CSE 435 – Pattern Recognition

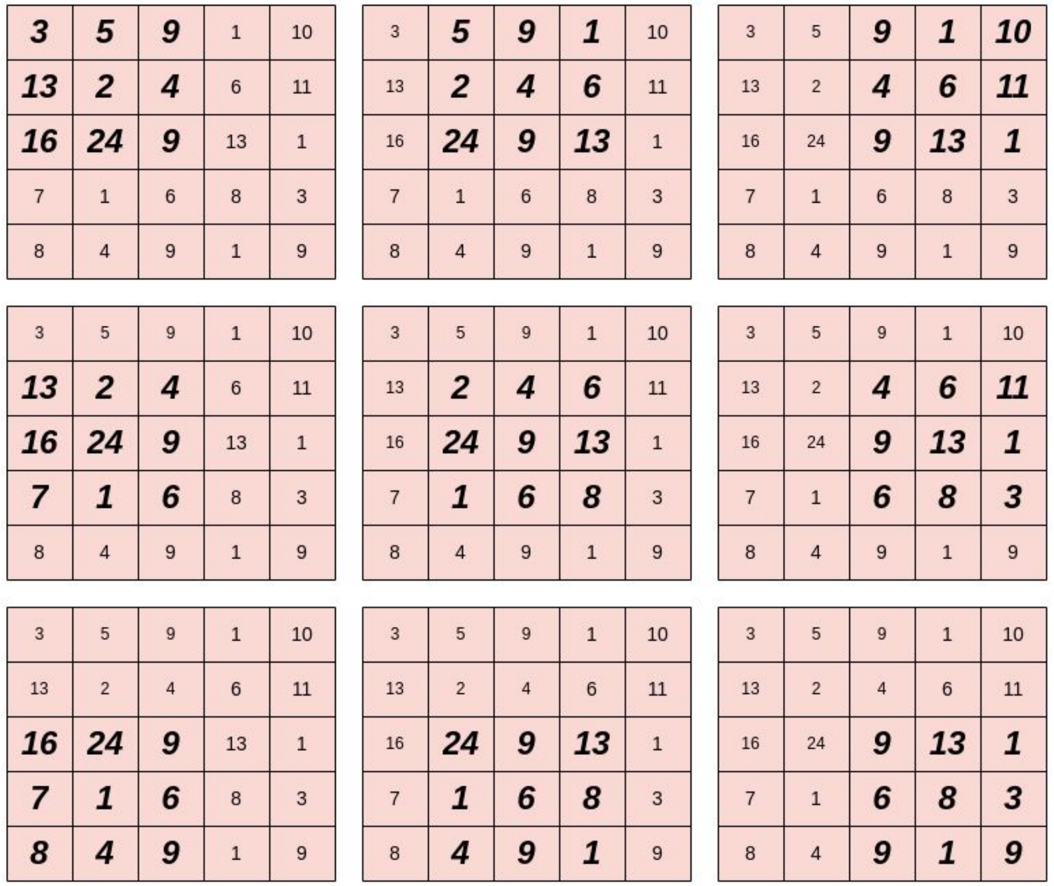
CNN Output Calculation

Tanvir Azhar
Lecturer, EDU

Convolution

- Generally, convolution is a mathematical operation on two functions where two sources of information are combined to generate an output function.
- In computer vision, convolution is performed between an image I and a filter K that is defined as a small matrix. First, the filter passes successively through every pixel of the 2D input image.
- In each step, we perform an elementwise multiplication between the pixels of the filter and the corresponding pixels of the image.
- Then, we sum up the results into a single output pixel. After repeating this procedure for every pixel, we end up with a 2D output matrix of features.

Convolution Operation



3	**5**	**9**	1	10
13	**2**	**4**	6	11
16	**24**	**9**	13	1
7	1	6	8	3
8	4	9	1	9

3	**5**	**9**	**1**	10
13	**2**	**4**	**6**	11
16	**24**	**9**	**13**	1
7	1	6	8	3
8	4	9	1	9

3	5	**9**	**1**	**10**
13	2	**4**	**6**	**11**
16	24	**9**	**13**	**1**
7	1	6	8	3
8	4	9	1	9

3	5	9	1	10
13	**2**	**4**	6	11
16	**24**	**9**	13	1
7	**1**	**6**	8	3
8	4	9	1	9

3	5	9	1	10
13	**2**	**4**	**6**	11
16	**24**	**9**	**13**	1
7	**1**	**6**	**8**	3
8	4	9	1	9

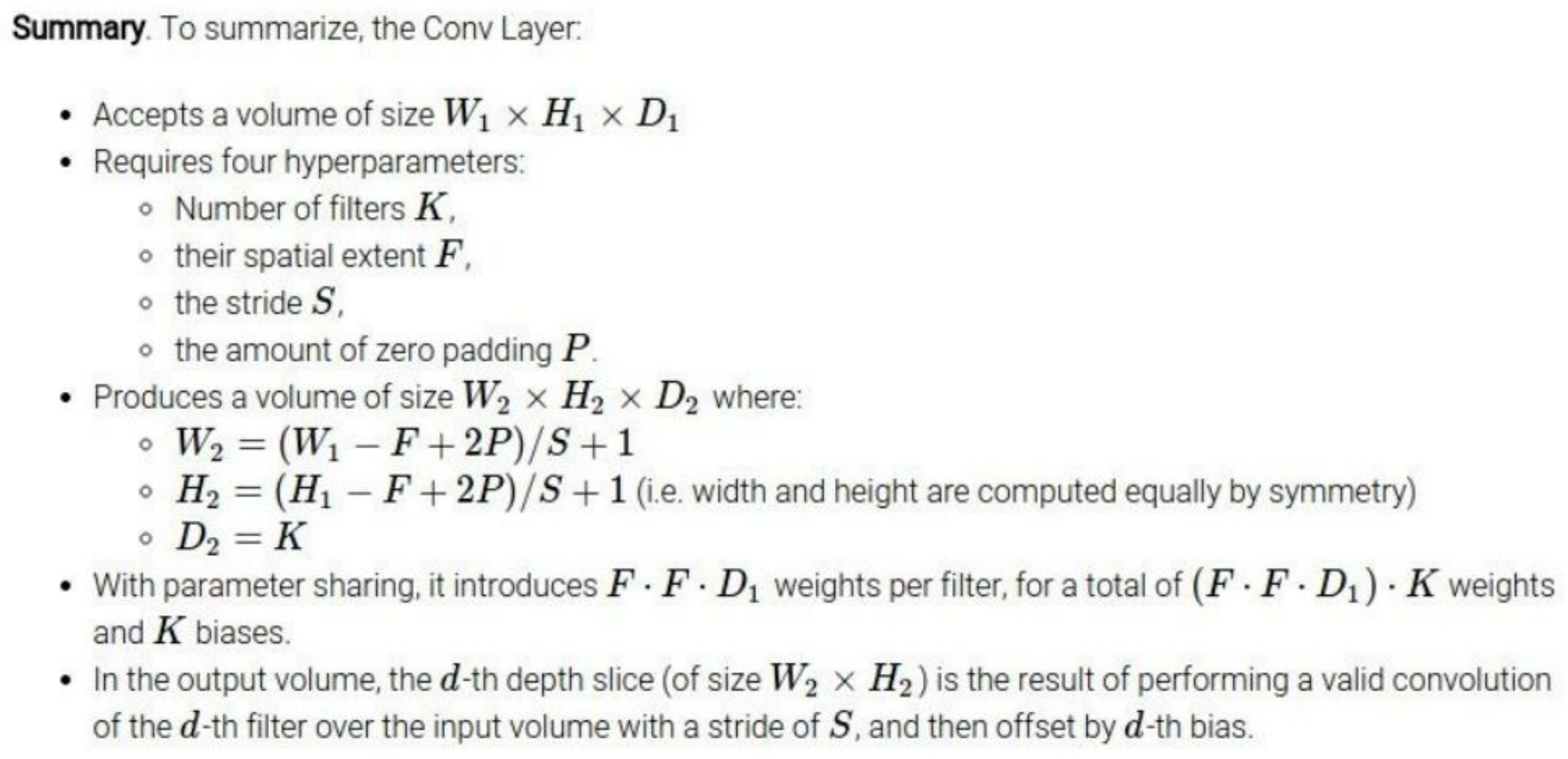
3	5	9	1	10
13	2	**4**	**6**	**11**
16	24	**9**	**13**	**1**
7	1	**6**	**8**	**3**
8	4	9	1	9

3	5	9	1	10
13	2	4	6	11
16	**24**	**9**	13	1
7	**1**	**6**	8	3
8	**4**	**9**	1	9

3	5	9	1	10
13	2	4	6	11
16	**24**	**9**	**13**	1
7	**1**	**6**	**8**	3
8	**4**	**9**	**1**	9

3	5	9	1	10
13	2	4	6	11
16	24	**9**	**13**	**1**
7	1	**6**	**8**	**3**
8	4	**9**	**1**	**9**

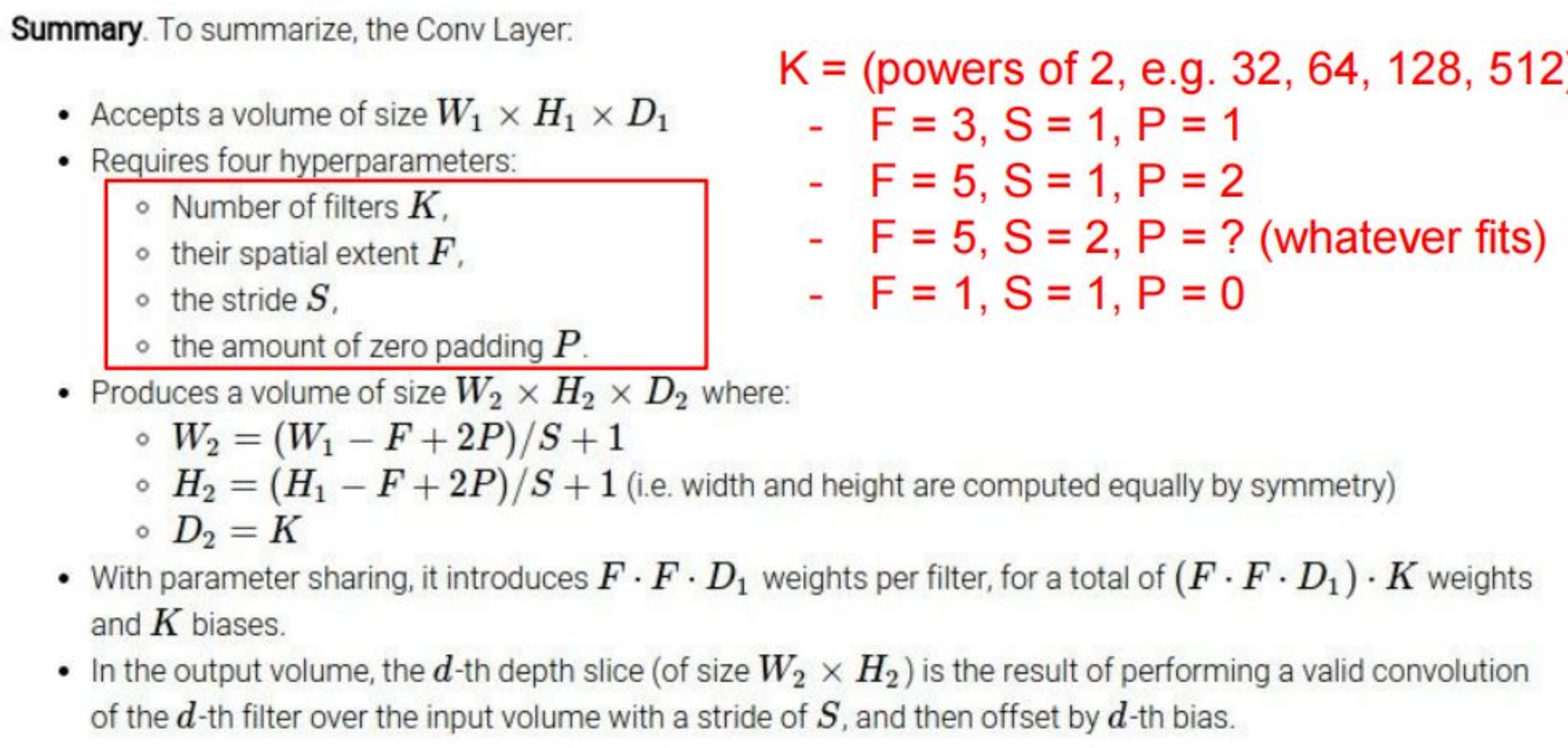
Summary of Conv Layer



Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary of Conv Layer



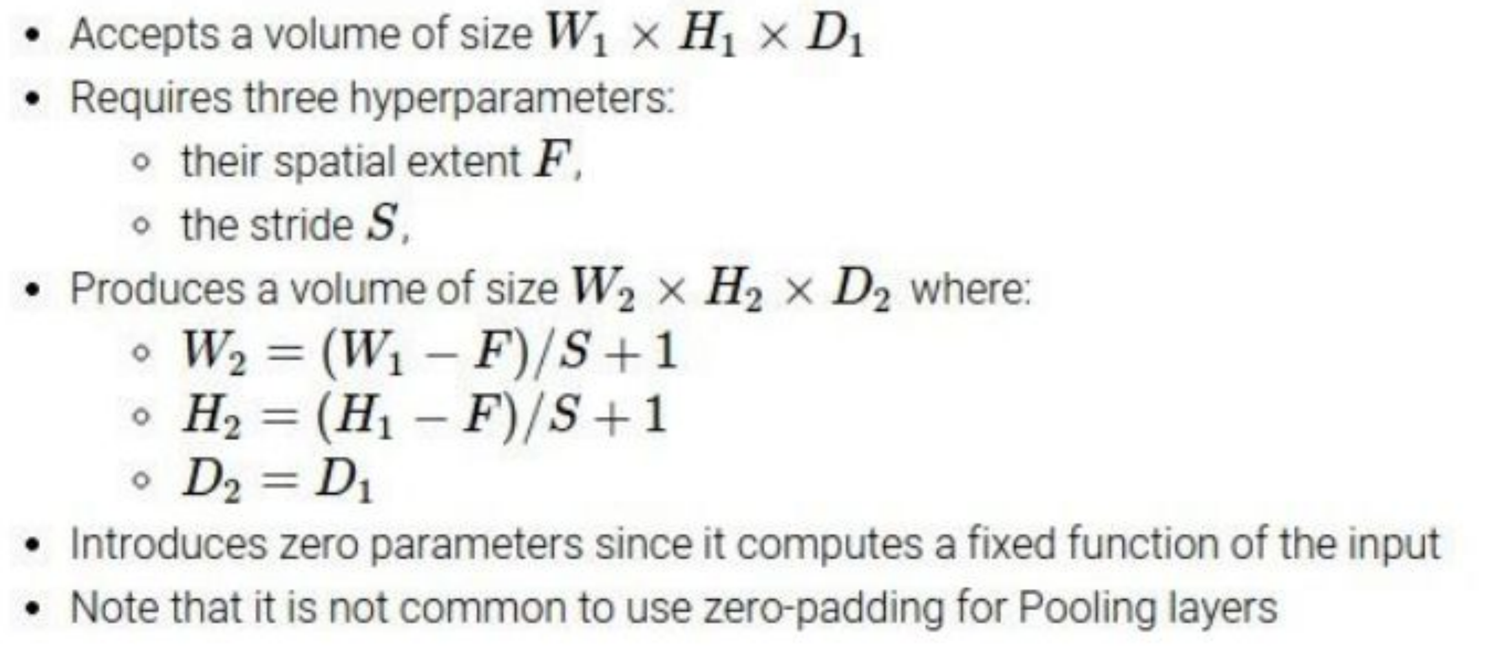
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

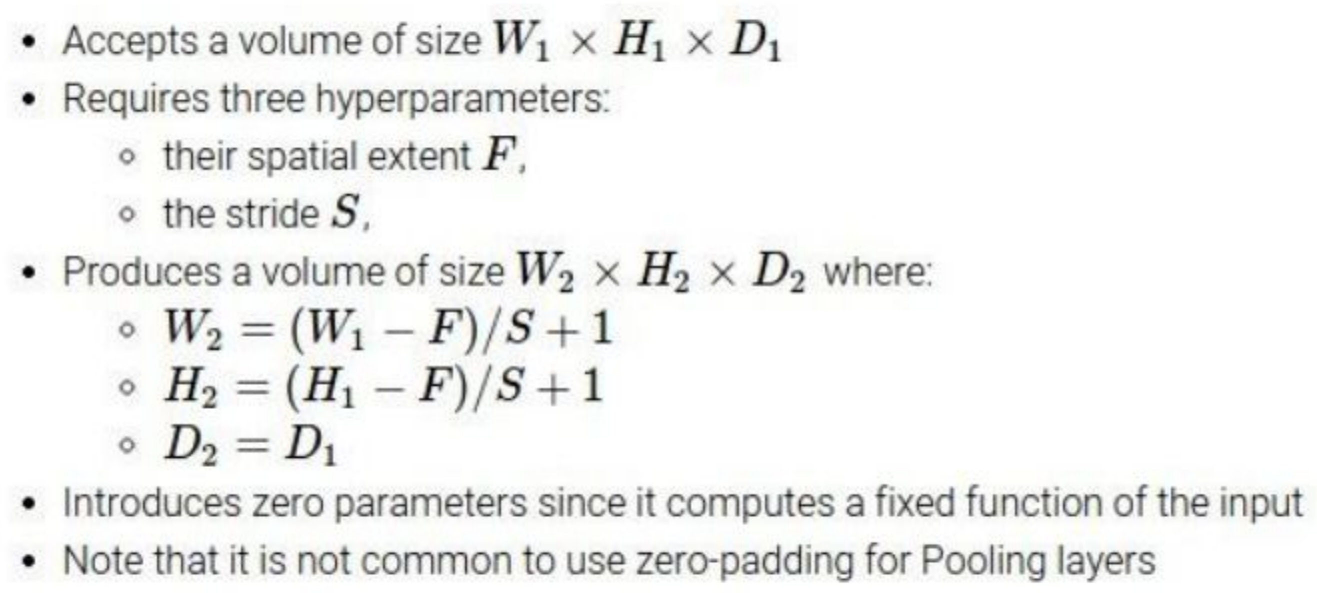
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary of MaxPooling



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Summary of MaxPooling



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

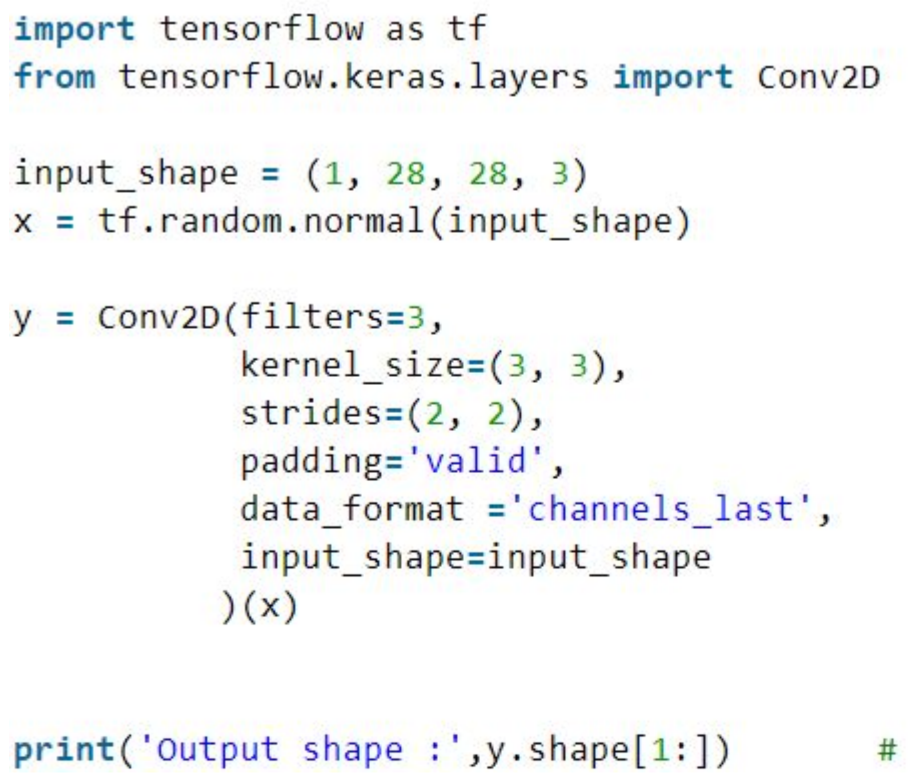
Common settings:

F = 2, S = 2
F = 3, S = 2

Padding

- Types of padding:
  - Valid Padding
  - same Padding
- Valid Padding
  - Valid padding is a technique used in convolutional neural networks (CNNs) to process the input data without adding any additional rows or columns of pixels around the edges of the data.
  - This means that the size of the output feature map is smaller than the size of the input data.
  - Valid padding is used when it is desired to reduce the size of the output feature map in order to reduce the number of parameters in the model and improve its computational efficiency.

Padding



```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D

input_shape = (1, 28, 28, 3)
x = tf.random.normal(input_shape)

y = Conv2D(filters=3,
           kernel_size=(3, 3),
           strides=(2, 2),
           padding='valid',
           data_format ='channels_last',
           input_shape=input_shape
          )(x)


print('Output shape :',y.shape[1:])       #
```

Formula to find the output shape of valid padding:

$$Output_w = \frac{I_w K_w + 1}{s_w}$$
$$Output_h = \frac{I_h K_h + 1}{s_h}$$

Where, I_w, I_h= Number of rows and column of input data.

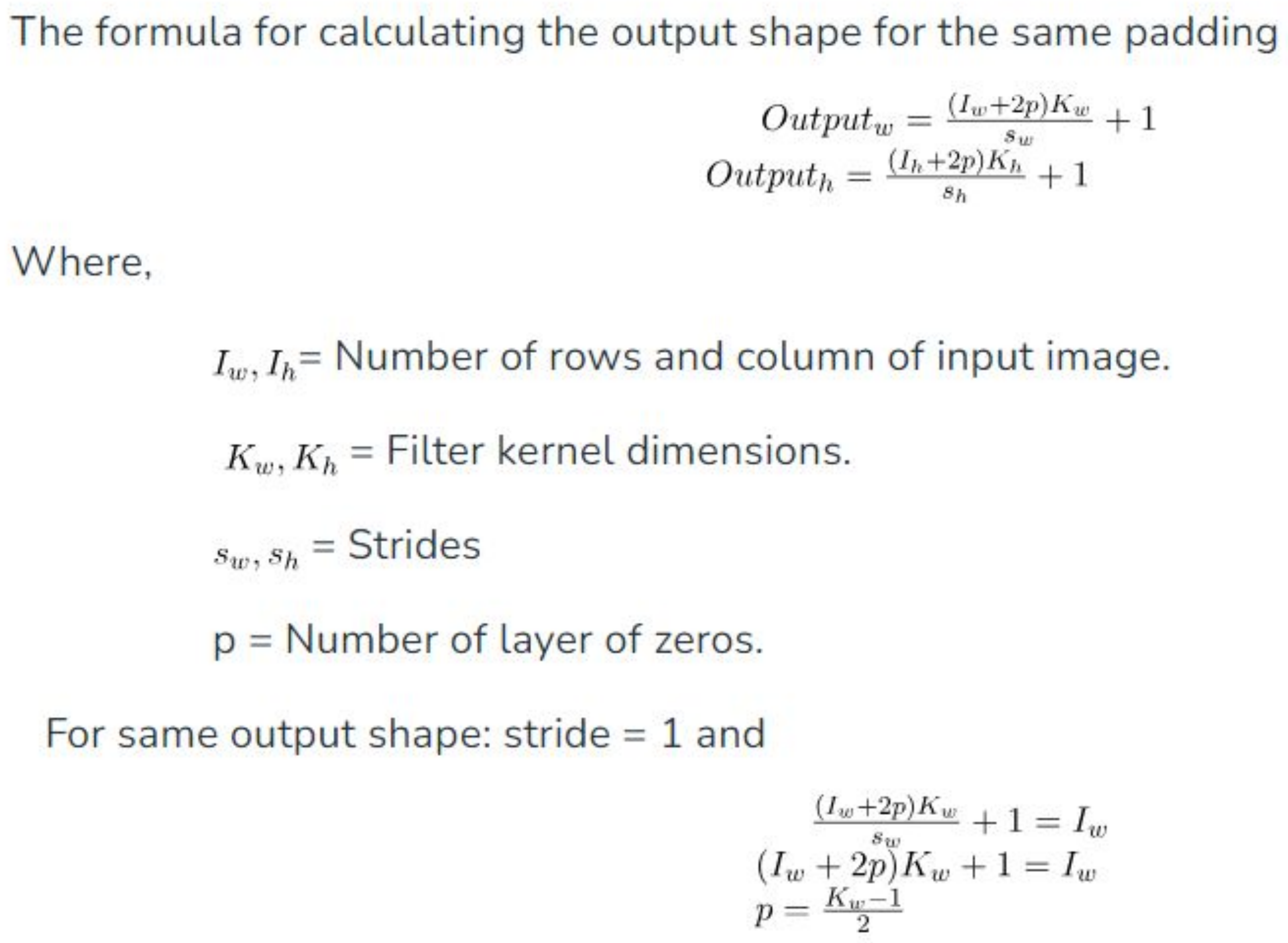
K_w, K_h = Filter kernel dimensions,

s_w, s_h = Stride

Padding

- Same Padding
    - Unlike valid padding, same padding adds additional rows and columns of pixels around the edges of the input data so that the size of the output feature map is the same as the size of the input data
    - This is achieved by adding rows and columns of pixels with a value of zero around the edges of the input data before the convolution operation.

Padding



The formula for calculating the output shape for the same padding

$$Output_w = \frac{(I_w+2p)K_w}{s_w} + 1$$
$$Output_h = \frac{(I_h+2p)K_h}{s_h} + 1$$

Where,

I_w, I_h= Number of rows and column of input image.

K_w, K_h = Filter kernel dimensions.

s_w, s_h = Strides

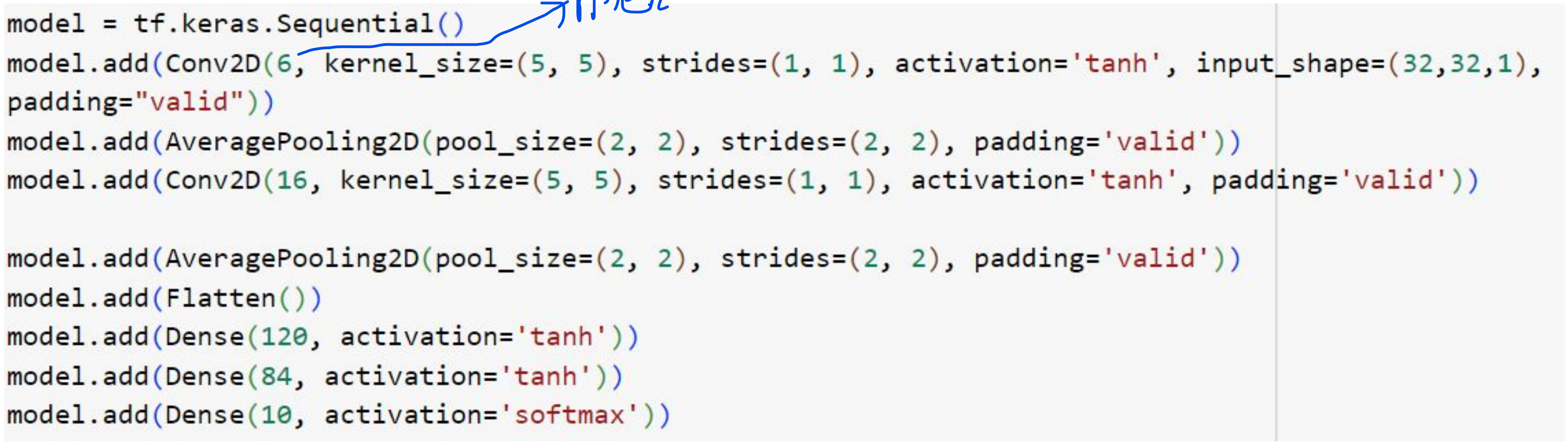
p = Number of layer of zeros.

For same output shape: stride = 1 and

$$\frac{(I_w+2p)K_w}{s_w} + 1 = I_w$$
$$(I_w + 2p)K_w + 1 = I_w$$
$$p = \frac{K_w - 1}{2}$$

Example

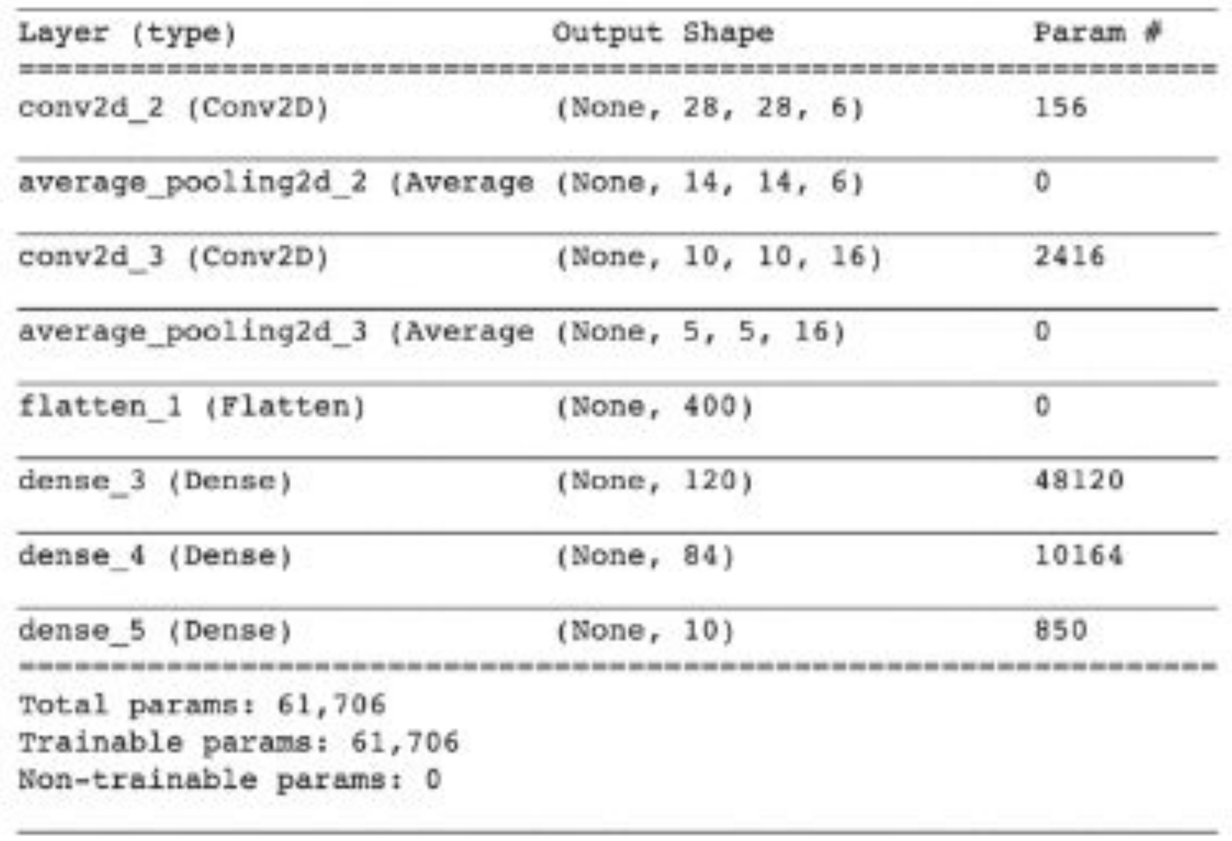
filter



```
model = tf.keras.Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=(32,32,1),
padding="valid"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(10, activation='softmax'))
```

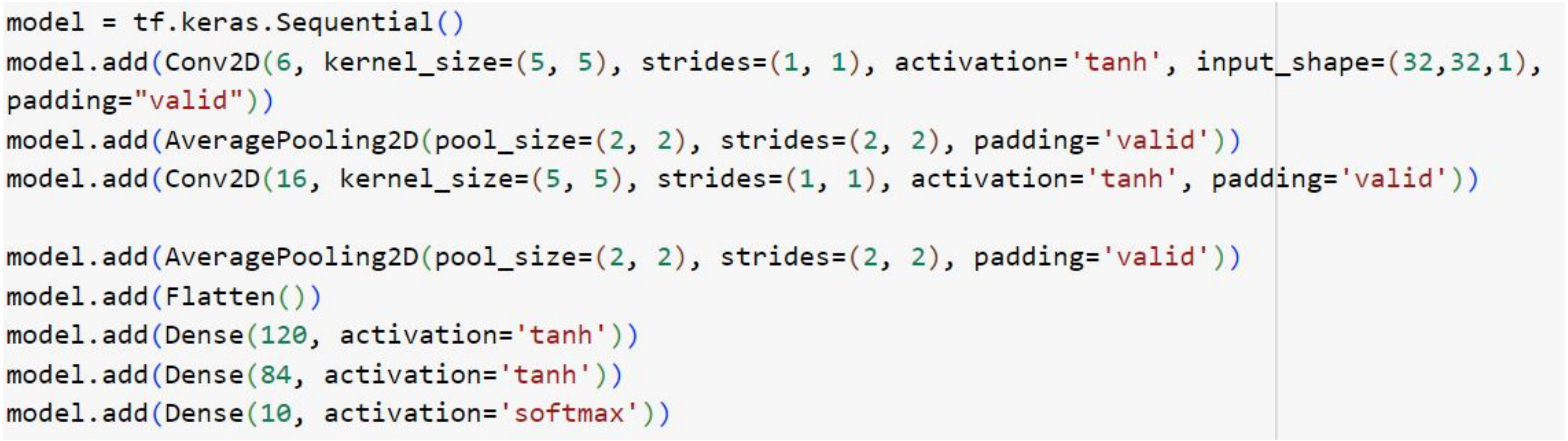
Example



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_2 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_5 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

Receptive Field Calculation

- Calculate the receptive field in the 2nd conv2D layer

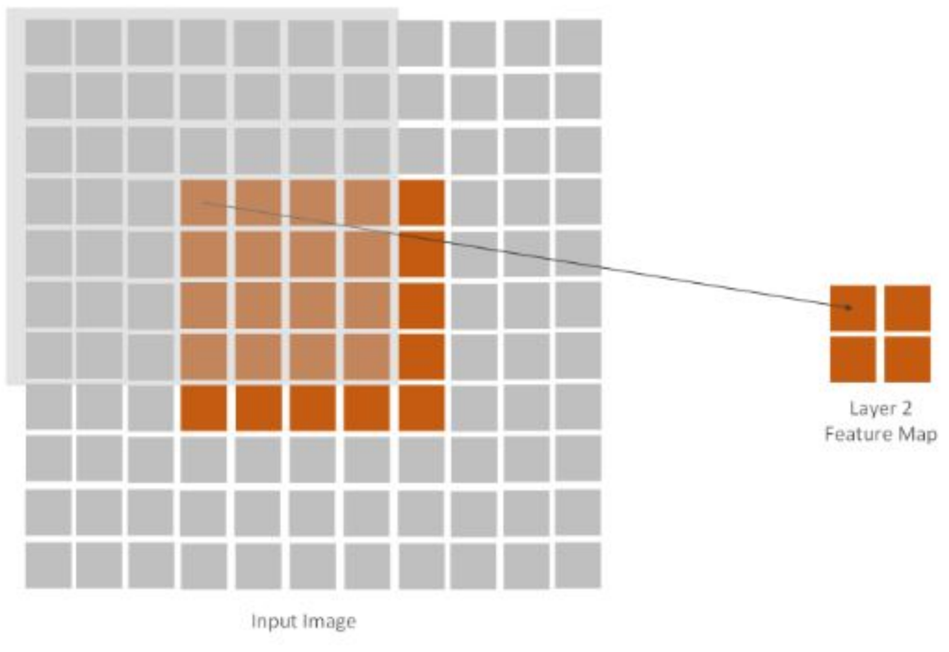


```
model = tf.keras.Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=(32,32,1),
padding="valid"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(10, activation='softmax'))
```

Receptive Field Calculation

- Receptive fields are defined by the center and the size. The center of the receptive field is very important since it decides the importance of the pixel. If the pixel is located closer to the center its importance in that particular computation is higher. This means that the CNN feature focuses more on the central pixel of the receptive field.



Receptive Field of the Second Layer

Receptive Field Calculation

Output of the network: $$n_i = \frac{n_{i-1} + 2p - k}{s} + 1$$

Jumping window: $$j_i = j_{i-1} * s$$

Receptive field: $$r_i = r_{i-1} + (k-1) \underbrace{\prod_{k=1}^{i-1} s_k}$$

$\rightarrow j_i = s_k$

Dilation Rate Changes

- Generally in CNN dilation rate is (1,1). If the dilation rate changes, it will affect the kernel size (filter size) and receptive field region.
- Thus, we need to calculate the new kernel size due to the dilation change effect. It only affects the conv layer where we have induced the changes.
- k= kernel Size, d=dilation size then new kernel will be calculated by the following formula.

$$\hat{k} = k + (k-1)(d-1).$$