|  |  |
| --- | --- |
| NAME | MAHESWARAN.S |
| REG.NO. | 420121104031 |
| DEPARTMENT | CSE |
| YEAR | III |
| COLLEGE NAME | AKT MCET |
| GROUP | IBM GROUP-5 |

**MEASURE ENERGY CONSUMPTION**

PHASE-3

INNOVATION

. In this phase you need to put your design into innovation to solve the problem.

.Explain detail the complete steps that will be taken by you to put your design that you thought of in previous phase in to transformation.

.Create a document around it and share the same for assessment.

Module 5: Tensor flow & keas -ANN

* Introduction to tensor flow

Tensor flow is an end-to-end open source platform for machine learning. Tensor flow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular tensor flow API to develop and train machine learning models.

## **Difference between ft. Placeholder and ft. Variable**

| **ft. placeholder** | **ft. Variable** |
| --- | --- |
| It is an empty variable in which data is fed at a later part of the code. | The variable once declared must be initialized with an initial value at the time of declaration. |
| Placeholders are bound inside expressions. | Variables are used to hold the value of weights and biases. |
| The values are specific and are not changed during the execution of the program. | The values are changed during the execution of the program. |
| It is used to handle external data. | It is used to store the values that will be required throughout the program. |

## **Ft. Placeholder**

As the name suggests, it is an empty place. It is an empty variable to which the training data is fed later. The ft. Placeholder allows us to create the structure first which is setting up of computational graphs and then feeding the data into it. It allows us to put the data during runtime. As the session starts, we feed the data into the placeholders.

***Syntax:*** *tf.compat.v1.placeholder (type, shape=none, name=none)*

* *type is the datatype of the input*
* *Shape is the tensor shape. However, it is an optional parameter.*
* *name is the operation name*

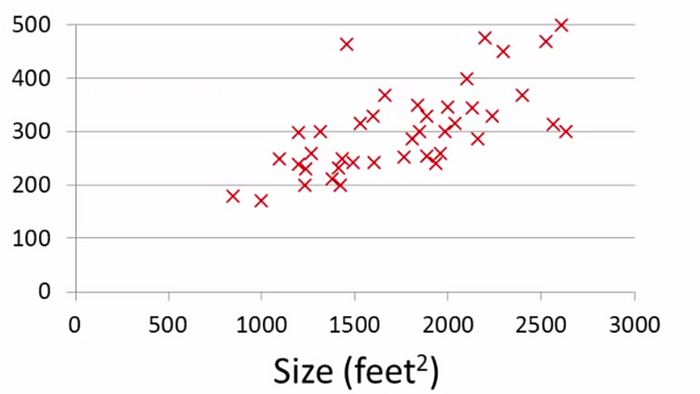
## **Ft. Variable**

Ft. Variable is a state that holds an initial value. The values are nothing but tensors. The variables can be added to the computational graph by calling the constructors. Whenever a variable is created, it is always initialized. They basically hold weights and biases during the session execution.

***Syntax:***

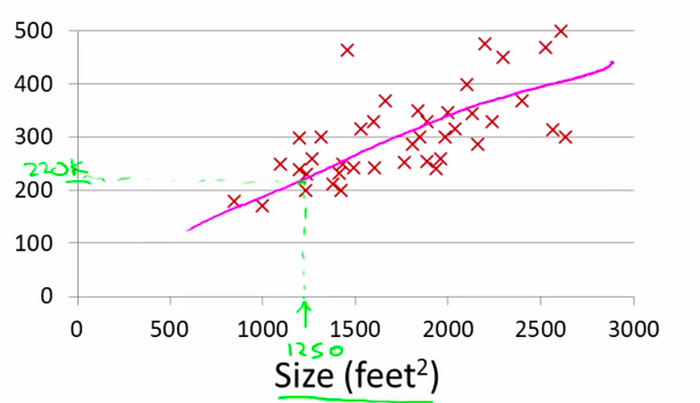
*Ft. Variable (initial value=none,  
   trainable=none,  
   validate shape=True,  
   caching device=none,  
   name=none,  
   variable den=none,  
   type=none,  
   import scope=none,  
   constraint=none,  
   synchronization=if variable synchronization. AUTO,  
   aggregation=tf.compat.v1.VariableAggregation.NONE,  
   shape=none)*

**Linear Regression**



To the left, it’s the plot of the ***size vs the price***from theboson house pricing dataset***.***Given this dataset we need to find a relationship between size and the house price so that we could suggest a fair price for a new house to be sold given its size.

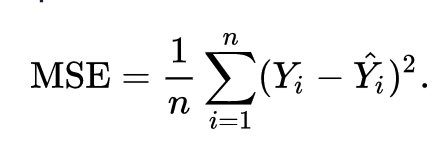
Linear Regression is a **Linear Model**. Which means, we will establish a linear relationship between the input variables(**X**) and single output variable(**Y**). When the input(**X**) is a single variable this model is called **Simple Linear Regression** and when there are multiple input variables(**X**), it is called **Multiple Linear Regression**.



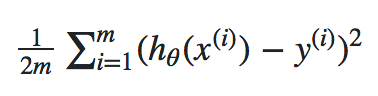
This is called supervised learning, we take the set of **right answers**, find a pattern in it and then use it to make predictions. Since we are predicting the house price, which is a real and continuous valued output, the prediction problem is called as regression.

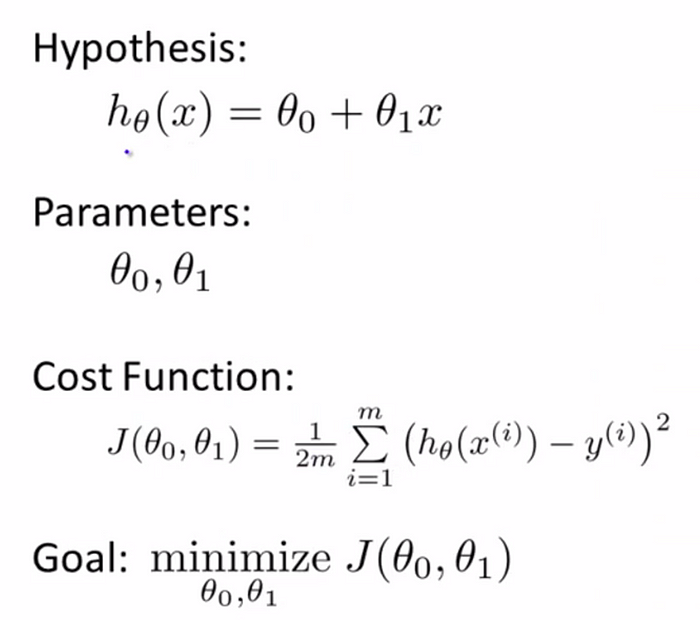
## **The cost function**

One common function that is often used is [mean squared error](https://en.wikipedia.org/wiki/Mean_squared_error), which measure the difference between the actual value from the dataset and the estimated value (the prediction). It looks like this:



We can [adjust the equation a little](https://datascience.stackexchange.com/questions/10188/why-do-cost-functions-use-the-square-error) to make the calculation little simpler.





***Module: 6 Convolutional Neural Networks***

# 

# Introduction to Convolution Neural Network

# A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

# Convolution and Feature Learning

He Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent

# MNIST Dataset

### **Context**

MNIST is a subset of a larger set available from NIST (it's copied from <http://yann.lecun.com/exdb/mnist/>)

### **Content**

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. .  
Four files are available:

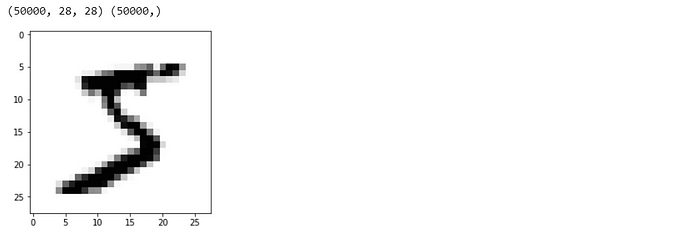
* train-images-idx3-ubyte.gz: training set images (9912422 bytes)
* train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
* t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
* t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

# Multi-Layer perceptron using tensor flow

In this blog, we are going to build a neural network (multilayer perceptron) using tensor flow and successfully train it to recognize digits in the image. Tensor flow is a very popular deep learning framework released by, and this notebook will guide to build a neural network with this library. If you want to understand what a Multi-layer perceptron is, you can look at my [previous blog](https://medium.com/@aayushmnit/building-neural-network-from-scratch-9c88535bf8e9) where I built a Multi-layer perceptron from scratch using Jumpy.

Let’s start by importing our data. As Kara’s, a high-level deep learning library already has MNIST data as part of their default data we are just going to import the dataset from there and split it into train and test set.

*## Loading MNIST dataset from keas*  
**import** **keas**  
**from** **sklearn.preprocessing** **import** LabelBinarizer  
**import** **matplotlib.pyplot** **as** **plot**  
%matplotlib inline**def** load dataset (flatten=**False**):  
 (X train, y train), (X test, y test) = keras.datasets.mnist.load\_data () *# normalize x*  
 X\_train = X\_train.astype (float) / 255.  
 Test = X\_test.astype (float) / 255. *# we reserve the last 10000 training examples for validation*  
 X\_train, Val = X\_train [:-10000], X\_train [-10000:]  
 y\_train, yawl = y\_train [:-10000], y\_train [-10000:] **if** flatten:  
 X\_train = X\_train. Reshape ([X\_train.shape [0], -1])  
 X\_val = X\_val.reshape ([X\_val.shape [0], -1])  
 X\_test = X\_test.reshape ([X\_test. Shape [0], -1]) **return** X\_train, y\_train, X\_val, yawl, X\_test, y\_testX\_train, y\_train, X\_val, yawl, X\_test, test = load dataset ()   
*## Printing dimensions*  
print (X\_train.shape, y\_train. Shape)   
*## Visualizing the first digit*  
Plt.imshow (X\_train [0], camp="Greys");



# Training of CNN in Tensor Flow

The MNIST database (**Modified National Institute of Standard Technology database**) is an extensive database of handwritten digits, which is used for training various image processing systems. It was created by "**reintegrating**" samples from the original dataset of the **MNIST**.

If we are familiar with the building blocks of Connects, we are ready to build one with Tensor Flow. We use the MNIST dataset for image classification.

Preparing the data is the same as in the previous tutorial. We can run code and jump directly into the architecture of CNN.

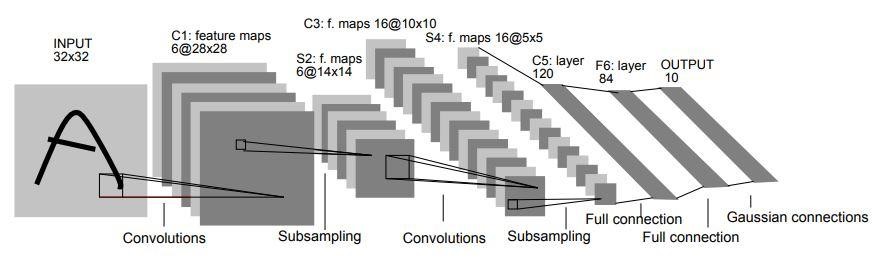
Here, we are executing our code in **Google Ecolab** (an online editor of machine learning).

## **Basic Architecture**

* Contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.



## **LeNet-5 CNN Architecture**



**Module 7: Open CV**

**Introduction to Open CV**

At viso.ai, we’ve integrated Open CV into Visa Suite, the end-to-end Computer Vision Platform. Leading organizations use it to build, deploy and scale real-world computer vision applications. We are also an AI software partner of Intel, the creator of Open CV. Get a demo here.

**What is Open CV?**

Initially developed by Intel, Open CV (Open Source Computer Vision) is a free cross-platform computer vision library for real-time image processing. The Open CV software has become a de-facto standard tool for all things related to Computer Vision. In 2023, Open CV is still highly popular, with over 29’000 downloads every week.

Open CV is written in C and C++. It runs under the most popular operating systems, such as GNU/Linux, OS X, Windows, Android, iOS, etc. It is available for free under the Apache 2 license. There is active development on interfaces for Python, Ruby, Mat lab, and other languages. The Open CV library contains over 2500 algorithms, extensive documentation, and sample codeforreal-timecomputervision.

**What is Open CV used for?**

Open CV was built for maximum efficiency and performance of computing-intensive vision tasks. Therefore, it has a strong focus on real-time applications of AI vision. The software is written in optimized C and is able to take advantage of multicore processors (multi-threading).

The goal of Open CV is to provide an easy-to-use computer vision infrastructure that helps people build sophisticated vision applications quickly by providing over 500 functions that span many areas in vision. Open CV is often used in factory product inspection, medical imaging, security analysis, human-machine interface, camera calibration, stereo vision (3D vision), and roboticvision.

**Who uses Open CV?**

Open CV is used by big enterprises and government institutions, for example, Google, Toyota, IBM, Microsoft, SONY, Siemens, and Facebook. Also, well-known computer vision stratus use Open CV to build powerful computer vision products and AI solutions – including viso.ai. Many research centre’s use Open CV, such as Stanford, MIT, INRIA, Cambridge, and CMU.

The use cases of computer vision are vast. While most are aware of rather popular use cases in security and video surveillance or self-driving cars, fewer people get to see the use cases in specific industries such as industrial manufacturing, restaurants, or retail analytics.

**Can I use Open CV for commercial projects?**

Yes. The open-source license for Open CV was chosen so that developers can build commercial products using all or part of Open CV. Therefore, companies are under no obligation to open source their product or to return improvements to the public domain (copy left), though many do and contribute to the large community.

**Downloading and Installing Open CV**

Since 2012, support for Open CV was taken over by a non-profit foundation named OpenCV.org, which maintains a developer site and a user website. You can get the official releases from Source Forge or take the latest sources from the open GitHub.

How to install Open CV on Windows: To install Open CV, download the executable installation and run it. It will install Open CV and register Direct Show filters .After, Open CV can be used.

**Applications built with Open CV**

Most people are aware of computer vision being widely popular in security and surveillance. Some people are aware of niche applications in safety monitoring, unmanned flying vehicles (drones), or biomedical analysis. But few are aware of how pervasive machine vision has become in industrial manufacturing – virtually everything that is mass-produced has been automatically inspected at some point using computer vision.

**Use cases built with Open CV**

Since Open CV is a development kit, there are countless use cases you can build with Open CV, including:

* 1. Recognition of objects for counting and object tracking.

2. Analysing medical images to support human diagnosis (Medical Imaging).

3. Recognition of advertisements in TV footage or logo recognition with AI vision.

4. Player tracking in sports and fitness, scene recognition, and execution quality analysis.

5. Counting the number of people in public places such as airports (Crowd analysis).

6. Robotic automation for intelligent, vision-based interfaces (Intelligent Screens).

7. Automatic inspection and video analysis with always-on computer vision (for example, at airports in aviation).

8. Image search on digital platforms, in web-based applications.

9. Detecting defects or faults during manufacturing processes (Smart Factory) Counting the number of vehicles on a highway (Traffic Analysis)

10. CCTV camera applications to detect physical violence, attacks.  