Convolutional Layer Implementation Details

Deep Learning

Brad Quinton, Scott Chin

Learning Objectives

- A lite introduction to implementation and vectorization details for Convolutional Layers
- Understand enough about vectorization to do Assignment 4
- Understand how Convolution can be implemented as matrix multiplication

Non-vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)

```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    for x in range(f):
     for y in range(f):
      for c in range(cin):
        output[…] += input[…] * weight[…]
```

Non-vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)

```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    for x in range(f):
     for y in range(f):
      for c in range(cin):
        output[…] += input[…] * weight[…]
```

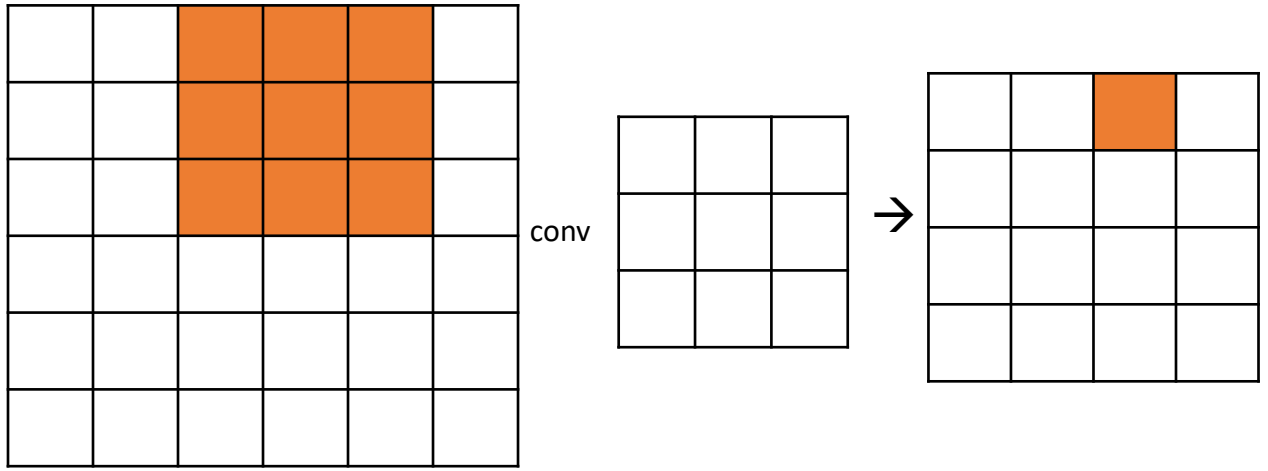
Foreach output spatial location

Non-vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    for x in range(f):
     for y in range(f):
      for c in range(cin):
        output[…] += input[…] * weight[…]
```

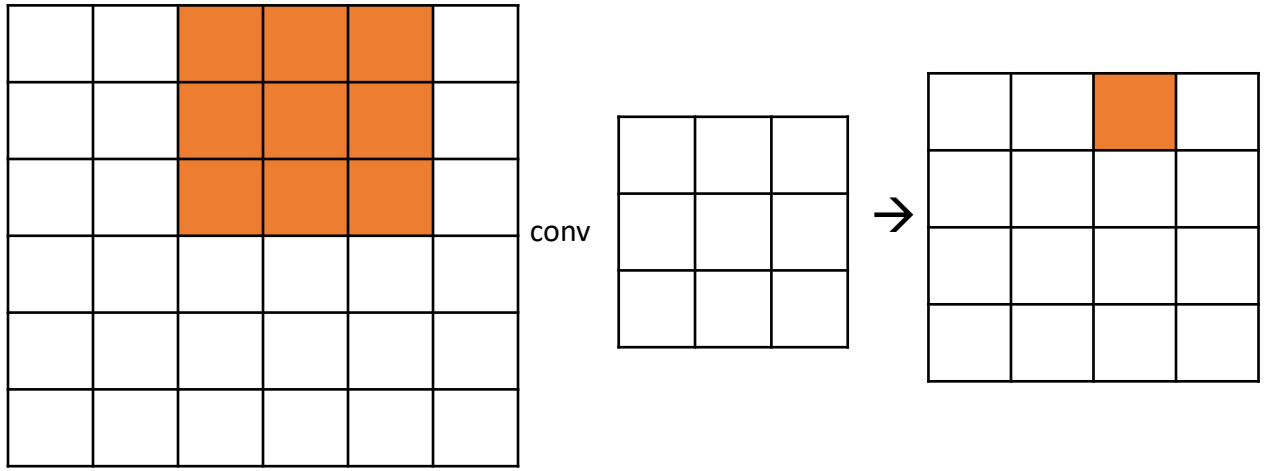
Convolution of filter with part of input

Non-vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    for x in range(f):
     for y in range(f):
      for c in range(cin):
        output[…] += input[…] * weight[…]
```

For Assignment 4, we will vectorize only this part

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)

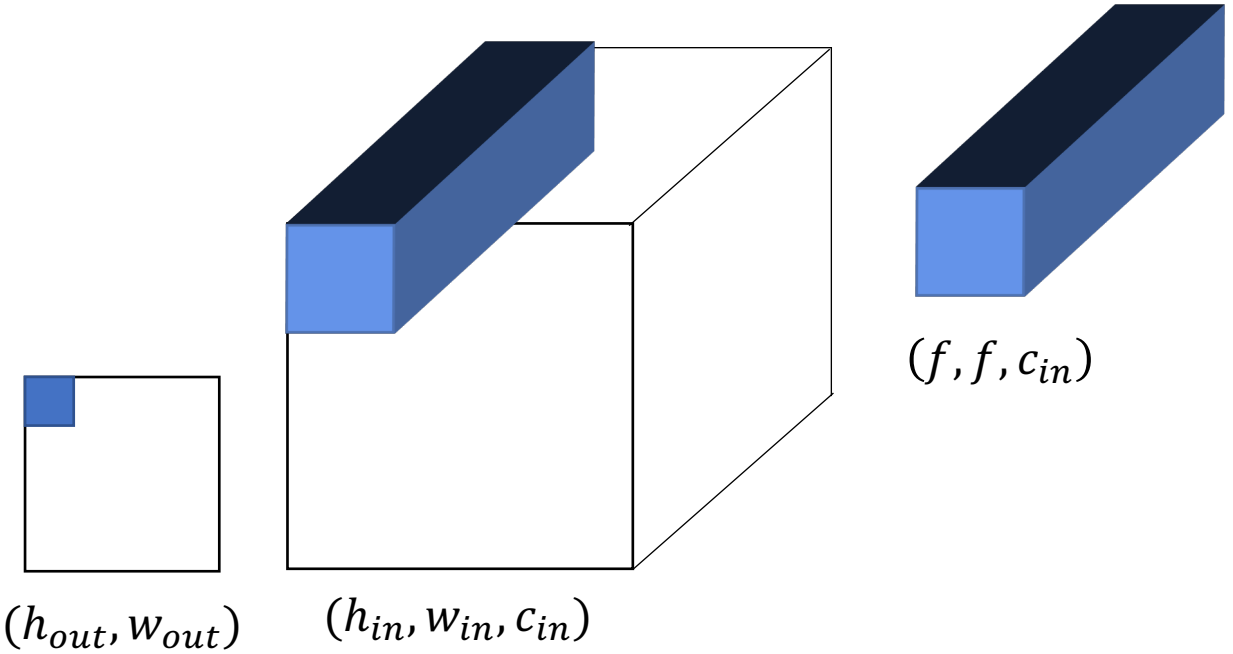
```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

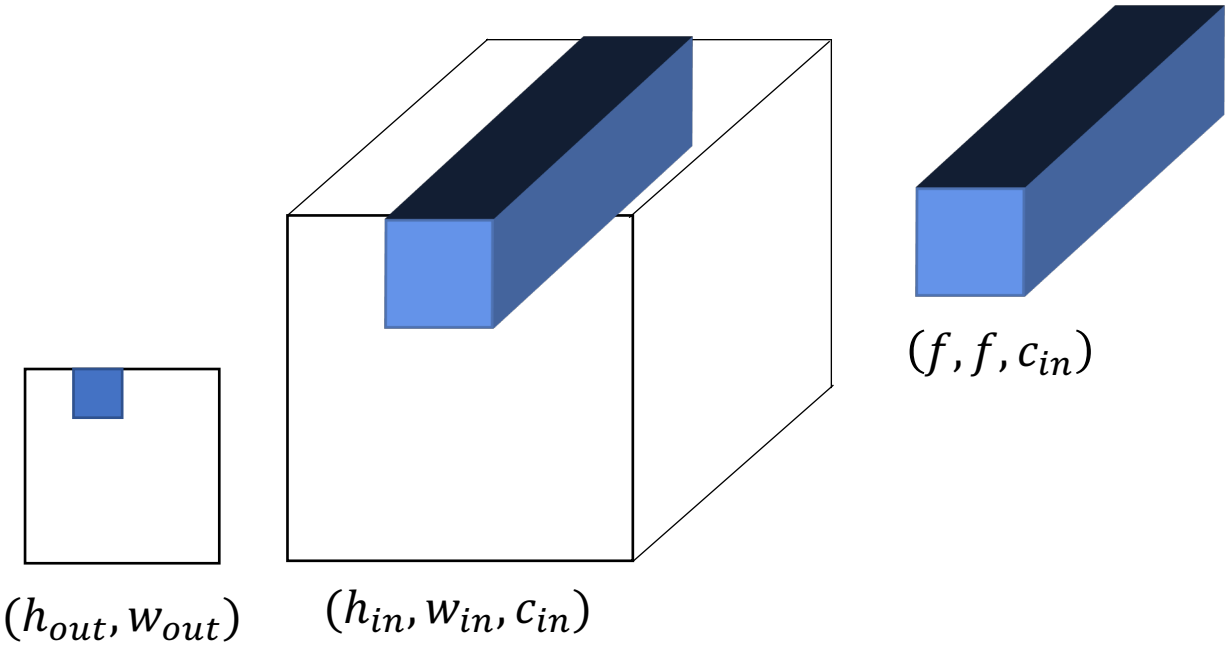
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

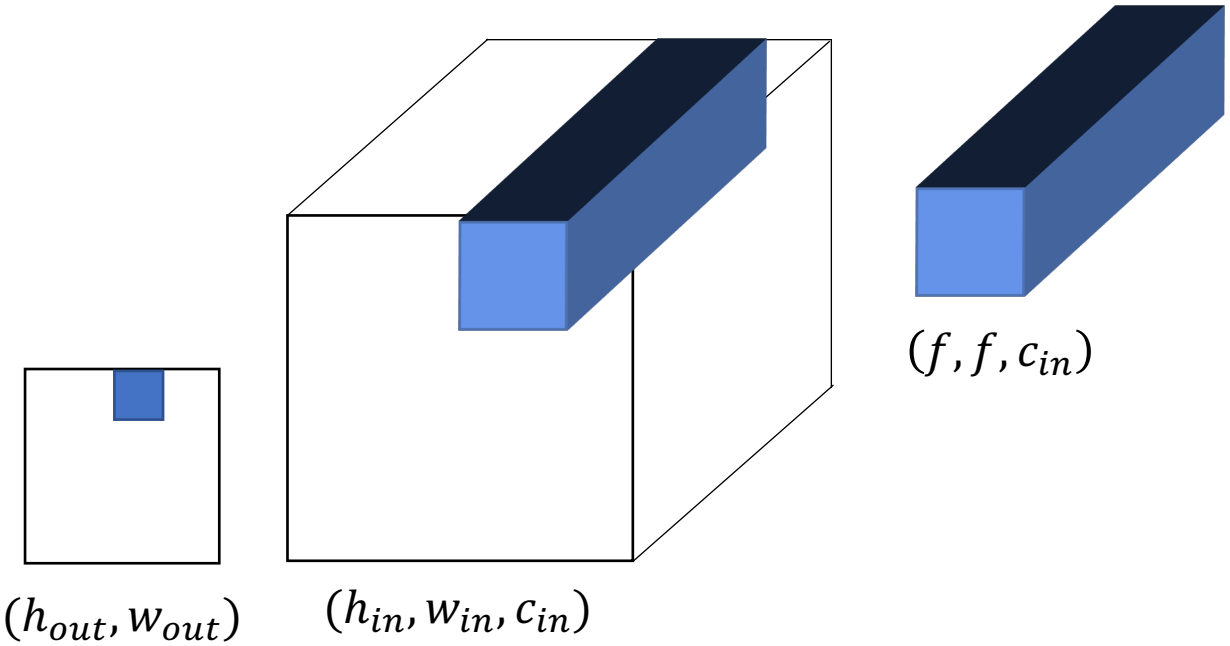
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

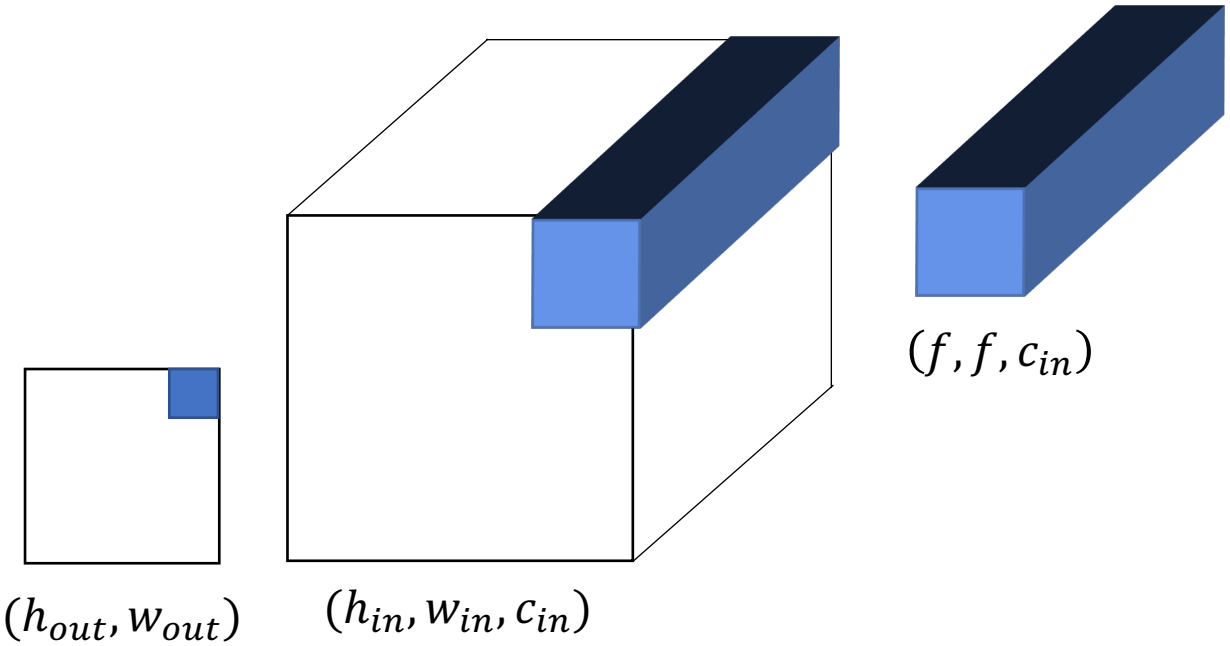
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

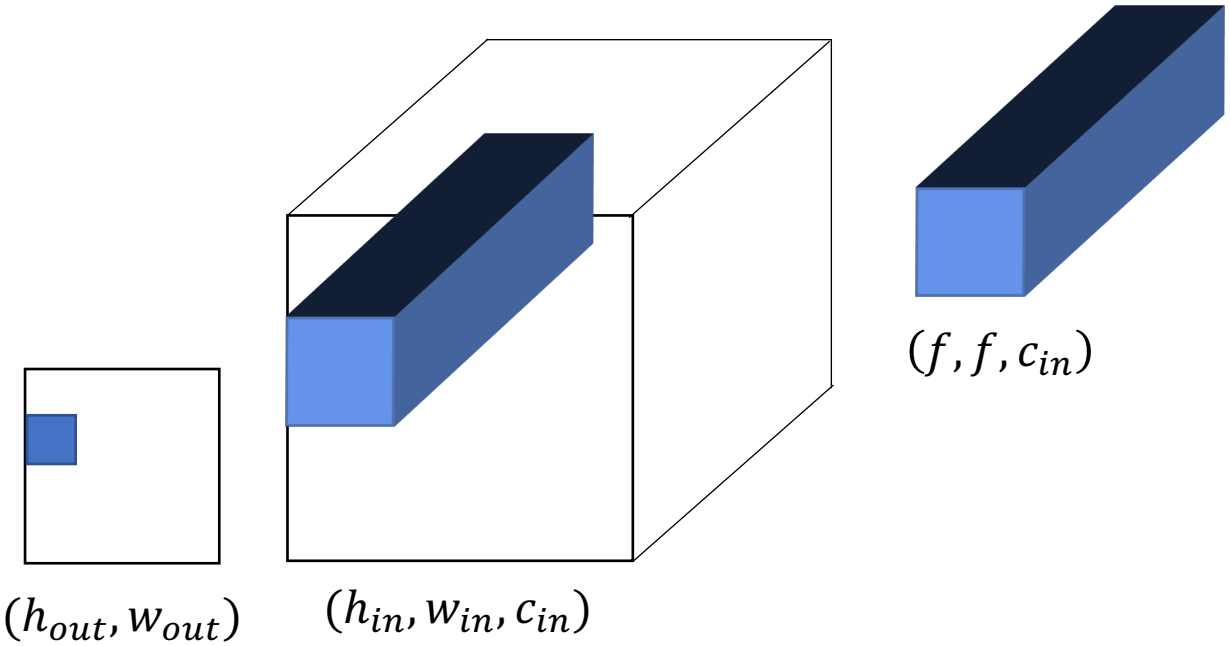
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

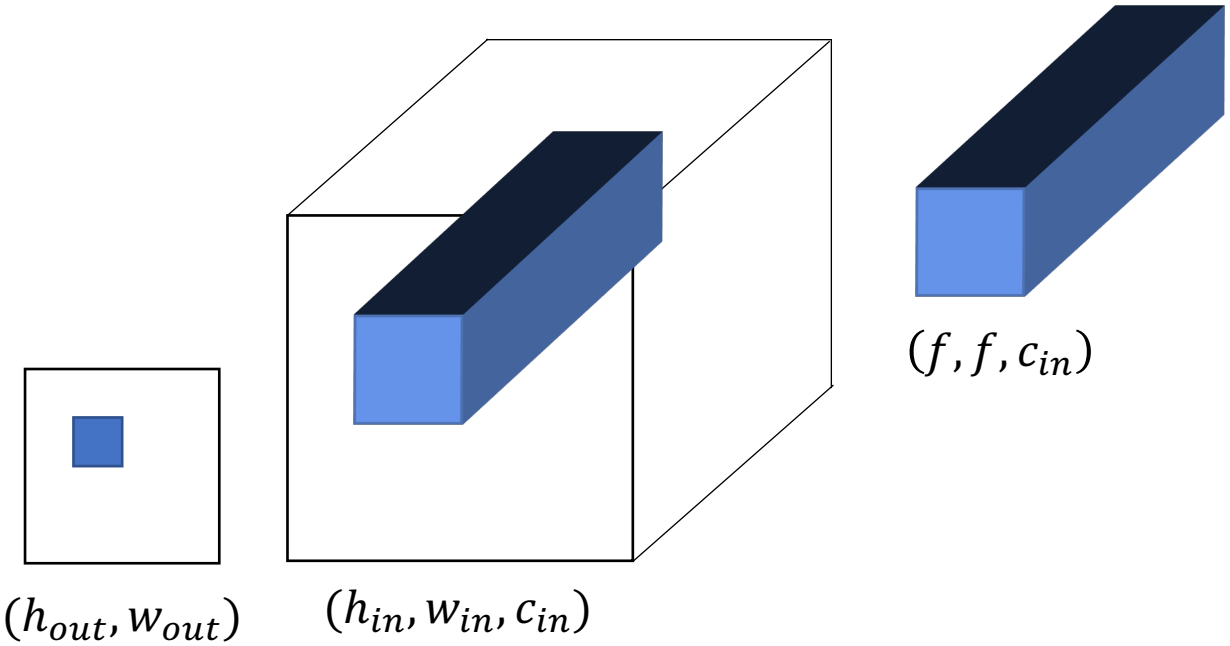
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

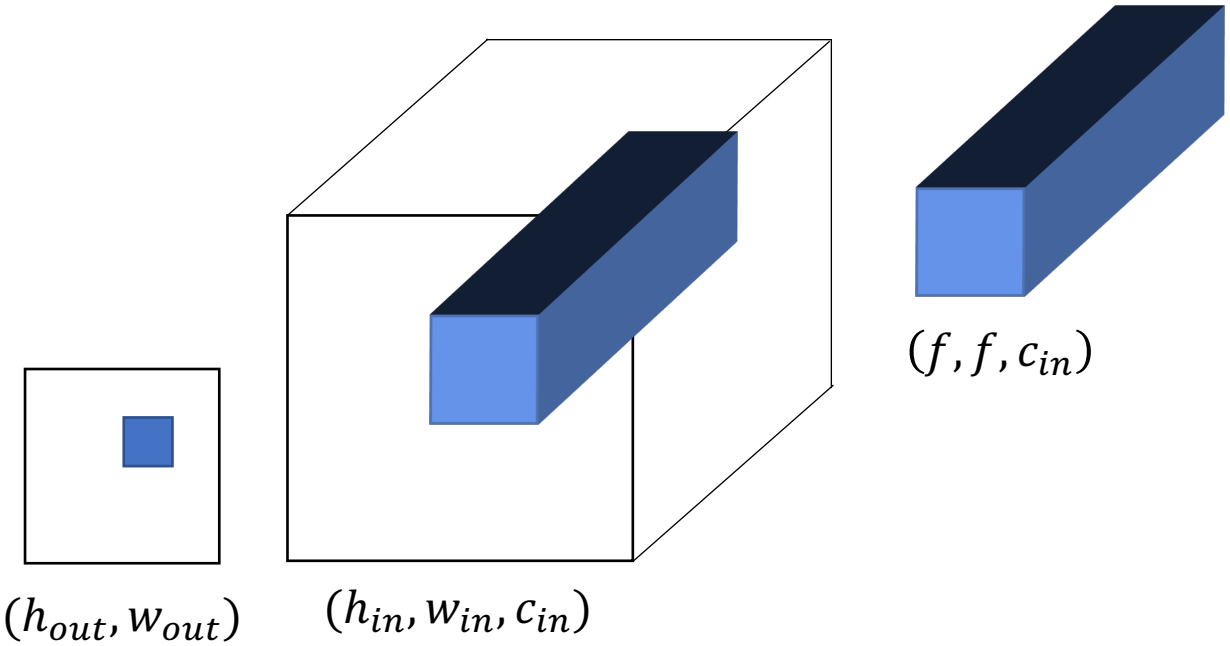
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

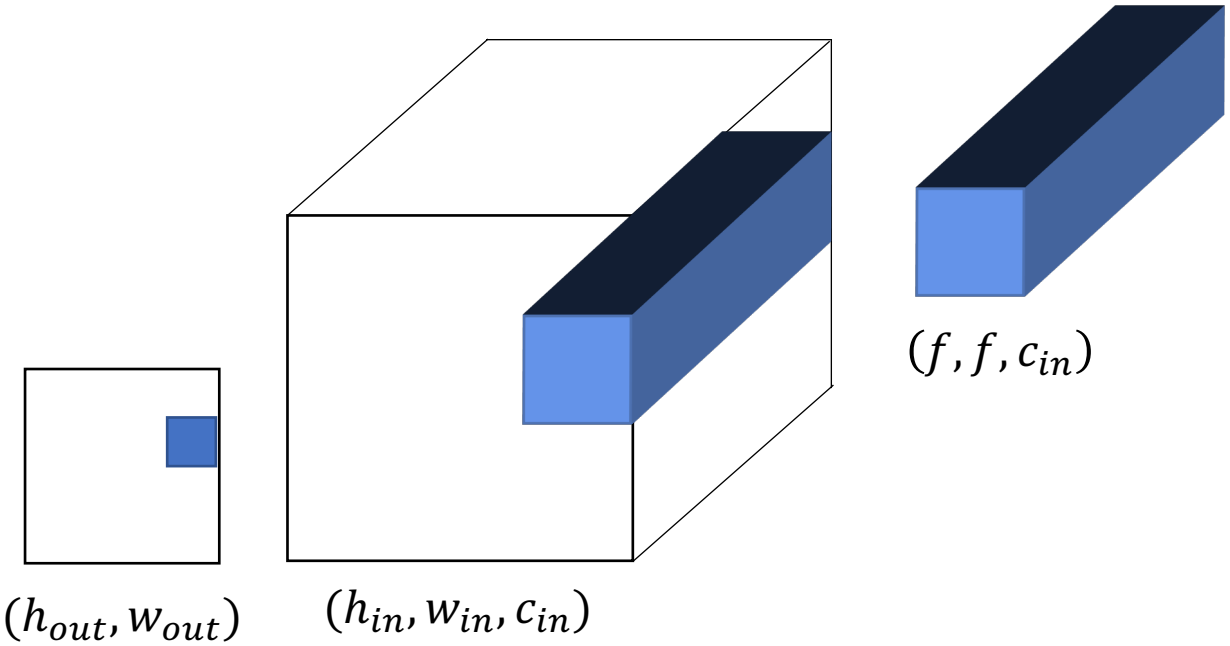
Showing One filter, one sample

Partially vectorized Implementation

Input is tensor $(h_{in}, w_{in}, c_{in}, m)$

Weight is tensor (K, f, f, c_{in})

Output is tensor (h_{out}, w_{out}, K, m)




```
for sample in range(m):
 for filter in range(K):
  for i in range(h_out):
   for j in range(w_out):
    output[…]=conv(input[…], weight[…])
```

Showing One filter, one sample

Vectorized Implementation

- In practice, convolutions are often implemented as matrix multiplication

https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

Vectorized Implementation

- In practice, convolutions are often implemented as matrix multiplication
- Decades of work on optimizing large matrix multiplication
- Basic Linear Algebra Subroutines (BLAS)
  - Specification for low-level linear algebra operations (started in 1979)
  - Implementations are highly optimized for performance (e.g. cuBLAS)
  - Many numerical software packages (NumPy, Matlab, etc) use BLAS libraries
- General Matrix Multiplication (GeMM)

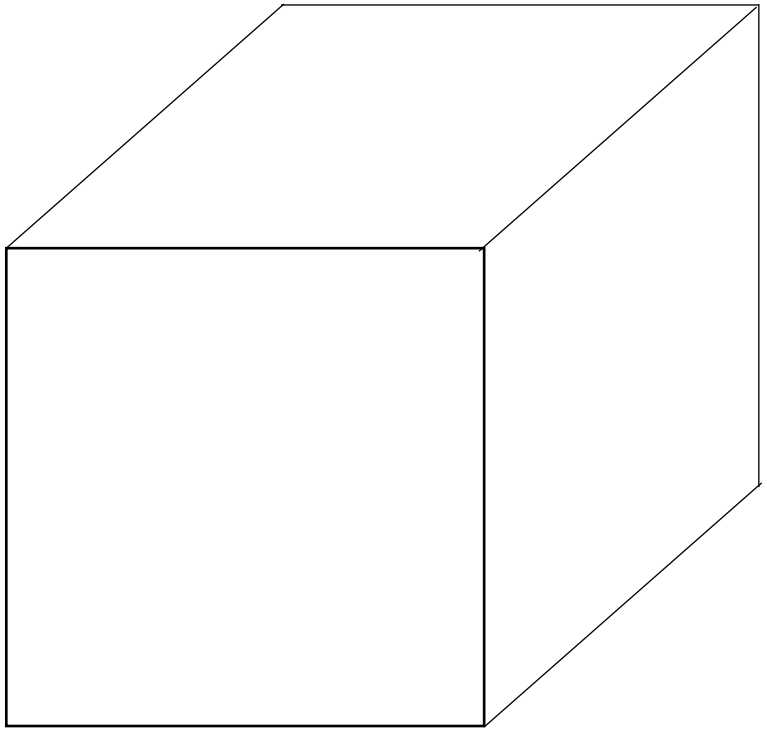
https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

Vectorized Implementation

- In practice, convolutions are often implemented as matrix multiplication
- Decades of work on optimizing large matrix multiplication
- Basic Linear Algebra Subroutines (BLAS)
  - Specification for low-level linear algebra operations (started in 1979)
  - Implementations are highly optimized for performance (e.g. cuBLAS)
  - Many numerical software packages (NumPy, Matlab, etc) use BLAS libraries
- General Matrix Multiplication (GeMM)
- Already using matrix multiplication for fully-connected layers
- How do we transform convolution into matrix multiplication?

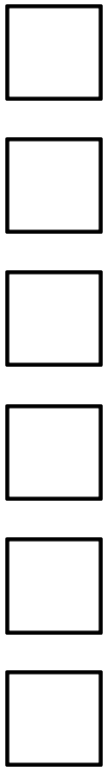
https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

Consider the following Conv Layer



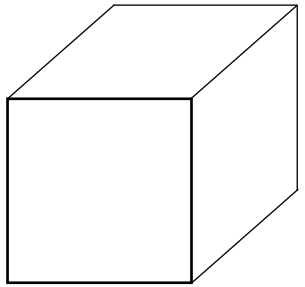
Input Volume
$(6,6,10)$

conv



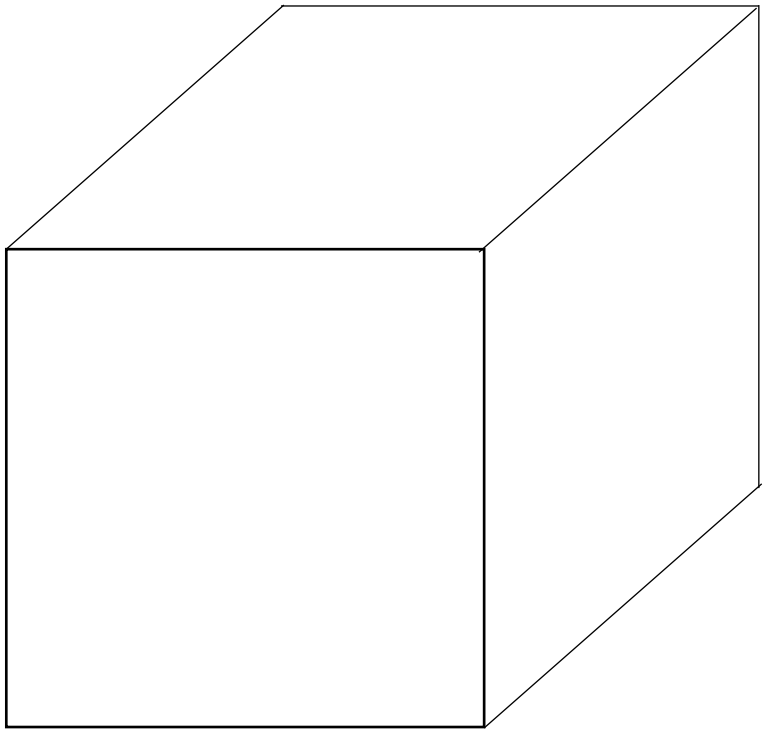
Layer has 6 filters
of shape $(3,3,10)$

=



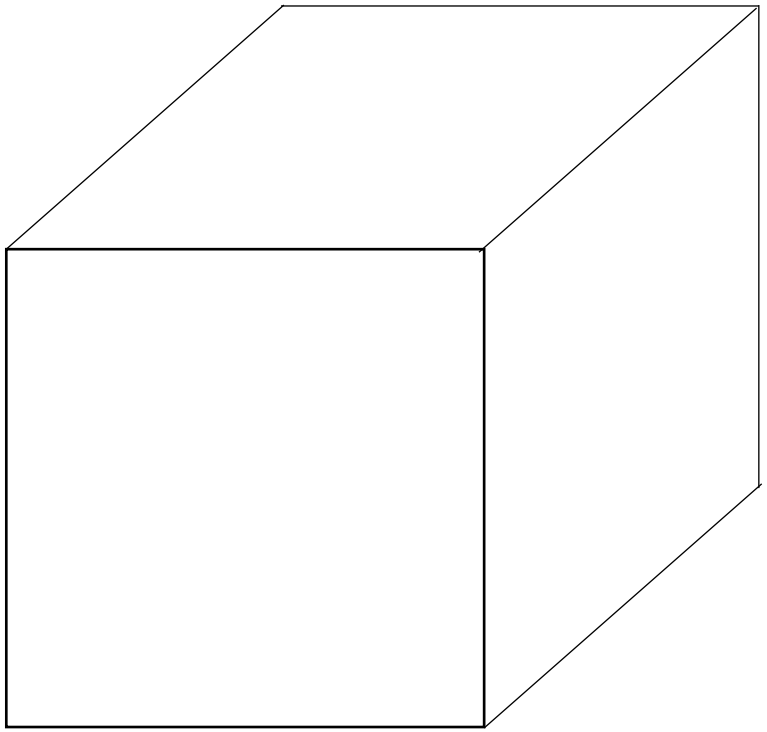
Output Volume
$(4,4,6)$

Transform Input Volume into 2D Matrix



Input Volume
(6,6,10)

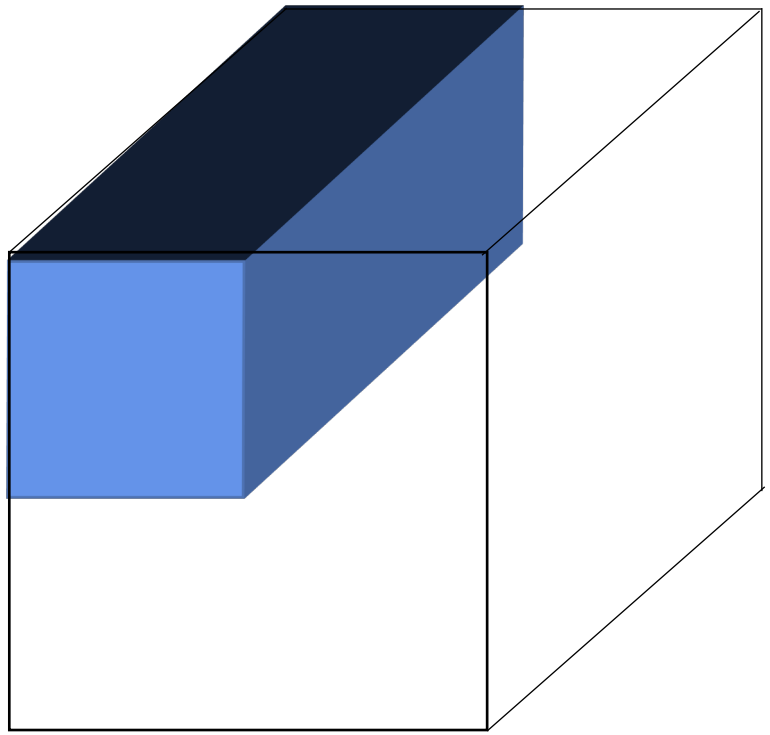
Transform Input Volume into 2D Matrix



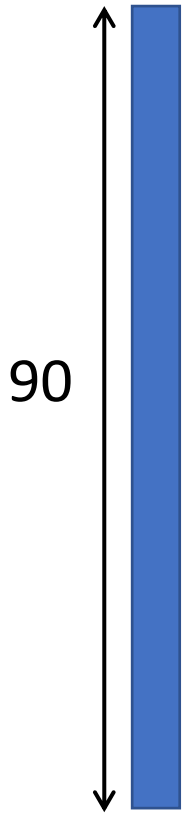
Input Volume
$(6,6,10)$

Filter Shape: $(3,3,10)$

Transform Input Volume into 2D Matrix

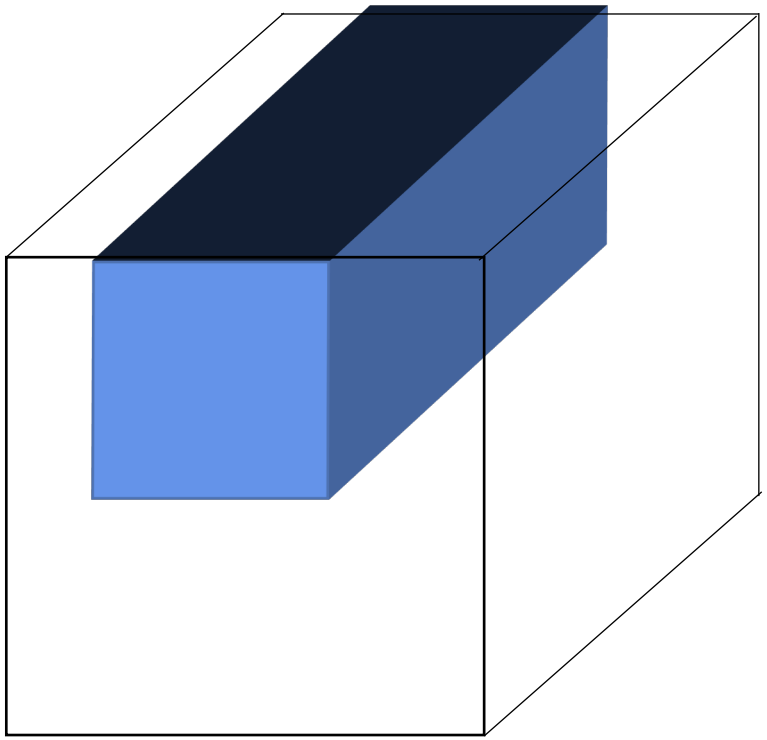


Input Volume
(6,6,10)

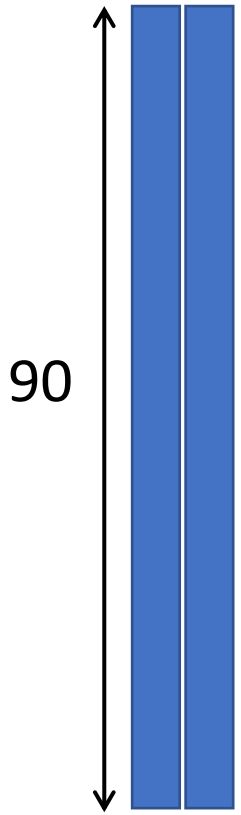


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

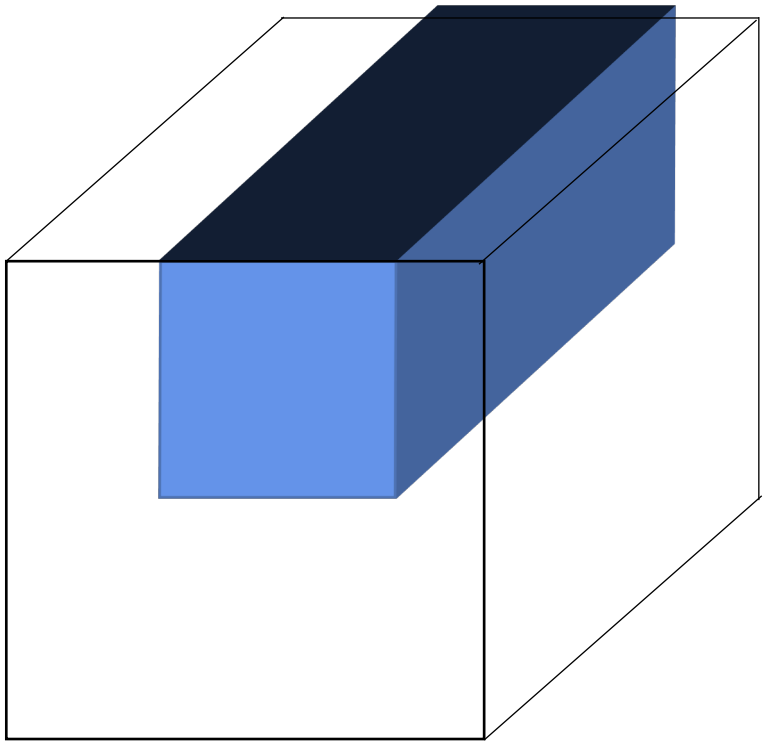


Input Volume
(6,6,10)

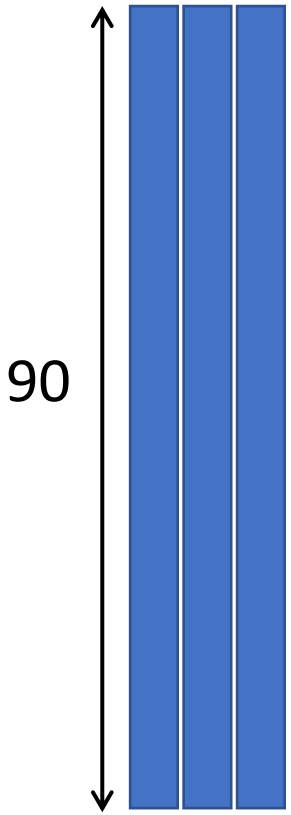


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

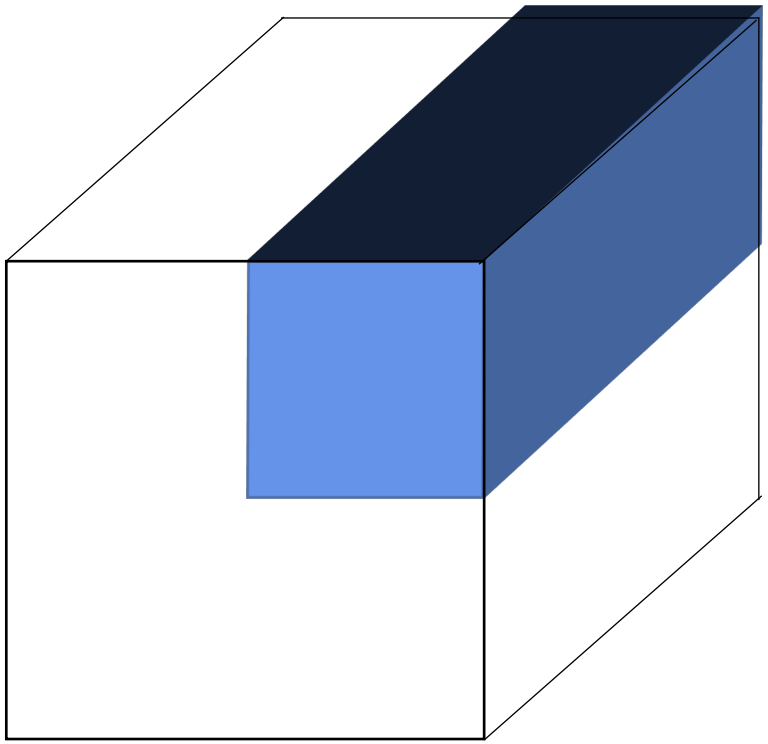


Input Volume
(6,6,10)

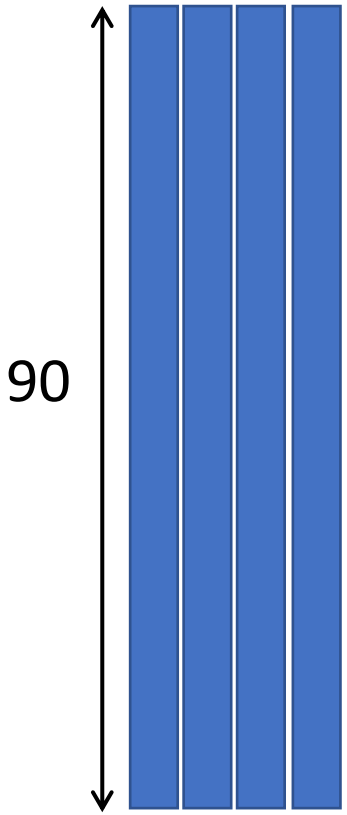


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

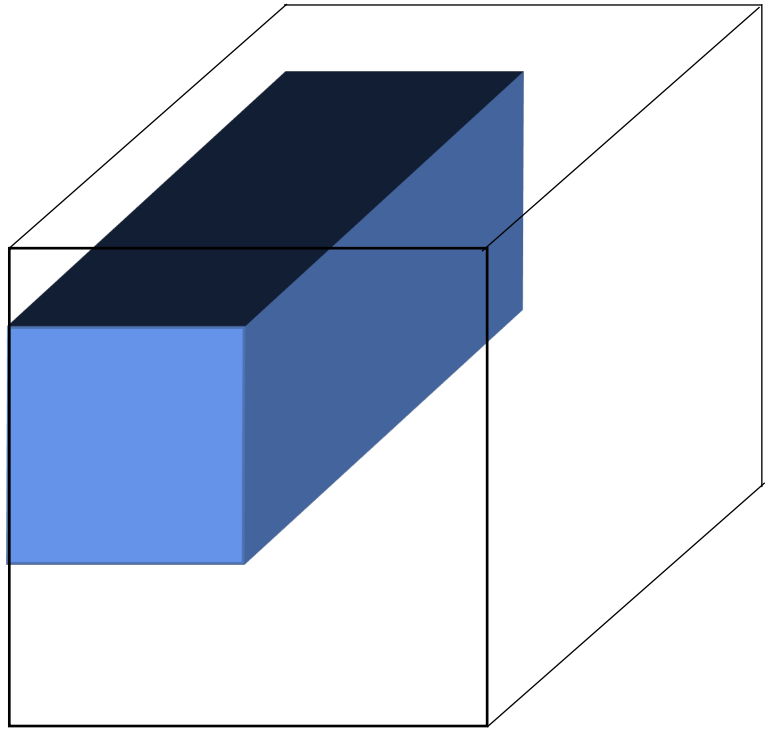


Input Volume
(6,6,10)

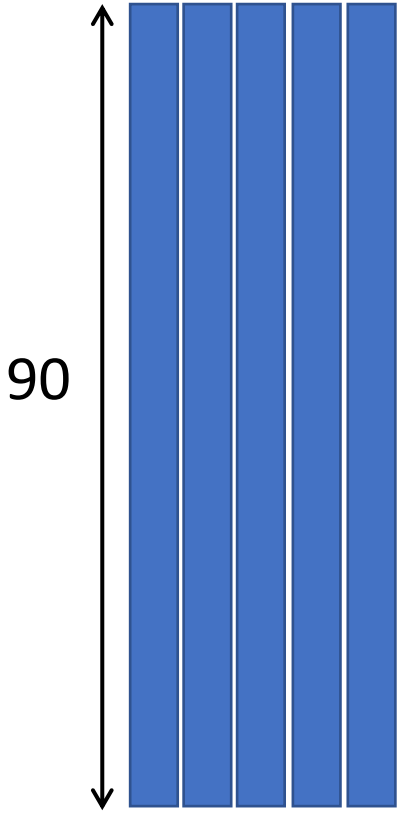


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

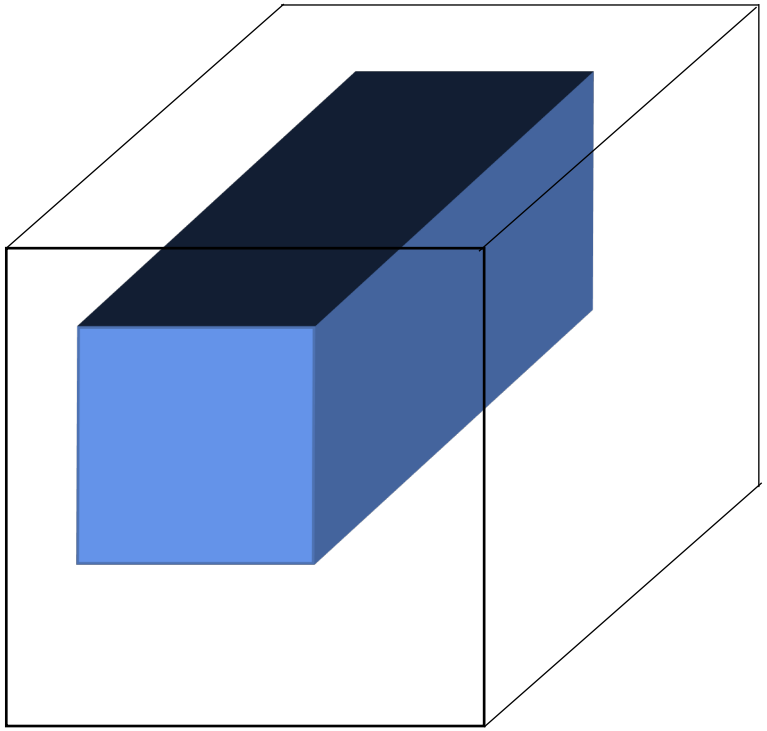


Input Volume
(6,6,10)

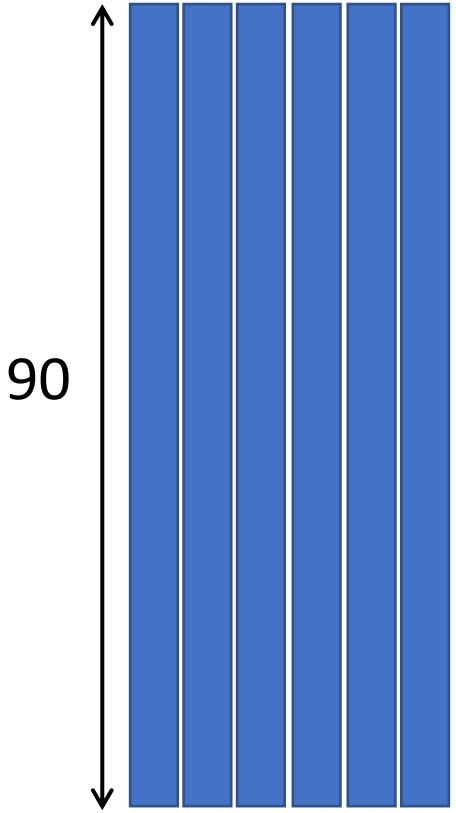


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

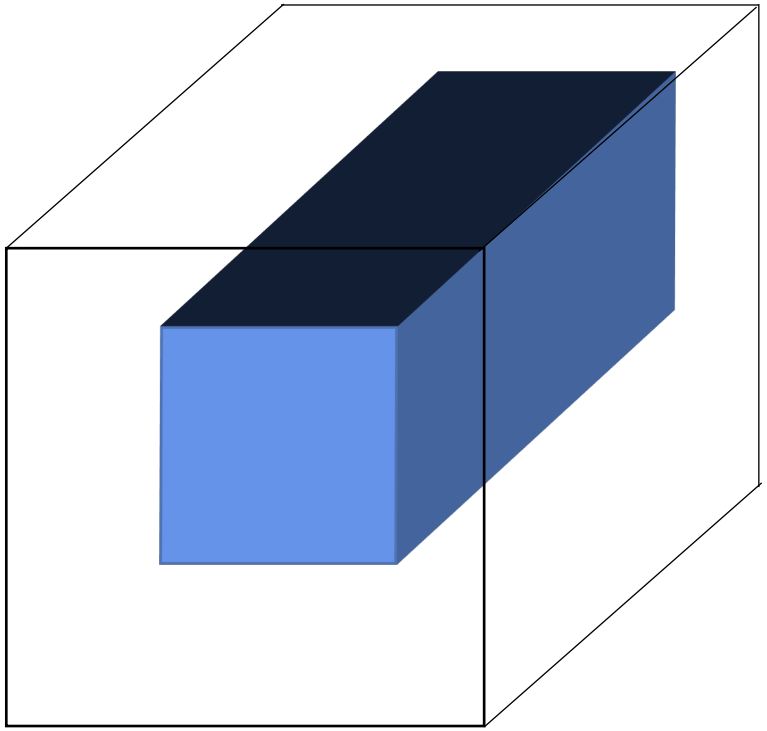


Input Volume
(6,6,10)

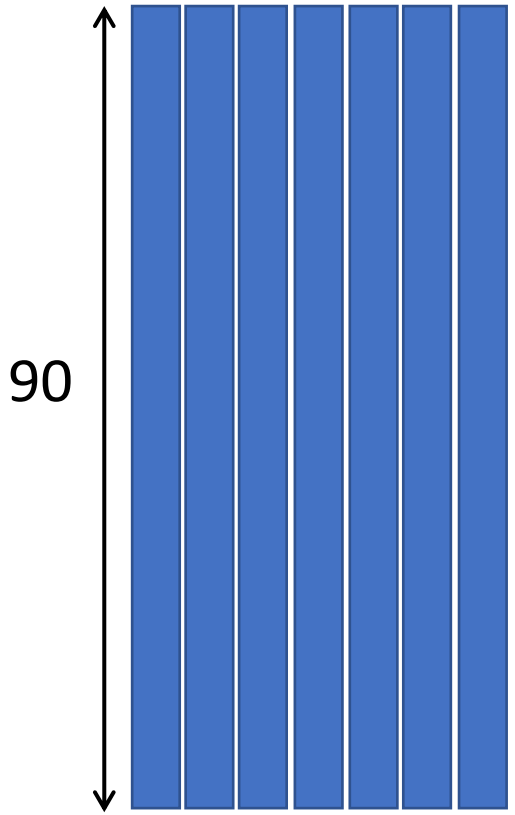


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

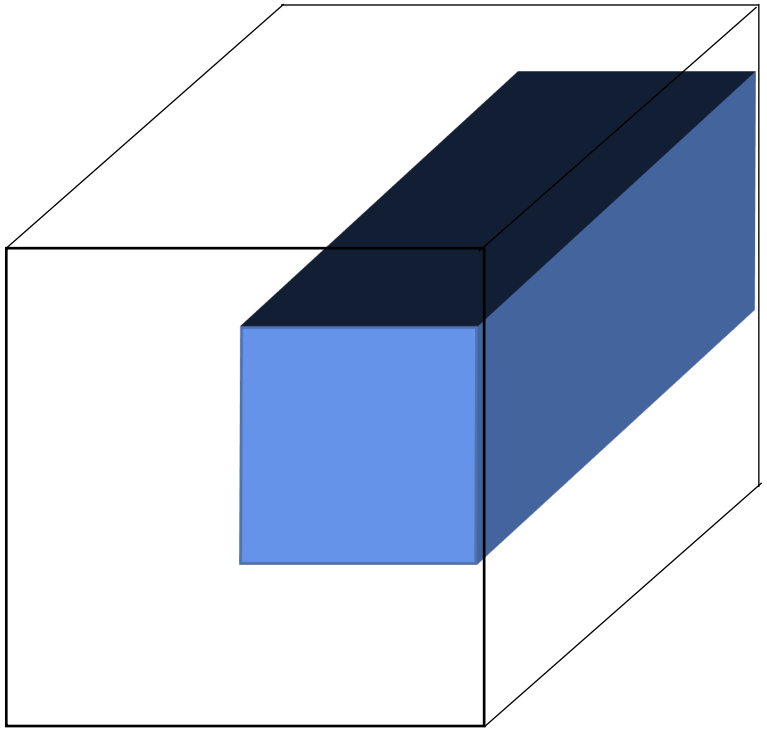


Input Volume
(6,6,10)

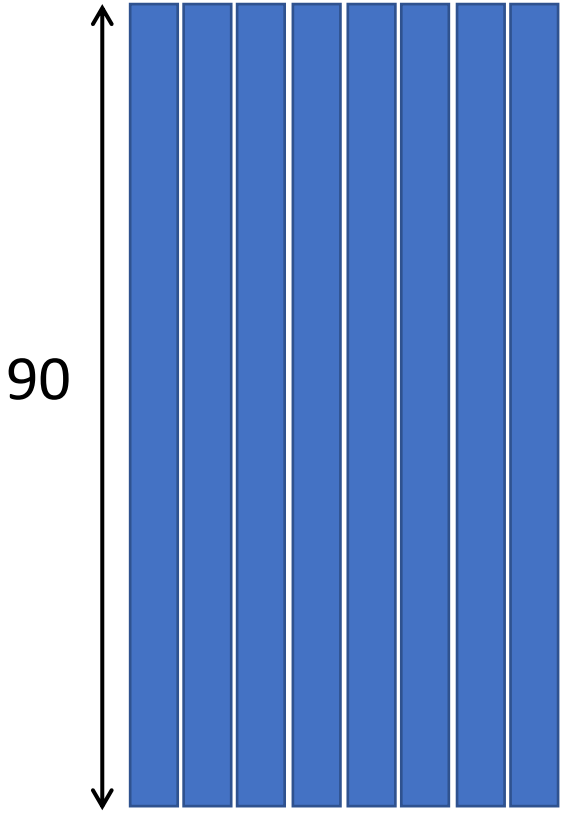


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

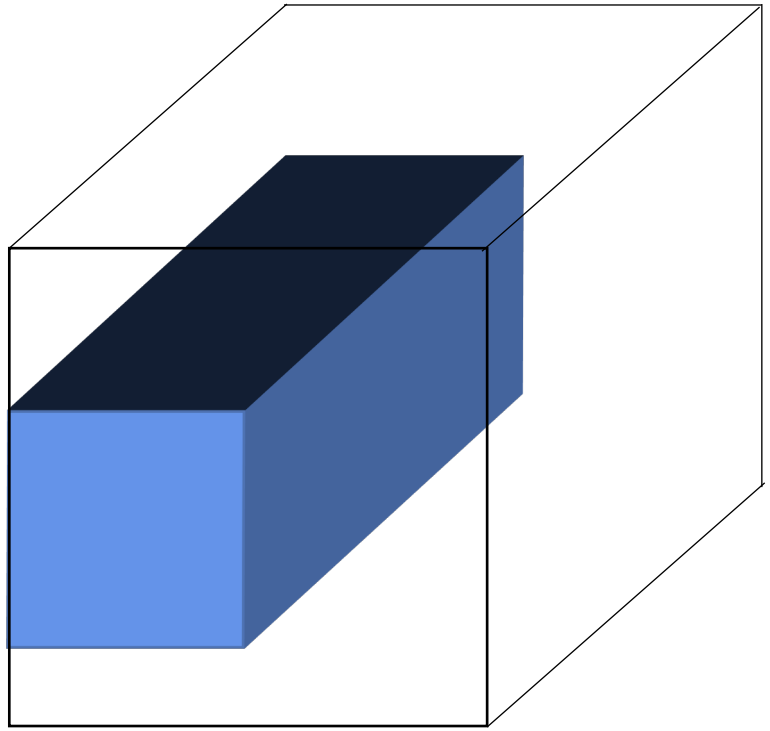


Input Volume
(6,6,10)

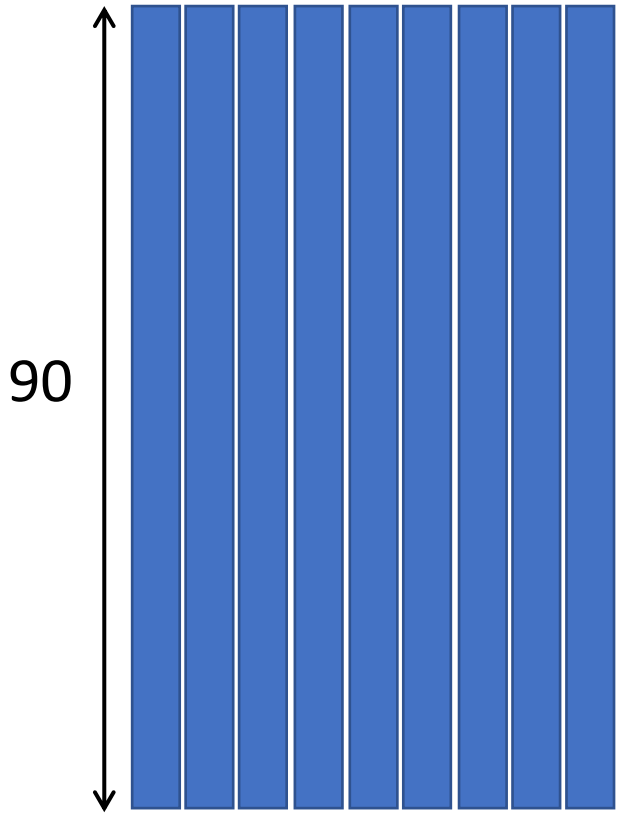


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

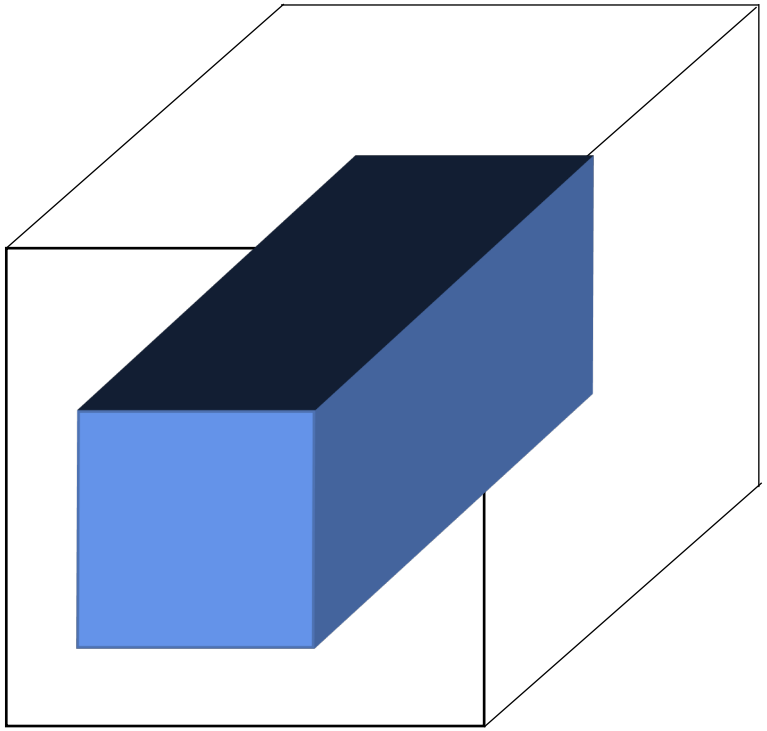


Input Volume
(6,6,10)

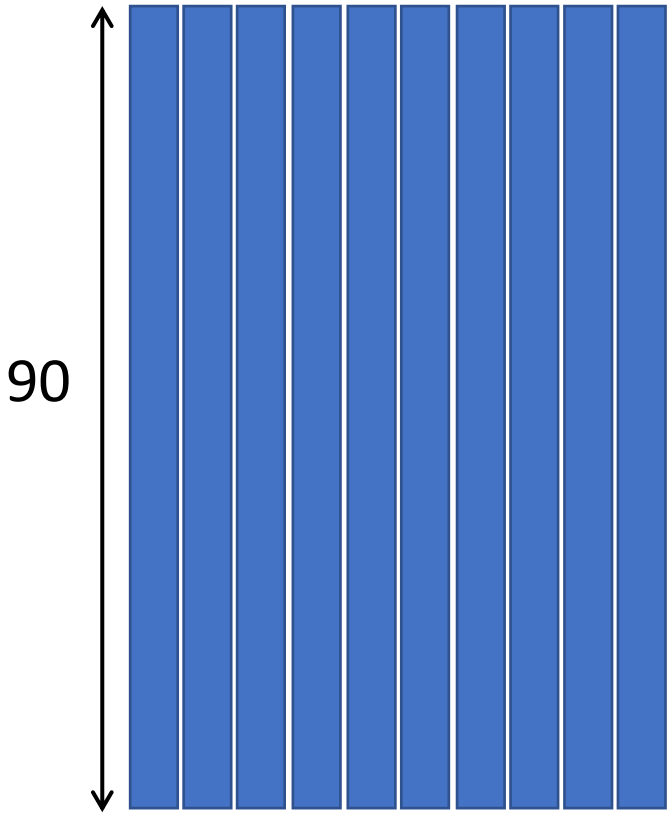


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

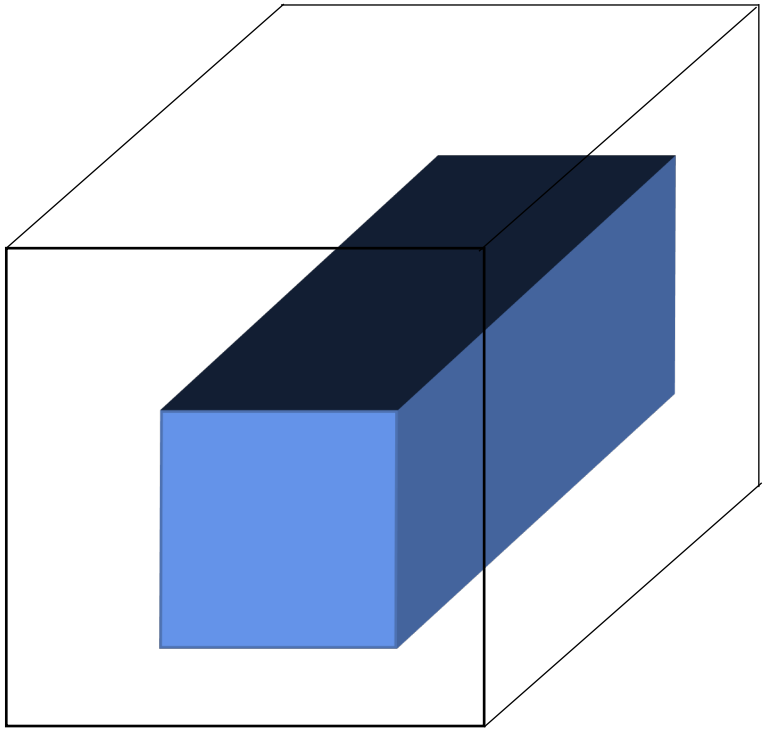


Input Volume
(6,6,10)

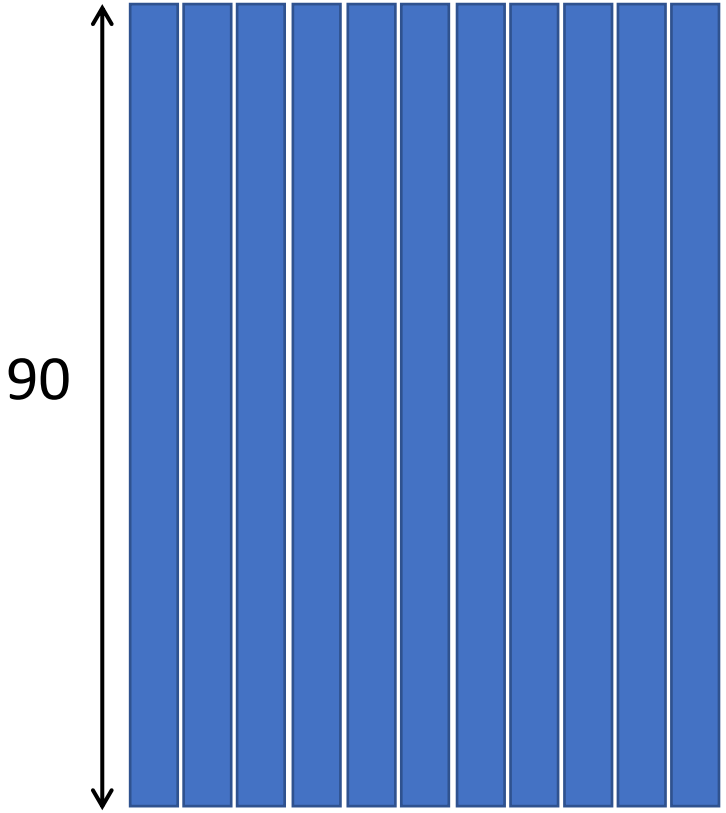


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

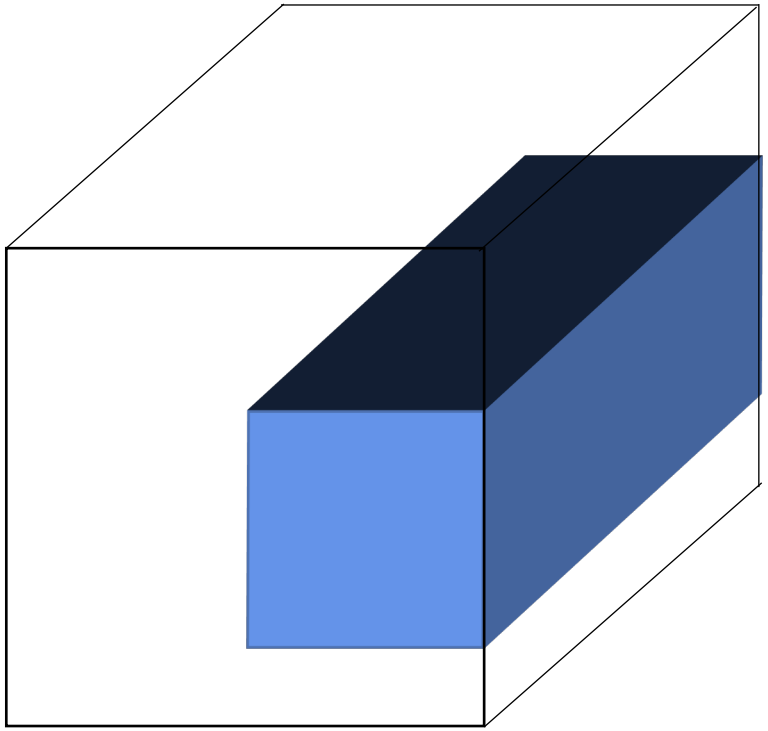


Input Volume
(6,6,10)

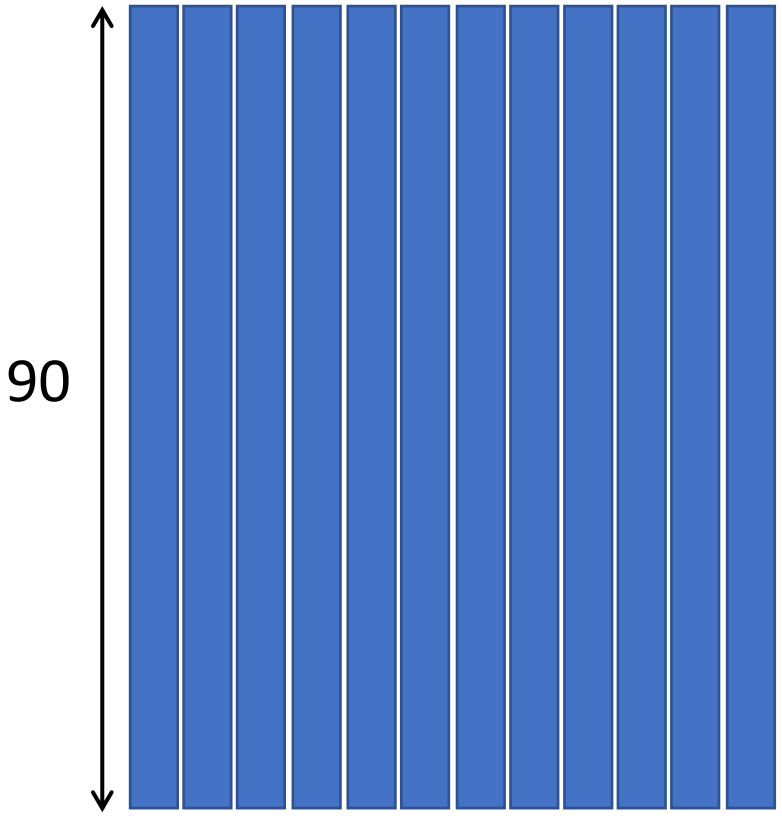


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

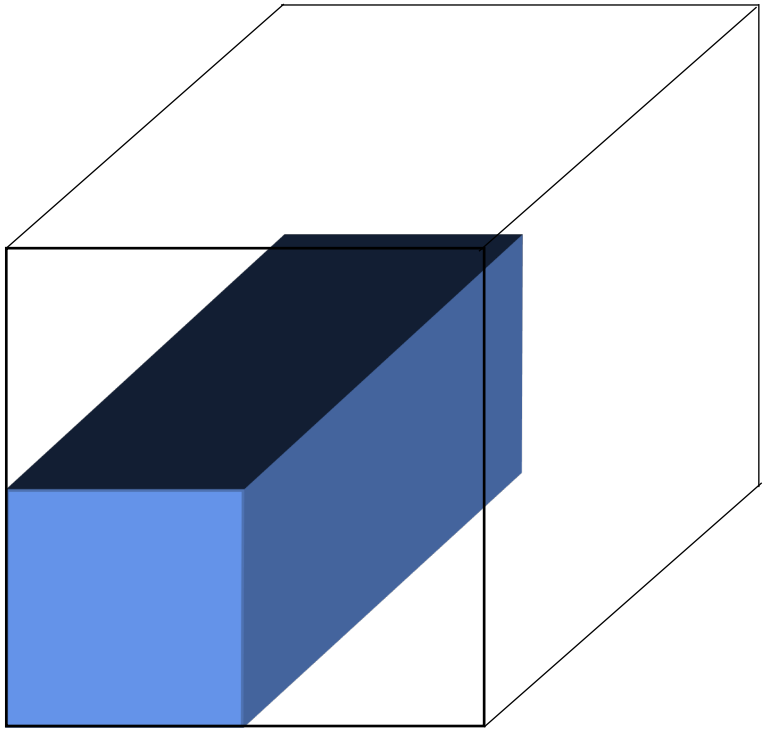


Input Volume
(6,6,10)

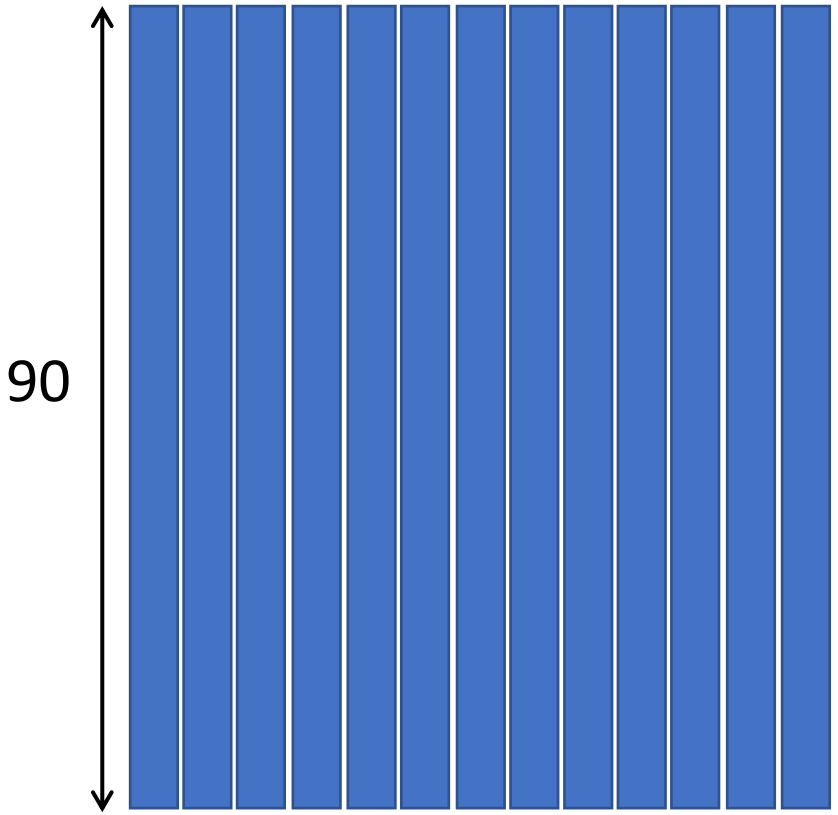


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

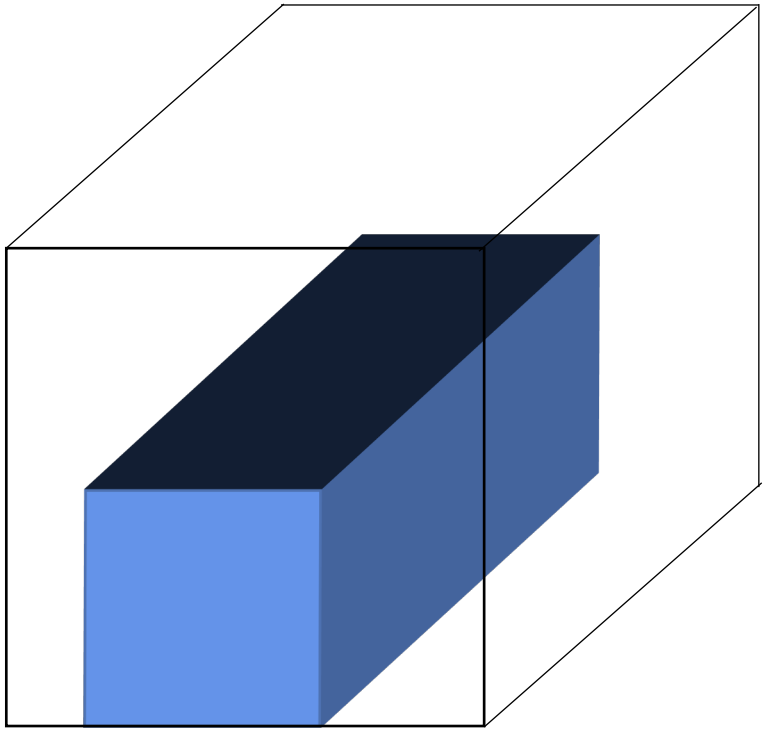


Input Volume
(6,6,10)

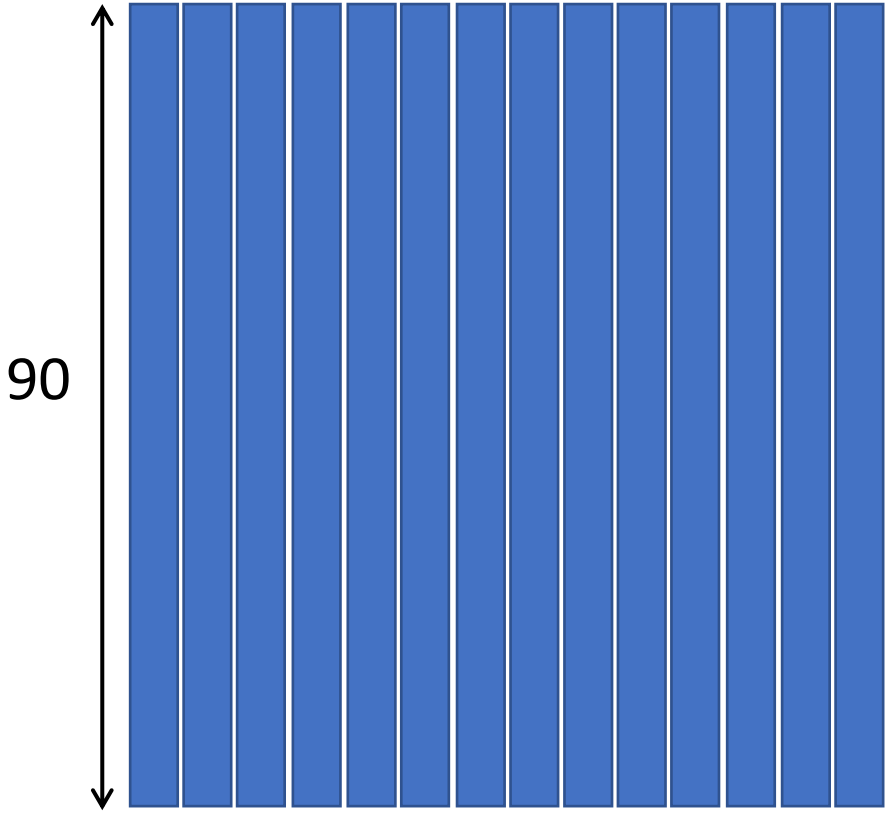


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

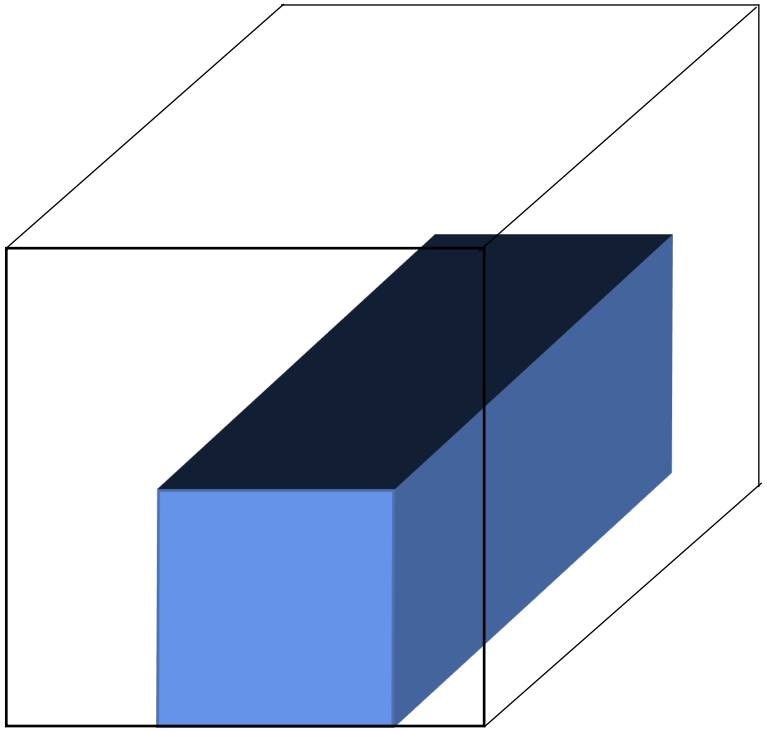


Input Volume
(6,6,10)

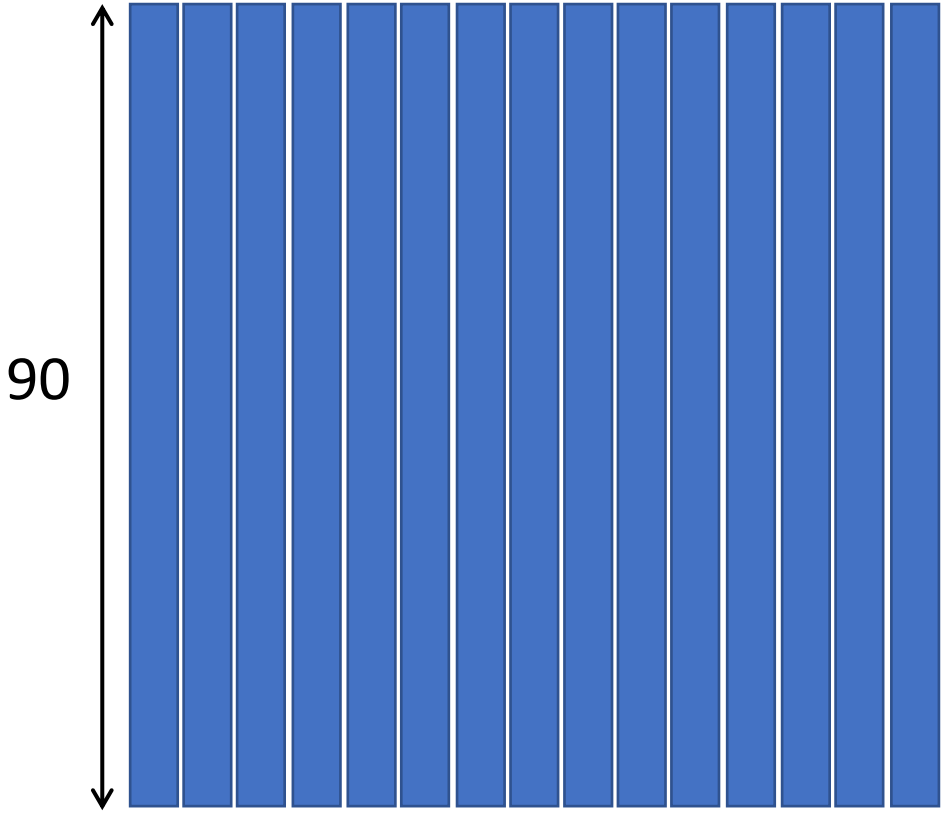


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

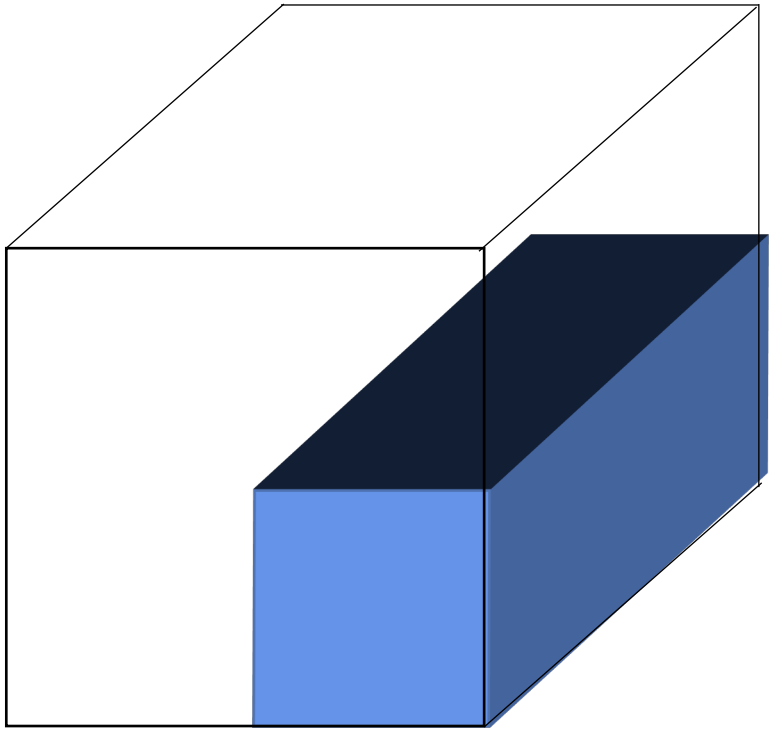


Input Volume
(6,6,10)

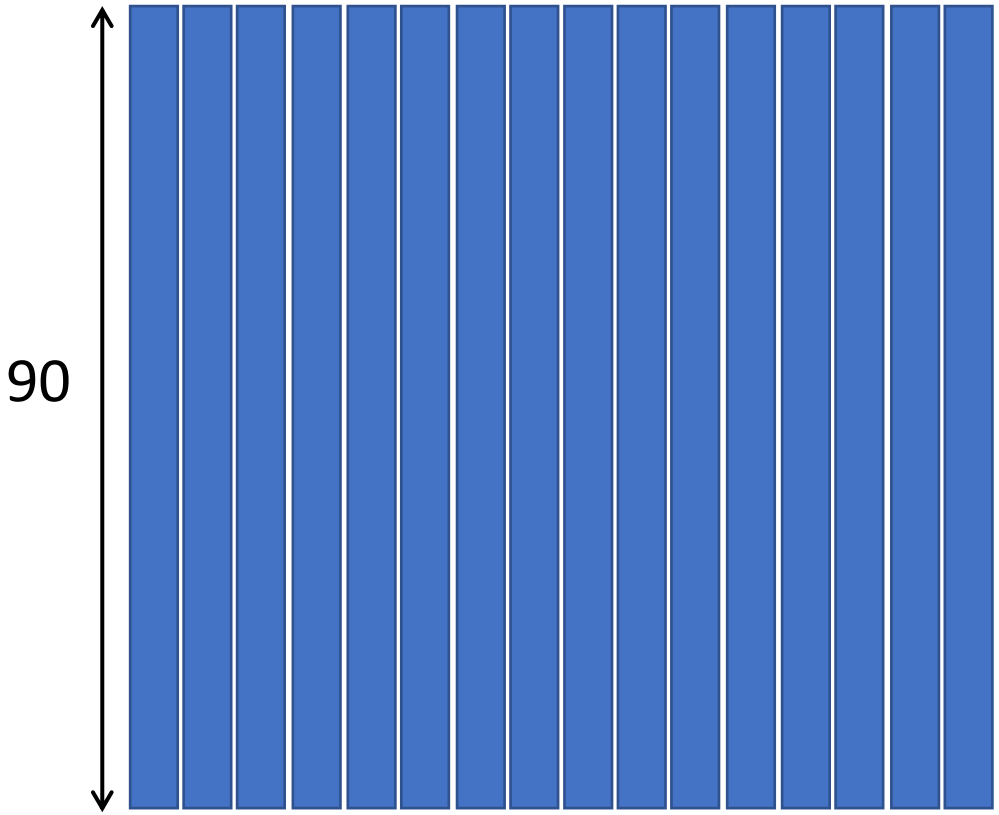


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

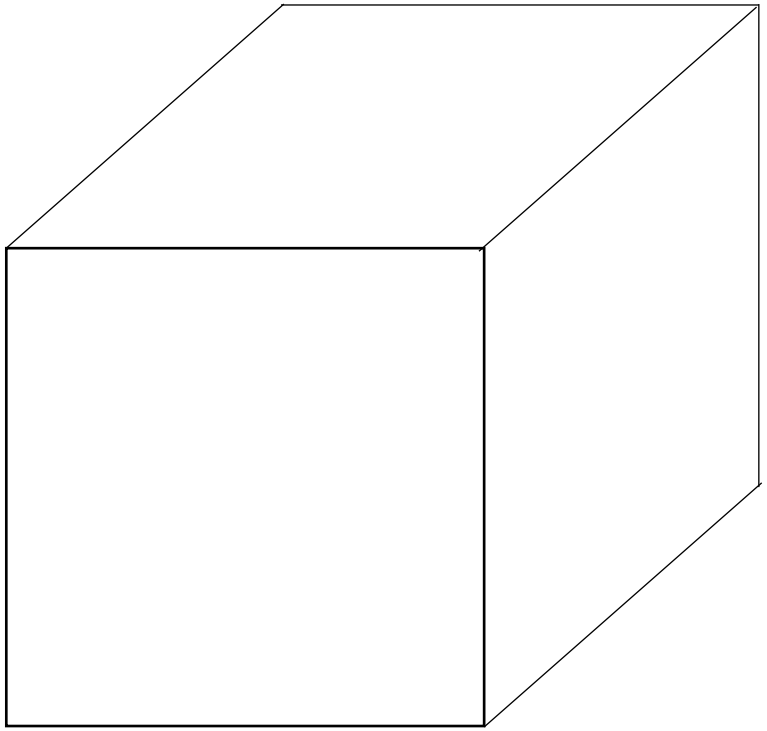


Input Volume
(6,6,10)

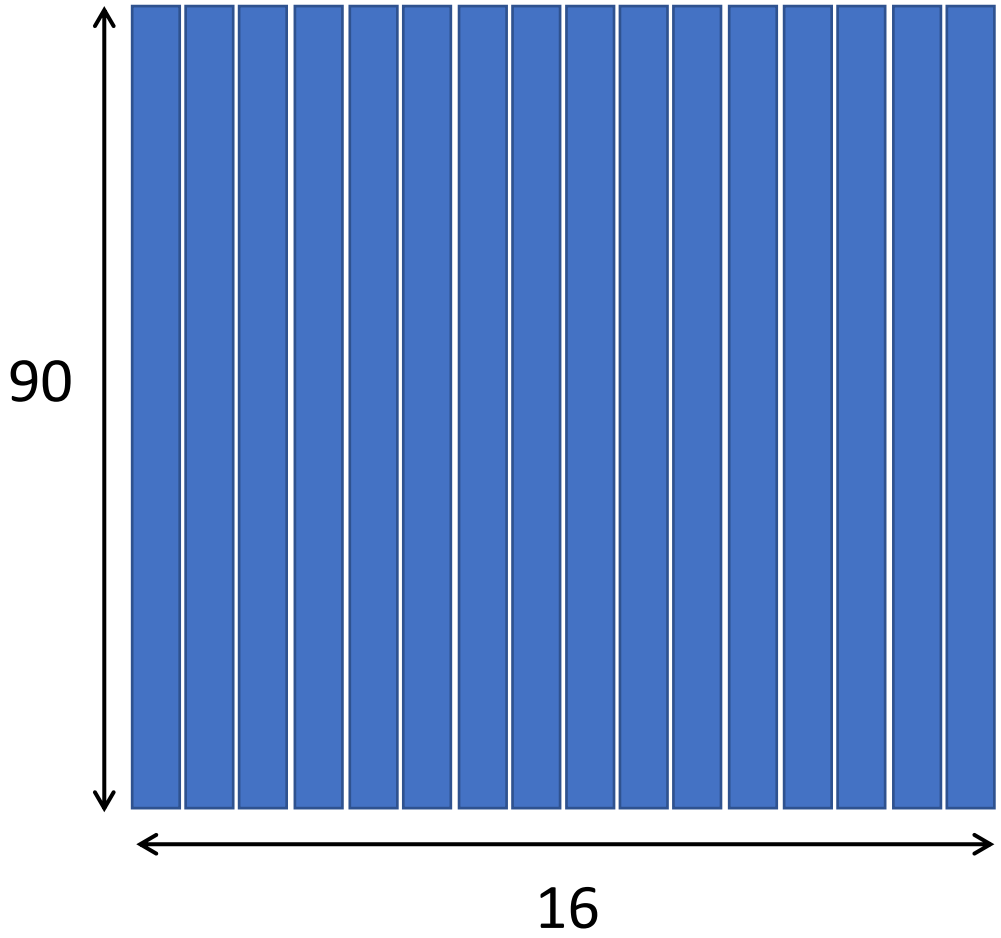


Filter Shape: (3,3,10)

Transform Input Volume into 2D Matrix

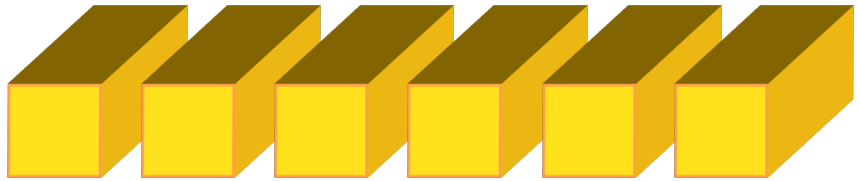


Input Volume
$(6,6,10)$



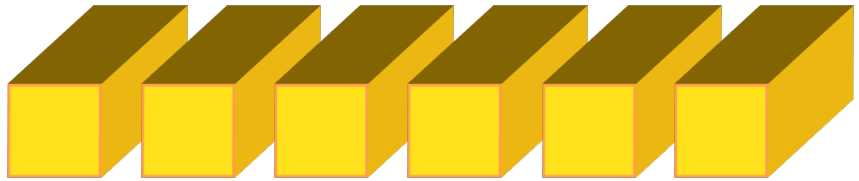
Filter Shape: $(3,3,10)$

Transform Filters in Conv Layer into 2D Matrix

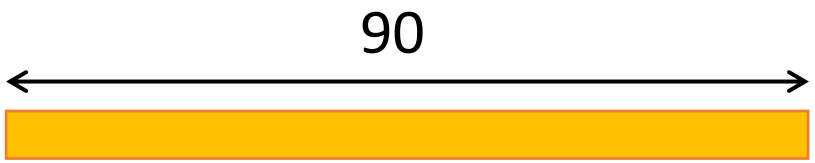


Layer has 6 filters of shape $(3,3,10)$

Transform Filters in Conv Layer into 2D Matrix

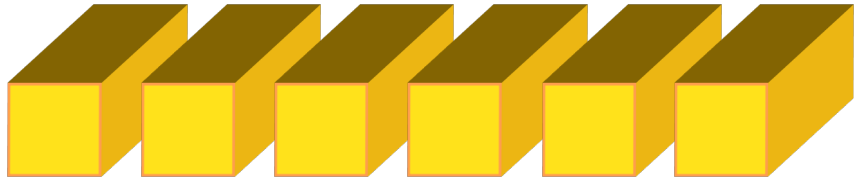


Layer has 6 filters of shape $(3,3,10)$

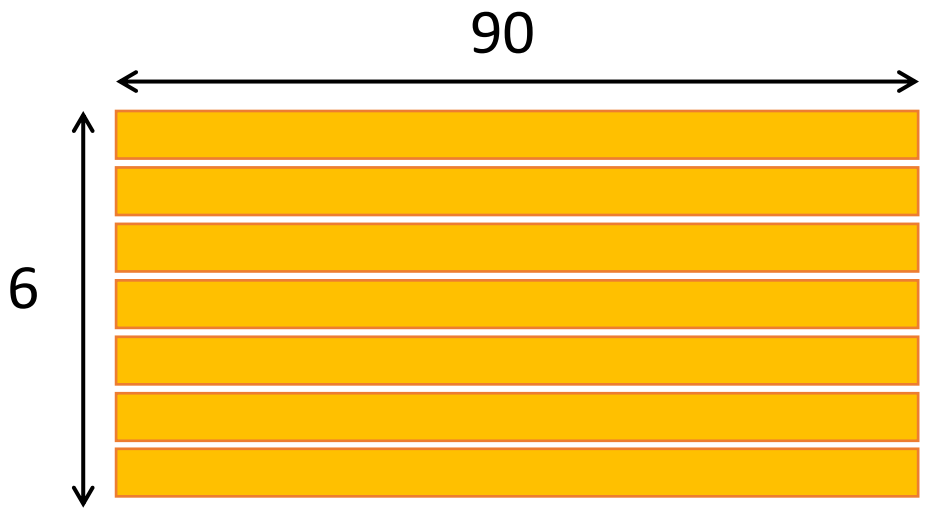


- Reshape each filter into a vector

Transform Filters in Conv Layer into 2D Matrix



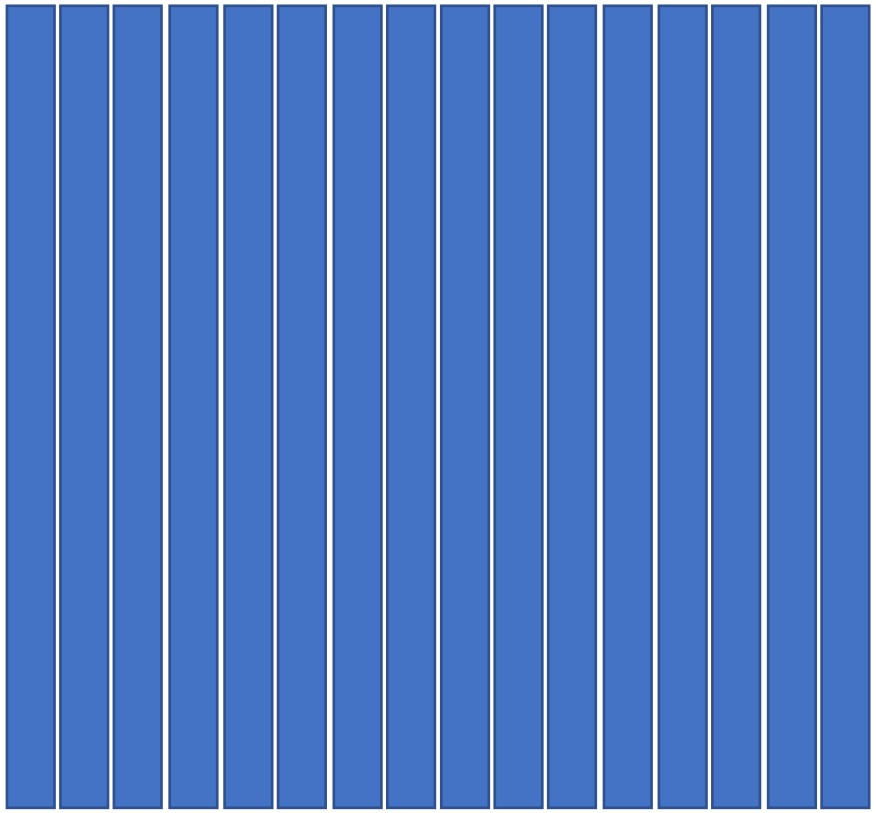
Layer has 6 filters of shape $(3,3,10)$



- Reshape each filter into a vector
- Each filter is a row in new matrix

Matrix Multiply

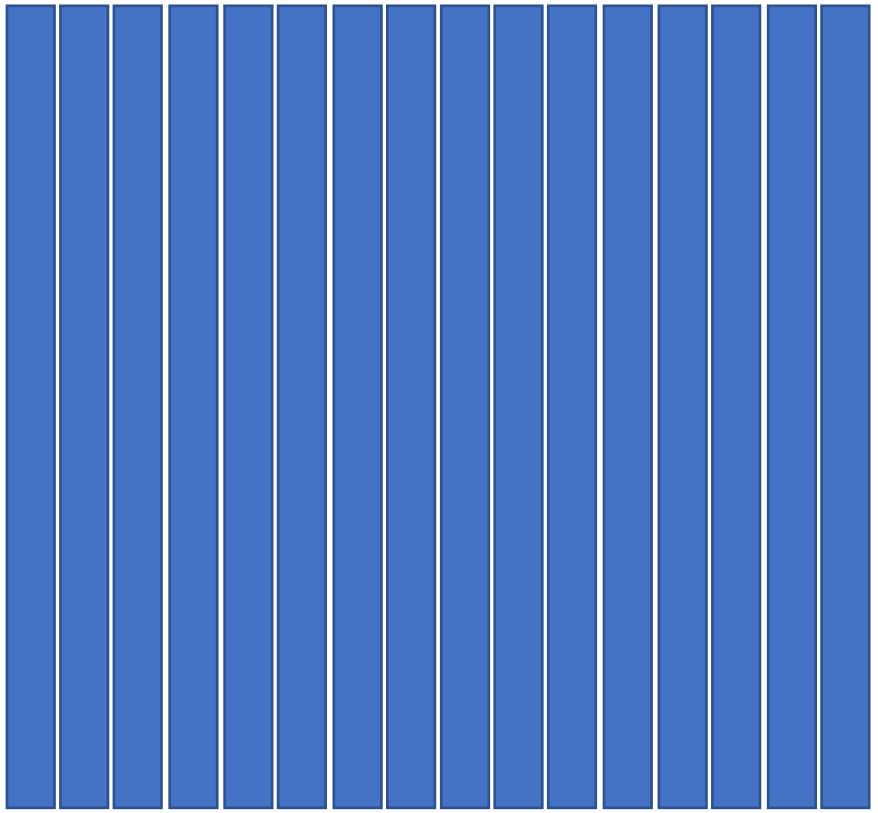
(6,90)



(90,16)

Matrix Multiply

(6,90)

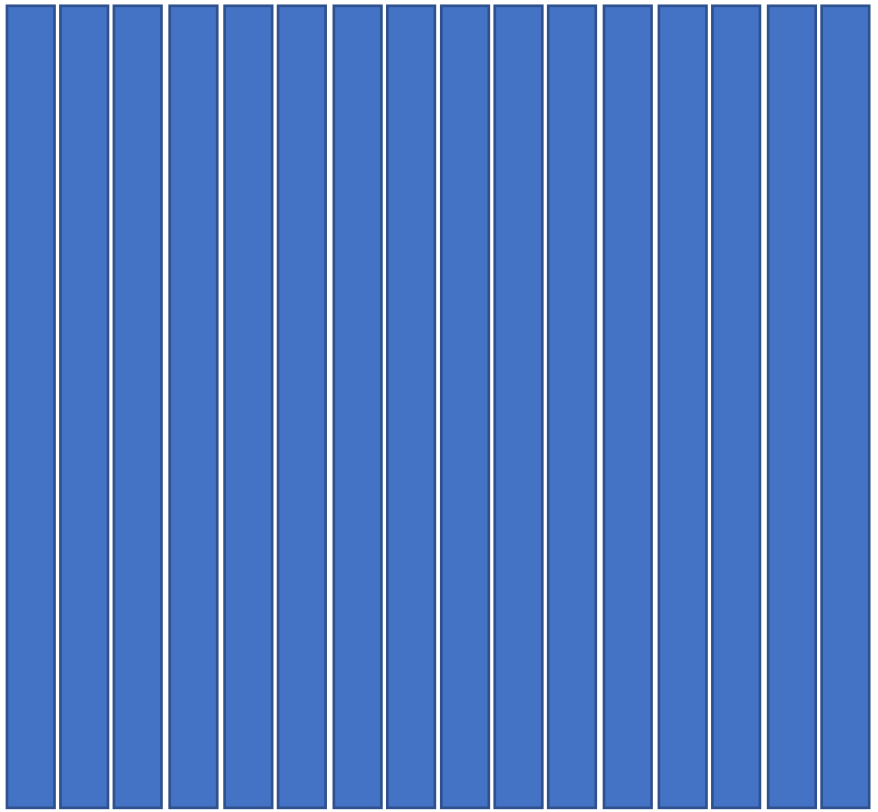


(90,16)

(6,16)

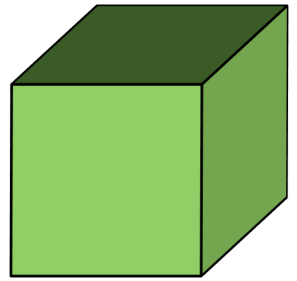
Reshape Output

(6,90)



(90,16)

(6,16)



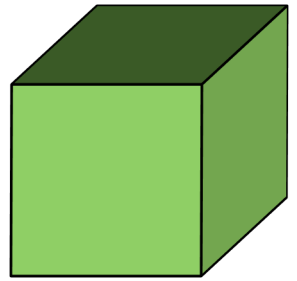
(4,4,6)

In General

(K, F)

(F, N)

(K, N)



$$F = f * f * c_{in}$$

$$N = \left(\frac{h + 2p - f}{s} + 1\right)\left(\frac{w + 2p - f}{s} + 1\right)$$

$$\left(\frac{h + 2p - f}{s} + 1, \frac{w + 2p - f}{s} + 1, K\right)$$

In Code

- Transforming the input volume is done by a well known function named im2col
  - This is the hard part
  - This part takes A LOT of memory due to repetition of elements
- Transforming weight matrix is simple reshape `w.reshape(K, -1)`
- Transforming the final output is also a simple reshape

Alternative: Use Fourier Transform

Convolution Theorem:

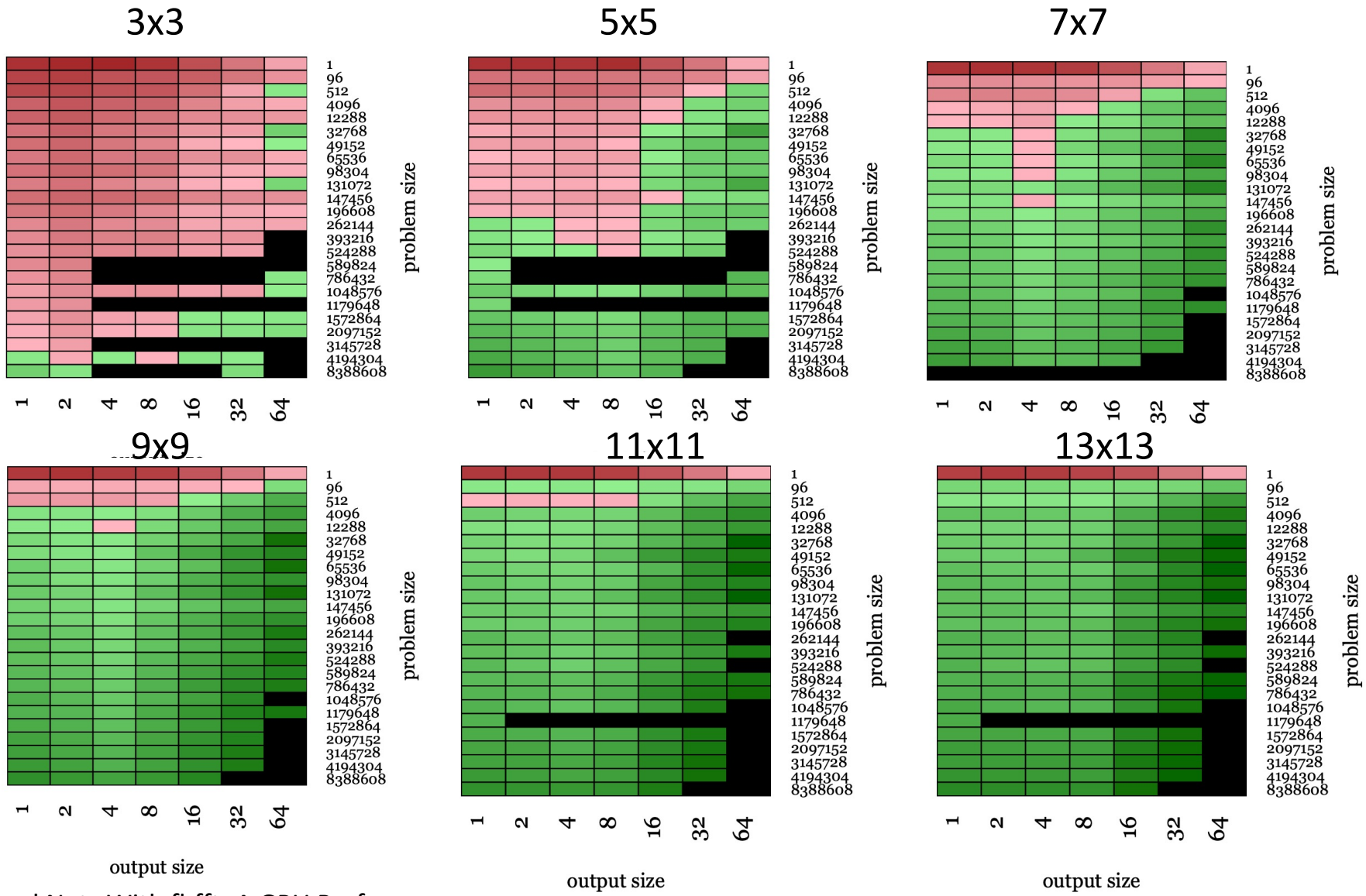
$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

The Fourier Transform of a convolution of two signals is equal to the elementwise product of the Fourier Transform of each respective signal

$$V_{in} * w = \mathcal{F}^{-1}\{\mathcal{F}\{V_{in}\} \cdot \mathcal{F}\{w\}\}$$

https://en.wikipedia.org/wiki/Convolution_theorem

Alternative: Use Fourier Transform



"Fast Convolutional Nets With fbfft: A GPU Performance Evaluation", Vasilache et al., 2015, https://arxiv.org/pdf/1412.7580.pdf

Learning Objectives

- A lite introduction to implementation and vectorization details for Convolutional Layers
- Understand enough about vectorization to do Assignment 4
- Understand how Convolution can be implemented as matrix multiplication