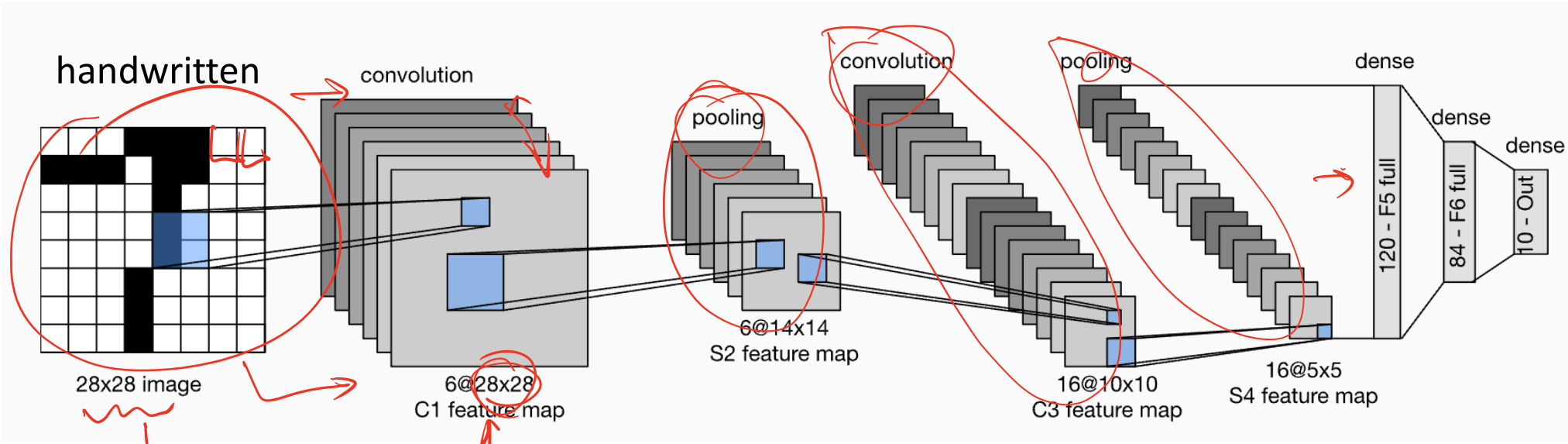
UNIVERSITY of WEST FLORIDA

Week 5: Applications of CNN

Shusen Pu

UNIVERSITY of WEST FLORIDA

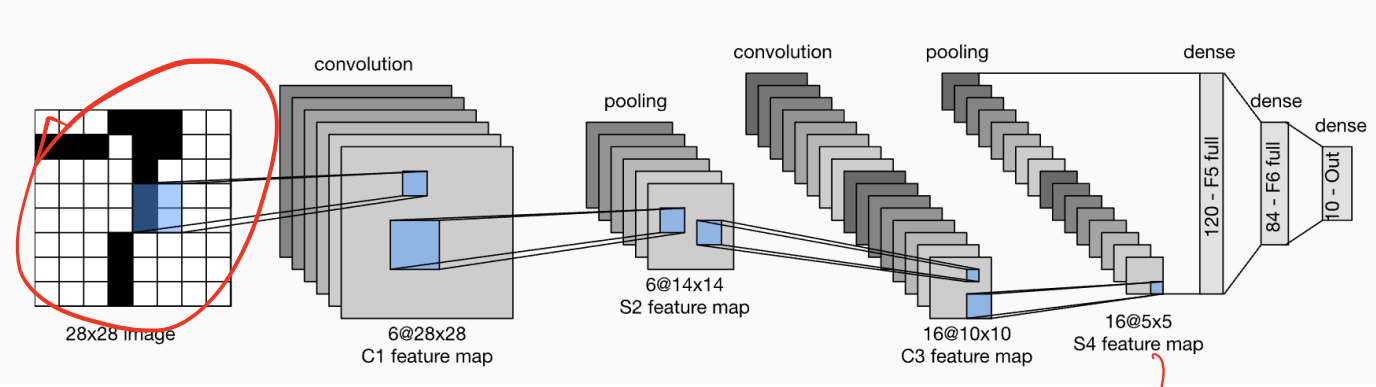
LeNet

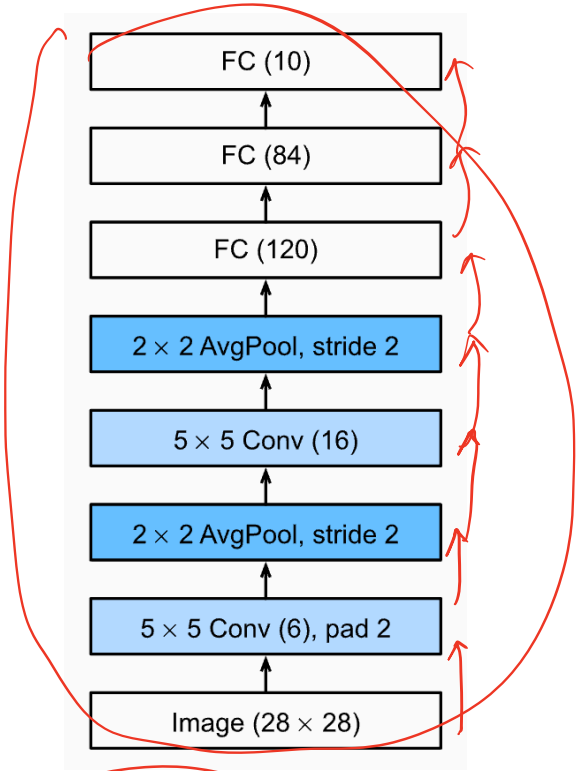


- LeNet was among the first published CNNs to capture wide attention for its performance on computer vision tasks. The model was introduced by (and named for) Yann LeCun, then a researcher at AT&T Bell Labs, for the purpose of recognizing handwritten digits in images.
- This work represented the culmination of a decade of research developing the technology. In 1989, LeCun's team published the first study to successfully train CNNs via backpropagation
- At the time LeNet achieved outstanding results matching the performance of support vector machines, then a dominant approach in supervised learning, achieving an error rate of less than 1% per digit. LeNet was eventually adapted to recognize digits for processing deposits in ATM machines. To this day, some ATMs still run the code that Yann LeCun and his colleague Leon Bottou wrote in the 1990s!

UNIVERSITY of WEST FLORIDA

LeNet



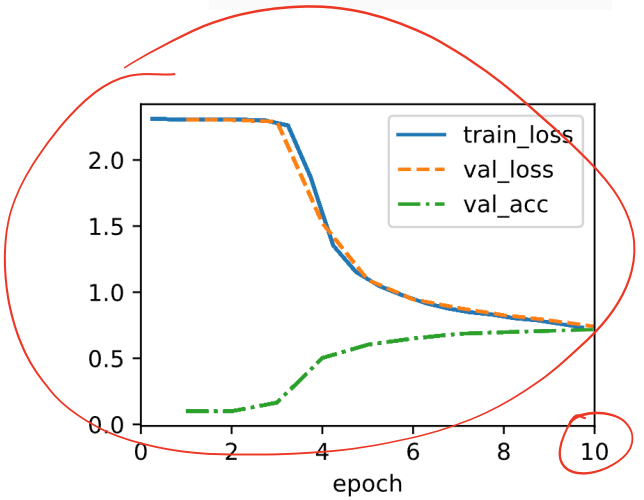



```
class LeNet(d2l.Classifier): #@save
"""The LeNet-5 model."""
 def __init__(self, lr=0.1, num_classes=10):
        super().__init__() self.save_hyperparameters()
        self.net = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=6, kernel_size=5, activation='sigmoid',
            padding='same'),
            tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
            tf.keras.layers.Conv2D(filters=16, kernel_size=5, activation='sigmoid'),
            tf.keras.layers.AvgPool2D(pool_size=2, strides=2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(120, activation='sigmoid'),
            tf.keras.layers.Dense(84, activation='sigmoid'),
            tf.keras.layers.Dense(num_classes)])
```

```
@d2l.add_to_class(d2l.Classifier) #@save
def layer_summary(self, X_shape):
        X = tf.random.normal(X_shape)
        for layer in self.net.layers:
            X = layer(X)
            print(layer.__class__.__name__,
            'output shape:\t', X.shape)

model = LeNet()
model.layer_summary((1, 28, 28, 1))
```

```
trainer = d2l.Trainer(max_epochs=10)
data =
d2l.FashionMNIST(batch_size=128)
with d2l.try_gpu():
        model = LeNet(lr=0.1)
        trainer.fit(model, data)
```



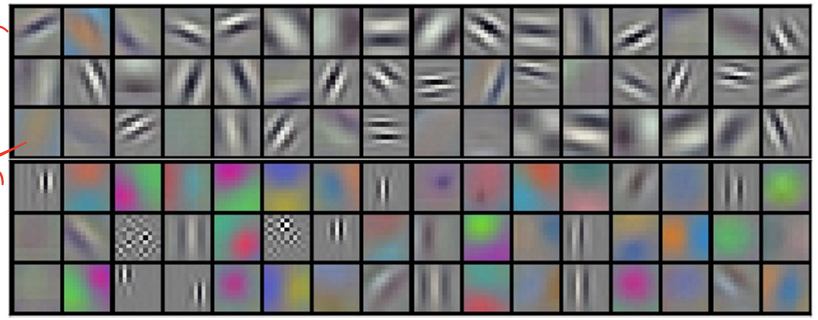
Modern CNN

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images. The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU 2.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* (pp. 1097–1105).

☆ Save 99 Cite Cited by 118831 Related articles All 111 versions »

☆ Save 99 Cite Cited by 22393 Related articles All 5 versions

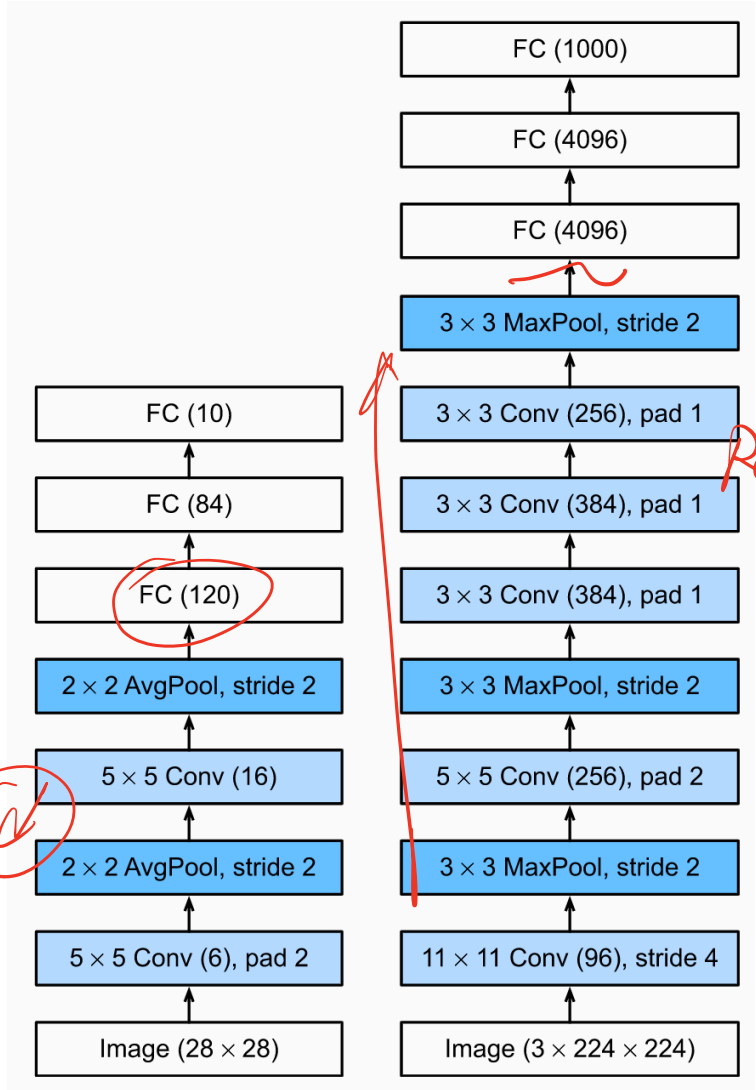
The first modern CNN (Krizhevsky *et al.*, 2012), named *AlexNet* after one of its inventors, Alex Krizhevsky, is largely an evolutionary improvement over LeNet. It achieved excellent performance in the 2012 ImageNet challenge.

Why did it take so long?

- Deep models with many layers require large amounts of data. However, given the limited storage capacity of computers, the relative expense of (imaging) sensors, and the comparatively tighter research budgets in the 1990s, most research relied on tiny datasets.
- CPUs are comparatively bad at any single task when compared to dedicated hardware. Modern laptops have 4–8 cores, and even high-end servers rarely exceed 64 cores per socket, simply because it is not cost-effective.
- *Graphical processing units* (GPUs) proved to be a game changer in making deep learning feasible. GPUs can consist of thousands of small processing elements often grouped into larger groups. While each core is relatively weak, running at about 1GHz clock frequency, it is the total number of such cores that makes GPUs orders of magnitude faster than CPUs.

AlexNet

- AlexNet, which employed an 8-layer CNN, won the ImageNet Large Scale Visual Recognition Challenge 2012 by a large margin (Russakovsky *et al.*, 2013). This network showed, for the first time, that the features obtained by learning can transcend manually-designed features, breaking the previous paradigm in computer vision.
- The architectures of AlexNet and LeNet are strikingly similar. Note that we provide a slightly streamlined version of AlexNet removing some of the design quirks that were needed in 2012 to make the model fit on two small GPUs.
- There are also significant differences between AlexNet and LeNet.
  - First, AlexNet is much deeper than the comparatively small LeNet5. AlexNet consists of eight layers: five convolutional layers, two fully connected hidden layers, and one fully connected output layer.
  - Second, AlexNet used the ReLU instead of the sigmoid as its activation function. Let's delve into the details below.

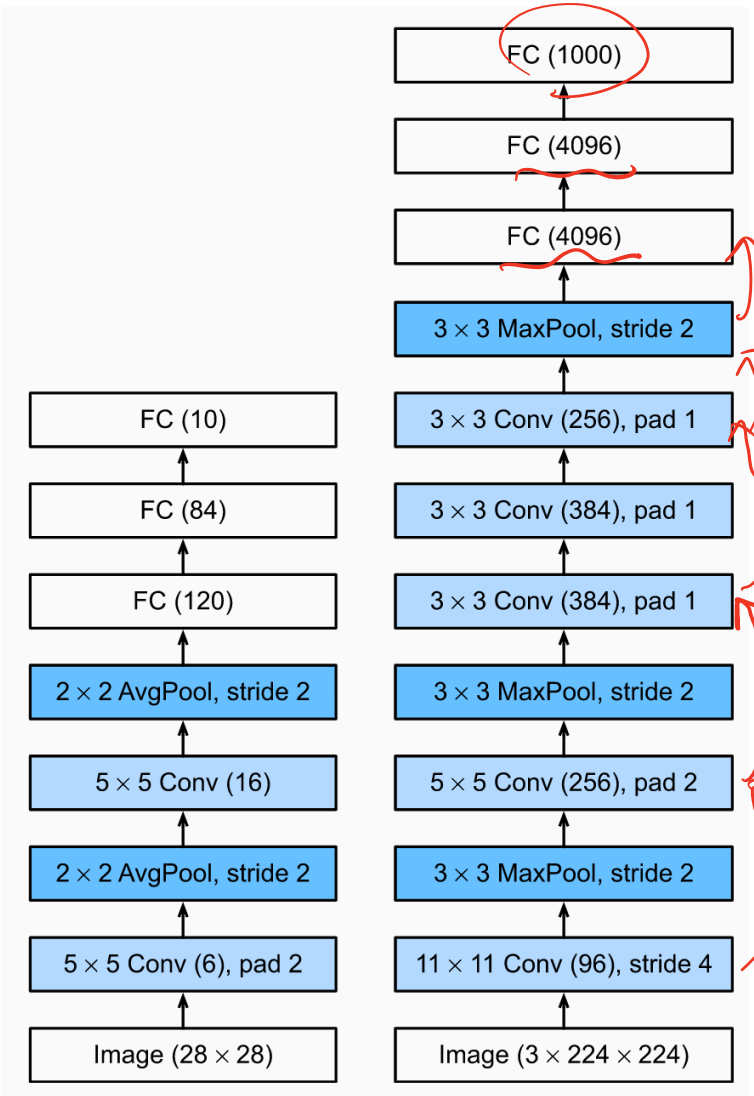


From LeNet (left) to AlexNet (right)

AlexNet

Architecture

- In AlexNet's first layer, the convolution window shape is 11×11.
- The convolution window shape in the second layer is reduced to 5×5, followed by 3×3.
- In addition, after the first, second, and fifth convolutional layers, the network adds max-pooling layers with a window shape of 3×3 and a stride of 2.
- Moreover, AlexNet has ten times more convolution channels than LeNet.
- After the last convolutional layer, there are two huge fully connected layers with 4096 outputs. These layers require nearly 1GB model parameters.
- Due to the limited memory in early GPUs, the original AlexNet used a dual data stream design, so that each of their two GPUs could be responsible for storing and computing only its half of the model.
- Fortunately, GPU memory is comparatively abundant now, so we rarely need to break up models across GPUs these days (our version of the AlexNet model deviates from the original paper in this aspect).



From LeNet (left) to AlexNet (right)

AlexNet

Activation Functions

Besides, AlexNet changed the sigmoid activation function to a simpler ReLU activation function.

- On the one hand, the computation of the ReLU activation function is simpler.
- On the other hand, the ReLU activation function makes model training easier when using different parameter initialization methods.
- When the output of the sigmoid activation function is very close to 0 or 1, the gradient of these regions is almost 0, so that backpropagation cannot continue to update some of the model parameters. In contrast, the gradient of the ReLU activation function in the positive interval is always 1. Therefore, if the model parameters are not properly initialized, the sigmoid function may obtain a gradient of almost 0 in the positive interval, so that the model cannot be effectively trained.

```
class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential()
        self.net.add(
            nn.Conv2D(96, kernel_size=11, strides=4, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Conv2D(256, kernel_size=5, padding=2, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
            nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
            nn.Conv2D(256, kernel_size=3, padding=1, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(num_classes))
        self.net.initialize(init.Xavier())
```

AlexNet

Capacity Control and Preprocessing

AlexNet controls the model complexity of the fully connected layer by dropout, while LeNet only uses weight decay.

To augment the data even further, the training loop of AlexNet added a great deal of image augmentation, such as flipping, clipping, and color changes. This makes the model more robust and the larger sample size effectively reduces overfitting.

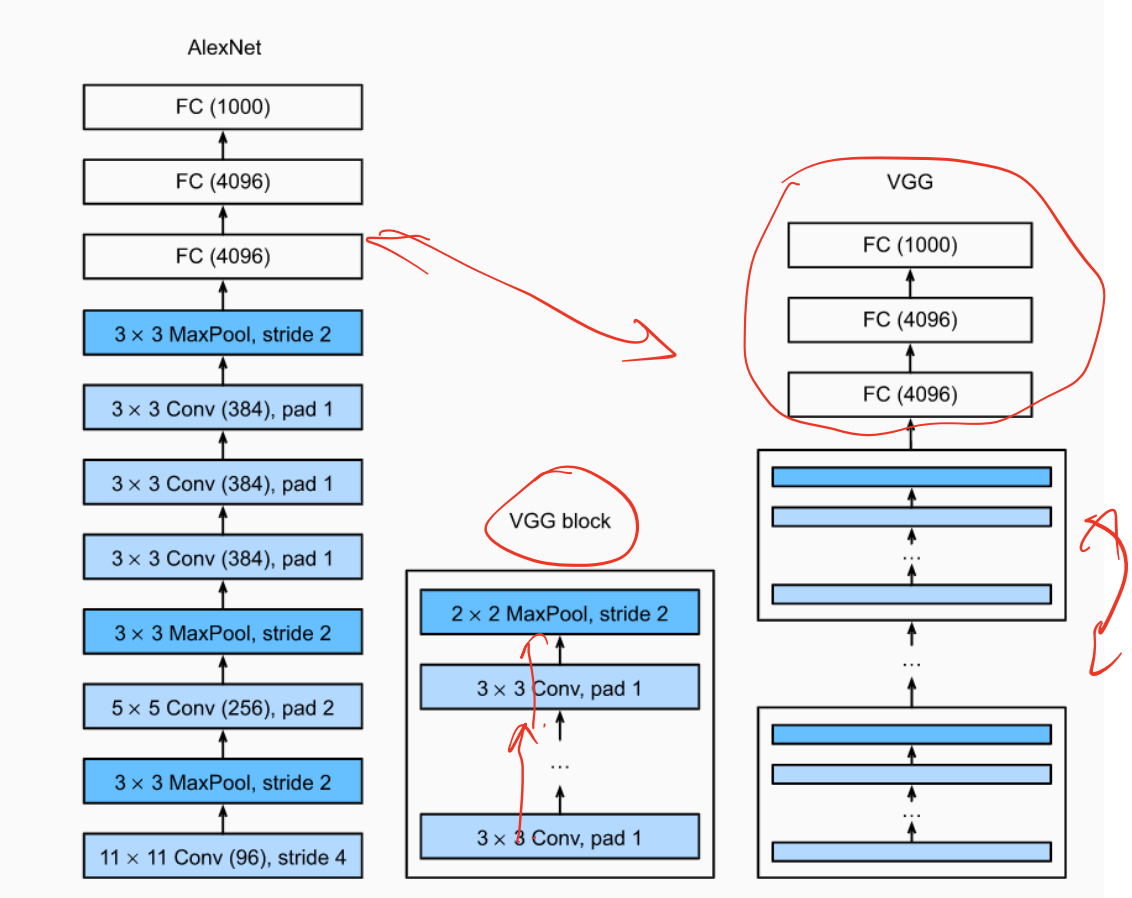
```
class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential()
        self.net.add(
            nn.Conv2D(96, kernel_size=11, strides=4, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Conv2D(256, kernel_size=5, padding=2, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
            nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
            nn.Conv2D(256, kernel_size=3, padding=1, activation='relu'),
            nn.MaxPool2D(pool_size=3, strides=2),
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),
            nn.Dense(num_classes))
        self.net.initialize(init.Xavier())
```

Networks Using Blocks (VGG)

Capacity Control and Preprocessing

While AlexNet offered empirical evidence that deep CNNs can achieve good results, it did not provide a general template to guide subsequent researchers in designing new networks. In the second half, we will introduce networks with blocks.

The idea of using blocks first emerged from the Visual Geometry Group (VGG) at Oxford University. It is easy to implement these repeated structures in code with any modern deep learning framework by using loops and subroutines.



From AlexNet to VGG. The key difference is that VGG consists of blocks of layers, whereas AlexNet's layers are all designed individually.

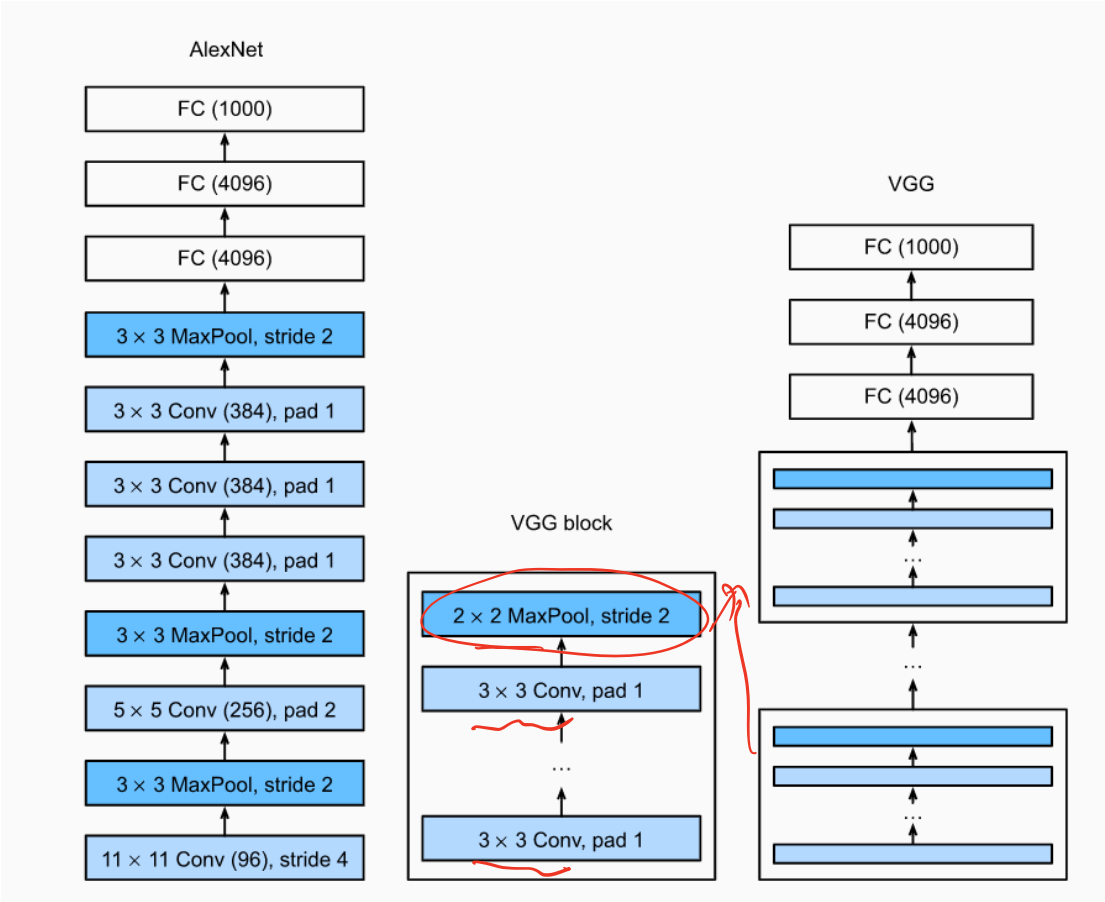
Networks Using Blocks (VGG)

VGG Blocks

The basic building block of CNNs is a sequence of the following:

- A convolutional layer with padding to maintain the resolution
- A nonlinearity such as a ReLU
- A pooling layer such as max-pooling to reduce the resolution.

A VGG block consists of a *sequence* of convolutions with 3×3 kernels with padding of 1 (keeping height and width) followed by a 2×2 max-pooling layer with stride of 2 (halving height and width after each block)



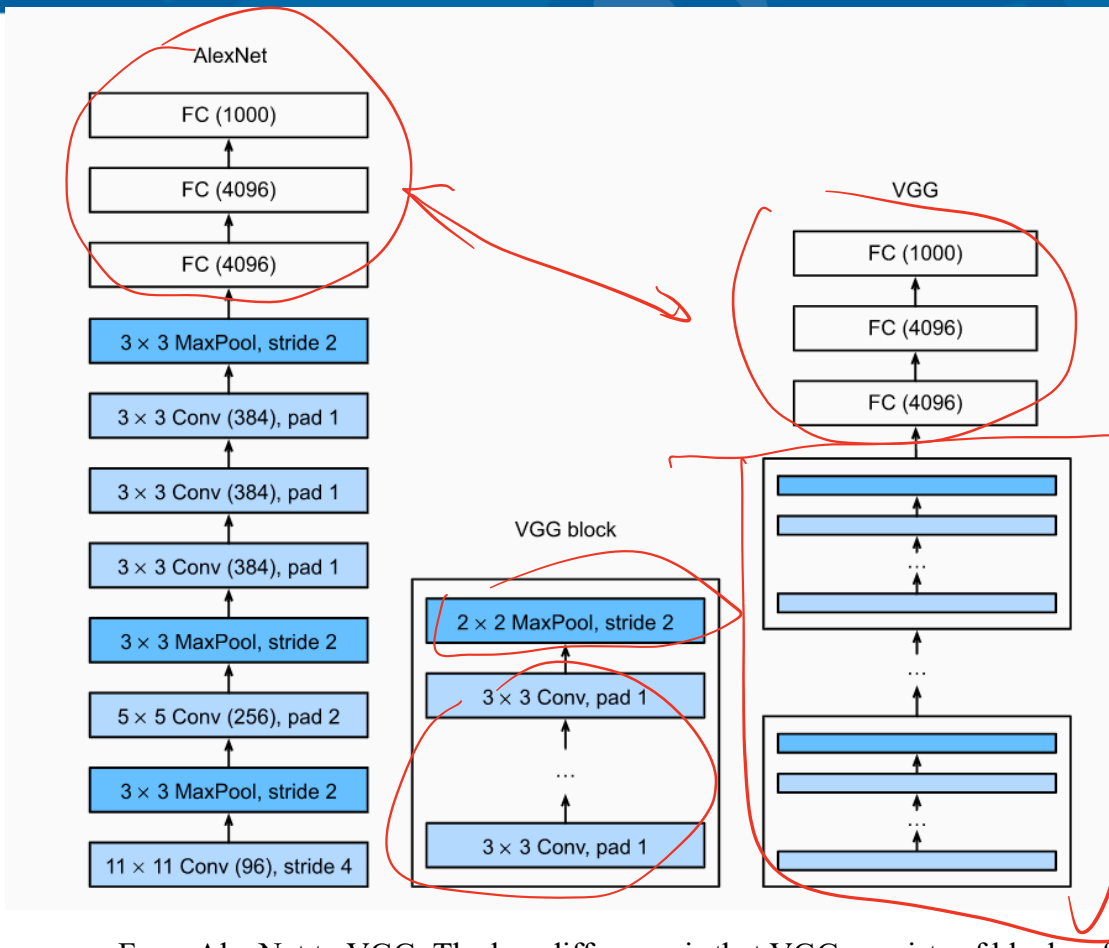
From AlexNet to VGG. The key difference is that VGG consists of blocks of layers, whereas AlexNet's layers are all designed individually.

Networks Using Blocks (VGG)

VGG Network

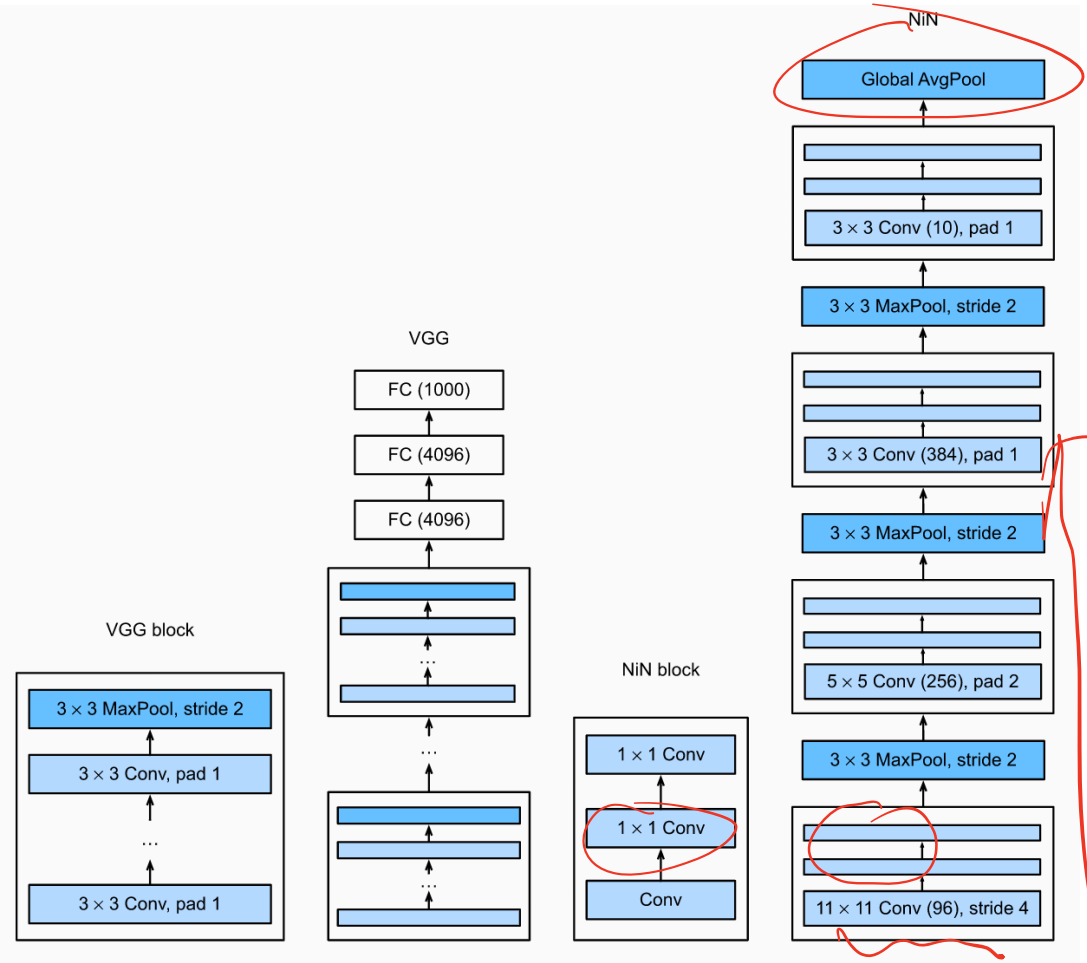
- VGG Network can be partitioned into two parts: the first consisting mostly of convolutional and pooling layers and the second consisting of fully connected layers that are identical to those in AlexNet.
- The key difference is that the convolutional layers are grouped in nonlinear transformations that leave the dimensionality unchanged, followed by a resolution-reduction step.
- The convolutional part of the network connects several VGG blocks in succession. This grouping of convolutions is a pattern that has remained almost unchanged over the past decade, although the specific choice of operations has undergone considerable modifications

```
class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = tf.keras.models.Sequential()
        for (num_convs, num_channels) in arch:
            self.net.add(vgg_block(num_convs, num_channels))
        self.net.add(
            tf.keras.models.Sequential([
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(4096, activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(4096, activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(num_classes)]))
```



From AlexNet to VGG. The key difference is that VGG consists of blocks of layers, whereas AlexNet's layers are all designed individually.

Network in Network (NiN)



The *network in network* (*NiN*) blocks (Lin *et al.*, 2013) offer an alternative, capable of solving both problems in one simple strategy. They were proposed based on a very simple insight:

- use 1×1 convolutions to add local nonlinearities across the channel activations
- use global average pooling to integrate across all locations in the last representation layer. Note that global average pooling would not be effective, were it not for the added nonlinearities.

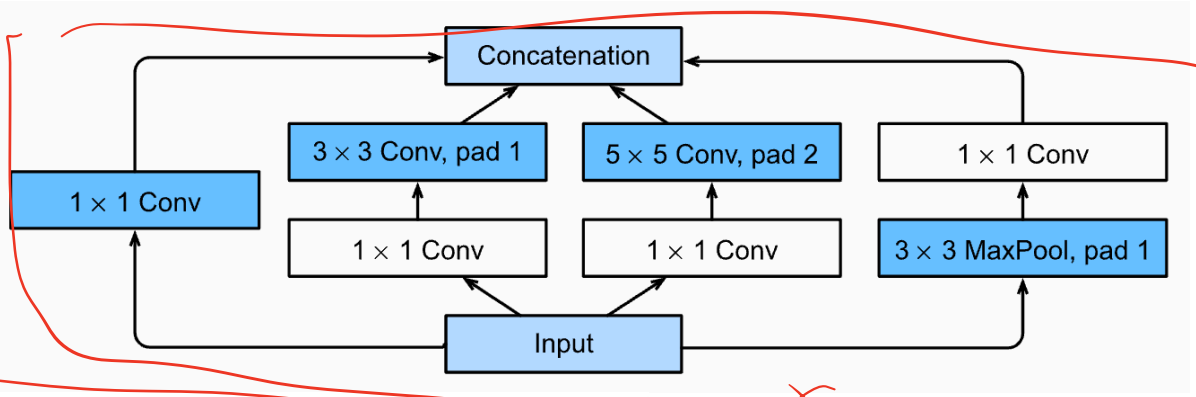
The idea behind NiN is to apply a fully connected layer at each pixel location (for each height and width). The resulting 1×1 convolution can be thought as a fully connected layer acting independently on each pixel location.

The figure illustrates the main structural differences between VGG and NiN, and their blocks. Both the difference in the NiN blocks (the initial convolution is followed by 1×1 convolutions, whereas VGG retains 3×3 convolutions) and in the end where we no longer require a giant fully connected layer.

Multi-Branch Networks (GooLeNet)

- In 2014, *GoogLeNet* won the ImageNet Challenge (Szegedy *et al.*, 2015), using a structure that combined the strengths of NiN (Lin *et al.*, 2013), repeated blocks (Simonyan and Zisserman, 2014), and a cocktail of convolution kernels. It is arguably also the first network that exhibits a clear distinction among the stem (data ingest), body (data processing), and head (prediction) in a CNN.
- The key contribution in GoogLeNet was the design of the network body. It solved the problem of selecting convolution kernels in an ingenious way. While other works tried to identify which convolution, ranging from 1×1 to 11×11 would be best, it simply *concatenated* multi-branch convolutions.

Structure of the Inception block



Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]

[1]Google Inc. [2]University of North Carolina, Chapel Hill

[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.

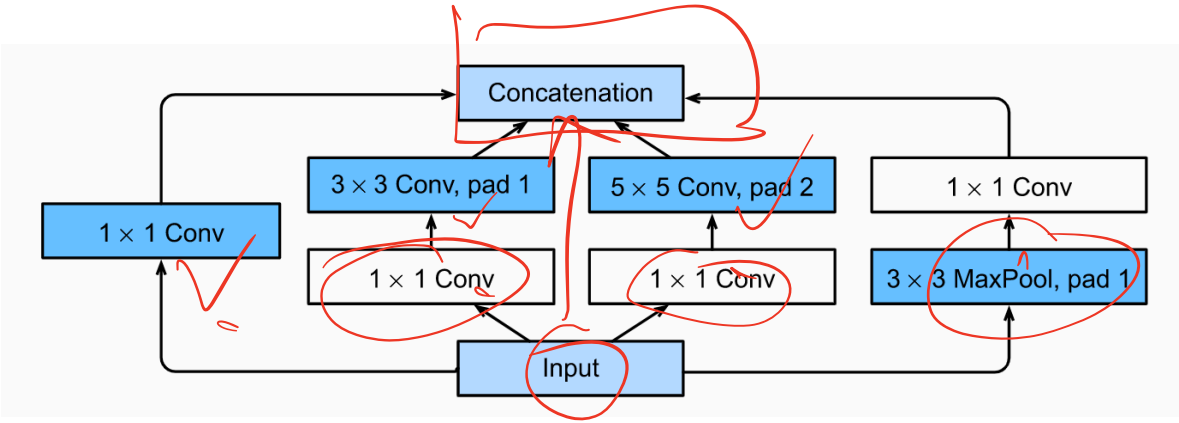
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com

[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Multi-Branch Networks (GooLeNet)

Inception Blocks

The basic convolutional block in GoogLeNet is called an *Inception block*, stemming from the meme "we need to go deeper" of the movie *Inception*.



Structure of the inception block

As depicted in the figure above, the inception block consists of four parallel branches.

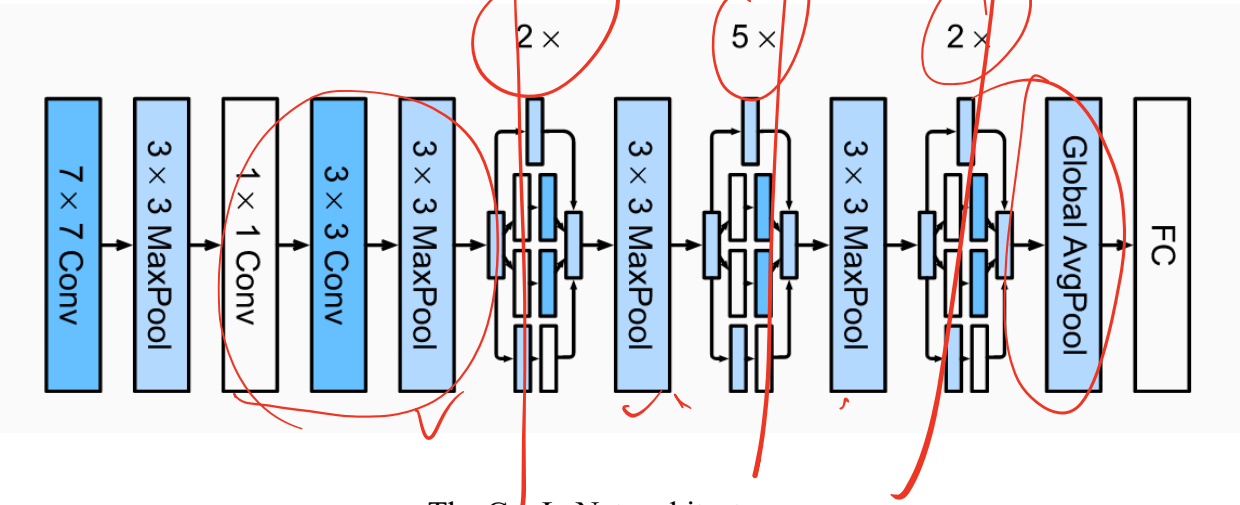
- The first three branches use convolutional layers with window sizes of 1×1, 3×3, and 5×5 to extract information from different spatial sizes.
- The middle two branches also add a 1×1 convolution of the input to reduce the number of channels, reducing the model's complexity.
- The fourth branch uses a 3×3 max-pooling layer, followed by a 1×1 convolutional layer to change the number of channels.

The four branches all use appropriate padding to give the input and output the same height and width. Finally, the outputs along each branch are concatenated along the channel dimension and comprise the block's output. The commonly-tuned hyperparameters of the Inception block are the number of output channels per layer, i.e., how to allocate capacity among convolutions of different size.

Multi-Branch Networks (GooLeNet)

GooLeNet Model

GoogLeNet uses a stack of a total of 9 inception blocks, arranged into 3 groups with max-pooling in between, and global average pooling in its head to generate its estimates. Max-pooling between inception blocks reduces the dimensionality. At its stem, the first module is similar to AlexNet and LeNet.



The GooLeNet architecture