Implementing VGG16 from Scratch

VGG16 is a convolutional neural network from paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

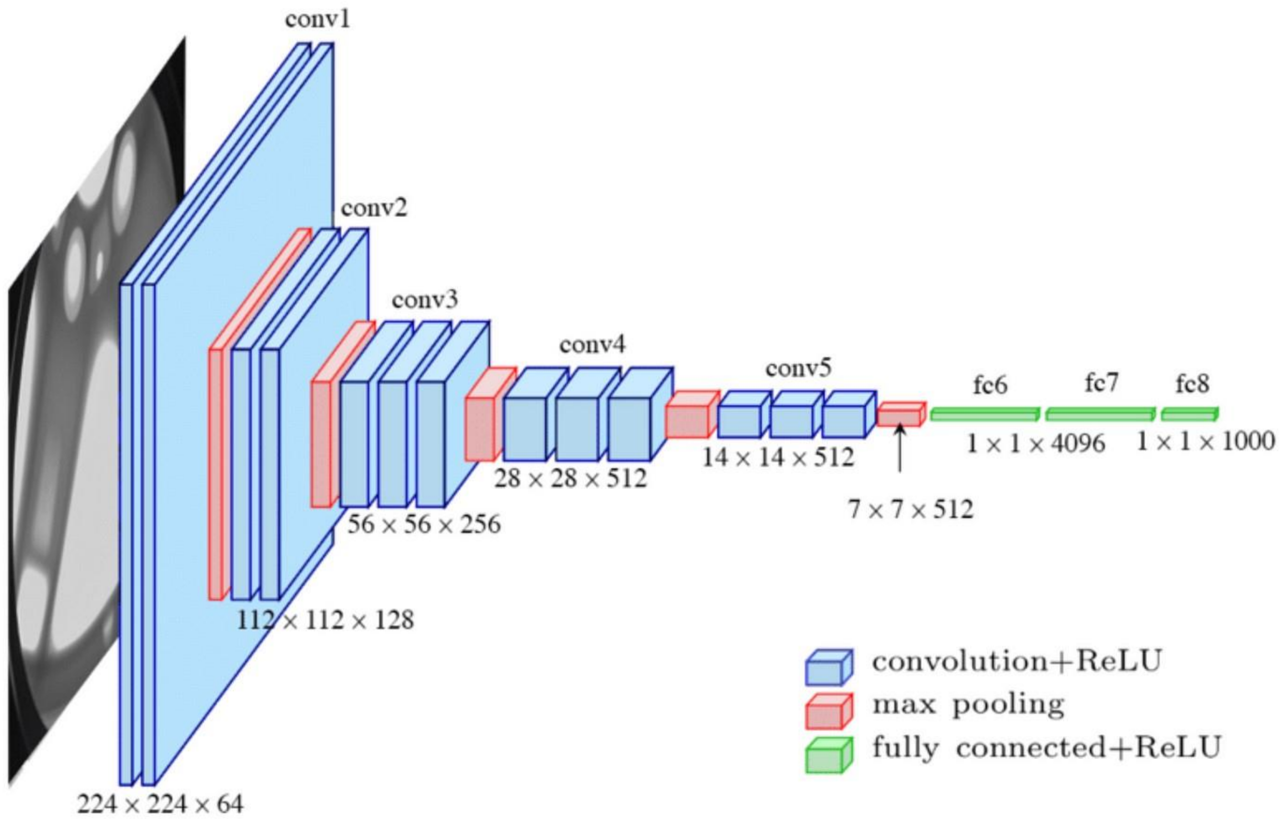
Overview

VGG16 was one of the famous models submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

Architecture

VGG16 includes 13 convolutional layers, 3 fully connected layers, and a softmax layer. There are also 5 max pool layers each of which is followed by a series of convolutional layers. The number 16 in VGG16 refers to the total number of layers that have weights.

The overall architecture can is visualized below.



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Implementation

The VGG16 model was implemented using PyTorch. Initially a list of the architecture features and states were created and thereafter the model was created by iteratively implementing through these using torch.nn.Module.

Architectural Features:
KERNEL = 3
STRIDE = 2 # For max pooling
CHANNEL = [3,64, 64, 128, 128, 256, 256, 512, 512, 512, 512, 512, 512]
FC = [51277, 4096, 4096]
POOL_POS = [2,4,6,9,12]

A custom Relu activation function layer was implemented and used instead of using the torch.nn.ReLU.

Finally the model was saved with weights for inferencing.

Dataset and Training

Dataset:

CIFAR-10 Dataset and its subset was used for training the network. It contains 10 classes, 50000 training images and 10000 validation images each of size 32*32.

Training Log

The training log keeps track of the training loss and accuracy of the model over epochs. Here is the snippets of part of training log.

```
Epoch 14/15
----------
train Loss: 0.0349 Acc: 0.9920
val Loss: 0.0222 Acc: 0.9980
Epoch 15/15
----------
train Loss: 0.0217 Acc: 0.9952
val Loss: 0.0118 Acc: 0.9980
Training complete in 19m 37s
Best val Acc: 0.998000
```

Inference/Test result

```python
from PIL import Image
import requests
img = Image.open(requests.get('https://t3.ftcdn.net/jpg/00/01/47/28/360_F_1472821_kMjcU0El8NkcU0k7zNtlVTU
img
```

```python
img = data_transforms(img)
img = torch.unsqueeze(img, 0)
img = img.to(device)
pred = torch.argmax(model(img))
print(class_names[pred.item()])
```

```
airplane
```

```
(ml) PS X:\LogicTronix\Network Implementation From Scratch\VGG16> python -u "x:\LogicTronix\Network Implementation From Scratch\VGG16\Scripts\evaluate.py" '.\model\custom_model.pth' '.\data\test_images\im
g.jpg'
X:\Softwares\anaconda\envs\ml\lib\site-packages\torchvision\io\image.py:13: UserWarning: Failed to load image Python extension: '[WinError 127] The specified procedure could not be found'If you don't plan
 on using image functionality from `torchvision.io`, you can ignore this warning. Otherwise, there might be something wrong with your environment. Did you have `libjpeg` or `libpng` installed before build
ing `torchvision` from source?
  warn(
Inferencing the model
airplane
(ml) PS X:\LogicTronix\Network Implementation From Scratch\VGG16>
```

Issues Faced

Issue1: Model couldn't go beyond 10% in accuracy.

Reason: The learning rate was high (0.1, 0.01)

Solution: It was reduced to 0.001 and the model started to converge.

Issue2: In custom implementation of ReLU the layer need to be instance of nn.Module.

Solution: Inherit nn.Module

Summary of the work

This work involves implementing the VGG16 model from scratch, training it on the CIFAR-10 dataset, and testing its performance. Thereafter involves refactoring code for production into python modules.

References

1. K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in arXiv preprint arXiv:1409.1556, 2014.
2. CIFAR-10, https://www.cs.toronto.edu/~kriz/cifar.html .
3. PyTorch Documentation, https://pytorch.org/docs/stable/index.html

Notes:

- If we want to train with different image size, the globalaverage pooling should be added before passing to FC layers to make match the weight matrices.