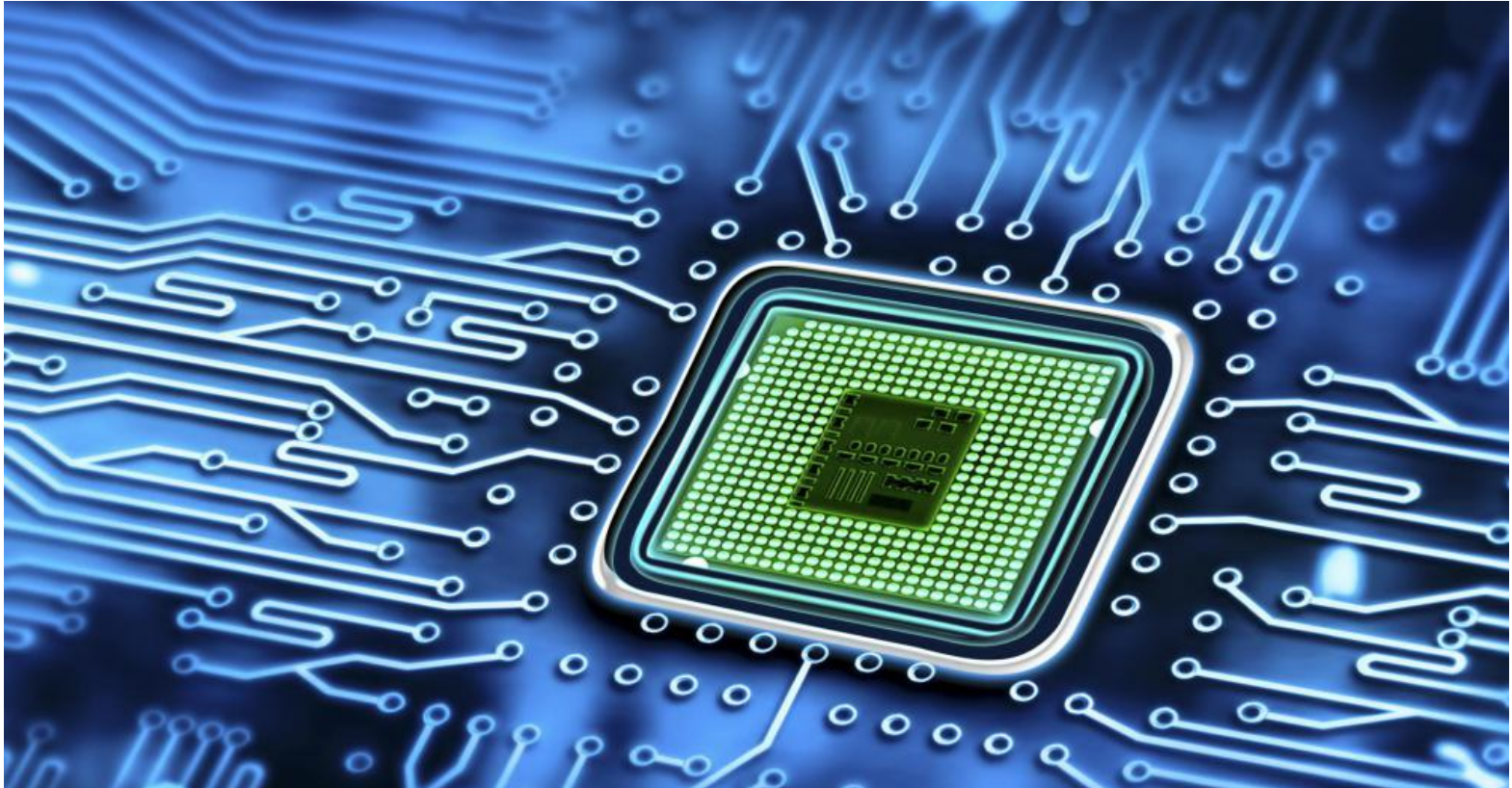


(https://praxis.ac.in/data-science-program/?utm_source=AIM&utm_medium=banner&utm_campaign=DS-July2021)

How to Implement Convolutional Autoencoder in PyTorch with CUDA

09/07/2020



The Autoencoders, a variant (<https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/>) of the artificial neural networks, are applied very successfully (<https://analyticsindiamag.com/how-to-create-your-first-artificial-neural-network-in-python/>) in the image process especially to reconstruct the images. The image reconstruction aims at generating a new set of images similar to the original input images. This helps in obtaining the noise-free or complete images if given a set of noisy or incomplete images respectively. In our last article, we demonstrated the implementation of Deep Autoencoder in image reconstruction.

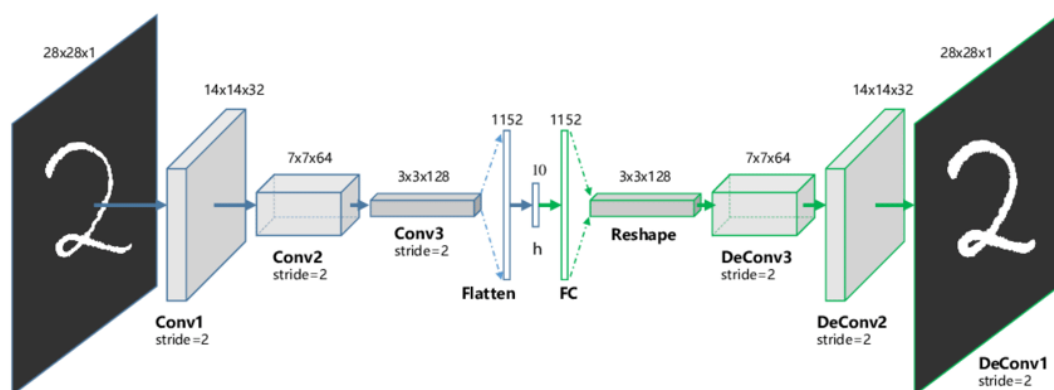
In this article, we will define a Convolutional Autoencoder in PyTorch and train it on the CIFAR-10 dataset in the CUDA environment to create reconstructed images.

Convolutional Autoencoder

Convolutional Autoencoder is a variant of Convolutional Neural Networks (<https://analyticsindiamag.com/fashion-apparel-recognition-using-convolutional-neural-network/>) that are used as the tools for unsupervised learning of convolution filters. They are generally applied in the task of image reconstruction to minimize reconstruction errors by learning

the optimal filters. Once they are trained in this task, they can be applied to any input in order to extract features. Convolutional Autoencoders are general purpose feature extractors differently from general autoencoders that completely ignore the 2D image structure. In autoencoders, the image must be unrolled into a single vector and the network must be built following the constraint on the number of inputs.

The block diagram of a Convolutional Autoencoder is given in the below figure.



(Image Source

(https://www.researchgate.net/profile/Xifeng_Guo/publication/320658590/figure/fig1/AS:614154637418504@1523437284408/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there.png))

Implementing in PyTorch

First of all, we will import the required libraries.

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
%matplotlib inline
import torch.nn as nn
import torch.nn.functional as F
```

After importing the libraries, we will download the CIFAR-10 dataset.


```
#Converting data to torch.FloatTensor
transform = transforms.ToTensor()

# Download the training and test datasets
train_data = datasets.CIFAR10(root='data', train=True, download=True, transform=transform)

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

Now, we will prepare the data loaders that will be used for training and testing.

```
#Prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size=32, num_workers=0)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=32, num_workers=0)
```

 [\(https://analyticsindiamag.com/\)](https://analyticsindiamag.com/)



We will print some random images from the training data set.

```

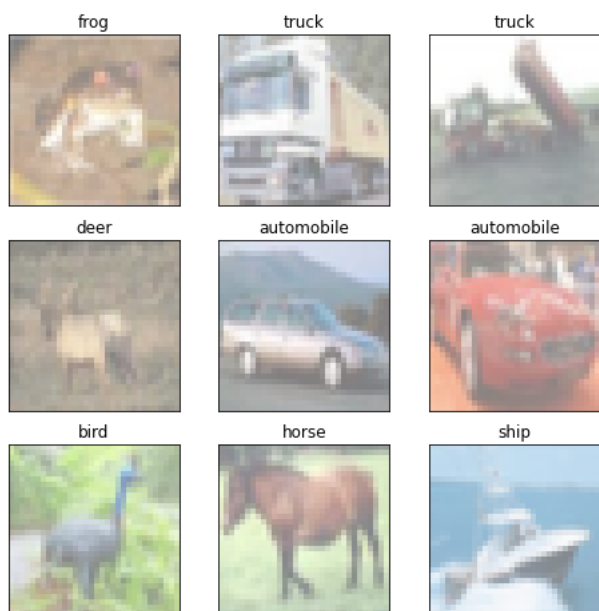
#Utility functions to un-normalize and display (https://analyticsindiamag.com/)
def imshow(img):
    img = img / 2 + 0.5
    plt.imshow(np.transpose(img, (1, 2, 0)))

#Define the image classes
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 's

#Obtain one batch of training images
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

#Plot the images
fig = plt.figure(figsize=(8, 8))
# display 20 images
for idx in np.arange(9):
    ax = fig.add_subplot(3, 3, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])

```



In the next step, we will define the Convolutional Autoencoder as a class that will be used to define the final Convolutional Autoencoder model.

SEE ALSO




(https://an
alyticsindi
amag.com/
7-
resources-
to-learn-
deep-
learning-
in-2021/)

DEVELOPERS CORNER (HTTPS://ANALYTICSINDIAMAG.COM/CATEGORY/DEVELOPERS_CORNER/)

7 Resources To Learn Deep Learning In 2021

(https://analyticsindiamag.com/7-resources-to-learn-deep-learning-in-2021/)

#Define the Convolutional Autoencoder  (<https://analyticsindiamag.com/>)

```

class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()

        #Encoder
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 4, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        #Decoder
        self.t_conv1 = nn.ConvTranspose2d(4, 16, 2, stride=2)
        self.t_conv2 = nn.ConvTranspose2d(16, 3, 2, stride=2)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.t_conv1(x))
        x = F.sigmoid(self.t_conv2(x))

        return x

#Instantiate the model
model = ConvAutoencoder()
print(model)

ConvAutoencoder(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (t_conv1): ConvTranspose2d(4, 16, kernel_size=(2, 2), stride=(2, 2))
  (t_conv2): ConvTranspose2d(16, 3, kernel_size=(2, 2), stride=(2, 2))
)

```

After that, we will define the loss criterion and optimizer.

```

#Loss function
criterion = nn.BCELoss()

#Optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)


```

Now, we will pass our model to the CUDA environment. Make sure that you are using GPU.

```
def get_device():
    if torch.cuda.is_available():
        device = 'cuda:0'
    else:
        device = 'cpu'
    return device

device = get_device()
print(device)
model.to(device)

cuda:0
ConvAutoencoder(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (t_conv1): ConvTranspose2d(4, 16, kernel_size=(2, 2), stride=(2, 2))
  (t_conv2): ConvTranspose2d(16, 3, kernel_size=(2, 2), stride=(2, 2))
)
```

 (<https://analyticsindiamag.com/>)



In the next step, we will train the model on CIFAR10 dataset.

```
#Epochs
n_epochs = 100

for epoch in range(1, n_epochs+1):
    # monitor training loss
    train_loss = 0.0

    #Training
    for data in train_loader:
        images, _ = data
        images = images.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, images)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*images.size(0)

    train_loss = train_loss/len(train_loader)
    print('Epoch: {} \tTraining Loss: {:.6f}'.format(epoch, train_loss))
```

```
Epoch: 1      Training Loss: 11.660579
Epoch: 2      Training Loss: 11.464775
Epoch: 3      Training Loss: 11.428341
Epoch: 4      Training Loss: 11.381888
Epoch: 5      Training Loss: 11.357757
Epoch: 6      Training Loss: 11.341049
Epoch: 7      Training Loss: 11.332663
Epoch: 8      Training Loss: 11.325787
Epoch: 9      Training Loss: 11.321006
Epoch: 10     Training Loss: 11.317692
```

Finally, we will train the convolutional autoencoder model on generating the reconstructed images.


```
#Batch of test images
dataiter = iter(test_loader)
images, labels = dataiter.next()

#Sample outputs
output = model(images)
images = images.numpy()

output = output.view(batch_size, 3, 32, 32)
output = output.detach().numpy()

#Original Images
print("Original Images")
fig, axes = plt.subplots(nrows=1, ncols=5, sharex=True, sharey=True, figsize=(12,4))
for idx in np.arange(5):
    ax = fig.add_subplot(1, 5, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])
plt.show()

#Reconstructed Images
print('Reconstructed Images')
fig, axes = plt.subplots(nrows=1, ncols=5, sharex=True, sharey=True, figsize=(12,4))
for idx in np.arange(5):
    ax = fig.add_subplot(1, 5, idx+1, xticks=[], yticks=[])
    imshow(output[idx])
    ax.set_title(classes[labels[idx]])
plt.show()
```