CLASSIFICATION (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/CLASSIFICATION/)

COMPUTER VISION (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/COMPUTER-VISION/)

DEEP LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/DEEP-LEARNING/)

IMAGE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/IMAGE/)

INTERMEDIATE (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/INTERMEDIATE/)

PROJECT (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PROJECT/)

PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)

PYTORCH (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTORCH/)

SUPERVISED (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/SUPERVISED/)

UNSTRUCTURED DATA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/UNSTRUCTURED-DATA/)

Build an Image Classification Model using Convolutional Neural Networks in PyTorch

PULKIT SHARMA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/PULKITS/), OCTOBER 1, 2019 LOGIN TO BOOKMARK THIS ...

Bootcamp

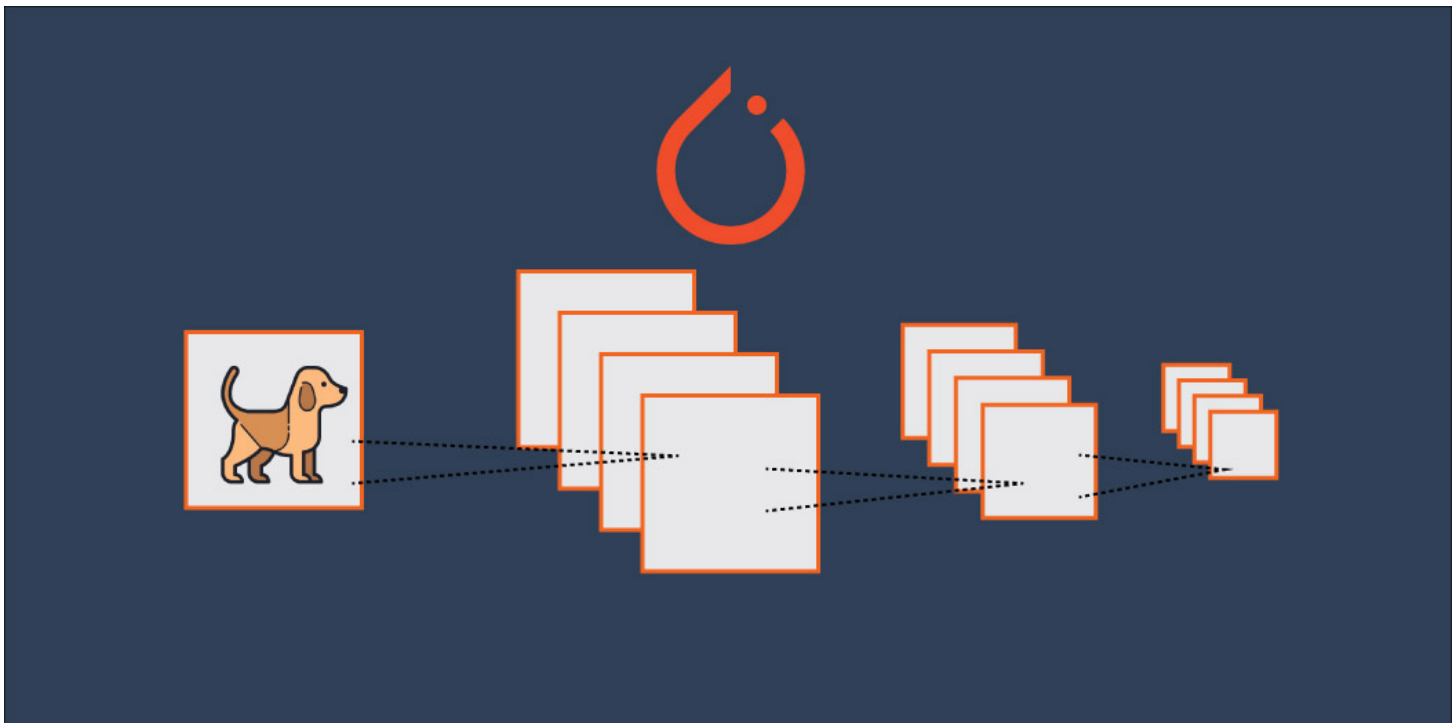
Overview

- A hands-on tutorial to build your own convolutional neural network (CNN) in PyTorch
- We will be working on an image classification problem – a classic and widely used application of CNNs
- This is part of Analytics Vidhya's series on PyTorch where we introduce deep learning concepts in a practical format

Introduction

I'm enthralled by the power and capability of neural networks. Almost every breakthrough happening in the machine learning and deep learning space right now has neural network models at its core.

This is especially prevalent in the field of computer vision (https://courses.analyticsvidhya.com/courses/computer-vision-using-deep-learning-version2?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch). Neural networks have opened up possibilities of working with image data – whether that's simple image classification or something more advanced like object detection. In short, it's a goldmine for a data scientist like me!



Bootcamp

Simple neural networks are always a good starting point when we're solving an image classification problem using deep learning. But they do have limitations and the model's performance fails to improve after a certain point.

This is where convolutional neural networks (CNNs) have changed the playing field. They are ubiquitous in computer vision applications. And it's honestly a concept I feel every computer vision enthusiast should pick up quickly.

This article is a continuation of my new series where I introduce you to new deep learning concepts using the popular PyTorch framework. In this article, we will understand how convolutional neural networks are helpful and how they can help us to improve our model's performance. We will also look at the implementation of CNNs in PyTorch.

This is the second article of this series and I highly recommend to go through the first part before moving forward with this article.

- *A Beginner-Friendly Guide to PyTorch and How it Works from Scratch (https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch)*

Also, the third article of this series is live now where you can learn how to use pre-trained models and apply transfer learning using PyTorch:

- Deep Learning for Everyone: Master the Powerful Art of Transfer Learning using PyTorch (https://www.analyticsvidhya.com/blog/2019/10/how-to-master-transfer-learning-using-pytorch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch)

Table of Contents

A Brief Overview of PyTorch, Tensors and NumPy

Let's quickly recap what we covered in the first article (https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch). We discussed the

Bootcamp

basics of PyTorch and tensors, and also looked at how PyTorch is similar to NumPy.

PyTorch is a Python-based library that provides functionalities such as:

- TorchScript for creating serializable and optimizable models
- Distributed training to parallelize computations
- Dynamic Computation graphs which enable to make the computation graphs on the go, and many more

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/02/pytorch.jpeg)

Tensors in PyTorch are similar to NumPy's n-dimensional arrays which can also be used with GPUs. Performing operations on these tensors is almost similar to performing operations on NumPy arrays. **This makes PyTorch very user-friendly and easy to learn.**

In part 1 of this series, we built a simple neural network to solve a case study. We got a benchmark accuracy of around 65% on the test set using our simple model. Now, we will try to improve this score using Convolutional Neural Networks.

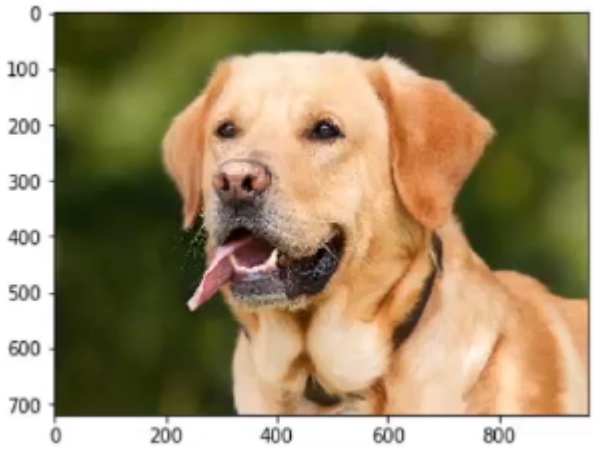
Why Convolutional Neural Networks (CNNs)?

Before we get to the implementation part, let's quickly look at why we need CNNs in the first place and how they are helpful.

> *We can consider Convolutional Neural Networks, or CNNs, as feature extractors that help to extract features from images.*

In a simple neural network, we convert a 3-dimensional image to a single dimension, right? Let's look at an example to understand this:

Can you identify the above image? Doesn't seem to make a lot of sense. Now, let's look at the below image:

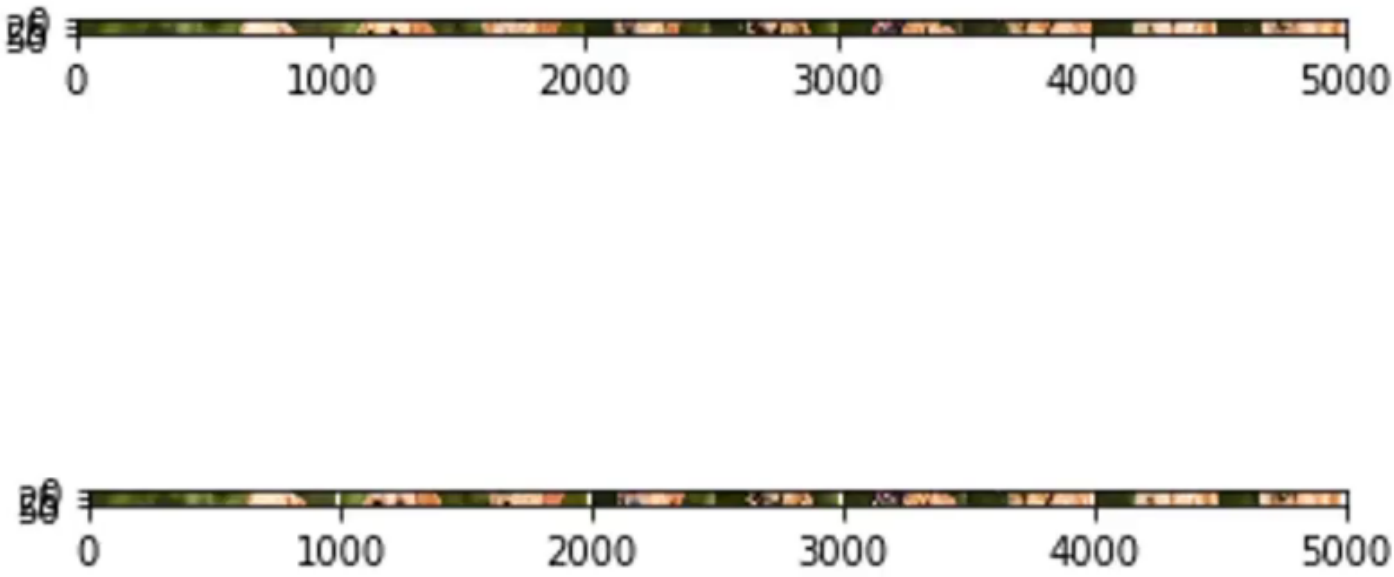


(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-11-37-15.png)
We can now easily say that it is an image of a dog. What if I tell you that both these images are the same? Believe me, they are! The only difference is that the first image is a 1-D representation whereas the second one is a 2-D representation of the same image.

Spatial Orientation

Artificial neural networks (ANNs) also lose the spatial orientation of the images. Let's again take an example and understand it:

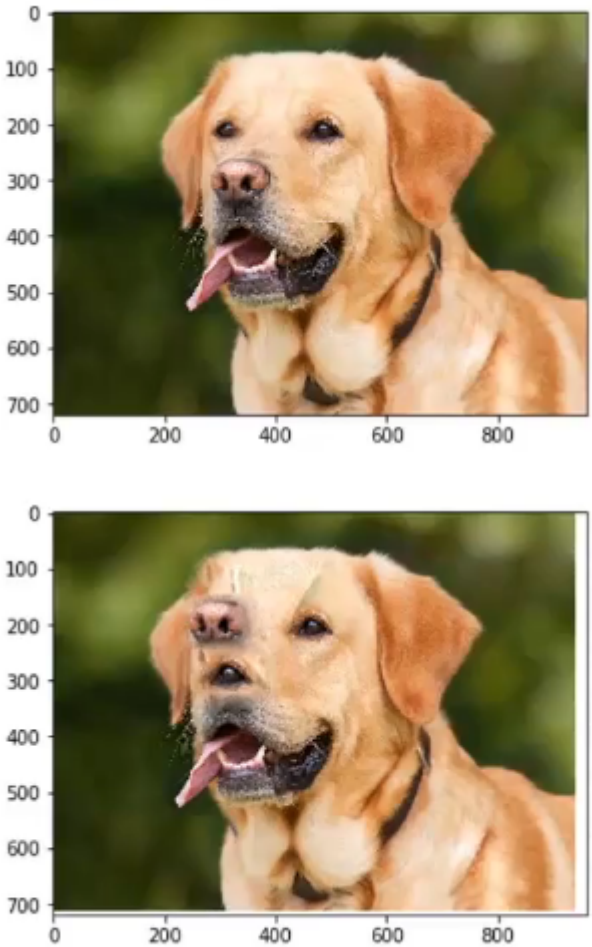
Bootcamp



(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-11-47-25.png)

Can you identify the difference between these two images? Well, at least I cannot. It is very difficult to identify the difference since this is a 1-D representation. Now, let's look at the 2-D representation of these images:

Bootcamp



(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-11-48-58.png)

Don't you love how different the same image looks by simply changing it's representation? Here, the orientation of the images has been changed but we were unable to identify it by looking at the 1-D representation.

This is the problem with artificial neural networks – they lose spatial orientation.

Large number of parameters

Another problem with neural networks is the large number of parameters at play. Let's say our image has a size of 28*28*3 – so the parameters here will be 2,352. What if we have an image of size 224*224*3? The number of parameters here will be 150,528.

And these parameters will only increase as we increase the number of hidden layers. So, the two major disadvantages of using artificial neural networks are:

1. Loses spatial orientation of the image
2. The number of parameters increases drastically

So how do we deal with this problem? How can we preserve the spatial orientation as well as reduce the learnable parameters?

This is where convolutional neural networks can be really helpful. **CNNs help to extract features from the images which may be helpful in classifying the objects in that image.** It starts by extracting low dimensional features (like edges) from the image, and then some high dimensional features like the shapes.

> *We use filters to extract features from the images and Pooling techniques to reduce the number of learnable parameters.*

We will not be diving into the details of these topics in this article. If you wish to understand how filters help to extract features and how pooling works, I highly recommend you go through A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch (https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch).

Understanding the Problem Statement: Identify the Apparels

Enough theory – let's get coding! We'll be taking up the same problem statement we covered in the first article (https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch). This is because we can directly compare our CNN model's performance to the simple neural network we built there.

You can download the dataset for this 'Identify' the Apparels' problem from here (https://datahack.analyticsvidhya.com/contest/practice-problem-identify-the-apparels/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch).

Let me quickly summarize the problem statement. Our task is to identify the type of apparel by looking at a variety of apparel images. There are a total of 10 classes in which we can classify the images of apparels:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover

3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

The dataset contains a total of 70,000 images. 60,000 of these images belong to the training set and the remaining 10,000 are in the test set. All the images are grayscale images of size (28*28). The dataset contains two folders – one each for the training set and the test set. In each folder, there is a .csv file that has the *id* of the image and its corresponding label, and a folder containing the images for that particular set.

Ready to begin? We will start by importing the required libraries:

```
# importing the libraries
import pandas as pd
import numpy as np

# for reading and displaying images
from skimage.io import imread
import matplotlib.pyplot as plt
%matplotlib inline

# for creating validation set
from sklearn.model_selection import train_test_split

# for evaluating the model
from sklearn.metrics import accuracy_score
from tqdm import tqdm

# PyTorch libraries and modules
import torch
from torch.autograd import Variable
from torch.nn import Linear, ReLU, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module, So
from torch.optim import Adam, SGD
```

```
ub.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/library.py)
```

library.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-library-py) hosted with ❤ by GitHub (https://github.com)

Loading the dataset

Now, let's load the dataset, including the train, test and sample submission file:

```
1  # loading dataset
2  train = pd.read_csv('train_LbELtWX/train.csv')
3  test = pd.read_csv('test_ScVgIM0/test.csv')
4
5  sample_submission = pd.read_csv('sample_submission_I5njJSF.csv')
6
7  train.head()
```

```
ub.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/dataset.py)
```

dataset.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-dataset-py) hosted with ❤ by GitHub (https://github.com)

	id	label
0	1	9
1	2	0
2	3	0
3	4	3
4	5	0

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-14-33-23.png)

- The train file contains the id of each image and its corresponding label
- **The test file, on the other hand, only has the ids and we have to predict their corresponding labels**
- The sample submission file will tell us the format in which we have to submit the predictions

We will read all the images one by one and stack them one over the other in an array. We will also divide the pixels of images by 255 so that the pixel values of images comes in the range [0,1]. This step helps in optimizing the performance of our model.

So, let's go ahead and load the images:

```
1  # loading training images
```

```
train_img = []
for img_name in tqdm(train['id']):
    # defining the image path
    image_path = 'train_LbELtWX/train/' + str(img_name) + '.png'
    # reading the image
    img = imread(image_path, as_gray=True)
    # normalizing the pixel values
    img /= 255.0
    # converting the type of pixel to float 32
    img = img.astype('float32')
    # appending the image into the list
    train_img.append(img)

# converting the list to numpy array
train_x = np.array(train_img)
# defining the target
train_y = train['label'].values
train_x.shape
```

.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/train_data.py)
train_data.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-train_data-py) hosted with ❤️ by GitHub (https://github.com)

```
(60000, 28, 28)
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-14-41-12.png)

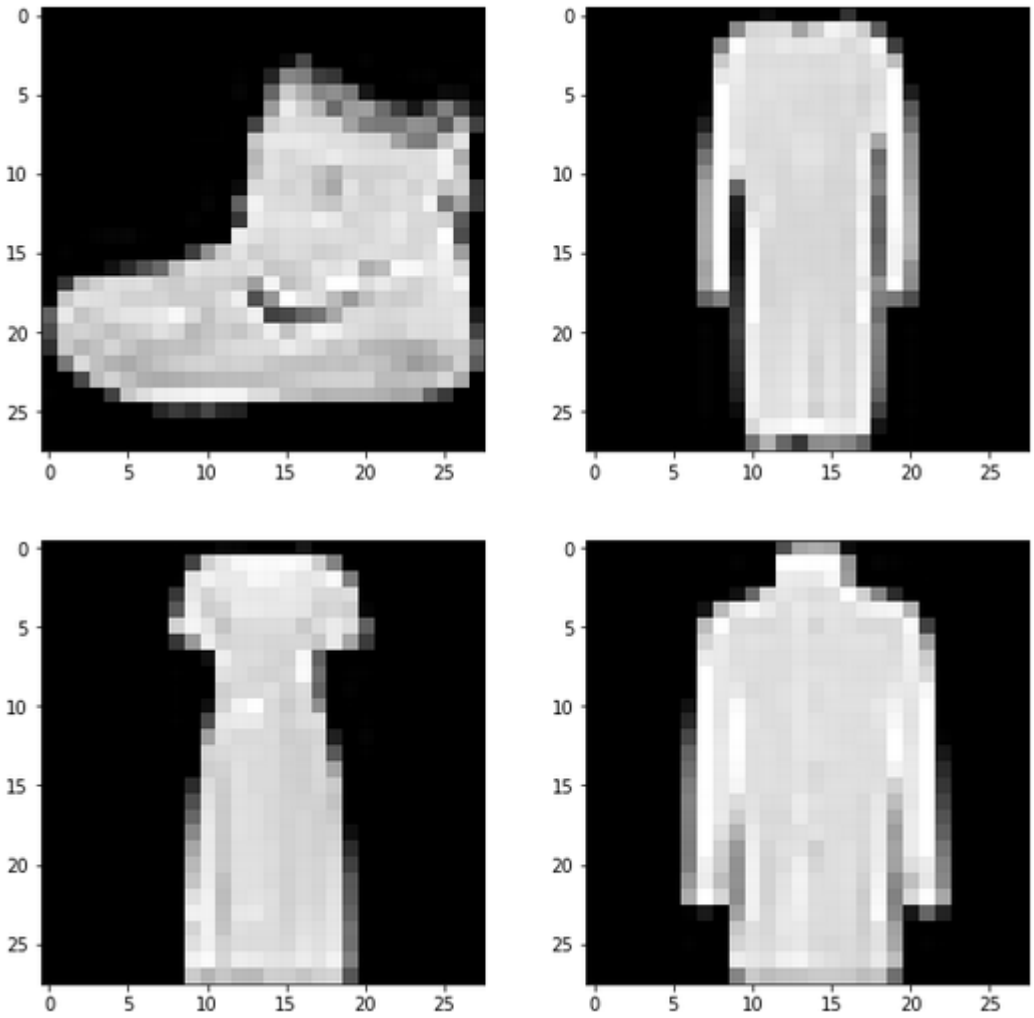
As you can see, we have 60,000 images, each of size (28,28), in the training set. Since the images are in grayscale format, we only have a single-channel and hence the shape (28,28).

Let's now explore the data and visualize a few images:

```
# visualizing images
i = 0
plt.figure(figsize=(10,10))
plt.subplot(221), plt.imshow(train_x[i], cmap='gray')
plt.subplot(222), plt.imshow(train_x[i+25], cmap='gray')
plt.subplot(223), plt.imshow(train_x[i+50], cmap='gray')
plt.subplot(224), plt.imshow(train_x[i+75], cmap='gray')
```

b.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/visualize.py)
visualize.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-visualize-py) hosted with ❤️ by GitHub (https://github.com)

Bootcamp



(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-14-58-49.png)

These are a few examples from the dataset. I encourage you to explore more and visualize other images. Next, we will divide our images into a training and validation set.

Creating a validation set and preprocessing the images

```
1   # create validation set
2   train_x, val_x, train_y, val_y = train_test_split(train_x, train_y, test_size = 0.1)
3   (train_x.shape, train_y.shape), (val_x.shape, val_y.shape)
```

'PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/validation_data.py)
validation_data.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-validation_data-py)
hosted with ❤️ by GitHub (https://github.com)

Bootcamp

```
(((54000, 28, 28), (54000,)), ((6000, 28, 28), (6000,)))
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-04-14.png)

We have kept 10% data in the validation set and the remaining in the training set. Next, let's convert the images and the targets into torch format:

```
# converting training images into torch format
train_x = train_x.reshape(54000, 1, 28, 28)
train_x  = torch.from_numpy(train_x)

# converting the target into torch format
train_y = train_y.astype(int);
train_y = torch.from_numpy(train_y)

# shape of training data
train_x.shape, train_y.shape
```

kitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/train_torch_format.py) train_torch_format.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-train_torch_format-py) hosted with ❤️ by GitHub (https://github.com)

```
(torch.Size([54000, 1, 28, 28]), torch.Size([54000]))
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-09-12.png)

Similarly, we will convert the validation images:

```
# converting validation images into torch format
val_x = val_x.reshape(6000, 1, 28, 28)
val_x  = torch.from_numpy(val_x)

# converting the target into torch format
val_y = val_y.astype(int);
val_y = torch.from_numpy(val_y)

# shape of validation data
val_x.shape, val_y.shape
```

1/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/validation_torch_format.py) validation_torch_format.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-validation_torch_format-py) hosted with ❤️ by GitHub (https://github.com)

Bootcamp

```
(torch.Size([6000, 1, 28, 28]), torch.Size([6000]))
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-12-05.png)

Our data is now ready. Finally, it's time to create our CNN model!

Implementing CNNs using PyTorch

We will use a very simple CNN architecture with just 2 convolutional layers to extract features from the images. We'll then use a fully connected dense layer to classify those features into their respective categories.

Let's define the architecture:

```
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(4),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
            # Defining another 2D convolution layer
            Conv2d(4, 4, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(4),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
        )

        self.linear_layers = Sequential(
            Linear(4 * 7 * 7, 10)
        )

    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
```

Bootcamp

```
27          return x
```

ulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/cnn_architecture.py)
cnn_architecture.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-cnn_architecture-py)
hosted with ❤️ by GitHub (https://github.com)

Let's now call this model, and define the optimizer and the loss function for the model:

```
# defining the model
model = Net()
# defining the optimizer
optimizer = Adam(model.parameters(), lr=0.07)
# defining the loss function
criterion = CrossEntropyLoss()
# checking if GPU is available
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()

print(model)
```

nub.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/model.py)
model.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-model-py) hosted with ❤️ by
GitHub (https://github.com)

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=196, out_features=10, bias=True)
  )
)
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-29-12.png)

This is the architecture of the model. We have two Conv2d layers and a Linear layer. Next, we will define a function to train the model:

Bootcamp

```
def train(epoch):
    model.train()
    tr_loss = 0
    # getting the training set
    x_train, y_train = Variable(train_x), Variable(train_y)
    # getting the validation set
    x_val, y_val = Variable(val_x), Variable(val_y)
    # converting the data into GPU format
    if torch.cuda.is_available():
        x_train = x_train.cuda()
        y_train = y_train.cuda()
        x_val = x_val.cuda()
        y_val = y_val.cuda()

    # clearing the Gradients of the model parameters
    optimizer.zero_grad()

    # prediction for training and validation set
    output_train = model(x_train)
    output_val = model(x_val)

    # computing the training and validation loss
    loss_train = criterion(output_train, y_train)
    loss_val = criterion(output_val, y_val)
    train_losses.append(loss_train)
    val_losses.append(loss_val)

    # computing the updated weights of all the model parameters
    loss_train.backward()
    optimizer.step()
    tr_loss = loss_train.item()
    if epoch%2 == 0:
        # printing the validation loss
        print('Epoch : ',epoch+1, '\t', 'loss :', loss_val)
```

n/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/train_function.py) train_function.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-train_function-py) hosted with ❤️ by GitHub (https://github.com)

Finally, we will train the model for 25 epochs and store the training and validation losses:

```
# defining the number of epochs
n_epochs = 25
```

```
# empty list to store training losses
train_losses = []
# empty list to store validation losses
val_losses = []
# training the model
for epoch in range(n_epochs):
    train(epoch)
```

ithub.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/train.py)
train.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-train-py) hosted with ❤ by GitHub (https://github.com)

```
Epoch :  1      loss : tensor(2.3163, device='cuda:0')
Epoch :  3      loss : tensor(1.5701, device='cuda:0')
Epoch :  5      loss : tensor(1.7231, device='cuda:0')
Epoch :  7      loss : tensor(1.4408, device='cuda:0')
Epoch :  9      loss : tensor(1.3423, device='cuda:0')
Epoch :  11     loss : tensor(1.2012, device='cuda:0')
Epoch :  13     loss : tensor(1.0581, device='cuda:0')
Epoch :  15     loss : tensor(0.9387, device='cuda:0')
Epoch :  17     loss : tensor(0.8601, device='cuda:0')
Epoch :  19     loss : tensor(0.8327, device='cuda:0')
Epoch :  21     loss : tensor(0.8020, device='cuda:0')
Epoch :  23     loss : tensor(0.7543, device='cuda:0')
Epoch :  25     loss : tensor(0.7362, device='cuda:0')
```

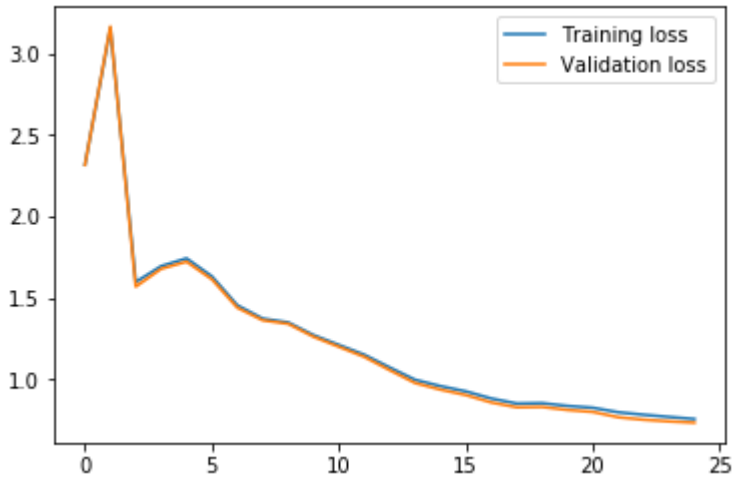
(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-34-34.png)

We can see that the validation loss is decreasing as the epochs are increasing. Let's visualize the training and validation losses by plotting them:

```
# plotting the training and validation loss
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.show()
```

ılkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/loss_visualization.py)
loss_visualization.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-loss_visualization-py) hosted with ❤ by GitHub (https://github.com)

Bootcamp



(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-36-14.png)

Ah, I love the power of visualization. We can clearly see that the training and validation losses are in sync. It is a good sign as the model is generalizing well on the validation set.

Let's check the accuracy of the model on the training and validation set:

```
# prediction for training set
with torch.no_grad():
    output = model(train_x.cuda())

softmax = torch.exp(output).cpu()
prob = list(softmax.numpy())
predictions = np.argmax(prob, axis=1)

# accuracy on training set
accuracy_score(train_y, predictions)
```

/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/train_accuracy.py) train_accuracy.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-train_accuracy-py) hosted with ❤ by GitHub (https://github.com)

```
0.7231851851851852
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-39-31.png)

An accuracy of ~72% accuracy on the training set is pretty good. Let's check the accuracy for the validation set as well:

```
# prediction for validation set
```

```
with torch.no_grad():
    output = model(val_x.cuda())

softmax = torch.exp(output).cpu()
prob = list(softmax.numpy())
predictions = np.argmax(prob, axis=1)

# accuracy on validation set
accuracy_score(val_y, predictions)
```

itS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/validation_accuracy.py) validation_accuracy.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-validation_accuracy-py) hosted with ❤️ by GitHub (https://github.com)

```
0.727
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-40-16.png)

As we saw with the losses, the accuracy is also in sync here – we got ~72% on the validation set as well.

Generating predictions for the test set

It's finally time to generate predictions for the test set. We will load all the images in the test set, do the same pre-processing steps as we did for the training set and finally generate predictions.

So, let's start by loading the test images:

```
# loading test images
test_img = []
for img_name in tqdm(test['id']):
    # defining the image path
    image_path = 'test_ScVgIM0/test/' + str(img_name) + '.png'
    # reading the image
    img = imread(image_path, as_gray=True)
    # normalizing the pixel values
    img /= 255.0
    # converting the type of pixel to float 32
    img = img.astype('float32')
    # appending the image into the list
    test_img.append(img)

```

Bootcamp

```
# converting the list to numpy array
test_x = np.array(test_img)
test_x.shape
```

.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/test_data.py)
test_data.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-test_data-py) hosted with ❤ by GitHub (https://github.com)

```
(10000, 28, 28)
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-45-27.png)

Now, we will do the pre-processing steps on these images similar to what we did for the training images earlier:

```
# converting training images into torch format
test_x = test_x.reshape(10000, 1, 28, 28)
test_x  = torch.from_numpy(test_x)
test_x.shape
```

lkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/test_torch_format.py)
test_torch_format.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-test_torch_format-py)
hosted with ❤ by GitHub (https://github.com)

```
torch.Size([10000, 1, 28, 28])
```

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-15-46-49.png)

Finally, we will generate predictions for the test set:

```
# generating predictions for test set
with torch.no_grad():
    output = model(test_x.cuda())

softmax = torch.exp(output).cpu()
prob = list(softmax.numpy())
predictions = np.argmax(prob, axis=1)
```

'PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/test_prediction.py)
test_prediction.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-test_prediction-py)
hosted with ❤ by GitHub (https://github.com)

Bootcamp

Replace the labels in the sample submission file with the predictions and finally save the file and submit it on the leaderboard:

```
# replacing the label with prediction
sample_submission['label'] = predictions
sample_submission.head()
```

om/PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/submission.py)
submission.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-submission-py) hosted with ❤️ by GitHub (https://github.com)

	id	label
0	60001	9
1	60002	2
2	60003	1
3	60004	1
4	60005	2

(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/09/Screenshot-from-2019-09-19-16-15-23.png)

```
# saving the file
sample_submission.to_csv('submission.csv', index=False)
```

PulkitS01/4b467252bbe0fbf520d91f3608e2284d/raw/57bacce3788993b8630996c1ec4a5a77364c9507/submission_file.py)
submission_file.py (https://gist.github.com/PulkitS01/4b467252bbe0fbf520d91f3608e2284d#file-submission_file-py)
hosted with ❤️ by GitHub (https://github.com)

You will see a file named *submission.csv* in your current directory. You just have to upload it on the solution checker of the problem page (https://datahack.analyticsvidhya.com/contest/practice-problem-identify-the-apparels/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch) which will generate the score.

Our CNN model gave us an accuracy of around 71% on the test set. That is quite an improvement on the 65% we got using a simple neural network in our previous article.

End Notes

In this article, we looked at how CNNs can be useful for extracting features from images. They helped us to improve the accuracy of our previous neural network model from 65% to 71% – a significant upgrade.

Bootcamp

You can play around with the hyperparameters of the CNN model and try to improve accuracy even further. Some of the hyperparameters to tune can be the number of convolutional layers, number of filters in each convolutional layer, number of epochs, number of dense layers, number of hidden units in each dense layer, etc.

In the next article (https://www.analyticsvidhya.com/blog/2019/10/how-to-master-transfer-learning-using-pytorch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch) of this series, we will learn how to use pre-trained models like VGG-16 and model checkpointing steps in PyTorch. And as always, if you have any doubts related to this article, feel free to post them in the comments section below!

You can also read this article on Analytics Vidhya's Android APP 

(//play.google.com/store/apps/details?id=com.analyticsvidhya.android&utm_source=blog_article&utm_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)

Share this:

(https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/?share=linkedin&nb=1)

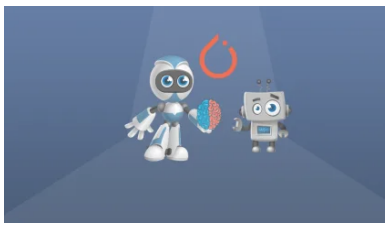
(https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/?share=facebook&nb=1)

(https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/?share=twitter&nb=1)

(https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/?share=pocket&nb=1)

(https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/?share=reddit&nb=1)

Related Articles



(https://www.analyticsvidhya.com/blog/2019/10/how-to-master-transfer-learning-using-pytorch/)
Deep Learning for Everyone: Master the Powerful Art of Transfer Learning using PyTorch

(https://www.analyticsvidhya.com/blog/2019/10/how-to-pytorch-from-scratch/)
A Beginner-Friendly Guide to PyTorch and How it Works from Scratch
(https://www.analyticsvidhya.com/b

(https://www.analyticsvidhya.com/blog/2019/09/4-tricks-improve-deep-learning-model-performance/)
4 Proven Tricks to Improve your Deep Learning Model's Performance

Bootcamp

(https://www.analyticsvidhya.com/blog/2019/10/how-to-master-transfer-learning-using-pytorch/)
October 22, 2019
In "Advanced"

log/2019/09/introduction-to-pytorch-from-scratch/)
September 17, 2019
In "Classification"

(https://www.analyticsvidhya.com/blog/2019/11/4-tricks-improve-deep-learning-model-performance/)
November 7, 2019
In "Advanced"

TAGS : CNN PYTORCH (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/CNN-PYTORCH/), CNNS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/CNNS/), CONVOLUTION NEURAL NETWORKS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/CONVOLUTION-NEURAL-NETWORKS/), IMAGE CLASSIFICATION (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/IMAGE-CLASSIFICATION/), NUMPY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NUMPY/), PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PYTHON/), PYTORCH (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PYTORCH/), TENSORS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TENSORS/)

NEXT ARTICLE

Add Shine to your Data Science Resume with these 8 Ambitious Projects on GitHub

(https://www.analyticsvidhya.com/blog/2019/10/8-ambitious-data-science-projects-github/)

•••

PREVIOUS ARTICLE

Become a Data Visualization Whiz with this Comprehensive Guide to Seaborn in Python

(https://www.analyticsvidhya.com/blog/2019/09/comprehensive-data-visualization-guide-seaborn-python/)

(https://www.analyticsvidhya.com/blog/author/pulkits/)

Bootcamp

Pulkit Sharma (Https://Www.Analyticsvidhya.Com/Blog/Author/Pulkits/)

My research interests lies in the field of Machine Learning and Deep Learning. Possess an enthusiasm for learning new skills and technologies.

24 COMMENTS

MANOJ (HTTP://WWW.COMPUTERSCIENCEJUNTION.IN) **Reply**

October 1, 2019 at 7:49 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159768)

Very Nice Article with proper coding and result explanation….!

PULKIT SHARMA **Reply**

October 3, 2019 at 11:01 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159780)

Glad you liked it Manoj!

DHRUVIT PATEL **Reply**

October 2, 2019 at 9:42 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159774)

This is a great Article. looking forward to see your next article.

PULKIT SHARMA **Reply**

October 3, 2019 at 11:03 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159782)

Hi Dhruvit,
Glad you liked it! I am currently working on the next article of this series and it will be out soon. I will inform you once it is live.

DHRUVIT PATEL **Reply**

October 2, 2019 at 9:55 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159775)

Bootcamp

I love this article. I would like to understand each of the libraries of torch.nn which you used in the building model, if you could share any documents then it would be better.

PULKIT SHARMA **Reply**

October 3, 2019 at 11:04 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159783)

Hi Dhruvit,
You can refer the following documentation to understand the nn module of torch:
https://pytorch.org/docs/stable/nn.html (https://pytorch.org/docs/stable/nn.html)

PAJEET **Reply**

October 12, 2019 at 8:31 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159862)

you should maybe explain what youre doing instead of just pasting a block of code, idiot. not all pictures are 28×28 grayscale

PULKIT SHARMA **Reply**

October 14, 2019 at 5:31 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159876)

Hi Pajeet,
I checked the data and found out that all the images are of shape 28*28. If you came across some image which is not of this shape, feel free to point out that. Also, I have tried my best to include comments in between the codes to simplify them. Even after looking at the comments, if you are unable to understand any line of code, feel free to ask it here and I will be happy to help.

MESAY SAMUEL **Reply**

October 16, 2019 at 6:25 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159910)

It is not clear for me how we get the score of test set. I am working with custom data set. Hence is that OK that I can get the score of test set in a way that we did for validation set? Thanks a lot and I really like your way of presenting things. Great work, can’t wait to see your next article.

PULKIT SHARMA **Reply**

Bootcamp

October 17, 2019 at 12:57 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159914)

Hi Mesay,
For the test set, we do not have the target variable and hence getting the score for the test set is not possible. Hence, in order to know how well our model will perform on the test set, we create a validation set and check the performance of the model on this validation set. If the validation score is high, generally we can infer that the model will perform well on test set as well.

MAJA Reply

October 22, 2019 at 5:49 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159954)

Great! Saves me !

PULKIT SHARMA Reply

October 22, 2019 at 5:27 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159961)

Glad you liked it!

JOSEPH Reply

October 23, 2019 at 6:56 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159974)

Hi Pulkit,
Thank you for posting this. I just had a quick question about defining the neural network architecture. If you were working with differently sized images (say, 500 x 500), what numbers would you have to change in the neural net class? Thank you.

PULKIT SHARMA Reply

October 25, 2019 at 12:59 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-159983)

Hi Joseph,
You have to make the changes in the code where we are defining the model architecture.

DSAM Reply

Bootcamp

November 13, 2019 at 10:17 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160157)

Hi Pulkit,
You effort is here is commendable. Just needed to know whether this code can be used for other images?

PULKIT SHARMA **Reply**

November 19, 2019 at 6:23 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160202)

Hi Dsam,
Yes! This code can be used for any image classification task.

MANIDEEP MITTAPALLI **Reply**

October 30, 2019 at 8:11 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160025)

Hi Pulkit,
Thanks for the wonderful blog, Can you explain how does the images size change through the convolutions conv1,conv2, with stride, padding, so that we can give the input image size to the fc?

PULKIT SHARMA **Reply**

November 1, 2019 at 11:31 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160041)

Hi Manideep,
Refer the following article where the output shapes have been explained after each layers, i.e. convolution, pooling, stride, etc.:
https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/ (https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/)

GEORGES **Reply**

November 17, 2019 at 3:37 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160177)

Hi Pulkit,

I am currently working on the CIFAR 10 database (with 50 000 32*32 RGB images), so the shape of my data is 50 000, 32, 32, 3.

Bootcamp

How should I change the shape of my data to make it work ? (Euclidean norm…?)
because I don't understand why you changed the shape of your data in the step "Creating a validation set and preprocessing the images" – you went from 5 400,28,28 to 5 400, 1, 28,28.

Thanks for the help

PULKIT SHARMA **Reply**

November 19, 2019 at 6:21 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160199)

Hi Georges,
PyTorch requires the input in some specific format. Here is the format that you have to use:
(sample_size, # of channel, width of image, height of image)
So, for your case it will be (50000, 3, 32, 32)

NEHA **Reply**

December 4, 2019 at 7:33 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160319)

Hi Pulkit,

First of all, Thank You! This and the previous article helped me understand the PyTorch framework.

While implementing the code, I came across an issue.

While running this code:
# defining the number of epochs
n_epochs = 25
# empty list to store training losses
train_losses = []
# empty list to store validation losses
val_losses = []
# training the model
for epoch in range(n_epochs):
train(epoch)

I got this error:
RuntimeError Traceback (most recent call last)
in

Bootcamp

```
7 # training the model
8 for epoch in range(n_epochs):
—-> 9 train(epoch)

in train(epoch)
8 # converting the data into GPU format
9 if torch.cuda.is_available():
—> 10 x_train = x_train.cuda()
11 y_train = y_train.cuda()
12 x_val = x_val.cuda()
```

RuntimeError: CUDA out of memory. Tried to allocate 162.00 MiB (GPU 0; 4.00 GiB total capacity; 2.94 GiB already allocated; 58.45 MiB free; 7.36 MiB cached)

How to solve this error?

PULKIT SHARMA **Reply**

December 5, 2019 at 11:20 am (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160329)

Hi Neha,
The error specifies that you need more RAM to run the codes. You can try these codes in google colab.

GÖKHAN **Reply**

December 14, 2019 at 6:00 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160409)

Hi,

I want to ask about train() function. In your code, you used model.train() for training. In some resources on the internet, they trained by using for loop. What is the differences between using model.train() and for loop? Does model.train() trains exactly or not? I am confused about this situation. If I use for loop and iterating for each batch, it takes almost 3-4 minutes to produce loss values on my dataset. But if I use model.train(), it takes only 1 second to produce loss values. can you explain this situation? I searched on the internet but I did not understand very well.

PULKIT SHARMA **Reply**

December 16, 2019 at 12:54 pm (https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/#comment-160420)

Bootcamp

Hi,
model.train() is for single epoch. I have also used a for loop to train the model for multiple epochs. If you just pass model.train() the model will be trained only for single epoch.

LEAVE A REPLY

Your email address will not be published.

Comment

Name (required)

Email (required)

Website

Notify me of new posts by email.

SUBMIT COMMENT

Bootcamp

RECOMMENDED READS

A Complete Python Tutorial to Learn Data Science from Scratch

(https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/?utm_source=building-image-classification-models-cnn-pytorch)

Bootcamp

Commonly used Machine Learning Algorithms (with Python and R Codes)

(https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/?utm_source=building-image-classification-models-cnn-pytorch)

7 Regression Techniques you should know!

(https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/?utm_source=building-image-classification-models-cnn-pytorch)

RECOMMENDED RESOURCES

Free Course for you

Comprehensive Learning path to become a data scientist

(https://courses.analyticsvidhya.com/courses/a-comprehensive-learning-path-to-become-a-data-scientist-in-2019/?utm_source=Recommendation_resource&utm_medium=blog&utm_campaign=Novlist)

Practice & Learn

Loan Prediction

(https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/?utm_source=Recommendation_resource&utm_medium=blog&utm_campaign=Novlist)

Try for free

Applied Machine Learning - Beginner to Professional

(https://courses.analyticsvidhya.com/courses/applied-machine-learning-beginner-to-professional/?utm_source=Recommendation_resource&utm_medium=blog&utm_campaign=Novlist)

Bootcamp

(https://www.analyticsvidhya.com/)

Download App (https://play.google.com/store/apps/details?id=com.analyticsvidhya.android) (https://apps.apple.com/us/app/analytics-vidhya/id1470025572)

Analytics Vidhya

About Us (https://www.analyticsvidhya.com/about-me/)

Our Team (https://www.analyticsvidhya.com/about-me/team/)

Careers (https://www.analyticsvidhya.com/about-me/career-analytics-vidhya/)

Contact us (https://www.analyticsvidhya.com/contact/)

Data Science

Blog (https://www.analyticsvidhya.com/blog/)

Hackathon (https://datahack.analyticsvidhya.com/)

Discussions (https://discuss.analyticsvidhya.com/)

Apply Jobs (https://www.analyticsvidhya.com/jobs/)

Companies

Post Jobs (https://www.analyticsvidhya.com/corporate/)

Trainings (https://courses.analyticsvidhya.com/)

Hiring Hackathons (https://datahack.analyticsvidhya.com/)

Advertising (https://www.analyticsvidhya.com/contact/)

Visit us

(https://www.linkedin.com/company/analytics-vidhya/) (https://www.facebook.com/analyticsvidhya) (https://twitter.com/analyticsvidhya) (https://www.youtube.com/channel/UCcIyqDteHtH4hg3o2343iObA)

Privacy Policy Terms of Use Refund Policy

Bootcamp