## 

## 

## 

## **Get Started with TensorFlow**

*Your first step in Data Science*

## [ https://www.tensorflow.org ]

**Table of Content**

**Contents Page**

1. **Machine Learning…………………………..……………………………... 2-9**
   1. Introduction to machine learning……………………………………………………… 2
   2. Types of Machine Learning……………………………………………………............ 3-7
      1. Supervised Learning………………………………………………………… .. 3
      2. Unsupervised Learning………………………………………………………. 4

.Batch and Online Learning………………………………………………… 5

* + 1. Instance-Based Versus Model-Based Learning………………………………. 6-7
  1. Main Challenges of Machine Learning……………………………………………….. 7-8
     1. Insufficient Quantity of Training Data…………………………………………. 7
     2. Poor Quality Data………………………………………………………………. 8
     3. Overfitting and Underfitting ……….….……………………………………….. 8
  2. Testing and Validating …………………..……………………………………………… 9
  3. MNIST Dataset………………………………………………………………………….. 9

1. **TensorFlow………………………………………………………………... 10-16**
   1. Introduction to Tensorflow…..……………………………………………………… 10
   2. Installation of Tensorflow…..…………………………………………………………. 11
   3. Hello World with Tensorflow…..……………………………………………………… 11
   4. Tensors, Variables and Placeholders…..…………………………………………….. 11-12
   5. Creating Your First Graph and Running It in a Session…..………………………….... 13
   6. Lifecycle of a Node Value…..………………………………………………………. 13-14
   7. Linear Regression…..……………………………………………….…………………. 14
   8. Gradient Descent ………………………………………………………………………. 15
   9. Model Optimization……………………………………………………………………. 16
   10. Tensorboard….... ………………………………………………………………………. 16
2. **Neural Network………………………………………………………….. 17-19**
   1. Convolutional Neural Network………………………………………………………... 17
   2. Convolution Layer — The Kernel……………………………………………………. 17-18
   3. Pooling Layer………………………………………………………………………….. 19
3. **End-to-End Machine Learning Project…………………………………. 20-21**
   1. Introduction to Image Processing……………………………………………………. 20
   2. Dataset………………………………………………………………………………... 21

\*\*\* **Supportive Contents** [ Page 22] \*\*\*

**Chapter 1**

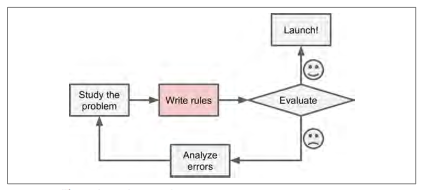
**Machine Learning**

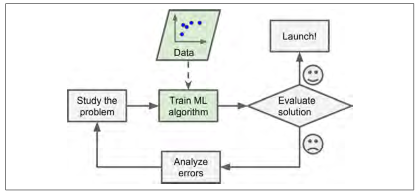
When most people hear “Machine Learning,” they picture a robot. But Machine Learning is not just a futuristic fantasy, it’s already here. In fact, it has been around for decades in some specialized applications, such as Optical Character Recognition (OCR). But the first ML application that really became mainstream, improving the lives of hundreds of millions of people, took over the world back in the 1990s: it was the spam filter.

**1.1 Introduction to Machine Learning**

Machine Learning is the science (and art) of programming computers so they can learn from data.

For example, your spam filter is a Machine Learning program that can learn to flag

spam given examples of spam emails (e.g., flagged by users) and examples of regular emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks.

**Fig 1.1.1** The traditional training approach **Fig 1.1.2** Machine Learning approach

In **fig 1.1.1** and **fig 1.1.2** we can see the approach for solving a problem in traditional way and machine learning way.

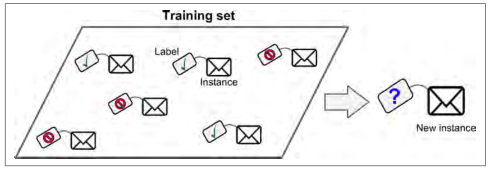
**1.2 Types of Machine Learning**

There are so many different types of Machine Learning(ML) systems that it is useful to

classify them in broad categories. Most common types of ML is supervised, unsupervised, semi supervised, reinforcement learning, batch learning, online learning, instance based learning, model based learning. We will learn the basics of supervised and unsupervised learning in this chapter.

**1.2.1 Supervised learning:**

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels (Fig 1.2.1).

**Fig 1.2.1** A labeled training set for supervised learning (e.g., spam classification)

A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails. Here are some of the most important supervised learning algorithms:

* k-Nearest Neighbors
* Linear Regression
* Logistic Regression
* Support Vector Machines (SVMs)
* Decision Trees and Random Forests
* Neural networks\*

Some neural network architectures can be unsupervised, such as autoencoders and restricted Boltzmann machines.

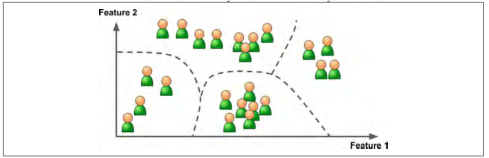
**1.2.2 Unsupervised learning**

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.

Here are some of the most important unsupervised learning algorithms:

* Clustering
  + k-Means
  + Hierarchical Cluster Analysis (HCA)
  + Expectation Maximization
* Visualization and dimensionality reduction
  + Principal Component Analysis (PCA)
  + Kernel PCA
  + Locally-Linear Embedding (LLE)
  + t-distributed Stochastic Neighbor Embedding (t-SNE)
* Association rule learning
  + Apriori
  + Eclat

For example, say you have a lot of data about your blog’s visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors (Figure 1.4). At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on. If you use a hierarchical clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.



**Fig 1.2.2** Clustering in unsupervised learning

**1.2.3 Batch and Online Learning:**

In **batch learning**, the system is incapable of learning incrementally: it must be trained

using all the available data. This will generally take a lot of time and computing

resources, so it is typically done offline. First the system is trained, and then it is

launched into production and runs without learning anymore; it just applies what it

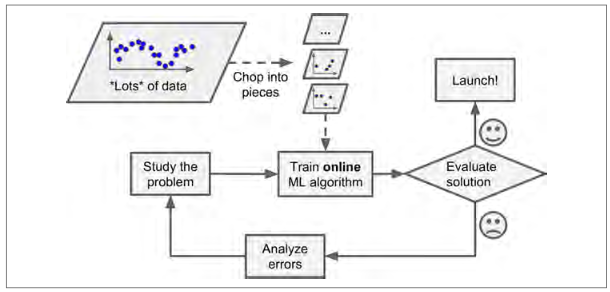
has learned. This is called offline learning.

In **online learning**, you train the system incrementally by feeding it data instances

sequentially, either individually or by small groups called mini-batches. Each learning

step is fast and cheap, so the system can learn about new data on the fly, as it arrives. Online learning is great for systems that receive data as a continuous flow (e.g., stock

prices) and need to adapt to change rapidly or autonomously.

**Fig 1.5** Using online learning to handle huge datasets

**1.2.4 Instance-Based Versus Model-Based Learning**

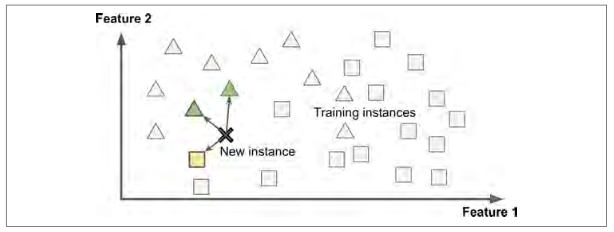
One more way to categorize Machine Learning systems is by how they generalize.

Most Machine Learning tasks are about making predictions. This means that given a

number of training examples, the system needs to be able to generalize to examples it

has never seen before.

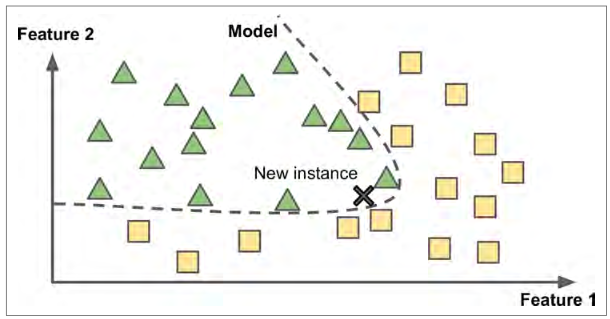
**Instance-based learning:** Possibly the most trivial form of learning is simply to learn by heart. If you were to create a spam filter this way, it would just flag all emails that are identical to emails that have already been flagged by users—not the worst solution, but certainly not the best.

Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails. This requires a measure of similarity between two emails. A (very basic) similarity measure between two emails could be to count the number of words they have in common. The system would flag an email as spam if it has many words in common with a known spam email. 

**Fig 1.2.4.1** Instance-based learning

This is called instance-based learning: the system learns the examples by heart, then generalizes to new cases using a similarity measure.

**Model-based learning**: Another way to generalize from a set of examples is to build a model of these examples, then use that model to make predictions. This is called model-based learning.

**Fig 1.2.4.2** Model-based Learning

For example, suppose you want to know if money makes people happy. You have the dataset in hand for better understanding please check the Example -lifesat.

After going through the example:

* You studied the data.
* You selected a model.
* You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
* Finally, you applied the model to make predictions on new cases (this is called

inference), hoping that this model will generalize well.

**1.3 Main Challenges of Machine Learning**

In short, since your main task is to select a learning algorithm and train it on some data, the two things that can go wrong are “bad algorithm” and “bad data.” Let’s start with examples of bad data.

**1.3.1 Insufficient Quantity of Training Data**

For a child to learn what an apple is, all it takes is for you to point to an apple and say “apple” (possibly repeating this procedure a few times).

Now the child is able to recognize apples in all sorts of colors and shapes. Genius. Machine Learning is not quite there yet; it takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples.

In a famous paper published in 2001, Microsoft researchers Michele Banko and Eric Brill showed that very different Machine Learning algorithms, including fairly simple ones, performed almost identically well on a complex problem of natural language disambiguation once they were given enough data.

The idea that data matters more than algorithms for complex problems was further

popularized by Peter Norvig et al. in a paper titled “[The Unreasonable Effectiveness](http://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/35179.pdf)

[of Data](http://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/35179.pdf)” published in 2009.

**1.3.2 Poor Quality Data**

If your training data is full of errors, outliers, and noise (e.g., due to poor quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well. It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that.

**1.3.3 Overfitting and Underfitting**

**Overfitting:** Say you are visiting a foreign country and the taxi driver rips you off. You might be tempted to say that all taxi drivers in that country are thieves. Overgeneralizing is something that we humans do all too often, and unfortunately machines can fall into the same trap if we are not careful. In Machine Learning this is called overfitting: it means that the model performs well on the training data, but it does not generalize

Well.



**Fig 1.3.3** Overfitting and Underfitting

**Underfitting:** Refers to a model that can neither model the training data nor generalize to new data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

**1.4 Testing and Validating**

The only way to know how well a model will generalize to new cases is to actually try it out on new cases. One way to do that is to put your model in production and monitor how well it performs. This works well, but if your model is horribly bad, your users will complain not the best idea.

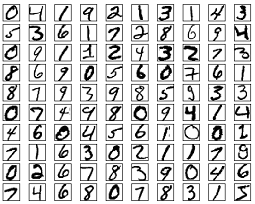
A better option is to split your data into two sets: the training set and the test set. As

these names imply, you train your model using the training set, and you test it using

the test set. The error rate on new cases is called the generalization error (or out of sample error), and by evaluating your model on the test set, you get an estimation of this error. This value tells you how well your model will perform on instances it has never seen before. If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

**1.5 MNIST Dataset**

Most examples are using MNIST dataset of handwritten digits. The dataset contains 60,000 examples for training and 10,000 examples for testing. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 1. For simplicity, each image has been flatten and converted to a 1-D numpy array of 784 features (28\*28). For visualization of data click [here](https://en.wikipedia.org/wiki/MNIST_database).



**Fig 1.5** : MNIST Dataset

**Chapter 2**

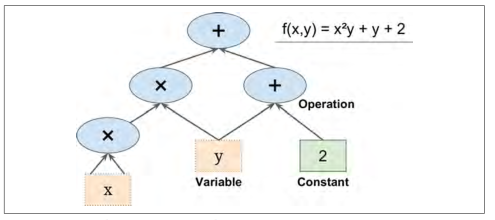
**Tensorflow**

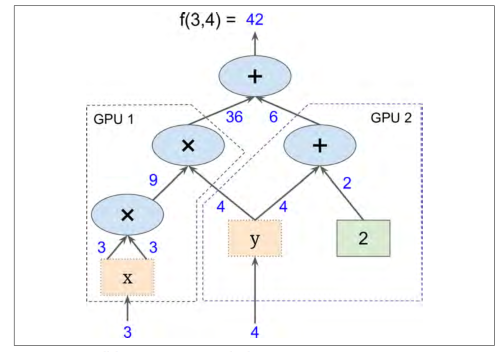
TensorFlow is a powerful open source software library. It was developed by the Google Brain team in 2012 for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

Examples are available in [Github](https://github.com/shakirul15-311/TensorFlow): [www.github.com/shakirul15-311/TensorFlow](http://www.github.com/shakirul15-311/TensorFlow)

**2.1 Introduction to Tensorflow**

TensorFlow is a symbolic math library, and is also used for machine learning applications such as neural networks. Its basic principle is simple: you first define in Python a graph of computations to perform (for example, the one in Fig 2.1), and then TensorFlow takes that graph and runs it efficiently using optimized C++ code.

**Fig 2.1** A simple computation graph

A simple computation graph. Most importantly, it is possible to break up the graph into several chunks and run them in parallel across multiple CPUs or GPUs( fig 2.2).

**Fig 2.2** Parallel computation on multiple CPUs/GPUs/servers

**2.2 Installation of Tensorflow**

Let’s started! At first install Python and Jupyter using the [Anaconda Distribution](https://www.anaconda.com/distribution/#download-section), which includes Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

**Download Anaconda:**

<https://www.anaconda.com/distribution/#download-section>

**Instead of Anaconda Use Google Colab online:**

[**https://colab.research.google.com**](https://colab.research.google.com/notebooks/welcome.ipynb#recent=true)/

**Next, install TensorFlow:**

$ pip install tensorflow

$ pip install --upgrade tensorflow

**2.3 Hello World with Tensorflow**

Hello world is like our first step in coding. Let’s start our journey in tensorflow with a simple example (Example 1)

>>import tensorflow as tf

>>a=tf.constant(3)

>>b=tf.constant(4)

>>c=tf.add(a,b)

>>session=tf.Session()

>>result=session.run(c)

>>print(result)

7

**2.4 Tensors, Variables and Placeholders**

Tensor is a special type of mathematical object which is used to analyze. Tensors define to multidimensional array. For better visualization in **multidimensional array, placeholder, variables** please have a look at notebook Example 2.

**Multidimensional array(Tensors):**

>>vector = tf.constant([3,4,5])

>>matrix = tf.constant([[1,2,3],[3,4,6],[9,5,2]])

>>with tf.Session() as session:

result=session.run(vector)

print('Vector:\n',result)

result=session.run(matrix)

print('Matrix:\n',result)

Output:

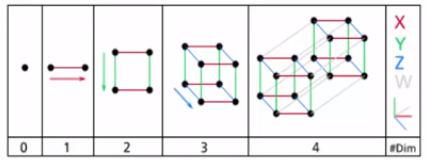
Vector:

[3 4 5]

Matrix:

[[1 2 3]

[3 4 6]

[9 5 2]]

**Fig 2.4** Multidimensional array

**Variables(**[**learn more**](https://www.tensorflow.org/guide/variables)**):**

>>x = tf.Variable(3)

>>y = tf.Variable(4)

>>sess = tf.Session()

>>sess.run(x.initializer)

>>sess.run(y.initializer)

>>result = sess.run(f)

>>print(result)

**2.5 Creating Your First Graph and Running It in a Session**

The following code creates the graph represented in Figure 2.2 (Example 3)

#Import tensorflow

>>import tensorflow as tf

#Declare variable

>>x = tf.Variable(3)

>>y = tf.Variable(4)

>>f = x\*x\*y + y + 2

Session takes care of placing the operations onto devices such as CPUs and GPUs and running them, and it holds all the variable values

>>sess = tf.Session()

>>sess.run(x.initializer)

>>sess.run(y.initializer)

>>result = sess.run(f)

>>print(result)

42

>>sess.close()

**2.6 Lifecycle of a Node Value**

When you evaluate a node, TensorFlow automatically determines the set of nodes

that it depends on and it evaluates these nodes first.(Example 4)

>>w = tf.constant(3)

>>x = w + 2

>>y = x + 5

>>z = x \* 3

>>with tf.Session() as sess:

print(y.eval()) # 10

print(z.eval()) # 15

Starts a session and runs the graph to evaluate y: TensorFlow automatically detects that y depends on w, which depends on x, so it first evaluates w, then x, then y, and returns the value of y. Finally, the code runs the graph to evaluate z. Once again, TensorFlow detects that it must first evaluate w and x. It is important to note that it will not reuse the result of the previous evaluation of w and x. In short, the preceding code evaluates w and x twice.

**2.7 Linear Regression**

TensorFlow operations (also called ops for short) can take any number of inputs and

produce any number of outputs. The inputs and outputs are multidimensional arrays, called tensors (hence the name “tensor flow”). In the Python API tensors are simply represented by NumPy ndarrays. They typically contain floats, but you can also use them to carry strings.

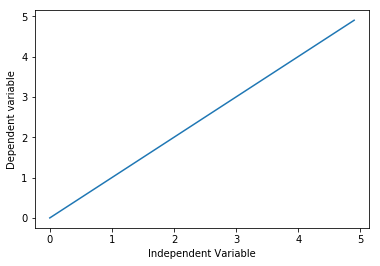
Linear regression is a simple technique to describing the relationship between two or more variables. Simple linear model equation:

Y=aX+b

For better understanding have a look to the Example 5.

>>X=np.arange(0.0, 5.0, 0.1) # 0.0 to 5.0 difference is 0.1

>>print(X)



>>a=1

>>b=0

>>Y=a\*X+b

**Fig 2.7** Linear regression

>>plt.plot(X,Y)

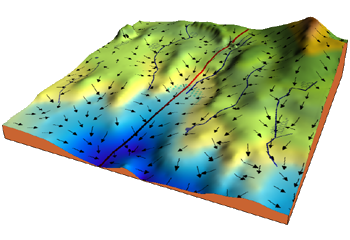
>>plt.xlabel('Independent Variable')

>>plt.ylabel('Dependent variable')

>>plt.plot()

**2.8 Gradient Descent**

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. Watch this [video](https://www.youtube.com/watch?v=nhqo0u1a6fw) for better understandings.

Consider the 3-dimensional graph below in the context of a cost function. Our goal is to move from the mountain in the top right corner (high cost) to the dark blue sea in the bottom left (low cost). The arrows represent the direction of steepest descent (negative gradient) from any given point–the direction that decreases the cost function as quickly as possible. (Example- Gradient Descent)

**Fig 2.8** 3D Visualization of Gradient Descent

**Learning rate:** The size of these steps is called the learning rate. With a high learning rate we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing. With a very low learning rate, we can confidently move in the direction of the negative gradient since we are recalculating it so frequently. A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.

**Cost function:** A Loss Functions tells us “how good” our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

**2.9 Model Optimization**

The Tensorflow Model Optimization Toolkit minimizes the complexity of optimizing inference. Inference efficiency is a critical issue when deploying machine learning models to mobile devices because of the model size, latency, and power consumption.

Among many uses, the toolkit supports techniques used to:

* Reduce latency and inference cost for cloud and edge devices (e.g. mobile, IoT).
* Deploy models to edge devices with restrictions on processing, memory, power-consumption, network usage, and model storage space.
* Enable execution on and optimize for existing hardware or new special purpose accelerators.

There are a list of optimizer function available in [Tensorflow documentation](https://www.tensorflow.org/api_docs/python/tf/train) a few of them listed below:

>>tf.train.RMSPropOptimizer(0.02).minimize(objective)

>>tf.train.GradientDescentOptimizer(0.002).minimize(objective)

>>tf.train.AdamOptimizer(0.3).minimize(objective)

>>tf.train.MomentumOptimizer(0.002, 0.9).minimize(objective)

>>tf.train.AdadeltaOptimizer(0.1).minimize(objective)

>>tf.train.AdagradOptimizer(0.1).minimize(objective)

**Example:**

Define an optimizer using tensorflow (Example- Optimization):

**>>optim = tf.train.RMSPropOptimizer(learning\_rate=0.01)**

**>>training\_op = optim.minimize(-J)**

**2.10 Tensorboard**

The computations in TensorFlow for - like training a massive deep neural network can be complex and confusing. To make it easier to understand, debug, and optimize TensorFlow programs, a suite of visualization tools called TensorBoard.

**Chapter 3**

**Neural Network**

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.

**3.1 Convolutional Neural Network**

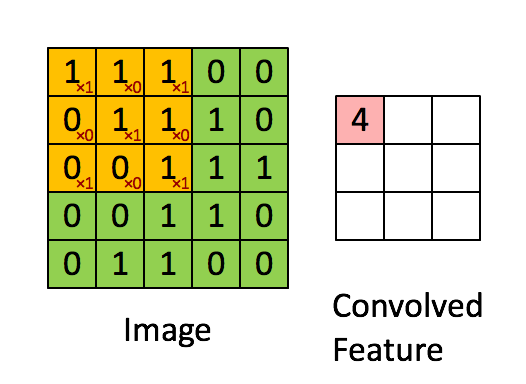
A Convolutional Neural Network ([ConvNet/CNN](https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/neural_network_raw.ipynb)) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. An example

Will be shown on next chapter. ( Example:Image classification problem using cnn )

**Fig 3.1** : CNN structure used for digit recognition

**3.2 Convolution Layer — The Kernel**

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB). This is a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature



**Fig 3.2.1**: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

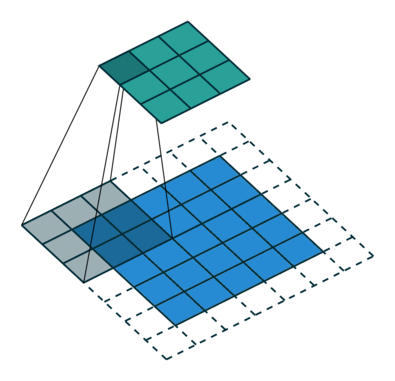
Kernel/Filter, K =

1 0 1

0 1 0

1 0 1

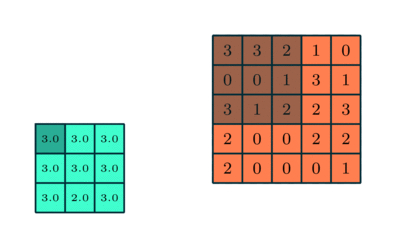
The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

**Fig 3.2.2**: Convolution Operation with Stride Length = 2 

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. The first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

**3.3 Pooling Layer**

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature.

**Fig 3.3:** 3x3 pooling over 5x5 convolved feature

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling return the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

**Chapter 4**

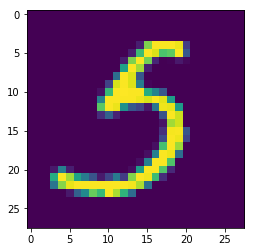
**End-to-End Machine Learning Project**

In this chapter, you will go through example projects end to end, get the idea, discover the algorithm, prepare the model and dataset, training and validation etc.

**4.1: Intro to Image Processing**

Image processing is one of the most interesting part of machine learning. Visualizing image after processing is seems the best part. (Example: Image processing).

MNIST dataset classification is shown in this example.



**Fig 4.1**: A plot image of MNIST dataset

Another Notebook example just explained in example image classification problem using cnn.

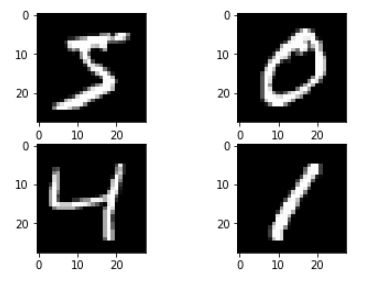
A few examples on MNIST Fashion data also discussed on a notebook. For better understanding complete the Tensorflow course from Coursera. (Example Fashion Classification)

**Courses:**

1. Introduction to TensorFlow for AI, Machine Learning, and Deep Learning( [Coursera](https://www.coursera.org/learn/introduction-tensorflow/home/welcome))
2. Convolutional Neural Networks in TensorFlow ([Coursera](https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/home/welcome))

**4.2: Dataset**

Data is the main power/asset of data science without data there is nothing we can do. By fetching data we can predict something, extract information , solving real life problems etc.



**Fig 4.2**: Sample of handwritten digit MNIST dataset

**Data repositories:**

There are many data repositories available online for free, most popular open data repositories:

— [UC Irvine Machine Learning Repository](http://archive.ics.uci.edu/ml/index.php)

— [Kaggle datasets](https://www.kaggle.com/datasets)

— [Amazon’s AWS datasets](https://registry.opendata.aws/)

**\*\*\*Supportive Contents\*\*\***

**Prerequisite:** Before starting with Tensorflow learners must have to know basic python, machine learning and neural network

**Tools/IDE:**

1. Jupyter Notebook([Anaconda](https://www.anaconda.com/distribution/#download-section))
2. [Colab](https://colab.research.google.com) by Google

**Book:** Hands-on Machine Learning with Scikit-Learn and TensorFlow By Aurélien Géron. Example codes of book in [git](https://github.com/ageron/handson-ml)

**Examples** are available in [Github](https://github.com/shakirul15-311/TensorFlow): [www.github.com/shakirul15-311/TensorFlow](http://www.github.com/shakirul15-311/TensorFlow)

**Online Courses:**

1. Introduction to TensorFlow for AI, Machine Learning, and Deep Learning( [Coursera](https://www.coursera.org/learn/introduction-tensorflow/home/welcome))
2. Convolutional Neural Networks in TensorFlow ([Coursera](https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/home/welcome))
3. Practical Machine Learning Tutorial with Python Introduction ([pythonprogramming.net](https://pythonprogramming.net/machine-learning-tutorial-python-introduction/))
4. [https://www.tensorflow.org/tutorials](https://www.tensorflow.org/tutorials/)

**Notebook Examples:**

1. Handwriting classification mnist([link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Exercises/Exercise%203%20-%20Convolutions/Exercise%203%20-%20Answer.ipynb#scrollTo=22hBZbxx98IS))
2. Fashion Recognition([link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%204%20-%20Lesson%202%20-%20Notebook.ipynb))
3. Improving Fashion recognition Accuracy using Convolutions([link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%206%20-%20Lesson%202%20-%20Notebook.ipynb))
4. Play with some convolutions([link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%206%20-%20Lesson%203%20-%20Notebook.ipynb))
5. Human or Horse classification([link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%208%20-%20Lesson%203%20-%20Notebook.ipynb#scrollTo=RXZT2UsyIVe_))
6. Happy or Sad([Link](https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Exercises/Exercise%204%20-%20Handling%20Complex%20Images/Exercise4-Answer.ipynb))

**Data repositories:**

Popular open data repositories

— [UC Irvine Machine Learning Repository](http://archive.ics.uci.edu/ml/index.php)

— [Kaggle datasets](https://www.kaggle.com/datasets)

— [Amazon’s AWS datasets](https://registry.opendata.aws/)

Meta portals (they list open data repositories):

— <http://dataportals.org>/

— <http://opendatamonitor.eu/>

— <http://quandl.com/>

Other pages listing many popular open data repositories:

—[Wikipedia’s list](https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research) of Machine Learning datasets

— [Quora.com](https://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public) question

— [Datasets](https://www.reddit.com/r/datasets) subreddit