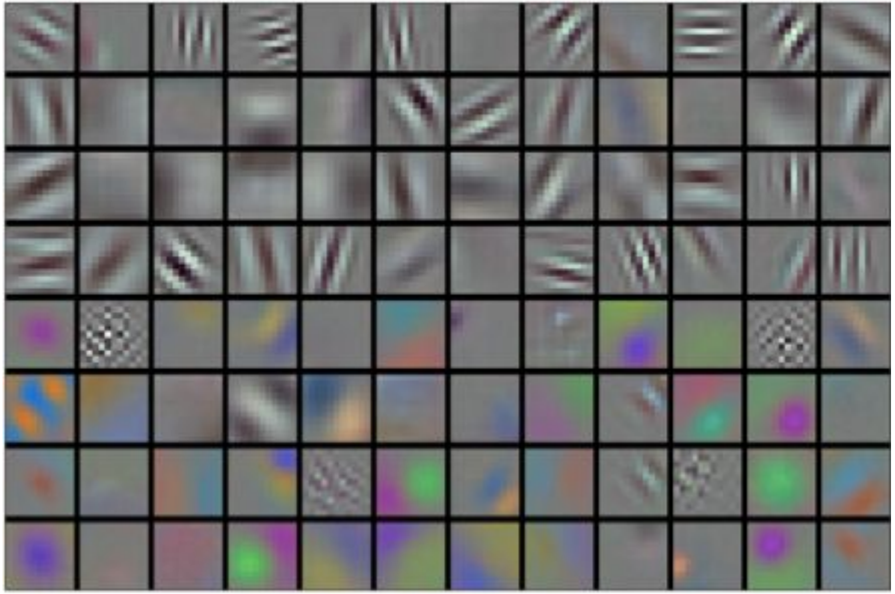
AlexNet and VGG

Chapter 7.1, 7.2 of D2L

Presented by Paul Rogozenski

The age before CNNs

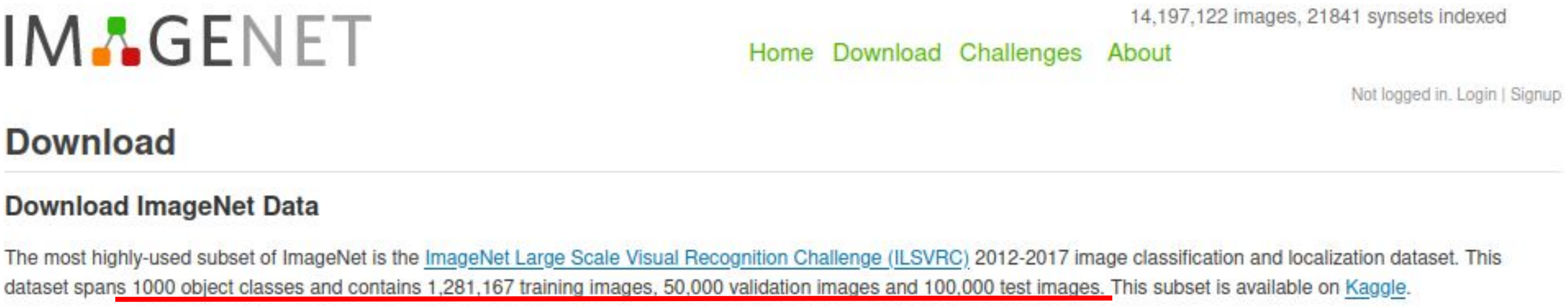
- Features were manually calculated and implemented
  - Speedy code, but networks were not *learning*, but rather looking for predetermined features
  - Examples: SIFT [Lowe, 2004], SURF [Bay et al., 2006], HOG [Dalal & Triggs, 2005]
- Data was not as abundant as today
- Hardware choice typically CPUs



Filters as determined by first layer of AlexNet

Brief introduction to AlexNet and VGG

- AlexNet and VGG both deep convolutional neural networks aiming to classify the ImageNet dataset
- AlexNet was developed in 2012 by researchers at UToronto with 8 layers
  - First working example of a deep convolutional network utilizing GPUs
- VGG was developed in 2014 by researchers at Oxford with 11+ layers


IMAGENET

14,197,122 images, 21841 synsets indexed

Home Download Challenges About

Not logged in. Login | Signup

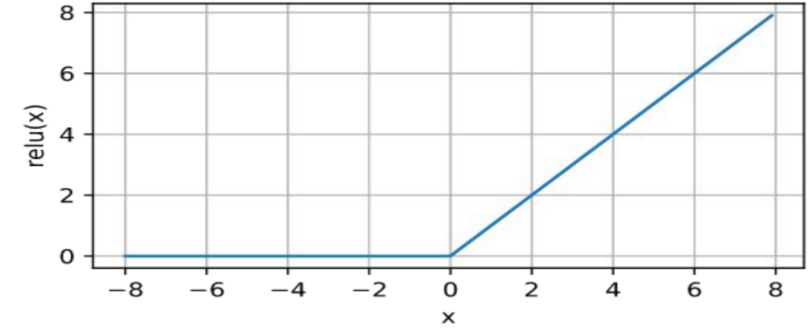
Download

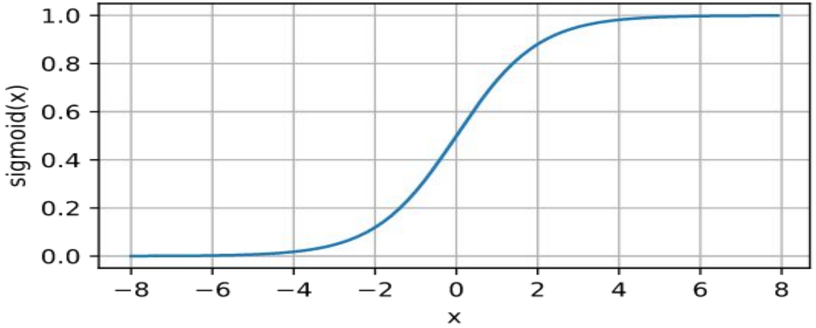
Download ImageNet Data

The most highly-used subset of ImageNet is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012-2017 image classification and localization dataset. This dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. This subset is available on Kaggle.

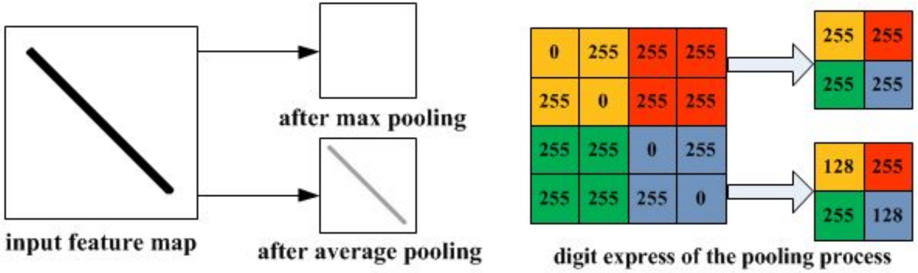
Review of Common ML Methods

Activation Functions

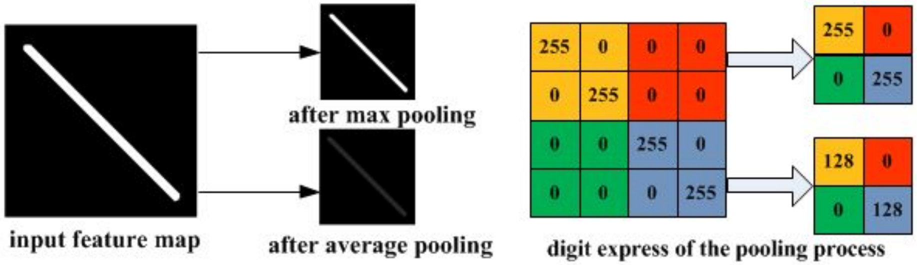




Pooling



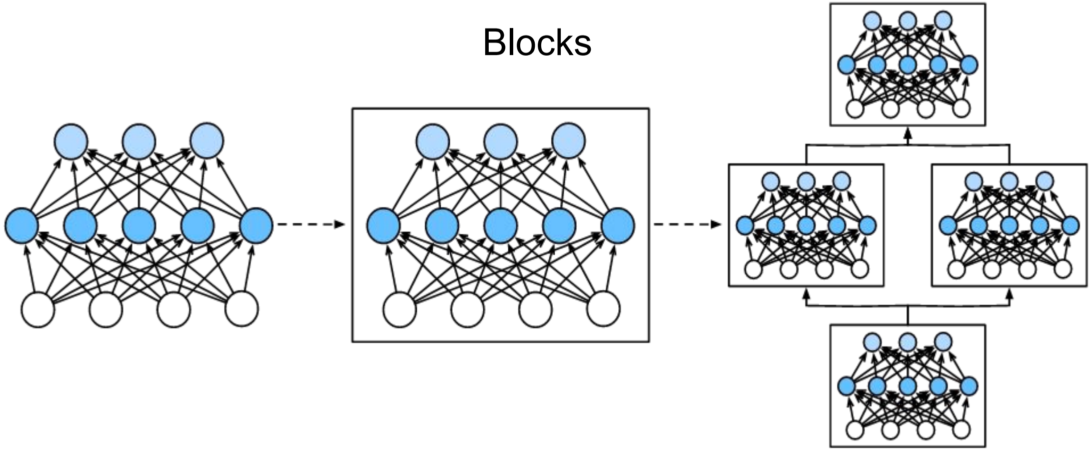
(a) Illustration of max pooling drawback



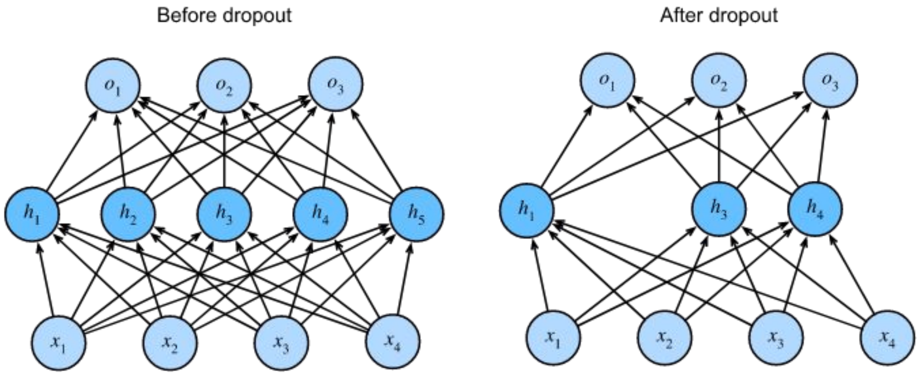
(b) Illustration of average pooling drawback

Review of Common ML Methods

Blocks

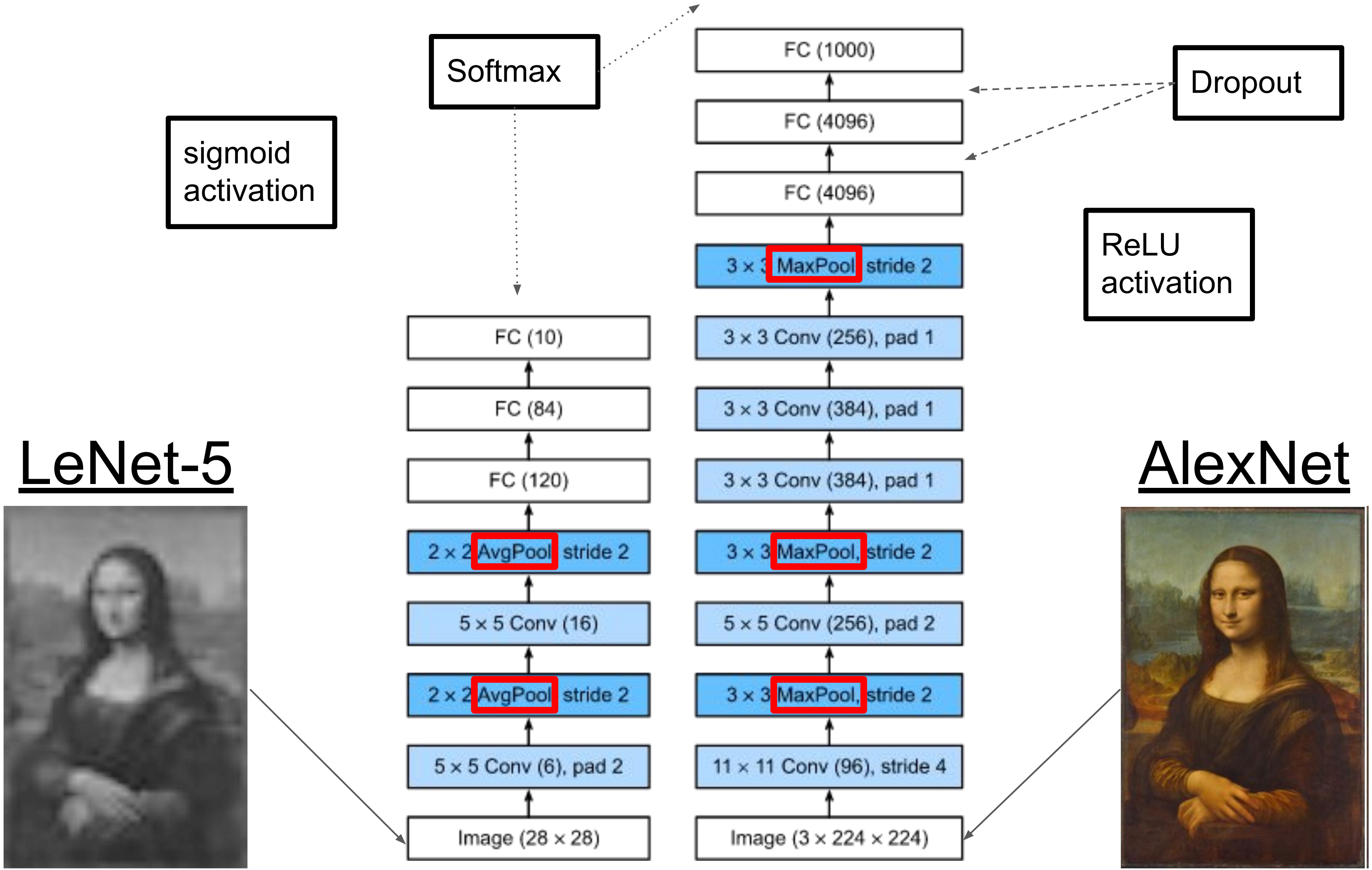


Dropout



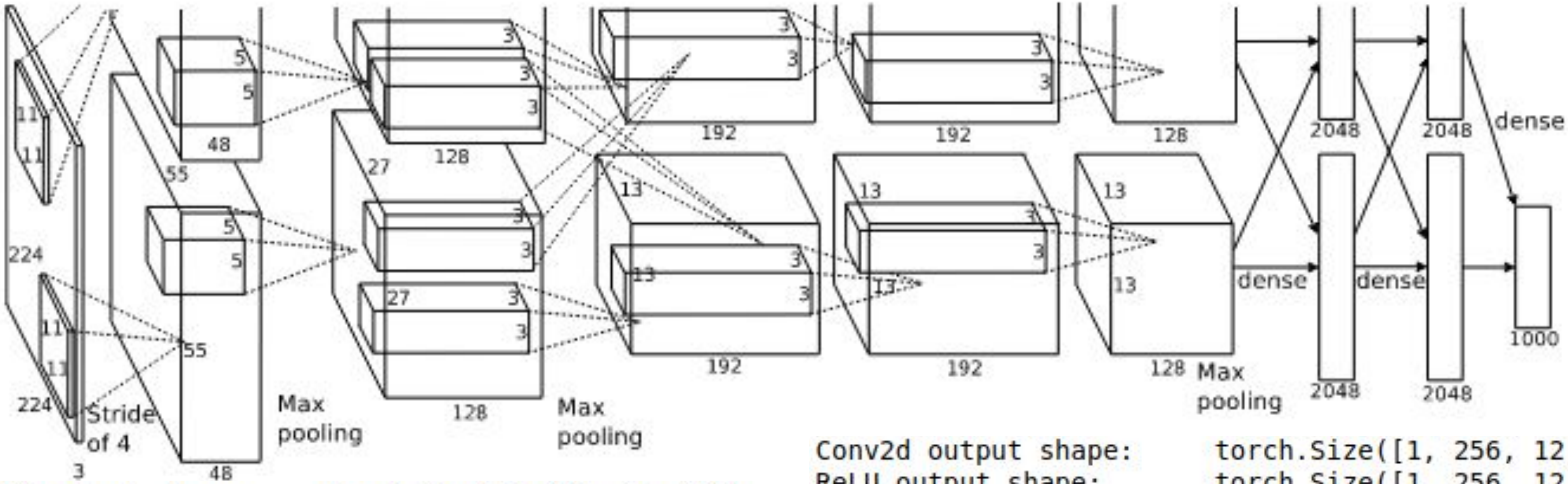
Softmax regression

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$



LeNet-5
AlexNet
sigmoid activation
Softmax
Dropout
ReLU activation
FC (10)
FC (84)
FC (120)
2 × 2 AvgPool, stride 2
5 × 5 Conv (16)
2 × 2 AvgPool, stride 2
5 × 5 Conv (6), pad 2
Image (28 × 28)
FC (1000)
FC (4096)
FC (4096)
3 × 3 MaxPool, stride 2
3 × 3 Conv (256), pad 1
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 MaxPool, stride 2
5 × 5 Conv (256), pad 2
3 × 3 MaxPool, stride 2
11 × 11 Conv (96), stride 4
Image (3 × 224 × 224)

AlexNet Visualized (different than D2L version at the bottom)




```
Conv2d output shape:     torch.Size([1, 96, 54, 54])
ReLU output shape:       torch.Size([1, 96, 54, 54])
MaxPool2d output shape:  torch.Size([1, 96, 26, 26])
Conv2d output shape:     torch.Size([1, 256, 26, 26])
ReLU output shape:       torch.Size([1, 256, 26, 26])
MaxPool2d output shape:  torch.Size([1, 256, 12, 12])
Conv2d output shape:     torch.Size([1, 384, 12, 12])
ReLU output shape:       torch.Size([1, 384, 12, 12])
Conv2d output shape:     torch.Size([1, 384, 12, 12])
ReLU output shape:       torch.Size([1, 384, 12, 12])
```

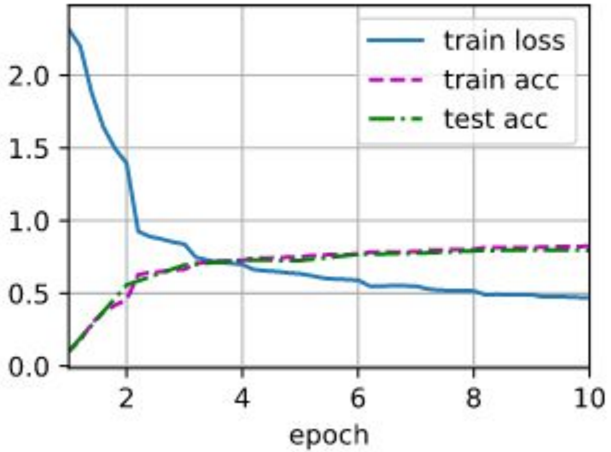
```
Conv2d output shape:     torch.Size([1, 256, 12, 12])
ReLU output shape:       torch.Size([1, 256, 12, 12])
MaxPool2d output shape:  torch.Size([1, 256, 5, 5])
Flatten output shape:    torch.Size([1, 6400])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 10])
```

LeNet vs. AlexNet on MNIST dataset

Using the original AlexNet framework, we need to UPSAMPLE MNIST images and decrease the number of categories

<u>LeNet-5</u>

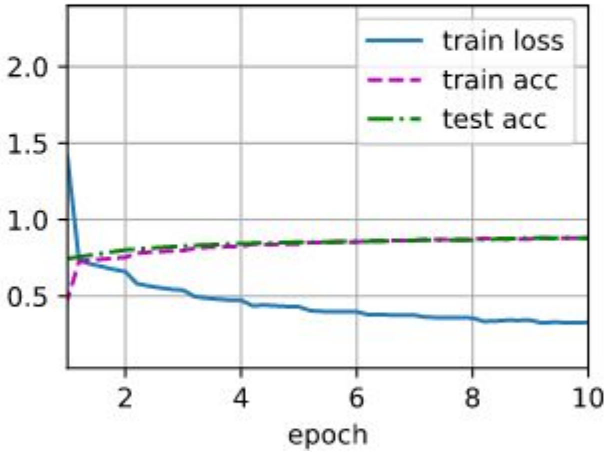
```
loss 0.470, train acc 0.824, test acc 0.796
103351.4 examples/sec on cuda:0
```



lr = 0.9, batch = 256

<u>AlexNet</u>

```
loss 0.329, train acc 0.880, test acc 0.879
1522.0 examples/sec on cuda:0
```



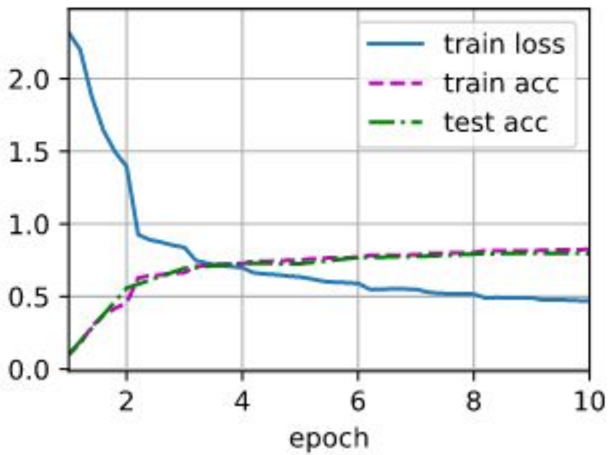
lr = 0.01, batch = 128

LeNet vs. AlexNet on MNIST dataset

WITHOUT upsampling (CNN parameters not optimized in AlexNet)

<u>LeNet-5</u>

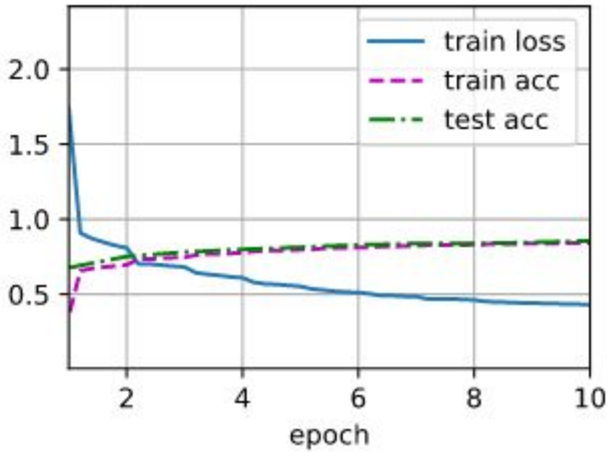
```
loss 0.470, train acc 0.824, test acc 0.796
103351.4 examples/sec on cuda:0
```



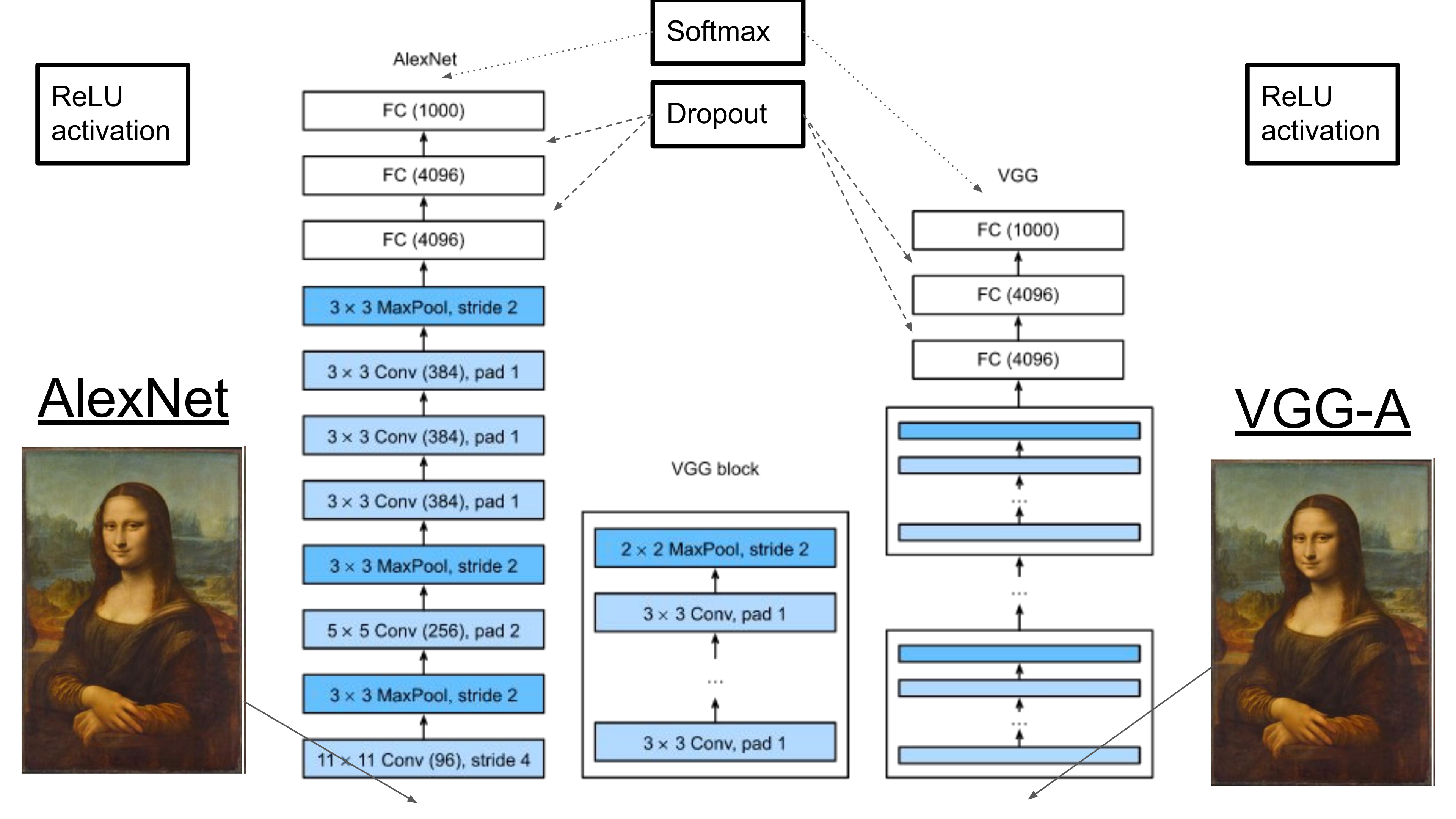
lr = 0.9, batch = 256

<u>Modified AlexNet</u>

```
loss 0.429, train acc 0.843, test acc 0.856
41446.7 examples/sec on cuda:0
```

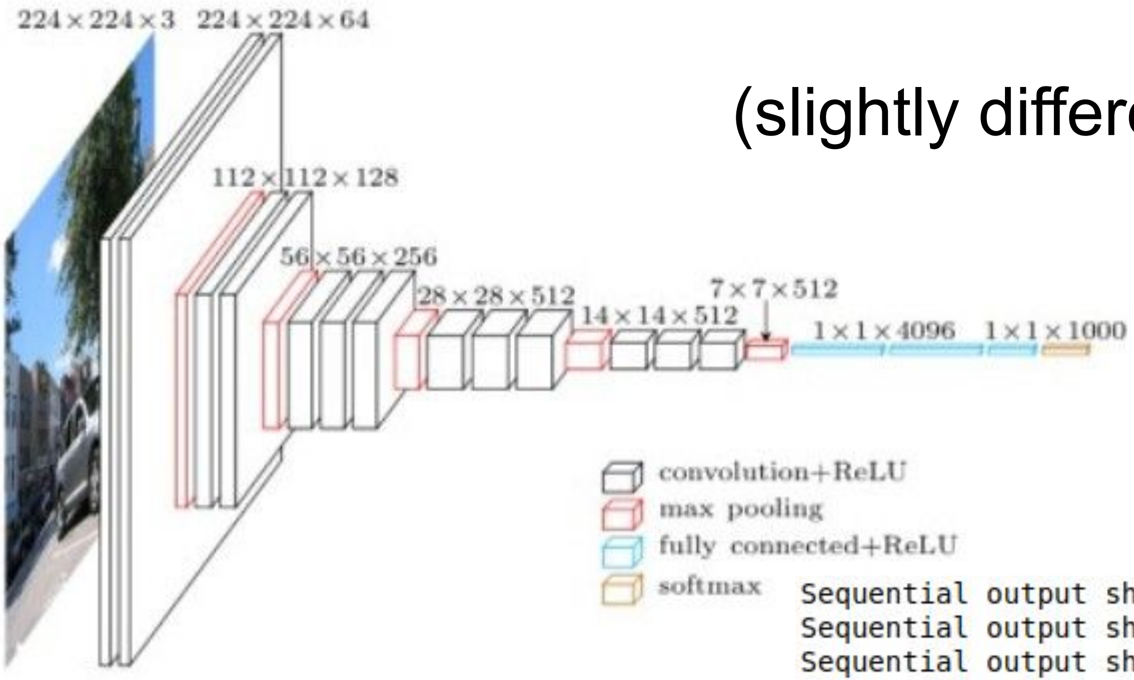


lr = 0.02, batch = 128



Softmax
Dropout
ReLU activation
ReLU activation
AlexNet
FC (1000)
FC (4096)
FC (4096)
3 × 3 MaxPool, stride 2
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 MaxPool, stride 2
5 × 5 Conv (256), pad 2
3 × 3 MaxPool, stride 2
11 × 11 Conv (96), stride 4
AlexNet
VGG block
2 × 2 MaxPool, stride 2
3 × 3 Conv, pad 1
...
3 × 3 Conv, pad 1
VGG
FC (1000)
FC (4096)
FC (4096)
VGG-A

VGG-A Visualized
(slightly different than D2L version)




```
Sequential output shape:         torch.Size([1, 64, 112, 112])
Sequential output shape:         torch.Size([1, 128, 56, 56])
Sequential output shape:         torch.Size([1, 256, 28, 28])
Sequential output shape:         torch.Size([1, 512, 14, 14])
Sequential output shape:         torch.Size([1, 512, 7, 7])
Flatten output shape:    torch.Size([1, 25088])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 4096])
ReLU output shape:       torch.Size([1, 4096])
Dropout output shape:    torch.Size([1, 4096])
Linear output shape:     torch.Size([1, 10])
```

A closer look at the VGG block

```
import torch
from torch import nn
from d2l import torch as d2l


def vgg_block(num_convs, in_channels, out_channels):
    layers = []
    for _ in range(num_convs):
        layers.append(nn.Conv2d(in_channels, out_channels,
                                kernel_size=3, padding=1))
        layers.append(nn.ReLU())
        in_channels = out_channels
    layers.append(nn.MaxPool2d(kernel_size=2,stride=2))
    return nn.Sequential(*layers)
```

- Each block performs a number of convolutions, followed by a ReLU activation, returning the convolutions with a MaxPool kernel
- After each block, the resolution is halved and the number of output channels is doubled
- Allows for a more modular CNN that could be tuned to different problems

A closer look at the VGG network

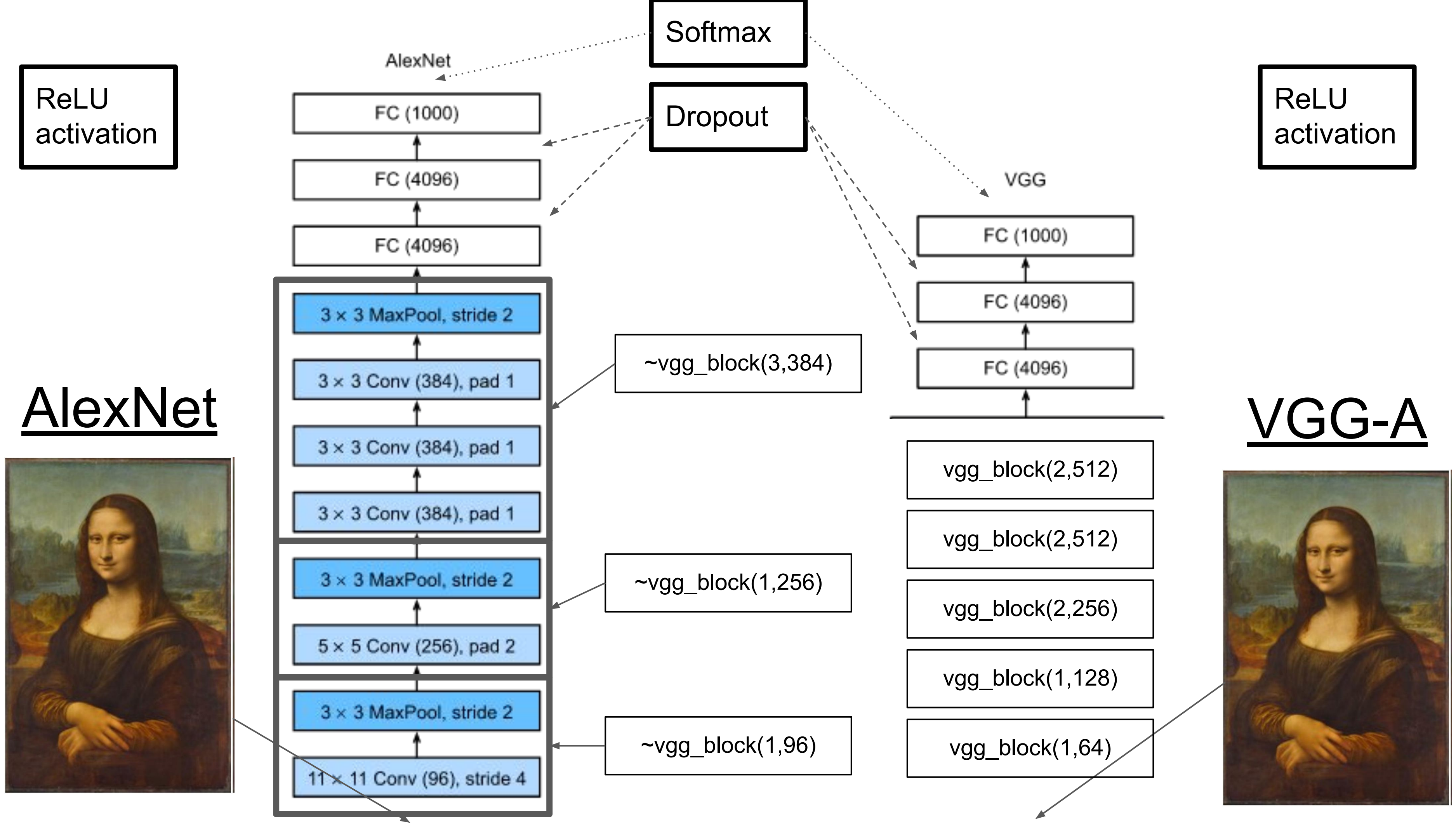
```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```

The following code implements VGG-11. This is a simple matter of executing a for-loop over `conv_arch`.

```
def vgg(conv_arch):
    conv_blks = []
    in_channels = 1
    # The convolutional part
    for (num_convs, out_channels) in conv_arch:
        conv_blks.append(vgg_block(num_convs, in_channels, out_channels))
        in_channels = out_channels

    return nn.Sequential(
        *conv_blks, nn.Flatten(),
        # The fully-connected part
        nn.Linear(out_channels * 7 * 7, 4096), nn.ReLU(), nn.Dropout(0.5),
        nn.Linear(4096, 4096), nn.ReLU(), nn.Dropout(0.5),
        nn.Linear(4096, 10))

net = vgg(conv_arch)
```



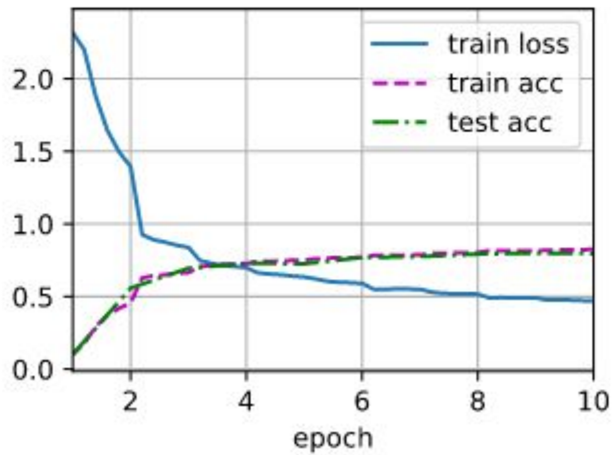
ReLU activation
Softmax
Dropout
ReLU activation
AlexNet
FC (1000)
FC (4096)
FC (4096)
3 × 3 MaxPool, stride 2
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 Conv (384), pad 1
3 × 3 MaxPool, stride 2
5 × 5 Conv (256), pad 2
3 × 3 MaxPool, stride 2
11 × 11 Conv (96), stride 4
~vgg_block(3,384)
~vgg_block(1,256)
~vgg_block(1,96)
VGG
FC (1000)
FC (4096)
FC (4096)
vgg_block(2,512)
vgg_block(2,512)
vgg_block(2,256)
vgg_block(1,128)
vgg_block(1,64)
AlexNet
VGG-A

LeNet vs. VGG on MNIST dataset

Using the original VGG framework, we need to UPSAMPLE MNIST images and decrease the number of categories

LeNet-5

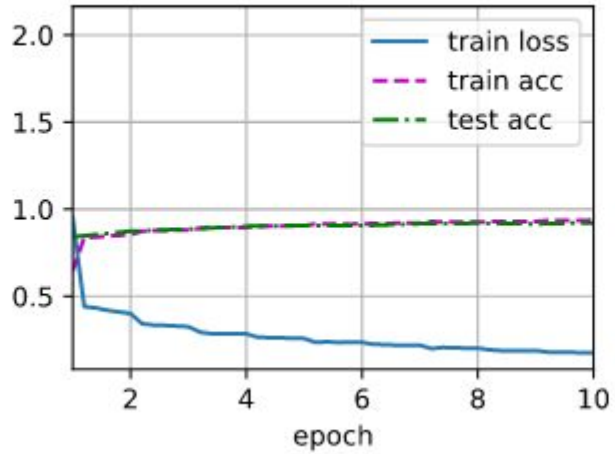
```
loss 0.470, train acc 0.824, test acc 0.796
103351.4 examples/sec on cuda:0
```



lr = 0.9, batch = 256

VGG

```
loss 0.175, train acc 0.937, test acc 0.922
649.8 examples/sec on cuda:0
```



lr = 0.01, batch = 128

Takeaways of AlexNet vs. VGG

- VGG and AlexNet were some of the first CNNs to utilize GPU architecture and apply it to the ImageNet dataset, making CNNs computationally feasible.
- Some similarities in the network structure exist when comparing to their precursor, LeNet, but with some advances in ML methodology.
- VGG is more computationally expensive than AlexNet, but yields significantly better performance. Both VGG and AlexNet perform better than LeNet.

Network	Loss	Training acc.	Test acc.	examples/sec	Trainable parameters
LeNet-5	0.470	0.824	0.796	~103351	~60,000
AlexNet	0.329	0.880	0.879	~1522	~47,000,000
VGG-A	0.175	0.937	0.922	~650	~130,000,000

References

- https://d2l.ai/chapter_convolutional-modern/
- https://arxiv.org/pdf/1409.1556.pdf
- https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- https://stackoverflow.com/questions/62856035/can-browsers-produce-different-resolutions-of-an-image (mona lisa image)
- https://www.researchgate.net/figure/Toy-example-illustrating-the-drawbacks-of-max-pooling-and-average-pooling_fig2_300020038 (max vs avg pool image)
- https://debuggercafe.com/implementing-vgg11-from-scratch-using-pytorch/ (VGG-11 network visual)