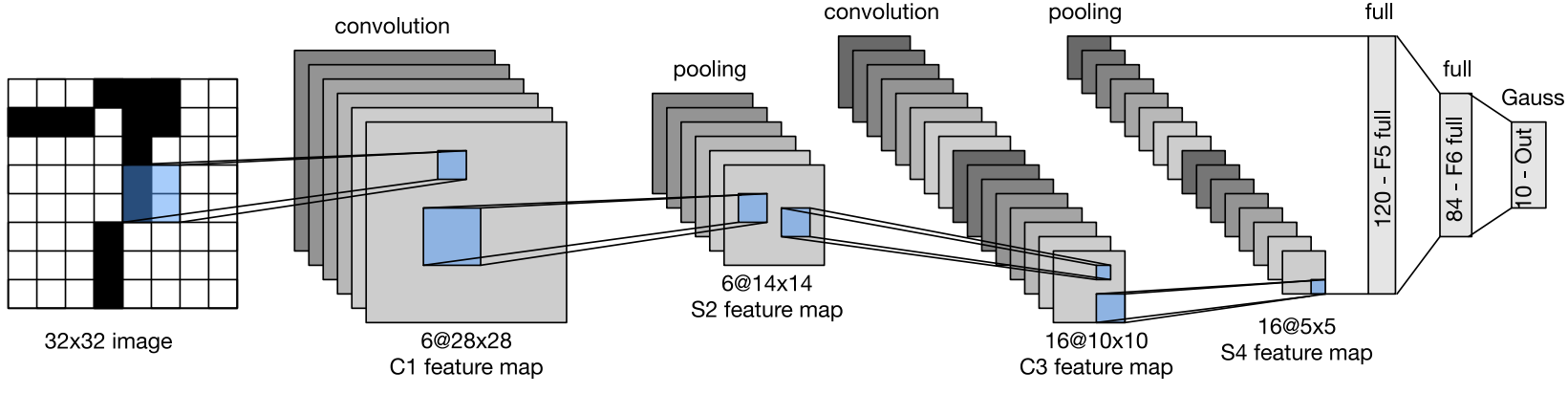
LeNet


convolution
pooling
convolution
pooling
full
full
Gauss
120 - F5 full
84 - F6 full
10 - Out
32x32 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map

In [1]:

```
import d2l
import mxnet as mx
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import loss as gloss, nn
import time

net = nn.Sequential()
net.add(nn.Conv2D(channels=6, kernel_size=5, padding=2, activation='sigmoid'),
        nn.AvgPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=5, activation='sigmoid'),
        nn.AvgPool2D(pool_size=2, strides=2),
        # Dense will transform the input of the shape (batch size, channel, heigh
t, width)
        # into the input of the shape (batch size, channel *height * width) automa
tically
        # by default.
        nn.Dense(120, activation='sigmoid'),
        nn.Dense(84, activation='sigmoid'),
        nn.Dense(10))
```

Feeding a single observation through the network

In [2]:
```
X = nd.random.uniform(shape=(1, 1, 28, 28))
net.initialize()
for layer in net:
    X = layer(X)
    print(layer.name, 'output shape:\t', X.shape)
```

```
conv0 output shape:	 (1, 6, 28, 28)
pool0 output shape:	 (1, 6, 14, 14)
conv1 output shape:	 (1, 16, 10, 10)
pool1 output shape:	 (1, 16, 5, 5)
dense0 output shape:	 (1, 120)
dense1 output shape:	 (1, 84)
dense2 output shape:	 (1, 10)
```

Data and training

In [3]:
```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size=batch_size)
```

In [4]:
```
# use a GPU if we have it
def try_gpu():
    try:
        ctx = mx.gpu()
        _ = nd.zeros((1,), ctx=ctx)
    except mx.base.MXNetError:
        ctx = mx.cpu()
    return ctx

ctx = try_gpu()
ctx
```

Out[4]:
```
cpu(0)
```

Accuracy

In [5]:
```
def evaluate_accuracy(data_iter, net, ctx):
    acc_sum, n = nd.array([0], ctx=ctx), 0
    for X, y in data_iter:
        # If ctx is the GPU, copy the data to the GPU.
        X, y = X.as_in_context(ctx), y.as_in_context(ctx).astype('float32')
        acc_sum += (net(X).argmax(axis=1) == y).sum()
        n += y.size
    return acc_sum.asscalar() / n
```

Training loop

In [6]:
```
# This function has been saved in the d2l package for future use.
def train(net, train_iter, test_iter, batch_size, trainer, ctx,
              num_epochs):
    print('training on', ctx)
    loss = gloss.SoftmaxCrossEntropyLoss()
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n, start = 0.0, 0.0, 0, time.time()
        for X, y in train_iter:
            X, y = X.as_in_context(ctx), y.as_in_context(ctx)
            with autograd.record():
                y_hat = net(X)
                l = loss(y_hat, y).sum()
            l.backward()
            trainer.step(batch_size)
            y = y.astype('float32')
            train_l_sum += l.asscalar()
            train_acc_sum += (y_hat.argmax(axis=1) == y).sum().asscalar()
            n += y.size
        test_acc = evaluate_accuracy(test_iter, net, ctx)
        print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f, '
              'time %.1f sec'
              % (epoch + 1, train_l_sum / n, train_acc_sum / n, test_acc,
                 time.time() - start))
```

Network initialization and training

In [7]:
```
lr, num_epochs = 0.9, 5
net.initialize(force_reinit=True, ctx=ctx, init=init.Xavier())
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': lr})
train(net, train_iter, test_iter, batch_size, trainer, ctx, num_epochs)
```

```
training on cpu(0)
epoch 1, loss 2.3176, train acc 0.102, test acc 0.100, time 15.7 sec
epoch 2, loss 2.1522, train acc 0.170, test acc 0.523, time 15.5 sec
epoch 3, loss 1.0306, train acc 0.581, test acc 0.665, time 15.5 sec
epoch 4, loss 0.7861, train acc 0.689, test acc 0.741, time 15.8 sec
epoch 5, loss 0.6708, train acc 0.736, test acc 0.766, time 15.5 sec
```