Curso Superior em Tecnologia em Análise de Dados
Guilherme Barbosa R.A. 8088513

Tópicos em Redes Neurais e Deep Learning: Portfólio 2

Trabalho apresentado ao Centro Universitário Claretiano para a disciplina Tópicos em Redes Neurais e Deep Learning, ministrada pelo Professor Israel Valdecir de Souza

São Carlos-SP
2021

Código com capturas de tela mostrando seu funcionamento

```python
# %%
# imports
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 CIFAR10
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
from tqdm import tqdm
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(f'PyTorch executando no device: {device}')
```

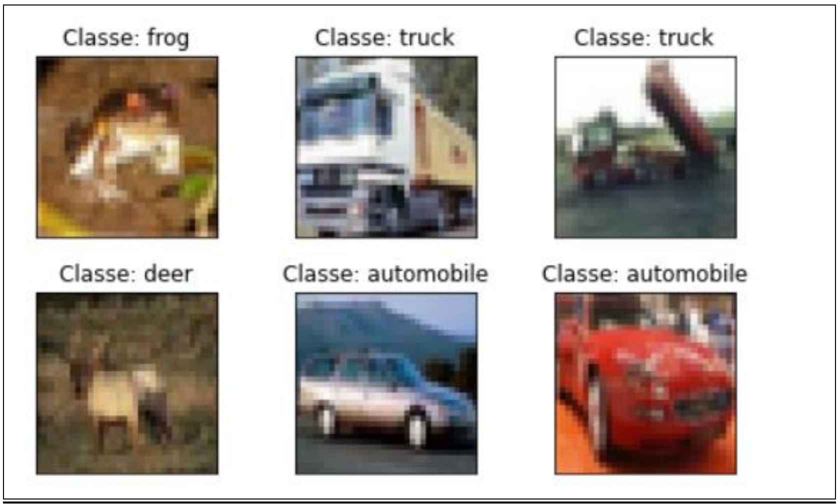
```python
def count_params(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

```python
# Carregamento do dataset

dataset = CIFAR10(root='data', download=True,train=True, transform=ToTensor())
dataset_test = CIFAR10(root='data', download=True,train=False, transform=ToTensor())
class_name = {
    0: 'airplane',
    1: 'automobile',
    2: 'bird',
    3: 'cat',
    4: 'deer',
    5: 'dog',
    6: 'frog',
    7: 'horse',
    8: 'ship',
    9: 'truck',
}
```

```python
# Exibindo dados do dataset
print(f'Quantidade imagens Treino: {len(dataset)}')
print(f'Shape da imagem: {dataset.data[0].shape}')
print(f'Quantidade imagens Teste: {len(dataset_test)}')
print(f'Shape da imagem: {dataset_test.data[0].shape}')
```

```
Quantidade imagens Treino: 50000
Shape da imagem: (32, 32, 3)
Quantidade imagens Teste: 10000
Shape da imagem: (32, 32, 3)
```

```python
# Exemplo de figuras e suas respectivas classes

fig = plt.figure()
for i in range(6):
    plt.subplot(2, 3, i + 1)
    plt.tight_layout()
    plt.imshow(dataset.data[i], cmap='gray')
    plt.title("Classe: {}".format(class_name[dataset.targets[i]]))
    plt.xticks([])
    plt.yticks([])
```




```python
val_ds = (dataset_test)   # 10000
train_ds = (dataset)  # 50000
len(train_ds), len(val_ds)
```

```python
# Definindo variáveis e hiper-parâmetros para a rede neural
num_classes = 10  # 10 tipos de imagens
batch_size = 30
lr = 1e-4
epochs = 30
```

```python
# Definindo os dataloaders
train_loader = DataLoader(train_ds, batch_size, shuffle=True, pin_memory=True)
val_loader = DataLoader(val_ds, batch_size, pin_memory=True)
# obtendo uma amostra do train_loader. Ela terá a quantidade de imagens
# definidas pelo batch_size,
images, labels = iter(train_loader).next()
print(images.shape)
print(labels.shape)
```

```
torch.Size([30, 3, 32, 32])
torch.Size([30])
```

```
class ConvCIFAR10(nn.Module):
    def __init__(self, num_classes):
        super(ConvCIFAR10, self).__init__()
        self.conv_layer = nn.Sequential(

            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout2d(p=0.05),

            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.fc_layer = nn.Sequential(
            nn.Dropout(p=0.1),
            nn.Linear(4096, 1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.1),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.conv_layer(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layer(x)
        return x
```

```python
cifar10_model = ConvCIFAR10(num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=cifar10_model.parameters(), lr=lr)
```

```python
# Checando se a rede neural está produzindo as 10 saídas
input_temp = torch.rand([1, 3, 32, 32], dtype=torch.float).to(device)
cifar10_model(input_temp).shape
```

```python
for t in cifar10_model.parameters():
    print(t.shape)

print(f'Total de parametros a serem treinados: {count_params(cifar10_model)}')
```

```
torch.Size([32, 3, 3, 3])
torch.Size([32])
torch.Size([32])
torch.Size([32])
torch.Size([64, 32, 3, 3])
torch.Size([64])
torch.Size([128, 64, 3, 3])
torch.Size([128])
torch.Size([128])
torch.Size([128])
torch.Size([128, 128, 3, 3])
torch.Size([128])
torch.Size([256, 128, 3, 3])
torch.Size([256])
torch.Size([256])
torch.Size([256])
torch.Size([256, 256, 3, 3])
torch.Size([256])
torch.Size([1024, 4096])
torch.Size([1024])
torch.Size([512, 1024])
torch.Size([512])
torch.Size([10, 512])
torch.Size([10])
Total de parametros a serem treinados: 5852170
```

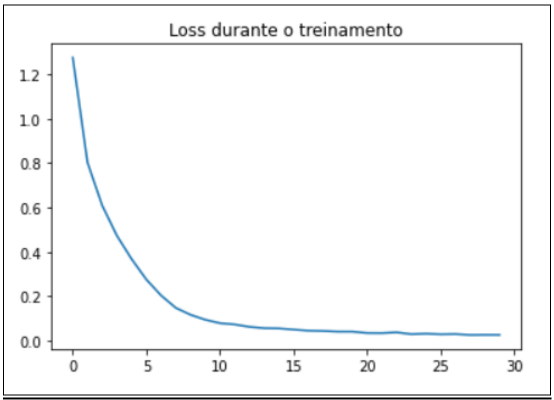
```python
# Loop de treinamento
train_loss = []  # utilizados para plotar o gráfico ao final do treinamento
train_acc = []

for e in range(epochs):
    _loss = []
    _acc = []
    for data in tqdm(train_loader):

        inputs, targets = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()
        outputs = cifar10_model(inputs)
```

```
        # calcular acurácia, comparando o valor y_pred aos labels
        # e adicionar na lista _acc
        _, y_pred = torch.max(torch.softmax(outputs, dim=-1), 1)
        acc = (y_pred.squeeze() == targets.squeeze()).detach().cpu().numpy()
        _acc.extend(acc)

        loss = criterion(outputs, targets)
        _loss.append(loss.item())
        loss.backward()
        optimizer.step()

# ao término de cada epoch, imprimir as estatísticas de treino
    train_loss.append(np.mean(_loss))
    train_acc.append(np.mean(_acc))
print(f'\nEpoch: {e+1} - loss: {np.mean(_loss):.4f} - acc.: {np.mean(_acc):.4f}')
```

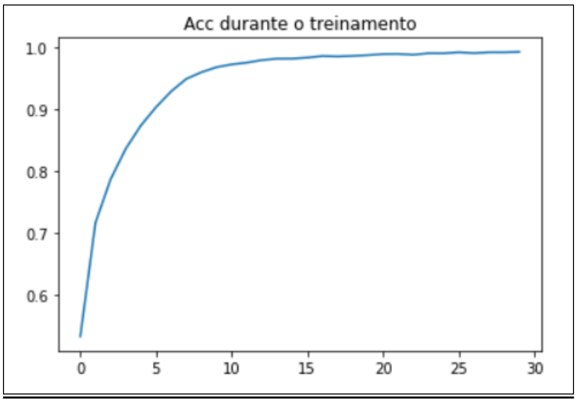
```
100%|██████████| 1667/1667 [00:16<00:00, 100.32it/s]
Epoch: 30 - loss: 0.0268 - acc.: 0.9917
```

```
# Salvando o modelo
torch.save(cifar10_model.state_dict(), 'cifar10_cnn.pt')
```

```
# Loss do Treinamento
x = len(train_loss)
plt.plot(np.arange(x), train_loss)
plt.title('Loss durante o treinamento')
plt.show()
```




```
# Acurácia do treinamento
plt.plot(np.arange(x), train_acc)
plt.title('Acc durante o treinamento')
plt.show()
```

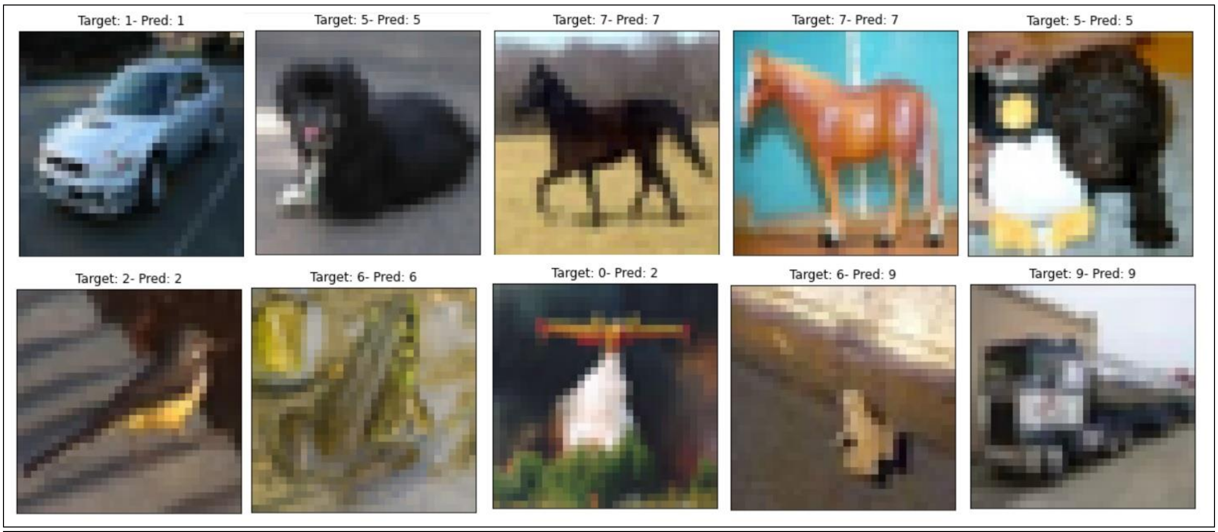



```
# Aplicando o modelo nos dados de validação
cifar10_model.load_state_dict(torch.load('cifar10_cnn.pt', map_location=device))
cifar10_model.eval()
_val_loss = []
_val_acc = []

for data in tqdm(val_loader):
    inputs, targets = data[0].to(device), data[1].to(device)
    with torch.no_grad():
        outputs = cifar10_model(inputs)

    _, y_pred = torch.max(torch.softmax(outputs, dim=-1), 1)
    acc = (y_pred.squeeze() == targets.squeeze()).detach().cpu().numpy()
    _val_acc.extend(acc)

    loss = criterion(outputs, targets)
    _val_loss.append(loss.item())

print(f'\nval_loss: {np.mean(_val_loss):.4f} - val_acc.: {np.mean(_val_acc):.4f}')
```

```
100%|██████████| 334/334 [00:01<00:00, 171.84it/s]
val_loss: 0.9457 - val_acc.: 0.8351
```

```
# Testando o modelo para verificar as fotos do dataset
cifar10_model.eval()

for i in range(10):
  idx = np.random.randint(0,1000)
  imagem_test =  dataset_test.data[idx]
  target_test = dataset_test.targets[idx]

  input_test = imagem_test / 255.
  input_test = np.expand_dims(input_test, 1)
  input_test = input_test.reshape(32,1,32,3)
  input_test = np.transpose(input_test, [1,3,0,2])
  input_test = input_test.astype(np.float32)
  input_tensor = torch.from_numpy(input_test).to(device)
```

```
    with torch.no_grad():

  input_tensor = cifar10_model(input_tensor)
  pred = torch.argmax(torch.softmax(input_tensor, dim=-1)).detach().cpu().numpy()


plt.tight_layout()
plt.xticks([])
plt.yticks([])
plt.imshow(imagem_test, cmap='gray')
plt.title(f'Target: {target_test}- Pred: {pred}')
plt.show()
```



Link do código:

https://colab.research.google.com/drive/17D-at348TRJ-8Bot_Suy4UKH2df3gt9m#scrollTo=lTrNv6UjQcr2