# Note_1_PreTrainedModel

April 19, 2022

# Introduction to PyTorch's AlexNet model

AlexNet is one of the architectures for classification task and may have huge applications in the computer vision sector of artificial intelligence problems. AlexNet was trained with ImageNet data set which has 14 million high-resolution images spread across 22,000 categories. In this notebook, we tested AlexNet with the MNIST dataset and observed its prediction

# Module Imports

Module is a part of a python package. Packages are available in various common sources like pypi. Using import statements, we can make the code in one module available in other modules.

PyTorch offers domain-specific libraries such as TorchText, TorchVision, and TorchAudio, all of which include datasets. The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities. To use our prefered module in program, we need to install it in our local machine using pip or conda installer.

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.

In our module import section, we have imported models, transforms, and datasets from torchvision package. Where models subpackage contains definitions for the different model (AlexNet,ResNet,GoogLeNet etc) architectures for image classification. Transforms are common image transformations availabel in torch vision to convert PIL image to tensor. Instead of building the datasets from the scratch, we can utilise some built-in datasets based on our needs and Torchvision helps us by providing many built-in datasets in the torchvision.datasets module.

So all these imports can be loaded like shown in the following code.

```
[ ]: import torch
     from torchvision import models          # Models are subpackage contains␣
      ↪definitions for the different model architectures for image classification
     from torchvision import transforms      # Transforms are common image transforms
     from PIL import Image                   # PIL is the Python Imaging Library␣
      ↪which provides the python interpreter with image editing capabilities.
     import requests                         # Requests module allows you to send␣
      ↪HTTP requests using Python.

     from torchvision import datasets
```

# Data and Model Download

Data can be any unprocessed fact, value, text, sound, or picture that is not being interpreted and analysed. A "model" in machine learning is the output of a machine learning algorithm run on data. A model represents what was learned by a machine learning algorithm. Each model will have its own architecture. The AlexNet model used in our code has eight layers with learnable parameters.

Model has weights and biases each input is multiplied by a weight to indicate the input's relative importance. The sum of the weighted input(s) is fed into the neuron. Bias is added to the sum of the weighted inputs. An activation function within the neuron performs a calculation on the total.

Pre-trained mode is a model created by someone else to solve a similar problem. Instead of building a model from scratch to solve a similar problem, you use the model trained on other problem as a starting point. Some of the popular pre-trained image classification are AlexNet, ResNet, and GoogleLeNet. We can also create our own model, by manually defining the each of the required layers.

To utilize the pre-trained feature of a model in pytorch, we need to pass the pretrained parameter as True in the model initialization step like shown in the below code snippet.

```
[ ]: AlexNet = models.alexnet(pretrained=True)      # Utilizing pre-trained alexnet␣
      ↪model from PyTorch
```

Just call the object name like in the below snippet, to view the details of each layers in the alexnet architect.

```
[ ]: AlexNet
```

```
[ ]: AlexNet(
       (features): Sequential(
         (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
         (1): ReLU(inplace=True)
         (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=False)
         (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
         (4): ReLU(inplace=True)
         (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=False)
         (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
         (7): ReLU(inplace=True)
         (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
         (9): ReLU(inplace=True)
         (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
         (11): ReLU(inplace=True)
         (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=False)
       )
       (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
       (classifier): Sequential(
         (0): Dropout(p=0.5, inplace=False)
```

```
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In Pytorch, we can set the model under two modes (Train mode and Evaluation mode). By default all the modules are initialized to train mode. model.train() sets the modules in the network in training mode. It tells our model that we are currently in the training phase so the model keeps some layers, like dropout, batch-normalization which behaves differently depends on the current phase, active. whereas the model.eval() does the opposite. Therefore, once the model.eval() has been called then, our model deactivate such layers so that the model outputs its inference as is expected. Since we are not going to perform any training here, we have directly called the evaluation mode.

[ ]:
```
AlexNet.eval()                                    # options which will help our model␣
↪to understand we are in evaluation mode. Also, Equivalent to model.
↪train(False).
                                                  # Eval will switch off the␣
↪requires_grad and dropout and batch normalization
```

[ ]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
```

```
      (2): ReLU(inplace=True)
      (3): Dropout(p=0.5, inplace=False)
      (4): Linear(in_features=4096, out_features=4096, bias=True)
      (5): ReLU(inplace=True)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
```

Alexnet is trained with ImageNet dataset and the Imagenet has the class of 1000 categories. During our evaluation process, alexnet will map our input image into one of these 1000 categories. So, in the below snippet, we fetch the labels from the URL using an HTTP request and store them in a variable called labels.

```
[ ]: url1 ="https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.
      ↪txt"
     labels = requests.get(url1).text.split('\n')

     print(len(labels))
```

```
1000
```

# Input Data

Model expect the input to be in a tensor batch. mini-batches of 3-channel RGB images of shape (C x H x W), where C is channels, H and W are expected to be at least 224. Our expected batch shape should be [B x C x H x W]. where b is the size of the batch. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]. This mean and standard deviation has been derived from the imagenet dataset and is used here because pretrained model is trained on these data.

Model can understand only numeric values. If we have some non numerical data, then we need to convert it into numeric representation through some kind of transformation.

```
[ ]: transform = transforms.Compose([      # Compose class Composes several␣
      ↪transforms together.
         transforms.Resize(256),           # Resize the input image to the given size.
         transforms.CenterCrop(224),       # Crops the given image at the center.
         transforms.ToTensor(),            # Convert a PIL Image or numpy.ndarray to␣
      ↪tensor.
         transforms.Normalize(
             mean=[0.485, 0.456, 0.406],
             std=[0.229, 0.224, 0.225])
             ])
```

In the below snippet, a sample strawberry image is chosen and loaded in a variable called img.

```
[ ]: #strwaberry
     url ="https://learnopencv.com/wp-content/uploads/2021/01/strawberries.jpg"
```

```
img = Image.open(requests.get(url,stream=True).raw)      # Performing http␣
 ↪request to fetch the image.

img
```

[ ]:

Then our img is passed into our transform object to achieve the required tensor and later this transformed_image object will be used as the input of our predicImage method.

```
[ ]: transformed_image = transform(img)                               #␣
      ↪Transforming our image into the required format.
```

# Output Prediction

Output of the model will be a confidence score over the classes. For example, in AlexNet output will be confidence scores over Imagenet's 1000 classes.

First step of the prediction process is to create a batch. Then we need to pass the batch to our alexnet model. Now the model will return a vector of number and we stored that in a variable called output as shown in the below snippet. Followed by that we need to convert this vetcor into probabilities, using softmax function in pyTorch, we can easily converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

Finally, with the help of torch.sort() function we sorted the maximum value of all elements in the output tensor in descending order. Then using a simple for loop, we iterated through the first five class labels and probabilities and returned a variable called top5. By printing it we will get the top 5 predictions of our model.

```
[ ]: def predictImage(tImage, model):                          # User defined␣
      ↪function with two inputs. Tensor image and Model type.
        bImage = torch.unsqueeze(tImage,0)                       # Matching the size␣
      ↪of our input with the trained set input size. Also for batch processing.
        #bImage = tImage.unsqueeze(0)
        output = model(bImage)
        print((output.size()))
        percentage = torch.nn.functional.softmax(output,dim =1)[0]*100    # Softmax␣
      ↪operation is applied to all slices of input along the specified dim, and␣
      ↪will rescale them so that the elements lie in the range (0, 1) and sum to 1.␣
      ↪
        _, ids = torch.sort(output, descending=True)                  # torch.
      ↪sort return the maximum value of all elements in the input tensor in␣
      ↪desceding order
        top5 = [(labels[idx],percentage[idx].item()) for idx in ids[0][:5] ]
        print(top5)
```

```
[ ]: predictImage(transformed_image, AlexNet) # Calling the predictImage method and␣
      ↪passing the url and model type. Expecting to print the top 5 matching␣
      ↪results.
```

```
torch.Size([1, 1000])
[('strawberry', 99.99411010742188), ('banana', 0.0008383141248486936), ('custard
apple', 0.0008333742734976113), ('orange', 0.0007468060939572752), ('lemon',
0.0005674778367392719)]
```

# Testing alexnet with MNIST dataset

Alexnet accept it input only in the form of tensor, so we need to transform the MNIST dataset into tensor. In the next step, we are downloading the MNIST data set into our local direcorty. Finally, we are passing a single MNIST data into our predictImage method, which will return the top 5 predections of AlexNet. MNIST dataset is in one dimensional, using transform we are creating 2 more dimensions by just copying the same dimension. This is required because the alexnet requires an RGB image which is 3 dimensional.

```
[ ]: MNIST_transform = transforms.Compose([
         transforms.Resize(224),
         transforms.ToTensor(),
         transforms.Lambda(lambda tensor:tensor.repeat(3,1,1)),
         transforms.Normalize(
             mean=[0.485, 0.456, 0.406],
             std=[0.229, 0.224, 0.225])
             ])

     training_data = datasets.MNIST(
         root="data",
         train=True,
         download=True,
         transform=MNIST_transform,
     )
```

First input from our training dataset is fetched and the corresponding numeric value of the data is 5. Then the tensor relvant to the dataset is passed into our predict Image function.

[ ]:
```
m_img, num = training_data[0]
num
```

[ ]:
```
5
```

[ ]:
```
predictImage(m_img,AlexNet)
```

```
torch.Size([1, 1000])
[('nematode', 20.916667938232422), ('can opener', 9.996562004089355), ('letter
opener', 9.48585319519043), ('safety pin', 5.598158836364746), ('cleaver',
5.529609680175781)]
```

Since, we haven't trained the model, with MNIST dataset. Our model predicted it as nematode.