In [1]:
```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
from torchvision.datasets import CIFAR100
import torchvision.transforms as transforms
from torchvision.utils import make_grid
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split,ConcatDataset

import model
import data
```

In [2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [3]:
```
net = model.resnet34().to(device)
```

In [ ]:
```
trainDataLoader, testDataLoader =  data.loadData(128)
```

In [5]:
```
loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.0001)
```

In [6]:
```
def evaluate(model,dataloader):
  acc = 0.0
  rights = 0
  wrongs = 0
  for i, test_examples in enumerate(dataloader, 0):
    #predicting using the nets
    inputs, labels = test_examples
    predicted_outputs = model(inputs.float().cuda())
    #Selecting the label which has the largest outputs
    outputs = torch.argmax(predicted_outputs, 1)

    #Counting successfully and unsuccessfully predicted cases
    for j, n in enumerate(outputs):
      if n == labels[j]:
        rights += 1
      else:
        wrongs += 1
  #calculate accuracy with the cases we recorded
  acc = rights/(rights+wrongs)
  #return the accuracy
  return acc
```

In [7]:
```
def train(model,train,test,loss_fn,optimizer,watch_iter):
    total_iter = 0
    loss = 0.0

    while total_iter < 10000:
        for batch in train:
            total_iter += 1
            train_inputs, train_labels = batch
            train_inputs, train_labels = train_inputs.to(device), train_labels.to(devic
            train_outputs = model(train_inputs)
            l = loss_fn(train_outputs, train_labels)
```

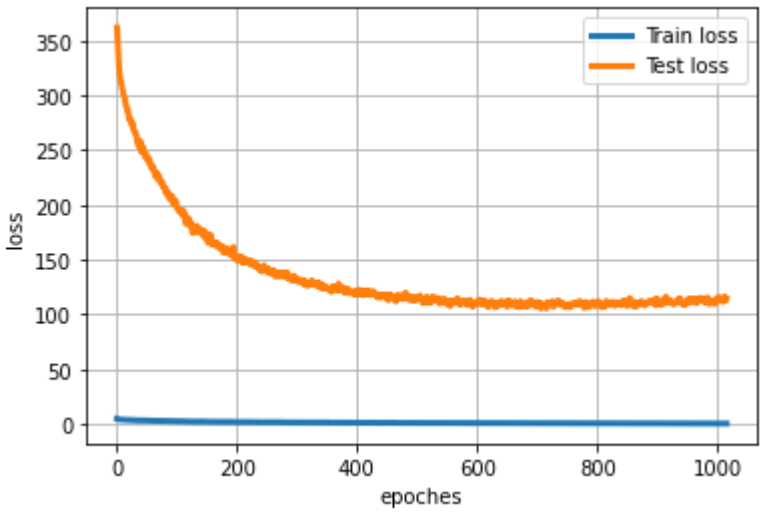
```
            loss += l.item()
            optimizer.zero_grad()
            l.backward()
            optimizer.step()

            if total_iter % watch_iter == 0:
                train_loss = loss / watch_iter
                train_loss_his.append(train_loss)
                loss = 0.0
                for batch in test:
                    test_inputs, test_labels = batch
                    test_inputs, test_labels = test_inputs.to(device), test_labels.to(d
                    test_outputs = model(test_inputs)
                    l = loss_fn(test_outputs, test_labels)
                    loss += l.item()
                test_loss_his.append(loss)
                txt = f'iter: {total_iter: 6d}, train loss: {train_loss}, test_loss: {l
                print(txt)
                print('accuracy: ' + str(evaluate(model,test)*100) + '%')
                loss = 0.0
    return
```

In [ ]:
```
train_loss_his = []
test_loss_his = []
train(net,trainDataLoader,testDataLoader,loss,optimizer,10)
```

In [11]:
```
plt.plot(range(len(train_loss_his)),train_loss_his,'-',linewidth=3,label='Train loss')
plt.plot(range(len(train_loss_his)),test_loss_his,'-',linewidth=3,label='Test loss')
plt.xlabel('epoches')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x1460fc111ac0>



In [13]:
```
print('accuracy: ' + str(evaluate(net,testDataLoader)*100) + '%')
```

```
accuracy: 63.94%
```