Pytorch Giriş

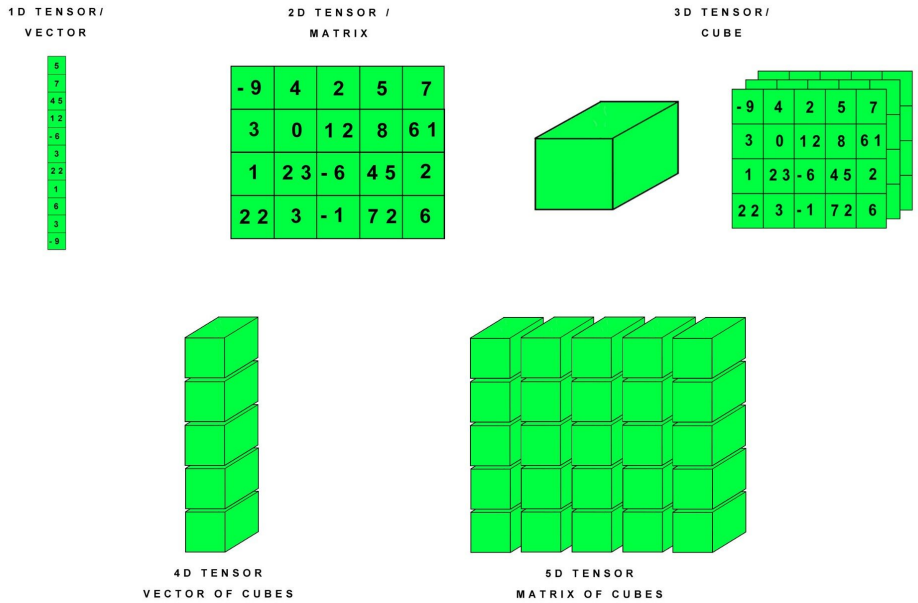
Pytorch Nedir?

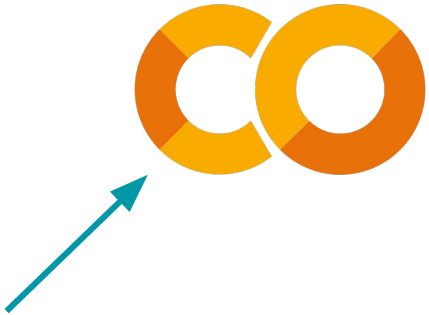
PyTorch is a Python-based scientific computing package serving two broad purposes:

- A replacement for NumPy to use the power of GPUs and other accelerators.
- An automatic differentiation library that is useful to implement neural networks.

Tensors

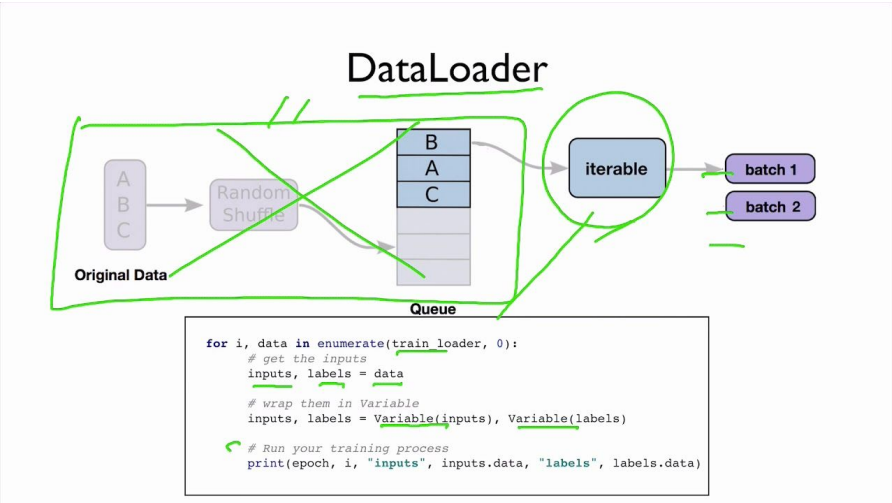
Tensors are a specialized data structure that are very similar to arrays and matrices. In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters.





Dataset & Dataloader

Like Tensorflow, PyTorch has a number of datasets included in the package (including **Text**, **Image**, and **Audio** datasets). The deep learning part of this tutorial will use one of these built-in image datasets: **CIFAR10**. These datasets are very common, and widely documented around the ML community, so they are great for prototyping and benchmarking models, since you can compare the performance of your model to what others were able to achieve with theirs.



Dataset Example

```
import torch
import torchvision
from torchvision.datasets import FashionMNIST # torchvision for image datasets
from torchtext.datasets import AmazonReviewFull # torchtext for text
from torchaudio.datasets import SPEECHCOMMANDS #torchaudio for audio

training_data = FashionMNIST(
    # the directory you want to store the dataset, can be a string e.g. "data"
    root = '',
    # if set to False, will give you the test set instead
    train = True,
    # download the dataset if it's not already available in the root path you specified
    download = True,
    # as the name implies, will transform images to tensor data structures so PyTorch can use them for training
    transform = torchvision.transforms.ToTensor()
)
```

```
1 training_data.classes
```

```
['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']
```

```
1 training_data.class_to_idx
```

```
{'T-shirt/top': 0,
 'Trouser': 1,
 'Pullover': 2,
 'Dress': 3,
 'Coat': 4,
 'Sandal': 5,
 'Shirt': 6,
 'Sneaker': 7,
 'Bag': 8,
 'Ankle boot': 9}
```

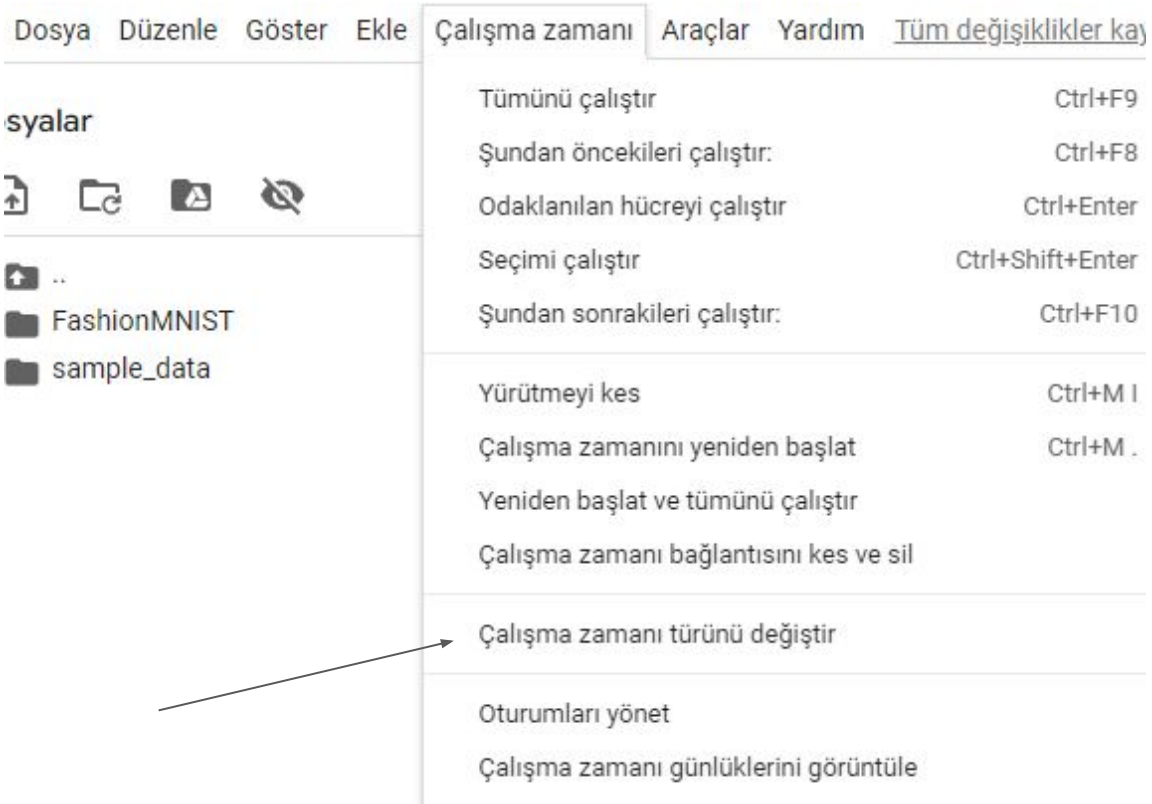
Dataloader

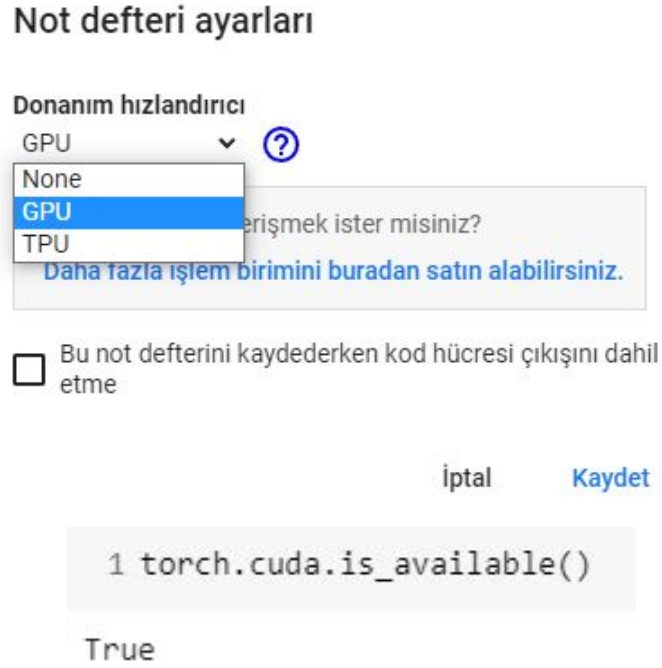
Iterating through the dataset will go through each sample 1 by 1, so PyTorch gives us the **DataLoader** module to easily create **minibatches** in our datasets. `DataLoader` allows us to specify the `batch_size` as well as shuffle the data:

```
1 from torch.utils.data import DataLoader
2 train_dataloader = DataLoader(training_data, batch_size = 32, shuffle = True)
```

So in your deep learning workflow, you'll want to feed your data to your model for training through `DataLoader` in minibatches.

Using GPU in Pytorch





Using GPU in Pytorch

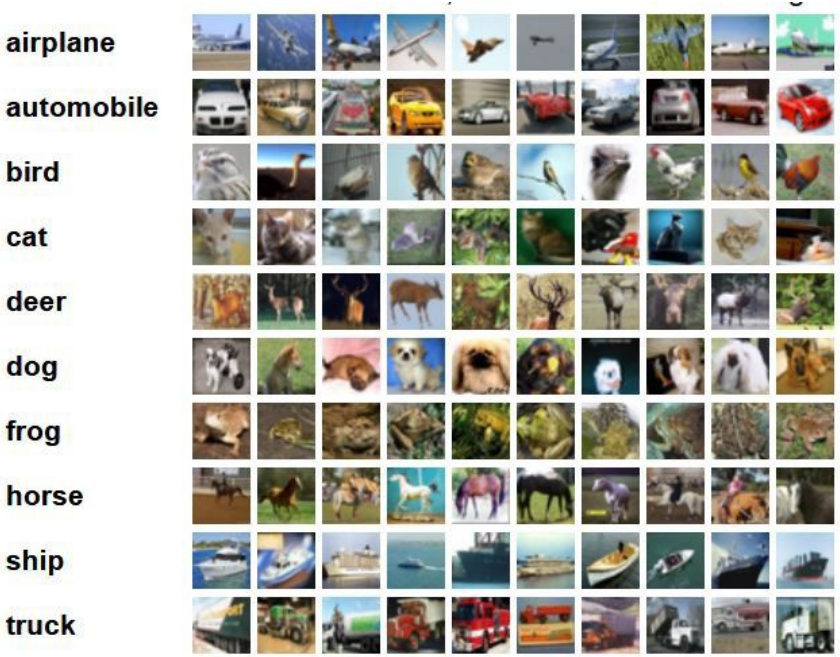
```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Device: {device}")
# 'cuda'

# you can specify .to("cuda") or .to(device)
tensor = tensor.to("cuda")

# attaching your neural network model to your GPU
model = model.to(device)
```

```
1 !nvidia-smi -L
```

```
GPU 0: Tesla T4 (UUID: GPU-ccc1ef61-2360-4e7b-b9d3-fb50212a91d7)
```

Örnek Proje: Görsel Sınıflandırma

Bugün proje kapsamında CIFAR-10 verisetini kullanarak resim sınıflandırma yapacağız. CIFAR-10 veriseti 10 sınıfta ayrılmış 32x32 boyutunda 60.000 resimden oluşuyor.



Verisetini yüklemek

```
training_data = CIFAR10(root="cifar",
                        train = True, # train set, 50k images
                        download = True,
                        transform=transform)
test_data = CIFAR10(root = "cifar",
                    train = False, # test set, 10k images
                    download = True,
                    transform = transform)
```

Verisetini yükledikten sonra neural network'e verebilmek için Dataloader'ı kullanacağız.

```
batch_size = 4
train_dataloader = DataLoader(training_data,
                              batch_size=batch_size,
                              shuffle=True)
test_dataloader = DataLoader(test_data,
                             batch_size=batch_size,
                             shuffle=True)
```

Verisetini yüklemek

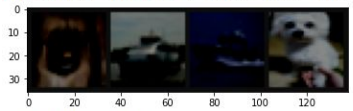
Dataloader iterasyonel bir yapıya sahip. Bunu denemek için bir for döngüsünde dataloader'ı çalıştırarak ilk elemana göz atalım.

```
for X, y in train_dataloader:
  print(f"Shape of X [N, C, H, W]: {X.shape}")
  print(f"Shape of y: {y.shape} {y.dtype}")
  break
```

```
Shape of X [N, C, H, W]: torch.Size([4, 3, 32, 32])
Shape of y: torch.Size([4]) torch.int64
```

```
def imshow(img):
  img = img / 2 + .05 # revert normalization for viewing
  npimg = img.numpy()
  plt.imshow(np.transpose(npimg, (1,2,0)))
  plt.show()

classes = training_data.classes
print(training_data.classes)

dataiter = iter(train_dataloader)
images, labels = dataiter.next()
imshow(make_grid(images))
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

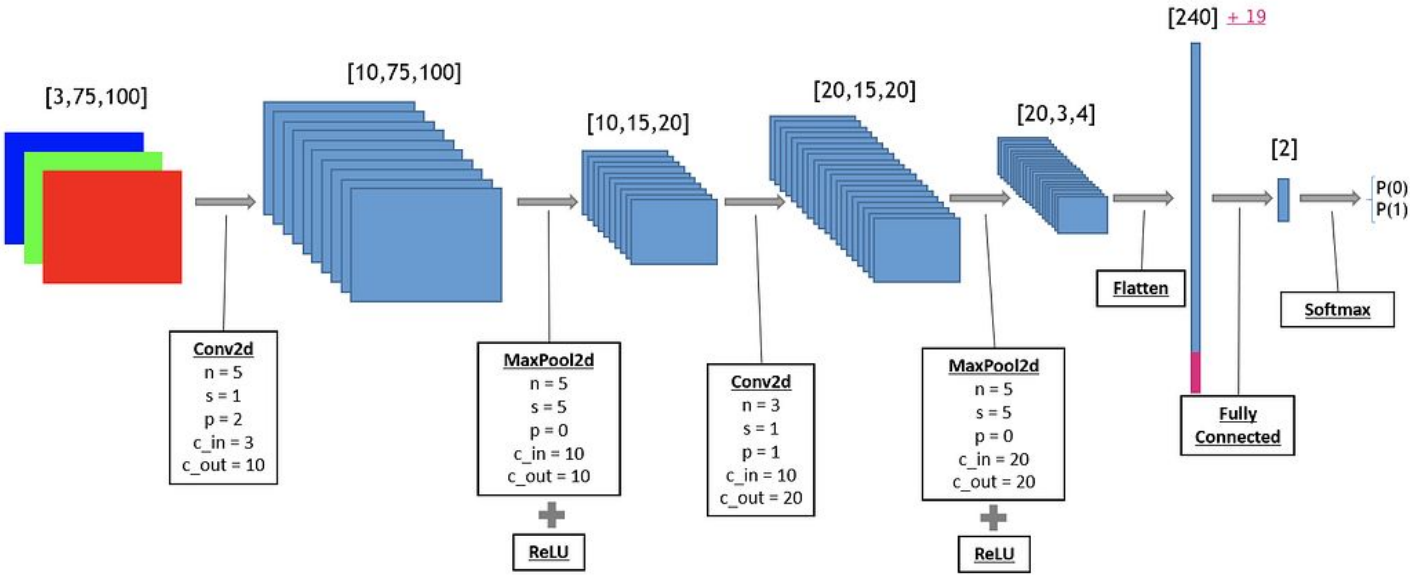



```
dog   ship  ship  dog
```

NN Yapısını kurmak

Modelimizi kurgulamak için ilk olarak NeuralNetwork adında bir class oluşturalım. Bu Pytorch nn.Module'ın bir subclass'ı olacak.

Verisetimizde Renkli görüntüler olduğundan 32x32'lik 3 RGB Renk kanallı tensorlerden oluştuğu için boyutları hep (3, 32, 32) olarak geliyor. İlk modelimiz tamamıyla birbiriyle bağlantılı katmanlardan oluşacağı için girdi görüntü verimizi düzleştirmemiz gerekiyor.



NN Yapısını kurmak

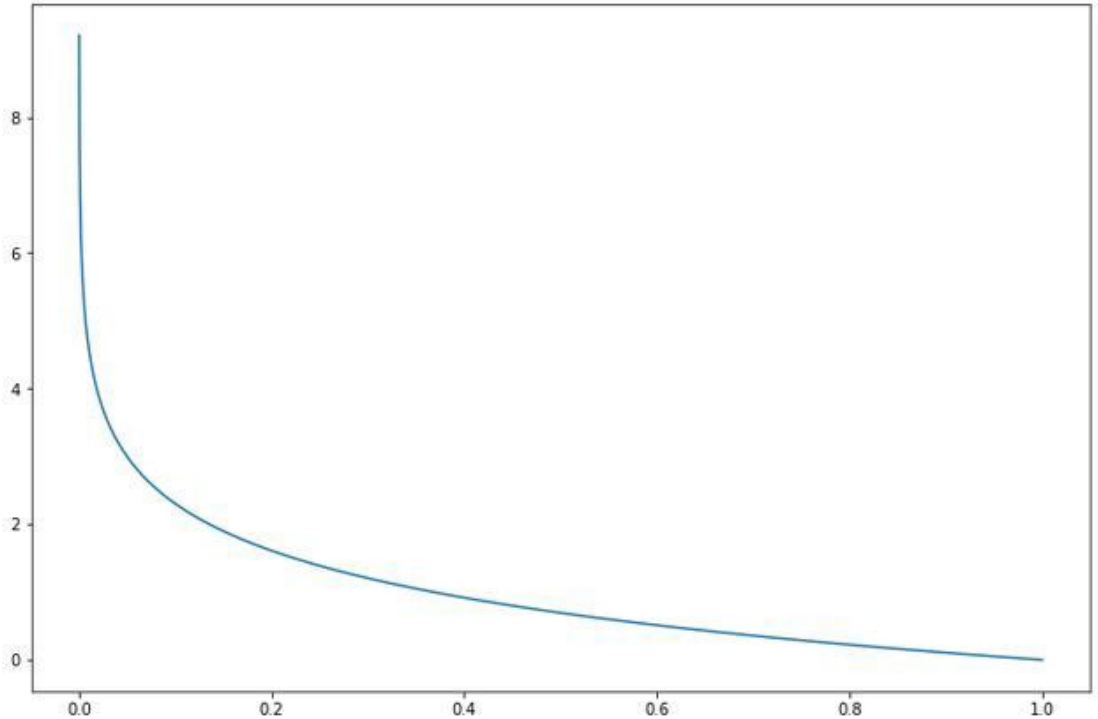
Modelimizin çıktısı ise veri kümemizdeki 10 sınıfa karşılık gelen 10 logittir. Modelin yapısını tanımladıktan sonra forward pass yapısını kurgulayacağız. Bu çalışmada oldukça basit bir model olduğu için giriş model kapsamında giriş tensörlerinden bir çıkış tensörü hesaplanacaktır.

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(32*32*3, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)

print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=3072, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

NN Yapısını kurmak

Bir sınıflandırma probleminde çalıştığımız için Loss fonksiyonu olarak Cross-Entropy Loss'u kullanacağız. Cross-Entropi model'in 0 ile 1 arasındaki tahmin edilen olasılık değerine göre log kaybını hesaplar. Dolayısıyla tahmin ettiğimiz olasılık gerçek değerden saptıkça loss değerimiz hızla artar.



Yandaki grafik tahmin edilen olasılıklar gerçek değere yaklaştıkça loss'un davranışını göstermektedir.

Training yapısını kurgulamak

```python
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)
        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if batch % 2000 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

Sol tarafta ise modelimizin eğitim sürecini gerçekleştirecek train loop'u tanımlıyoruz.

İlk adımda verisetinin boyutunu belirlemek için dataloader'dan aldığımız verisetinin uzunluğunu size olarak tanımlıyoruz.

Sonrasında yukarıda tanımladığımız pytorch nn yapısının train haline gelmesini sağlıyoruz.

For loop'unun içerisinde adım adım tensorleri gpu'ya geçip sonrasında olasılıkları ve loss'u hesaplatıyoruz. Sonrasında ise gradyanı sıfırlayıp ağırlıkları hesaplatıp güncelliyoruz.

En son kısımda ise 2000 veri için hesaplanan kaybın çıktısını alarak güncel loss'u yazdırıyoruz.

Test yapısını kurgulamak

```python
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Model eğitimine geçmeden önce modelimizi her epoch'ta değerlendirebilmek için test fonksiyonunu tanımlayalım. Test fonksiyonundaki ana farklılık test sırasında geri yayılımı kullanmadığımız için gradyan hesaplamasını torch.no_grad() ile devre dışı bırakmamız. Son olarak ölçüm için ortalama kaybı ve genel doğruluğu hesaplıyoruz.

Test yapısını kurgulamak

```python
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Model eğitimine geçmeden önce modelimizi her epoch'ta değerlendirebilmek için test fonksiyonunu tanımlayalım. Test fonksiyonundaki ana farklılık test sırasında geri yayılımı kullanmadığımız için gradyan hesaplamasını torch.no_grad() ile devre dışı bırakmamız. Son olarak ölçüm için ortalama kaybı ve genel doğruluğu hesaplıyoruz.

Modelin Eğitilmesi

Bütün tanımları yaptıktan sonra eğitim için hazırız :) Son olarak epoch sayısını belirterek eğitim sürecini başlatacağız. Her epoch'ta train loop'una gireceğiz ardından genel epoch'un test sonuçlarına bakacağız.

```
epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 2.303754  [    0/50000]
loss: 1.926201  [ 8000/50000]
loss: 1.925548  [16000/50000]
loss: 1.889365  [24000/50000]
loss: 1.399667  [32000/50000]
loss: 1.670289  [40000/50000]
loss: 1.553211  [48000/50000]
Test Error:
 Accuracy: 39.9%, Avg loss: 1.712941

Epoch 2
-------------------------------
loss: 1.492062  [    0/50000]
loss: 1.718390  [ 8000/50000]
loss: 1.148451  [16000/50000]
loss: 1.823087  [24000/50000]
loss: 1.500117  [32000/50000]
loss: 1.297272  [40000/50000]
loss: 1.012112  [48000/50000]
Test Error:
 Accuracy: 45.4%, Avg loss: 1.569835
```

Modelin Kaydedilmesi

Süper! Modelimizi eğittik. Bir sonraki adımda bu modeli canlıda veya daha sonra kullanabilmek için kaydedeceğiz. Bunun için torch.save() fonksiyonunu kullanıyoruz. Bu fonksiyona 2 parametre veriyoruz, bunlardan biri modelin ağırlıkları (bunun için model.state_dict()'i kullanıyoruz. İkinci olarak modelin adı (Bunun için deneme_model.pt veya model.pth gibi bir uzantı verebilirsiniz. Pytorch modellerinde .pt/.pth kullanmak yaygın bir kural :))

```
torch.save(model.state_dict(), "cifar_fc.pth")
```

Daha sonrasında bu modeli kullanmak isterseniz torch.load() fonksiyonunu kullanabilirsiniz.

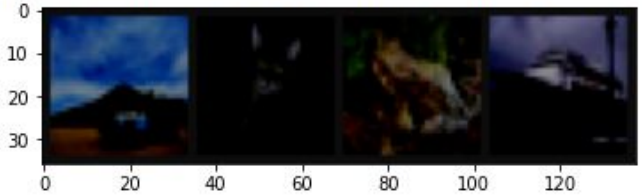
```
model = NeuralNetwork()
model.load_state_dict(torch.load("cifar_fc.pth"))
```

Model Çıktılarına Bakılması

Model çıktılarına bakmak için daha önce tanımladığımız test_loader objesini kullanacağız. Burada örnek görseller alarak onlar için tahmin yapabiliriz.

```
dataiter = iter(test_dataloader)
images, labels = dataiter.next()

imshow(make_grid(images))
print('Ground Truth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




```
Ground Truth:  truck cat   frog  ship
```

```
outputs = model(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{classes[predicted[j]]:5s}' for j in range(4)))
```

```
Predicted:  airplane frog  frog  airplane
```

Model Çıktılarına Bakılması

Modelimiz bütün verisetinde ne kadar iyi çalıştığına bakmak için bütün test_loader objelerine tek tek tahmin yaptırıp ne kadar doğru ve ne kadar yanlış yaptığına bakalım.

```
correct = 0
total = 0

with torch.no_grad():
   for data in test_dataloader:
     images, labels = data
     outputs = model(images)
     _, predicted = torch.max(outputs.data, 1)
     total += labels.size(0)
     correct += (predicted == labels).sum().item()

print(f'Model accuracy: {100 * correct // total} %')
# Model accuracy: 53 %
```

```
Model accuracy: 53 %
```

Çok süper olmasa da en azından bir şeyler tahmin edebiliyor :D

Model Performansının Ölçülmesi

Tek tek sınıflar bazında ne kadar iyi yaptığına bakalım.

```
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0  for classname in classes}

with torch.no_grad():
  for data in test_dataloader:
    images, labels = data
    outputs = model(images)
    _, predictions = torch.max(outputs, 1)
    for label,prediction in zip(labels, predictions):
      if label == prediction:
        correct_pred[classes[label]] += 1
      total_pred[classes[label]] += 1

for classname, correct_count in correct_pred.items():
  accuracy = 100 * float(correct_count) / total_pred[classname]
  print(f'Accuracy for class {classname:5s}: {accuracy:.1f}%')
```

```
Accuracy for class airplane: 68.8%
Accuracy for class automobile: 63.5%
Accuracy for class bird : 37.0%
Accuracy for class cat  : 36.8%
Accuracy for class deer : 45.2%
Accuracy for class dog  : 49.2%
Accuracy for class frog : 65.0%
Accuracy for class horse: 57.4%
Accuracy for class ship : 60.1%
Accuracy for class truck: 55.7%
```

Kaynakça

- Tensors
- CIFAR-10 Örneği & Data Loaders