

UNIT -4

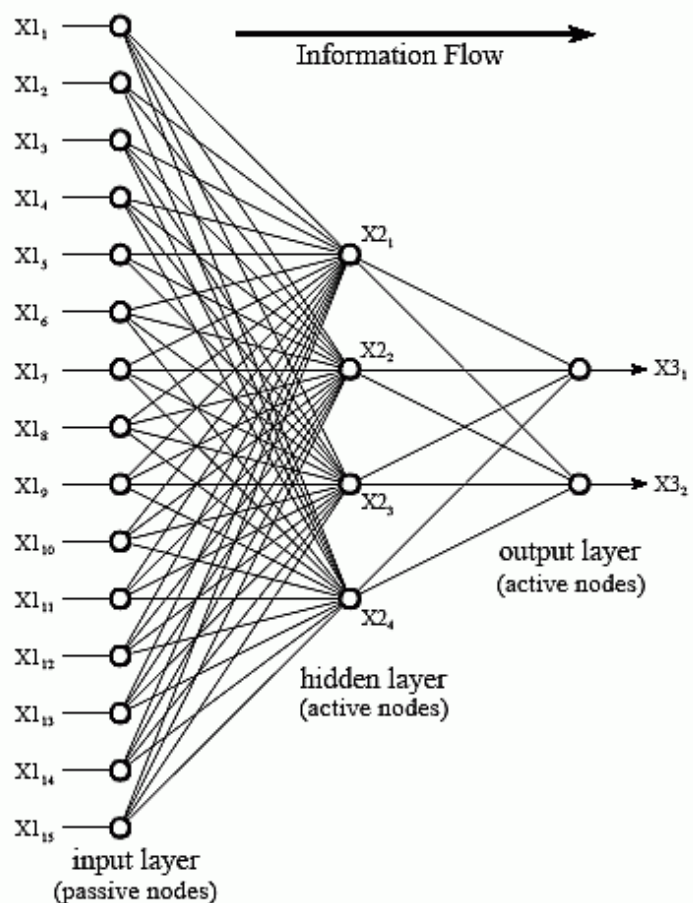
CONVOLUTION NEURAL NETWORK

Neural Networks (Representation)

Architecture of Convolutional Network :

FIGURE 26-5

Neural network architecture. This is the most common structure for neural networks: three layers with full interconnection. The input layer nodes are passive, doing nothing but relaying the values from their single input to their multiple outputs. In comparison, the nodes of the hidden and output layers are active, modifying the signals in accordance with Fig. 26-6. The action of this neural network is determined by the weights applied in the hidden and output nodes.



A Convolutional Neural Network (CNN) is a deep learning algorithm that can recognize and classify features in images for computer vision. It is a multi-layer neural network designed to analyze visual inputs and perform tasks such as image classification, segmentation and object detection, which can be useful for autonomous vehicles. CNNs can also be used for deep learning applications in healthcare, such as medical imaging.

There are two main parts to a CNN:

- A convolution tool that splits the various features of the image for analysis

- A fully connected layer that uses the output of the convolution layer to predict the best description for the image.

Basic Convolutional Neural Network Architecture

CNN architecture is inspired by the organization and functionality of the visual cortex and designed to mimic the connectivity pattern of neurons within the human brain.

The neurons within a CNN are split into a three-dimensional structure, with each set of neurons analyzing a small region or feature of the image. In other words, each group of neurons specializes in identifying one part of the image. CNNs use the predictions from the layers to produce a final output that presents a vector of probability scores to represent the likelihood that a specific feature belongs to a certain class.

How a Convolutional Neural Network Works—The CNN layers

A CNN is composed of several kinds of layers:

- **Convolutional layer**—creates a feature map to predict the class probabilities for each feature by applying a filter that scans the whole image, few pixels at a time.
- **Pooling layer (downsampling)**—scales down the amount of information the convolutional layer generated for each feature and maintains the most essential information (the process of the convolutional and pooling layers usually repeats several times).
- **Fully connected input layer**—“flattens” the outputs generated by previous layers to turn them into a single vector that can be used as an input for the next layer.
- **Fully connected layer**—applies weights over the input generated by the feature analysis to predict an accurate label.
- **Fully connected output layer**—generates the final probabilities to determine a class for the image.

MULTICHANNEL CONVOLUTION OPERATION:

Multiple Input Channels

So far in this convolution series we’ve been applying:

- 1D Convolutions to 1 dimensional data (temporal)
- 2D Convolutions to 2 dimensional data (height and width)
- 3D Convolutions to 3 dimensional data (height, width and depth)

You’ll see an obvious pattern here, but this simple correspondence hides an important detail. Our input data usually defines *multiple variables at each position* (through time, or space), and not just a single value. We call these **channels**.

Conv1D with Multiple Input Channels

As a sugar-coated example, let’s take the case of ice cream sales forecasting. Our input might be defined at daily intervals along our temporal dimension and have normalised values for the product’s price, marketing spend, outside temperature and whether it was the weekend or not. *4 channels* in total.

Input						
Date	1st	2nd	3rd	4th	5th	6th
Price	1	3	3	0	1	2
Marketing spend	0	2	1	1	2	0
Outside temperature	3	2	2	3	1	1
Weekend	0	1	1	0	0	0

Figure 1: an input array with 4 channels over 6 time steps.

Although the input data looks like it's two dimensional, only one of the dimensions is spatial. We'd only expect to find patterns in a local neighbourhood of values through time, and not across a local neighbourhood the channel variables. Starting with a random order of 4 variables (A, B, C, D), we *would not* expect to find a similar spatial relationship between A&B, B&C and C&D (if we set a kernel shape of 2 along this dimension).

So for this reason, when working with multi-channel temporal data, it's best to use a 1D Convolution; even though the data looks two dimensional.

Applying a 1D Convolution (with kernel size of 3), our kernel will look different to the single channel case shown in the last post. Given we have 4 input channels this time, our kernel will be initialised with 4 channels too. So even though we are using a 1D Convolution, we have a 2D kernel! A 1D Convolution just means we slide the kernel along one dimension, it doesn't necessarily define the shape of the kernel, since that depends on the shape of the input channels too.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1																					
2																					
3																					
4		1	3	3	0	1	2				2	0	1				24	25	22	15	
5		0	2	1	1	2	0				1	1	2								
6		3	2	2	3	1	1				3	0	1								
7		2	1	2	0	0	1				1	2	0								
8																					

Figure 2: A 1D Convolution with kernel of size 3, applied to a 4x6 input matrix to give a 1x4 output.

Figure 3: Excel formula used for Cell Q4

Advanced: a 2D Convolution with kernel shape (3,4) would be equivalent in this situation, but with a 1D Convolution you don't need to specify the channel dimension. We usually rely on shape inference for this (i.e. after passing the first batch of data), but we can manually specify the number of input channels with in_channels. When adding padding, stride and dilation to the equation, the equivalence between 2D and 1D Convolutions might not hold.

```
# define input_data and kernel as above
# input_data.shape is (4, 6)
# kernel.shape is (4, 3) conv = mx.gluon.nn.Conv1D(channels=1, kernel_size=3) # see
# appendix for definition of `apply_conv`
output_data = apply_conv(input_data, kernel, conv)
print(output_data) # [[[24. 25. 22. 15.]]]
# <NDArray 1x1x4 @cpu(0)>
```

Our code remains unchanged from the single input channel case.

Just before we wrap up with 1D Convolutions, it's worth mentioning another common use-case. One of the first stages in many Natural Language Processing models is to convert a sequence of raw text into a sequence of embeddings, either character, word or sentence embeddings. At every time step we now have an embedding with a certain number of values (e.g. 128) that each represent different attributes about the character, word or sentence. We should think about these just as with the time series example, and treat them as channels. Since we've just got channels over time, a 1D Convolution is perfect for picking up on useful local temporal patterns.

Conv2D with Multiple Input Channels

Colour images are a great example of multi-channel spatial data too. We usually have 3 *channels* to represent the colour at each position: for the intensities of *red*, *green* and *blue* colour. What's new this time though, is that we're dealing with two spatial dimensions: height and width.

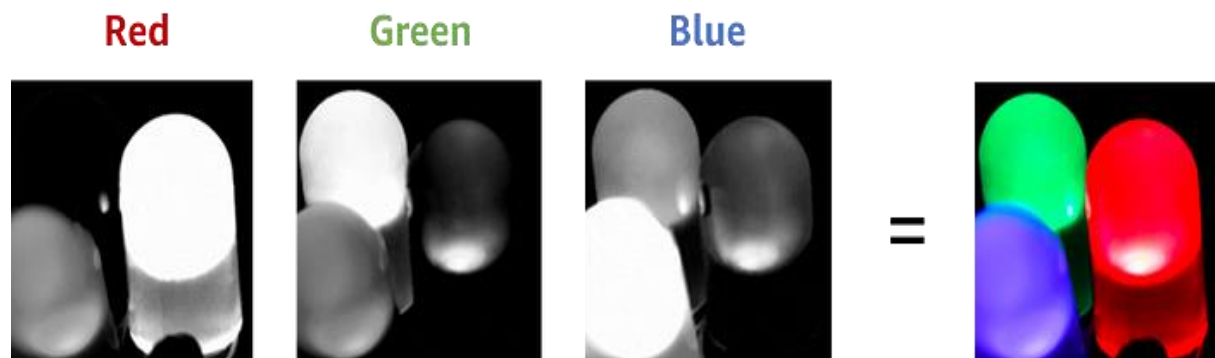


Figure 4: Color images are often represented with 3 channels ([Source](#))

Our kernel will adjust to the channels accordingly and even though we define a 3x3 kernel, the true dimensions of the kernel when initialised will be 3x3x3, since we have 3 input channels. Since we're back to 3 dimensions again, let's take a look at a [three.js](#) diagram first before blowing our minds with MS Excel.

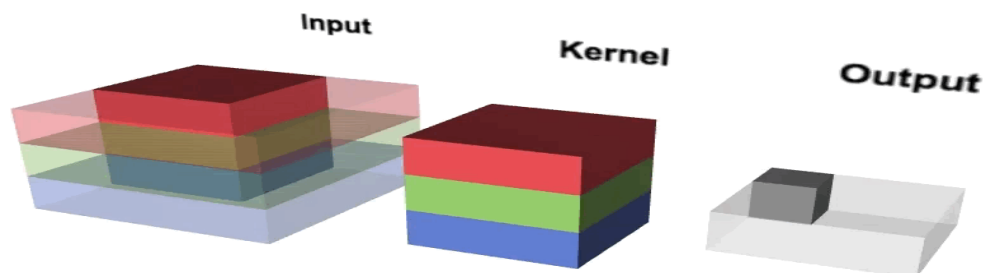


Figure 6: A 2D Convolution with a 3x3 kernel applied to an 3 channel RGB input of size 5x5 to give output of 3x3. ([source](#))

With the diagram above you should think about viewing from above if you wanted to view the actual image, but we're interested in seeing the channels, hence why we're looking from a slide angle. Check out [this link](#) for an interactive version of the diagram above. One interesting observation is that each 'layer' of the kernel interacts with a corresponding channel of the input.

We can actually see this in more detail when looking at MS Excel.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1																									
2		<u>Input</u>								<u>Kernel</u>						<u>Intermediate Output</u>									
3																									
4		1	0	1	0	2																			
5		1	1	3	2	1				0	1	0				7	5	3							
6		1	1	0	1	1				0	0	2				4	7	5							
7		2	3	2	1	3				0	1	0				7	2	8							
8		0	2	0	1	0																			
9																									
10		1	0	0	1	0																			
11		2	0	1	2	0				2	1	0				5	3	10					19	13	15
12		3	1	1	3	0				0	0	0				13	1	13					28	16	20
13		0	3	0	3	2				0	3	0				7	12	11					23	18	25
14		1	0	3	2	1																			
15																									
16		2	0	1	2	1																			
17		3	3	1	3	2				1	0	0				7	5	2							
18		2	1	1	1	0				1	0	0				11	8	2							
19		3	1	3	2	0				0	0	2				9	4	6							
20		1	1	2	1	1																			
21																									

Figure 5: A 2D Convolution with a 3x3 kernel applied to an 3 channel RGB input of size 5x5 to give output of 3x3.

Viewed like this, we think as if each channel has its own 3x3 kernel. We apply each "layer" of the kernel to the corresponding input channel and obtain intermediate values, a single value for each channel. Our final step is sum up these values, to obtain our final result for the output. So ignoring 0s we get:

$$\text{red_out} = (3*2)+(1*1) = 7$$

$$\text{green_out} = (1*2)+(1*3) = 5$$

$$\text{blue_out} = (2*1)+(3*1)+(1*2) = 7$$

$$\text{output} = \text{red_out} + \text{green_out} + \text{blue_out} = 7+5+7 = 19$$

We don't actually calculate these intermediate results in practice but it shows the initial separation between channels. A kernel still looks at patterns across channels though, since we have the cross channel summation at the end.

Advanced: a 3D Convolution with kernel shape (3,3,3) would be equivalent in this situation, but with a 2D Convolution you don't need to specify the channel dimension. Again, we usually rely on shape inference for this (i.e. after passing the first batch of data), but we can manually specify the number of input channels with `in_channels`. When adding padding, stride and dilation to the equation, the equivalence between 3D and 2D Convolutions might not hold.

```
# define input_data and kernel as above
# input_data.shape is (3, 5, 5)
# kernel.shape is (3, 3, 3) conv = mx.gluon.nn.Conv2D(channels=1,
kernel_size=(3,3)) output_data = apply_conv(input_data, kernel, conv)
print(output_data) # [[[[[19. 13. 15.]
# [28. 16. 20.]
# [23. 18. 25.]]]]
# <NDArray 1x1x3x3 @cpu(0)>
```

Code in MXNet Gluon looks the same as with a single channel input, but notice that the shape of the kernel is (3,3,3) because we have a kernel applied to an input with 3 channels and it has a height of 3 and a width of 3. Our layout for the kernel is therefore (in_channels, height, width).

Multiple Output Convolutions

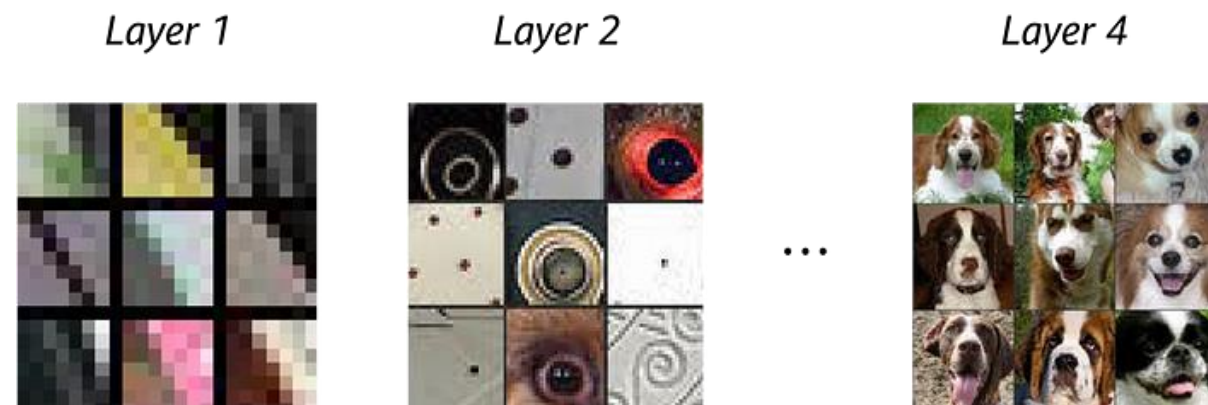


Figure 1: ImageNet patches that maximise the output of selected kernels from convolutional layers. ([Source](#))

All the way back in the [first blog post of the series](#) we introduced the convolution as a feature detector, with the kernel defining the type of feature we wanted to detect. So far, in all of our examples, we've been using a single kernel as part of our convolution, meaning that we've just been looking for a single feature. Clearly for tasks as complex as image classification, object detection and segmentation we need to be looking at more than one feature at each layer of the network.

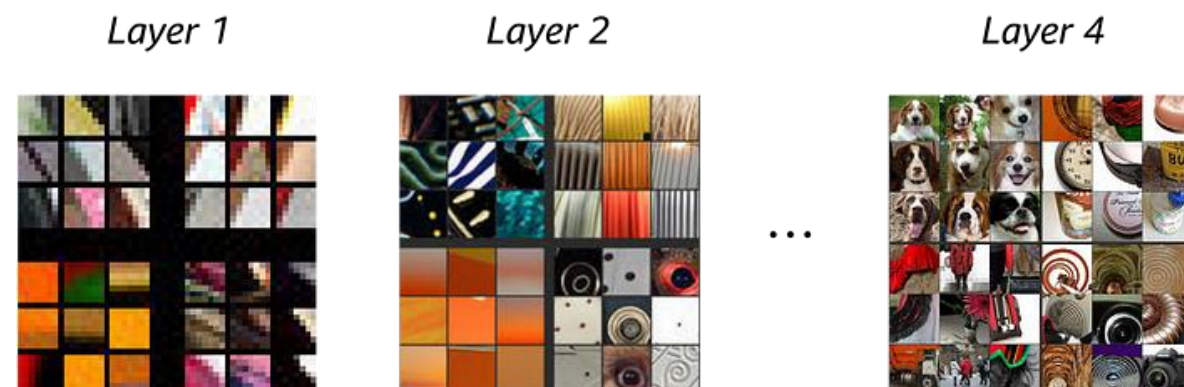


Figure 2: ImageNet patches that maximise the output of 4 selected kernels from convolutional layers. ([Source](#))

Starting with the first convolutional layers, it might be useful to detect edges at different angles and different colour contrasts (an example of multiple input channel kernels that we've just learnt about). And then in later layers, it might be useful to detect spirals, as well as dog faces (always useful!).

Computationally, it's very easy to add multiple output channels. We just repeat the whole process from before for as many output channels as we require, each time with a different and independent kernel, and just stack all of the outputs. Voilà!

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1																					
2																					
3																					
4																					
5																					
6																					
7																					
8																					
9																					
10																					
11																					

Figure 7: A 1D Convolution with 4 kernels of size 3 applied to a 1x6 input matrix to give a 4x4 output.

Compare this with the Figure 2 from the [last blog post](#). Our first kernel is the same as in that example and we get the same output (of shape 1x4), but this time we add 3 more kernels and get an final output of shape 4x4.

As usual, this is simple to add to our convolutions in MXNet Gluon. All we need to change is the channels parameter and set this to 4 instead of 1.

```
conv = mx.gluon.nn.Conv1D(channels=4, kernel_size=3)
```

Advanced: We previously mentioned the similarly named parameter called `in_channels`. It's important to note the difference. `in_channels` is used if you want to specify the number of channels expected in the input data, instead of using shape inference (by passing the first batch of data). `channels` is used to specify the number of output channels required, i.e. the number of kernels/filters.

```
# define input_data and kernel as above
```

```
# input_data.shape is (1, 6)
```

```
# kernel.shape is (4, 1, 3)output_data = apply_conv(input_data, kernel, conv)
```

```
print(output_data)# [[[ 5.  6.  7.  2.]
```

```
# [ 6.  6.  0.  2.]
```

```
# [ 9. 12. 10.  3.]
```

```
# [10.  6.  5.  5.]]]
```

```
# <NDArray 1x4x4 @cpu(0)>
```

Once upon a time, back in [the last post](#), our 1D Convolution's kernel was also 1D. When we had multiple input channels, we had to add an extra dimension to the handle this. And now we're looking at multiple output channels, we've got to add another! Which takes us up to 3D

in total for a 1D Convolution. We've had these dimensions all along (apologies for hiding the truth from you again!), but it was simpler to ignore them since they were of unit length and just let `apply_conv` add them for us.

So for the case above we can see our kernel is of shape (4,1,3). We have 4 kernels, each of which is applied to the same input data with 1 channel, and they all have a width of 3 in the temporal dimension. Our layout for the kernel is therefore (channels, in_channels, width). Check the following table for a more complete list of default dimension layouts.

Convolution	Layout	
1D	Input	(batch_size, in_channels, width)
	Kernel	(channels, in_channels, width)
	Output	(batch_size, channels, width)
2D	Input	(batch_size, in_channels, height, width)
	Kernel	(channels, in_channels, height, width)
	Output	(batch_size, channels, height, width)
3D	Input	(batch_size, in_channels, depth, height, width)
	Kernel	(channels, in_channels, depth, height, width)
	Output	(batch_size, channels, depth, height, width)

Table 1: MXNet Gluon's default dimension layouts.

RECURRENT NEURAL NETWORKS

INTRODUCTION TO RNN:

A recurrent neural network (RNN) is a type of artificial neural network commonly used in speech recognition and natural language processing (NLP). RNNs are designed to recognize a data's sequential characteristics and use patterns to predict the next likely scenario.

RNNs are used in deep learning and in the development of models that simulate the activity of neurons in the human brain. They are especially powerful in use cases in which context is critical to predicting an outcome and are distinct from other types of artificial neural networks because they use feedback loops to process a sequence of data that informs the final output, which can also be a sequence of data. These feedback loops allow information to persist; the effect is often described as memory.

RNN use cases tend to be connected to language models in which knowing the next letter in a word or the next word in a sentence is predicated on the data that comes before it. A compelling experiment involves an RNN trained with the works of Shakespeare to produce Shakespeare-like prose -- successfully. Writing by RNNs is a form of computational creativity. This simulation of human creativity is made possible by the AI's understanding of grammar and semantics learned from its training set.

RNN CODE

Time Series in RNN

In this tutorial, we will use an RNN with time-series data. Time series is dependent on the previous time, which means past values include significant information that the network can learn. The time series prediction is to estimate the future value of any series, let's say, stock price, temperature, GDP, and many more.

The data preparation for RNN and time-series make a little bit tricky. The objective is to predict the other value of the series, and we will use the past information to estimate the cost at $t + 1$. The label is equal to the input succession one period along.

Secondly, the number of inputs is set to 1, i.e., one observation per time. In the end, the time step is equal to the sequence of the numerical value. If we set the time step to 10, the input sequence will return ten consecutive times.

Look at the graph below, and we have to represent the time series data on the left and a fictive input sequence on the right. We create a function to return a dataset with a random value for each day from January 2001 to December 2016

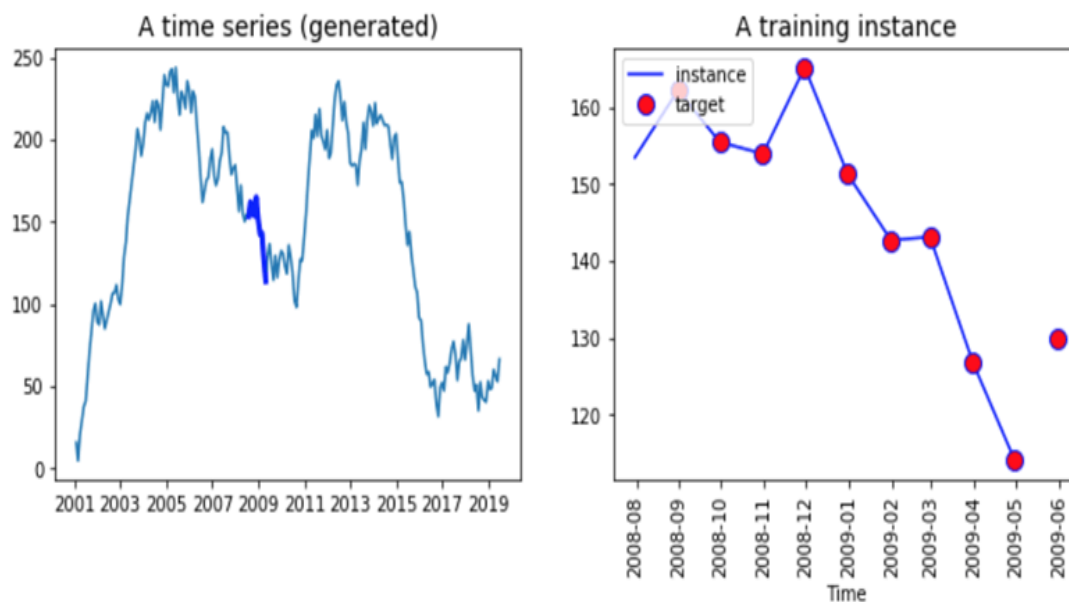
```
1. # To plotting amazing figure
2. %matplotlib inline
3. import matplotlib
4. import pandas as pd
5. import matplotlib.pyplot as plt
6. def create_ts(start = '2001', n = 201, freq = 'M'):
7.     ring = pd.date_range(start=start, periods=n, freq=freq)
8.     ts = pd.Series(np.random.uniform(-18, 18, size=len(rng)), ring).cumsum()
9.     return ts
10. ts= create_ts(start = '2001', n = 192, freq = 'M')
11. ts.tail(5)
```

Output:

```
2016-08-31   -93.459631
2016-09-30   -95.264791
2016-10-31   -95.551935
2016-11-30  -105.879611
2016-12-31  -123.729319
Freq: M, dtype: float64
```

```
ts = create_ts(start = '2001', n = 222)
```

```
1. # Left plotting diagram
2. plt.figure(figsize=(11,4))
3. plt.subplot(121)
4. plt.plot(ts.index, ts)
5. plt.plot(ts.index[90:100], ts[90:100], "b-
   ",linewidth=3, label="A train illustration in the plotting area")
6. plt.title("A time series (generated)", fontsize=14)
7.
8. ## Right side plotted Diagram
9. plt.subplot(122)
10. plt.title("A training instance", fontsize=14)
11. plt.plot(ts.index[90:100], ts[90:100], "b-", markersize=8, label="instance")
12. plt.plot(ts.index[91:101], ts[91:101], "bo", markersize=10, label="target", markerface
   color='red')
13. plt.legend(loc="upper left")
14. plt.xlabel("Time")
15. plt.show()
```



The right part of the graph shows all the series. It starts in 2001 and finishes in 2019. There is no sense to makes no sense to feed all the data in the network; instead, we have to create a batch of data with a length equal to the time step. This batch will be the X variable. The Y variable is the same as the X but shifted by one period (i.e., we want to forecast $t+1$).

Both vectors have the same length. We can see this in the right part of the graph above. The line represents ten values of the x input, while the red dots label has ten values, y. Note that, the label starts one period forward of X and ends after one period.

Build an RNN to analyze Time Series in TensorFlow

It is time to build our first **RNN** to predict the series. We have to specify some hyperparameters (the parameters of the model, i.e., number of neurons, etc.) for the model.

- Number of input: 1
- Time step (windows in time series): 10
- Number of neurons: 120
- Number of output: 1

Our network will learn from a sequence of **10** days and contain **120** recurrent neurons. We feed the model with one input.

Before constructing the model, we need to split the dataset into the train set and test set. The full dataset has **222** data points; We will use the first 201 points to train the model and the last 21 points to test our model.

After we define a train and test set, we need to create an object containing the batches. In these batches, we have X values and Y values. Remember that the X value is one period straggle. Therefore, We use the first 200 observations, and the time step is equal to 10. The x_batches object must have 20 batches of size 10 or 1. The size of the Y_batches is the same as the X_batches object, but with a period above.

Step 1) Create the train and test

Firstly, we convert the series into a numpy array; then, we define the windows (the number of time networks will learn from), the number of input, output, and the size of the train set.

1. `series = np.array(ts)`
2. `n_windows = 20`
3. `n_input = 1`

4. `n_output = 1`
5. `size_train = 201`

After that, we split the array into two datasets.

1. `# Split data`
2. `train = series[:size_train]`
3. `test = series[size_train:]`
4. `print(train.shape, test.shape)`
5. `(201) (21)`

Step 2) Create the function return **X_batches** and **y_batches**

We can create a function that returns two different arrays, one for **X_batches** and one for **y_batches**. To make it easier.

Let's make a function to construct the batches.

Note that, the **X_batches** are logged by one period (we take value $t-1$). The output of the function has three dimensions. The first dimensions are equal to the number of batches, the second is the size of the windows, and the last one is the number of input.

The tricky part of the time series is to select the data points correctly. For the **X** data points, we choose the observations from **$t = 1$ to $t = 200$** , while for the **Y** data point, we return the observations from **$t = 2$ to 201** . Once we have the correct data points, it is effortless to reshape the series.

To construct the object with the batches, we need to split the dataset into ten batches of the same length. We can use the reshape method and pass -1 so that the series is the same as the batch size. The value 20 is the number of comments per batch, and 1 is the number of inputs. We need to do the same step for the label.

Note that we need to shift the data to the number of times we want to forecast. For instance, if we want to predict one time, then we shift the series by 1. If we want to forecast two days, then shift the data by 2 points.

1. `x_data = train[:size_train-1]`: Select the training instance.
2. `X_batches = x_data.reshape(-1, Windows, input)`: creating the right shape **for** the batch.
3. `def create_batches(df, Windows, input, output):`
4. `## Create X`
5. `x_data = train[:size_train-1]` # Select the data
6. `X_batches = x_data.reshape(-1, windows, input)` # Reshaping the data in **this** line of code
7. `## Create y`
8. `y_data = train[n_output:size_train]`
9. `y_batches = y_data.reshape(-1, Windows, output)`
10. `return X_batches, y_batches` **#return** the function

Now the function is defined, we call it for creating the batches.

1. `Windows = n_`
2. `Windows, # Creating windows`
3. `input = n_input,`
4. `output = n_output)`

We can print the shape to make sure the dimensions are correct.

1. `print(X_batches.shape, y_batches.shape)`
2. `(10, 20, 1) (10, 20, 1)`

We need to create the test set with only one batch of data and 20 observations.

Note that our forecast days after days, it means the second predicted value will be based on the actual value of the first day ($t+1$) of the test dataset. The true value will be known. If you want to forecast $t+2$, we need to use the predicted value $t+1$; if you're going to predict $t+3$, we need to use the expected value $t+1$ and $t+2$. It makes it difficult to predict precisely " $t+n$ " days.

1. `X_test, y_test = create_batches(df = test, windows = 20, input = 1, output = 1)`
2. `print(X_test.shape, y_test.shape)`
3. `(10, 20, 1) (10, 20, 1)`

Our batch size is ready, we can build the RNN architecture. Remember, we have 120 recurrent neurons.

Step 3) Build the model

To create the model, we need to define three parts:

1. The variable with the tensors
2. The RNN
3. The loss and optimization

1. Variables

We need to specify the X and y variables with an appropriate shape. This step is trivial. The tensors are the same dimension as the objects **X_batches** and the object **y_batches**.

For instance, the tensors X is a placeholder has almost three dimensions:

- Note: size of the batch
- `n_windows`: Length of the windows.
- `n_input`: Number of input

The result is:

1. `tf.placeholder(tf.float32, [None, n_windows, n_input])`
2. `## 1. Construct the tensors`
3. `X = tf.placeholder(tf.float32, [None, n_windows, n_input])`
4. `y = tf.placeholder(tf.float32, [None, n_windows, n_output])`

2. Create the RNN

In the second part, we need to define the architecture of the network. As before, we use the object `BasicRNNCell` and the `dynamic_rnn` from TensorFlow estimator.

1. `## 2. create the model`
2. `basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron, activation=tf.nn.relu)`
3. `rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)`

The next part is trickier but allows faster computation. We need to transform the run output to a dense layer and then convert it to has the same dimension like the input field.

1. `stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])`
2. `stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)`
3. `outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])`

3. Create the loss and optimization

The model optimization depends on the task which we are performing.

This difference is important because it can change the optimization problem. The optimization problem for a continuous variable use to minimize the mean square error. To construct these metrics in **TF**, we can use:

1. `tf.reduce_sum(tf.square(outputs - y))`

The enduring code is the same as before; we use an Adam optimizer to reduce the loss.

1. `tf.train.AdamOptimizer(learning_rate=learning_rate)`
2. `optimizer.minimize(loss)`

We can pack everything together, and our model is ready to train.

```
1. tf.reset_default_graph()
2. r_neuron = 120
3.
4. ## 1. Constructing the tensors
5. X = tf.placeholder(tf.float32, [None, n_windows, n_input])
6. y = tf.placeholder(tf.float32, [None, n_windows, n_output])

1. ## 2. creating our models
2. basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron, activation=tf.nn.relu)

3. rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)
4.
5. stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
6. stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
7. outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
8.
9. ## 3. Loss optimization of RNN
10. learning_rate = 0.001
11.
12. loss = tf.reduce_sum(tf.square(outputs - y))
13. optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
14. training_op = optimizer.minimize(loss)
15.
16. init = tf.global_variables_initializer()
```

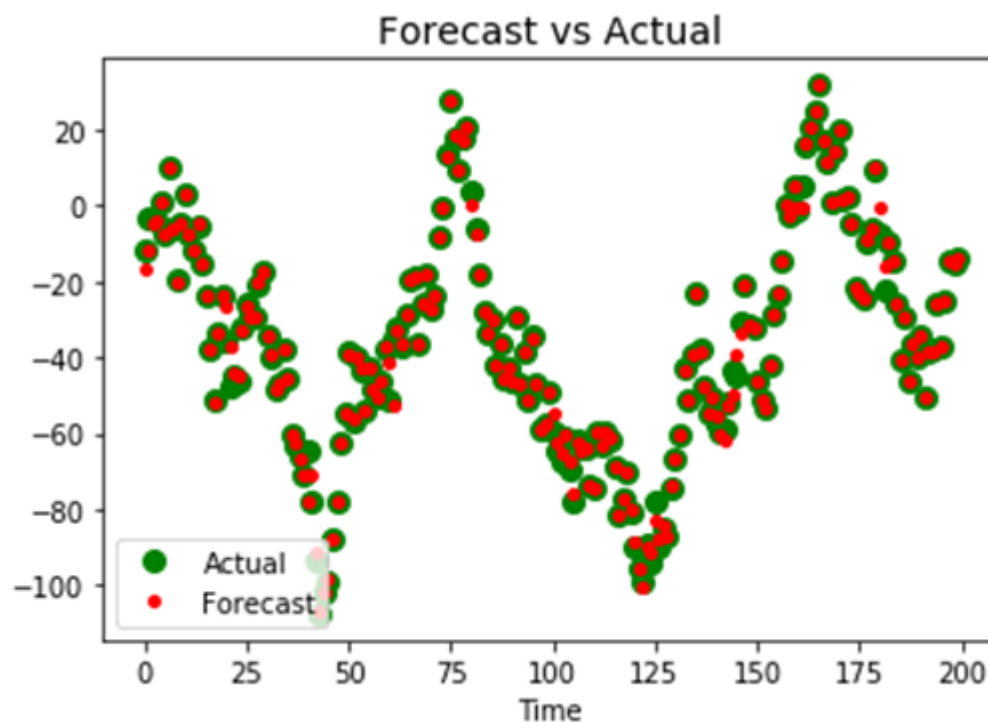
We will train the model using **1500** epochs and print the loss every 150 iterations. Once the model is trained, we evaluate the model on the test set and create an object containing the prediction.

```
1. iteration = 1500
2. with tf.Session() as sess:
3.     init.run()
4.     for iters in range(iteration):
5.         sess.run(training_op, feed_dict={X: X_batches, y: y_batches})
6.         if iters % 150 == 0:
7.             mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
8.             print(iters, "\tMSE:", mse)
9.     y_pred = sess.run(outputs, feed_dict={X: X_test})
10. "0 MSE: 502893.34
11. 150 MSE: 13839.129
12. 300 MSE: 3964.835
13. 450 MSE: 2619.885
14. 600 MSE: 2418.772
15. 750 MSE: 2110.5923
16. 900 MSE: 1887.9644
17. 1050 MSE: 1747.1377
18. 1200 MSE: 1556.3398
19. 1350 MSE: 1384.6113"
```

At last, we can plot the actual value of the series with the predicted value. If our model is corrected, the predicted values should be put on top of the actual values.

As we can see, the model has room of improvement. It is up to us to change the hyper parameters like the windows, the batch size of the number of recurrent neurons in the current files.

1. `plt.title("Forecast vs Actual", fontsize=14)`
2. `plt.plot(pd.Series(np.ravel(y_test)), "bo", markersize=8, label="actual", color='green')`
3. `plt.plot(pd.Series(np.ravel(y_pred)), "r.", markersize=8, label="forecast", color='red')`
4. `plt.legend(loc="lower left")`
5. `plt.xlabel("Time")`
6. `plt.show()`



A recurrent neural network is an architecture to work with time series and text analysis. The output of the previous state is used to conserve the memory of the system over time or sequence of words.

PyTorch Tensors

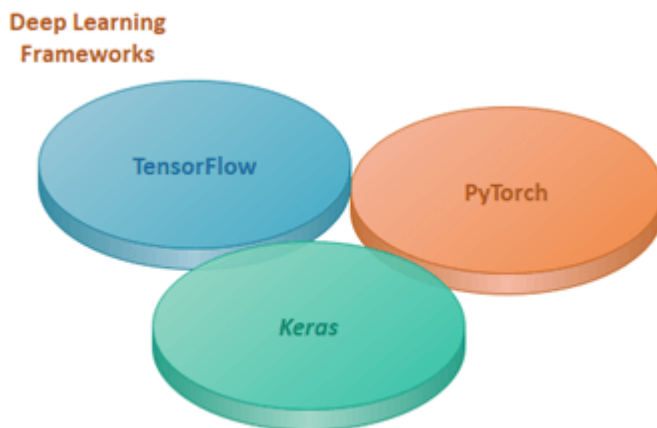
DEEP LEARNING WITH PyTorch

PyTorch is a small part of a computer software which is based on **Torch** library. It is a Deep Learning framework introduced by **Facebook**. PyTorch is a **Machine Learning Library** for **Python** programming language which is used for applications such as **Natural Language Processing**.

The high-level features which are provided by PyTorch are as follows:

1. With the help of the **Graphics Processing Unit** (GPU), it gives tensor computing with strong acceleration.
2. It provides **Deep Neural Network** which is built on a tape-based auto diff system.

PyTorch was developed to provide high flexibility and speed during implementing and building the **Deep Learning Neural Network**. As you already know, it is a machine learning library for **Python** programming language, so it's quite simple to install, run, and understand. Pytorch is **completely pythonic** (using widely adopted python idioms rather than writing Java and C++ code) so that it can quickly build a **Neural Network Model** successfully.



History of PyTorch

PyTorch was released in 2016. Many researchers are willing to adopt PyTorch increasingly. It was operated by **Facebook**. Facebook also operates **Caffe2** (Convolutional Architecture for Fast Feature Embedding). It is challenging to transform a PyTorch-defined model into Caffe2. For this purpose, Facebook and Microsoft invented an **Open Neural Network Exchange** (ONNX) in September 2017. In simple words, ONNX was developed for converting models between frameworks. Caffe2 was merged in March 2018 into PyTorch.

PyTorch makes ease in building an extremely complex neural network. This feature has quickly made it a go-to library. In research work, it gives a tough competition to TensorFlow. Inventors of PyTorch wants to make a highly imperative library which can easily run all the numerical computation, and finally, they invented PyTorch. There was a big challenge for Deep learning scientist, Machine learning developer, and Neural Network debuggers to run and test part of the code in real-time. PyTorch completes this challenge and allows them to run and test their code in real-time. So they don't have to wait to check whether it works or not.

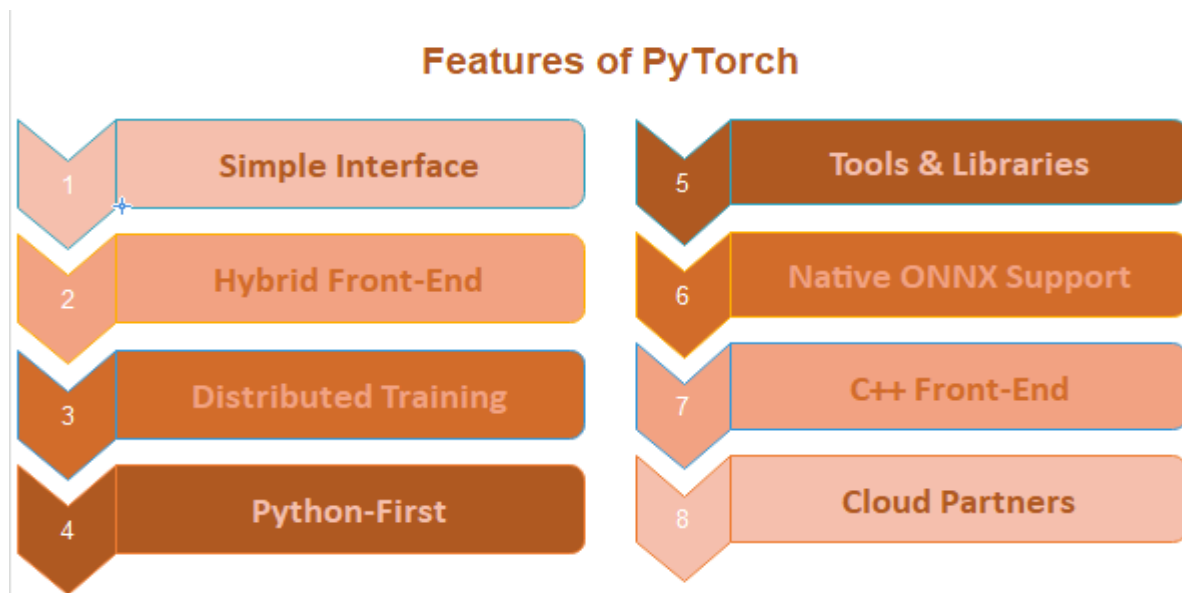
Note: To use the PyTorch functionality and services, you can use Python packages such as NumPy, SciPy, and Cython.

Why use PyTorch?

Why PyTorch? What is special in PyTorch which makes it special to build Deep learning model. PyTorch is a dynamic library. Dynamic library means a flexible library, and you can use that library as per your requirements and changes. At present in Kaggle competition, it is continuously used by finishers.

There are so many features which makes deep learning scientist to use it in making Deep learning model.

These features are as follows.



Simple interface

PyTorch has a very simple interface like Python. It provides an easy way to use API. This framework is very easy to run and operate like Python. PyTorch can easily understand or implement on both Windows and Linux.

Hybrid Front-End

PyTorch provides a new hybrid front-end which provides flexibility and ease of use in eager mode, while originally transition to graph mode for speed, optimization, and functionality in C++ runtime environment.

For example:

1. `@torch.jit.script`
2. `def Rnn(h, x, Wh, Uh, Wy, bh, by):`
3. `y = []`
4. `for t in range(x.size(0)):`
5. `h = torch.tanh(x[t] @ Wh + h @ Uh + bh)`
6. `y += [torch.tanh(h @ Wy + by)]`
7. `if t % 10 == 0:`
8. `print("stats: ", h.mean(), h.var())`
9. `return torch.stack(y), h`

Distributed Training

PyTorch allows developers to train a neural network model in a distributed manner. It provides optimized performance in both research and production with the help of native support for peer to peer communication and asynchronous execution of collective operation from Python and C++.

For example:

1. `import torch.distributed as dist1`
2. `from torch.nn.parallel import DistributedDataParallel`
3. `dist1.init_process_group(backend='gloo')`
4. `model = DistributedDataParallel(model)`

Python-First

PyTorch is completely based on Python. PyTorch is used with most popular libraries and packages of Python such as Cython and Numba. PyTorch is built deeply into Python. Its code is completely pythonic. **Pythonic** means using widely adopted Python idioms rather than writing java and C++ code in your code.

For example:

1. `import torch`
2. `import numpy as np`
3. `x = np.ones(5)`
4. `y = torch.from_numpy(x)`
5. `np.add(x, 1, out=x)`
6. `print(x)`
7. `print(y)`

Tools and Libraries

A rich ecosystem of tools and libraries are available for extending PyTorch and supporting development in areas from computer vision and reinforcement learning. This ecosystem was developed by an active community of developers and researchers. These ecosystems help them to build flexible and fast access Deep learning neural network.

For example:

1. **import** torchvision.models as models
2. resnet18 = models.resnet18(pretrained=True)
3. alexnet = models.alexnet(pretrained=True)
4. squeezenet = models.squeezenet1_0(pretrained=True)
5. vgg16 = models.vgg16(pretrained=True)
6. densenet = models.densenet161(pretrained=True)
7. inception = models.inception_v3(pretrained=True)

Native ONNX Support

ONNX was developed for converting models between frameworks. For direct access to ONNX-compatible platforms, runtimes, visualizers, and more, you need to export models in the standard ONNX.

For example:

1. **import** torch.onnx
2. **import** torchvision
3. dum_input = torch.randn(1, 3, 224, 224)
4. model = torchvision.models.alexnet(pretrained=True)
5. torch.onnx.export(model, dum_input, "alexnet.onnx")

C++ Front-End

The C++ front-end is a c++ interface for PyTorch which follows the design and architecture of the established Python frontend. It enable research in high performance, low latency and bare metal C++ application.

For example:

1. #include <torch/torch.h>
2. torch::nn::Linear model(num_features, 1);


```

3. torch::optim::SGD optimizer(model->parameters());
4. auto data_loader = torch::data::data_loader(dataset);
5. for (size_t epoch = 0; epoch < 10; ++epoch)
6. {
7.     for (auto batch : data_loader)
8.     {
9.         auto prediction = model->forward(batch.data);
10.        auto loss = loss_function(prediction, batch.target);
11.        loss.backward();
12.        optimizer.step();
13.    }
14.}

```

Cloud Partners

PyTorch is supported by many major cloud platforms such as AWS. With the help of prebuilt images, large scale training on GPU's and ability to run models in a production scale environment etc.; it provides frictionless development and easy scaling.

For example:

```

1. export IMAGE_FAMILY="pytorch-latest-cpu"
2. export ZONE="us-west1-b"
3. export INSTANCE_NAME="my-instance"
4. gcloud compute instances create $INSTANCE_NAME \
5. --zone=$ZONE \
6. --image-family=$IMAGE_FAMILY \

```

Installation of PyTorch

For installation, first, you have to choose your preference and then run the install command. You can start installation locally or with a cloud partner. In the below diagram, Stable shows the most currently supported and tested version of PyTorch (1.1), which is suitable for many users. If you want the latest 1.1 builds but not fully tested and supported, then you have to choose Preview (Nightly).

For installation, it's necessary that you have met the prerequisites which are suited to your package manager. We recommend you to use **Anaconda** package manager because it installs all the dependencies.

Platforms, Os, Language and other prerequisites					
PyTorch Build	Suitable (1.1)		Preview		
Os	Linux		Mac		Windows
package	Conda	Pip	LibTorch	Source	
language	Python 2.7	Python 3.5	Python 3.6	Python 3.7	C++
Cuda	9.0	10.0		None	

LibTorch is available only for C++. Now, we first install PyTorch in windows with the pip package, and after that we use Conda.

Installation on Windows using Pip

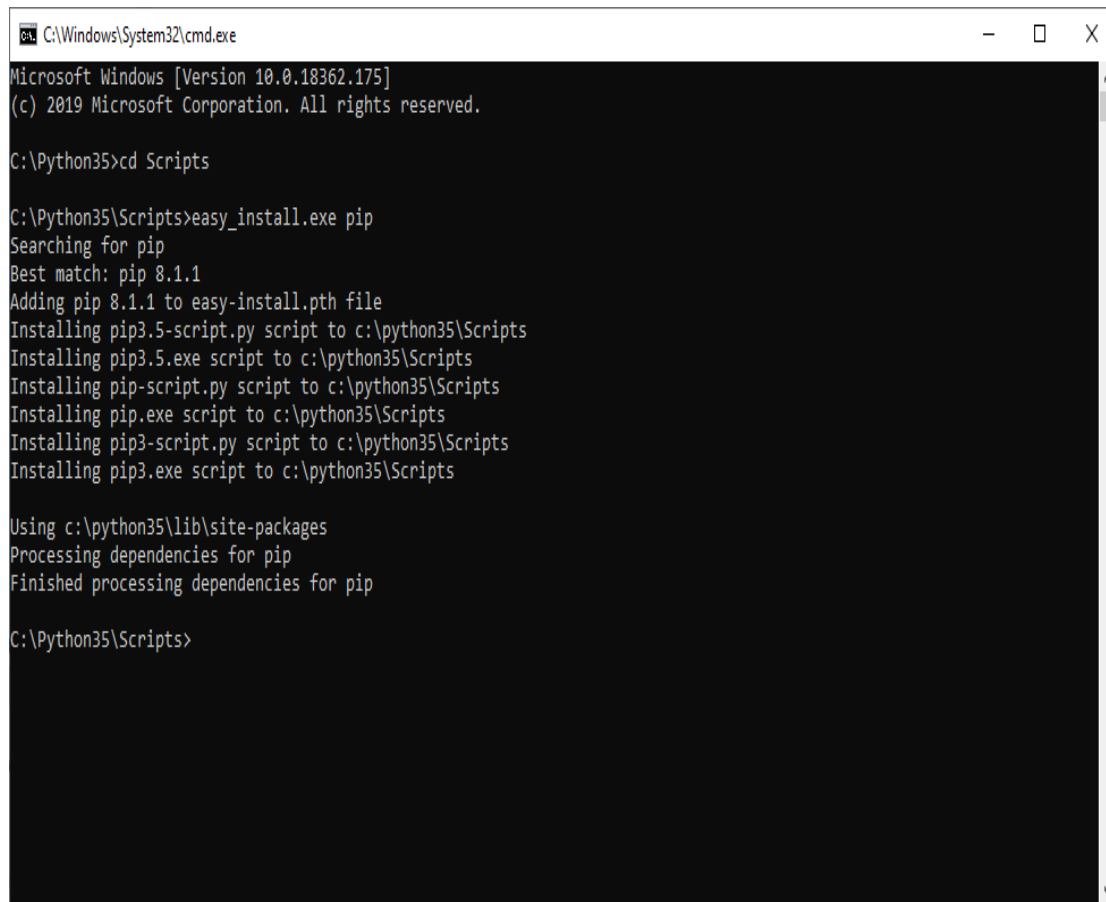
To install PyTorch, you have to install python first, and then you have to follow the following steps.

Step 1:

At very first you have to enter on the python37 folder and then in its Scripts folder using cd Scripts command.

Step 2:

In the second step, you have to install pip as per your required version with the help of **easy_install.exe** pip command on your command prompt. Once processing of dependencies is finished, you will back to the Scripts folder automatically.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Python35>cd Scripts

C:\Python35\Scripts>easy_install.exe pip
Searching for pip
Best match: pip 8.1.1
Adding pip 8.1.1 to easy-install.pth file
Installing pip3.5-script.py script to c:\python35\Scripts
Installing pip3.5.exe script to c:\python35\Scripts
Installing pip-script.py script to c:\python35\Scripts
Installing pip.exe script to c:\python35\Scripts
Installing pip3-script.py script to c:\python35\Scripts
Installing pip3.exe script to c:\python35\Scripts

Using c:\python35\lib\site-packages
Processing dependencies for pip
Finished processing dependencies for pip

C:\Python35\Scripts>
```

Step 3:

Now, your next steps is to install numpy package of python for pip. Numpy installation will be done with the help of the **pip install numpy command**. If your python has already this package, then it will show you "**Requirement already satisfied**" otherwise, it will install the package. **Pip list** command is used to check packages.

```
C:\Windows\System32\cmd.exe - pip install numpy

C:\Python35>cd Scripts

C:\Python35\Scripts>easy_install.exe pip
Searching for pip
Best match: pip 8.1.1
Adding pip 8.1.1 to easy-install.pth file
Installing pip3.5-script.py script to c:\python35\Scripts
Installing pip3.5.exe script to c:\python35\Scripts
Installing pip-script.py script to c:\python35\Scripts
Installing pip.exe script to c:\python35\Scripts
Installing pip3-script.py script to c:\python35\Scripts
Installing pip3.exe script to c:\python35\Scripts

Using c:\python35\lib\site-packages
Processing dependencies for pip
Finished processing dependencies for pip

C:\Python35\Scripts>pip list
pip (8.1.1)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install numpy
Collecting numpy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/b4/1b/36bd20a4a1f41729c406014974925598edaeca1ca2510a2843892329b2f1/numpy-1.16.4-cp35-cp35m-win_amd64.whl (11.9MB)
    20% |#####| 2.4MB 74kB/s eta 0:02:08
```

When downloading is finished, it shows a successful message and takes back your cursor in the scripts folder.

Step 4:

Next step is to install pip another package scipy with the help of **pip install scipy** command.

```

C:\Windows\System32\cmd.exe - pip install scipy
Installing pip3-script.py script to c:\python35\Scripts
Installing pip3.exe script to c:\python35\Scripts

Using c:\python35\lib\site-packages
Processing dependencies for pip
Finished processing dependencies for pip

C:\Python35\Scripts>pip list
pip (8.1.1)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install numpy
Collecting numpy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/b4/1b/36bd20a4a1f41729c4060149749
/numpy-1.16.4-cp35-cp35m-win_amd64.whl (11.9MB)
    100% |#####| 11.9MB 43kB/s
Installing collected packages: numpy
Successfully installed numpy-1.16.4
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install scipy
Collecting scipy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/ff/c5/64e6312d301c77f2a7acb5e7552
/scipy-1.3.0-cp35-cp35m-win_amd64.whl (30.4MB)

```

Once downloading is finished your cursor comes back in the scripts folder.


```
C:\Windows\System32\cmd.exe
C:\Python35\Scripts>pip list
pip (8.1.1)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install numpy
Collecting numpy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/b4/1b/36bd20a4a1f41729c406014974925598edaeca1ca2510a2843892329b2f1/numpy-1.16.4-cp35-cp35m-win_amd64.whl (11.9MB)
    100% |#####| 11.9MB 43kB/s
Installing collected packages: numpy
Successfully installed numpy-1.16.4
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install scipy
Collecting scipy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/ff/c5/64e6312d301c77f2a7acb5e755238bb8ba57e93feaade41ed73334ae2768/scipy-1.3.0-cp35-cp35m-win_amd64.whl (30.4MB)
    100% |#####| 30.5MB 23kB/s
Requirement already satisfied (use --upgrade to upgrade): numpy>=1.13.3 in c:\python35\lib\site-packages (from scipy)
Installing collected packages: scipy
Successfully installed scipy-1.3.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>
```

Step 5:

Now, check all the installed packages that are required for PyTorch using the **pip list** command.

```
C:\Windows\System32\cmd.exe
Cache entry deserialization failed, entry ignored
Downloading https://files.pythonhosted.org/packages/b4/1b/36bd20a4a1f41729c406014974925598edaeca1ca2510a2843892329b2f1
/numpy-1.16.4-cp35-cp35m-win_amd64.whl (11.9MB)
100% |#####| 11.9MB 43kB/s
Installing collected packages: numpy
Successfully installed numpy-1.16.4
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip install scipy
Collecting scipy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/ff/c5/64e6312d301c77f2a7acb5e755238bb8ba57e93feaade41ed73334ae2768
/scipy-1.3.0-cp35-cp35m-win_amd64.whl (30.4MB)
  100% |#####| 30.5MB 23kB/s
Requirement already satisfied (use --upgrade to upgrade): numpy>=1.13.3 in c:\python35\lib\site-packages (from scipy)
Installing collected packages: scipy
Successfully installed scipy-1.3.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip list
numpy (1.16.4)
pip (8.1.1)
scipy (1.3.0)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>
```

Step 6

Now, you have to go on <https://pytorch.org/> to get the installation command of PyTorch.

Here, you have to select your preferred PyTorch build, Operating System, Package, Language, and CUDA. It provides you two commands to install PyTorch in your windows.

Step 7:

Next step is to run both the command on your command prompt. Remember if you make any changes in this command, it will not install PyTorch and give an error message.

- pip3 install https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl

```
C:\Windows\System32\cmd.exe
Collecting scipy
  Cache entry deserialization failed, entry ignored
  Downloading https://files.pythonhosted.org/packages/ff/c5/64e6312d301c77f2a7acb5e755238bb8ba57e93feade41ed73334ae2768/scipy-1.3.0-cp35-cp35m-win_amd64.whl (30.4MB)
    100% |#####| 30.5MB 23kB/s
Requirement already satisfied (use --upgrade to upgrade): numpy>=1.13.3 in c:\python35\lib\site-packages (from scipy)
Installing collected packages: scipy
Successfully installed scipy-1.3.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip list
numpy (1.16.4)
pip (8.1.1)
scipy (1.3.0)
setuptools (20.10.1)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip3 install https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl
Collecting torch==1.1.0 from https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl
  Downloading https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl (99.6MB)
    100% |#####| 99.6MB 7.0kB/s
Requirement already satisfied (use --upgrade to upgrade): numpy in c:\python35\lib\site-packages (from torch==1.1.0)
Installing collected packages: torch
Successfully installed torch-1.1.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>
```

- o pip3 install https://download.pytorch.org/whl/cpu/torchvision-0.3.0-cp35-cp35m-win_amd64.whl

```
C:\Windows\System32\cmd.exe
Collecting torch==1.1.0 from https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl
  Downloading https://download.pytorch.org/whl/cpu/torch-1.1.0-cp35-cp35m-win_amd64.whl (99.6MB)
    100% |#####| 99.6MB 7.0kB/s
Requirement already satisfied (use --upgrade to upgrade): numpy in c:\python35\lib\site-packages (from torch==1.1.0)
Installing collected packages: torch
Successfully installed torch-1.1.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip3 install https://download.pytorch.org/whl/cpu/torchvision-0.3.0-cp35-cp35m-win_amd64.whl
Collecting torchvision==0.3.0 from https://download.pytorch.org/whl/cpu/torchvision-0.3.0-cp35-cp35m-win_amd64.whl
  Downloading https://download.pytorch.org/whl/cpu/torchvision-0.3.0-cp35-cp35m-win_amd64.whl (232kB)
    100% |#####| 235kB 853kB/s
Collecting pillow>=4.1.1 (from torchvision==0.3.0)
  Downloading https://files.pythonhosted.org/packages/e0/80/fbfcfccb93bc5ec296b3c910ff4086d247ed7a2f111c0d9a19ed9950b317/Pillow-6.0.0-cp35-cp35m-win_amd64.whl (2.0MB)
    100% |#####| 2.0MB 320kB/s
Requirement already satisfied (use --upgrade to upgrade): torch>=1.1.0 in c:\python35\lib\site-packages (from torchvision==0.3.0)
Collecting six (from torchvision==0.3.0)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
Requirement already satisfied (use --upgrade to upgrade): numpy in c:\python35\lib\site-packages (from torchvision==0.3.0)
Installing collected packages: pillow, six, torchvision
Successfully installed pillow-6.0.0 six-1.12.0 torchvision-0.3.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>
```

Step 8:

Now, rerun pip list command to check PyTorch is run successfully or not.

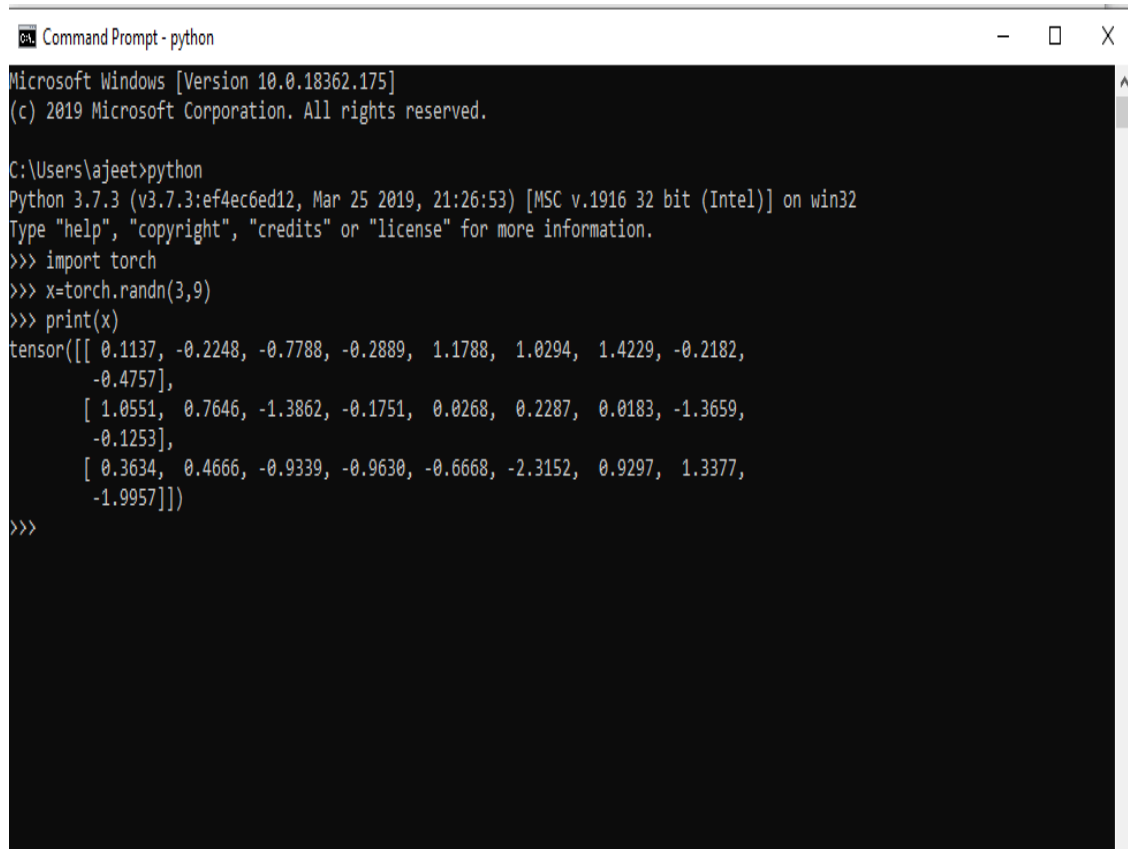
```
C:\Windows\System32\cmd.exe
100% |#####| 235kB 853kB/s
Collecting pillow>=4.1.1 (from torchvision==0.3.0)
  Downloading https://files.pythonhosted.org/packages/e0/80/fbfcfccb93bc5ec296b3c910ff4086d247ed7a2f111c0d9a19ed9950b317/Pillow-6.0.0-cp35-cp35m-win_amd64.whl (2.0MB)
100% |#####| 2.0MB 320kB/s
Requirement already satisfied (use --upgrade to upgrade): torch>=1.1.0 in c:\python35\lib\site-packages (from torchvision==0.3.0)
Collecting six (from torchvision==0.3.0)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
Requirement already satisfied (use --upgrade to upgrade): numpy in c:\python35\lib\site-packages (from torchvision==0.3.0)
Installing collected packages: pillow, six, torchvision
Successfully installed pillow-6.0.0 six-1.12.0 torchvision-0.3.0
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>pip list
numpy (1.16.4)
Pillow (6.0.0)
pip (8.1.1)
scipy (1.3.0)
setuptools (20.10.1)
six (1.12.0)
torch (1.1.0)
torchvision (0.3.0)
You are using pip version 8.1.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python35\Scripts>
```

Step 9:

Now, test PyTorch. Run python command to work with python. Import torch to work with PyTorch and perform the operation.



```
Command Prompt - python
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

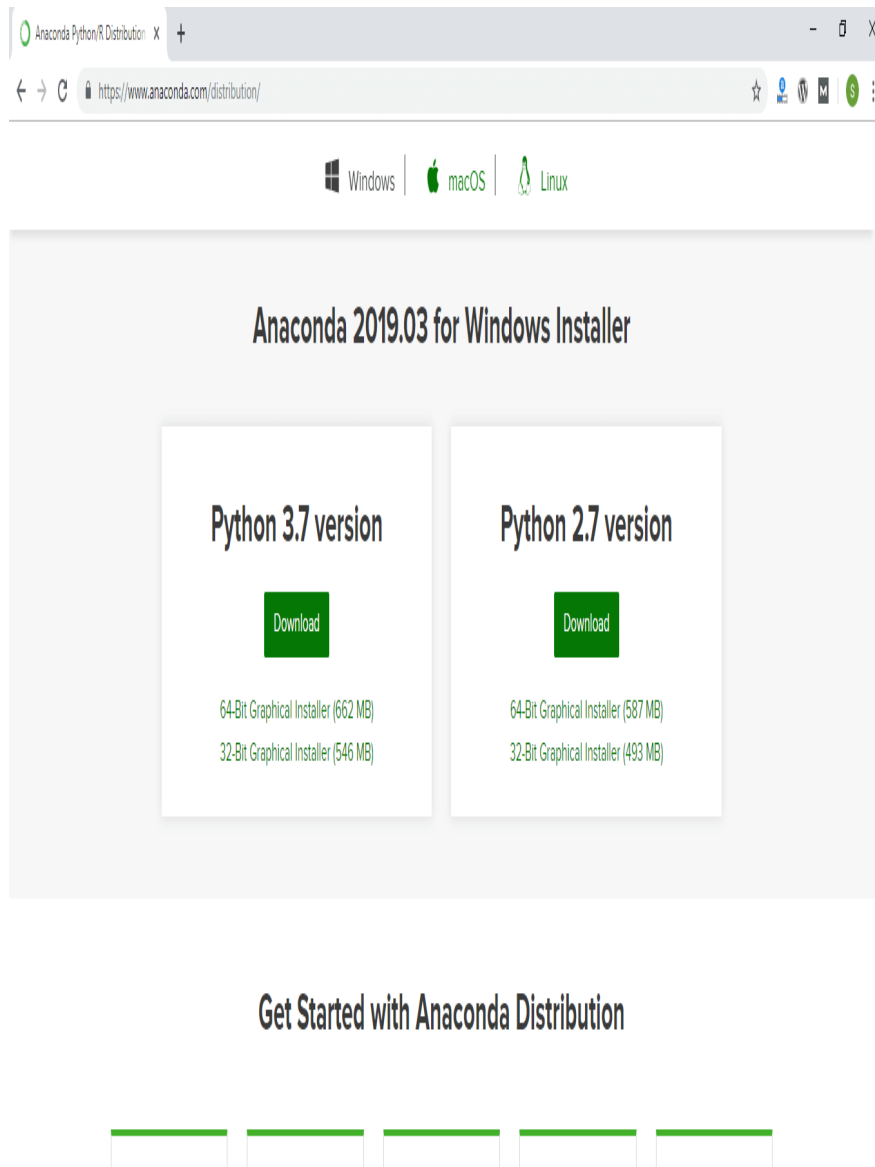
C:\Users\ajeet>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> x=torch.randn(3,9)
>>> print(x)
tensor([[ 0.1137, -0.2248, -0.7788, -0.2889,  1.1788,  1.0294,  1.4229, -0.2182,
         -0.4757],
        [ 1.0551,  0.7646, -1.3862, -0.1751,  0.0268,  0.2287,  0.0183, -1.3659,
         -0.1253],
        [ 0.3634,  0.4666, -0.9339, -0.9630, -0.6668, -2.3152,  0.9297,  1.3377,
        -1.9957]])
>>>
```

Installation on Windows using Conda

This tutorial defines step by step installation of PyTorch. To install PyTorch using Conda you have to follow the following steps

Step 1:

First, you have to install Anaconda's latest version in your system. To install Anaconda, you have to go through <https://www.anaconda.com/distribution/>.



Step 2:

Now, run your Anaconda setup and install it completely. Once your installation of Anaconda is complete, an Anaconda command prompt appears to you.



Step 3:

Next step is to install PyTorch using Anaconda command prompt. To install PyTorch, you have to run the installation command of PyTorch on your command prompt. This command is available on <https://pytorch.org/>.

The screenshot shows the PyTorch website with a navigation bar and a main content area. The main content area includes a table for selecting PyTorch build, OS, package, language, and CUDA version, and a section for cloud platforms.

PyTorch Build	Stable (1.1)	Preview (Nightly)			
Your OS	Linux	Mac	Windows		
Package	Conda	Pip	LibTorch	Source	
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7	C++
CUDA	9.0	10.0	None		

Run this Command:

```
conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
```

Previous versions of PyTorch

Cloud Platforms:


- AWS
- Google Cloud Platform
- Microsoft Azure

Select language and cuda version as per your requirement.

Step 4

Now, run **python -version**, and **Conda -version** command to check **Conda** and **python** packages are installed or not. After that, you run the given command in your command prompt. Remember the command which you run on

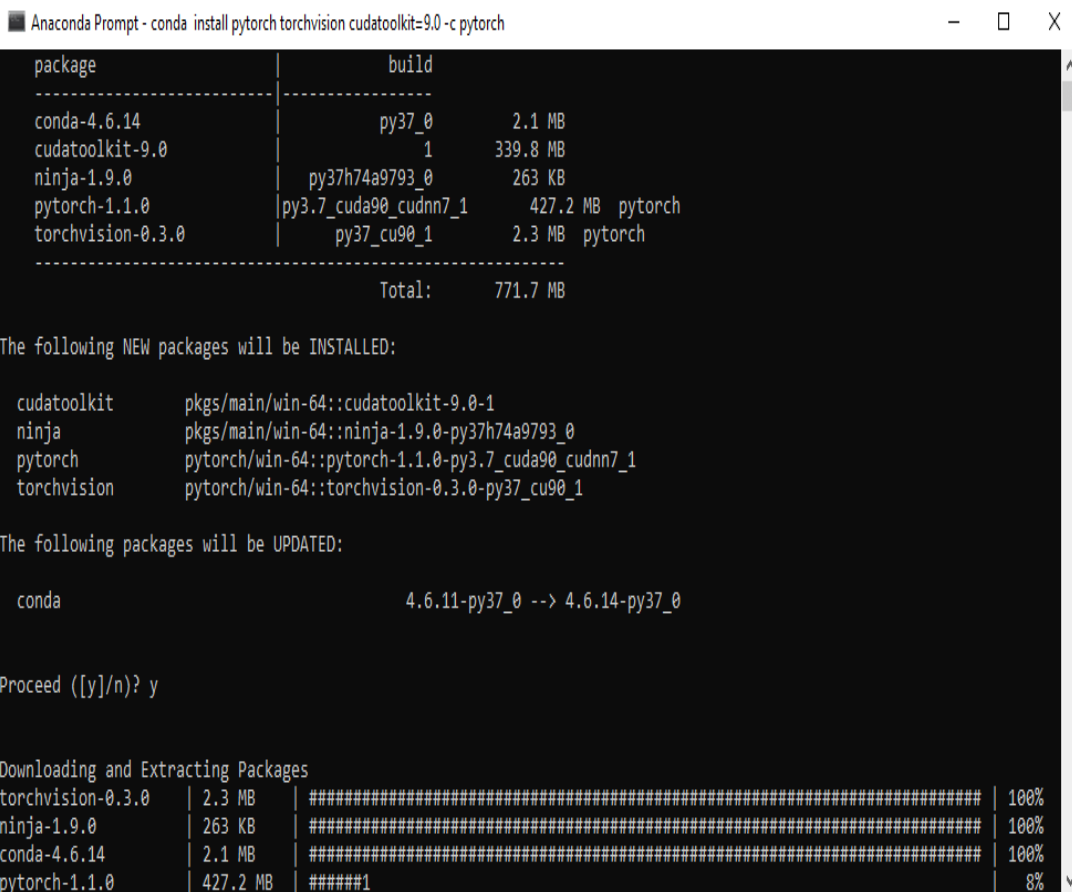
command prompt is similar to the given command. If it is not similar, then it will generate error message and installation will become unsuccessful.

 Anaconda Prompt - conda install pytorch torchvision cudatoolkit=9.0 -c pytorch

```
(base) C:\Users\ajeet>python --version
Python 3.7.3

(base) C:\Users\ajeet>conda --version
conda 4.6.11

(base) C:\Users\ajeet>conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: | _
```



Anaconda Prompt - conda install pytorch torchvision cudatoolkit=9.0 -c pytorch

package	build	size
conda-4.6.14	py37_0	2.1 MB
cudatoolkit-9.0	1	339.8 MB
ninja-1.9.0	py37h74a9793_0	263 KB
pytorch-1.1.0	py3.7_cuda90_cudnn7_1	427.2 MB pytorch
torchvision-0.3.0	py37_cu90_1	2.3 MB pytorch
Total:		771.7 MB

The following NEW packages will be INSTALLED:

cudatoolkit	pkgs/main/win-64::cudatoolkit-9.0-1
ninja	pkgs/main/win-64::ninja-1.9.0-py37h74a9793_0
pytorch	pytorch/win-64::pytorch-1.1.0-py3.7_cuda90_cudnn7_1
torchvision	pytorch/win-64::torchvision-0.3.0-py37_cu90_1

The following packages will be UPDATED:

conda	4.6.11-py37_0 --> 4.6.14-py37_0
-------	---------------------------------

Proceed ([y]/n)? y

Downloading and Extracting Packages

torchvision-0.3.0	2.3 MB	#####	100%
ninja-1.9.0	263 KB	#####	100%
conda-4.6.14	2.1 MB	#####	100%
pytorch-1.1.0	427.2 MB	#####1	8%

It will take some time to download and install all the packages. After completion of your command, your cursor switch to your directory's folder.

```
Anaconda Prompt

conda                                4.6.11-py37_0 --> 4.6.14-py37_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
torchvision-0.3.0 | 2.3 MB | ##### | 100%
ninja-1.9.0       | 263 KB | ##### | 100%
conda-4.6.14      | 2.1 MB | ##### | 100%
pytorch-1.1.0     | 427.2 MB | ##### | 100%
cudatoolkit-9.0   | 339.8 MB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\ajeet>ET _syp=%~dpA
'ET' is not recognized as an internal or external command,
operable program or batch file.

(base) C:\Users\ajeet>IF NOT EXIST "!_syp!\Scripts\conda.exe"
Collecting package metadata: done
Solving environment: done

# All requested packages already installed.

(base) C:\Users\ajeet>
(base) C:\Users\ajeet>
```

Step 5:

Now, perform `conda list pytorch` command to check all the package are installed successfully or not.

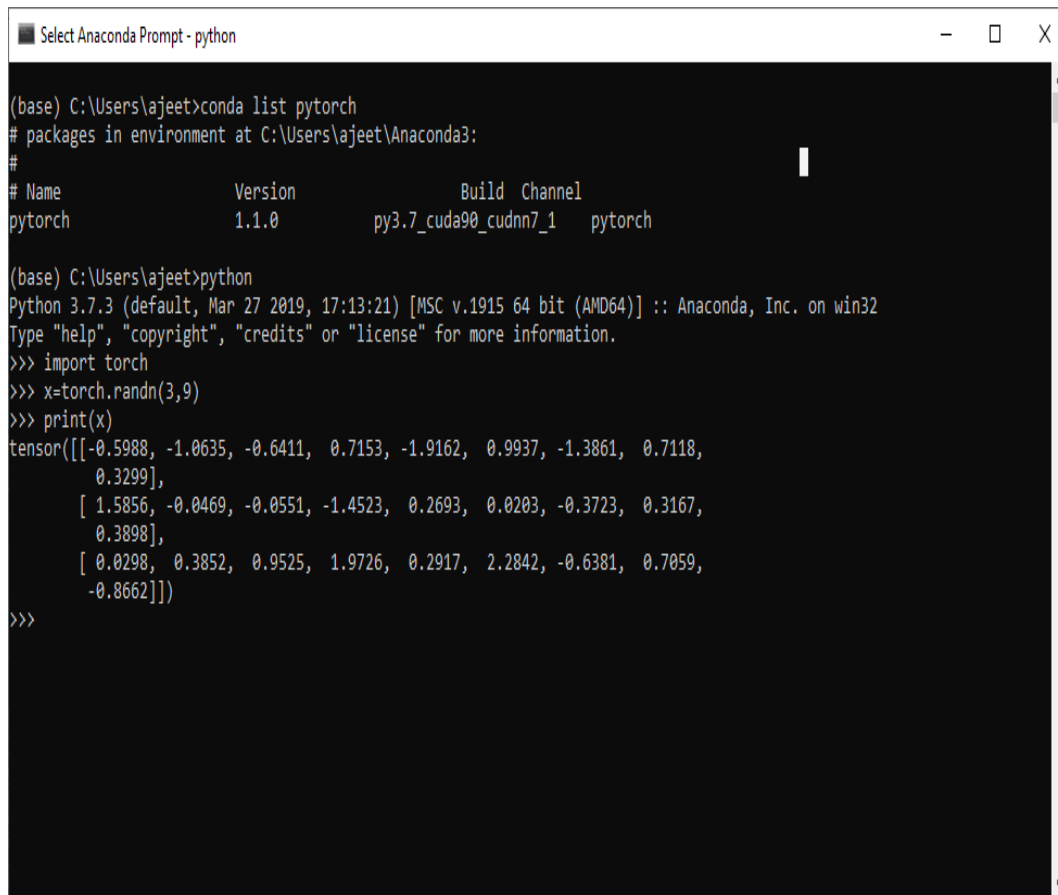
```
Anaconda Prompt

(base) C:\Users\ajeet>conda list pytorch
# packages in environment at C:\Users\ajeet\Anaconda3:
#
# Name          Version      Build Channel
pytorch         1.1.0        py3.7_cuda90_cudnn7_1 pytorch

(base) C:\Users\ajeet>
```

Step 6:

Now, test PyTorch. Run `python` command to work with python. Import `torch` to work with PyTorch and perform the operation.



```
Select Anaconda Prompt - python

(base) C:\Users\ajeet>conda list pytorch
# packages in environment at C:\Users\ajeet\Anaconda3:
#
# Name          Version          Build Channel
pytorch         1.1.0            py3.7_cuda90_cudnn7_1  pytorch

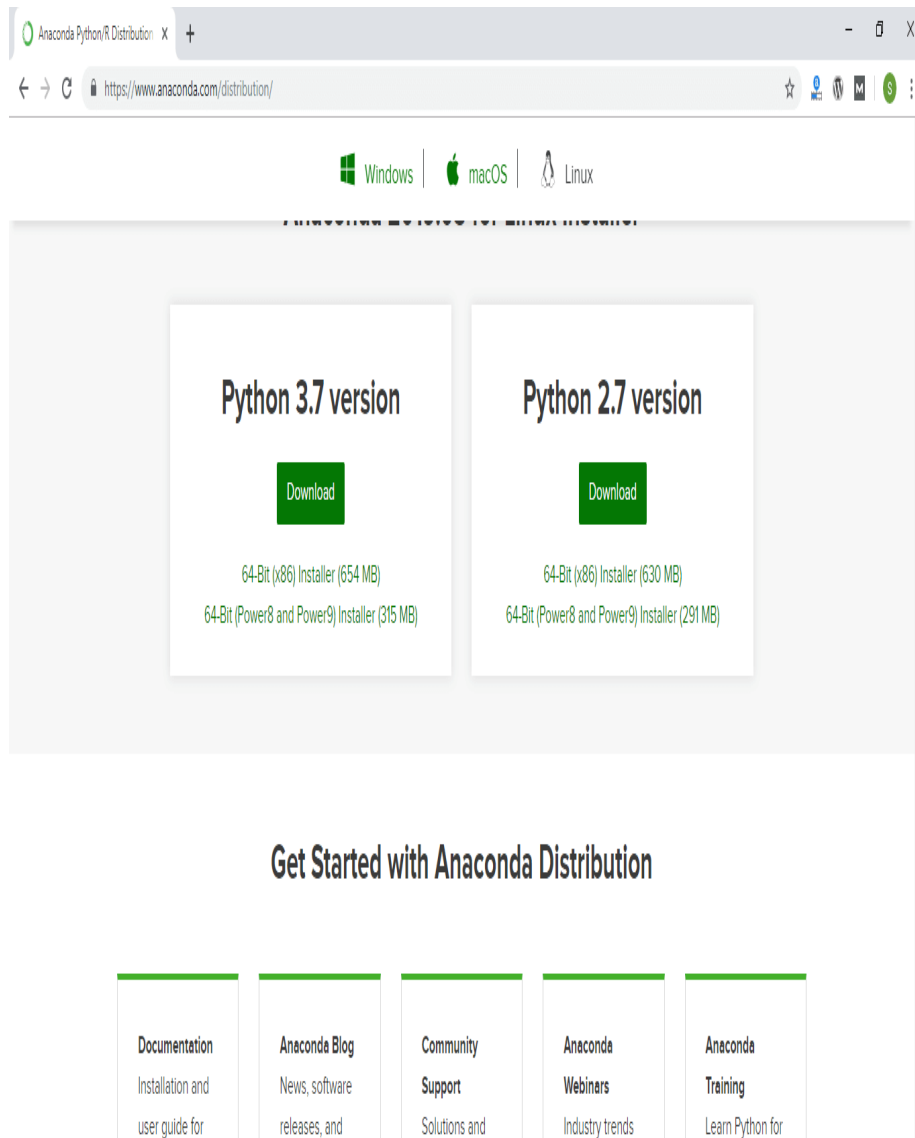
(base) C:\Users\ajeet>python
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> x=torch.randn(3,9)
>>> print(x)
tensor([[ -0.5988, -1.0635, -0.6411,  0.7153, -1.9162,  0.9937, -1.3861,  0.7118,
          0.3299],
        [ 1.5856, -0.0469, -0.0551, -1.4523,  0.2693,  0.0203, -0.3723,  0.3167,
          0.3898],
        [ 0.0298,  0.3852,  0.9525,  1.9726,  0.2917,  2.2842, -0.6381,  0.7059,
        -0.8662]])
>>>
```

Installation on Linux

PyTorch installation in Linux is similar to the installation of Windows using Conda. To install PyTorch in your Linux system, you have to follow the steps which are giving below.

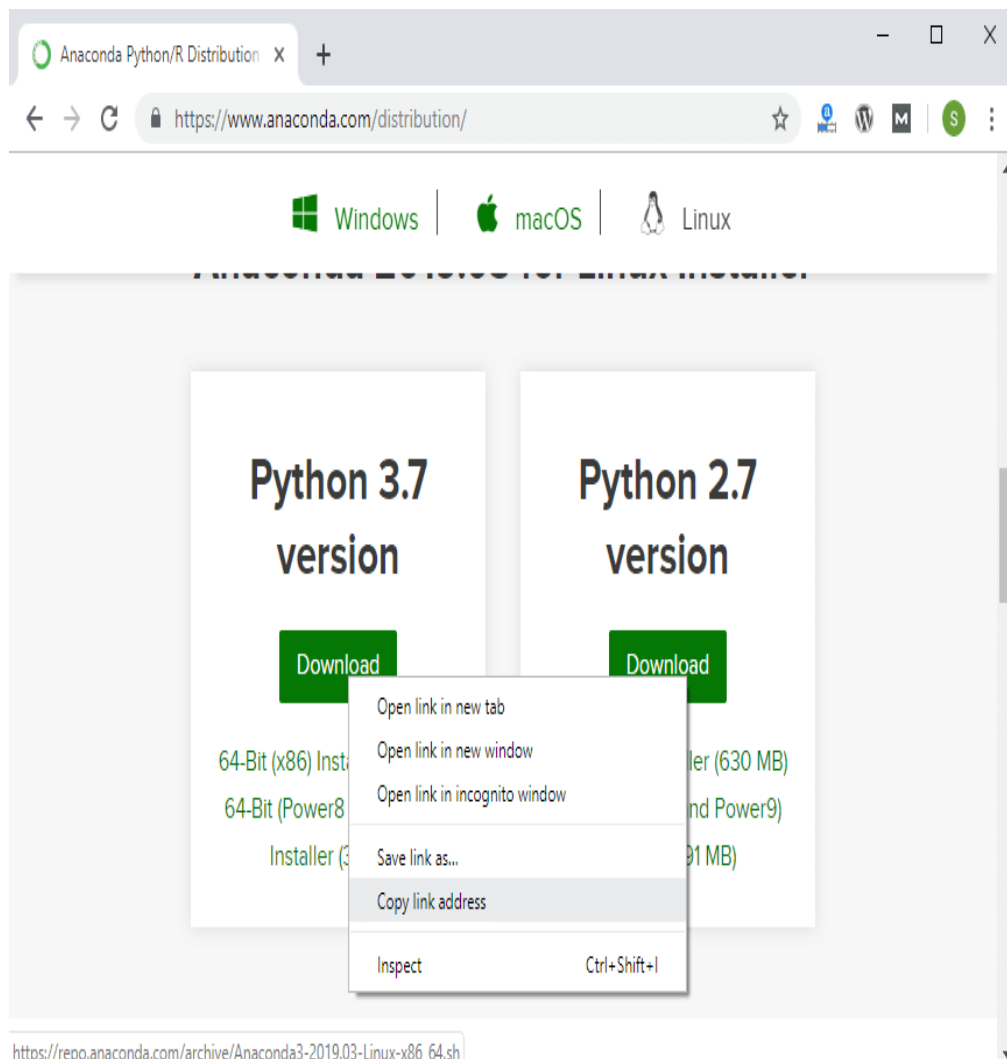
Step 1:

Your first step is to download Anaconda in your Linux operating system. To download it, you have to go through the following link <https://www.anaconda.com/distribution/>.



Step 2:

Here, you choose the latest version of python, i.e., 3.7 and click right button of the mouse and copy link address to install it.



Step 3:

Open your terminal and run the copy link on the terminal using `wget <link>` command. This command download Anaconda in your Linux system.


```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~  
File Edit View Search Terminal Help  
javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~$ wget https://repo.anaconda.com/arch  
ive/Anaconda3-2019.03-Linux-x86_64.sh  
--2019-06-18 18:05:39-- https://repo.anaconda.com/archive/Anaconda3-2019.03-Lin  
ux-x86_64.sh  
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2  
606:4700::6810:8303, ...  
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connect  
ed.  
HTTP request sent, awaiting response... 200 OK  
Length: 685906562 (654M) [application/x-sh]  
Saving to: 'Anaconda3-2019.03-Linux-x86_64.sh'  
  
An 7%[> ] 45.98M 4.95MB/s eta 2m 0s
```

It takes a few seconds to download. Once downloading is complete your cursor go back to your home directory.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~
File Edit View Search Terminal Help
javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~$ wget https://repo.anaconda.com/arch
ive/Anaconda3-2019.03-Linux-x86_64.sh
--2019-06-18 18:05:39-- https://repo.anaconda.com/archive/Anaconda3-2019.03-Lin
ux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2
606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connect
ed.
HTTP request sent, awaiting response... 200 OK
Length: 685906562 (654M) [application/x-sh]
Saving to: 'Anaconda3-2019.03-Linux-x86_64.sh'

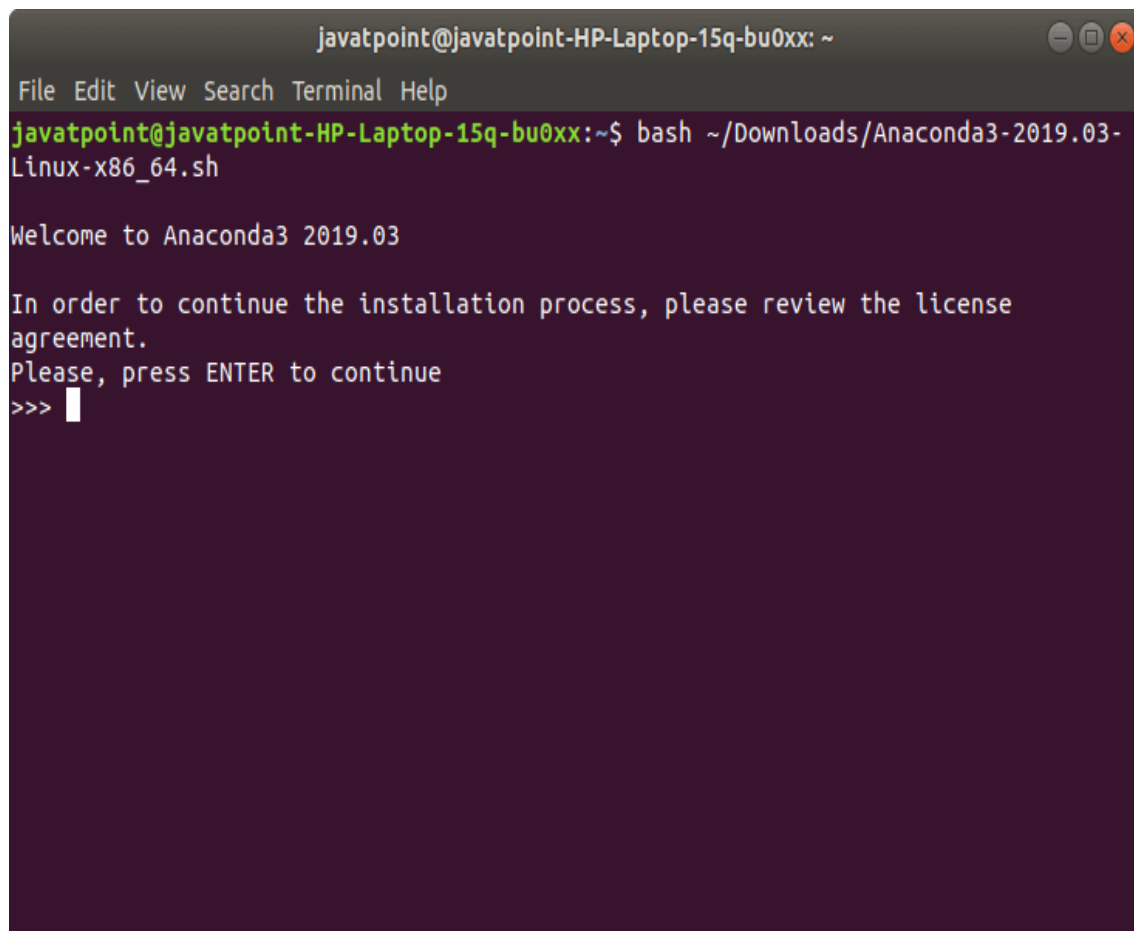
Anaconda3-2019.03-L 100%[=====] 654.13M 4.93MB/s in 2m 14s

2019-06-18 18:07:53 (4.89 MB/s) - 'Anaconda3-2019.03-Linux-x86_64.sh' saved [685
906562/685906562]

javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~$
```

Step 4:

Next step is to install Anaconda in your system. To install it, you have to first enter in downloads directory or where you have downloaded your anaconda. Installation is done with bash file because in Linux when you download Anaconda, It downloaded as bash file. So, to install Anaconda, you have to run the **bash ~/Downloads/Anaconda3-2019.03-Linux-x86_64.sh** command for the latest version of python 3.7 or to run the **bash ~/Downloads/Anaconda2-2019.03-Linux-x86_64.sh** command for python 2.7.

A terminal window titled 'javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~\$' and the command executed is 'bash ~/Downloads/Anaconda3-2019.03-Linux-x86_64.sh'. The output shows 'Welcome to Anaconda3 2019.03' followed by a license agreement notice: 'In order to continue the installation process, please review the license agreement. Please, press ENTER to continue'. The prompt is now '>>>' with a cursor.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~
File Edit View Search Terminal Help
javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~$ bash ~/Downloads/Anaconda3-2019.03-
Linux-x86_64.sh

Welcome to Anaconda3 2019.03

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> 
```

Here, you have to press the enter button to continue. When you press enter your installation is started. After a few installations it asks you one more question, i.e., Do you accept the license terms? You give it an answer by typing, yes.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~  
File Edit View Search Terminal Help  
  
openssl  
    The OpenSSL Project is a collaborative effort to develop a robust, commercia  
l-grade, full-featured, and Open Source toolkit implementing the Transport Layer  
Security (TLS) and Secure Sockets Layer (SSL) protocols as well as a full-stren  
gth general purpose cryptography library.  
  
pycrypto  
    A collection of both secure hash functions (such as SHA256 and RIPEMD160), a  
nd various encryption algorithms (AES, DES, RSA, ElGamal, etc.).  
  
pyopenssl  
    A thin Python wrapper around (a subset of) the OpenSSL library.  
  
kerberos (krb5, non-Windows platforms)  
    A network authentication protocol designed to provide strong authentication  
for client/server applications by using secret-key cryptography.  
  
cryptography  
    A Python library which exposes cryptographic recipes and primitives.  
  
Do you accept the license terms? [yes|no]  
[no] >>>
```

Step 5:

When you type yes and press enter, your installation of anaconda starts. After a few installations once again it asks you one last question, i.e., Do you wish the installer to initialize Anaconda3 by running conda init? You again type yes as an answer, and after that, your cursor comes back to downloads directory.

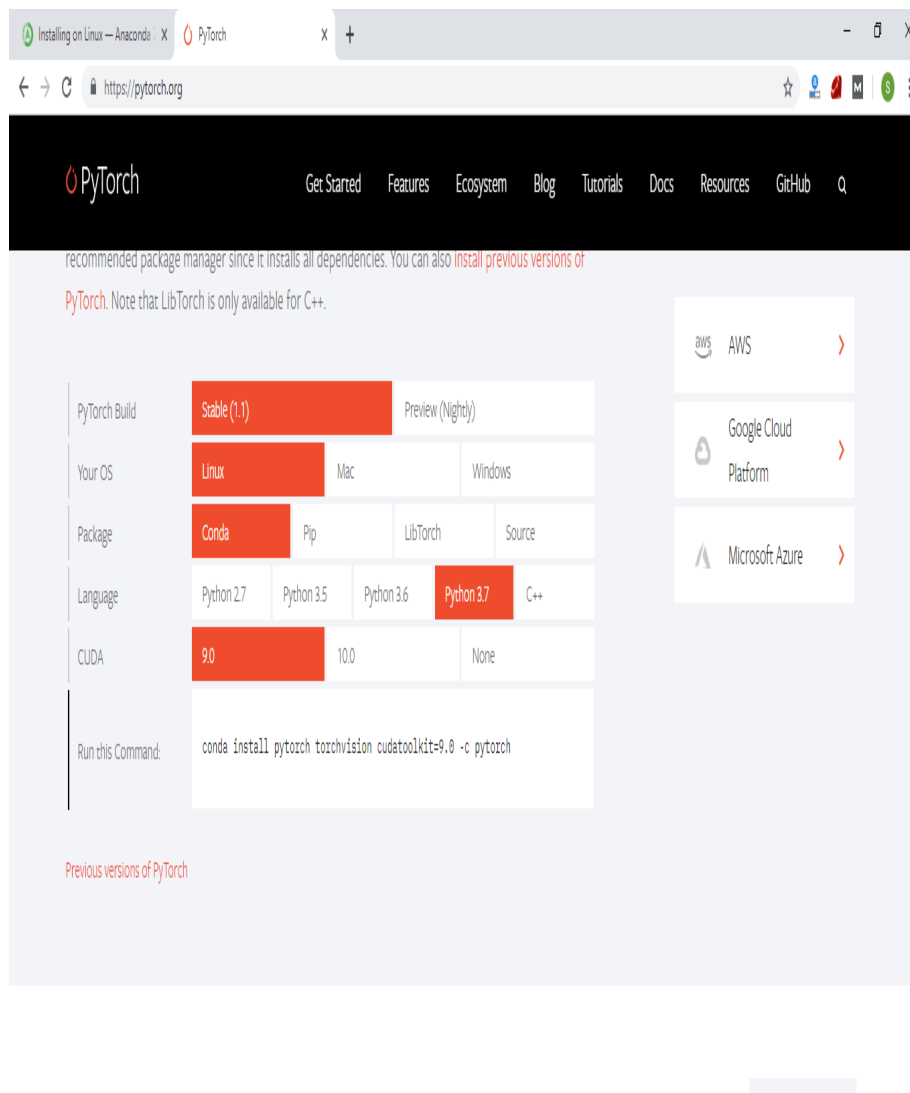
```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~
File Edit View Search Terminal Help
installing: mkl_fft-1.0.10-py37ha843d7b_0 ...
installing: numpy-1.16.2-py37h7e9f1db_0 ...
installing: numba-0.43.1-py37h962f231_0 ...
installing: numexpr-2.6.9-py37h9e4a6bb_0 ...
installing: pandas-0.24.2-py37he6710b0_0 ...
installing: pytest-arraydiff-0.3-py37h39e3cac_0 ...
installing: pytest-doctestplus-0.3.0-py37_0 ...
installing: pywavelets-1.0.2-py37hdd07704_0 ...
installing: scipy-1.2.1-py37h7c811a0_0 ...
installing: bkcharts-0.2-py37_0 ...
installing: dask-1.1.4-py37_1 ...
installing: patsy-0.5.1-py37_0 ...
installing: pytables-3.5.1-py37h71ec239_0 ...
installing: pytest-astropy-0.5.0-py37_0 ...
installing: scikit-image-0.14.2-py37he6710b0_0 ...
installing: scikit-learn-0.20.3-py37hd81dba3_0 ...
installing: astropy-3.1.2-py37h7b6447c_0 ...
installing: statsmodels-0.9.0-py37h035aef0_0 ...
installing: seaborn-0.9.0-py37_0 ...
installing: anaconda-2019.03-py37_0 ...
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> 
```

Step 6:

Now next step is to run **source ~/.bashrc** and **anaconda-navigator** and then we install the PyTorch.

Step 7:

Next step is to go to the official site of PyTorch using <https://pytorch.org/> link.



Here, you have to select your operating system, package, language, and CUDA version. I am using Conda package with python 3.7 and CUDA 9.0.

Step 8:

Now you have to run the command given by the official website on your terminal. Remember the command which you run is similar to the given command; otherwise, it will generate the error message with the unsuccessful installation.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~/Downloads
File Edit View Search Terminal Help
(base) javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~$ cd Downloads
(base) javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~/Downloads$ python --version
Python 3.7.3
(base) javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~/Downloads$ conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
WARNING: The conda.compat module is deprecated and will be removed in a future release.
█
```

After a few seconds, it asks you to update packages if available. We have to give its answer by writing y.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~/Downloads
File Edit View Search Terminal Help
pytorch-1.1.0 | py3.7_cuda9.0.176_cudnn7.5.1_0 376.8 MB p
pytorch
torchvision-0.3.0 | py37_cu9.0.176_1 3.7 MB pytorch
-----
Total: 724.7 MB

The following NEW packages will be INSTALLED:

cudatoolkit pkgs/main/linux-64::cudatoolkit-9.0-h13b8566_0
ninja pkgs/main/linux-64::ninja-1.9.0-py37hfd86e86_0
pytorch pytorch/linux-64::pytorch-1.1.0-py3.7_cuda9.0.176_cudnn7.5.1_0
torchvision pytorch/linux-64::torchvision-0.3.0-py37_cu9.0.176_1

The following packages will be UPDATED:

conda 4.6.11-py37_0 --> 4.6.14-py37_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
cudatoolkit-9.0 | 340.4 MB | #### | 12%
```

Once you give its answer, it starts downloading all packages such as PyTorch, Cudatoolkit, Conda, torch, etc.

```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~/Downloads
File Edit View Search Terminal Help
  cudatoolkit      pkgs/main/linux-64::cudatoolkit-9.0-h13b8566_0
  ninja            pkgs/main/linux-64::ninja-1.9.0-py37hfd86e86_0
  pytorch          pytorch/linux-64::pytorch-1.1.0-py3.7_cuda9.0.176_cudnn7.5.
1_0
  torchvision      pytorch/linux-64::torchvision-0.3.0-py37_cu9.0.176_1

The following packages will be UPDATED:

  conda                                4.6.11-py37_0 --> 4.6.14-py37_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
cudatoolkit-9.0      | 340.4 MB | ##### | 100%
ninja-1.9.0          | 1.6 MB   | ##### | 100%
pytorch-1.1.0        | 376.8 MB | ##### | 100%
torchvision-0.3.0    | 3.7 MB   | ##### | 100%
conda-4.6.14         | 2.1 MB   | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~/Downloads$
```

Step 9:

Now, PyTorch installs successfully. It's time to test PyTorch by executing torch program.


```
javatpoint@javatpoint-HP-Laptop-15q-bu0xx: ~/Downloads
File Edit View Search Terminal Help

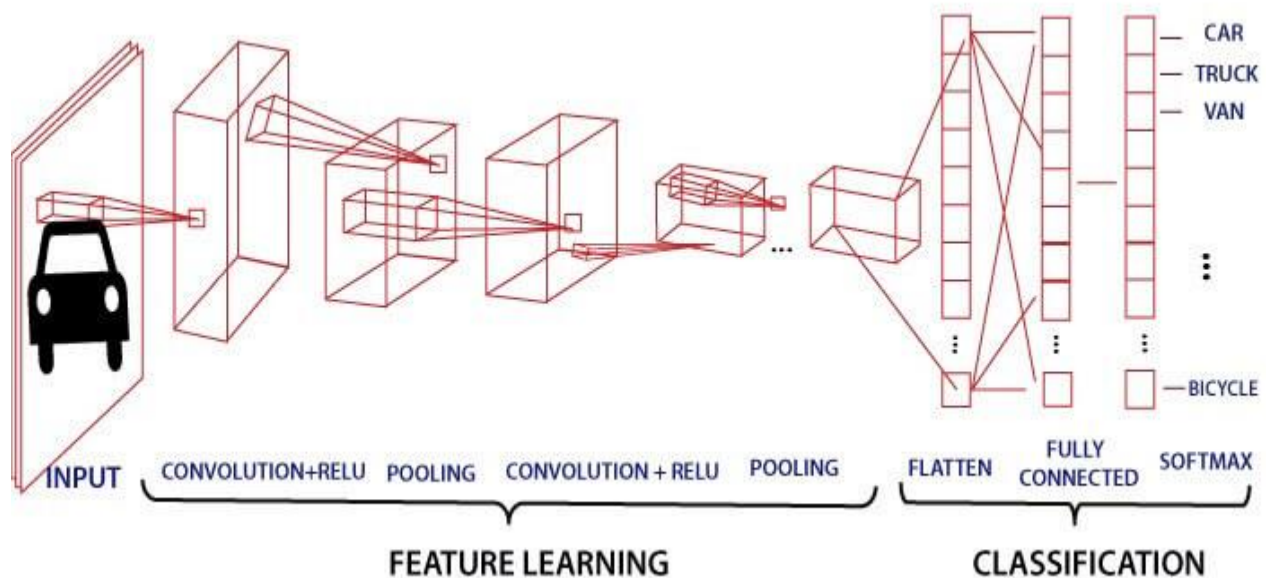
Downloading and Extracting Packages
cudatoolkit-9.0      | 340.4 MB | ##### | 100%
ninja-1.9.0         | 1.6 MB  | ##### | 100%
pytorch-1.1.0       | 376.8 MB | ##### | 100%
torchvision-0.3.0   | 3.7 MB  | ##### | 100%
conda-4.6.14        | 2.1 MB  | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) javatpoint@javatpoint-HP-Laptop-15q-bu0xx:~/Downloads$ python
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> x=torch.randn(3,9)
>>> print(x)
tensor([[ 0.9651,  1.1381,  0.2266,  1.4546, -0.7287,  0.4803, -0.4339,  1.0448,
         -1.6596],
        [ 0.5412, -2.8954, -0.6761,  1.8242, -0.7009, -1.1213,  1.4487, -1.5461,
          0.6742],
        [-0.0953,  1.4084, -0.2466,  1.3165,  0.9270,  1.1698,  1.8685,  1.5470,
         -2.1337]])
>>> 
```

Convolutional Neural Network In PyTorch

Convolutional Neural Network is one of the main categories to do image classification and image recognition in neural networks. Scene labeling, objects detections, and face recognition, etc., are some of the areas where convolutional neural networks are widely used.

CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc. The computer sees an image as an array of pixels and depends on the resolution of the image. Based on image resolution, it will see as $h * w * d$, where h = height w = width and d = dimension. For example, An RGB image is $6 * 6 * 3$ array of the matrix, and the grayscale image is $4 * 4 * 1$ array of the matrix.

In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object with probabilistic values 0 and 1.



Convolution Layer

Convolution layer is the first layer to extract features from an input image. By learning image features using a small square of input data, the convolutional layer preserves the relationship between pixels. It is a mathematical operation which takes two inputs such as image matrix and a kernel or filter.

- The dimension of the image matrix is $h \times w \times d$.
- The dimension of the filter is $f_h \times f_w \times d$.
- The dimension of the output is $(h-f_h+1) \times (w-f_w+1) \times 1$.

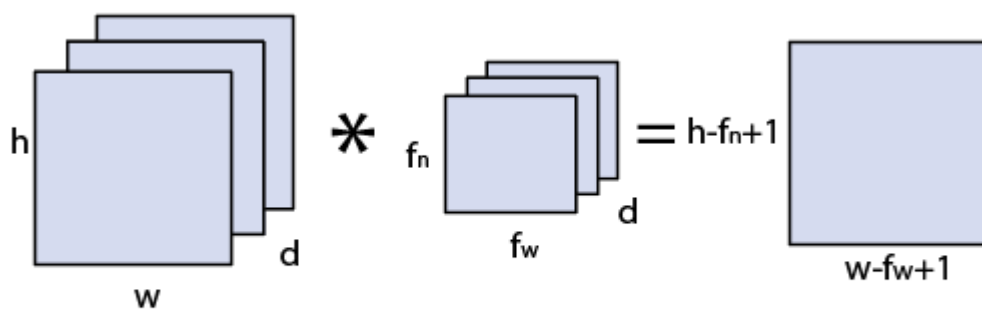


Image matrix multiplies kernel or filter matrix

Let's start with consideration a 5×5 image whose pixel values are 0, 1, and filter matrix 3×3 as:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

5×5 – Image Matrix 3×3 – Filter Matrix

The convolution of 5*5 image matrix multiplies with 3*3 filter matrix is called "**Features Map**" and show as an output.

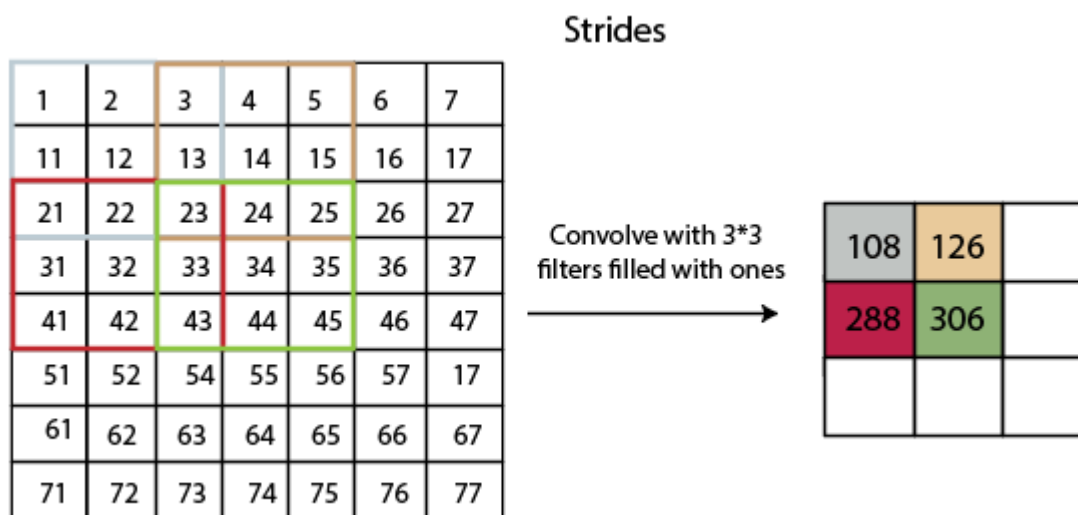
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

Convolved Feature

Convolution of an image with different filters can perform an operation such as blur, sharpen, and edge detection by applying filters.

Strides

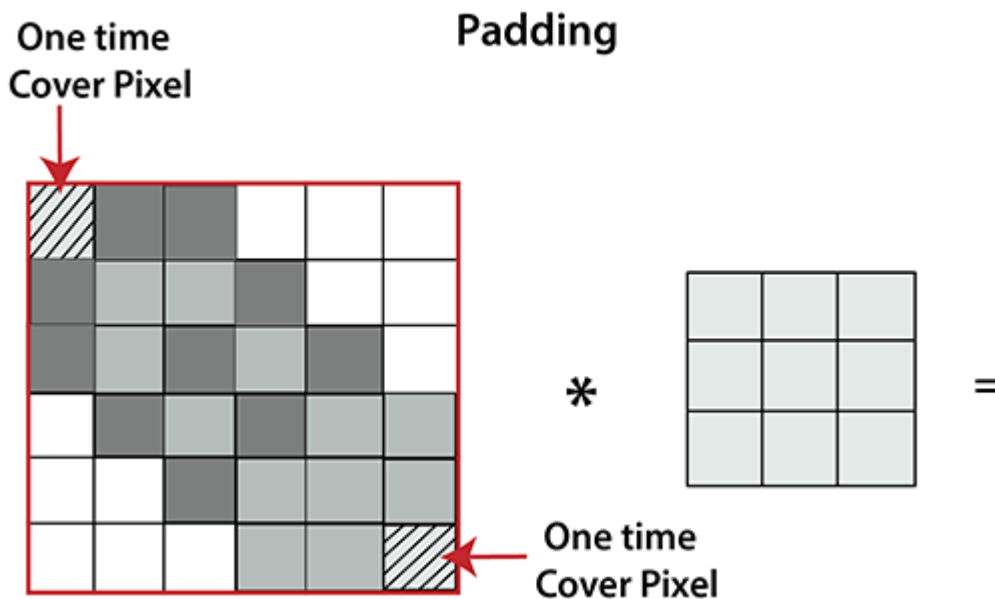
Stride is the number of pixels which are shift over the input matrix. When the stride is equaled to 1, then we move the filters to 1 pixel at a time and similarly, if the stride is equaled to 2, then we move the filters to 2 pixels at a time. The following figure shows that the convolution would work with a stride of 2.



Padding

Padding plays a crucial role in building the convolutional neural network. If the image will get shrink and if we will take a neural network with 100's of layers on it, it will give us a small image after filtered in the end.

If we take a three by three filter on top of a grayscale image and do the convolving then what will happen?



It is clear from the above picture that the pixel in the corner will only get covered one time, but the middle pixel will get covered more than once. It means that we have more information on that middle pixel, so there are two downsides:

- Shrinking outputs
- Losing information on the corner of the image.

To overcome this, we have introduced padding to an image. **"Padding is an additional layer which can add to the border of an image."**

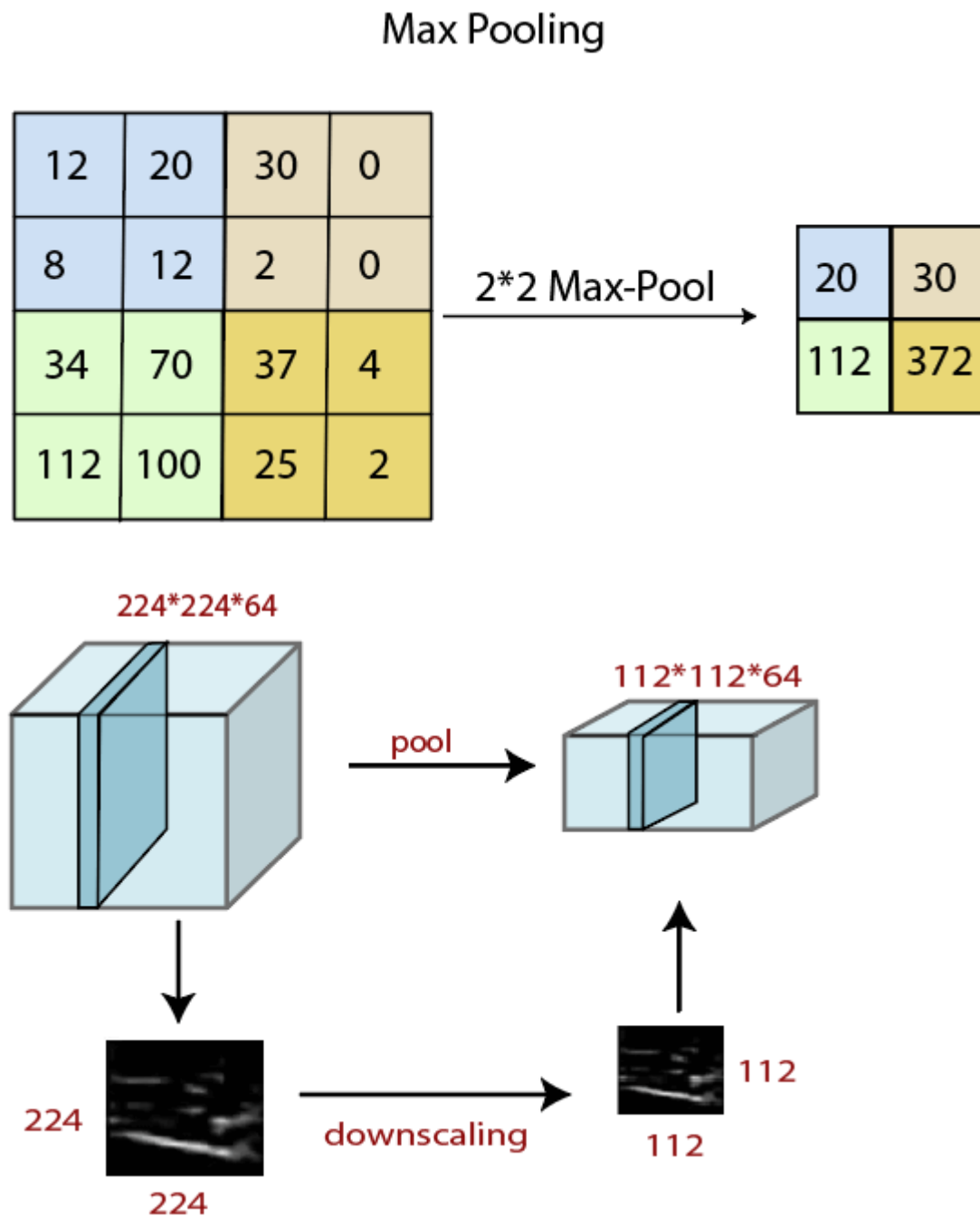
Pooling Layer

Pooling layer plays an important role in pre-processing of an image. Pooling layer reduces the number of parameters when the images are too large. Pooling is "**downscaling**" of the image obtained from the previous layers. It can be compared to shrinking an image to reduce its pixel density. Spatial pooling is also called downsampling or subsampling, which reduces the dimensionality of each map but retains the important information. There are the following types of spatial pooling:

Max Pooling

Max pooling is a **sample-based discretization process**. Its main objective is to downscale an input representation, reducing its dimensionality and allowing for the assumption to be made about features contained in the sub-region binned.

Max pooling is done by applying a max filter to non-overlapping sub-regions of the initial representation.



Average Pooling

Down-scaling will perform through average pooling by dividing the input into rectangular pooling regions and computing the average values of each region.

Syntax

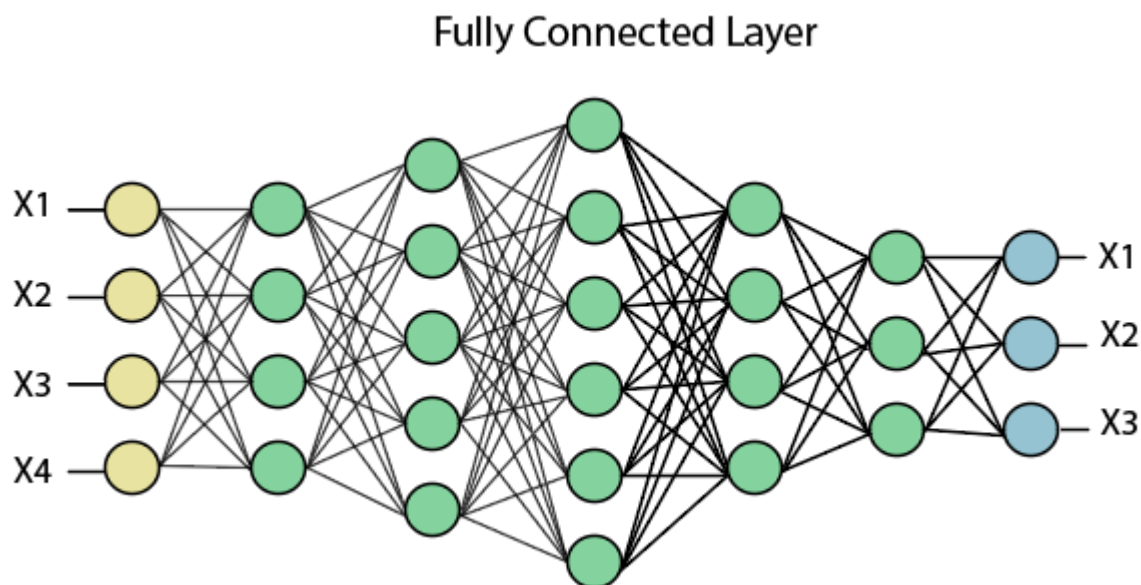
```
layer = averagePooling2dLayer(poolSize)
layer = averagePooling2dLayer(poolSize,Name,Value)
```

Sum Pooling

The sub-region for **sum pooling** or **mean pooling** are set exactly the same as for **max-pooling** but instead of using the max function we use sum or mean.

Fully Connected Layer

The fully connected layer is a layer in which the input from the other layers will be flattened into a vector and sent. It will transform the output into the desired number of classes by the network.



In the above diagram, the feature map matrix will be converted into the vector such as **x1, x2, x3... xn** with the help of fully connected layers. We will combine features to create a model and apply the activation function such as **softmax** or **sigmoid** to classify the outputs as a car, dog, truck, etc.

