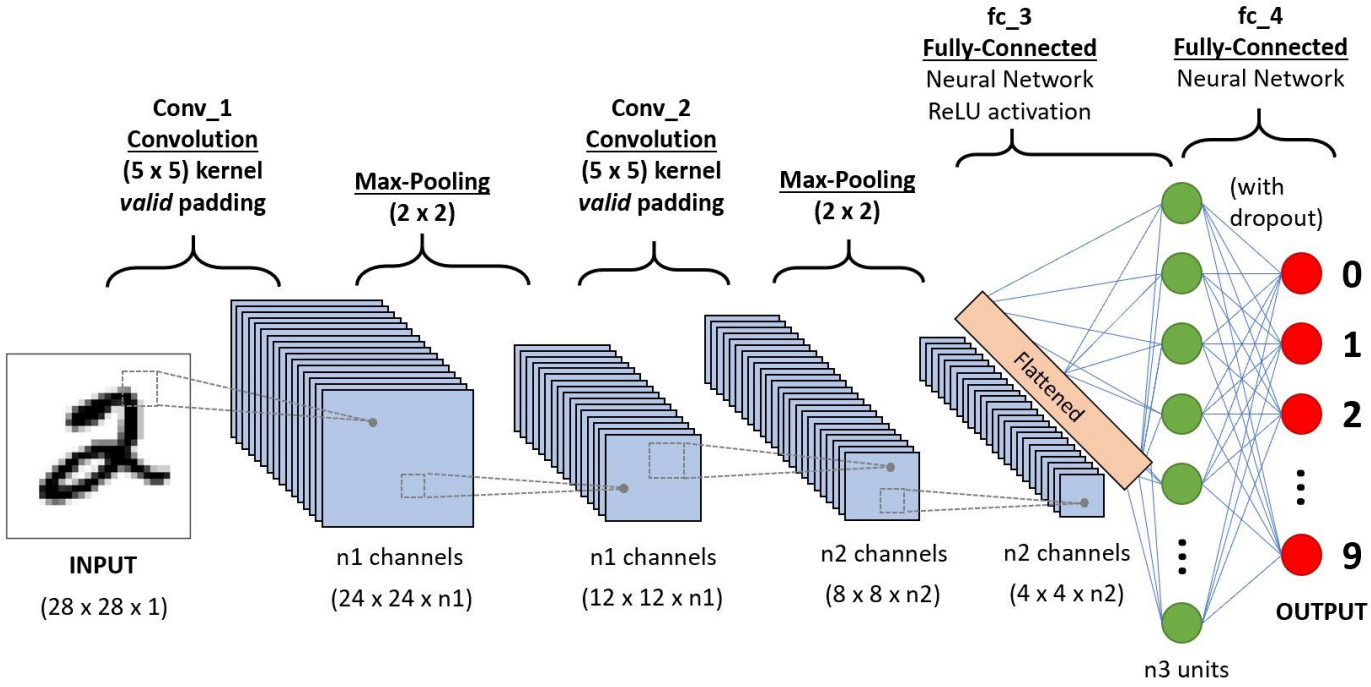
# Convolutional neural networks (CNN)

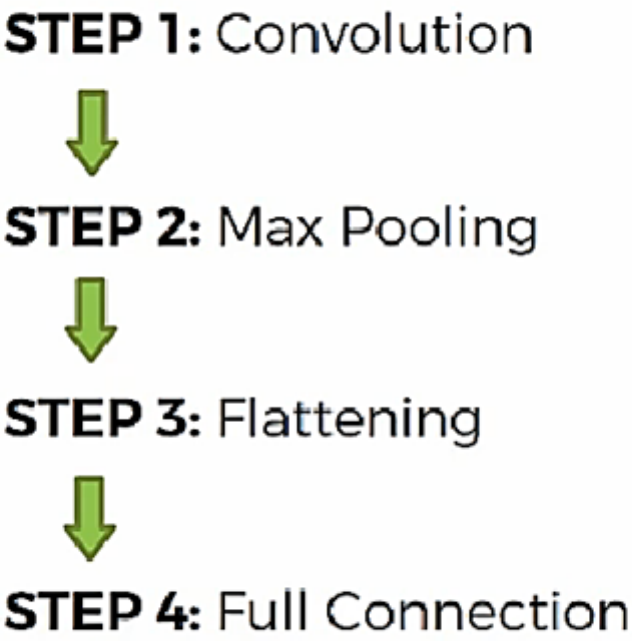
- Cnn are one of the most popular models used today. This neural network computational model uses a variation of multilayer perceptrons and contains one or more convolutional layers that can be either entirely connected or pooled.
- These convolutional layers create feature maps that record a region of image which is ultimately broken into rectangles and sent out for nonlinear processing.




- Let us suppose this in the input matrix of 5×5 and a filter of matrix 3X3, for those who don't know what a filter is a set of weights in a matrix applied on an image or a matrix to obtain the required features, please search on convolution if this is your first time!

**Note: We always take the sum or average of all the values while doing a convolution.**

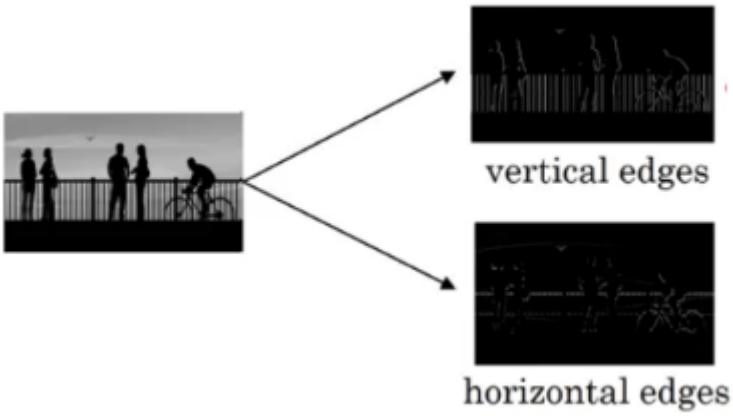
## Steps Involve in CNN

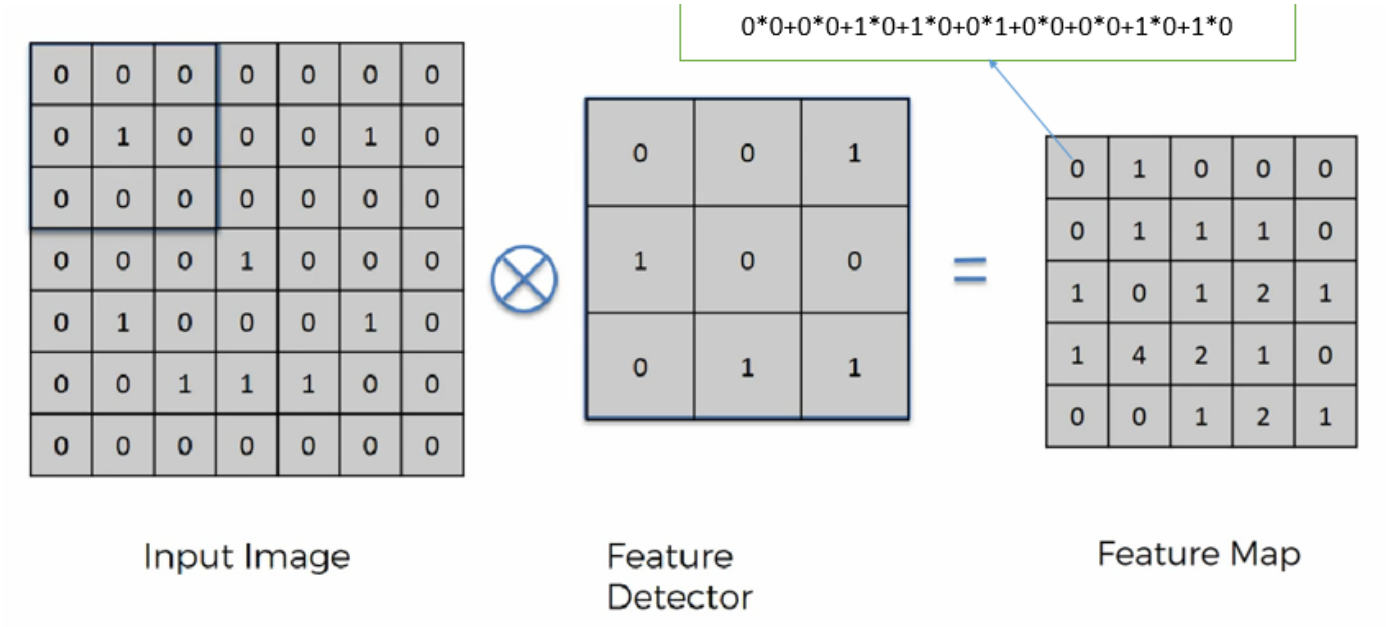



## 1. Edge Detection (Convolution)

- In the previous article, we saw that the early layers of a neural network detect edges from an image. Deeper layers might be able to detect the cause of the objects and even more deeper layers might detect the cause of complete objects (like a person's face).

In this section, we will focus on how the edges can be detected from an image. Suppose we are given the below image: As you can see, there are many vertical and horizontal edges in the image. The first thing to do is to detect these edges:






- So, we take the first 3 X 3 matrix from the 7 X 7 image and multiply it with the filter. Now, the first element of the (n-k+1 x n-k+1) i.e (7-3+1 X 7-3+1) 5 X 5 output will be the sum of the element-wise product of these values, i.e. 0*0*+0*0+1*0*+1*0+0*1*+0*0+0*0*+1*0+1*0 =0. To calculate the second element of the 5 X 5 output, we will shift our filter one step towards the right and again get the sum of the element-wise product:

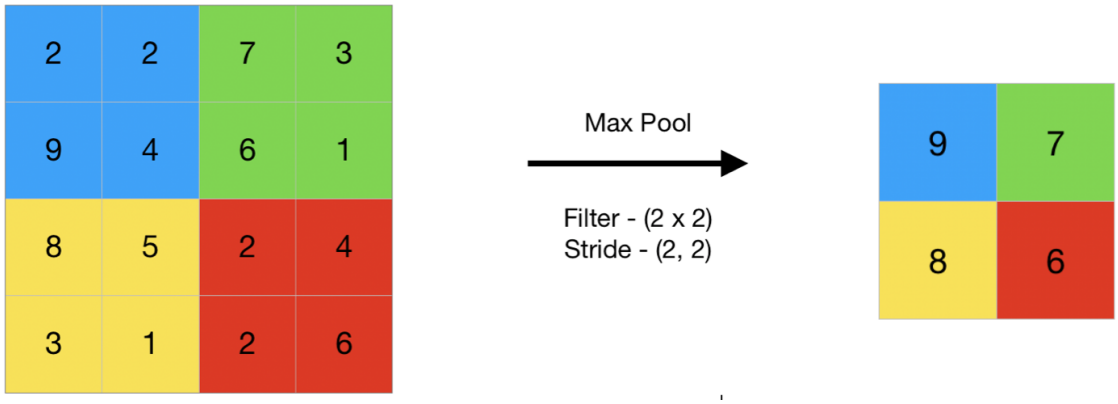
## 2. Pooling

- A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling.

**Types of Pooling Layers :-**

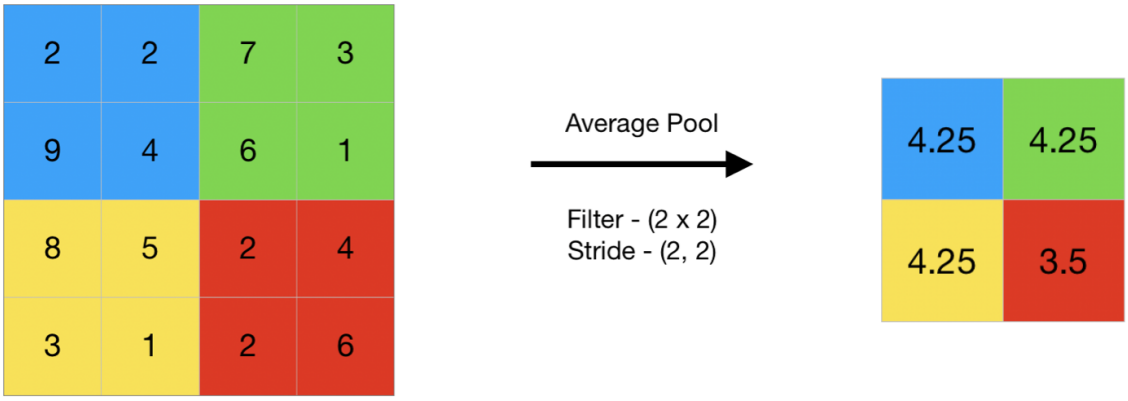
`1. Max Pooling`

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



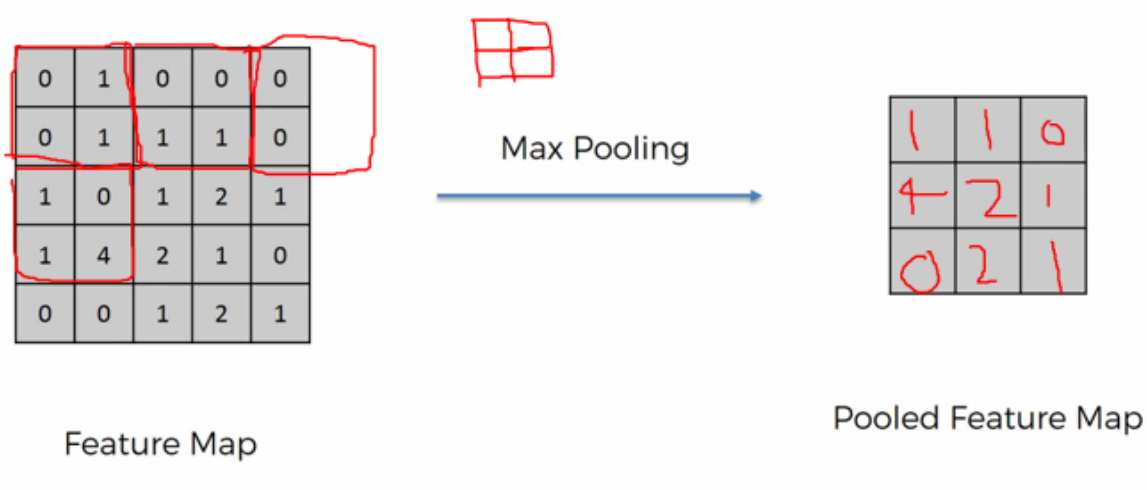

`2. Average Pooling`

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.




- More On Pooling https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/)

**Now Apply Pooling in our above Feature Map**

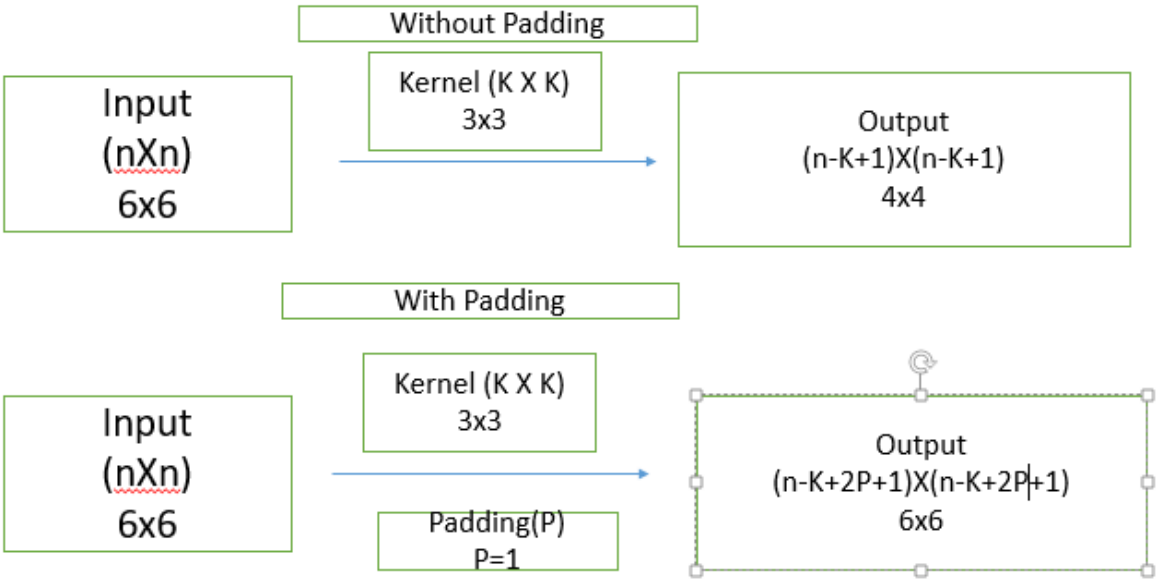



# Problem with Simple Convolution Layers

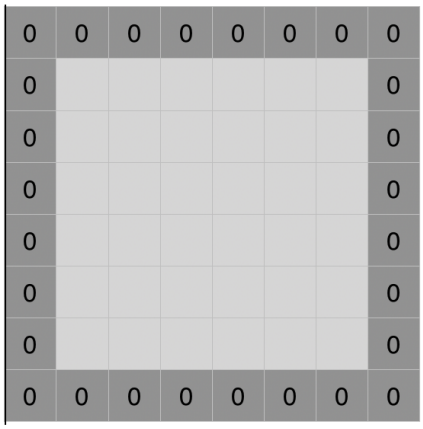
- While applying convolutions we will not obtain the output dimensions the same as input we will lose data over borders so we append a border of zeros and recalculate the convolution covering all the input values.

    1. Padding
    2. Striding

## 1. Padding




- See In without padding our input is 6x6 but output image goes down into 4x4 . so by using padding we got the same result.Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above.




Zero-padding added to image

- So padding prevents shrinking as, if p = number of layers of zeros added to the border of the image, then our (n x n) image becomes (n + 2p) x (n + 2p) image after padding. So, applying convolution-operation (with (f x f) filter) outputs (n + 2p – f + 1) x (n + 2p – f + 1) images. For example, adding one layer of padding to an (8 x 8) image and using a (3 x 3) filter we would get an (8 x 8) output after performing convolution operation.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 60 | 113 | 56 | 139 | 85 | 0 |
| 0 | 73 | 121 | 54 | 84 | 128 | 0 |
| 0 | 131 | 99 | 70 | 129 | 127 | 0 |
| 0 | 80 | 57 | 115 | 69 | 134 | 0 |
| 0 | 104 | 126 | 123 | 95 | 130 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

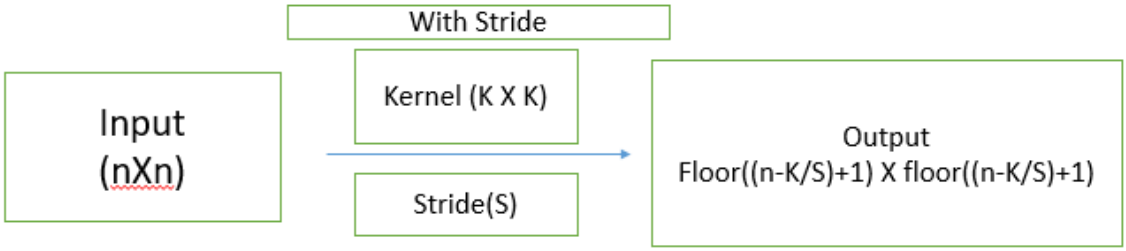
Kernel

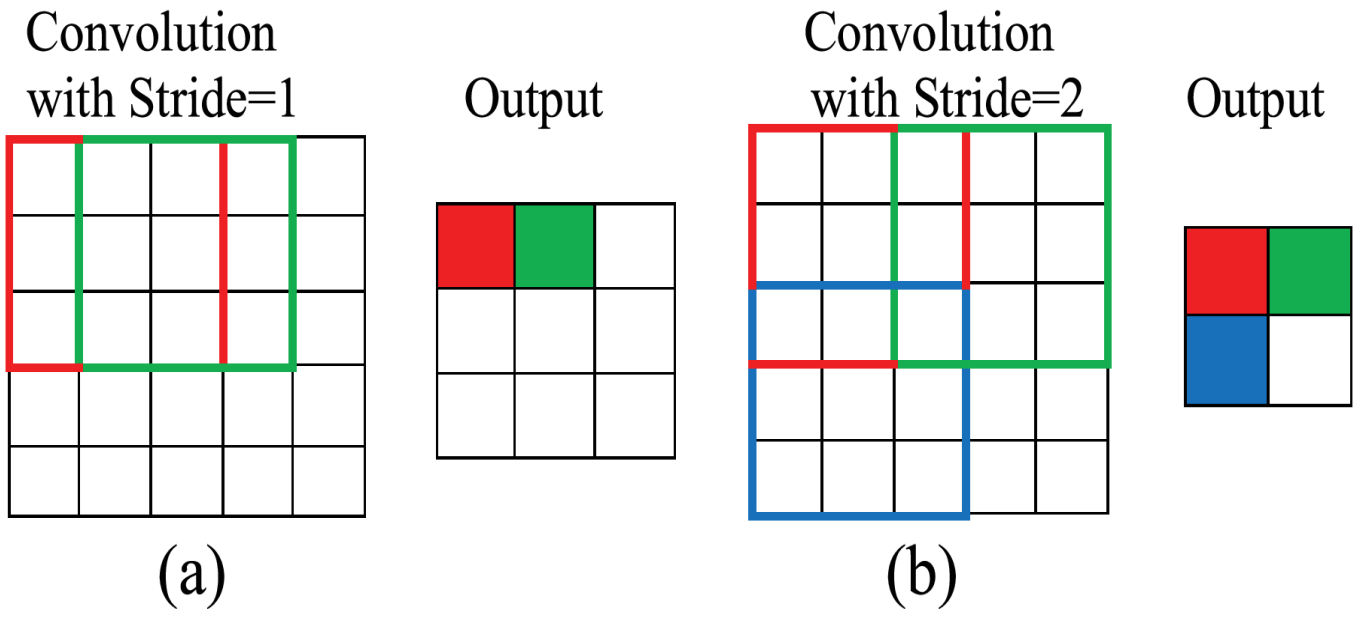
| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

| | | | | |
|---|---|---|---|---|
| 114 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

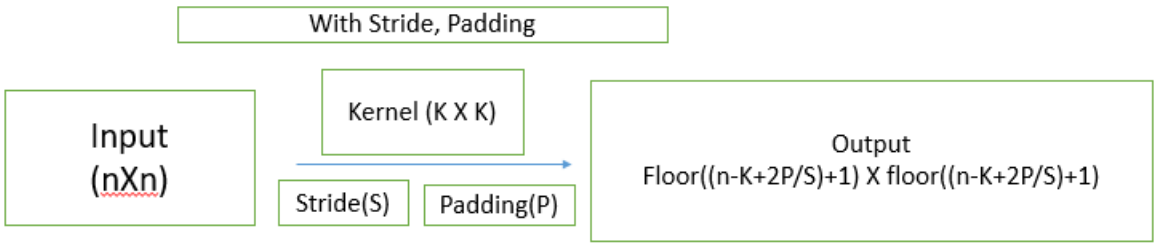
## 2. Strides

- It uses to reduce the size of matrix. if we sfited by 1 then we called stride=1 and if we sfited by 2 means stride = 2 so on.



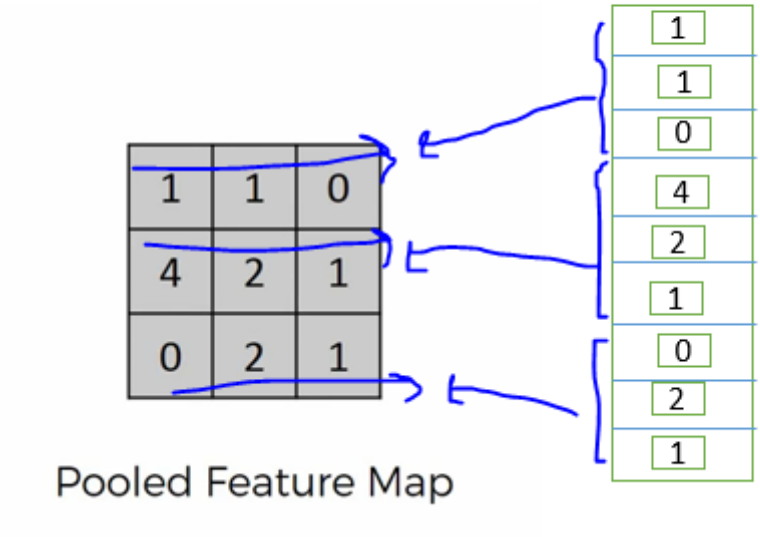




**Padding,Stride Put in One Equation**




# Step3 : Flattening

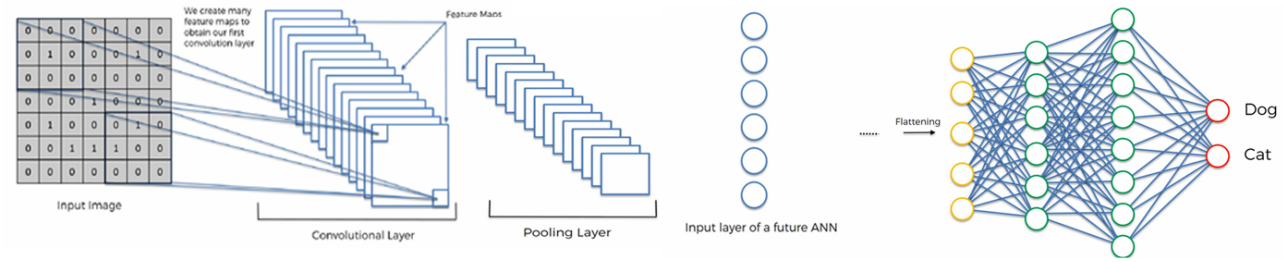
- Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.




# Step 4

## Complete CNN in one View

### here in last step we use full connection network



## This is a simple CNN Network

In [1]:

```
# Importing the libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [2]:

```
#data sugmentation
# Preprocessing the Training set
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
training_set = train_datagen.flow_from_directory('image_data/training',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 198 images belonging to 2 classes.
```

In [3]:

```
# Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('image_data/validation',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

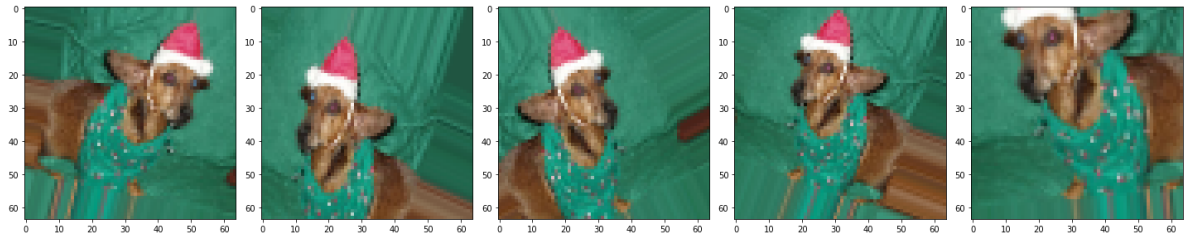
```
Found 100 images belonging to 2 classes.
```

In [4]:

```
## showing some image from training
import matplotlib.pyplot as plt
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20, 20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

In [5]:

```
images = [training_set[0][0][0] for i in range(5)]
plotImages(images)
```



## Model Build Use Only CNN

In [6]:

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

In [7]:

```
# Part 2 - Building the CNN

# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - # Adding a first convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32,padding="same",kernel_size=3, activation='relu',
## step 2 - #apply maxpool
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2)) ## Apply pooing stride

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32,padding='same',kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))


# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
tf.keras.layers.Dropout(0.5)

# Step 5 - Output Layer
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

In [8]:

```
# Part 3 - Training the CNN

# Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [9]:

```
# Training the CNN on the Training set and evaluating it on the Test set
history = cnn.fit(x = training_set, validation_data = test_set, epochs = 2)
```

```
Epoch 1/2
7/7 [==============================] - 4s 410ms/step - loss: 0.7529 - accura
cy: 0.4600 - val_loss: 0.6988 - val_accuracy: 0.5000
Epoch 2/2
7/7 [==============================] - 2s 308ms/step - loss: 0.6898 - accura
cy: 0.5349 - val_loss: 0.6932 - val_accuracy: 0.5100
```

## Save And Load Model

In [10]:

```
#save model
from tensorflow.keras.models import load_model
cnn.save('model_rcat_dog.h5')
```

In [11]:

```
from tensorflow.keras.models import load_model
# load model
model = load_model('model_rcat_dog.h5')
```

In [12]:

```
# Part 4 - Making a single prediction

import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('image_data/test/3285.jpg', target_size = (64,64))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
result
```

Out[12]:

```
array([[0.5059088]], dtype=float32)
```

In [13]:

```
if result[0]<=0.5:
    print("The image classified is cat")
else:
    print("The image classified is dog")

from IPython.display import Image
Image(filename='image_data/test/3285.jpg',height='200',width='200')
```

```
The image classified is dog
```

Out[13]: