7.6. Residual Networks (ResNet)

:label: `sec_resnet`

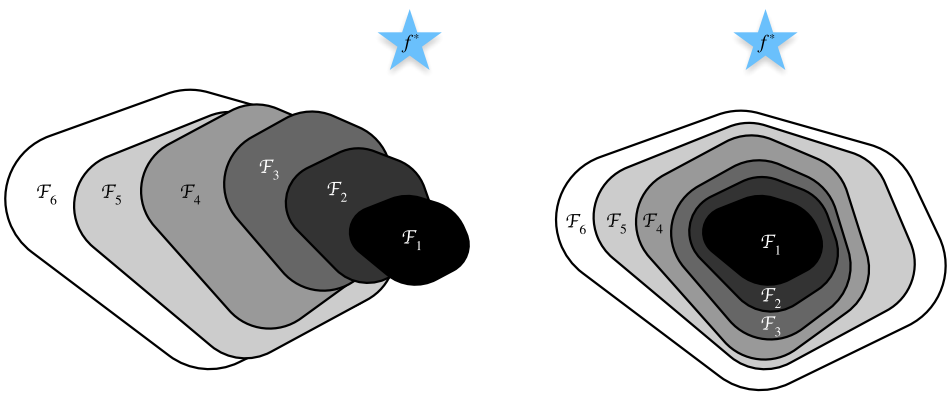
As we design increasingly deeper networks it becomes imperative to understand how adding layers can increase the complexity and expressiveness of the network. Even more important is the ability to design networks where adding layers makes networks strictly more expressive rather than just different. To make some progress we need a bit of mathematics.

7.6.1. Function Classes

Consider $\mathcal{F}$, the class of functions that a specific network architecture (together with learning rates and other hyperparameter settings) can reach. That is, for all $f \in \mathcal{F}$ there exists some set of parameters (e.g., weights and biases) that can be obtained through training on a suitable dataset. Let us assume that f^* is the "truth" function that we really would like to find. If it is in $\mathcal{F}$, we are in good shape but typically we will not be quite so lucky. Instead, we will try to find some $f^*_\mathcal{F}$ which is our best bet within $\mathcal{F}$. For instance, given a dataset with features $\mathbf{X}$ and labels $\mathbf{y}$, we might try finding it by solving the following optimization problem:

$$f^*_\mathcal{F} \stackrel{\mathrm{def}}{=} \operatorname*{argmin}_f L(\mathbf{X}, \mathbf{y}, f) \text{ subject to } f \in \mathcal{F}.$$

It is only reasonable to assume that if we design a different and more powerful architecture $\mathcal{F}'$ we should arrive at a better outcome. In other words, we would expect that $f^*_{\mathcal{F}'}$ is "better" than $f^*_\mathcal{F}$. However, if $\mathcal{F} \not\subseteq \mathcal{F}'$ there is no guarantee that this should even happen. In fact, $f^*_{\mathcal{F}'}$ might well be worse. As illustrated by :numref: `fig_functionclasses` , for non-nested function classes, a larger function class does not always move closer to the "truth" function f^*. For instance, on the left of :numref: `fig_functionclasses` , though $\mathcal{F}_3$ is closer to f^* than $\mathcal{F}_1$, $\mathcal{F}_6$ moves away and there is no guarantee that further increasing the complexity can reduce the distance from f^*. With nested function classes where $\mathcal{F}_1 \subseteq \ldots \subseteq \mathcal{F}_6$ on the right of :numref: `fig_functionclasses` , we can avoid the aforementioned issue from the non-nested function classes.



Non-nested function classes Nested function classes

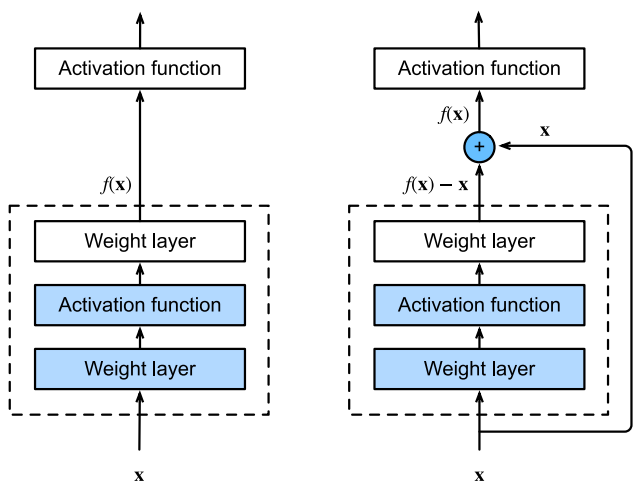
:label: `fig_functionclasses`

Thus, only if larger function classes contain the smaller ones are we guaranteed that increasing them strictly increases the expressive power of the network. For deep neural networks, if we can train the newly-added layer into an identity function $f(\mathbf{x}) = \mathbf{x}$, the new model will be as effective as the original model. As the new model may get a better solution to fit the training dataset, the added layer might make it easier to reduce training errors.

This is the question that He et al. considered when working on very deep computer vision models :cite: `He.Zhang.Ren.ea.2016` . At the heart of their proposed *residual network* (*ResNet*) is the idea that every additional layer should more easily contain the identity function as one of its elements. These considerations are rather profound but they led to a surprisingly simple solution, a *residual block*. With it, ResNet won the ImageNet Large Scale Visual Recognition Challenge in 2015. The design had a profound influence on how to build deep neural networks.

7.6.2. Residual Blocks

Let us focus on a local part of a neural network, as depicted in :numref: `fig_residual_block` . Denote the input by $\mathbf{x}$. We assume that the desired underlying mapping we want to obtain by learning is $f(\mathbf{x})$, to be used as the input to the activation function on the top. On the left of :numref: `fig_residual_block` , the portion within the dotted-line box must directly learn the mapping $f(\mathbf{x})$. On the right, the portion within the dotted-line box needs to learn the *residual mapping* $f(\mathbf{x}) - \mathbf{x}$, which is how the residual block derives its name. If the identity mapping $f(\mathbf{x}) = \mathbf{x}$ is the desired underlying mapping, the residual mapping is easier to learn: we only need to push the weights and biases of the upper weight layer (e.g., fully-connected layer and convolutional layer) within the dotted-line box to zero. The right figure in :numref: `fig_residual_block` illustrates the *residual block* of ResNet, where the solid line carrying the layer input $\mathbf{x}$ to the addition operator is called a *residual connection* (or *shortcut connection*). With residual blocks, inputs can forward propagate faster through the residual connections across layers.



:label: `fig_residual_block`

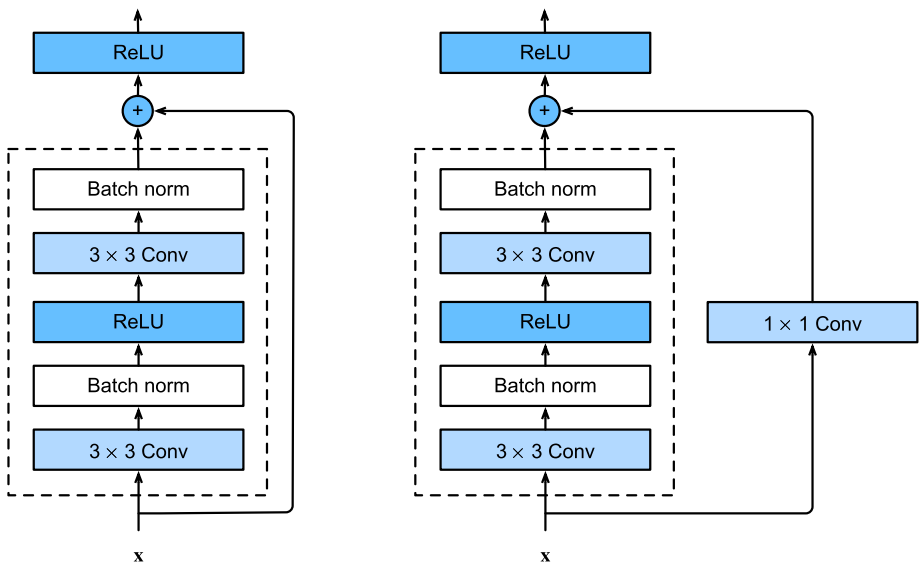
ResNet follows VGG's full 3×3 convolutional layer design. The residual block has two 3×3 convolutional layers with the same number of output channels. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. Then, we skip these two convolution operations and add the input directly before the final ReLU activation function. This kind of design requires that the output of the two convolutional layers has to be of the same shape as the input, so that they can be added together. If we want to change the number of channels, we need to introduce an additional 1×1 convolutional layer to transform the input into the desired shape for the addition operation. Let us have a look at the code below.

In [1]:
```
use strict;
use warnings;
use Data::Dump qw(dump);
use AI::MXNet qw(mx);
use d2l;
use d2l::Accumulator;
use d2l::Animator;
use d2l::Timer;
IPerl->load_plugin('Chart::Plotly'); # Jupyter
# use Chart::Plotly qw(show_plot); # Locally
```

In [2]:
```
package Residual{
    use strict;
    use warnings;
    use Data::Dump qw(dump);
    use AI::MXNet qw(mx);
    use base qw(AI::MXNet::Gluon::Block);

    sub new {#@save
      my ($class, $num_channels, $use_1x1conv, $strides, %args) = (shift, @_);
      if ( !defined ($use_1x1conv)) {
          $use_1x1conv = 0;
      }
      if ( !defined ($strides)) {
          $strides = 1;
      }
      my $self = $class->SUPER::new(%args);
      $self->{conv1} = mx->gluon->nn->Conv2D($num_channels, kernel_size => 3, padding => 1, strides => $strides);
      $self->{conv2} = mx->gluon->nn->Conv2D($num_channels, kernel_size => 3, padding => 1);
      if ($use_1x1conv) {
          $self->{conv3} = mx->gluon->nn->Conv2D($num_channels, kernel_size => 1, strides => $strides);
      }else {
          $self->{conv3} = undef;
      }

      $self->{bn1} = mx->gluon->nn->BatchNorm();
      $self->{bn2} = mx->gluon->nn->BatchNorm();
      # Initialization of each layer
      foreach my $name ('conv1', 'conv2', 'conv3', 'bn1', 'bn2'){
          if (defined ($self->{$name}) ) {
              $self->register_child($self->{$name}) ;
          }
       }
       return bless($self, $class);
    }

    sub forward {
        my ($self, $X) = @_;
        my $Y = mx->nd->relu($self->{bn1}->forward($self->{conv1}->forward($X)));
        $Y = $self->{bn2}->forward($self->{conv2}->forward($Y));
        if ( defined ($self->{conv3}) ) {
            $X = $self->{conv3}->forward($X);
        }
        return mx->nd->relu($Y + $X);
    }

    1;

}
```

Out[2]: 1

This code generates two types of networks: one where we add the input to the output before applying the ReLU nonlinearity whenever `use_1x1conv=False`, and one where we adjust channels and resolution by means of a 1×1 convolution before adding. :numref: `fig_resnet_block` illustrates this:



:label: `fig_resnet_block`

Now let us look at [**a situation where the input and output are of the same shape**].

In [3]:
```
my $blk = Residual->new(3);
$blk->initialize();
my $X = mx->nd->random->uniform(shape => [4, 3, 6, 6]);
print dump $blk->($X)->shape;
```

```
[4, 3, 6, 6]
```

Out[3]: 1

We also have the option to [**halve the output height and width while increasing the number of output channels**].

In [4]:
```
$blk = Residual->new (6, 1, 2);
$blk->initialize();
print dump $blk->($X)->shape;
```

```
[4, 6, 3, 3]
```

Out[4]: 1

7.6.3 ResNet Model

The first two layers of ResNet are the same as those of the GoogLeNet we described before: the 7×7 convolutional layer with 64 output channels and a stride of 2 is followed by the 3×3 maximum pooling layer with a stride of 2. The difference is the batch normalization layer added after each convolutional layer in ResNet.

In [5]:
```
my $net = mx->gluon->nn->Sequential();
$net->add(mx->gluon->nn->Conv2D(64, kernel_size => 7, strides => 2, padding => 3),
        mx->gluon->nn->BatchNorm(), mx->gluon->nn->Activation('relu'),
        mx->gluon->nn->MaxPool2D(pool_size => 3, strides => 2, padding => 1))
```

GoogLeNet uses four modules made up of Inception blocks. However, ResNet uses four modules made up of residual blocks, each of which uses several residual blocks with the same number of output channels. The number of channels in the first module is the same as the number of input channels. Since a maximum pooling layer with a stride of 2 has already been used, it is not necessary to reduce the height and width. In the first residual block for each of the subsequent modules, the number of channels is doubled compared with that of the previous module, and the height and width are halved.

Now, we implement this module. Note that special processing has been performed on the first module.

In [6]:
```
sub resnet_block {
    my ($num_channels, $num_residuals, $first_block) = @_;
    if ( !defined ($first_block)) {
          $first_block = 0;
      }
    my $blk = mx->gluon->nn->Sequential();
    for my $i (0 .. $num_residuals){
        if ($i == 0 and !$first_block){
            $blk->add(Residual->new($num_channels, 1, 2));
        }else {
            $blk->add(Residual->new($num_channels));
        }
    }
    return $blk;
}
```

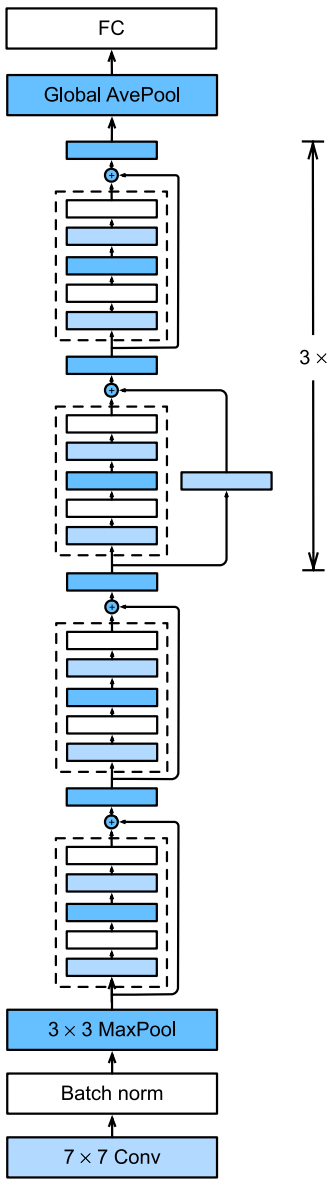
Then, we add all the modules to ResNet. Here, two residual blocks are used for each module.

In [7]:
```
$net->add(resnet_block(64, 2, 1),
        resnet_block(128, 2),
        resnet_block(256, 2),
        resnet_block(512, 2));
```

Finally, just like GoogLeNet, we add a global average pooling layer, followed by the fully-connected layer output.

In [8]:
```
$net->add(mx->gluon->nn->GlobalAvgPool2D(), mx->gluon->nn->Dense(10));
```

There are 4 convolutional layers in each module (excluding the 1×1 convolutional layer). Together with the first 7×7 convolutional layer and the final fully-connected layer, there are 18 layers in total. Therefore, this model is commonly known as ResNet-18. By configuring different numbers of channels and residual blocks in the module, we can create different ResNet models, such as the deeper 152-layer ResNet-152. Although the main architecture of ResNet is similar to that of GoogLeNet, ResNet's structure is simpler and easier to modify. All these factors have resulted in the rapid and widespread use of ResNet. :numref: `fig_resnet18` depicts the full ResNet-18.



:label: `fig_resnet18`

Before training ResNet, let us [**observe how the input shape changes across different modules in ResNet**]. As in all the previous architectures, the resolution decreases while the number of channels increases up until the point where a global average pooling layer aggregates all features.

In [9]:
```
$X = mx->nd->random->uniform(shape => [1, 1, 224, 224]);
$net->initialize(mx->init->Xavier(), force_reinit=>1);
foreach my $layer (@$net) {
    $X = $layer->forward($X);
    print $layer->name, " output shape:\t", dump ($X->shape), "\n";
}
```

```
conv5 output shape:     [1, 64, 112, 112]
batchnorm4 output shape:        [1, 64, 112, 112]
relu0 output shape:     [1, 64, 112, 112]
pool0 output shape:     [1, 64, 56, 56]
sequential1 output shape:       [1, 64, 56, 56]
sequential2 output shape:       [1, 128, 28, 28]
sequential3 output shape:       [1, 256, 14, 14]
sequential4 output shape:       [1, 512, 7, 7]
pool1 output shape:     [1, 512, 1, 1]
dense0 output shape:    [1, 10]
```

7.6.4. Training

We train ResNet on the Fashion-MNIST dataset, just like before.

In [*]:
```
my ($lr, $num_epochs, $batch_size) = (0.05, 10, 256);
my ($train_iter, $test_iter) = d2l->load_data_fashion_mnist(batch_size => $batch_size, resize => 96);
my ($model_file_name, $is_train, $animator) = ('GoogLeNet.mdl', 1);
if ($is_train){
  $animator = d2l->train_ch6($net, $train_iter, $test_iter, $num_epochs, $lr, d2l->try_gpu());
  $net->save_parameters($model_file_name);
  $animator->plot;
}else{
  $net->load_parameters($model_file_name);
}
#3. Debido a que las imágenes en tamaño 224 x 224 son demasiado pesadas para avanzar con el procesamiento,  no hace falta
#que se ejecute el entrenamiento del modelo, ya que el entrenamiento de un único modelo ocuparía todos recursos de cpu
#y memoria del servidor Jupyter. La revisión y calificación del notebook entregado se realizará mediante la comparación
#con las demás  lineas de codigo del mismo.
#Da lo mismo usar la funcion
```

Summary

- Nested function classes are desirable. Learning an additional layer in deep neural networks as an identity function (though this is an extreme case) should be made easy.
- The residual mapping can learn the identity function more easily, such as pushing parameters in the weight layer to zero.
- We can train an effective deep neural network by having residual blocks. Inputs can forward propagate faster through the residual connections across layers.
- ResNet had a major influence on the design of subsequent deep neural networks, both for convolutional and sequential nature.

Exercises

1. What are the major differences between the Inception block in :numref: `fig_inception` and the residual block? After removing some paths in the Inception block, how are they related to each other?
2. Refer to Table 1 in the ResNet paper :cite: `He.Zhang.Ren.ea.2016` to implement different variants.
3. For deeper networks, ResNet introduces a "bottleneck" architecture to reduce model complexity. Try to implement it.
4. In subsequent versions of ResNet, the authors changed the "convolution, batch normalization, and activation" structure to the "batch normalization, activation, and convolution" structure. Make this improvement yourself. See Figure 1 in :cite: `He.Zhang.Ren.ea.2016*1` for details.
5. Why can't we just increase the complexity of functions without bound, even if the function classes are nested?