ICT303 – Advanced Machine Learning and Artificial Intelligence

Topic 6: Convolutional Neural Networks (CNN) – Part II

Hamid Laga
H.Laga@murdoch.edu.au
Office: 245.1.020

How to Get in Touch with the Teaching Team

- Internal and External Students
  - Email: H.Laga@murdoch.edu.au.
- Important
  - In any communication, please make sure that you
    - Start the subject of your email with ICT303
    - Include your student ID, name, and the lab slot in which you are enrolled.
  - We will do all our best to answer your queries within 24 hrs.

In this Lecture

- Training Deep Neural Networks
  - Data preparation
  - Training, validation and testing
  - Finetuning the hyper parameters
- Modern CNNs
  - Batch Normalization
  - Deep CNN (AlexNet)
  - Networks using Blocks (VGG)
  - Network in Network (NiN)
  - Multi-branch Networks (GoogLeNet)
  - Residual Networks (ResNet and ResNeXt)
  - Densely Connected Networks
- Summary

- Learning objectives
  - Implement the training procedure of a CNN
  - Understand and implement popular deep neural networks
- Additional readings
  - Chapter 8 of the textbook, available at: https://d2l.ai/

Training Neural Networks – Data Preparation

- Normalization

Training Neural Networks – Data Preparation

- Split the data set into three subsets: training, validation and testing
- Usually, we use
  - 70% of the data for training, 20% for validation and 10% for testing, or
  - 70% of the data for training, 10% for validation and 20% for testing

Training Neural Networks – Data Preparation

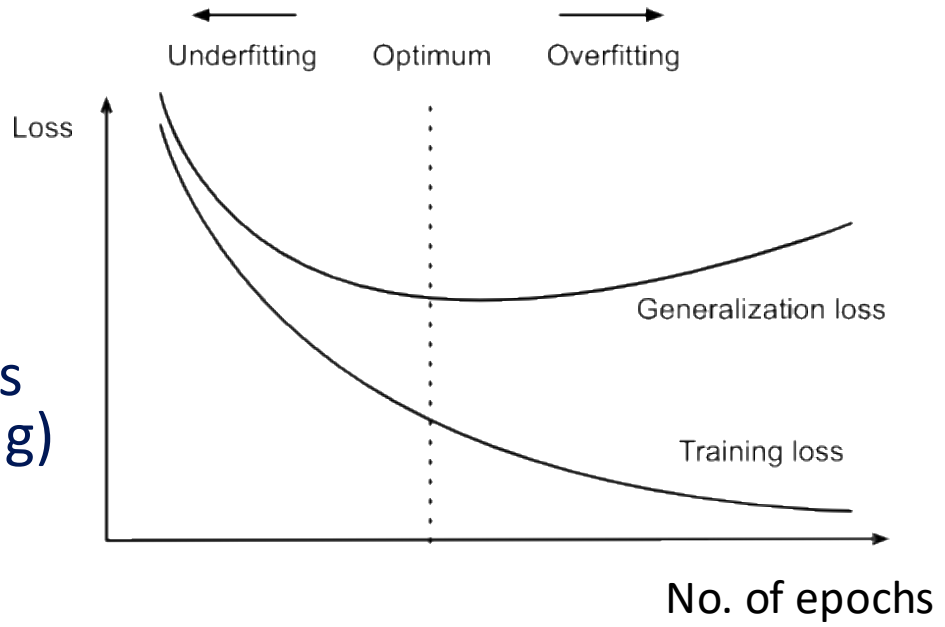
- Training set
  - This is the data set you will use to learn the best parameters of the network
  - At each epoch,
    - update the network parameters and
    - measure the overall loss on this data set – this is called training loss.
- Validation set
  - Used only to decide when to stop training to avoid overfitting, and to tune the hyper parameters
  - DO NOT use it to update the network parameters
  - At each epoch,
    - measure the loss on this validation data set.
    - This is called generalization loss
- Testing set
  - Used to measure the actual performance of the trained network
  - It should never ever be used to learn the parameters or tune the hyper parameters

Training Neural Networks – Training

- When should I stop training?
- Training for a very large number of iterations is not necessarily good
  - Instead of learning, the model will start memorizing the answers
  - It will then perform very poorly on an input that it did not see during training
  - This is called overfitting

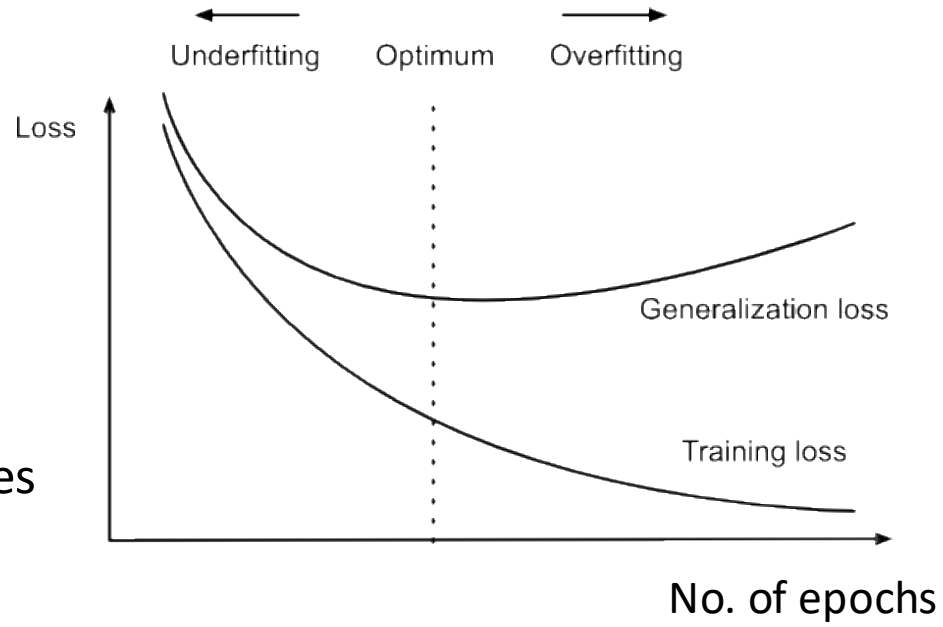
Training Neural Networks – Training

- When should I stop training?
- At each epoch
  - Training loss will decrease
  - Generalization loss will decrease, but at a certain point it starts increasing
- In the early epochs
  - The model is underfitting since both the training loss and generalization loss are high (although decreasing)
- When the generalization loss starts increasing, it means the model starts overfitting
  - You need to stop the training right before this point



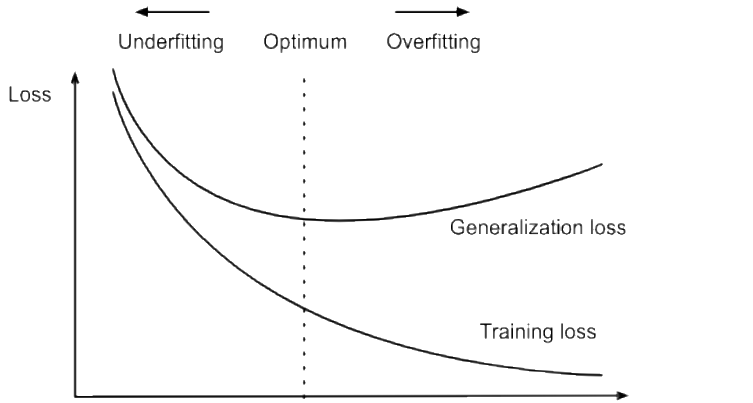
Training Neural Networks – Training

- In your implementation, you need to
  - Compute the training and validation loss at each epoch
  - Save the state (the learned parameters) of the network at regular times, e.g., every 5 or 10 epochs
    - If the machine crashes, you only need to restart from the last saved state
  - Once training is finished,
    - Find the epoch no. where the generalization loss starts going up
    - The network parameters at that point are the optimal ones

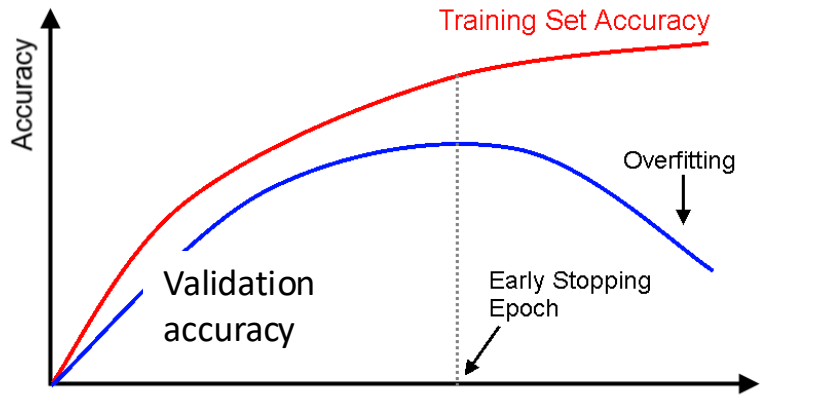


Training Neural Networks – Training

- In addition to the loss, it is good to visualize the accuracy of the network on training and on validation
  - In classification, accuracy is the percentage of correct predictions obtained
  - The higher the accuracy is the better



Loss curves



Accuracy curves

Training Neural Networks – Hyper Parameters

- A Deep Neural Network has many hyper parameters that can significantly affect performance
  - Architecture
    - Network type
    - Number of layers and number of neurons in each layer
    - Type of activation function at each layer
  - Learning
    - Learning rate
    - Optimizer type
    - Type and amount of training data
    - Batch size
- How to set them?

Training Neural Networks – Hyper Parameters

- How to set them?
  - Test every possible configuration → impossible (a lot of time and expensive)
  - Use some intuitions to narrow down the list of options
    - Network type
      - Images → CNN, non-spatially structured data → MLPs
    - No. of layers and no. of neurons per layer
      - The more complex the problem the more layers you will likely need
      - Small training data set → small no. of layers and neurons per layer,
    - Activation functions → likely ReLU is the best (tested and validated by others)
    - Learning rate
      - Use scheduling: start with relatively large one and reduce it toa small one
    - Optimizers
      - Look at what others did, e.g., Adams is very popular
    - Batch size
      - Need to find a good trade-off between memory requirements, speed and accuracy (large batch size → faster, more memory but performance may drop at a certain point)

Training Neural Networks – Hyper Parameters

- How to set them?
  - Test every possible configuration → impossible (a lot of time and expensive)
  - Use some intuitions to narrow down the list of options
    - Many tools have been developed to help automate/streamline this process
      - TensorBoard – runs for TensorFlow but can also be set for PyTorch
      - TensorBoardX – TensorBoard's extension to PyTorch
      - There are many other alternatives, see: https://neptune.ai/blog/the-best-tensorboard-alternatives

In this Lecture

- Training Deep Neural Networks
  - Data preparation
  - Training, validation and testing
- Modern CNNs
  - Deep CNN (AlexNet)
  - Networks using Blocks (VGG)
  - Network in Network (NiN)
  - Multi-branch Networks (GoogLeNet)
  - Residual Networks (ResNet and ResNeXt)
  - Densely Connected Networks
  - Batch Normalization
- Summary

- Learning objectives
  - Implement the training procedure of a CNN
  - Understand and implement popular deep neural networks
- Additional readings
  - Chapter 8 of the textbook, available at: https://d2l.ai/

Deep Convolutional Neural Network

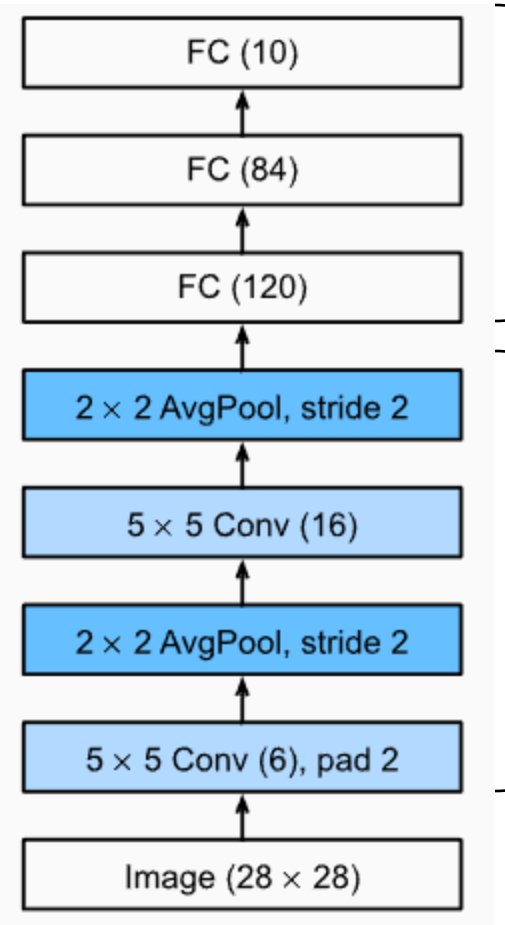
- Between 1990s (introduction of LeNet) and 2012,
  - NN were surpassed by traditional Machine Learning models (e.g., SVM, AdaBoost, etc.)
  - Issues with computation power and insufficient data for training
- Classical pipeline (e.g., for people recognition)
  - Obtain an interesting data set, e.g., RGB images of people and their associated identities
  - Representation
    - Represent each image with some pre-defined types of features (e.g., histogram)
    - These are called hand-crafted features since their type is predefined
  - Dump the representations into classical machine learning algorithm (e.g., a classifier such as SVM)

Deep Convolutional Neural Network

- In the classical pipeline
  - The representation is NOT learned – it is predefined (called handcrafted)
  - Researchers design the representation(usually by try and error, guided by intuition and experience)
- The deep CNN revolution
  - The representation can also be learned from data

Deep Convolutional Neural Network

- In the classical pipeline
  - The representation is NOT learned – it is predefined (called handcrafted)
  - Researchers design (usually by try and error, guided by intuition and experience)
- The deep CNN revolution
  - The representation can also be learned from data



Example of LeNet

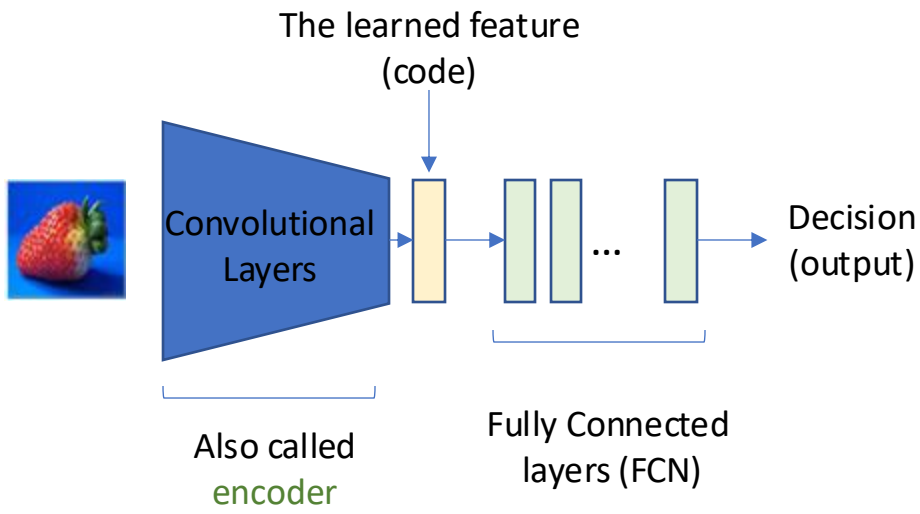
Classifier (an MLP, also designated by FCN for fully connected layers)
- Takes the code and outputs the class label (e.g., horse, cat, …)

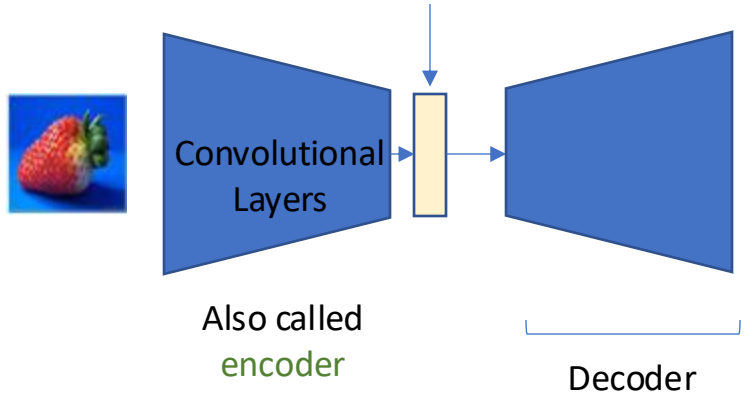
Encoder
- Encodes the input image into a compact code (the feature)

https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

Deep Convolutional Neural Network

- In the classical pipeline
  - The representation is NOT learned – it is predefined (called handcrafted)
  - Researchers design (usually by try and error, guided by intuition and experience)
- The deep CNN revolution
  - The representation can also be learned from data

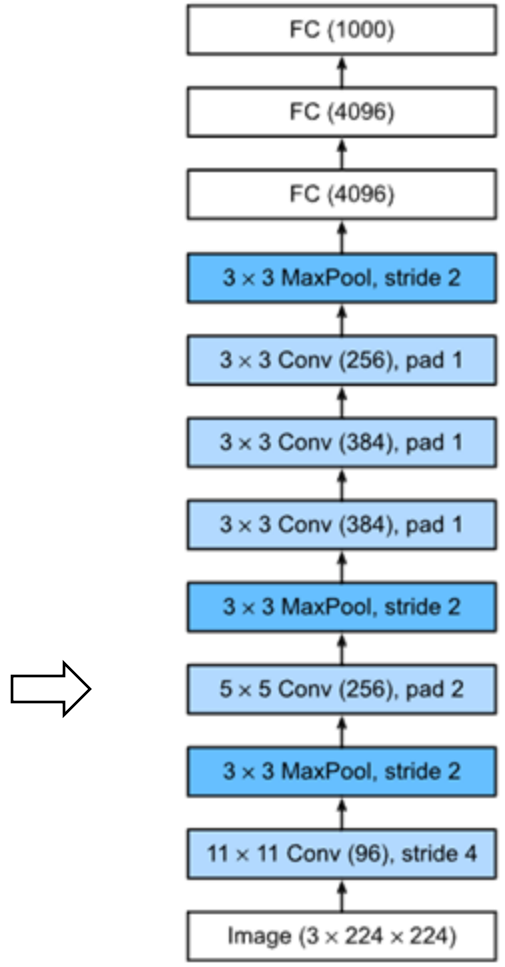




Deep Convolutional Neural Network - AlexNet

- Architecture of AlexNet similar to LeNet
  - AlexNet is deeper (more convolutional layers)
  - Changed the activation function from Sigmoid (LeNet) to the simpler ReLU
- Exercise
  - Write Python code of AlexNet and train on GPU using MNIST dataset
  - How much memory the last three FC layers would need?
  - How much memory the entire network would require

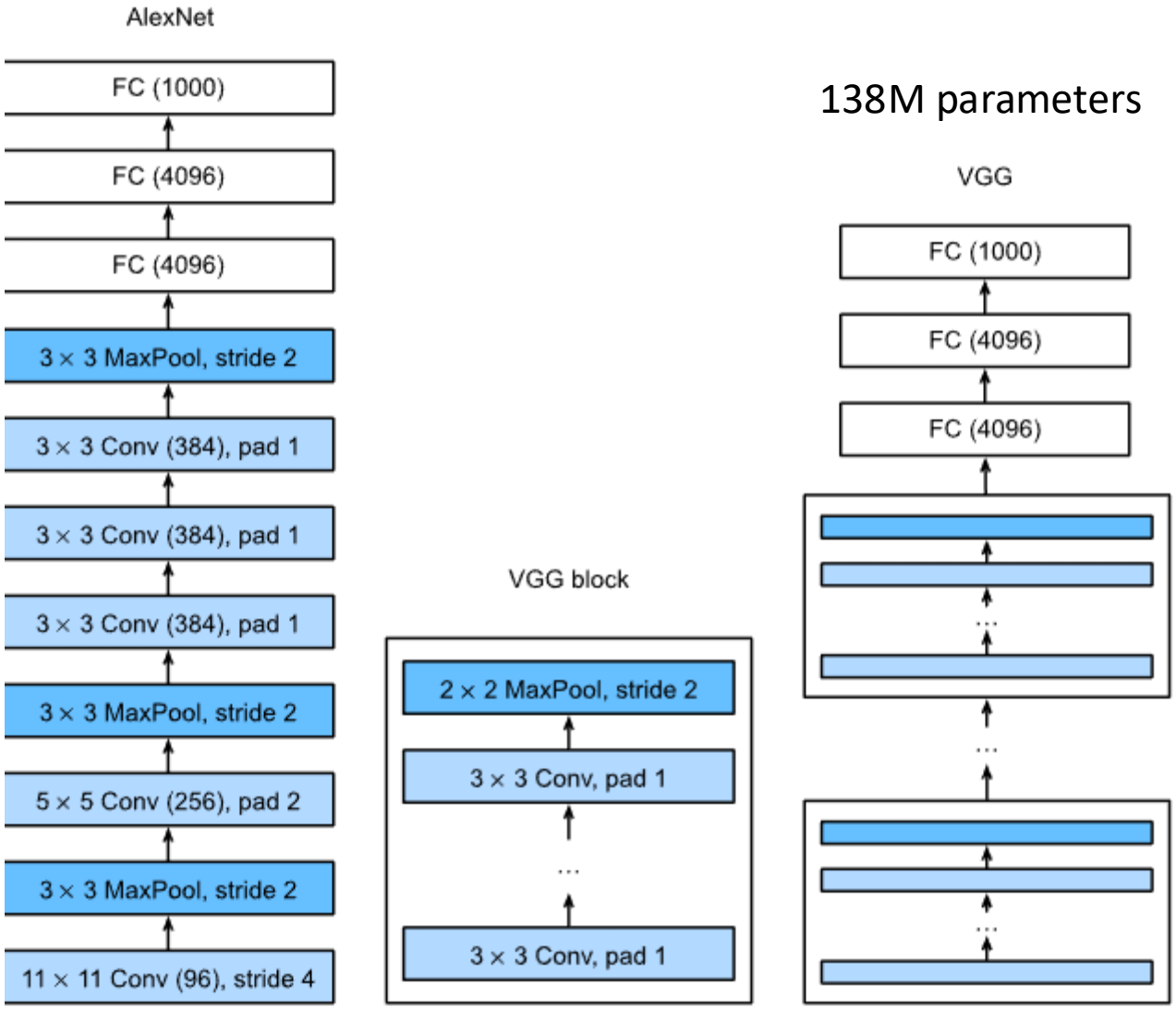
LeNet
(60K Parameters))



AlexNet
(60M parameters))

Networks using Blocks (VGG)

- A VGG Block is composed of a series of convolutional and pooling layers,
  - 3x3 convolutional filters (Conv = 3x3), stride 1
  - Last layer is 2x2 MaxPool, stride = 2
- A VGG Network is composed of
  - A series of VGG Blocks followed at the end by a series of FC layers
- Simplified Deep NN by using
  - Conv: 3x3, stride – 1
  - MaxPool: 2x, stride = 2
- VGG-16 refers to 16 layers that have weights
  - Followed simple principle of
    - Halving the height/width sizes and
    - But doubling the channels (#filters) after each VGG Block



https://d2l.ai/chapter_convolutional-modern/vgg.html

Challenges Posed by LeNet, AlexNet and VGG

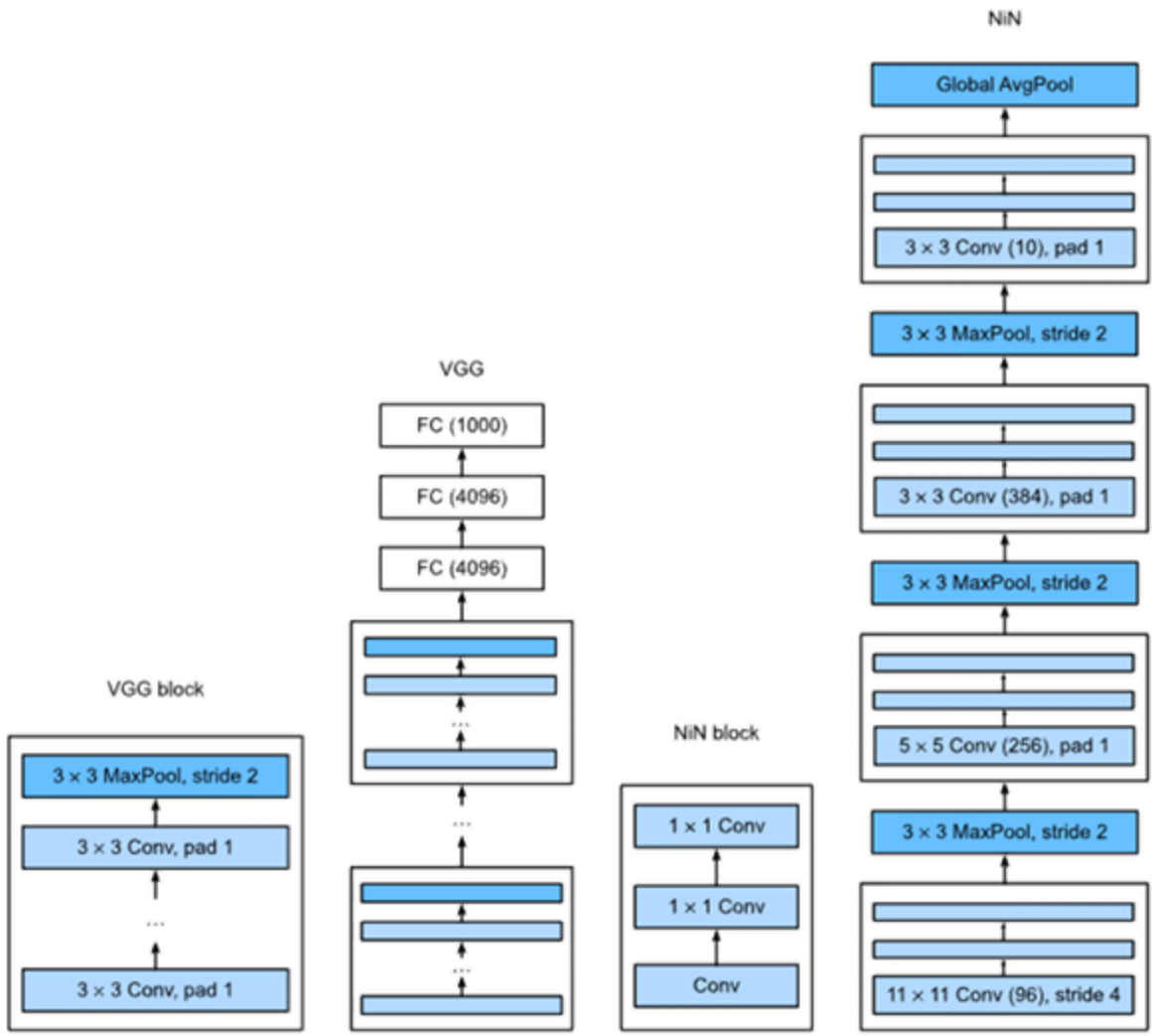
- Require a huge amount of memory
  - Fully connected layers at the end of the architecture consume tremendous numbers of parameters.
    - For instance, even a simple model such as VGG-11 requires a monstrous 25088×4096 matrix, occupying almost 400MB of RAM in single precision (FP32).
  - Even high-end mobile phones sport no more than 8GB of RAM.
  - At the time VGG was invented, this was an order of magnitude less (the iPhone 4S had 512MB

Network in Network (NiN)

- Removes the need for FC layers
  - uses two 1x1 Conv layers for each Conv block,
  - This effectively replaces the FC layers.
  - Significantly fewer params
- Uses global average pooling for final classification layer.

```
def nin_block(out_channels, kernel_size, strides, padding):
    return nn.Sequential(
        nn.LazyConv2d(out_channels, kernel_size, strides, padding), nn.ReLU(),
        nn.LazyConv2d(out_channels, kernel_size=1), nn.ReLU(),
        nn.LazyConv2d(out_channels, kernel_size=1), nn.ReLU())
```

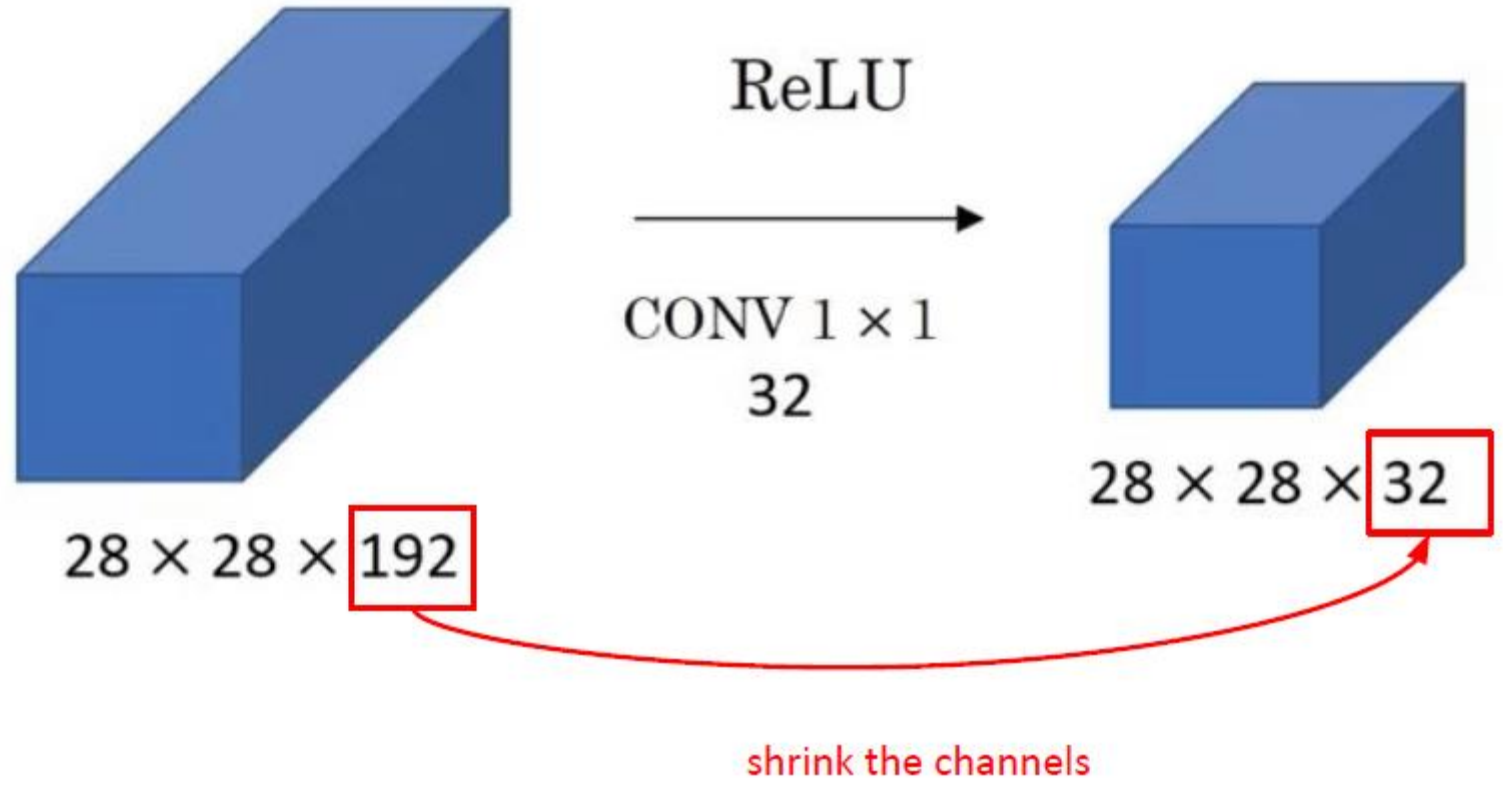
Why LazyConv2d?



https://d2l.ai/chapter_convolutional-modern/nin.html

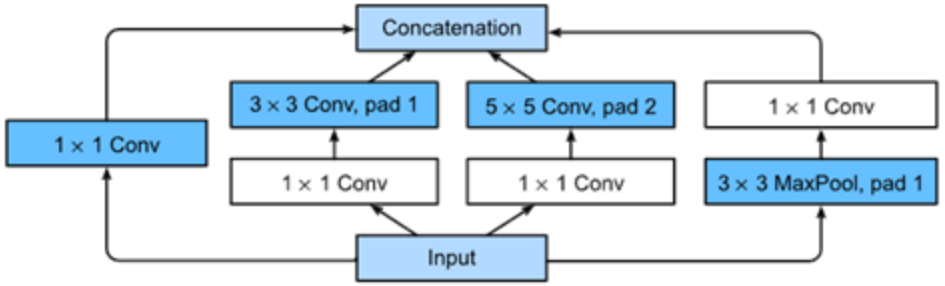
Why use 1x1 Convolution

- Shrink the channels but keep height and width unchanged

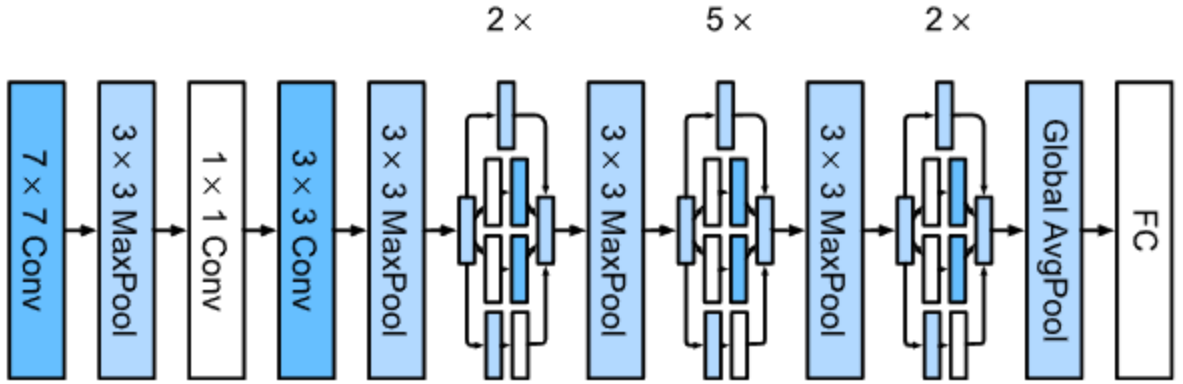


Multi-Branch Networks (GoogLeNet)

- Which convolutions to use at each convolutional layers?
  - You can try kernel sizes of 1x1 until 11x11 and then pick up the best
  - Time consuming and tedious!
- GoogLeNet solved this problem by concatenating multi-branch convolutions, known as Inception Network



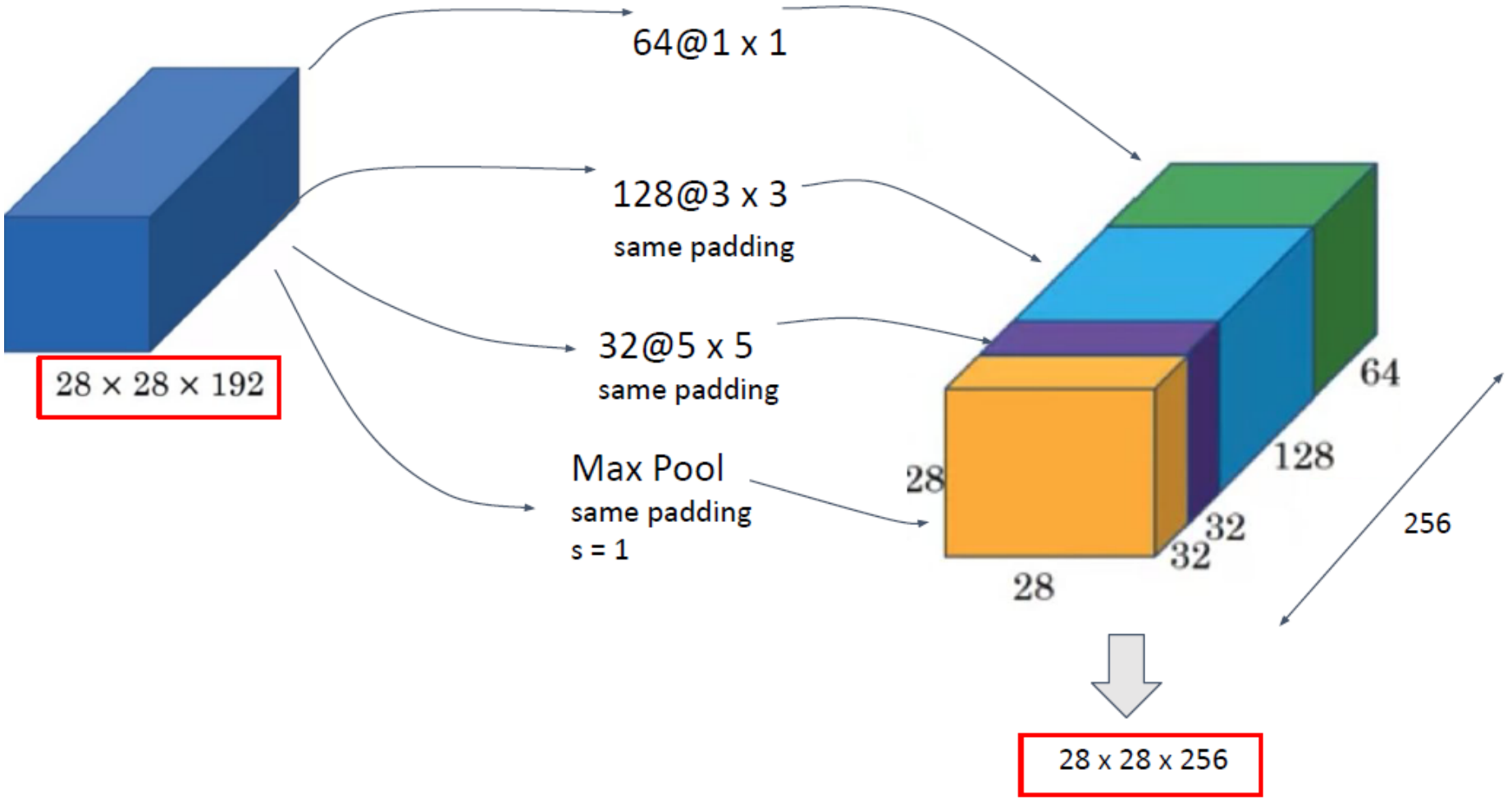
Inception Block
(It tries many filter sizes)



GoogLeNet Model

https://d2l.ai/chapter_convolutional-modern/googlenet.html

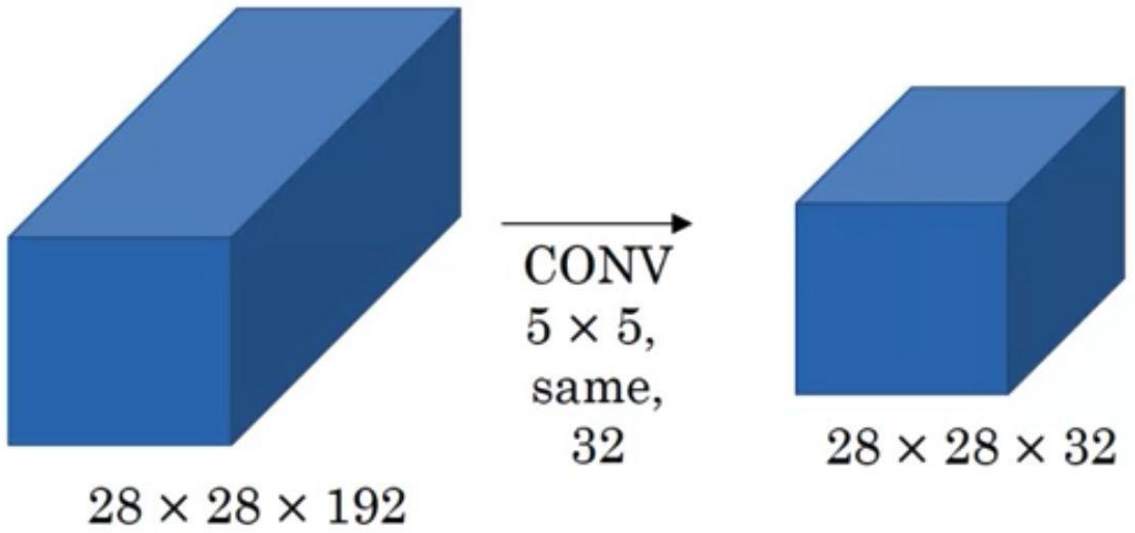
Multi-Branch Networks (GoogLeNet) - Illustration



https://d2l.ai/chapter_convolutional-modern/googlenet.html

Multi-Branch Networks (GoogLeNet) – Computation Cost!

- Every filter is 5 x 5 x 192, and for every position in the resulting 28 x 28,
- we do 5 x 5 x 192 multiplications
  - So 5 x 5 x 192 x 28 x 28.
- But there are 32 of these
  - So the total is 5 x 5 x 192 x 28 x 28 x 32 = 120 million multiplications!!



https://d2l.ai/chapter_convolutional-modern/googlenet.html

Computational Complexity of Multi-channel Convolution

For an image of size

$$h \times w$$

then the cost of computing a

$$k \times k$$

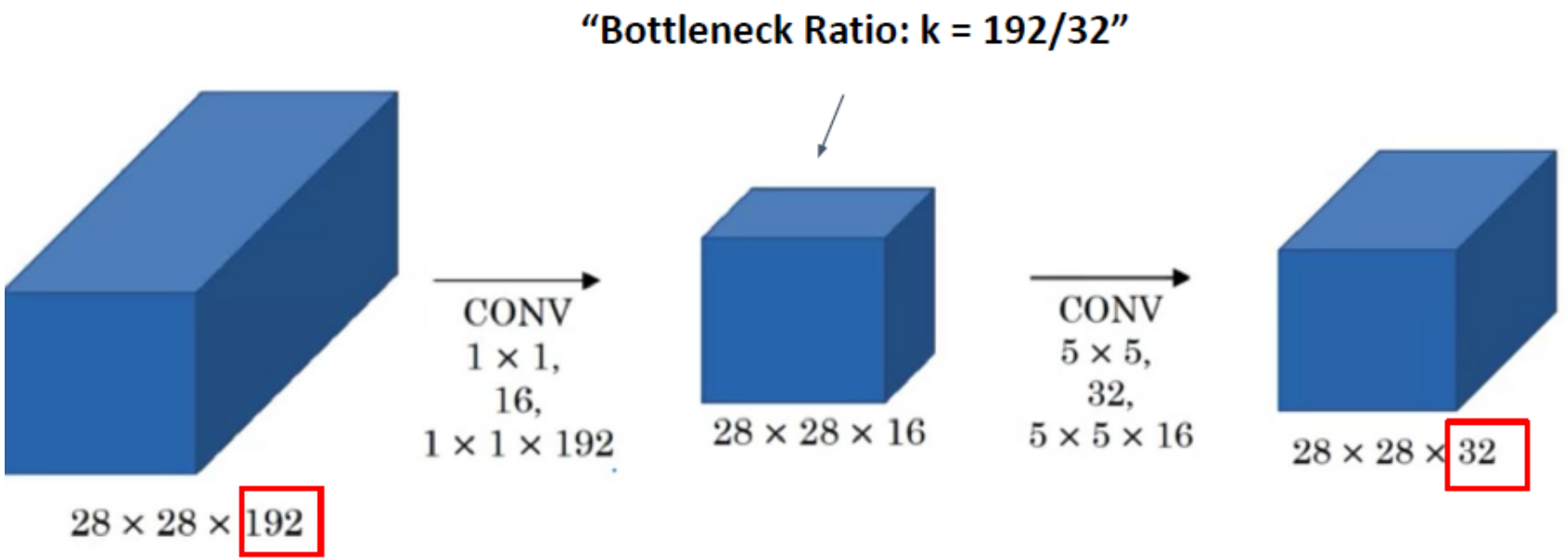
is

$$\mathrm{O}(hwk^2).$$

With c_i and c_o channels the complexity increases to

$$\mathrm{O}(hwk^2 c_i c_o)$$

Using 1x1 Convolution to Reduce Complexity



Total Multiplications:

1 x 1 x 192 x 28 x 28 x 16

= 2.4 million

Total Multiplications:

5 x 5 x 16 x 28 x 28 x 32

= 10 million

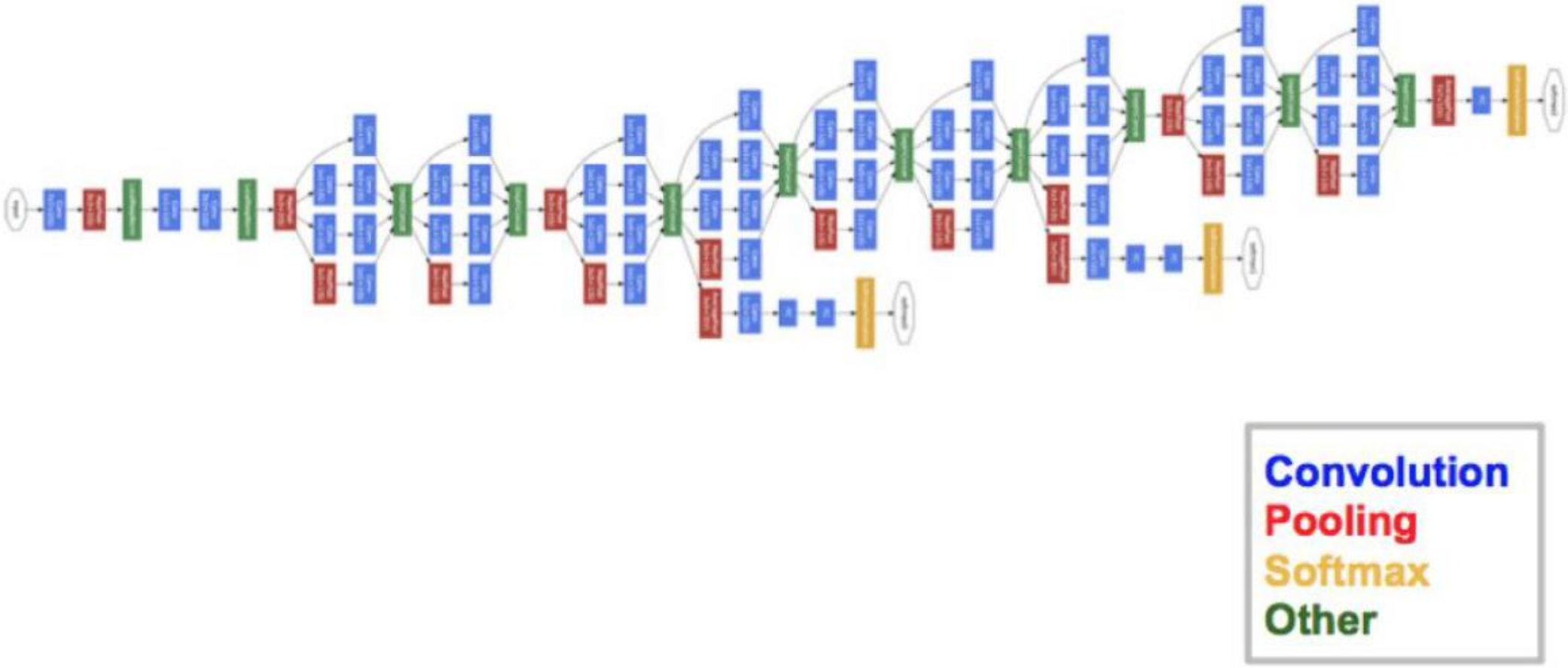
Total = 12.4 million!

compared to 120 million.

And channels reduced from 192 to 32

GoogleNet (Inception Network)

- Go Deeper with convolutions

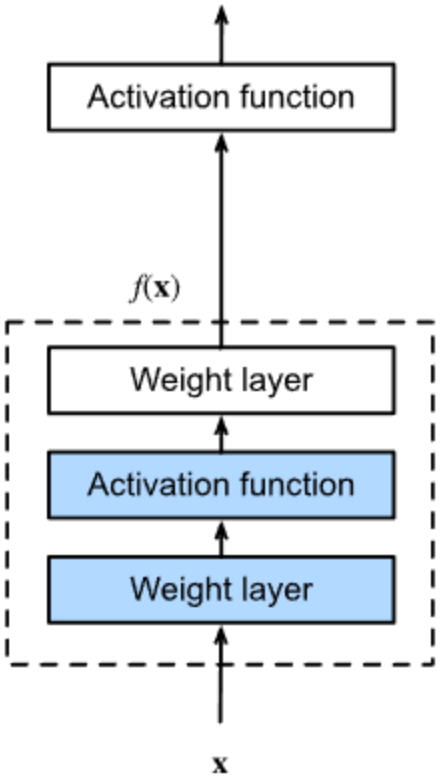


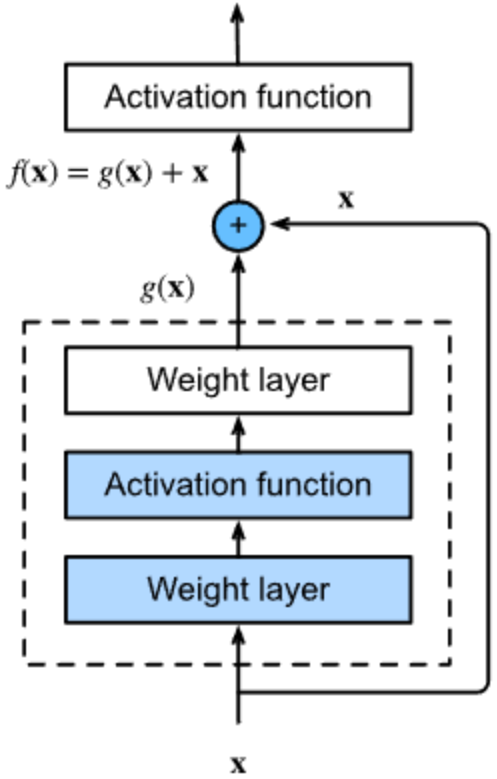
Residual Networks (ResNet) and ResNeXt

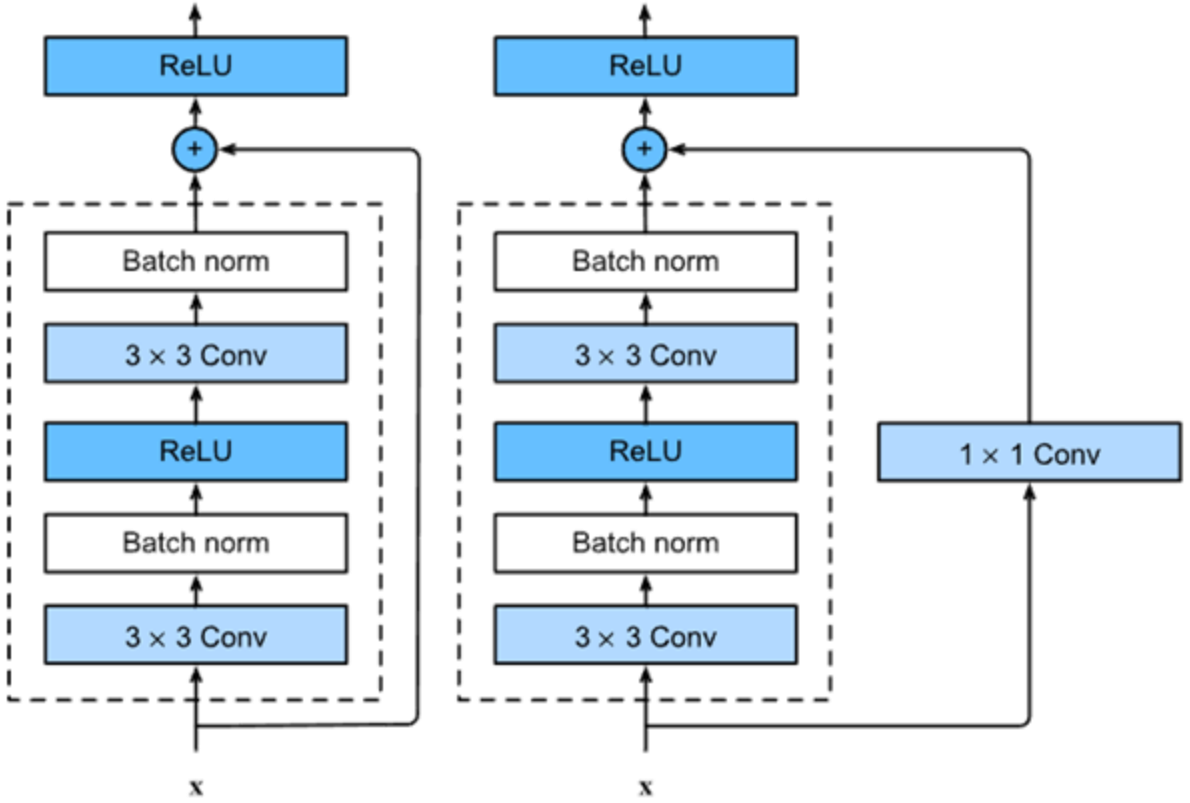
- Problem with Deep Neural Networks
  - Vanishing or exploding gradients

Residual Networks (ResNet) and ResNeXt

- Plain
- Residual block
- Residual block without and with 1 x 1 convolution (2 different architectures)



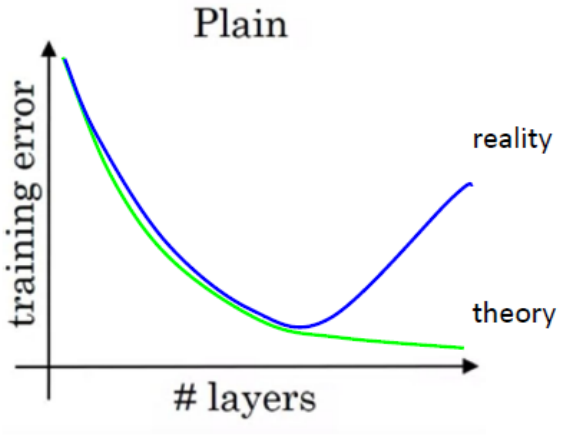


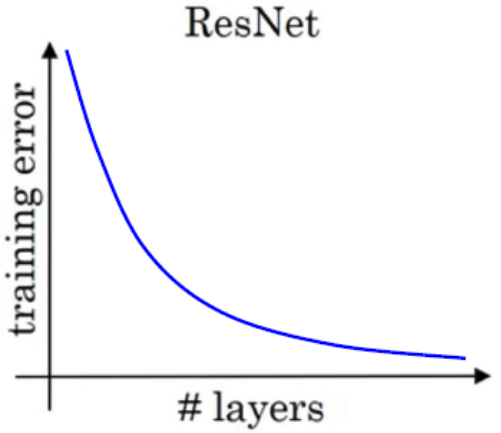


https://d2l.ai/chapter_convolutional-modern/resnet.html

Residual Networks (ResNet) and ResNeXt

- Why use ResNets

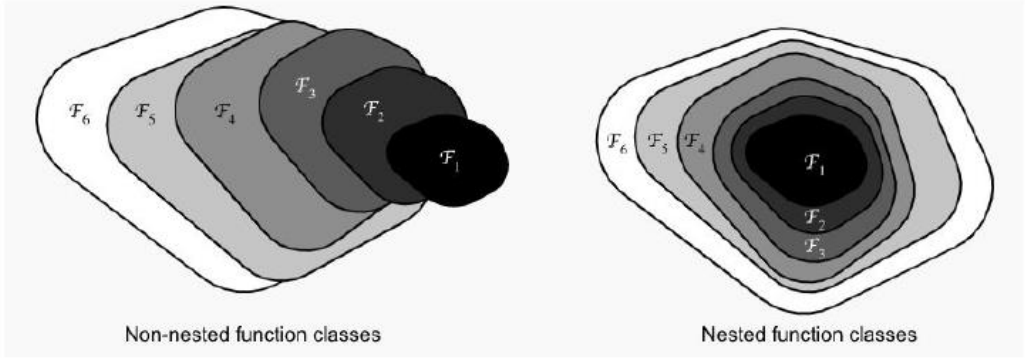




> 100

Residual Networks (ResNet) and ResNeXt

- Three possible reasons
  - It allows the network to more easily learn identity functions, that is to propagate activations from one part of the network to upstream nodes unchanged.
  - Learning identity functions is like finding nested function classes rather than non-nested function classes. The former is more likely to find the global minimum compared to the latter.
  - Inputs can forward propagate faster through the residual connections.

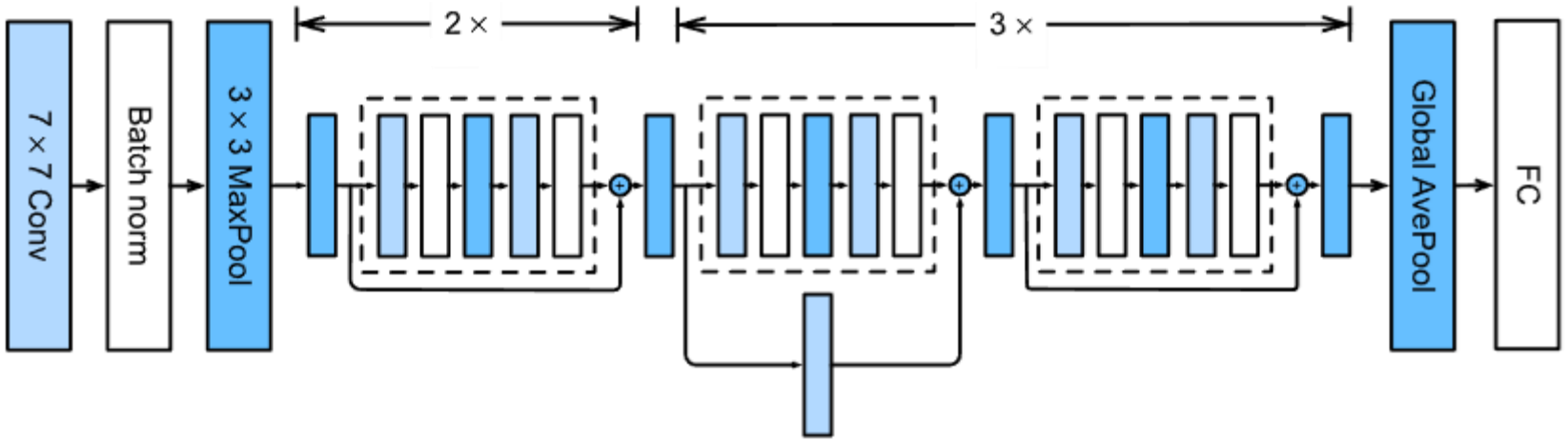


https://d2l.ai/chapter_convolutional-modern/resnet.html

- As a consequence, we can train deeper Neural Networks

Residual Networks (ResNet) and ResNeXt

- ResNet-18 architecture (18 layers)

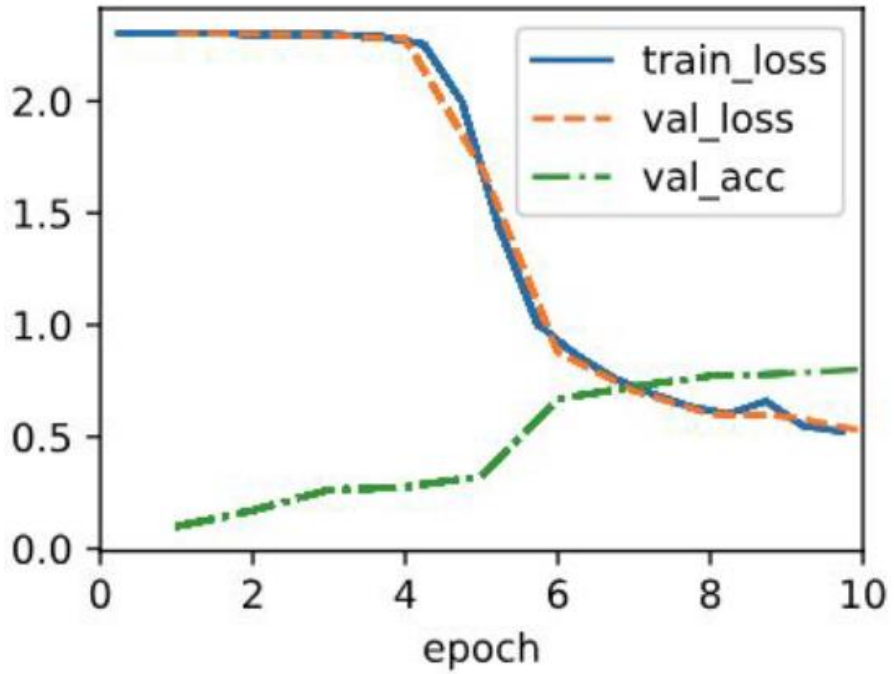



```
Sequential output shape:     torch.Size([1, 64, 24, 24])
Sequential output shape:     torch.Size([1, 64, 24, 24])
Sequential output shape:     torch.Size([1, 128, 12, 12])
Sequential output shape:     torch.Size([1, 256, 6, 6])
Sequential output shape:     torch.Size([1, 512, 3, 3])
Sequential output shape:     torch.Size([1, 10])
```


https://d2l.ai/chapter_convolutional-modern/resnet.html

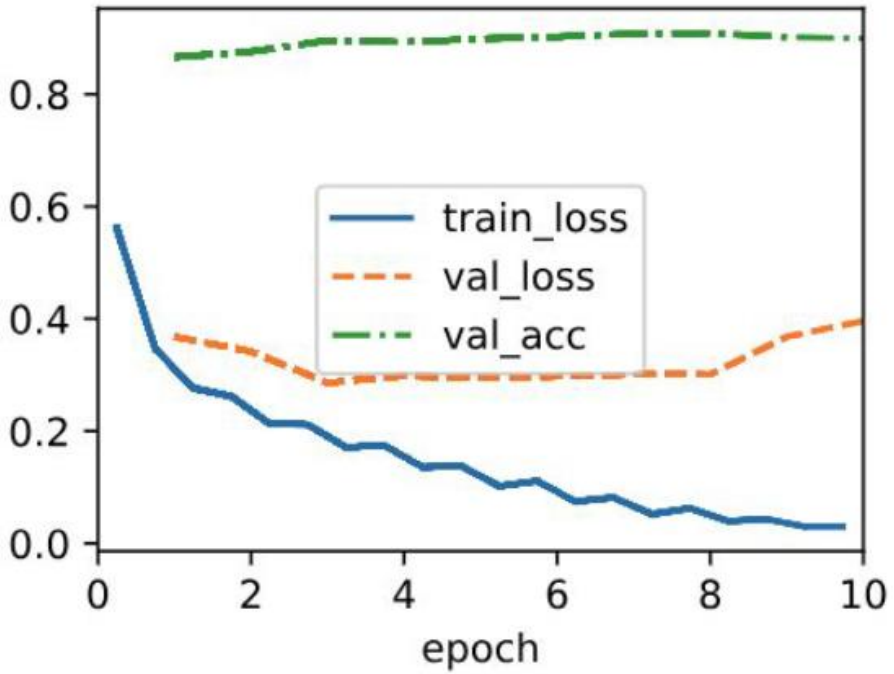
Residual Networks (ResNet) and ResNeXt – Performance

- Using Fashion-MNIST dataset



GoogleNet

2014, *GoogLeNet* won the ImageNet Challenge



RestNet

2015, *RestNet* won the ImageNet Challenge

Residual Networks (ResNet) and ResNeXt

- Effects of these changes
  - Layers
    - Adding Layers increases non-linearity.
  - Channels
    - Adding Channels increases more information carried forward.
    - Naive: Each channel acts as an independent feature detector
    - Multiple channels allow the CNN to reason with multiple features such as edge and shape at the same time

But we can see that as

Layers increase (in number or in width)

the number of channels decrease.

and vice versa.

Residual Networks (ResNet) and ResNeXt

- ResNeXt

With g groups,

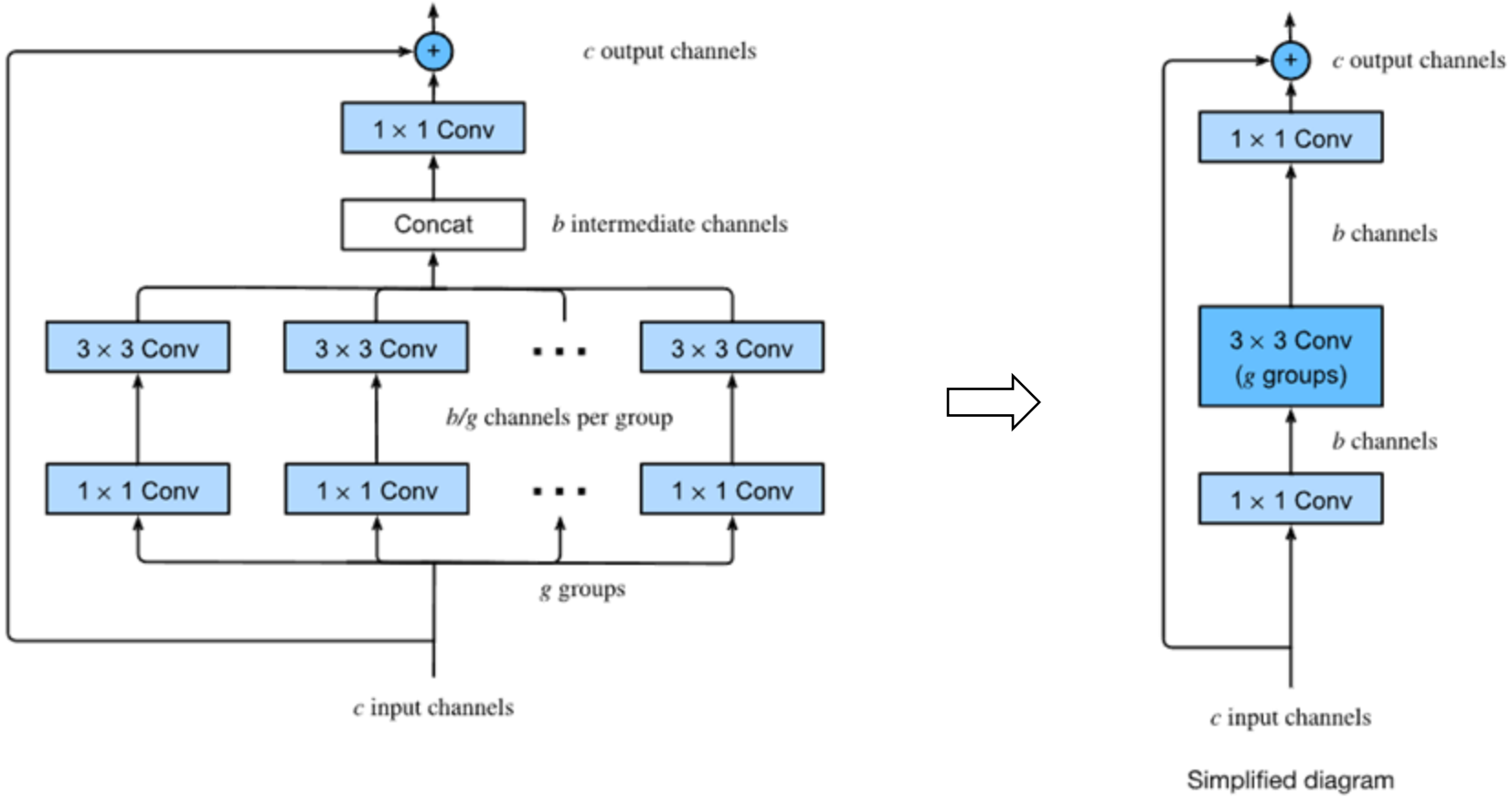
Computational Complexity is reduced from

$$O(c_i c_o)$$

to

$$O(c_i c_o / g)$$

ie g times faster with g times less multiplications.

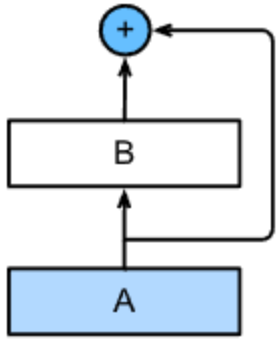


https://d2l.ai/chapter_convolutional-modern/resnet.html

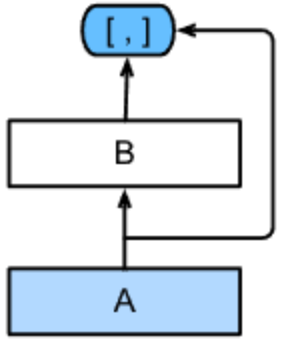
Densely Connected Networks (DenseNet)

With inspiration from Taylor expansion for functions

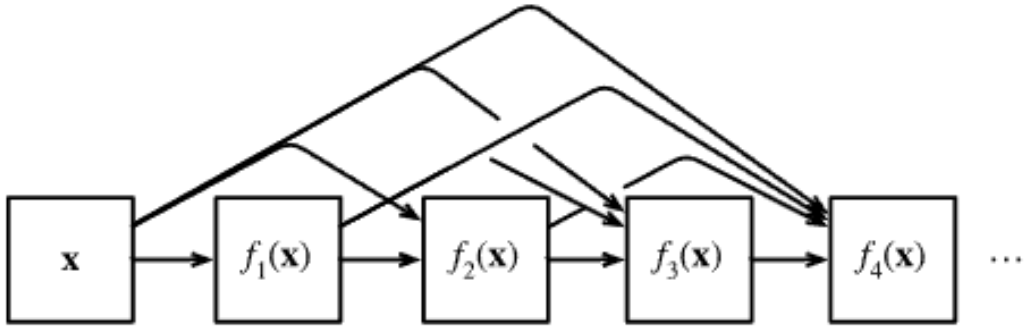
$$f(x) = f(0) + x \cdot \left[f'(0) + x \cdot \left[\frac{f''(0)}{2!} + x \cdot \left[\frac{f'''(0)}{3!} + \ldots \right] \right] \right].$$



ResNet block
uses addition



DenseNet block
uses concatenation



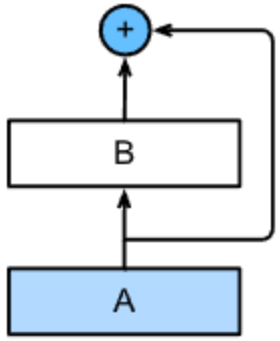
DenseNet

https://d2l.ai/chapter_convolutional-modern/densenet.html

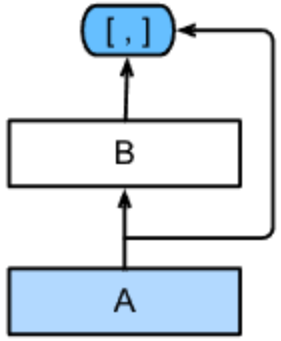
Densely Connected Networks (DenseNet)

With inspiration from Taylor expansion for functions

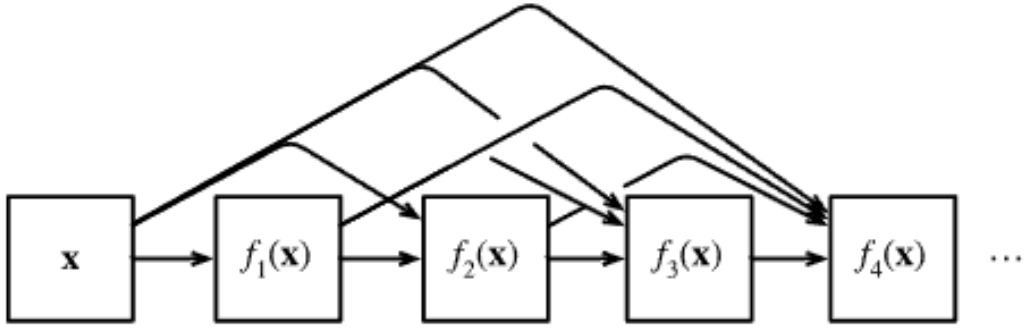
$$\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots].$$



ResNet block
uses addition



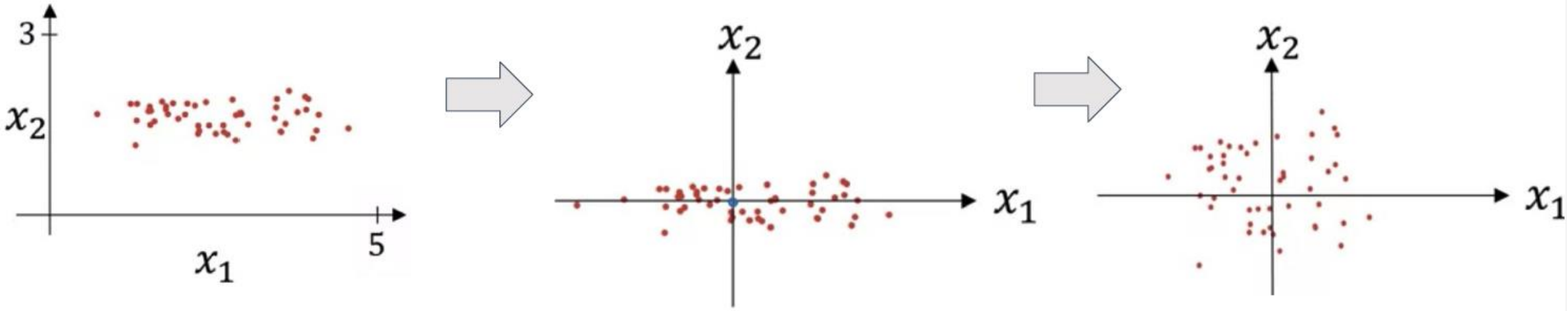
DenseNet block
uses concatenation



DenseNet

https://d2l.ai/chapter_convolutional-modern/densenet.html

Normalizing the Inputs



$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\mathrm{x} = \mathrm{x} - \mu$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x_i^2$$

$$\mathrm{x} \mathrel{/}= \sigma$$

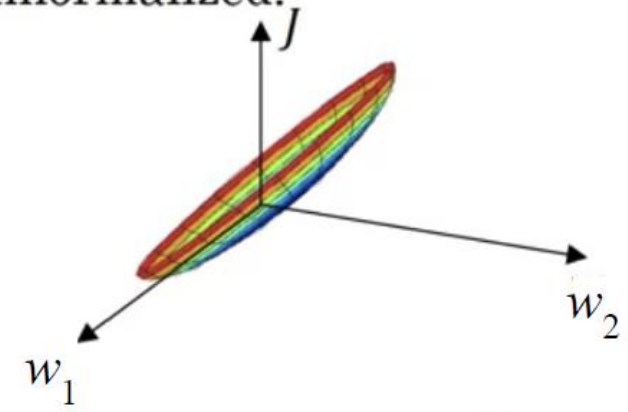
Normalizing the Inputs – Why is needed?

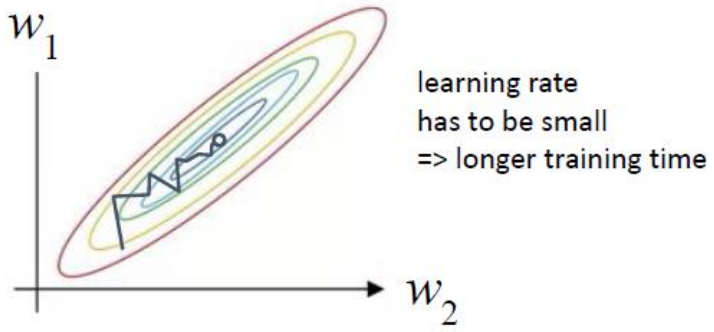
$w_1 \quad x_1 : 1...1000$

$w_2 \quad x_2 : 0...1$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

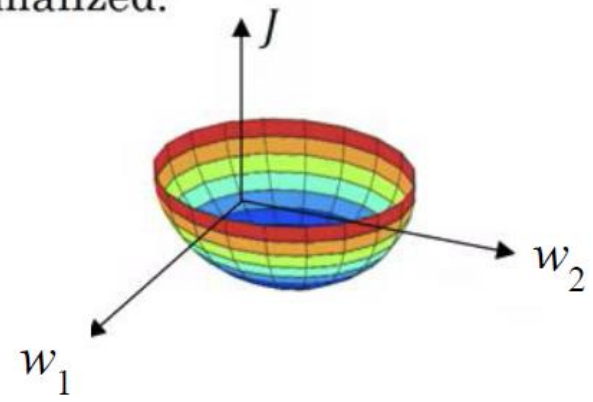
Unnormalized:

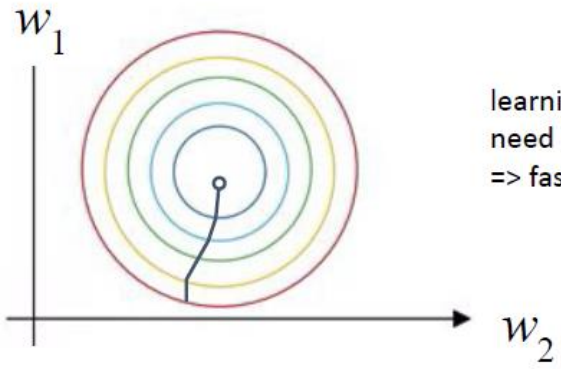




learning rate
has to be small
=> longer training time

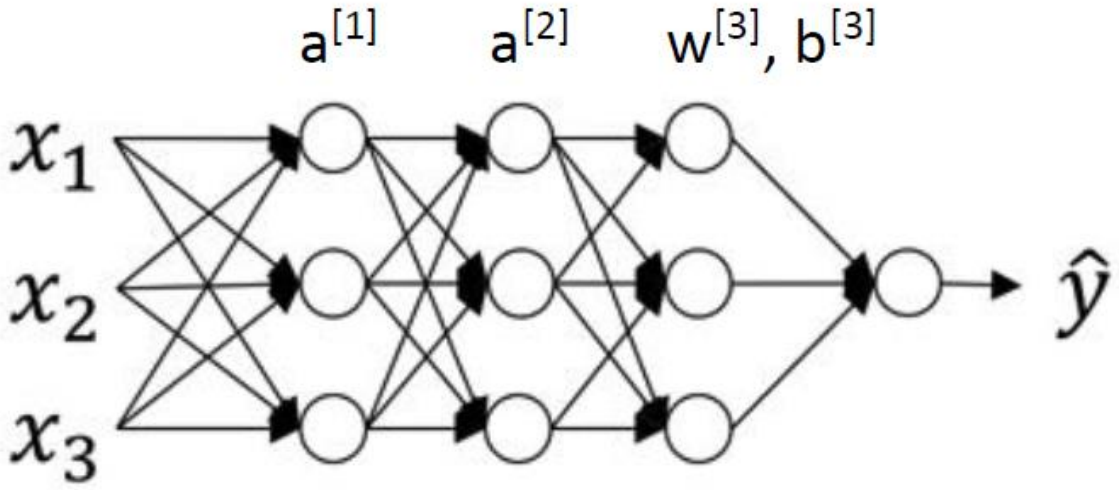
Normalized:





learning rate
need not be small
=> faster training time

Normalizing the Inputs – How about a deeper network?



Can normalize $a^{[2]}$ so as to train $w^{[3]}, b^{[3]}$ faster?

Preprocessing – Batch Normalization

- Issues
  - Training deep NN is difficult – they are slow to converge
  - Variables in intermediate layers may take values with widely varying magnitude
  - Deeper networks are complex and tend to overfit
- Batch Normalization is used to prevent the problem from occurring
  - Batch normalization is an effective technique used to accelerate the convergence of deep NN training
  - Applied to individual layers, or optionally, to all of them
  - At each training iteration
    - Normalize the inputs (by subtracting their mean and dividing by the standard deviation, estimated from the current batch)
    - Apply a scale coefficient and an offset
  - The choice of the batch size is very important

Preprocessing – Batch Normalization Layers

- For Fully Connected layers
  - Early works, batch normalization is inserted after the linear (affine) transform and before the nonlinear activation function
  - Later, other works also explored inserting batch normalization after the nonlinear activation
- Convolutional Layers
  - Can be applied before or after the nonlinear activation
  - The operation is applied for every channel of the output across all locations
- Layer Normalization
  - Similar to batch normalization but applied to one observation at a time

Preprocessing – Batch Normalization Layers

- LeNet with batch normalization
  - Each layer is followed by a batch normalization layer, before the linear activation

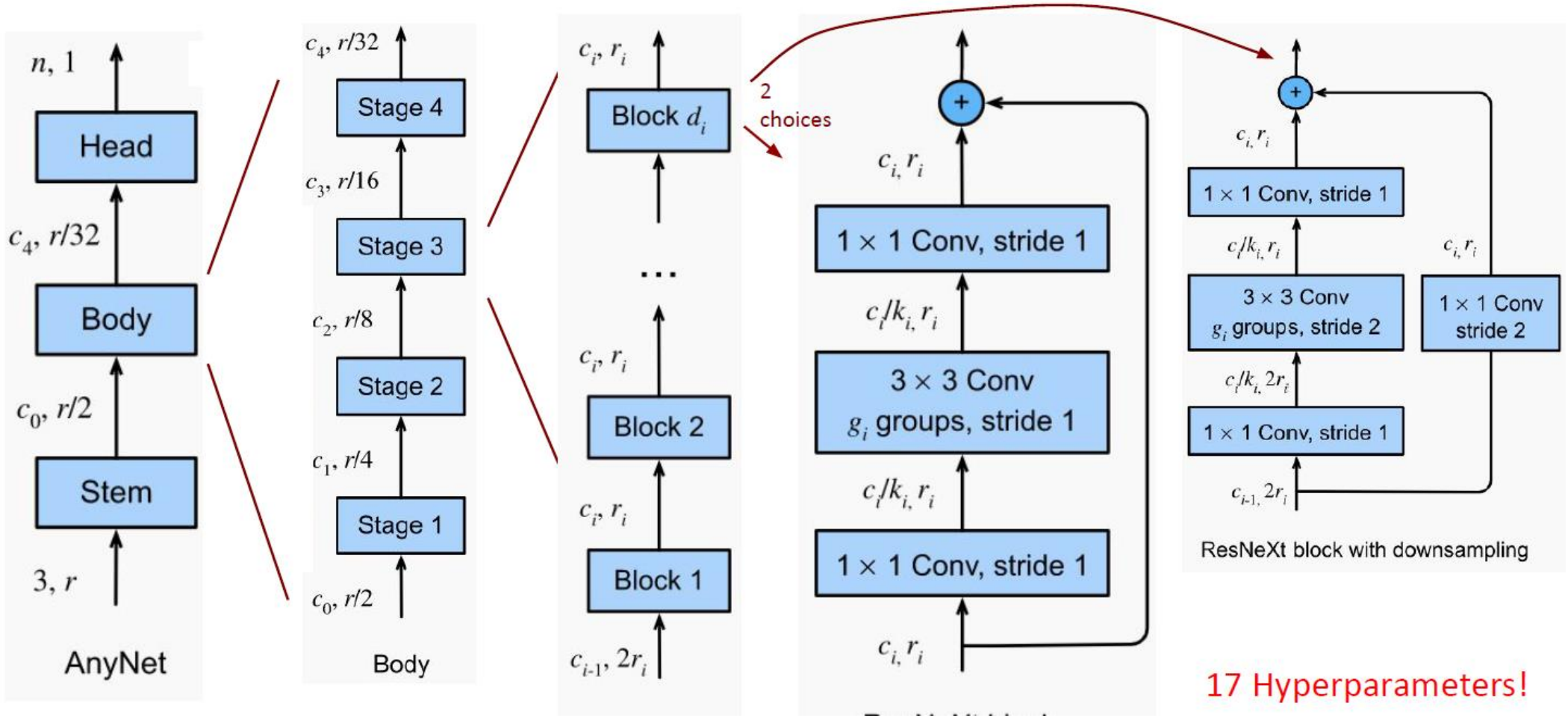
```
self.net = nn.Sequential(
    nn.LazyConv2d(6, kernel_size=5), nn.LazyBatchNorm2d(),
    nn.Sigmoid(), nn.AvgPool2d(kernel_size=2, stride=2),
    nn.LazyConv2d(16, kernel_size=5), nn.LazyBatchNorm2d(),
    nn.Sigmoid(), nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(), nn.LazyLinear(120), nn.LazyBatchNorm1d(),
    nn.Sigmoid(), nn.LazyLinear(84), nn.LazyBatchNorm1d(),
    nn.Sigmoid(), nn.LazyLinear(num_classes))
```

  - Notice that Batch Norm comes before Sigmoid, and if we normalize to mean 0 and standard deviation 1, is this a problem? see https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

Automating the Architecture Design

- The whole problem now became “how to design the best architecture” for a specific task
- Neural Architecture Search (NAS)
  - The process of automating neural network architectures
  - Given a search space, NAS uses a search strategy to automatically select an architecture within the search space based on the returned performance estimation.
  - The outcome of NAS is a single network instance.

Design Space for Deep CNNs



Searching for a Good Design

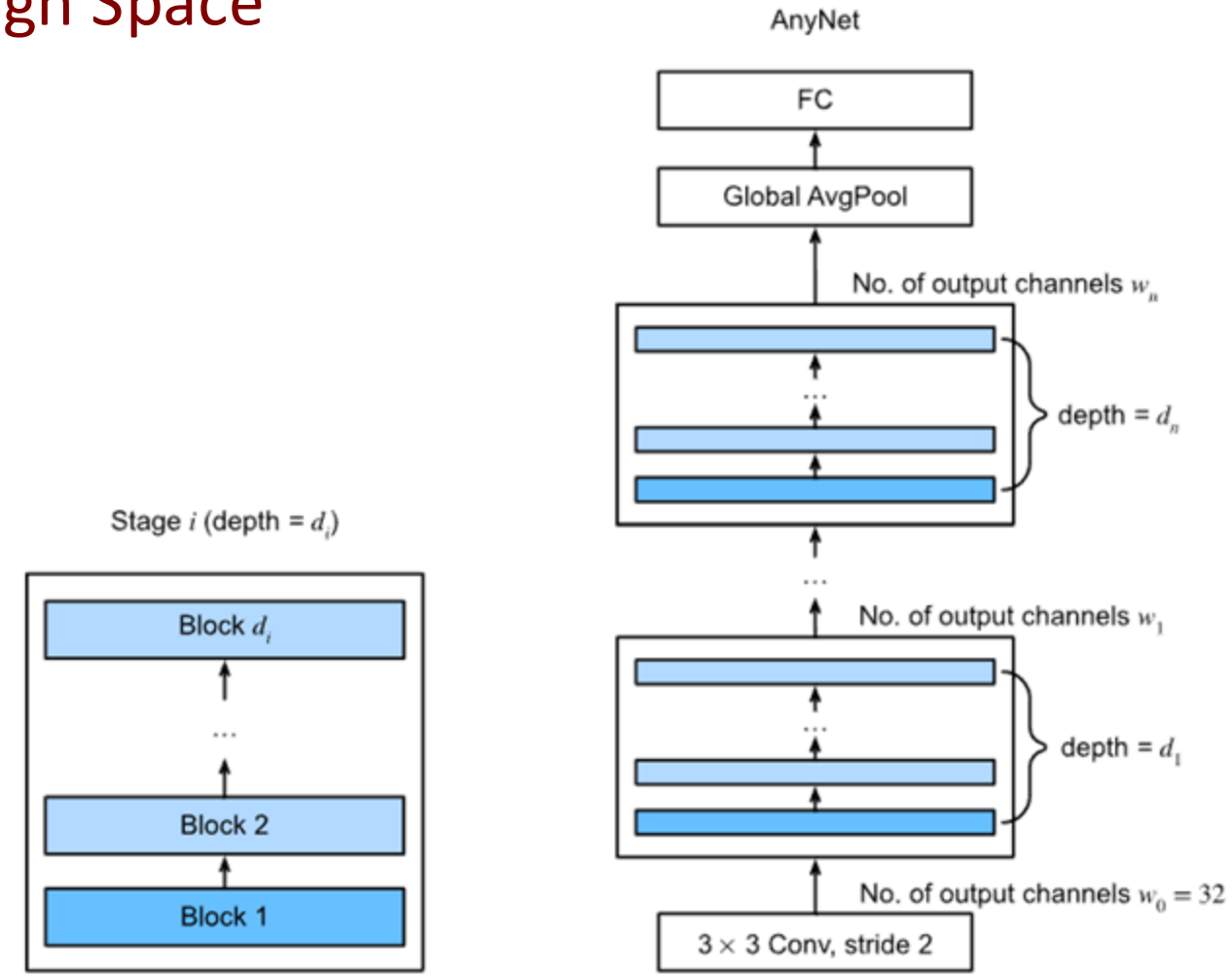
2^{17} = 131 072 possibilities

Three Problems:

- Huge space to search
- Each change in the result would require a completely new search
- No two runs are completely the same (due to inherent stochasticity)

Automating the Architecture Design

- The AnyNet Design Space

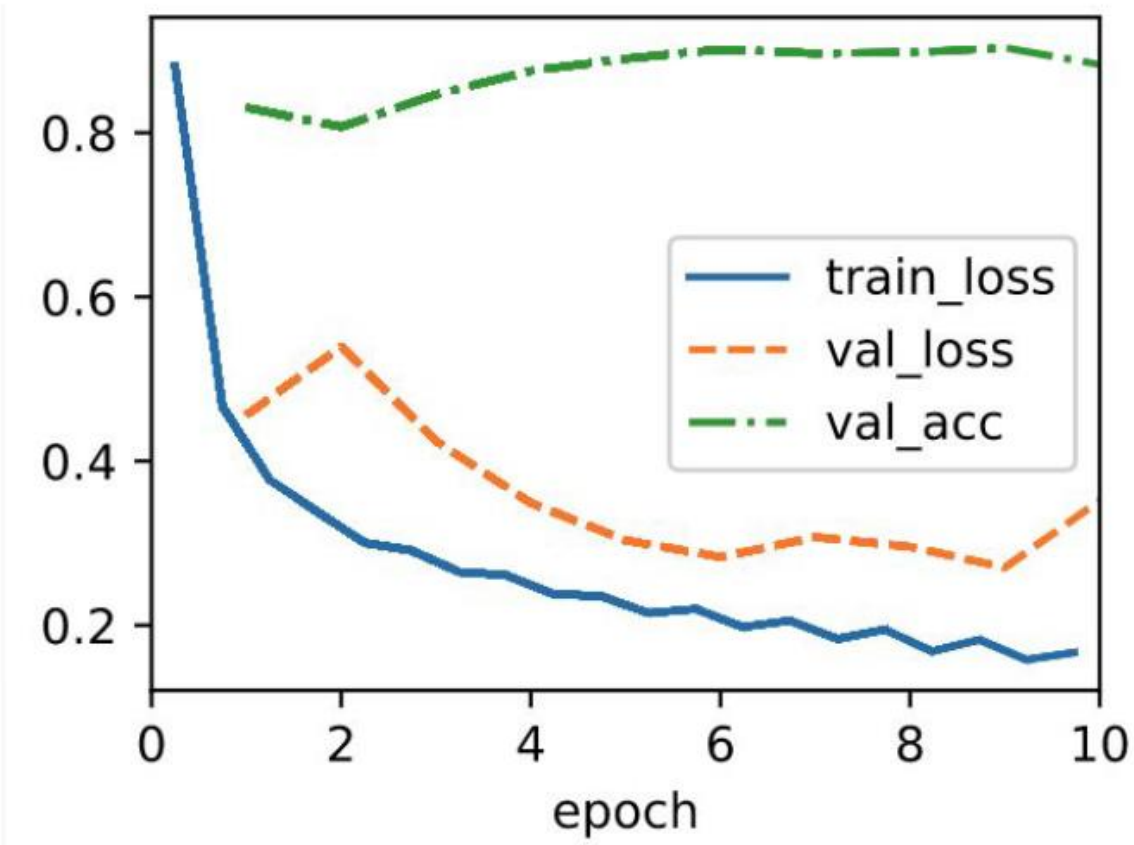


Result of Design Search

- Share the bottleneck ratio $k_i = k$ for all stages i;
- Share the group width $g_i = g$ for all stages i;
- Increase network width across stages: $c_i \leq c_{i+1}$;
- Increase network depth across stages: $d_i \leq d_{i+1}$.

For example:

RegNetX used k=1, g=16, c_1=32, c_2=80, d_1=4 and d_2=6



Summary

- Many deep neural networks, each one is designed to address the limitations of others
- In the lab
  - You will create and train some of these networks (not all of them!)
- Next week
  - No formal lecture – we will focus on finishing the labs
  - I will be present during the lecture session, but there are no formal slides
    - I will answer any questions you may have about previous topics and labs
    - Focus on labs and Assignment 1

Questions