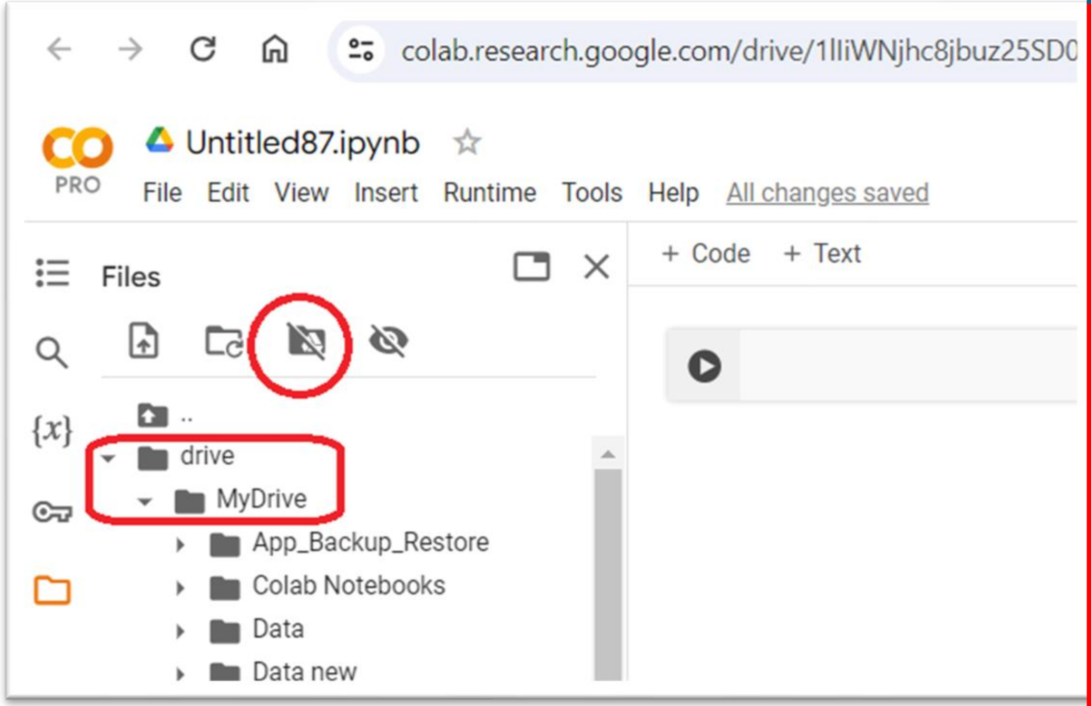
Neural Network

Optimization

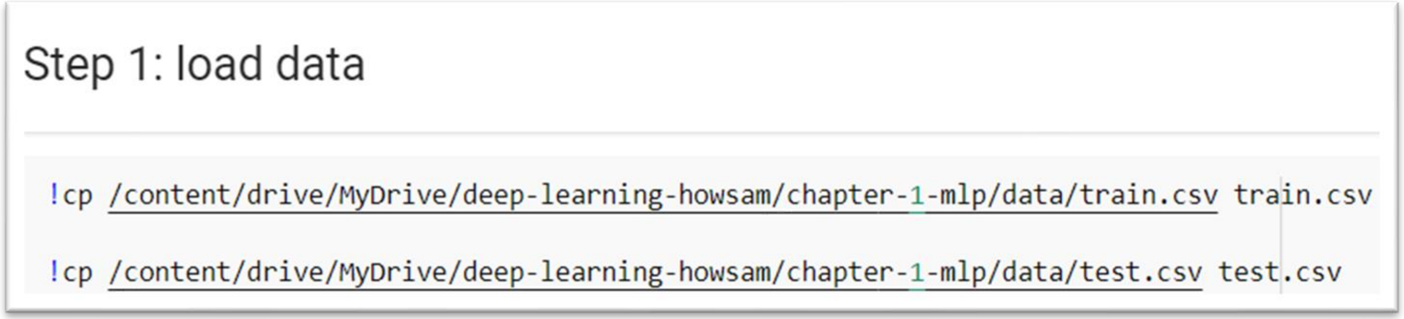
Reading csv files

Plotting

# Loading dataset



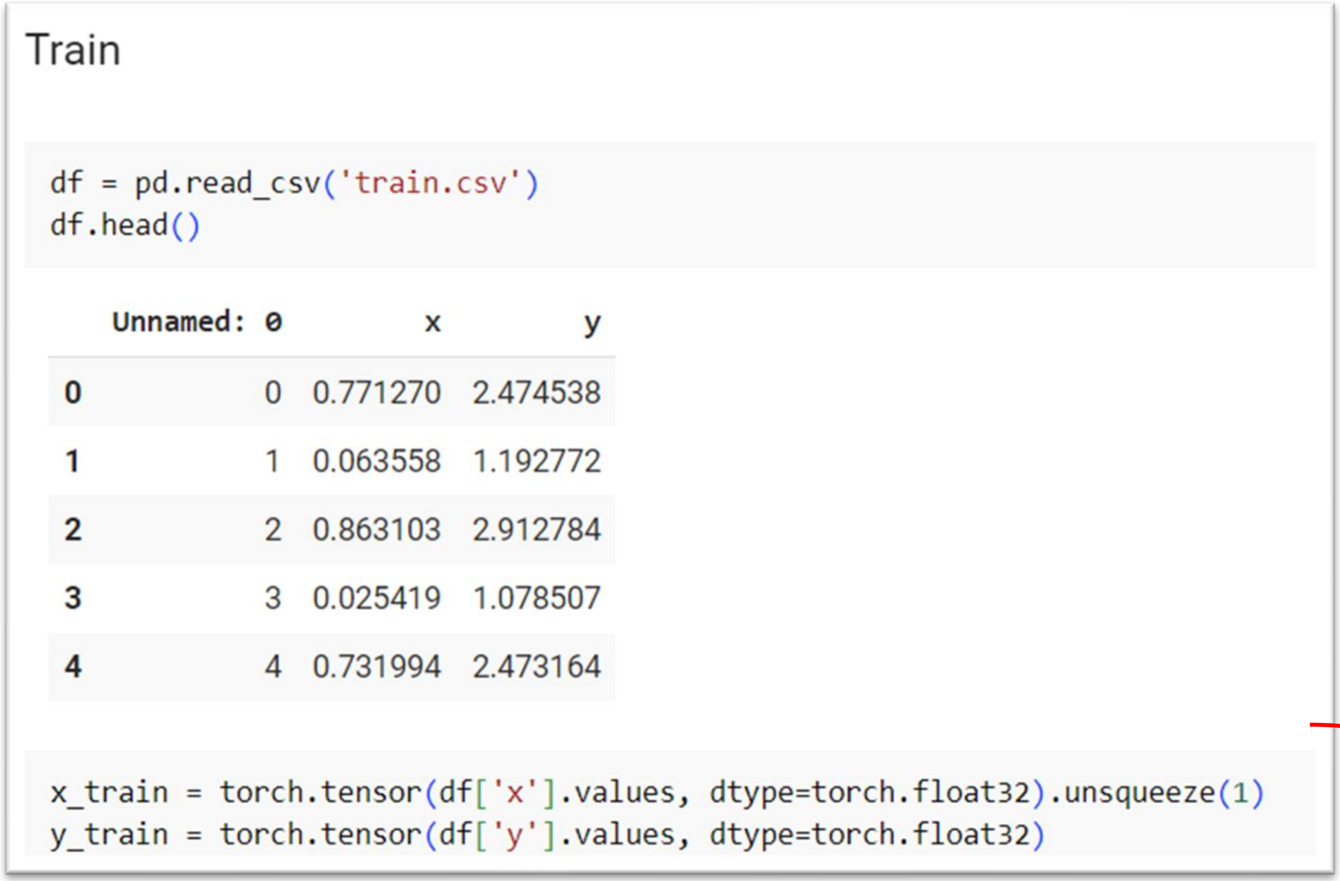
Mounting google drive



## Step 1: load data

```
!cp /content/drive/MyDrive/deep-learning-howsam/chapter-1-mlp/data/train.csv train.csv

!cp /content/drive/MyDrive/deep-learning-howsam/chapter-1-mlp/data/test.csv test.csv
```

# Loading dataset (train dataset)



Train

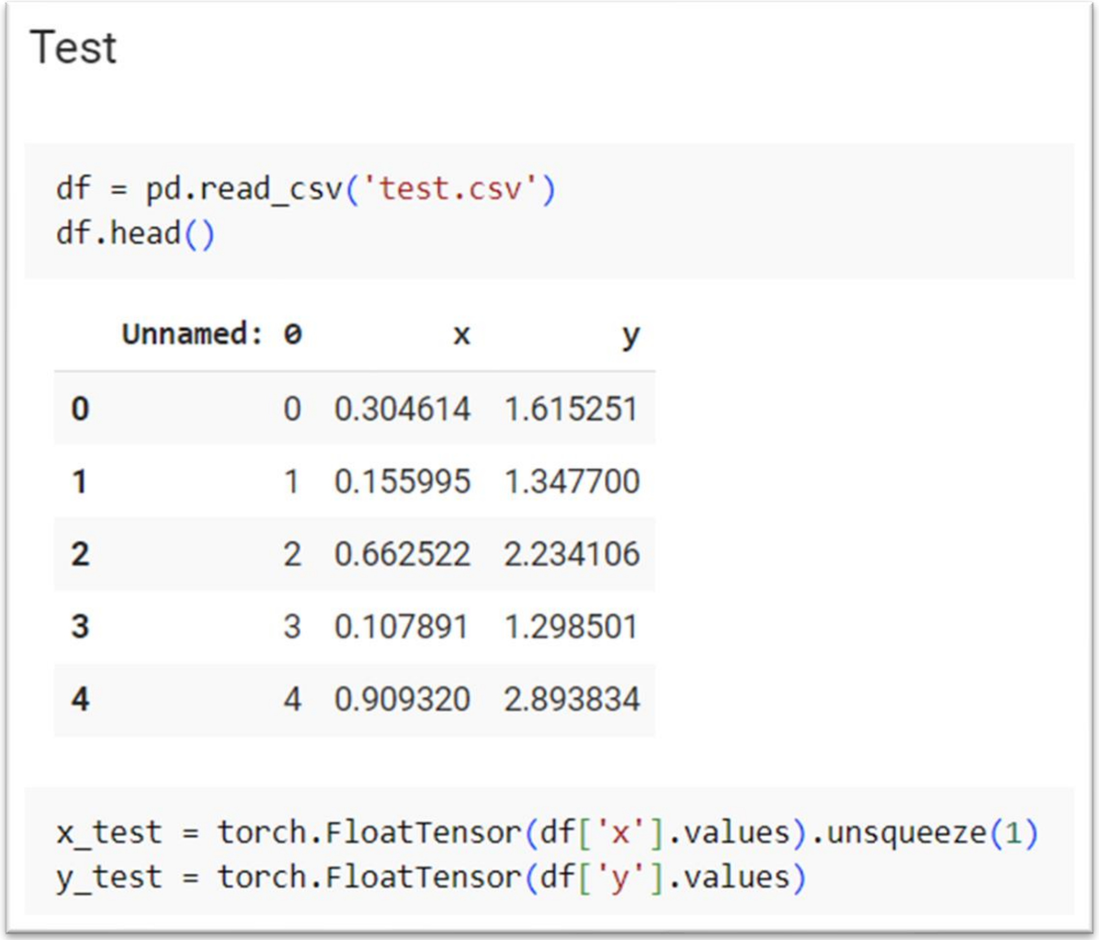
```
df = pd.read_csv('train.csv')
df.head()
```

| | Unnamed: 0 | x | y |
|---|---|---|---|
| **0** | 0 | 0.771270 | 2.474538 |
| **1** | 1 | 0.063558 | 1.192772 |
| **2** | 2 | 0.863103 | 2.912784 |
| **3** | 3 | 0.025419 | 1.078507 |
| **4** | 4 | 0.731994 | 2.473164 |

```
x_train = torch.tensor(df['x'].values, dtype=torch.float32).unsqueeze(1)
y_train = torch.tensor(df['y'].values, dtype=torch.float32)
```

Converting dataset to tensors

# Loading dataset (test dataset)



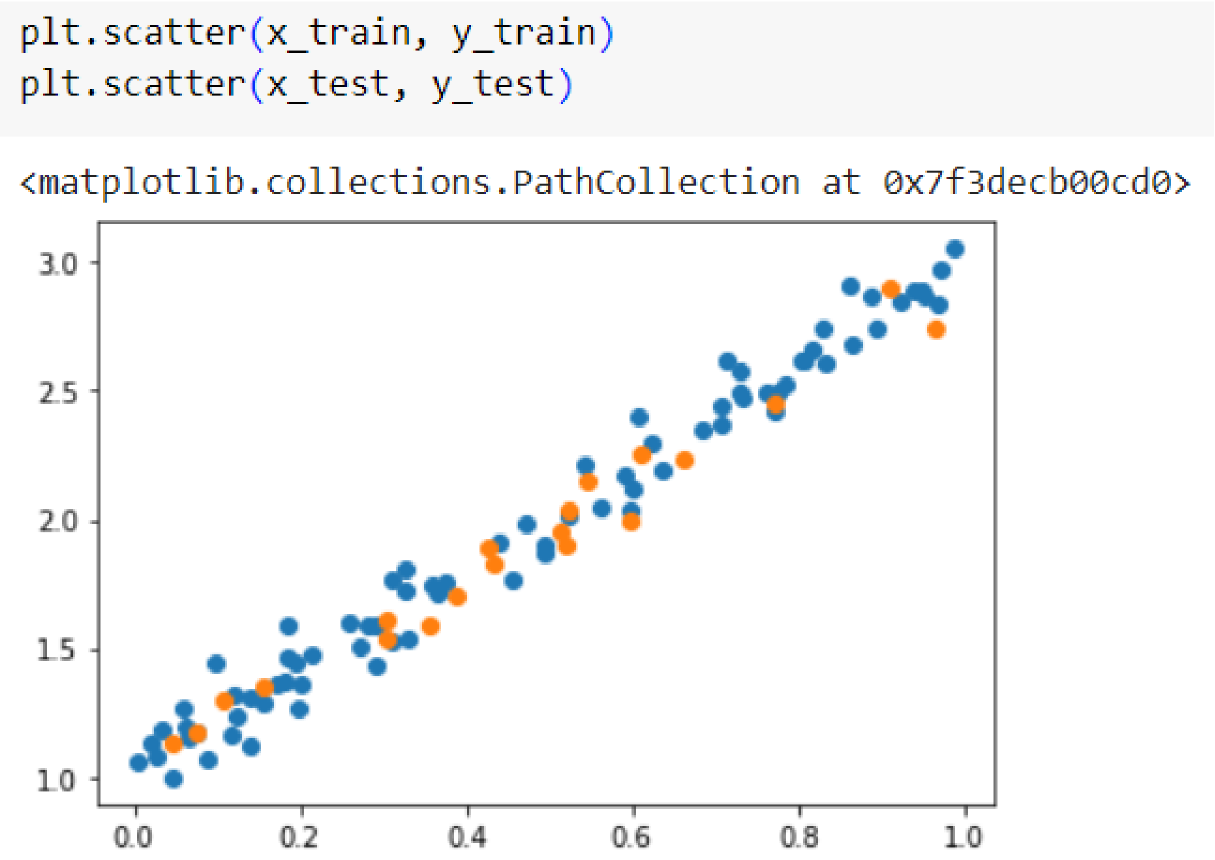
Test

```
df = pd.read_csv('test.csv')
df.head()
```

|   | Unnamed: 0 | x | y |
|---|---|---|---|
| **0** | 0 | 0.304614 | 1.615251 |
| **1** | 1 | 0.155995 | 1.347700 |
| **2** | 2 | 0.662522 | 2.234106 |
| **3** | 3 | 0.107891 | 1.298501 |
| **4** | 4 | 0.909320 | 2.893834 |

```
x_test = torch.FloatTensor(df['x'].values).unsqueeze(1)
y_test = torch.FloatTensor(df['y'].values)
```

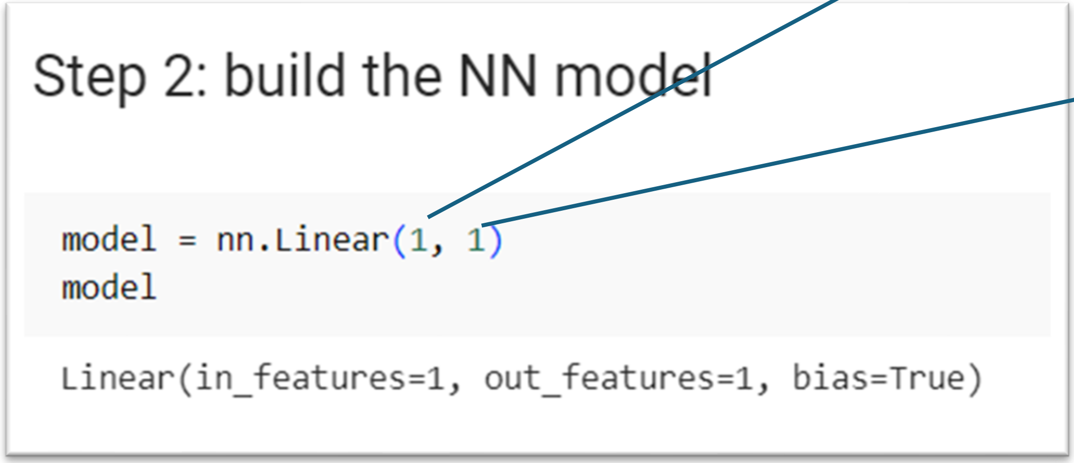
# Plotting dataset



```
plt.scatter(x_train, y_train)
plt.scatter(x_test, y_test)
```

```
<matplotlib.collections.PathCollection at 0x7f3decb00cd0>
```

# Building the NN model



## Step 2: build the NN model

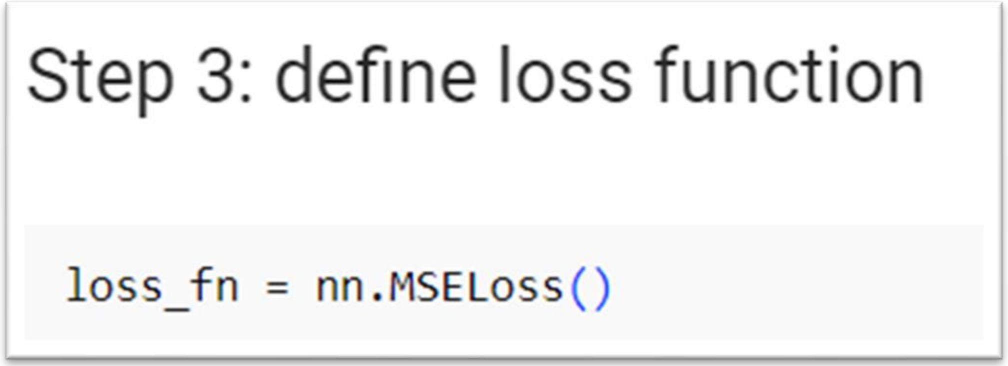
```
model = nn.Linear(1, 1)
model
```

```
Linear(in_features=1, out_features=1, bias=True)
```

In this dataset we have only one feature

Output Neuron

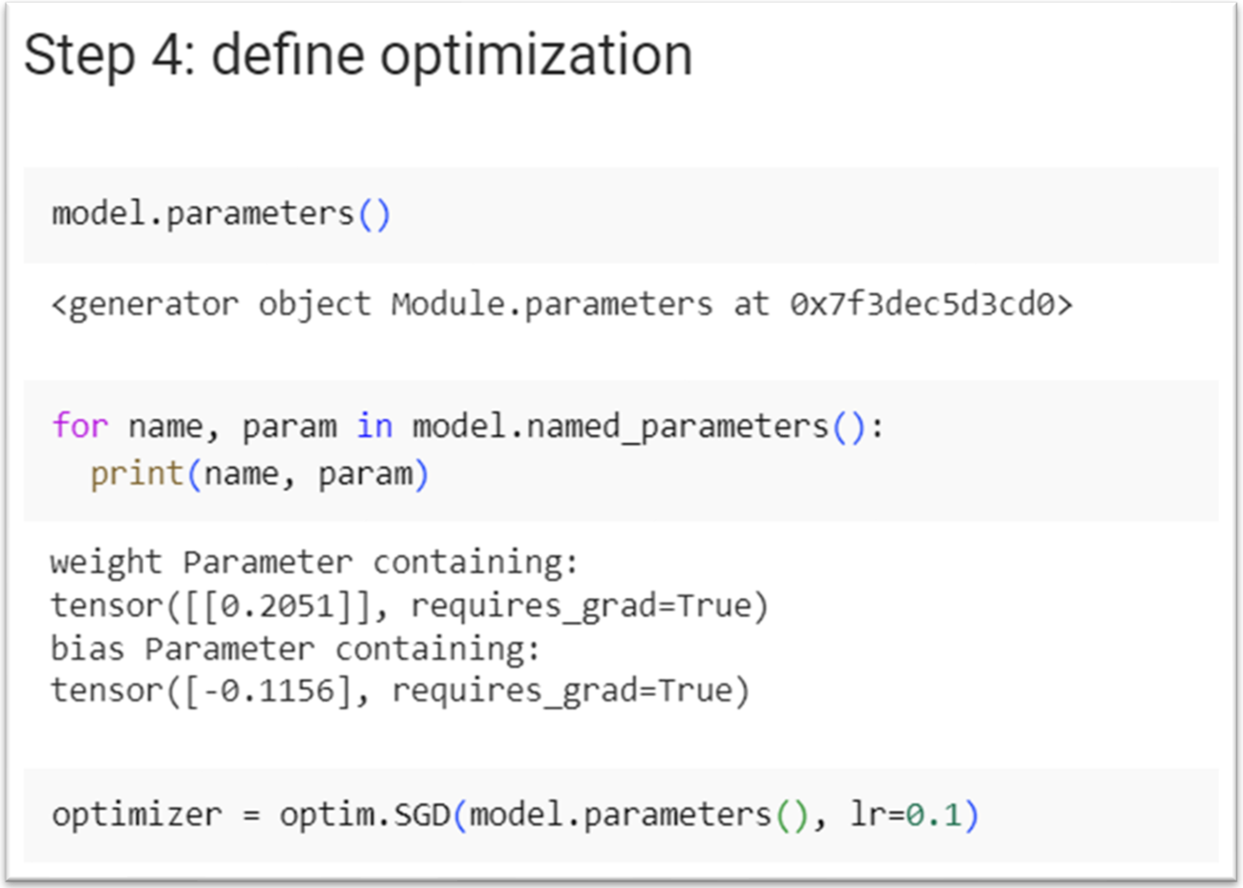
NN model with one inpute feauture and one output neuron



## Step 3: define loss function

```
loss_fn = nn.MSELoss()
```

# Optimizer



## Step 4: define optimization

```
model.parameters()
```

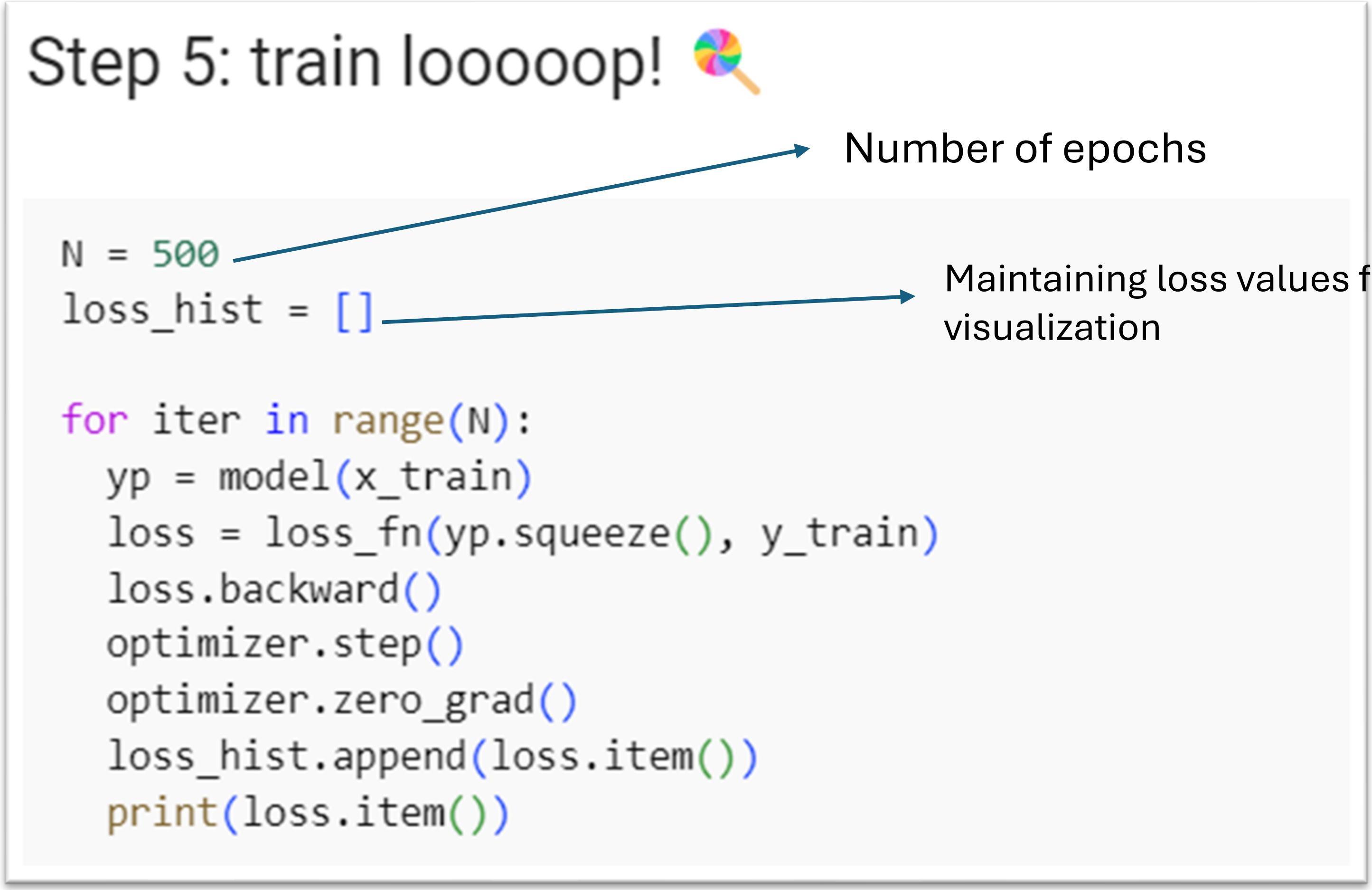
```
<generator object Module.parameters at 0x7f3dec5d3cd0>
```

```
for name, param in model.named_parameters():
  print(name, param)
```

```
weight Parameter containing:
tensor([[0.2051]], requires_grad=True)
bias Parameter containing:
tensor([-0.1156], requires_grad=True)
```

```
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

# Training loop



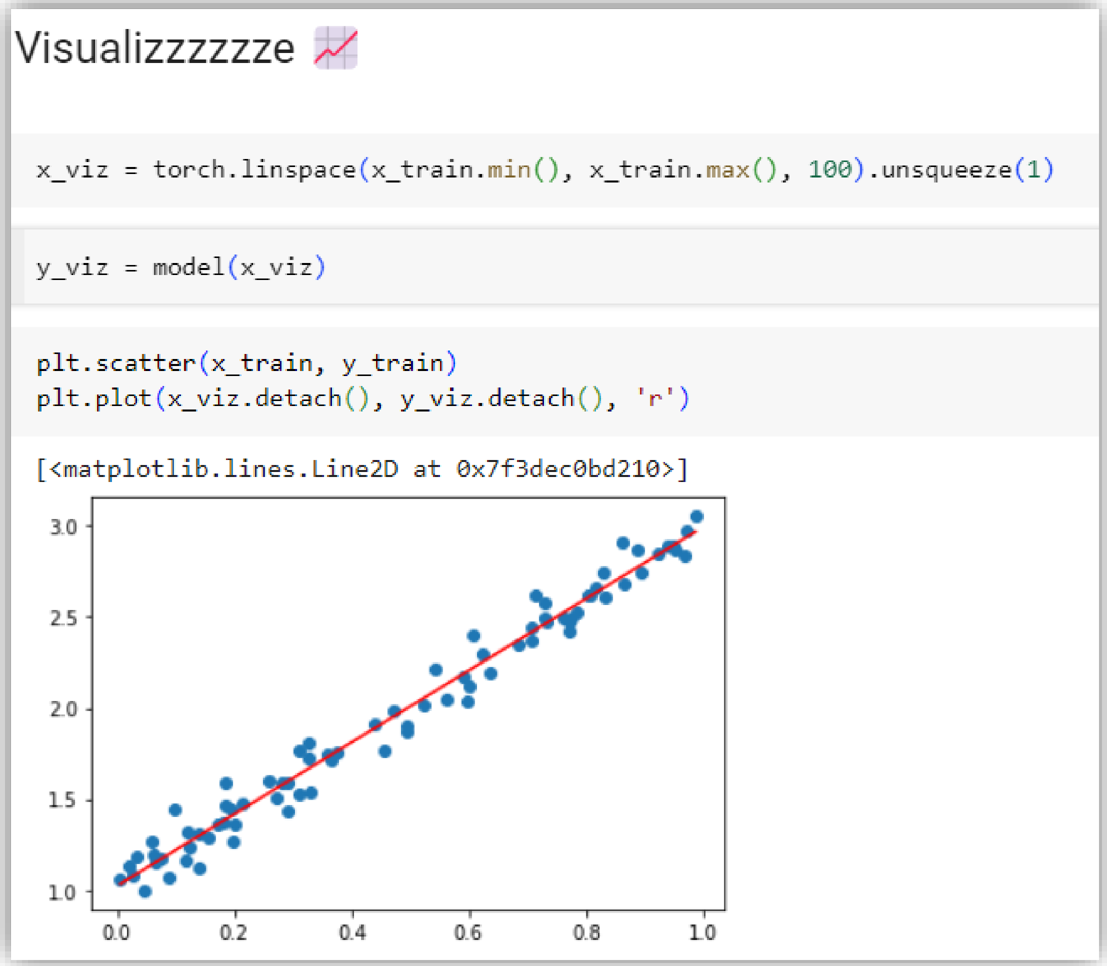
## Step 5: train loooooop! 🍭

```python
N = 500
loss_hist = []

for iter in range(N):
  yp = model(x_train)
  loss = loss_fn(yp.squeeze(), y_train)
  loss.backward()
  optimizer.step()
  optimizer.zero_grad()
  loss_hist.append(loss.item())
  print(loss.item())
```

Number of epochs

Maintaining loss values for visualization

# Testing some random samples on model



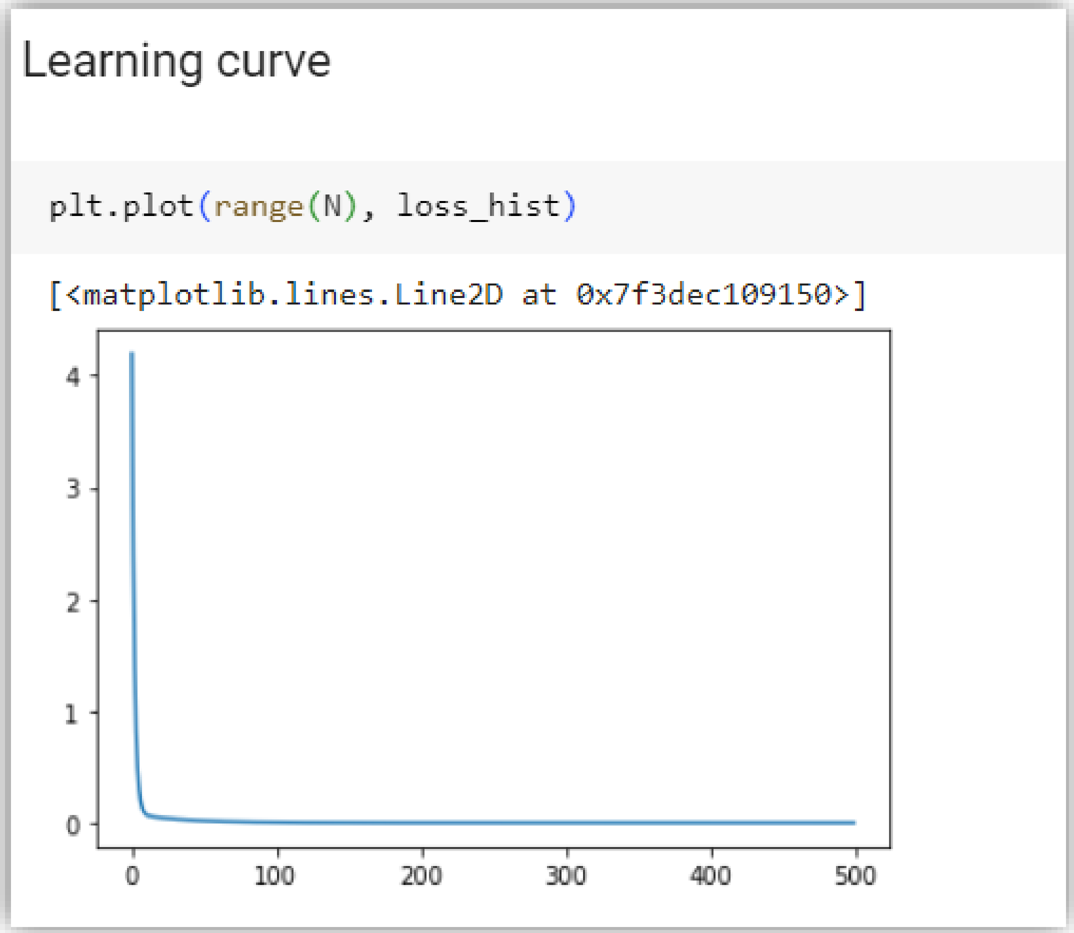
## Visualizzzzzze 📈

```
x_viz = torch.linspace(x_train.min(), x_train.max(), 100).unsqueeze(1)
```

```
y_viz = model(x_viz)
```

```
plt.scatter(x_train, y_train)
plt.plot(x_viz.detach(), y_viz.detach(), 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f3dec0bd210>]
```

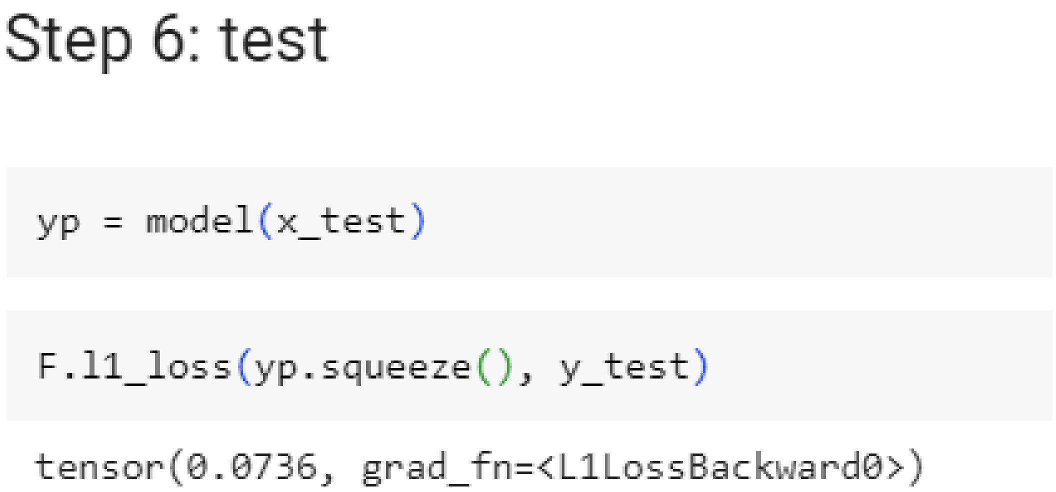


## Learning curve

```
plt.plot(range(N), loss_hist)
```

```
[<matplotlib.lines.Line2D at 0x7f3dec109150>]
```

# Testing model on test dataset



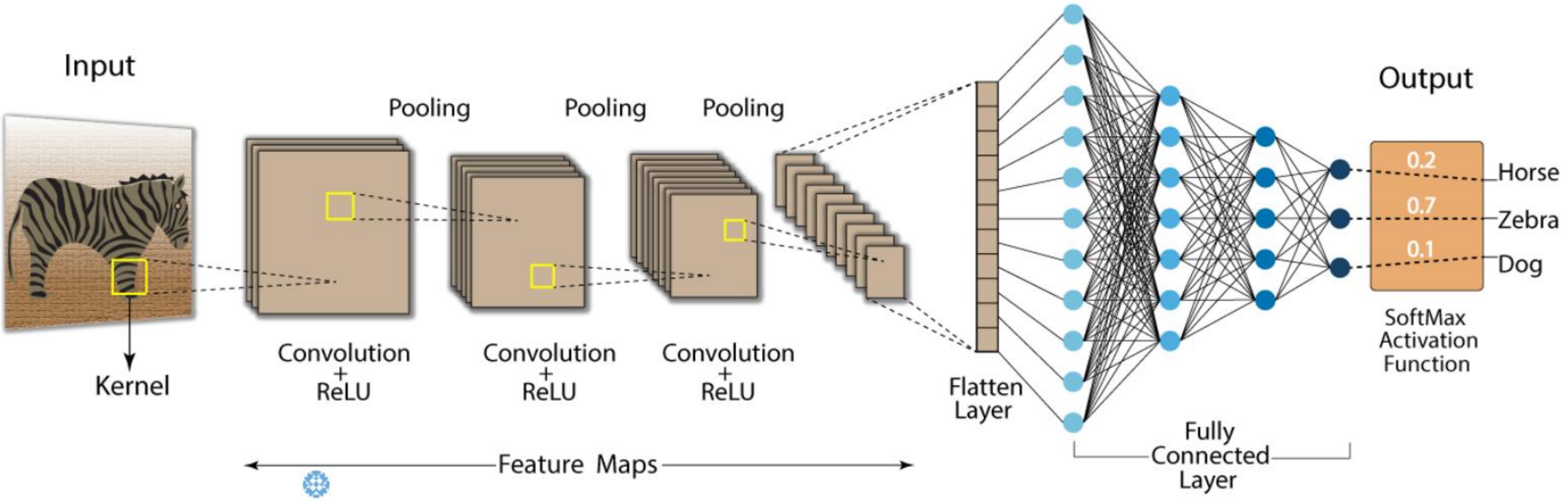
## Step 6: test

```
yp = model(x_test)
```

```
F.l1_loss(yp.squeeze(), y_test)
```

```
tensor(0.0736, grad_fn=<L1LossBackward0>)
```

# Second Implementation:
# Training a convolutional neural network (CNN)

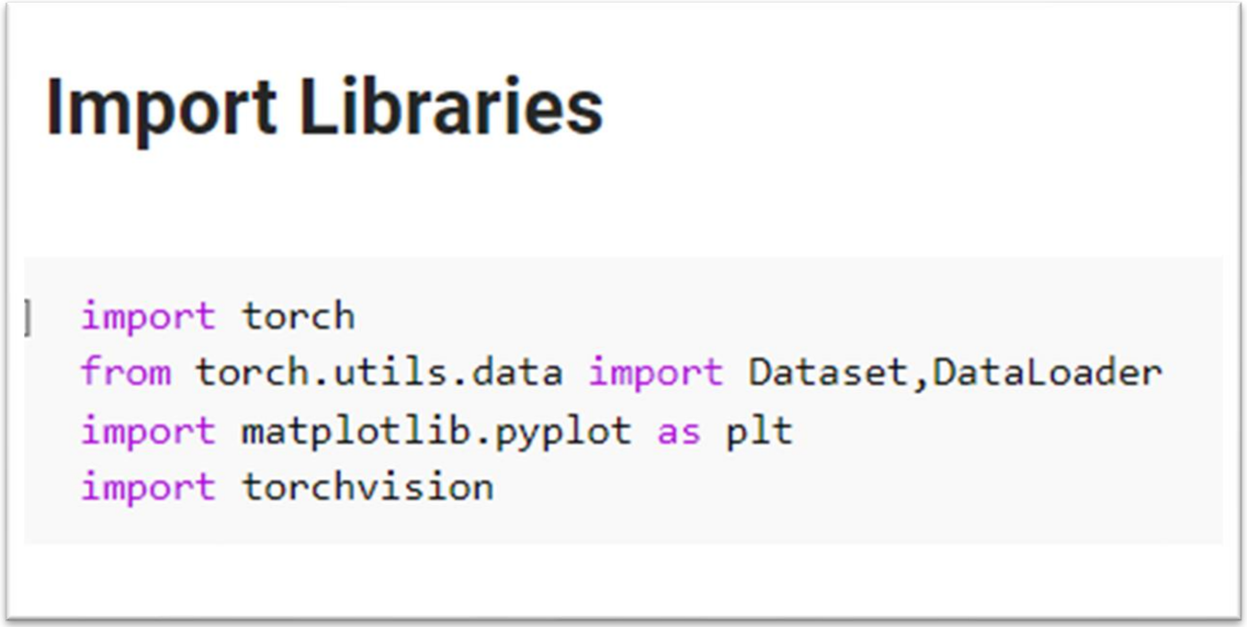


# Notes:

- In this image classification implementation, we read Cifar10 dataset with pytorch, then build and train a CNN network model , and finally calculate the accuracy on test dataset.

- If you want to use other datasets, check this link:
- https://pytorch.org/vision/stable/datasets.html#image-classification

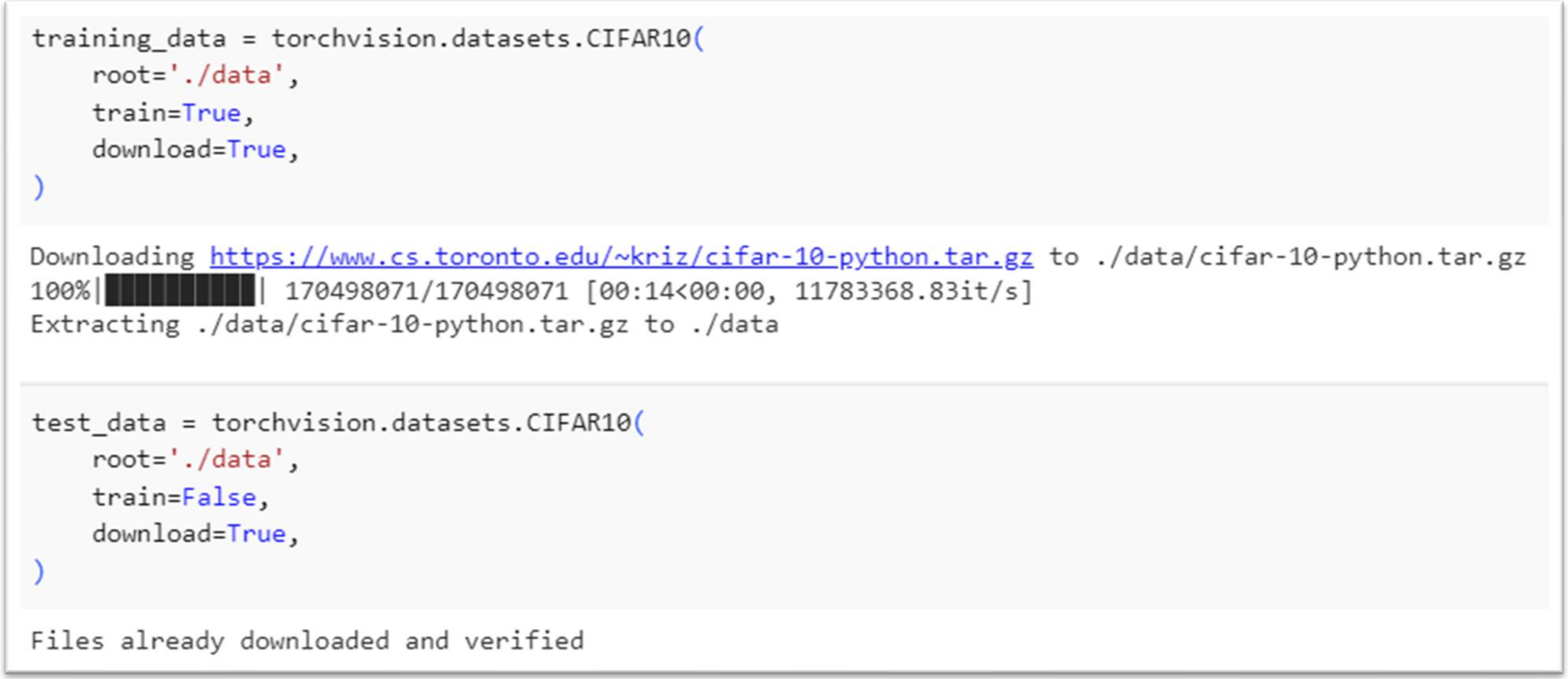
# Importing libraries



## Import Libraries

```
import torch
from torch.utils.data import Dataset,DataLoader
import matplotlib.pyplot as plt
import torchvision
```

# Downloaing dataset



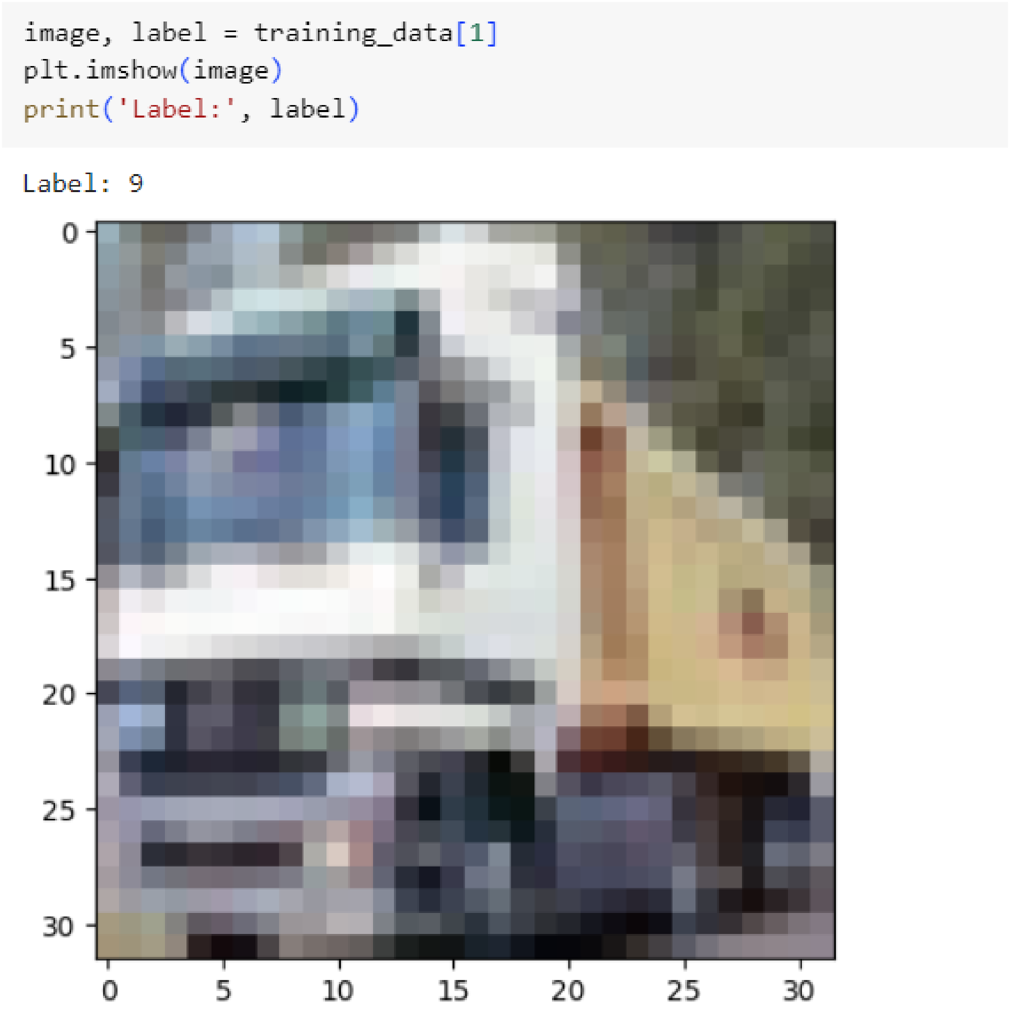
```
training_data = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:14<00:00, 11783368.83it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
```

```
test_data = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
)
```

```
Files already downloaded and verified
```

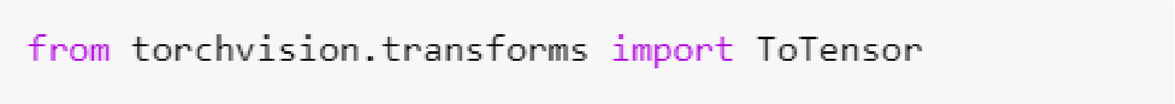
# Checking one of the images



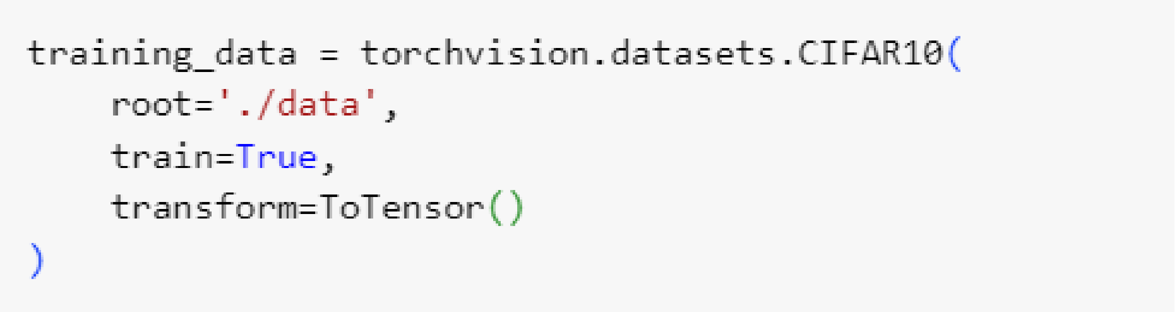
```
image, label = training_data[1]
plt.imshow(image)
print('Label:', label)
```

```
Label: 9
```

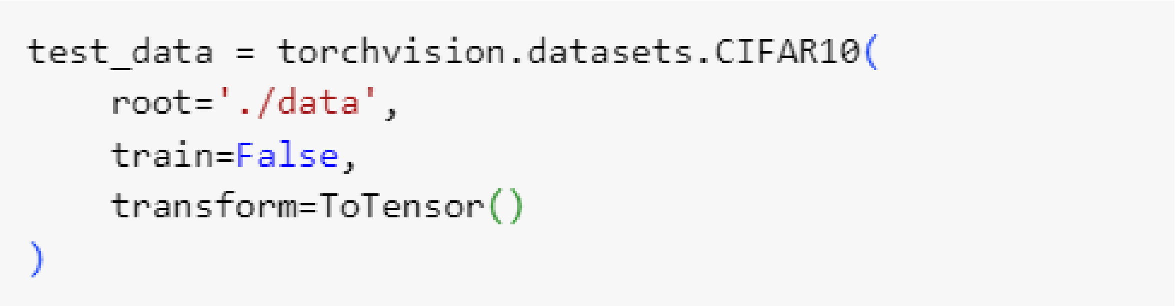
# Converting images to tensors



```
from torchvision.transforms import ToTensor
```

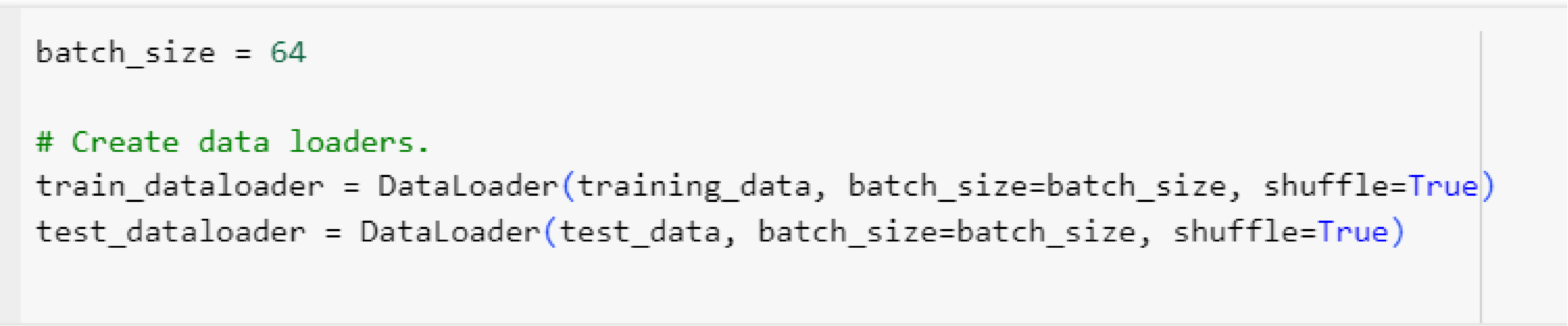


```
training_data = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    transform=ToTensor()
)
```



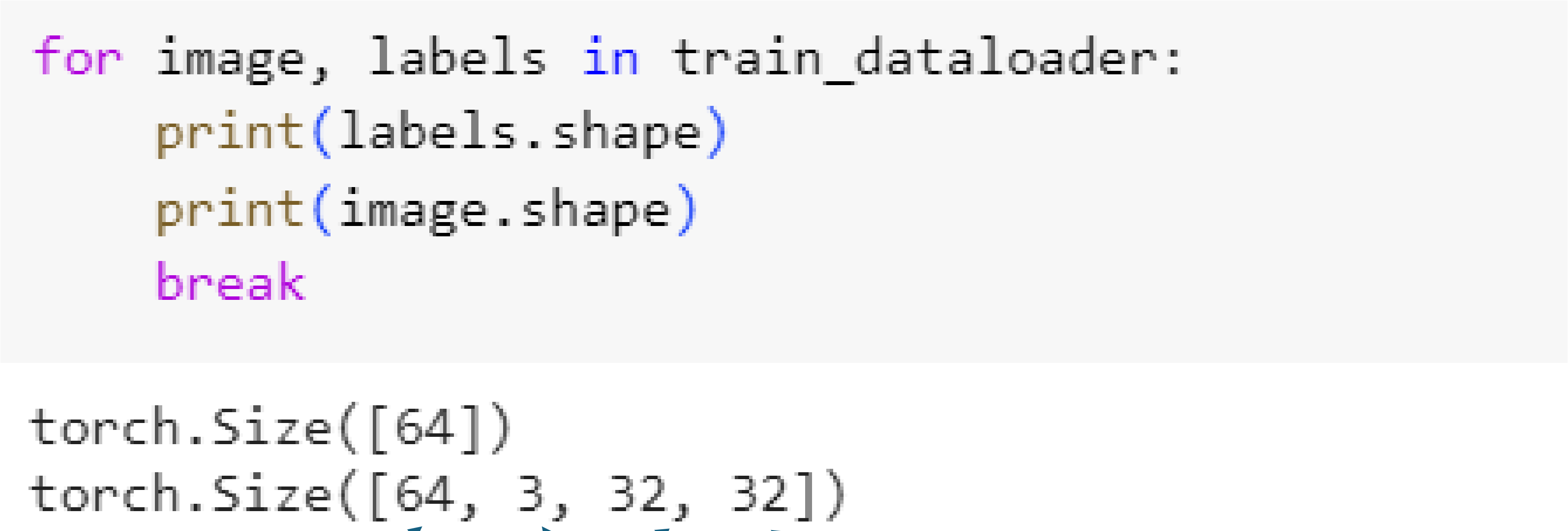
```
test_data = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    transform=ToTensor()
)
```

# Batch size and DataLoader



```
batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

# Checking one batch of dataloader



```
for image, labels in train_dataloader:
    print(labels.shape)
    print(image.shape)
    break
```

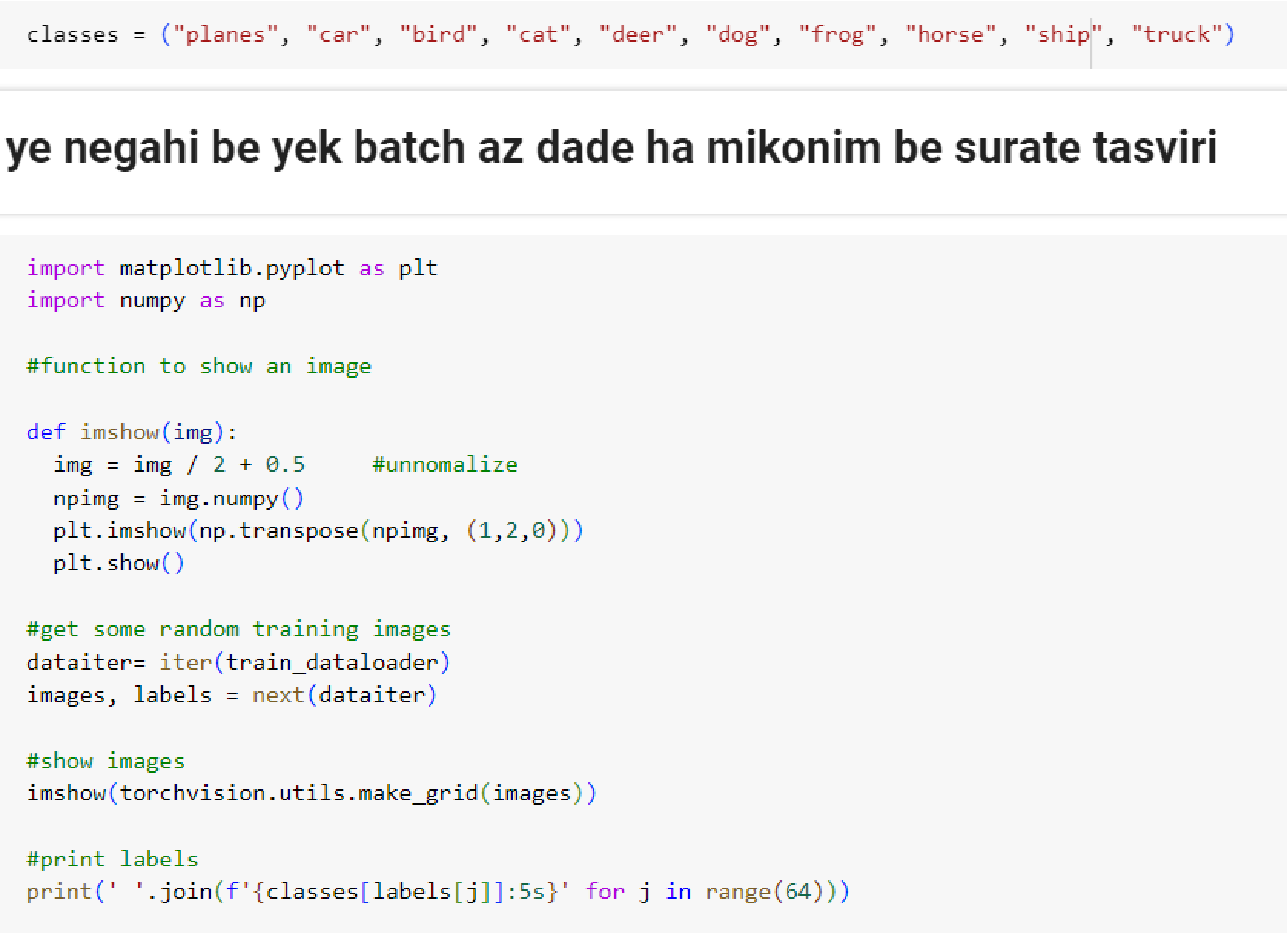
```
torch.Size([64])
torch.Size([64, 3, 32, 32])
```

Batch size

3 channels

image size

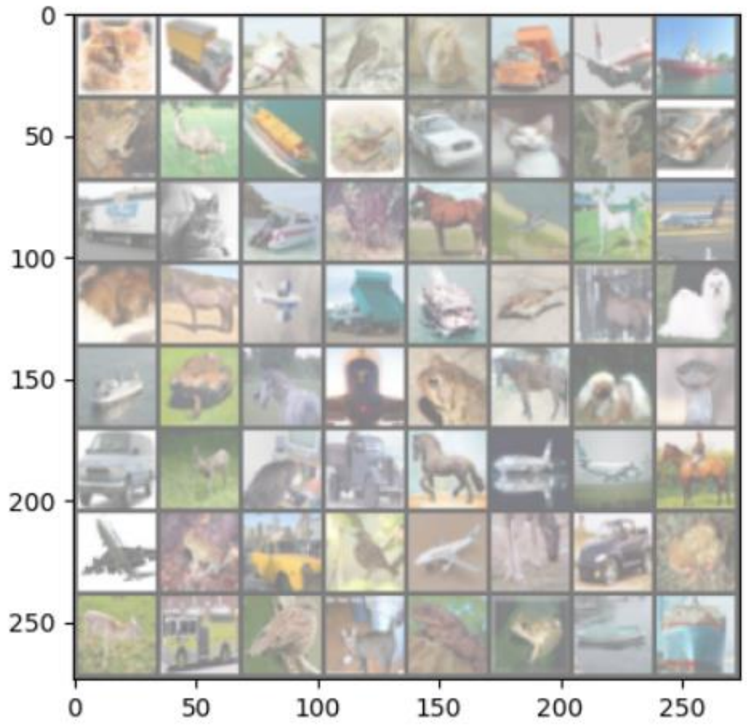
# Visualizing one batch of images



```
classes = ("planes", "car", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck")
```

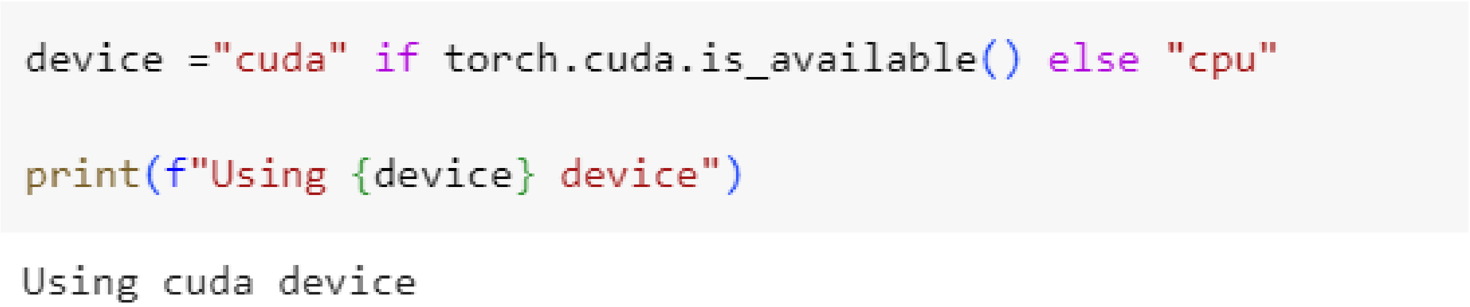
ye negahi be yek batch az dade ha mikonim be surate tasviri

```
import matplotlib.pyplot as plt
import numpy as np

#function to show an image

def imshow(img):
  img = img / 2 + 0.5     #unnomalize
  npimg = img.numpy()
  plt.imshow(np.transpose(npimg, (1,2,0)))
  plt.show()

#get some random training images
dataiter= iter(train_dataloader)
images, labels = next(dataiter)

#show images
imshow(torchvision.utils.make_grid(images))

#print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(64)))
```



# Checking GPU

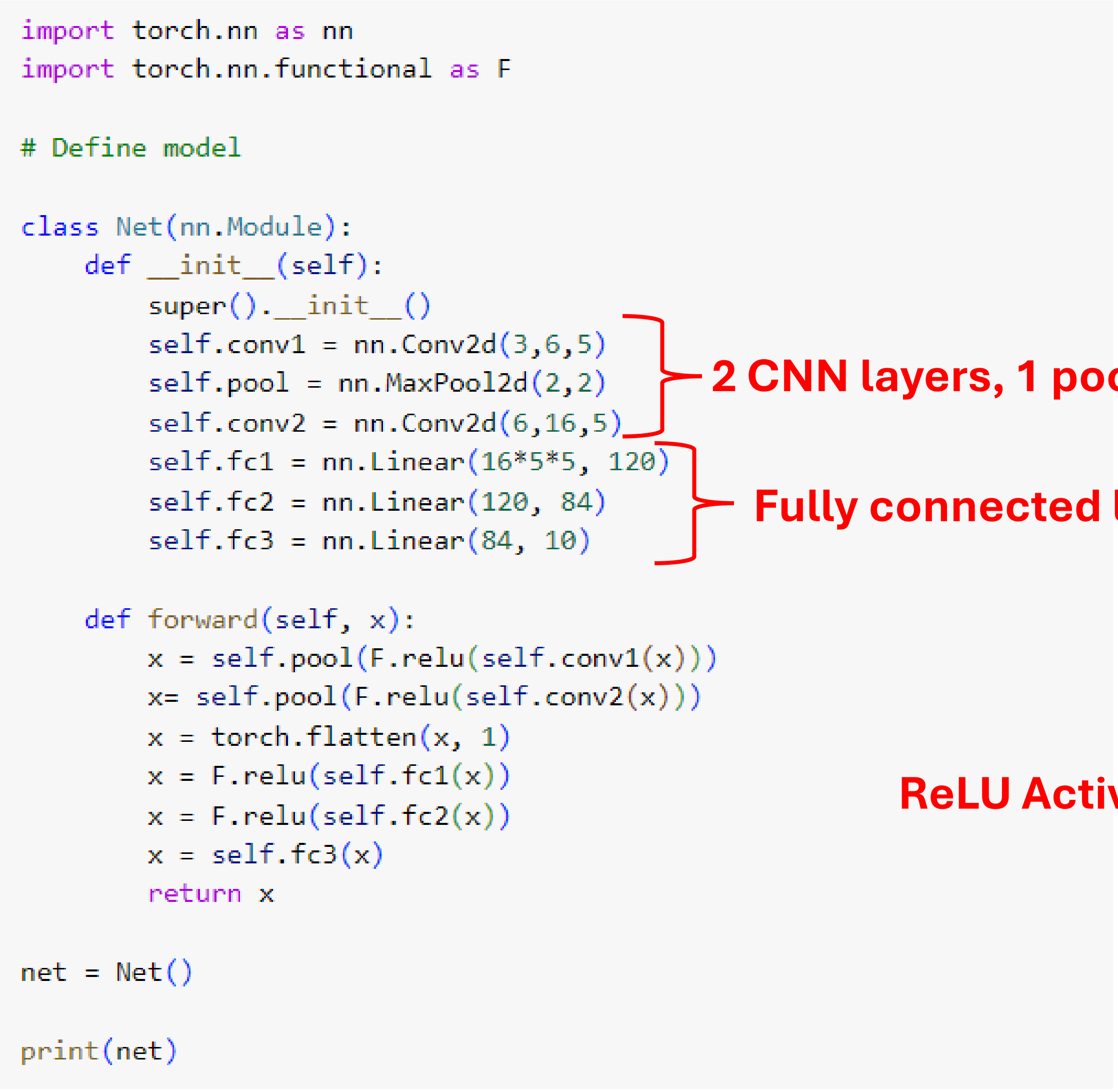
If you are using Google Colab, set the runtime to GPU in the runtime section to achieve faster training



```
device ="cuda" if torch.cuda.is_available() else "cpu"

print(f"Using {device} device")
```

```
Using cuda device
```

# Building a CNN model



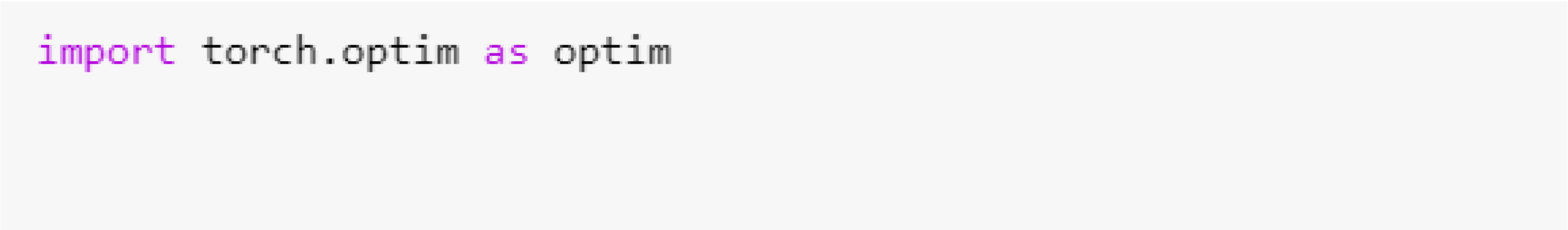
```
import torch.nn as nn
import torch.nn.functional as F

# Define model

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,6,5)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(6,16,5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x= self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

print(net)
```

2 CNN layers, 1 pooling layer

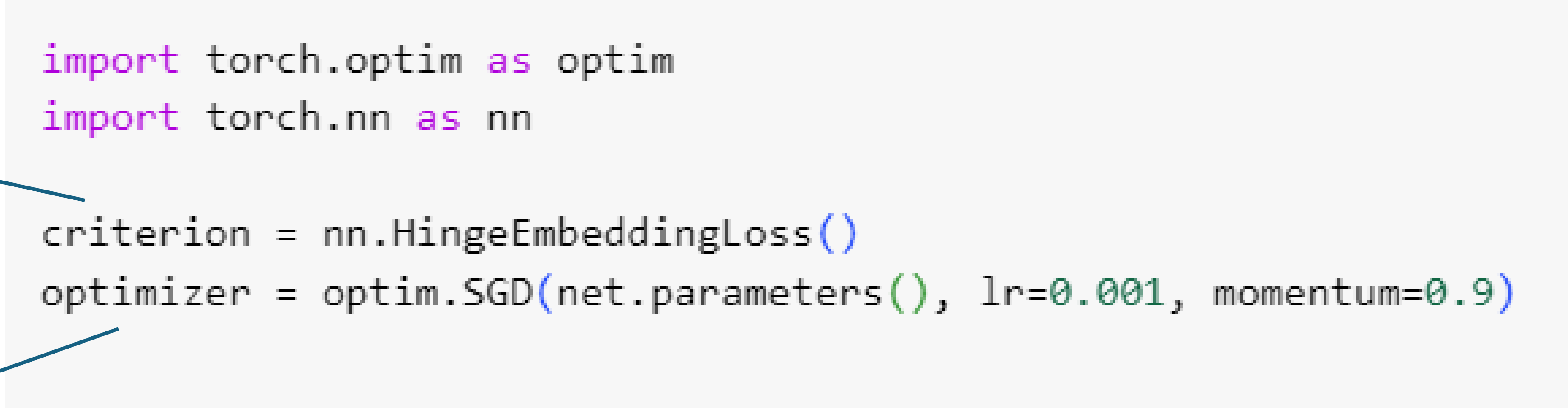
Fully connected layers

ReLU Activation Function

# Loss Function and Optimizer



```python
import torch.optim as optim
```

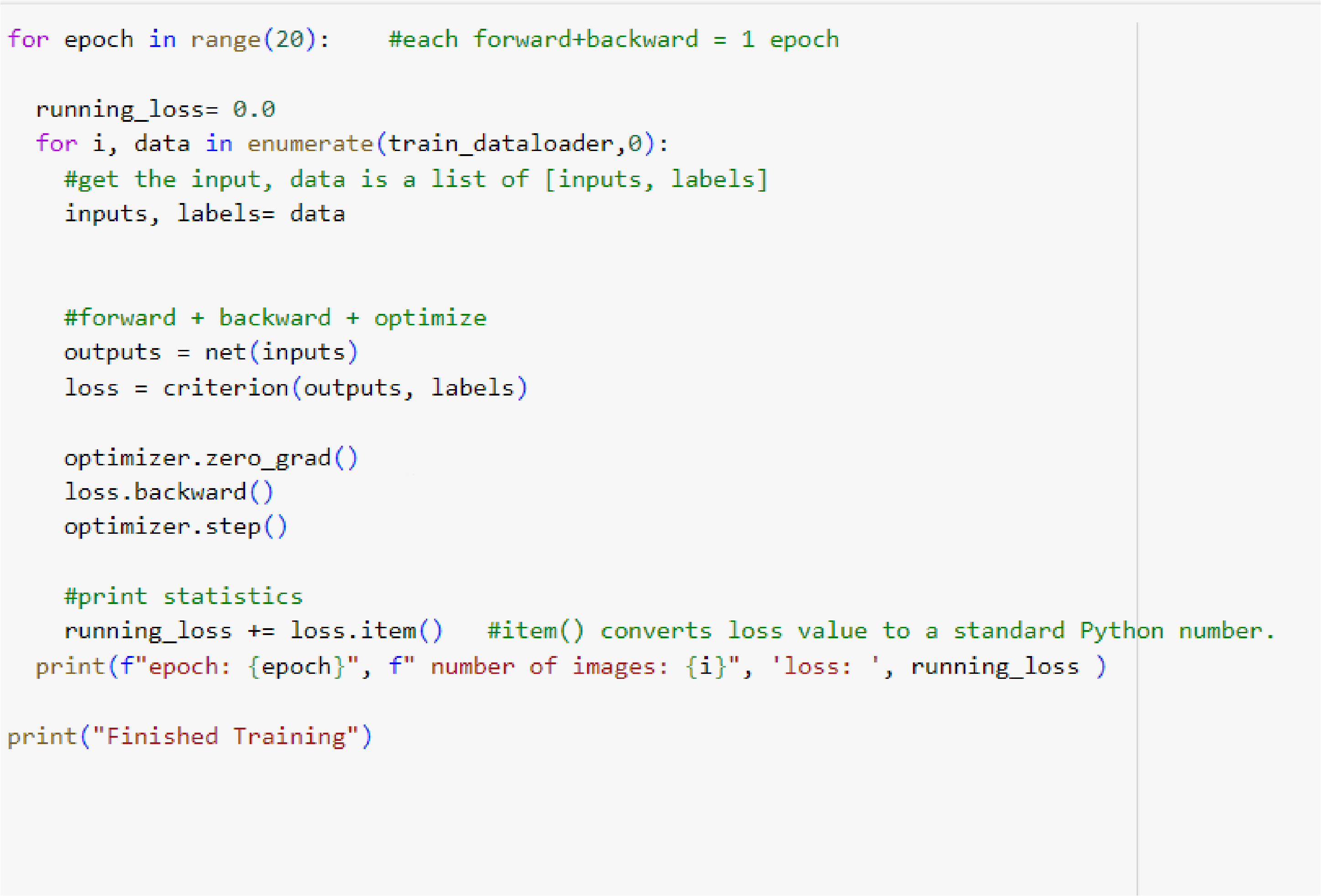


```python
import torch.optim as optim
import torch.nn as nn

criterion = nn.HingeEmbeddingLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Loss function

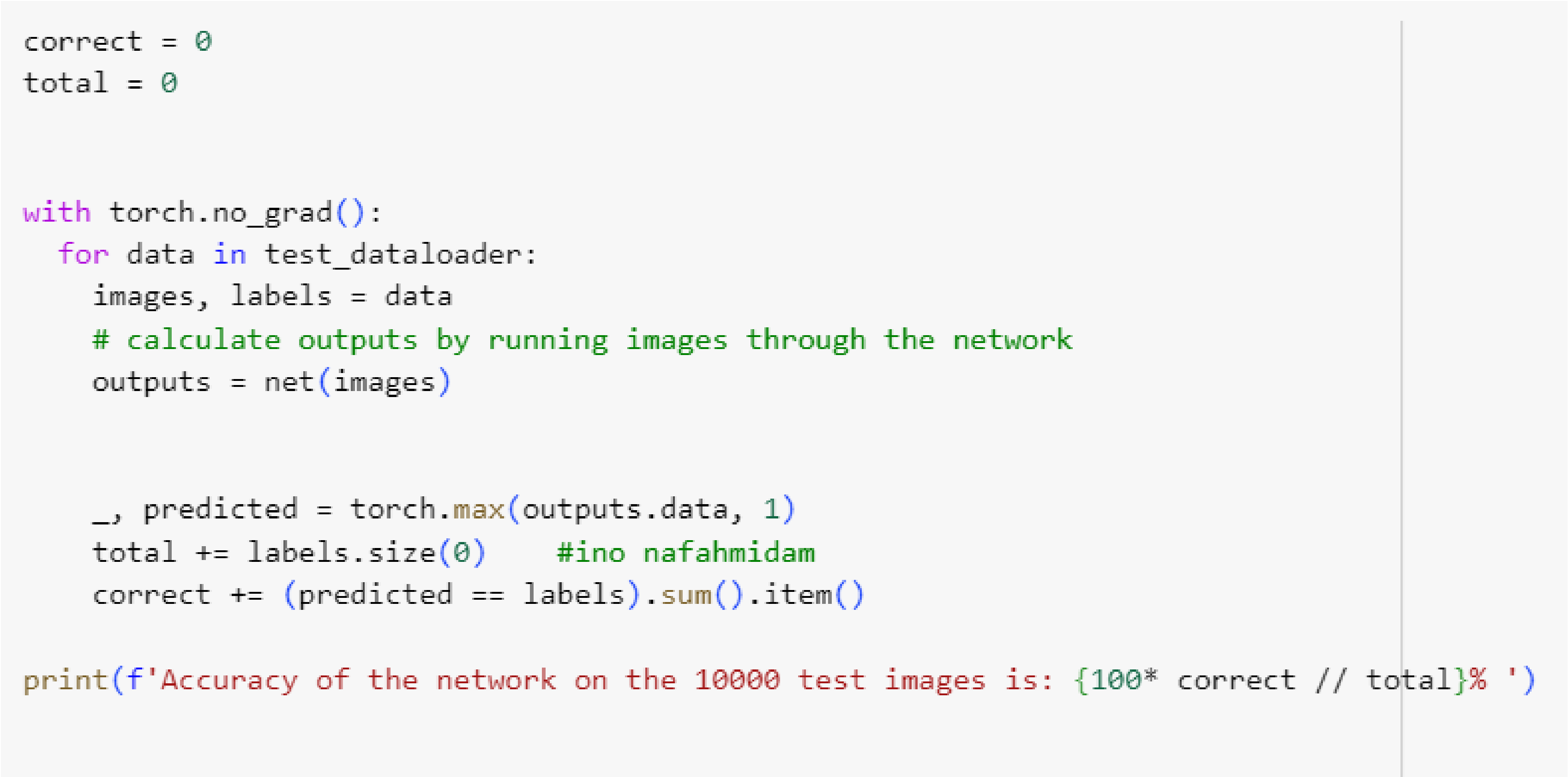
Optimizer

# Training the model



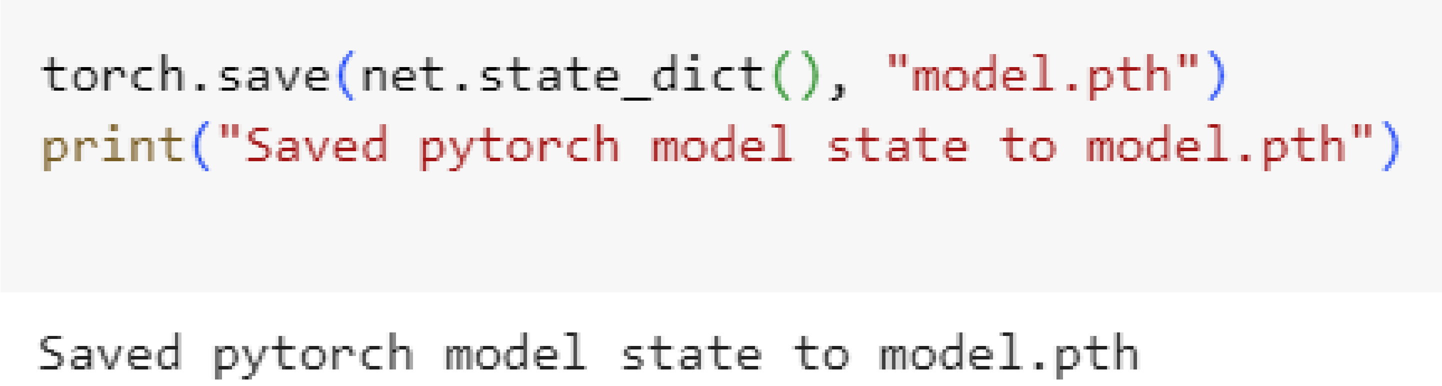
```
for epoch in range(20):    #each forward+backward = 1 epoch

  running_loss= 0.0
  for i, data in enumerate(train_dataloader,0):
    #get the input, data is a list of [inputs, labels]
    inputs, labels= data


    #forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    #print statistics
    running_loss += loss.item()   #item() converts loss value to a standard Python number.
  print(f"epoch: {epoch}", f" number of images: {i}", 'loss: ', running_loss )

print("Finished Training")
```

# Accuracy of the model



```
correct = 0
total = 0


with torch.no_grad():
  for data in test_dataloader:
    images, labels = data
    # calculate outputs by running images through the network
    outputs = net(images)


    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)    #ino nafahmidam
    correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images is: {100* correct // total}% ')
```

# Saving the model



```
torch.save(net.state_dict(), "model.pth")
print("Saved pytorch model state to model.pth")
```

```
Saved pytorch model state to model.pth
```

- If you need more guidance and additional training samples, please check the following link:
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- https://pytorch.org/vision/stable/index.html