Agenda:

Convolution neural networks:

- **Motivation for Convolutional neural networks (CNN)**
- **Edge detector**
- **Padding**
- **Strided convolution**
- **Conv on RGB**
- **One layers of CNN**
- **Example networks**

Keras library:

- **Example codes of keras programs**
- **Using open source implementation: available at GitHub**

Resources used Convolutional neural networks:

- **https://www.youtube.com/watch?v=ArPaAX_PhIs&index=1&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF**
- **https://github.com/Kulbear/deep-learning-coursera/blob/master/Convolutional%20Neural%20Networks/Convolution%20model%20-%20Step%20by%20Step%20-%20v1.ipynb**
- **https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/**

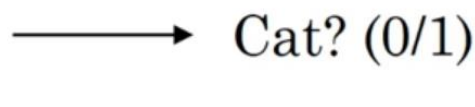
Resources used for Keras:

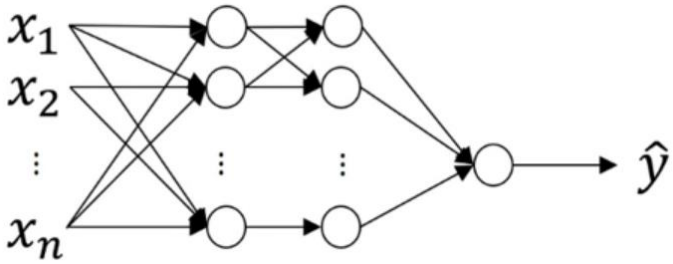
- **https://www.tensorflow.org/guide/keras**
- **https://github.com/Kulbear/deep-learning-coursera/blob/master/Convolutional%20Neural%20Networks/Keras%20-%20Tutorial%20-%20Happy%20House%20v1.ipynb**

Motivation for Convolutional neural networks (CNN):

In CV we deal with the problem, like image classification, object detection and location, style transformation, and super-resolution etc.

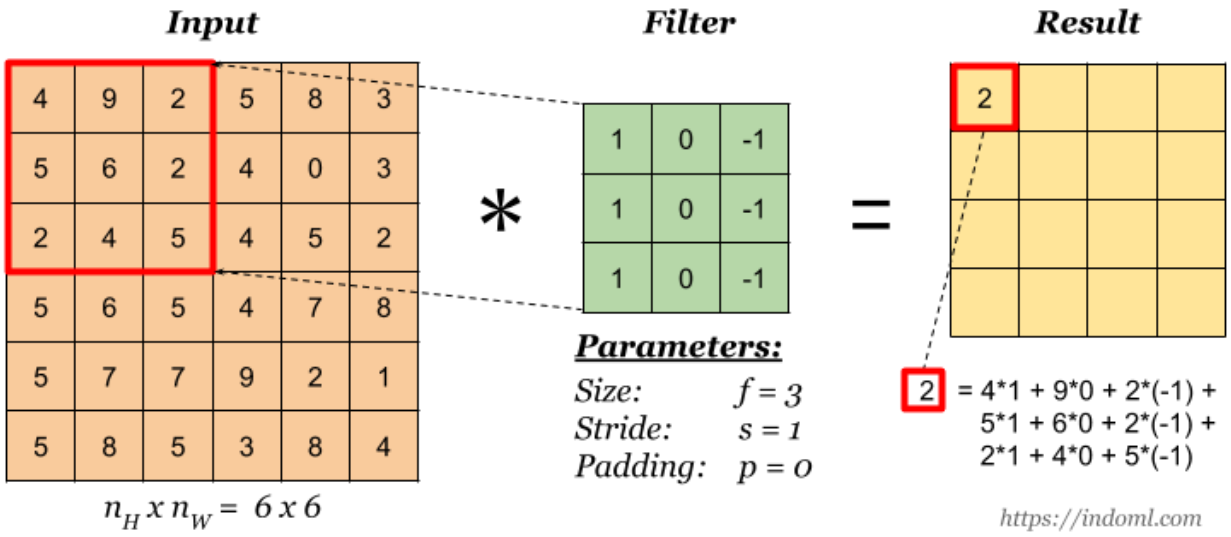
Deep learning on large images: consider the following LR (64x64x3=12288) and HR (1000x1000x3 = 3M)



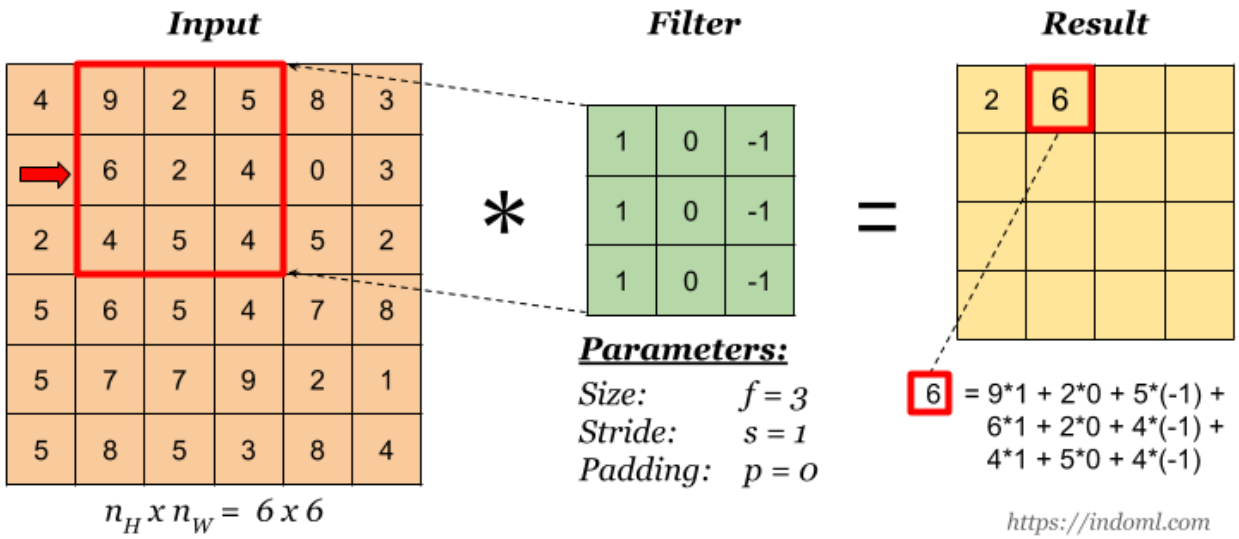


- In NN with input of HR image (3M), say hidden layer has 1000 neuron, it means weight matrix size is (1000 x3M = 3Billion) = > very larger number parameters.
- Issue with NN due to large number of parameters:
    - Computation cost
    - Overfitting: needs very large dataset to train the network
- Solution is CNN: needs lesser number of parameters to learn
    - Convolution operation is the fundamental building block of CNN.
    - In CNN the convolution operation detects edges at the starting layer, the smaller objects at intermediate layer detects, and at the last layer complete objects are detected

- **Edge Detection:** Vertical and horizontal edge detector.
- Vertical Edge detector:



- For next step move the overlay right one position (or according to the **stride** setting) and do the same calculation above to get the next result. And so on.

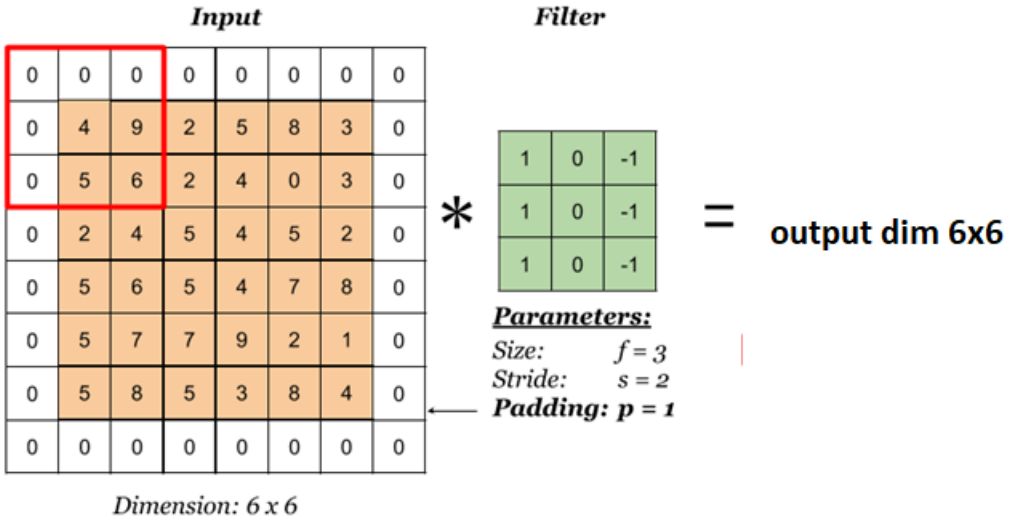


- **Fill the empty cells by performing convolution operations???**
- Let consider another example to see how convolution operation helps to identify the vertical edges more clearly.
- Horizontal edge detectors
- Sobel filter is an example of edge detector
- **In CNN, the weights (parameters) of detectors/ filters/kernels of different orientation (vertical, horizontal, 45, 90, 70 degree, etc.) are learnt during training operation.**

- **In Keras library: conv2D () function is used convolution operation**

Padding:

1. We know that after CONV operation, output dim = n-f+1 , n-f+1
2. Padding allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
3. It helps us keep more of the information at the border of an image.



Notice the dimension of the result has changed due to padding. Some padding terminologies used in machine learning libraries (like tensorflow):

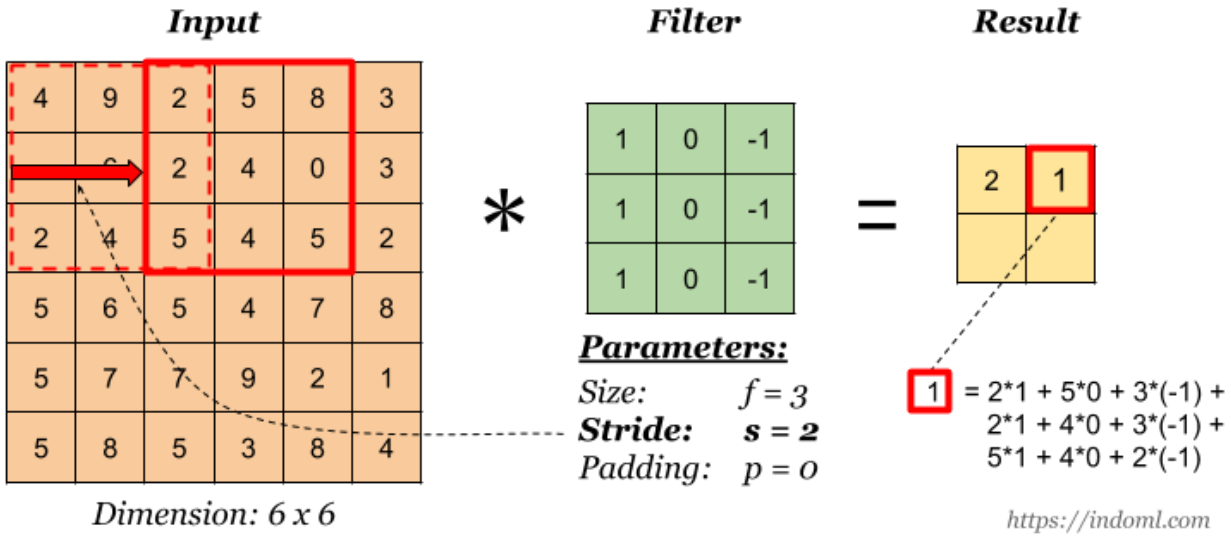
- "**valid**" padding: no padding
  - Output dim = n - f +1, n - f + 1
- "**same**" padding: padding so that the output dimension is the same as the input
  - Output dim = n +2p -f +1, n+2p-f+1
  - **Padding vs kernel size: Due to padding input size is equal to output,**

    n +2p -f +1 = n, p = (f-1)/2

    So, for kernel of size 3x3, p = 1

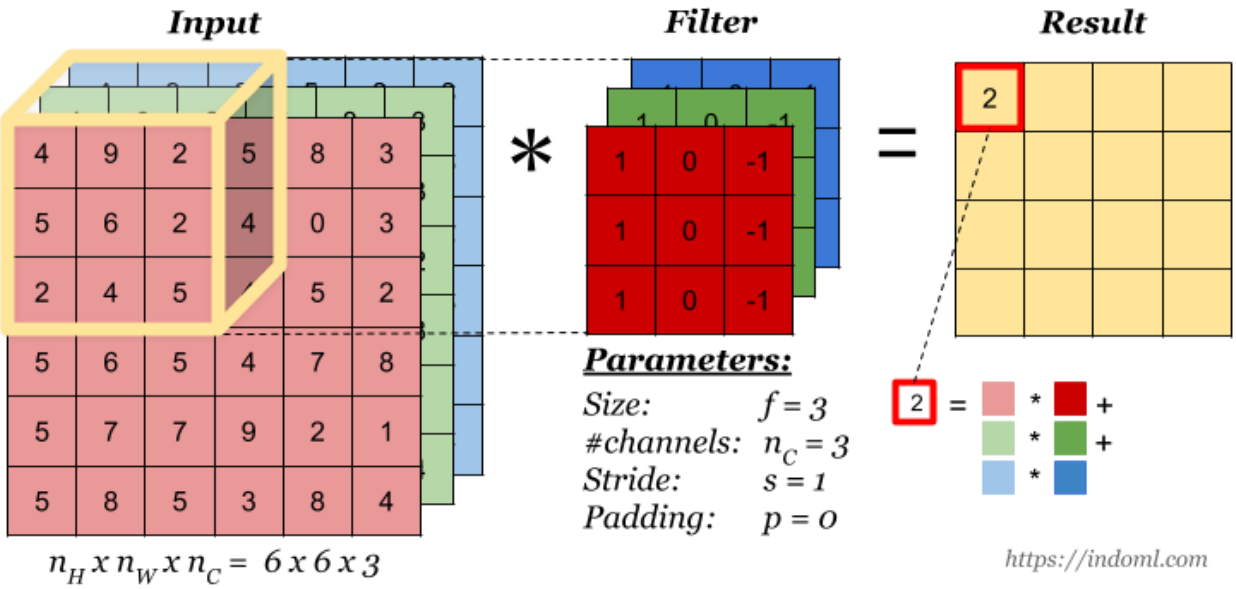
    For kernel of size 5x5, p = 2

Strided convolutions:



- Output dim = floor ((n +2p -f)/s+1, (n+2p-f)/s+1)

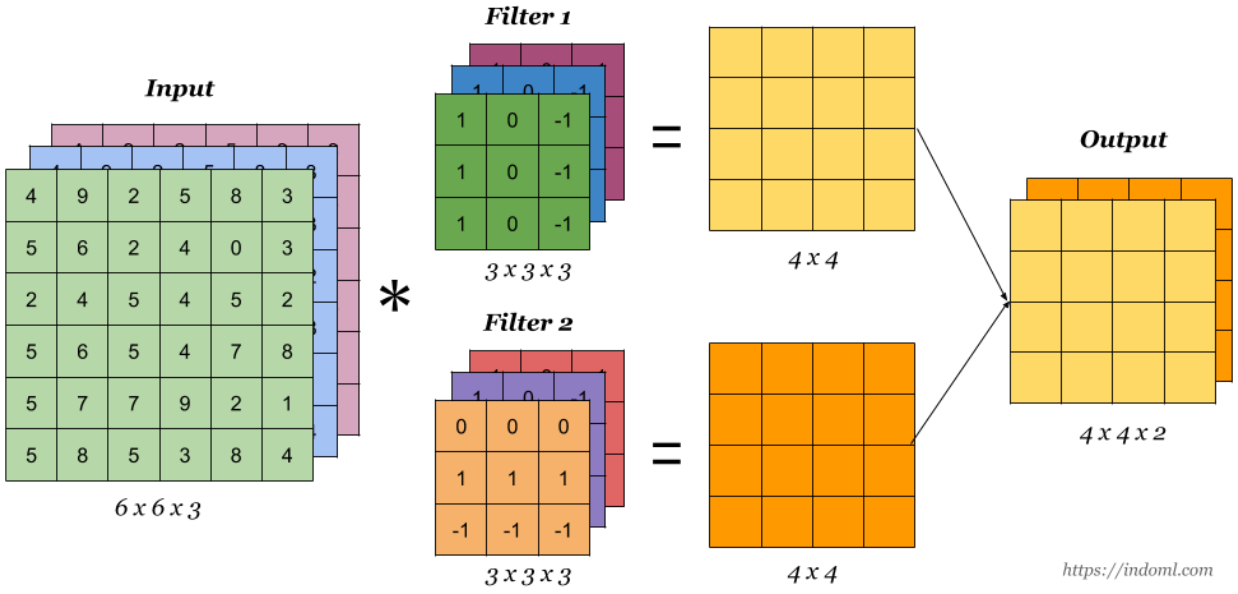
Convolution on RGB Image:



- Here for CONV operation 27 numbers are multiplied with corresponding 27 numbers on RGB image and added to produce the single activation value.
- Input dimension of feature maps must match with the number of channels of filters

Convolution Operation with Multiple Filters

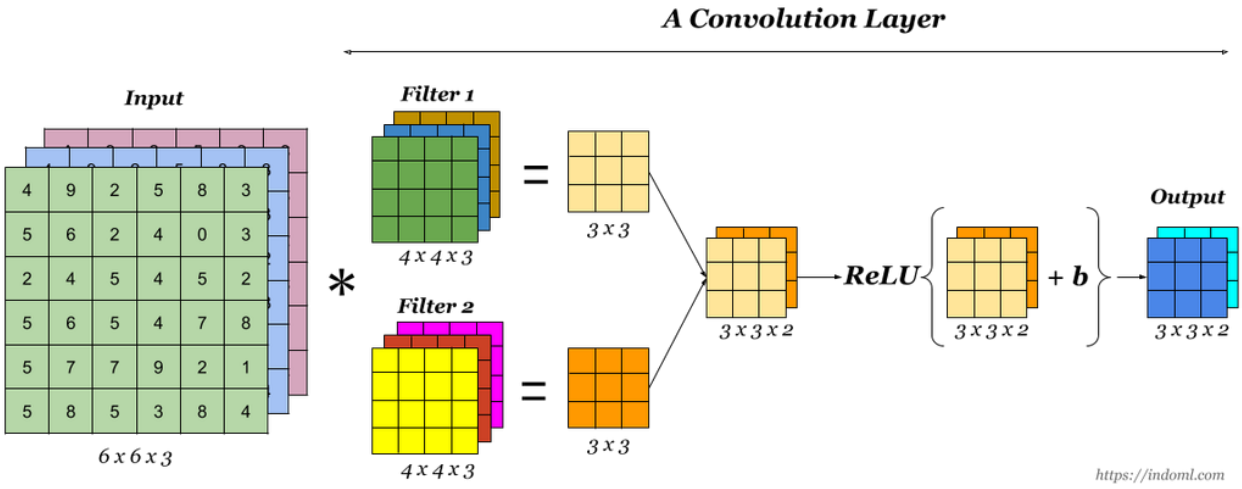
- Multiple filters can be used in a convolution layer to detect multiple features. The output of the layer then will have the same number of channels as the number of filters in the layer.



Output dim: $n \times n \times n_c * f \times f \times n_c \rightarrow n - f + 1 \times n - f + 1 \times n_{oc}$

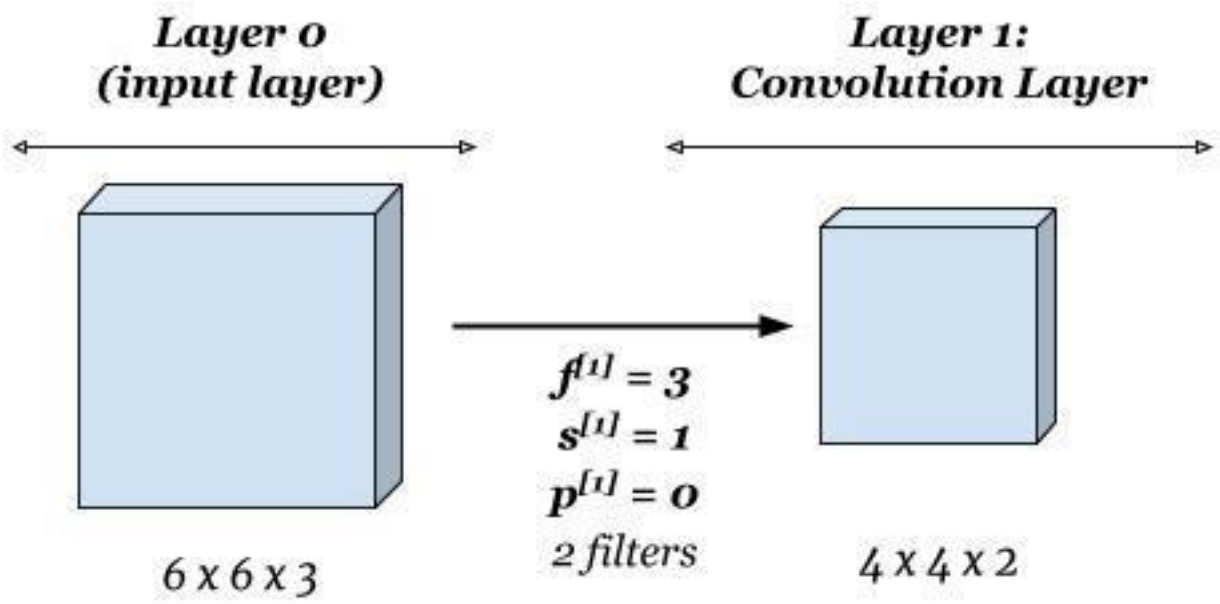
One Convolution Layer

- Finally, to make up a convolution layer, a bias (ϵ R) is added and an activation function such as **ReLU** or **tanh** is applied.



Shorthand Representation

- This simpler representation will be used from now on to represent one convolutional layer:

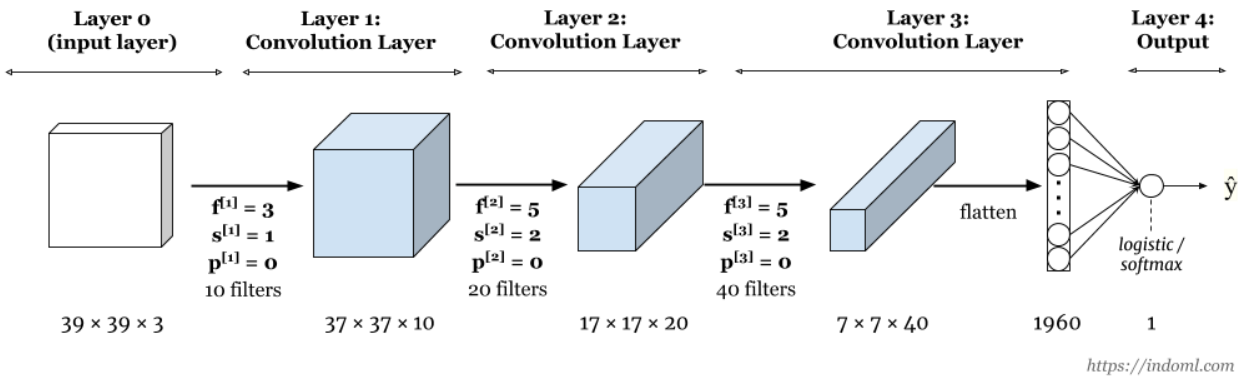


Question: Number of parameters in one layer

- If you have 10 filters of dim 3x3x3 in one layer of a CNN, how many parameters does that layer have??
- Answer: ____ parameters???
- For larger image of size says 1000x1000, still the no of parameters is only 280. So, less overfitting in case of CNN.

Sample Complete Network

- This is a sample network with three convolution layers. At the end of the network, the output of the convolution layer is flattened and is connected to a logistic regression or a softmax output layer.

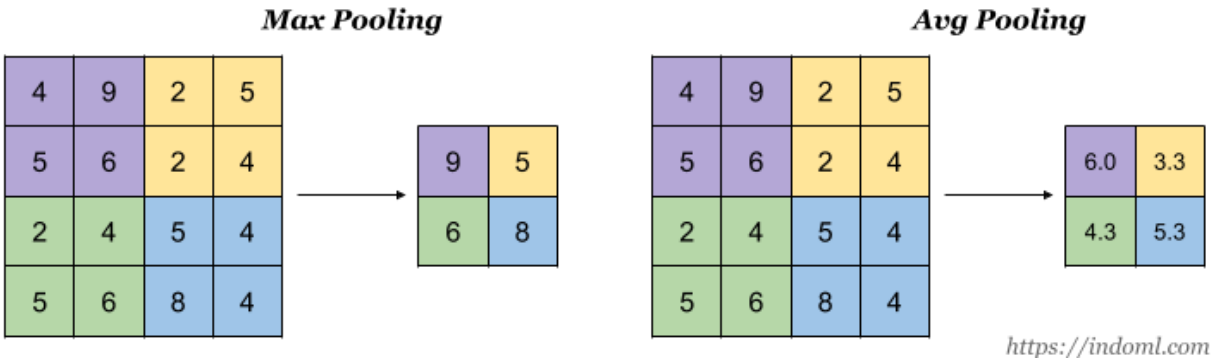


General trend for hyperparameters:

- Size of feature maps decrease with depth
- The number of channels are increased with depth
- Type of layers are CONV- POOL – FC – logistic/softmax

Pooling Layer

- Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.
- Sample types of pooling are **max pooling** and **avg pooling**, but these days max pooling is more common.



- It has hyper-parameters:
  - **size** (***f***)
  - **stride** (***s***)
  - **type** (max or avg)
  - **p = 0** (Generally)
- **but it doesn't have parameters; there's nothing for gradient descent to learn**

Why convolution is good??

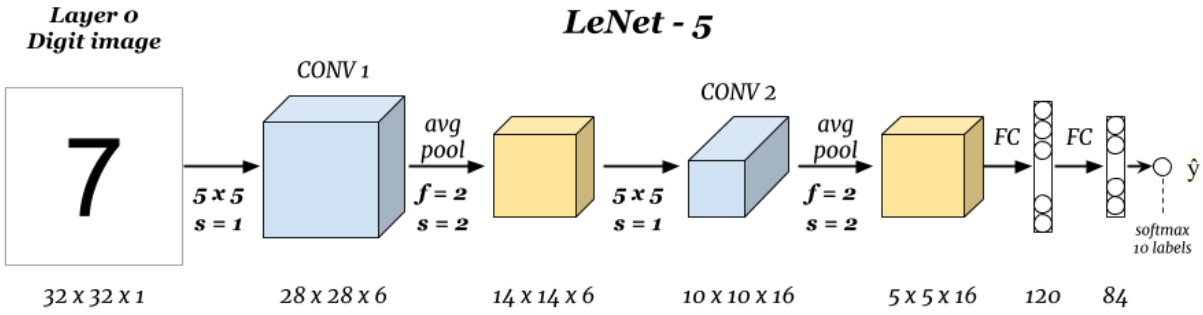
- Small number of parameters:
    - 32 x32 x 3 * 5x5x3 = > 28 x28 x6 (#filter: 6)
    - In NN 3037(32x32x3) x 4704(28x28x6) = 14 million parameters
    - But in CNN on [5x5x3+1]x6 = 456 parameters are needed
- Small number of parameters are because of:
    - Parameter sharing
    - Sparsity of connection

Well Known Architectures: Classic Network

- If a network is good for cat detection, I may work for other problems, like a car detection.
- To see the general trend for hyperparameters

LeNet – 5 (for digit recognition)

One example of classic networks is LeNet-5, from *Gradient-Based Learning Applied to Document Recognition* paper by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998):



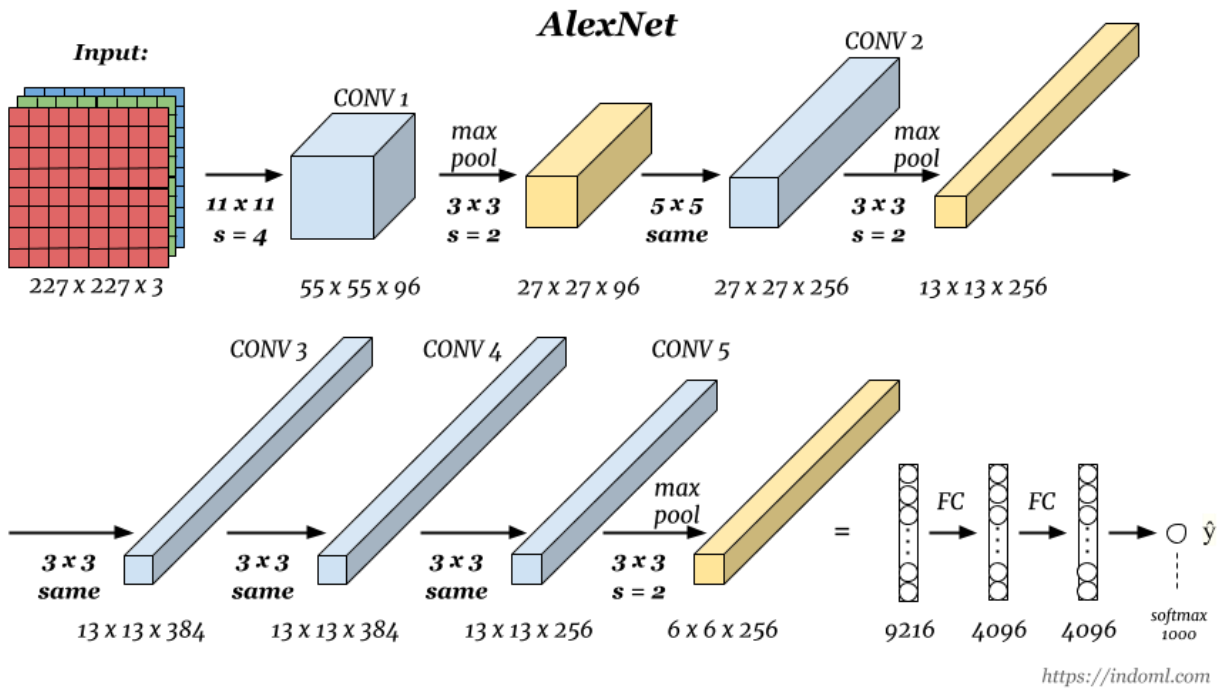
- Number of parameters: ~ 60 thousands.

General trend for hyperparameters:

- Size of feature maps are decreased with depth
- While the number of channels are increased with depth
- General pattern of layers are CONV- POOL- CONV- POOL – FC – logistic/softmax

AlexNet

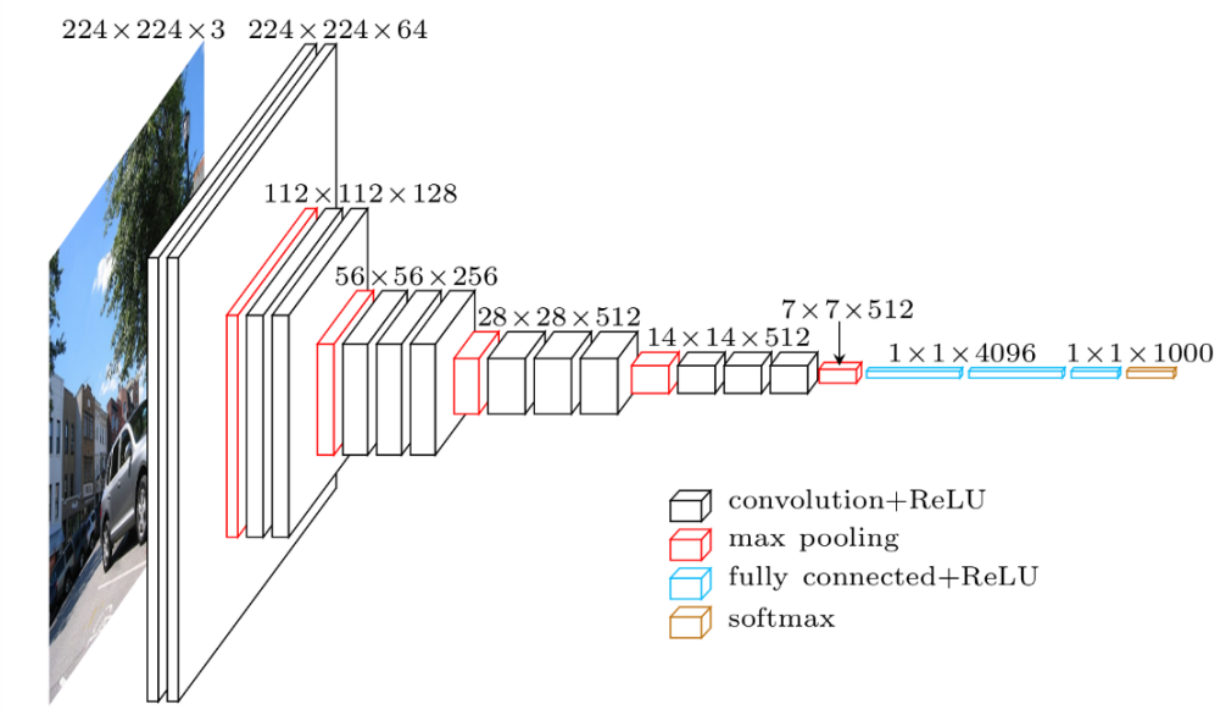
AlexNet is another classic CNN architecture from *ImageNet Classification with Deep Convolutional Neural Networks* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012).



- CONV-POOL-FC pattern
- Size of fm decrease, while no of fm increase wrt depth
- Number of parameters: ~ 60 millions.

Classic Network: VGG-16

VGG-16 from *Very Deep Convolutional Networks for Large-Scale Image Recognition* paper by Karen Simonyan and Andrew Zisserman (2014). The number 16 refers to the fact that the network has 16 trainable layers (i.e. layers that have weights).

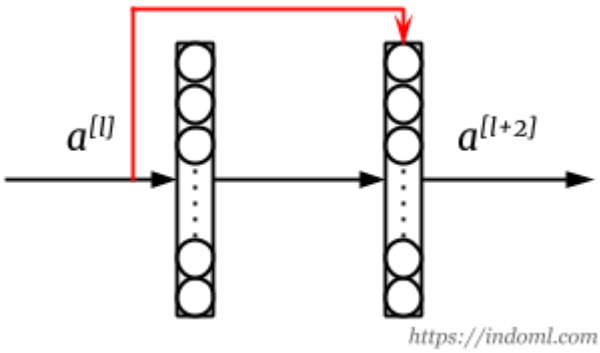


- Number of parameters: ~ 138 millions.
- The strength is in the simplicity: the dimension is halved and the depth is increased on every step (or stack of layers)

ResNet

The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, **the training error starts to raise again**. Deep networks are also harder to train due to **exploding and vanishing gradients problem**.

ResNet (Residual Network), proposed by He at all in *Deep Residual Learning for Image Recognition paper* (2015), solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network:



In the image above, the skip connection is depicted by the red line. The activation $a^{[l+2]}$ is then calculated as:

$$z^{[l+2]} = W^{[l+2]}\, a^{[l+1]} + b^{[l+2]}$$

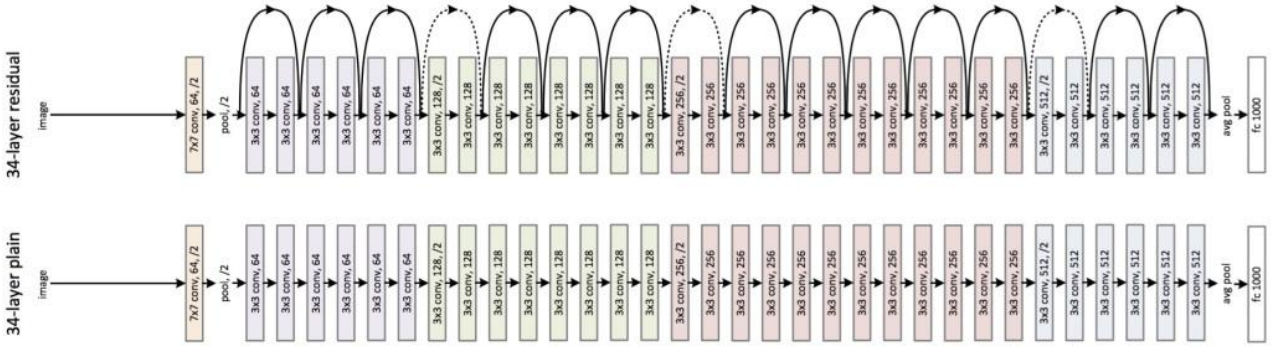
$$a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + a^{[l]})$$

The advantages of ResNets are:

- performance doesn't degrade with very deep network
- ability to train very very deep network

ResNet works because:

- Identify function is easy for residual block to learn
- Using a skip-connection helps the gradient to back-propagate and thus helps you to train deeper networks



Training loss is decrease with increase number of layers in Resnet but in plain network it increases

Keras

Using open source implementation: available at GitHub

Keras is a high-level API to build and train deep learning models.

It is high level API of tensorflow

Three key advantages:

1. User friendly
   Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.

2. Modular and composable
   Keras models are made by connecting configurable building blocks together, with few restrictions.
3. Easy to extend
   Write custom building blocks to express new ideas for research. Create new layers, loss functions, and develop state-of-the-art models.

- Import tf.keras : Give the interface to access all features of tensorflow liberay in easier way. TensorFlow easier to use without sacrificing flexibility and performance.

To run the keras programs:

- Make anaconda environment for keras or
- Use Kaggal kernel as there is no need to install anything

Pipe line to build Network:

1. Define network: To build a network
2. Compile network: To configure learning process
3. Fit network: To fit the model with training data
4. Evaluate network: To evaluate evaluation time for trained model on test data
5. Make predict: predict the output of model on test data provided

Examples code1: **https://www.tensorflow.org/guide/keras**

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

print(tf.VERSION)
print(tf.keras.__version__)


model = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model:
model.add(layers.Dense(64, activation='relu'))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add a softmax layer with 10 output units:
model.add(layers.Dense(10, activation='softmax'))



model.compile(optimizer=tf.train.AdamOptimizer(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

val_data = np.random.random((100, 32))
val_labels = np.random.random((100, 10))

model.fit(data, labels, epochs=10, batch_size=32,
          validation_data=(val_data, val_labels))

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.evaluate(data, labels, batch_size=32)

result = model.predict(data, batch_size=32)
print(result.shape)
```

Example code2:

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

….

…

```
Model=Sequential()
                    model.add(Conv2D(32, kernel_size=(3, 3),
```

```
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

Example code3: Fashion Mnist (Kaggle kernel)