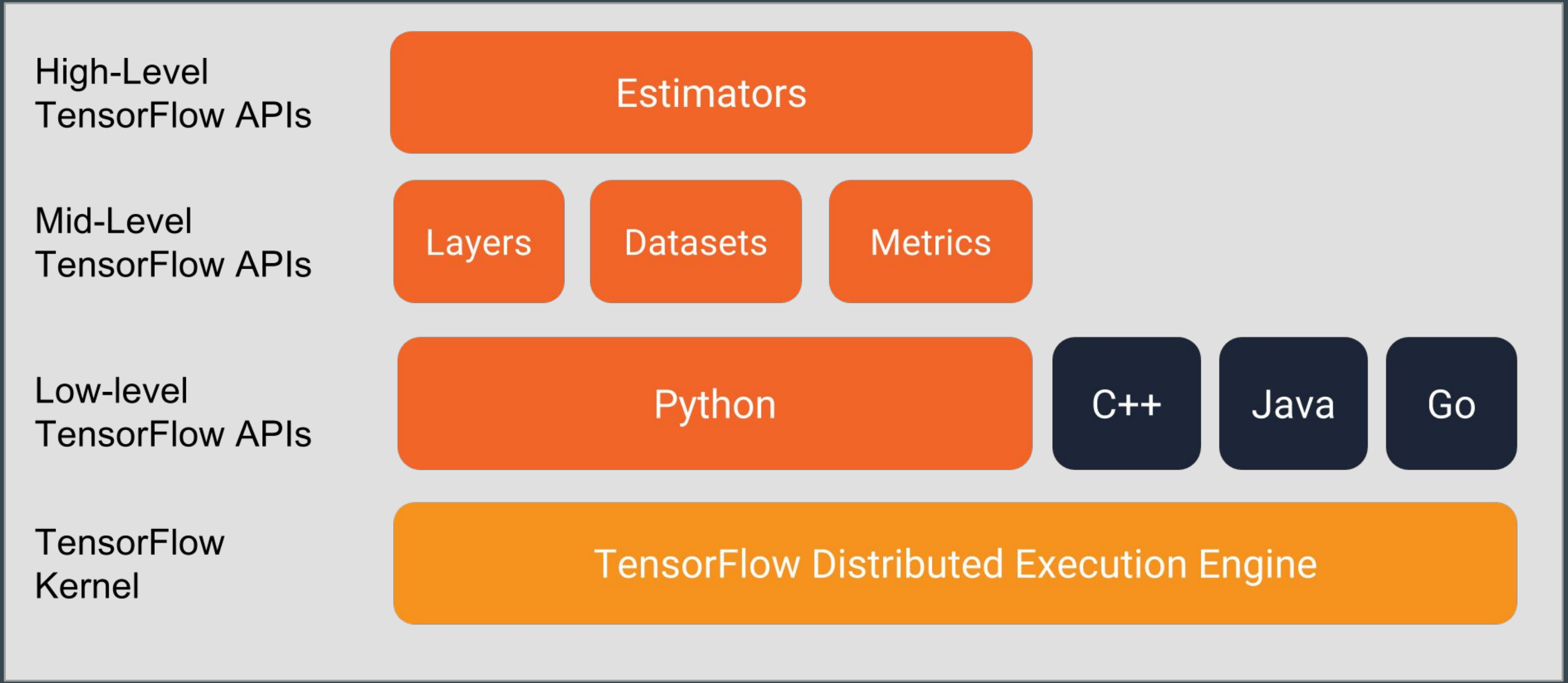
Implementing Neural Network Structures

...

The **Tensorflow** Way

IWPAA 2018

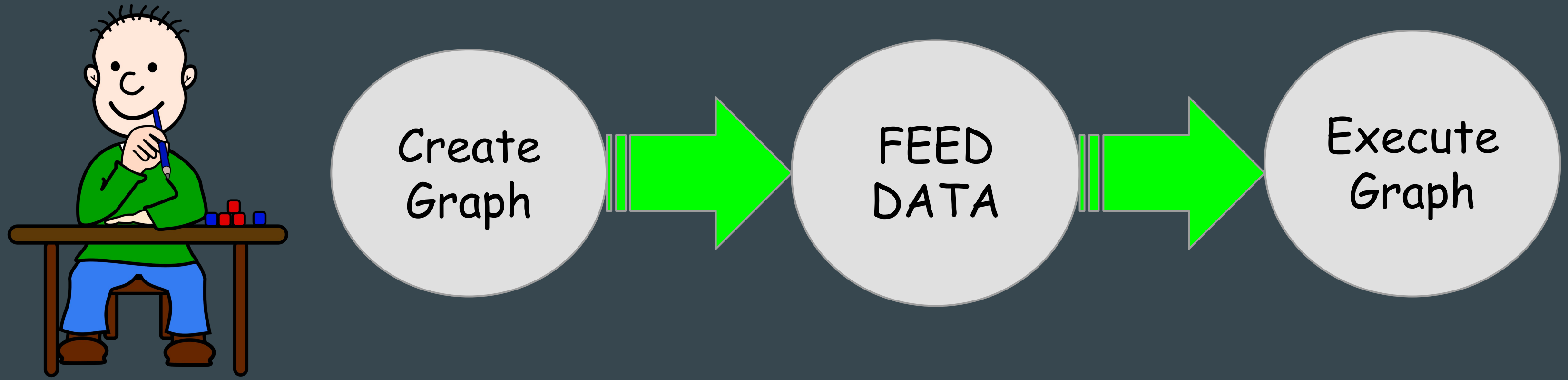
Why Tensorflow ?



What lies ahead ?

- **Beginners**
    a. Understanding Tensorflow Execution Principle
    b. Doing simple arithmetic operations using Tensorflow
    c. Creating our first Neural Network program - **MLP**
- **Advanced**
    a. Making our NN "DEEPER" - **CNN**
    b. Giving our NN "MEMORY" - **RNN**
    c. Bridging CNN and RNN
    d. Customizing our Network

Understanding Tensorflow - Computation Graph



HAVE AN IDEA ?

Understanding Tensorflow - Computation Graph

Example: **Simple Mathematical Operations**

Evaluate $Z = (X + Y) - (X * Y)$

Traditionally

Operations (Operators): + , - , * , =

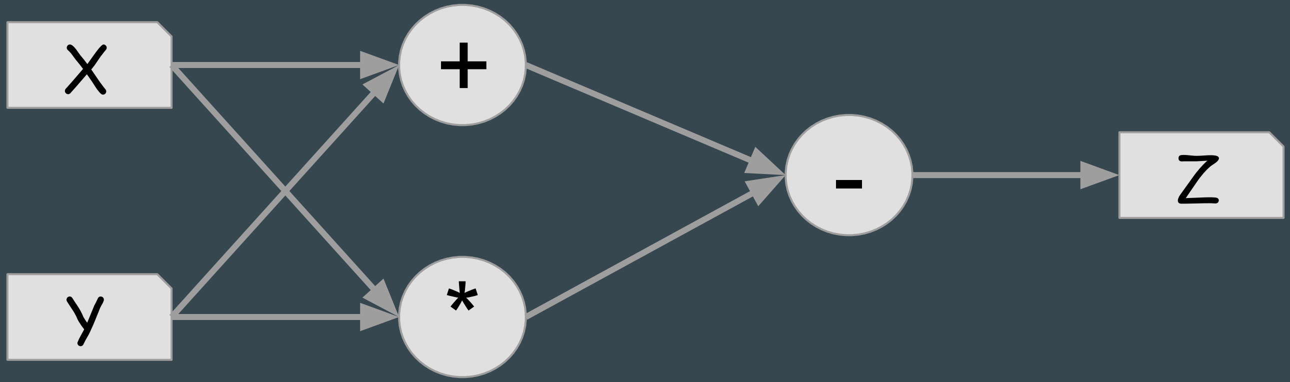
Values (Variables) : X , Y , Z

Understanding Tensorflow - Computation Graph

- Values are stored in
  - VARIABLE
  - CONSTANT
  - PLACEHOLDER
- Operators are <u>**Nodes of Graph**</u>

Understanding Tensorflow - Computation Graph

- Z=(X+Y)-(X*Y)



Understanding Tensorflow - Computation Graph

```
import tensorflow as tf # import TF library

Yourgraph = tf.Graph()  # create an empty graph
```

Understanding Tensorflow - Computation Graph

```
import tensorflow as tf # import TF library
Yourgraph = tf.Graph()  # create an empty graph
with yourgraph.as_default():
   x = tf.placeholder(tf.float32,shape=[None])
   y = tf.placeholder(tf.float32, shape=[None])
```

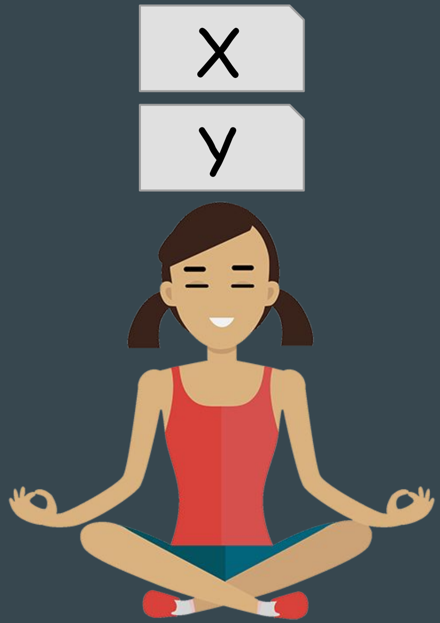
Understanding Tensorflow - Computation Graph

Placeholders

- Used to store External Values (given by user)
- Not needed to INITIALIZE

Variables

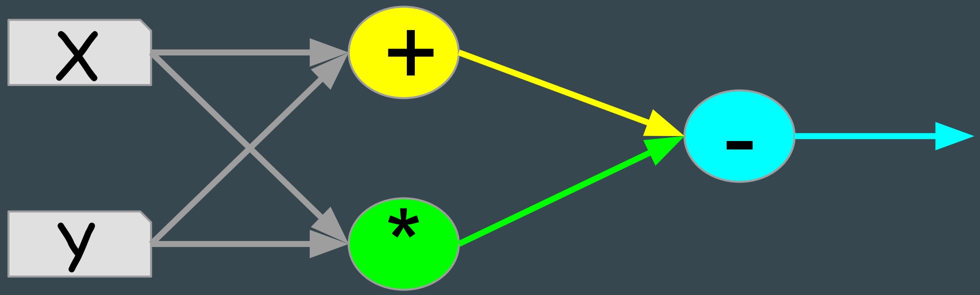
- Used to store internal state of the GRAPH
- Must be INITIALIZED



PLACEHOLDER it is ...

Understanding Tensorflow - Computation Graph

```
import tensorflow as tf # import TF library
yourgraph = tf.Graph()  # create an empty graph
With yourgraph.as_default()
      x = tf.placeholder(tf.float32,shape=[None])
      y = tf.placeholder(tf.float32, shape=[None])
      node_plus = tf.add(x,y)
      node_multiply = tf.mul(x,y)
      node_minus = tf.sub(node_plus,node_multiply)
```



Understanding Tensorflow - Computation Graph

- Graph creation complete
- Prepare the DATA now

Understanding Tensorflow - Prepare DATA

```
import numpy as np
x_data=np.random.randint(0,high=50,size=[3])
y_data=np.random.randint(0,high=20,size=[3])
```

Understanding Tensorflow - Computation Graph

- Graph creation complete
- DATA is Ready
- FEED DATA to Graph
- Execute Graph

Understanding Tensorflow - Execute Graph

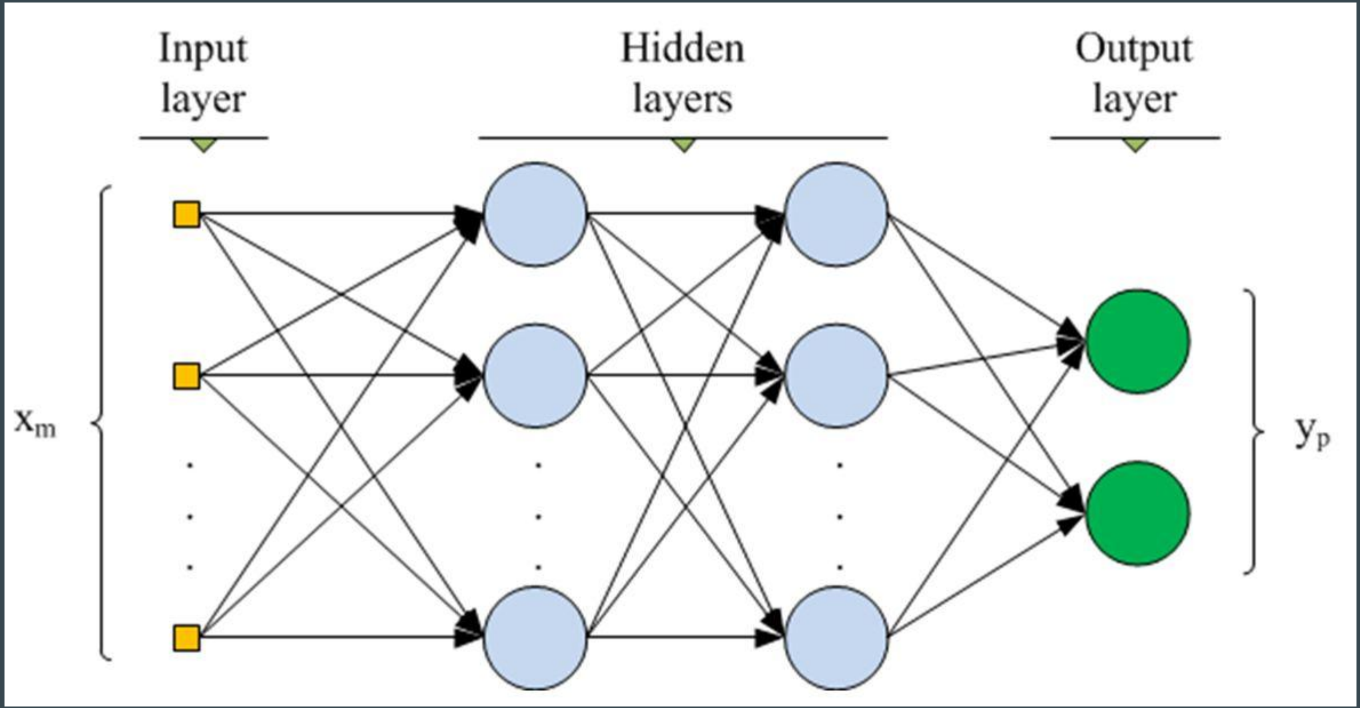
```
import tensorflow as tf # import TF library
yourgraph = tf.Graph()  # create an empty graph
import numpy as np

With yourgraph.as_default()
x = tf.placeholder(tf.float32,shape=[None])
y = tf.placeholder(tf.float32, shape=[None])
      node_plus = tf.add(x,y)
      node_multiply = tf.mul(x,y)
      node_minus = tf.sub(node_multiply,node_plus)
x_data=np.random.randint(0,high=50,size=[3])
y_data=np.random.randint(0,high=20,size=[3])
with tf.Session(graph=yourgraph) as s:
   feed={x:x_data,y:y_data}
   out=s.run([node_minus],feed_dict=feed)
```

Understanding Tensorflow - Executed !

Tensorflow - First Neural Network (MLP)

Tensorflow - First Neural Network (MLP)



Tensorflow - A Simple Classification Problem

Soybean Disease Dataset
by
Michalski, R. S.

Number of features 35
Number of classes 19

https://archive.ics.uci.edu/ml/machine-learning-databases/soybean/soybean-large.names

HAVE ANY IDEA ?

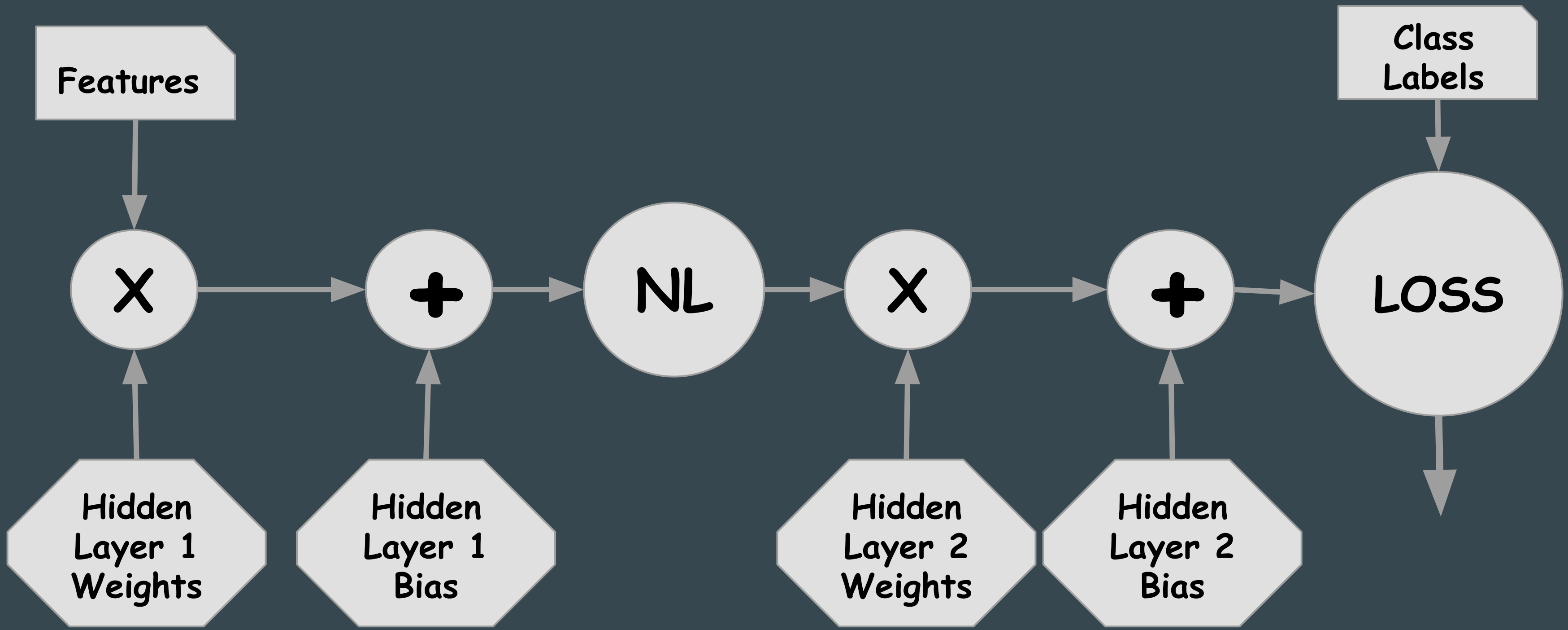
Tensorflow - First Neural Network (MLP)

Number of Features / Nodes in Input Layer - **35**
Number of Nodes in Layer 1 - 16
Number of Nodes in Layer 2 - 24
Number of Classes / Nodes in Output Layer - **4**

Loss - Cross Entropy
Optimizer - Stochastic Gradient Descent

HAVE ANY IDEA ?
YES !!

Tensorflow - MLP Computation Graph



Tensorflow - MLP Computation Graph

```
mlpgraph=tf.Graph()

with mlpgraph.as_default():

    x=tf.placeholder(tf.float32,[None,nb_features])

    y=tf.placeholder(tf.float32,[None,nb_classes])
```

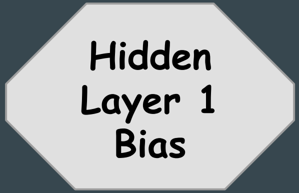
Features

Class Labels

Tensorflow - MLP Computation Graph

```
w12=tf.Variable(tf.truncated_normal([nb_features,16],stddev=0.1),
```


```
b1=tf.constant(0.1,shape=[16])
```




```
input_h1=tf.matmul(x,w12)
```


```
input_h1_nl=tf.tanh(input_h1+b1)
```

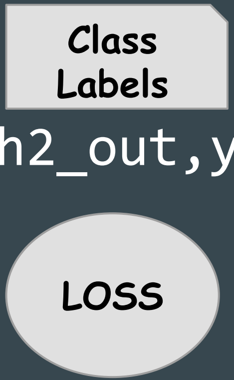
Tensorflow - MLP Computation Graph

```
w12=tf.Variable(tf.truncated_normal([nb_features,16],stddev=0.1),
b1=tf.constant(0.1,shape=[16])
input_h1=tf.matmul(x,w12)
input_h1_nl=tf.tanh(input_h1+b1)
```

```
Continue for more layers...
```

Tensorflow - MLP Computation Graph

Finally 

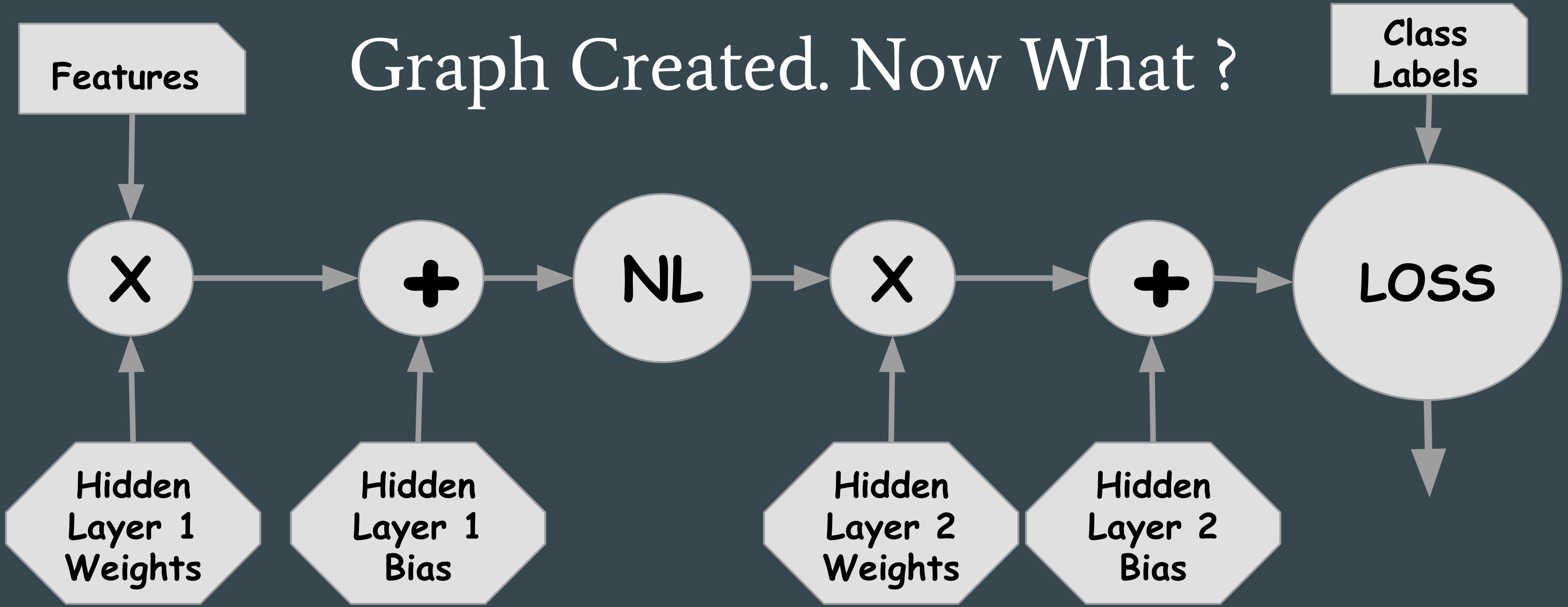



```
loss=tf.nn.softmax_cross_entropy_with_logits(h2_out,y)
loss=tf.reduce_mean(loss)
```

```
optimizer=tf.train.GradientDescentOptimizer(0.001).minimize(loss)
```

Tensorflow - MLP Computation Graph

Graph Created. Now What ?



Tensorflow - MLP Computation Graph (Execution)

Graph Created. Now What ?

- Prepare a dictionary of SOYBEAN DATA with Placeholders as **Keys.**
- Initialize Graph Variables
- Run a **Session** with Dictionary and Computation Graph.
- Collect the Results (Node Outputs) in some **Array**

Tensorflow - MLP Computation Graph (Execution)

Dictionary of SOYBEAN DATA with Placeholders as Keys.

```
x_train,y_train=load_data(data file)

feed={x:x_train,y:y_train}
```

Features

Class Labels

Tensorflow - MLP Computation Graph (Execution)

Initialize Graph Variables

```
with tf.Session(graph=yourgraph) as yoursession:
    init=tf.global_variables_initializer()
    yoursession.run([init])#Initialize all Variables
```

Tensorflow - MLP Computation Graph (Execution)

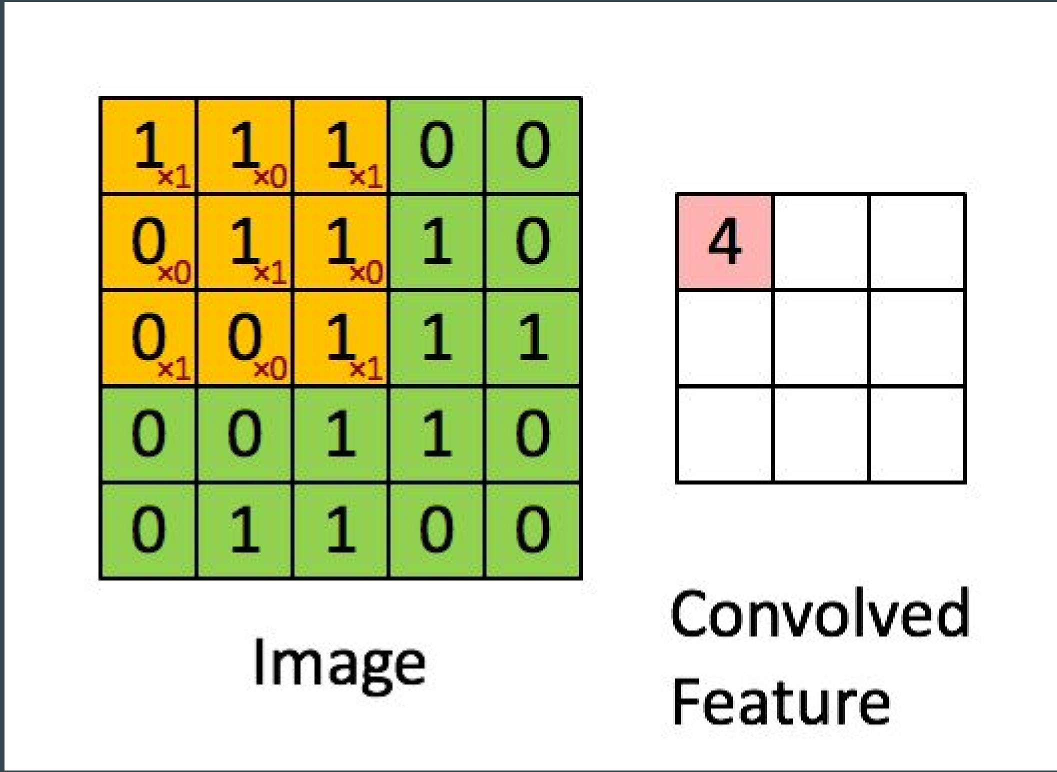
Run a Session with Dictionary and Computation Graph

```
with tf.Session(graph=yourgraph) as yoursession:
    init=tf.global_variables_initializer()
    yoursession.run([init])#Initialize all Variables
_,L=yoursession.run([optimizer,loss],feed_dict=feed)
```

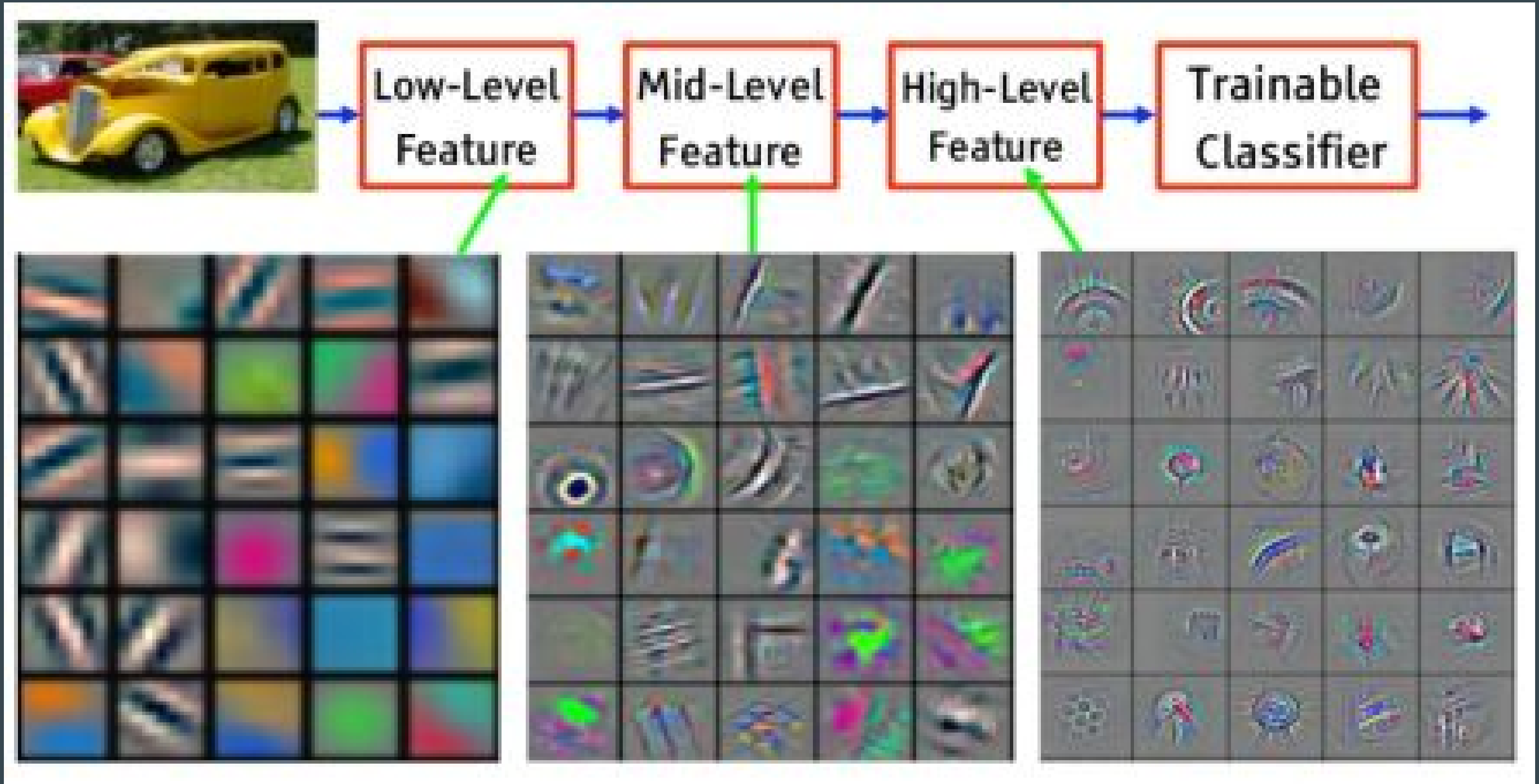
L Contains the **loss** associated with **feed**

Tensorflow - MLP Implemented !

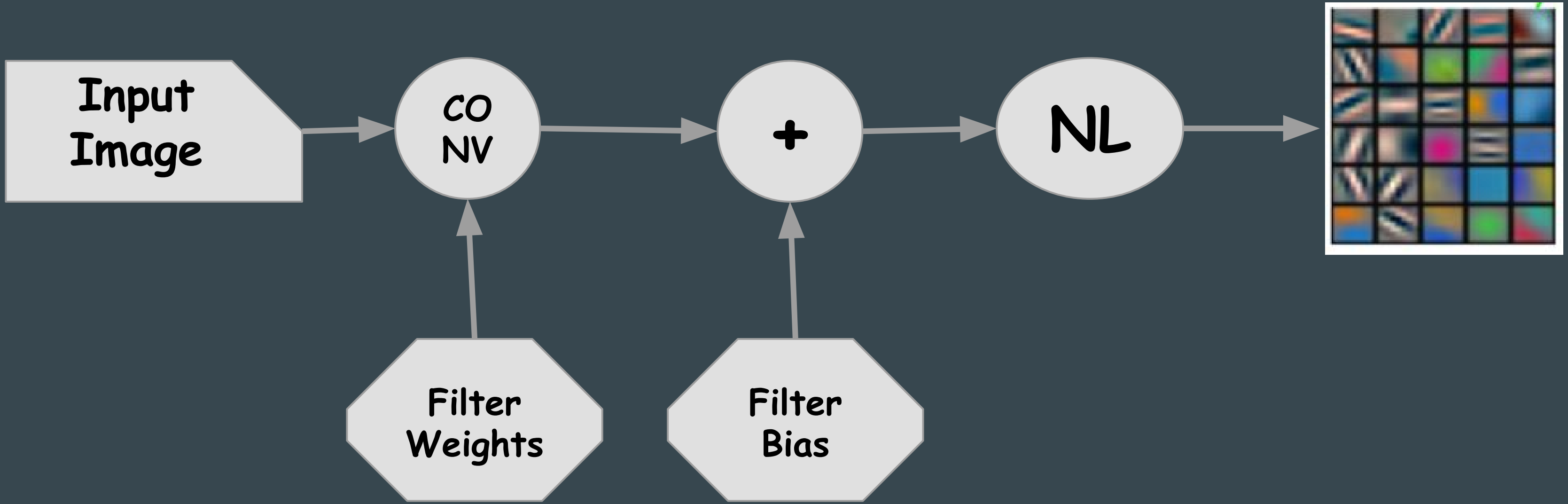
Tensorflow - CNN Operation



Tensorflow - CNN Operation



Tensorflow - CNN Computation Graph



Tensorflow - CNN Implementation

```
f1 =
[filterheight,filterwidth,nb_features,nb_filter]

W1 = tf.Variable(tf.truncated_normal(f1,
stddev=0.1), name="W1")

b1 = tf.Variable(tf.constant(0.1,
shape=[nb_filters[0]]), name="b1")
```

Tensorflow - CNN Implementation

```
conv_1 = tf.nn.conv2d(x, W1, strides=[1,2,2, 1],
padding='SAME')

nl_1 = tf.nn.relu(tf.add(conv_1, b1))
# batchsize,H,W,nb_filter
```

Tensorflow - CNN Take Away

Input Tensor of Dimension
Batch Size x **Height** x **Width** x **Channels**

↓

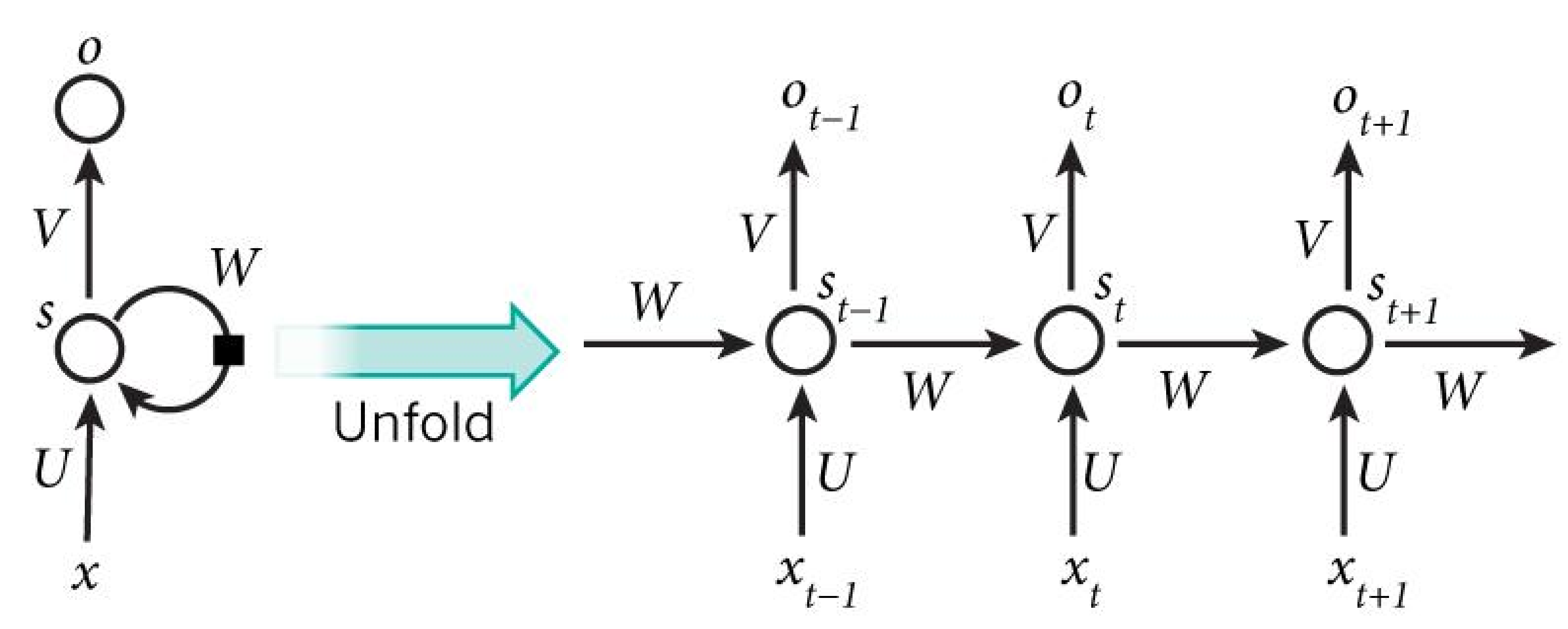
Convolution Block with Non Linearity

- Number of Filters
- Subsampling / Stride
- Padding

↓

Output Tensor of Dimension
Batch Size x **New Height** x **New Width** x **Filters**
New width and height determined by stride and padding

Tensorflow - RNN Implementation



Tensorflow - RNN Implementation

```
cell = tf.nn.rnn_cell.LSTMCell(32,
state_is_tuple=True)

# 32 is number of nodes in hidden layer

val, state = tf.nn.dynamic_rnn(cell, x,
dtype=tf.float32)

# x is a placeholder, val is RNN output
```

Tensorflow - RNN Output

Input Tensor of Dimension
Batch Size x **Timesteps** x **Features**

Recurrent Neural Network Block

- Number of Nodes

Sequence Labelling

Output Tensor of Dimension
Batch Size x **Timesteps** x **Nodes**
(Take All Last Timestep)

Sequence Classification

Output Tensor of Dimension
Batch Size x **Nodes**
(Take Only Last Timestep)

Tensorflow - RNN Output

Sequence Classification

Output Tensor of Dimension
Batch Size x **Nodes**
(Take Only Last Timestep) OK... But How ?

```
val, state = tf.nn.dynamic_rnn(cell, x,
dtype=tf.float32)

# val is RNN output Batch_Size x Timesteps x Nodes
```

Tensorflow - RNN Output

Sequence Classification

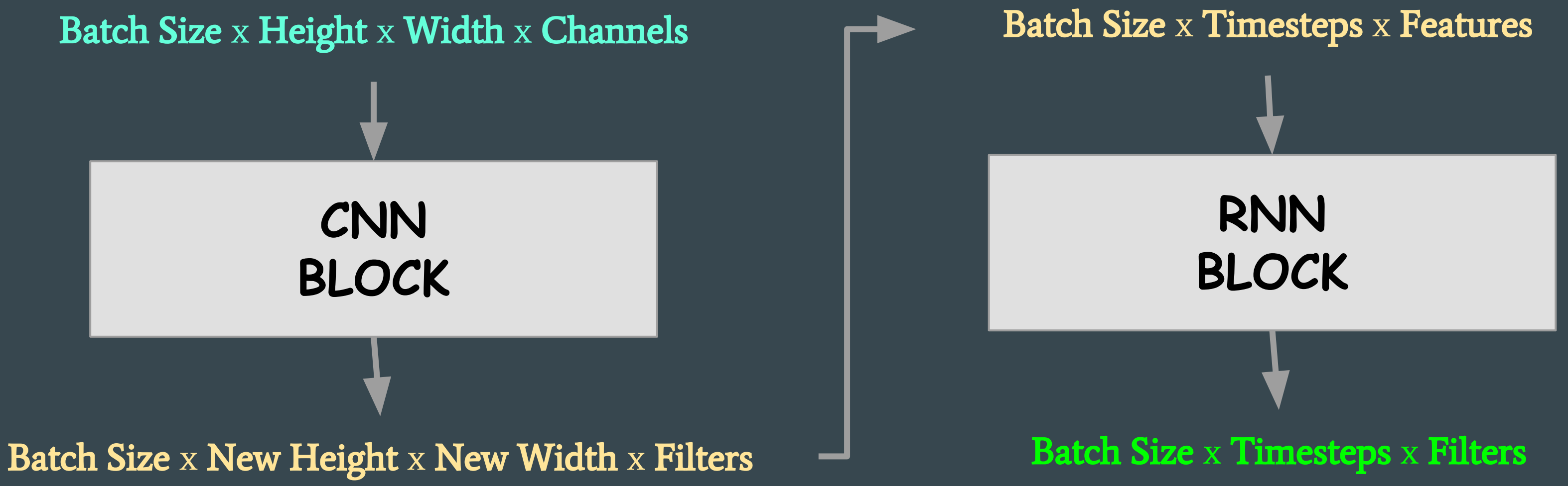
(Take Only Last Timestep) OK... But How ?

```
val, state = tf.nn.dynamic_rnn(cell, x, dtype=tf.float32)
# val is RNN output Batch_Size x Timesteps x Nodes

time_major=tf.transpose(val,[1,0,2])
# time_major is now Timesteps x Batch_Size x Nodes

last_step = time_major[-1]
# last_step is now Batch_Size x Nodes
# Use regular Fully Connected blocks like MLP afterwards
```

Tensorflow - Bridge CNN and RNN



Filters of CNN = Features of RNN
New Height x New Width = Timesteps (Row Major / Column Major)

Tensorflow - Bridge CNN and RNN

Batch Size x Height x Width x Channels

↓

CNN BLOCK

↓

Batch Size x New Height x New Width x Filters

Batch Size x Timesteps x Features

↓

RNN BLOCK

↓

Batch Size x Timesteps x Filters

```
tf.Tensor.shape()
tf.reshape()
```

Example of such transformation
https://github.com/xisnu/CNN-BLSTM-CTC

Tensorflow - Bridge CNN and RNN

Is that useful ?

YES ! !

Online Handwriting Recognition

14th IAPR Conference on Document Analysis and Recognition 2017

A Hybrid Model for End to End Online Handwriting Recognition

Partha S. Mukherjee, Ujjwal Bhattacharya, Swapan K. Parui
Computer Vision and Pattern Recognition Unit
Indian Statistical Institute
Kolkata, India
parthosarothimukherjee@gmail.com, {ujjwal,swapan}@isical.ac.in

Bappaditya Chakraborty
Department of Computer Science and Engineering
Brainware University
Kolkata, India
bappa.chakraborty84@gmail.com

***Abstract*—Automatic recognition of online handwritten words in a generic way has significant application potentials. However, this recognition job is challenging for unconstrained handwriting data. The challenge is more serious for Indic scripts like Devanagari or Bangla due to the inherent cursiveness of their characters, large volumes of respective alphabets, existence of several groups of shape similar characters etc. On the other hand, with the recent development of powerful machine learning tools, major research initiatives in this area of pattern recognition studies have been observed. Feature extraction and classification are two major modules of such a recognizer. Deep architectures of convolutional neural network (CNN) models have been found to be efficient in extraction of useful features from raw signal. On the other hand, a recurrent neural network (RNN) along with connectionist temporal classification (CTC) has been shown to be able to label unsegmented sequence data. In the present article, we propose a hybrid layered architecture consisting of three networks CNN, RNN and CTC for recognition of online handwriting without use of any specific lexicon. In this study, we have also observed that feeding hand-crafted features to the CNN at the first level of the proposed model provides better performance than feeding the raw signal to the CNN. We have simulated the proposed model on two large databases of Devanagari and Bangla online unconstrained handwritten words. The recognition accuracies provided by the proposed model are encouraging.**

I. Introduction

From the perspective of automatic recognition, handwriting data are often categorized into offline and online formats. Offline handwriting sample is stored in the form of a two-dimensional image while online handwriting sample is stored as a temporal sequence of two-dimensional coordinate points determining the trajectory of pen tip movement along with some additional information such as pen status ('up' or 'down') etc. Automatic recognition or interpretation of both of these two types of handwriting data has their respective challenges. Since the beginning, study of handwriting data has attracted attention of the researchers in the area of pattern recognition. However, automatic recognition of unconstrained cursive handwriting has always been met with serious challenges. In such type of handwriting, information about the boundaries of individual characters are not readily available because while a writer writes in an unconstrained way, the lifting of pen depends upon his/her idiosyncracy instead of the end of a character. Thus, the task of recognition of cursive handwriting is far more challenging [1] than recognition of isolated handwritten characters. In this work, we have studied recognition of unconstrained online handwriting of Devenagari and Bangla, the two most popular Indian scripts. This type of handwriting data are captured by touch screen devices, pen tablets etc. Such devices store coordinates of points on the writing surface along the path of movement of finger tip or stylus as temporal sequence. The part of such a sequence between a pair of successive 'pen down' and 'pen up' situations is often termed as a stroke. A piece of online handwritten data is composed of one or more such strokes. An example of such online handwriting data is shown in Fig. 1.

Figure 1. A piece of online handwritten Hindi text written in Devanagari script is shown. Circles show the positions of captured coordinates on the writing surface. Different colors mark different strokes.

A. Devanagari Script

Devanagari is one of the most widely used scripts in south eastern part of Asia. This is a descendant of old Brahmi script and its early use was found around 1000 CE. Devanagari script is used to write several languages like Sanskrit, Hindi, Nepali, Marathi, Kashmiri etc. The type of Devanagari script is *alpha-syllabary* (also known as *Abugida*) [2] where a consonant and vowel composition is often written as a single unit. Also, two or more of its basic consonant characters can combine together to form another compound character. Due to this chracteristics of this script, the size of its alphabet is large consisting of many compound characters. Fig. 2 shows modified forms of a basic consonant of Devanagari when it is attached with different basic vowel characters. On the other hand, the first two rows of Fig. 3 show formation of Devanagari compound characters due to combinations of two basic consonant characters.

Tensorflow - Give it a try