Deep Learning using TensorFlow

-By Prakash V

Intro

Introduction to Tensorflow and brief overview on architecture, trends and performance.

Neural Network using Numpy.

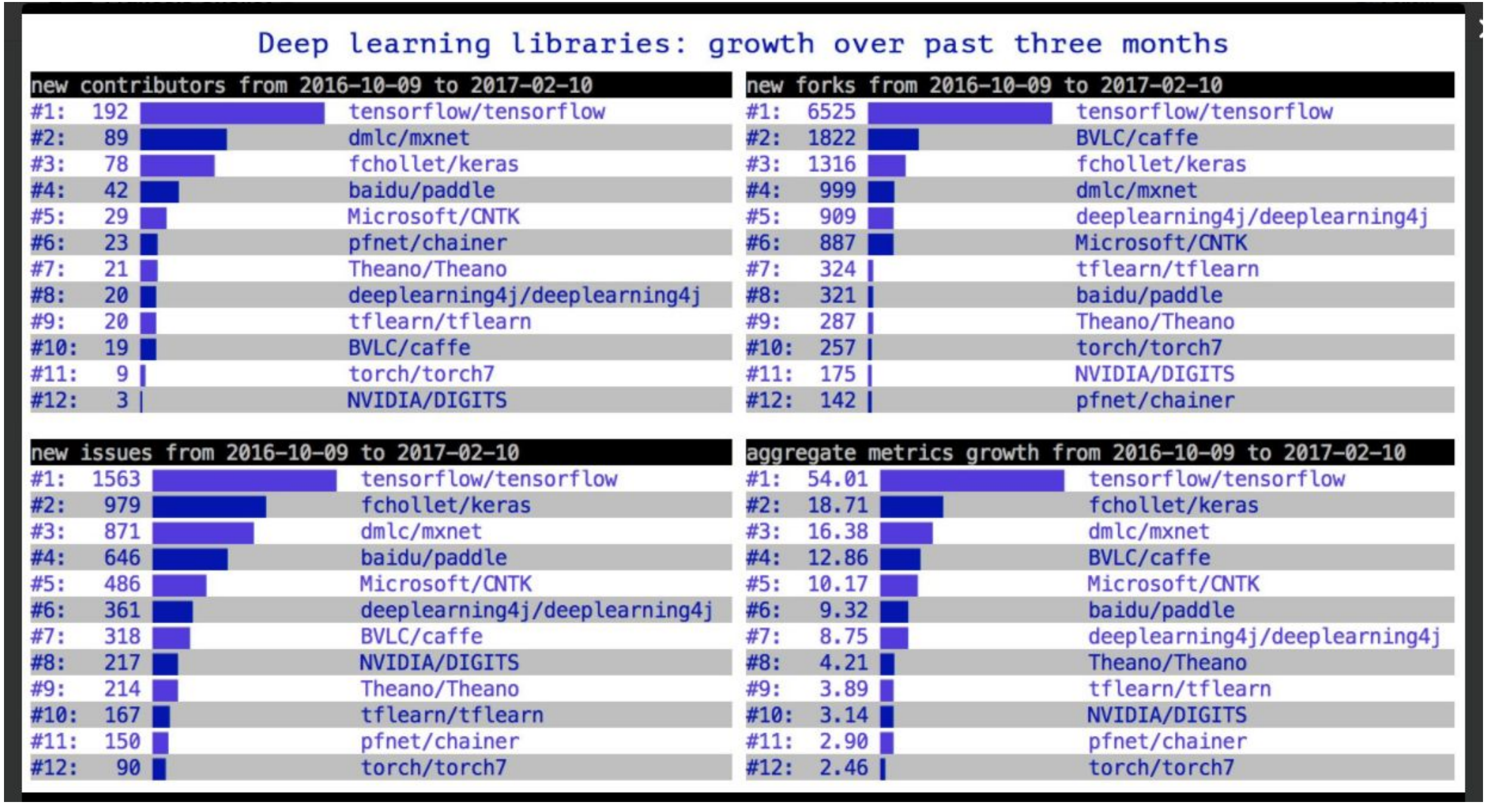
Neural Network using Tensorflow.

Deeper into Computer Vision

Fractional Max Pooling on MNIST Dataset.

Fractional Max Pooling on CIFAR-100 Dataset.

Trends - Deep Learning Libraries



```
Deep learning libraries: growth over past three months

new contributors from 2016-10-09 to 2017-02-10
#1:   192    tensorflow/tensorflow
#2:    89    dmlc/mxnet
#3:    78    fchollet/keras
#4:    42    baidu/paddle
#5:    29    Microsoft/CNTK
#6:    23    pfnet/chainer
#7:    21    Theano/Theano
#8:    20    deeplearning4j/deeplearning4j
#9:    20    tflearn/tflearn
#10:   19    BVLC/caffe
#11:    9    torch/torch7
#12:    3    NVIDIA/DIGITS

new forks from 2016-10-09 to 2017-02-10
#1:   6525   tensorflow/tensorflow
#2:   1822   BVLC/caffe
#3:   1316   fchollet/keras
#4:    999   dmlc/mxnet
#5:    909   deeplearning4j/deeplearning4j
#6:    887   Microsoft/CNTK
#7:    324   tflearn/tflearn
#8:    321   baidu/paddle
#9:    287   Theano/Theano
#10:   257   torch/torch7
#11:   175   NVIDIA/DIGITS
#12:   142   pfnet/chainer

new issues from 2016-10-09 to 2017-02-10
#1:   1563   tensorflow/tensorflow
#2:    979   fchollet/keras
#3:    871   dmlc/mxnet
#4:    646   baidu/paddle
#5:    486   Microsoft/CNTK
#6:    361   deeplearning4j/deeplearning4j
#7:    318   BVLC/caffe
#8:    217   NVIDIA/DIGITS
#9:    214   Theano/Theano
#10:   167   tflearn/tflearn
#11:   150   pfnet/chainer
#12:    90   torch/torch7

aggregate metrics growth from 2016-10-09 to 2017-02-10
#1:   54.01   tensorflow/tensorflow
#2:   18.71   fchollet/keras
#3:   16.38   dmlc/mxnet
#4:   12.86   BVLC/caffe
#5:   10.17   Microsoft/CNTK
#6:    9.32   baidu/paddle
#7:    8.75   deeplearning4j/deeplearning4j
#8:    4.21   Theano/Theano
#9:    3.89   tflearn/tflearn
#10:   3.14   NVIDIA/DIGITS
#11:   2.90   pfnet/chainer
#12:   2.46   torch/torch7
```

Neural networks

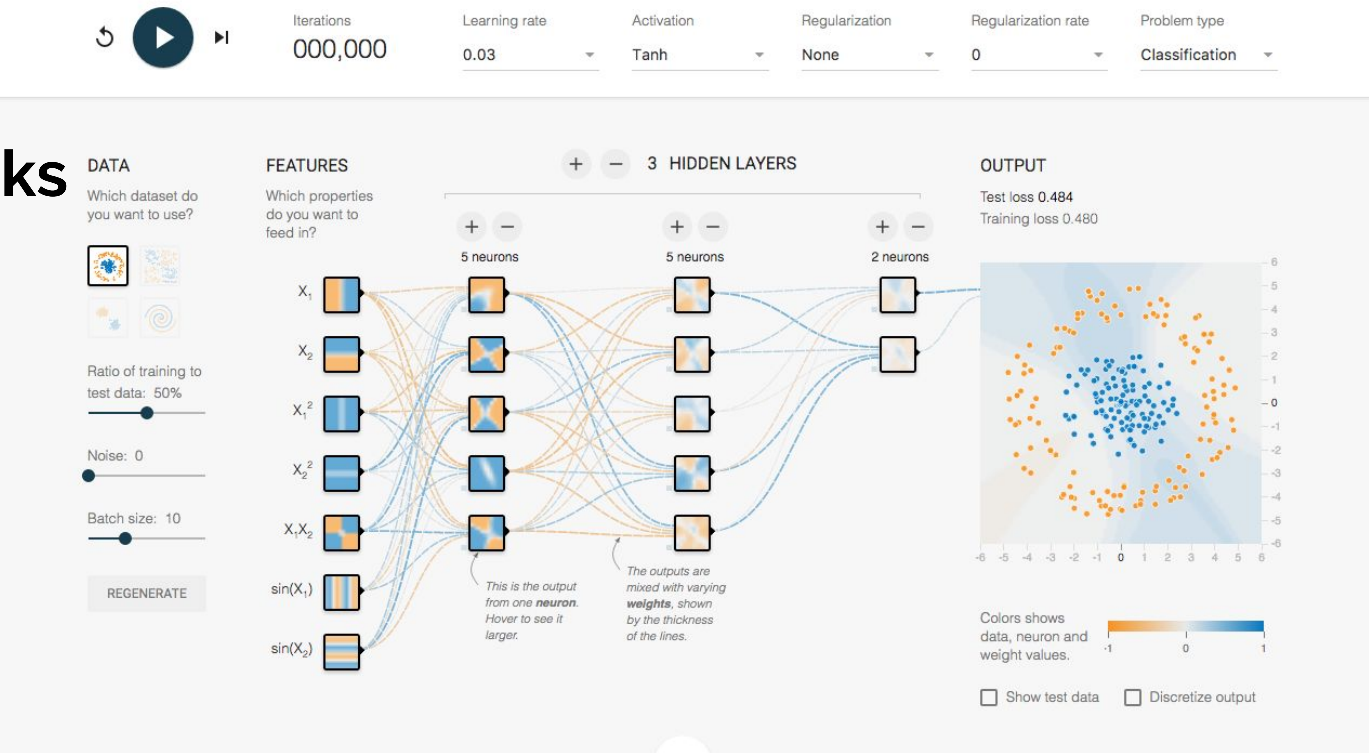
Input Data

Number of Layers

Activation Functions

Loss Function

Gradient optimizer



Iterations 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

3 HIDDEN LAYERS

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

OUTPUT

Test loss 0.484

Training loss 0.480

Colors shows data, neuron and weight values.

□ Show test data □ Discretize output

http://playground.tensorflow.org/

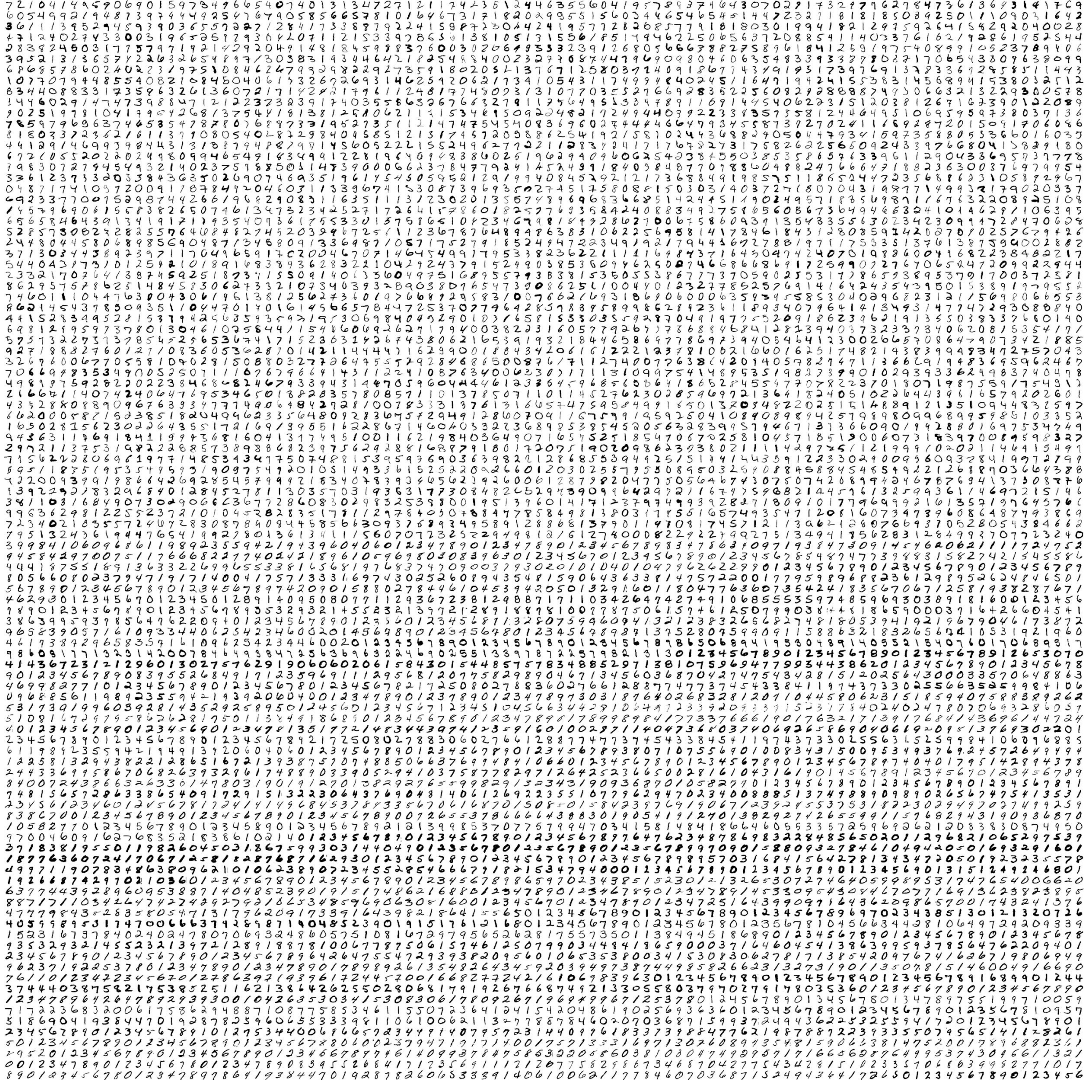
MNIST DATASET

Train Images - 60000

Test Images - 10000

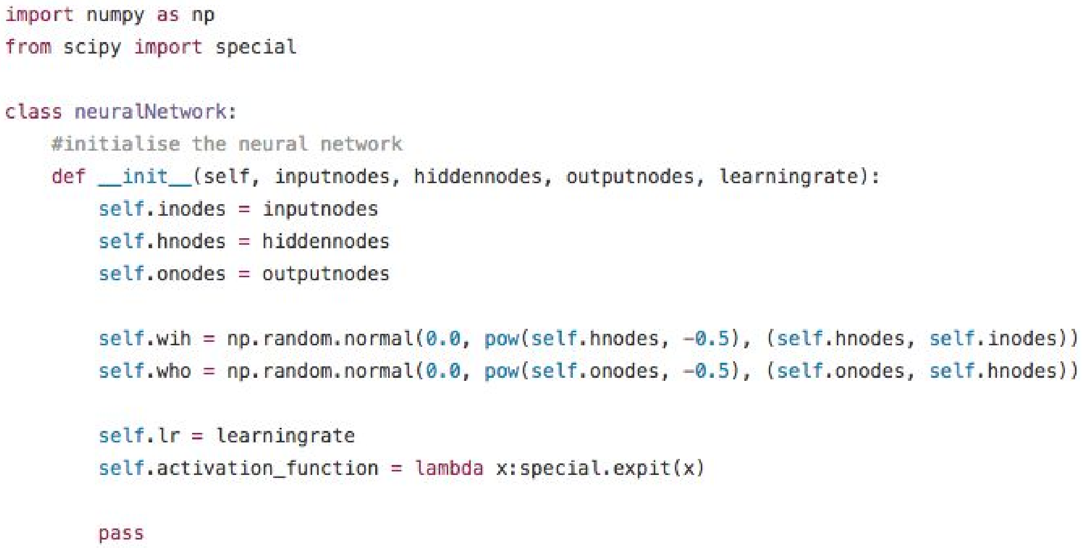
Image size - 28 * 28 pixels

Class - 0 , 1, 2, 3, 4, 5, 6, 7, 8 , 9



Neural Network using Numpy

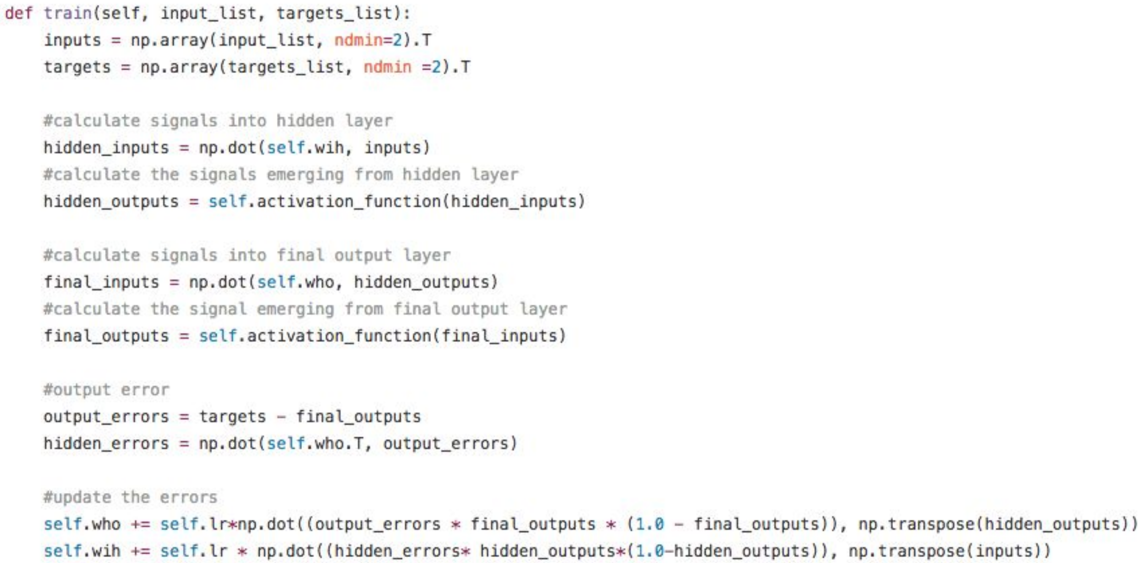
- A 3 layer Neural network.
- Initialize the weights randomly.
- Fix a learning rate.
- Use sigmoid as the activation function.



```
import numpy as np
from scipy import special

class neuralNetwork:
    #initialise the neural network
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        self.wih = np.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))
        self.who = np.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, self.hnodes))

        self.lr = learningrate
        self.activation_function = lambda x:special.expit(x)

        pass
```

Neural Network using Numpy-2

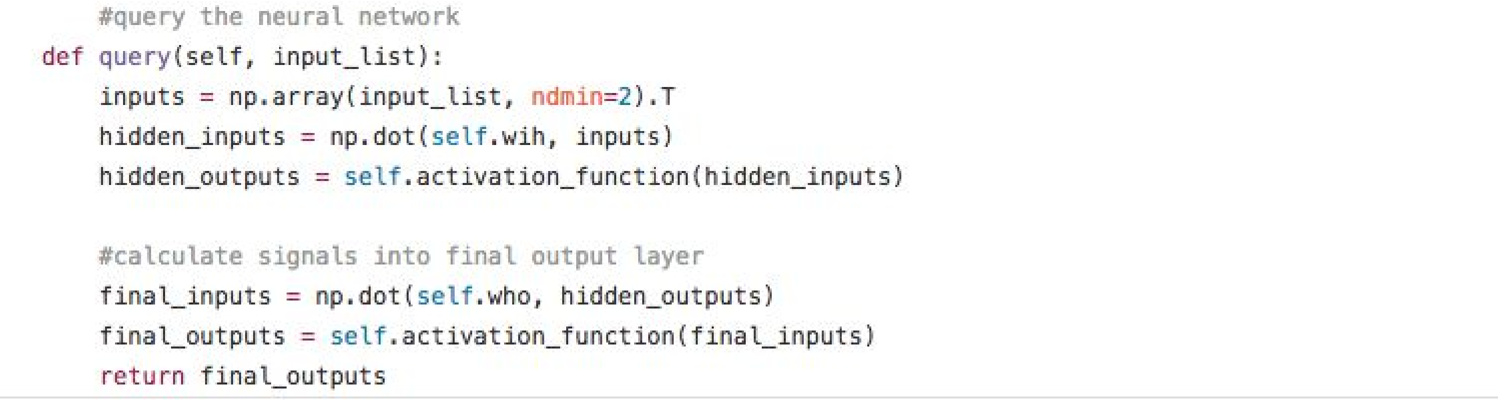
- Give Inputs and Targets
- For each call to the train
    - A forward pass happens
    - Errors are calculated
    - Mean squared error is used for calculating the error.
    - Errors are back propagated.



```
def train(self, input_list, targets_list):
    inputs = np.array(input_list, ndmin=2).T
    targets = np.array(targets_list, ndmin =2).T

    #calculate signals into hidden layer
    hidden_inputs = np.dot(self.wih, inputs)
    #calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    #calculate signals into final output layer
    final_inputs = np.dot(self.who, hidden_outputs)
    #calculate the signal emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    #output error
    output_errors = targets - final_outputs
    hidden_errors = np.dot(self.who.T, output_errors)

    #update the errors
    self.who += self.lr*np.dot((output_errors * final_outputs * (1.0 - final_outputs)), np.transpose(hidden_outputs))
    self.wih += self.lr * np.dot((hidden_errors* hidden_outputs*(1.0-hidden_outputs)), np.transpose(inputs))
```

Neural Network using Numpy-3

- How to test the result ?
- We write a query function to do that.



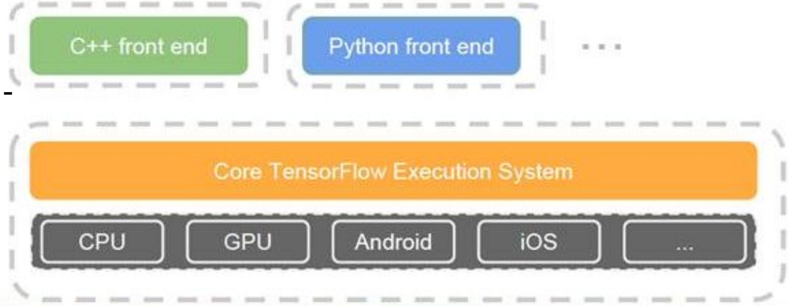
```
    #query the neural network
def query(self, input_list):
    inputs = np.array(input_list, ndmin=2).T
    hidden_inputs = np.dot(self.wih, inputs)
    hidden_outputs = self.activation_function(hidden_inputs)

    #calculate signals into final output layer
    final_inputs = np.dot(self.who, hidden_outputs)
    final_outputs = self.activation_function(final_inputs)
    return final_outputs
```

Feed Forward Neural Networks using Numpy

About TensorFlow

- Open source software library for numerical computation using data flow graphs
- Nodes - operations , Edges - multi-dimensional Arrays (Tensors)
- Deploy computations to one or more CPUs and GPUs in Desktop, server, mobile device with single API
- Connect research and production - TensorFlow Serving.

- Auto Differentiation - Tensorflow handles computing the derivatives for you.
- Available in your favourite languages- Python , C++, Java, Go. Interfaces- Lua, JavaScript and R

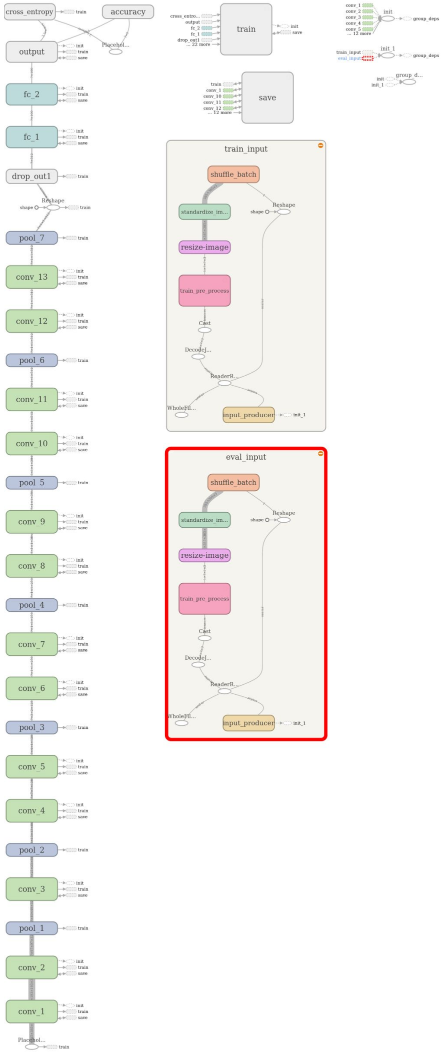


Advantage of using Tensorflow

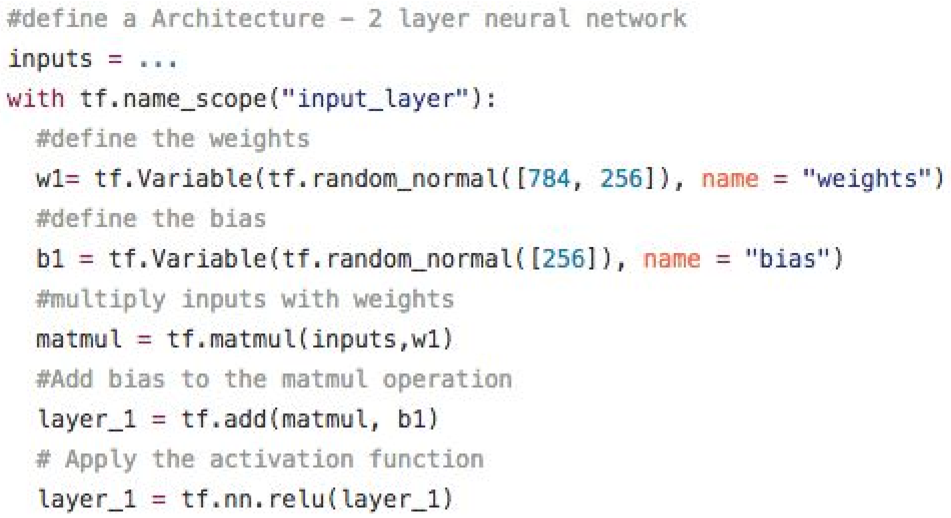
- Less Overhead
- Queues for input data
- Multi-GPU, CPU support
- Tensorboard for Visualization
- Model Checkpointing
- TF-slim, Keras, TFlearn

Operations

- Design the Architecture
- Choose the cost function.
- Choose the Gradient to minimize the cost.
- Choose the metrics to calculate (optional).
- Define the placeholders
- Launch the graph and feed data to network

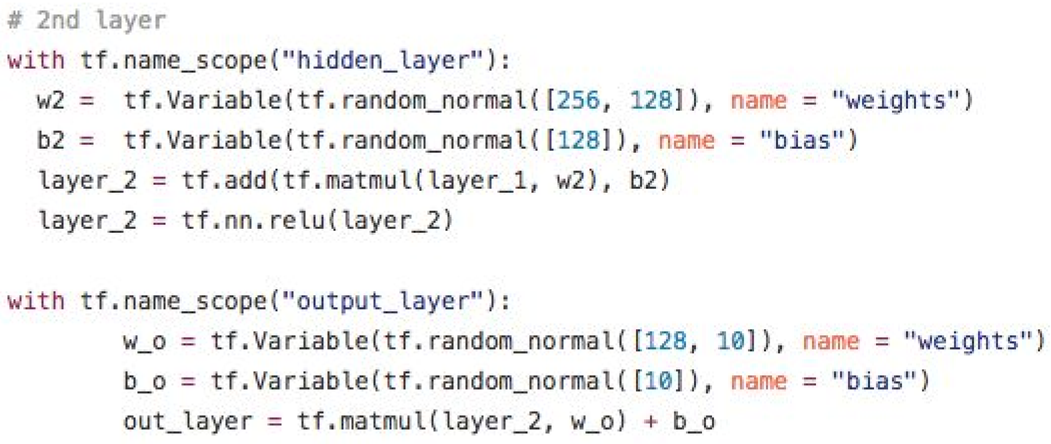


Design the Architecture using TensorFlow



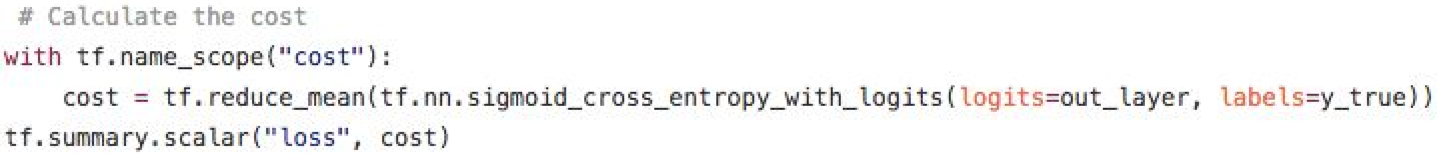
```
#define a Architecture - 2 layer neural network
inputs = ...
with tf.name_scope("input_layer"):
  #define the weights
  w1= tf.Variable(tf.random_normal([784, 256]), name = "weights")
  #define the bias
  b1 = tf.Variable(tf.random_normal([256]), name = "bias")
  #multiply inputs with weights
  matmul = tf.matmul(inputs,w1)
  #Add bias to the matmul operation
  layer_1 = tf.add(matmul, b1)
  # Apply the activation function
  layer_1 = tf.nn.relu(layer_1)
```

Design the Architecture using TensorFlow



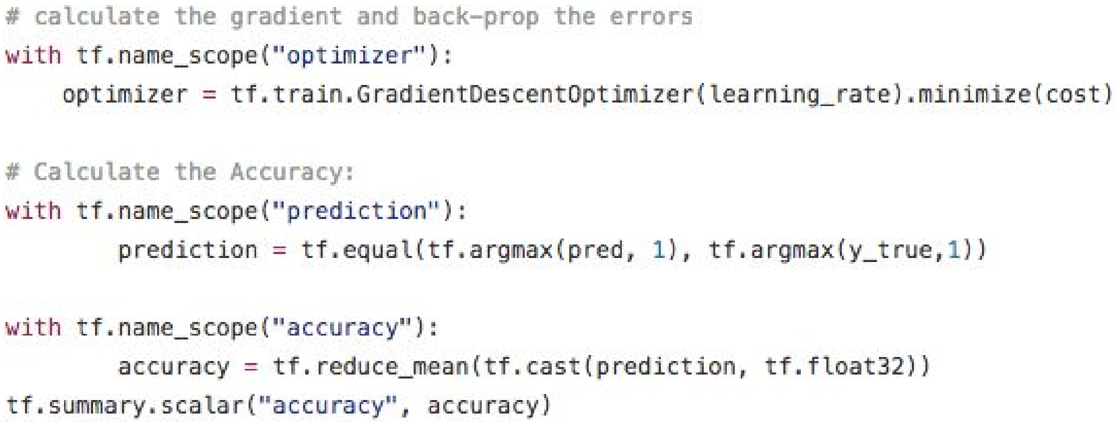
```
# 2nd layer
with tf.name_scope("hidden_layer"):
 w2 =  tf.Variable(tf.random_normal([256, 128]), name = "weights")
 b2 =  tf.Variable(tf.random_normal([128]), name = "bias")
 layer_2 = tf.add(tf.matmul(layer_1, w2), b2)
 layer_2 = tf.nn.relu(layer_2)

with tf.name_scope("output_layer"):
    w_o = tf.Variable(tf.random_normal([128, 10]), name = "weights")
    b_o = tf.Variable(tf.random_normal([10]), name = "bias")
    out_layer = tf.matmul(layer_2, w_o) + b_o
```

Calculate the loss



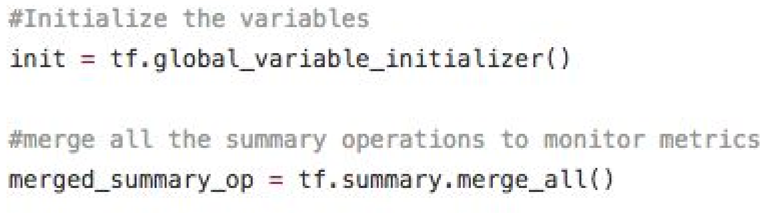
```
 # Calculate the cost
with tf.name_scope("cost"):
    cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=out_layer, labels=y_true))
tf.summary.scalar("loss", cost)
```

Optimize the cost, Calculate the Accuracy



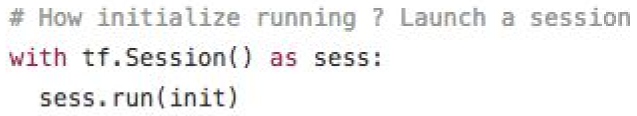
```
# calculate the gradient and back-prop the errors
with tf.name_scope("optimizer"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Calculate the Accuracy:
with tf.name_scope("prediction"):
        prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_true,1))

with tf.name_scope("accuracy"):
        accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))
tf.summary.scalar("accuracy", accuracy)
```

Initialize all the variables, Merge all the metrics calculated.



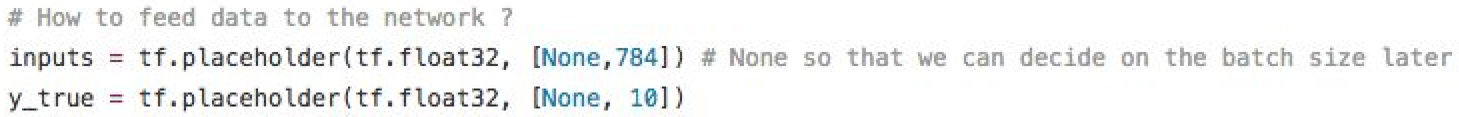
```
#Initialize the variables
init = tf.global_variable_initializer()

#merge all the summary operations to monitor metrics
merged_summary_op = tf.summary.merge_all()
```

Launch the graph



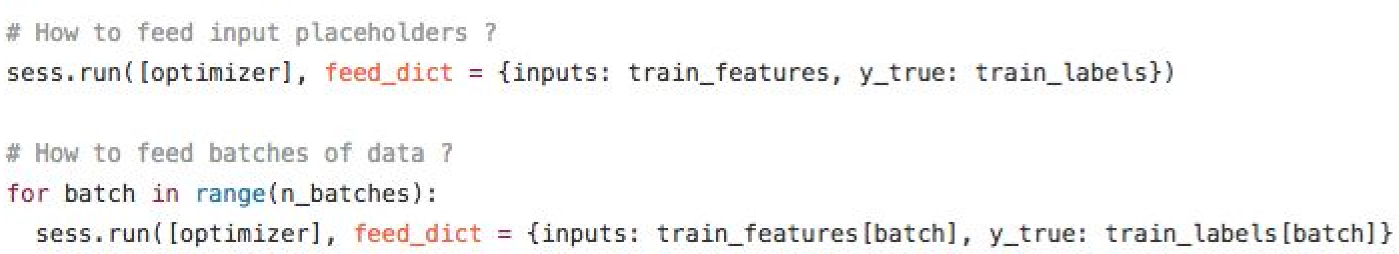
```
# How initialize running ? Launch a session
with tf.Session() as sess:
  sess.run(init)
```

How to feed data to network ?



```
# How to feed data to the network ?
inputs = tf.placeholder(tf.float32, [None,784]) # None so that we can decide on the batch size later
y_true = tf.placeholder(tf.float32, [None, 10])
```

Feed Data to Network



```
# How to feed input placeholders ?
sess.run([optimizer], feed_dict = {inputs: train_features, y_true: train_labels})

# How to feed batches of data ?
for batch in range(n_batches):
  sess.run([optimizer], feed_dict = {inputs: train_features[batch], y_true: train_labels[batch]}
```

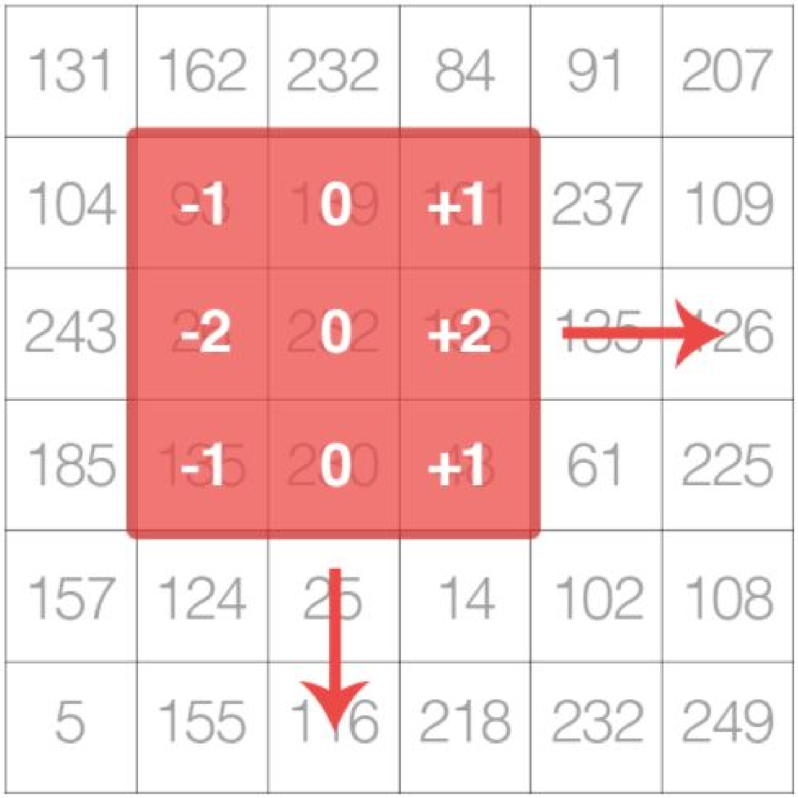
Feed Forward Neural Networks using Tensorflow

Logistic Regression using TensorFlow

A little deeper into Computer Vision

What are kernels?

- Kernels or convolutional matrix is a tiny matrix that is used for blurring, sharpening and edge detection.
- Convolution Process:
  - An input image
  - A kernel matrix that we are going to apply to the input image.
  - An output image to store the output of the input image convolved with the kernel.



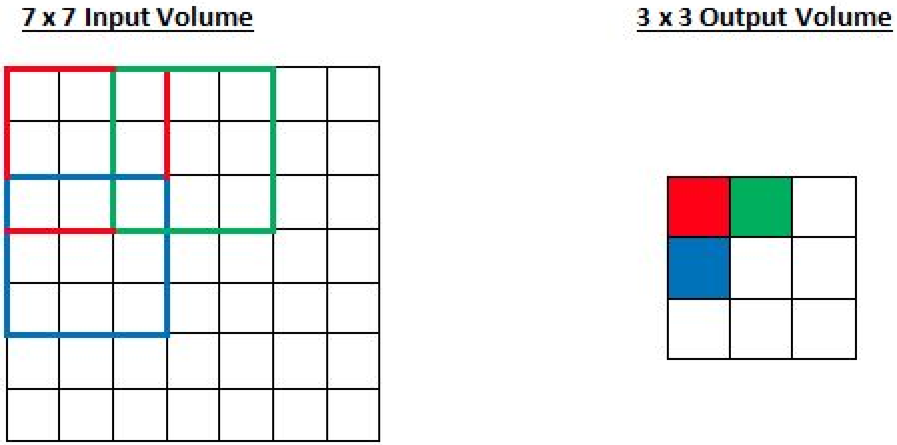
131	162	232	84	91	207
104	-1	0	+1	237	109
243	-2	0	+2	135	126
185	-1	0	+1	61	225
157	124	25	14	102	108
5	155	1[illegible]6	218	232	249

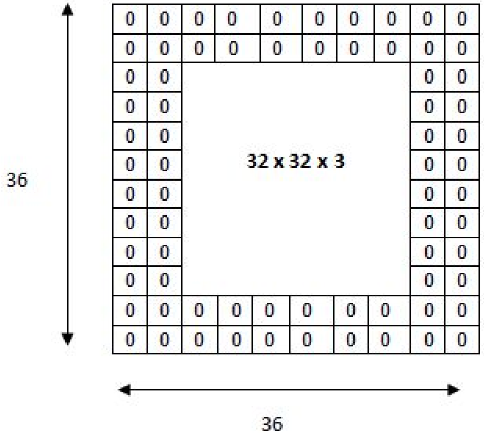
FIGURE 1: A KERNEL IS A SMALL MATRIX THAT SLIDES ACROSS FROM LEFT-TO-RIGHT AND TOP-TO-BOTTOM OF A LARGER IMAGE. AT EACH PIXEL IN THE INPUT IMAGE, THE NEIGHBORHOOD OF THE IMAGE IS CONVOLVED WITH THE KERNEL AND THE OUTPUT STORED.

http://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/

Terminology in Convolution

- Strides
- Padding
- Filter maps



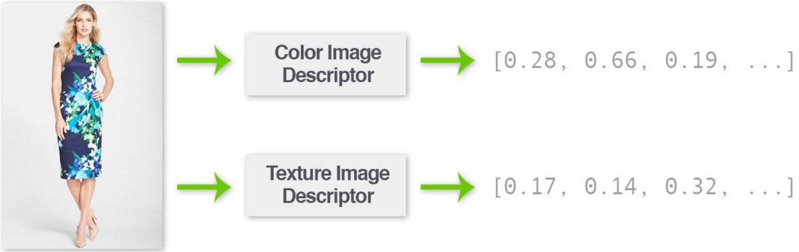


The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

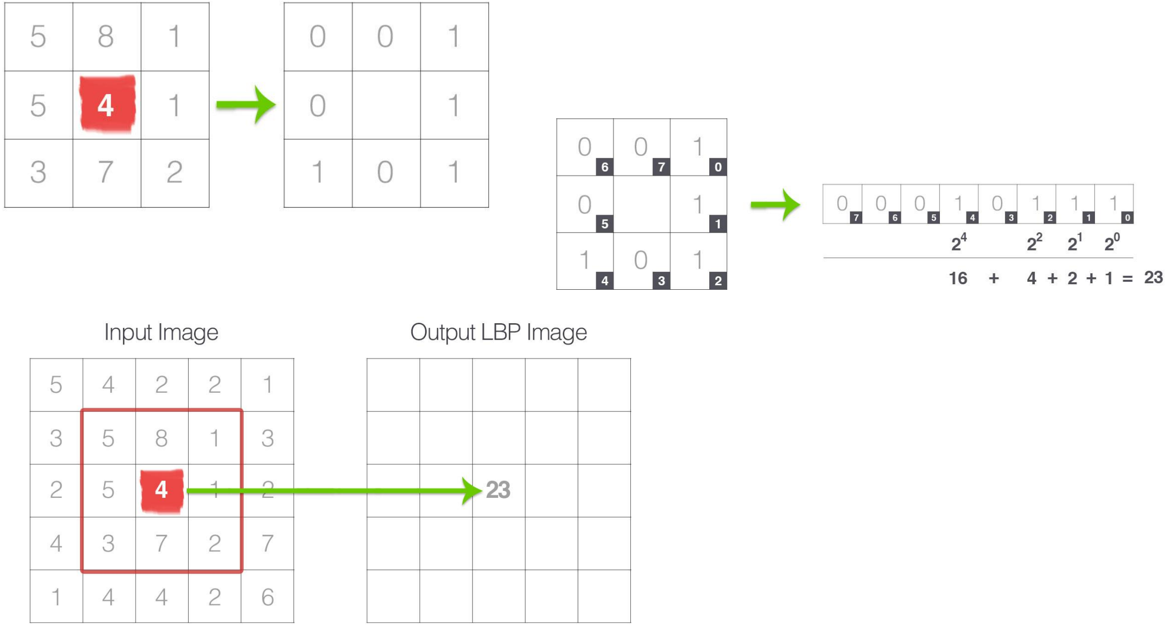
How to Vectorize an Image?

- An abstraction of an image used to characterize and numerically quantify the contents of an image. Normally real, integer, or binary valued. Simply put, a feature vector is a list of numbers used to represent an image.
- Color: Make a histogram with n-bins using a pixels. Normalize it and it becomes your color Image Descriptor
- Local Binary Patterns : Used to detect the patterns in Images locally.
- Most popularly used - Histogram of oriented Gradients.



Color Image Descriptor → [0.28, 0.66, 0.19, ...]

Texture Image Descriptor → [0.17, 0.14, 0.32, ...]



5	8	1
5	4	1
3	7	2

0	0	1
0		1
1	0	1

0 6	0 7	1 0
0 5		1 1
1 4	0 3	1 2

0 7	0 6	0 5	1 4	0 3	1 2	1 1	1 0

$2^4 \quad 2^2 \quad 2^1 \quad 2^0$

16 + 4 + 2 + 1 = 23

Input Image

5	4	2	2	1
3	5	8	1	3
2	5	4	1	2
4	3	7	2	7
1	4	4	2	6

Output LBP Image

		23		

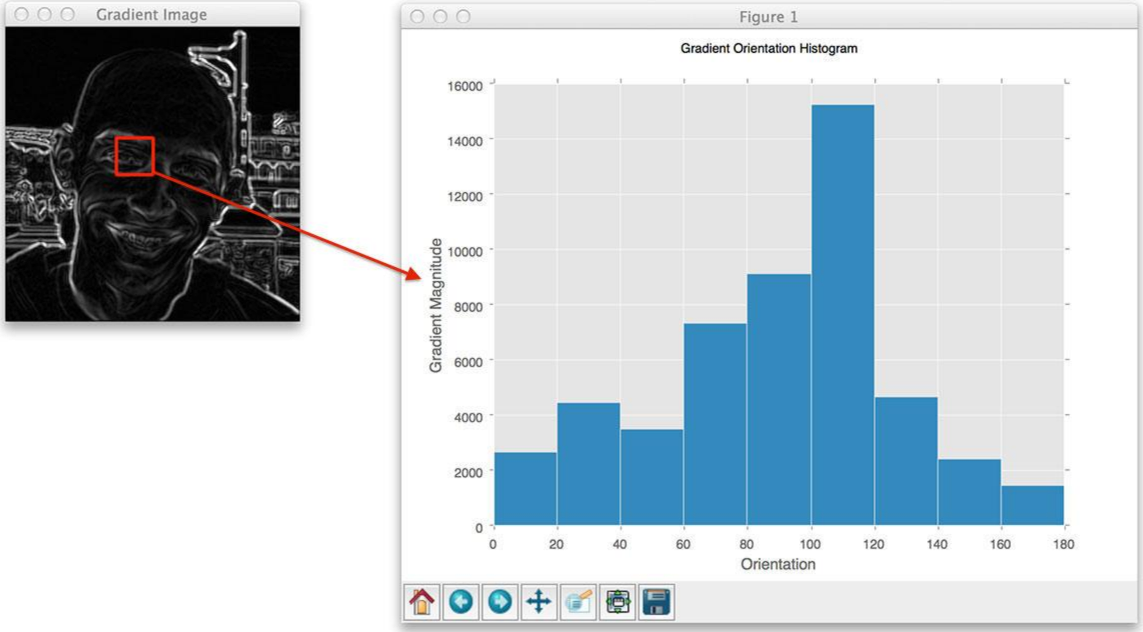
How to do Classification in Images using Image Descriptors?

- Histogram of Oriented Gradients.
    - Pixels per cell
    - Orientations
    - Cells_per_block

Example :

- 32 * 32 input Image
- Pixels per cell - 3* 3 = 9
- Cells per block = 2*2
- Orientation = 9

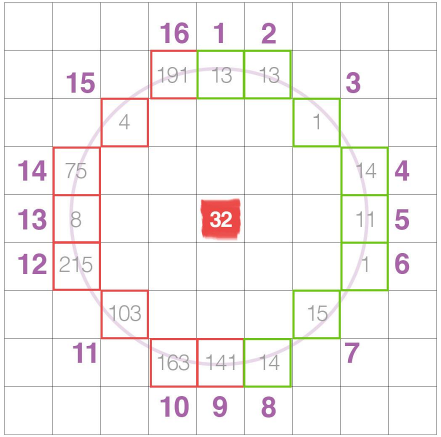
Total Features = ?

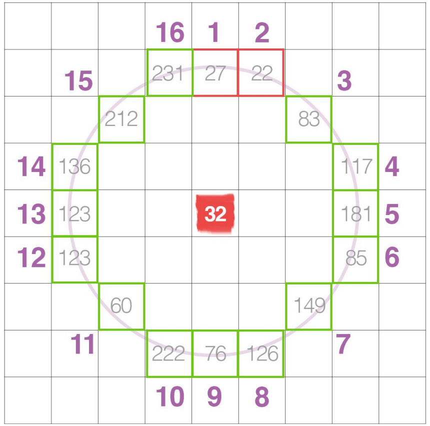


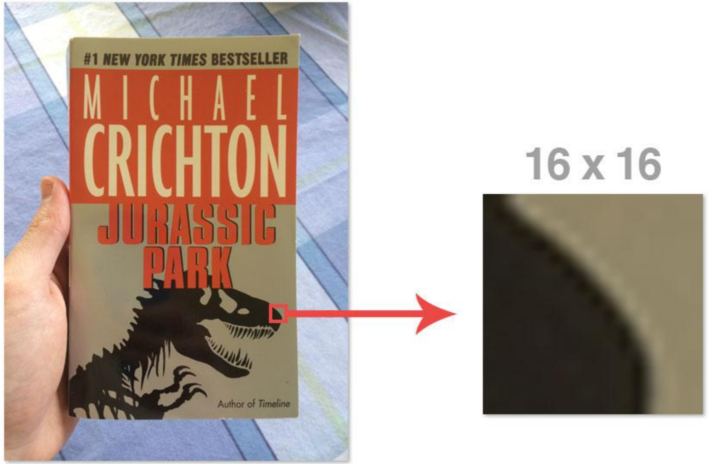
http://www.pyimagesearch.com/tag/hog/

How to do Classification in Images using Feature Descriptors?

- An Algorithm to detect Key points - example : FAST
- R = 3 Generally and we only consider points along the circle
- Now use SIFT Algorithm to detect features.
- FAST - detect edges, blobs etc
- SIFT extract features from this edge.
- We extract a patch for every key point and apply HOG to get a feature vector.
- RootSIFT: Take square root of each of your element.







Building an Image Classifier.

Extract Key Points and Feature Descriptors

N*M Dimensional vector for each Image

BAG of Visual Words, Do clustering.

Define Number of clusters. Domain specific

Multiple Features to Single Histogram

Create a zero vector with length as number of visual words

For each Feature Descriptor. Find nearest cluster and update vector with +1

Resultant vector with count of occurrence of each visual word is our histogram.

CHOOSE YOUR MACHINE LEARNING CLASSIFIER

Can We build a Image Search Engine In this Process?

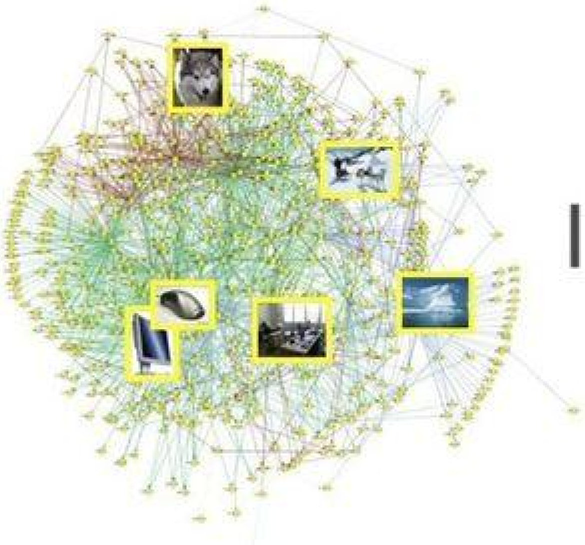
Where does Neural Networks comes into Picture?

Convolutional Neural Networks

Space Invariant Artificial Neural Network.

Inspired by How Human Perceive Object.

The First paper on CNNs is dating back to 1998. The Paper is called LeNet published by Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.



IMAGENET

Early Problems

Not Enough Computational Power

Not Enough Data

Back Propagation was not deep enough

Important Terminology

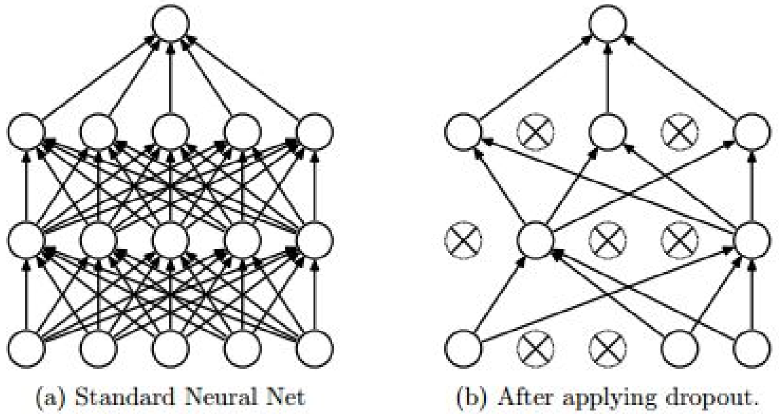
Convolution Layer - Strides , Filters, Padding

Pooling Layers - Max , Avg, Fractional

Dropout , DropConnect

Activation Function

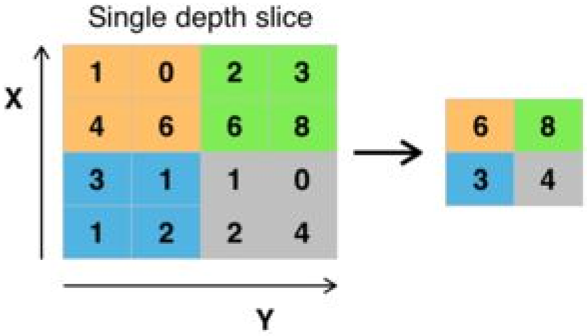
Loss Metrics



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.



Single depth slice

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

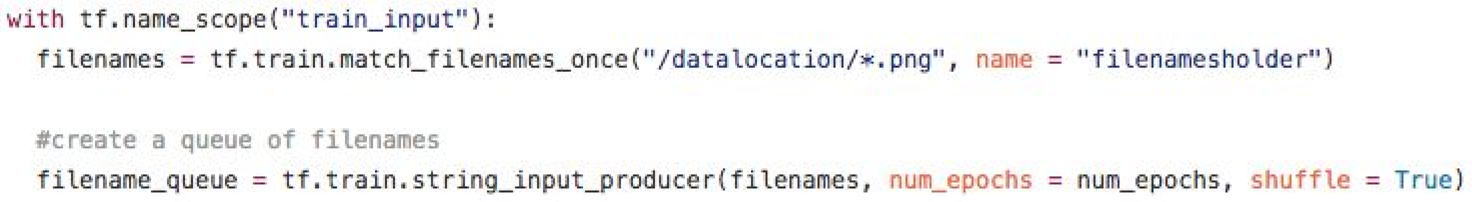
6	8
3	4

Activation Functions

- Sigmoid
- Relu
- Leaky relu
- Tanh
- Maxout
- Softplus
- softmax

Data Pipelines in TensorFlow

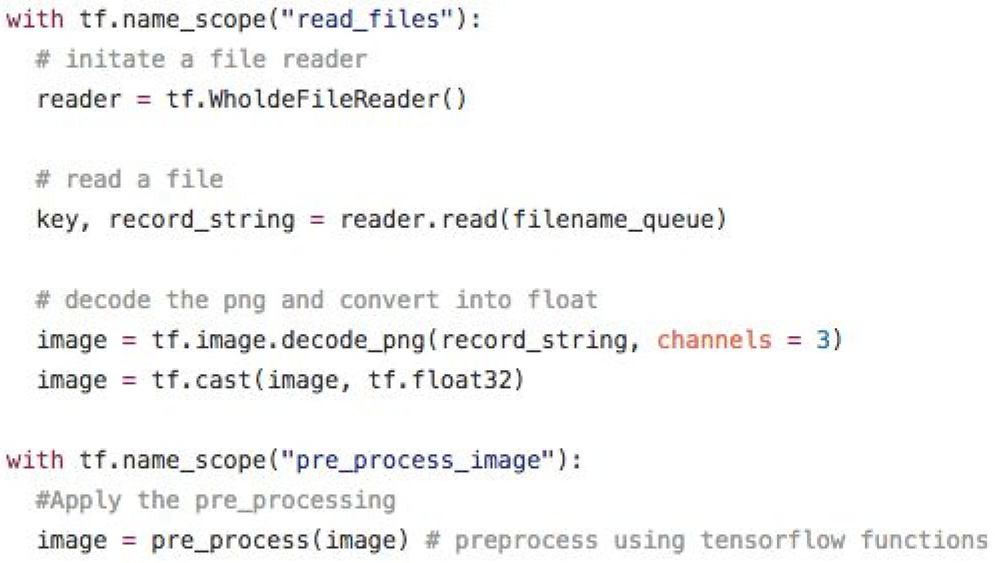
Read the Image Locations



```
with tf.name_scope("train_input"):
  filenames = tf.train.match_filenames_once("/datalocation/*.png", name = "filenamesholder")

  #create a queue of filenames
  filename_queue = tf.train.string_input_producer(filenames, num_epochs = num_epochs, shuffle = True)
```

Data Pipelines in TensorFlow

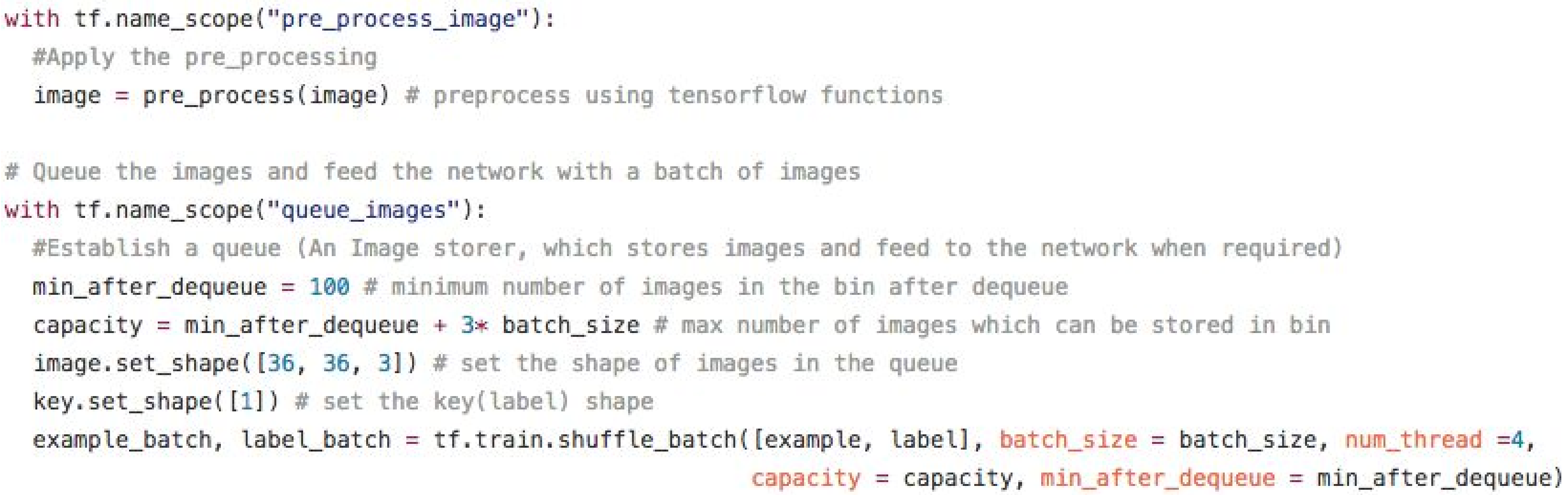
Decode the Image



```
with tf.name_scope("read_files"):
  # initate a file reader
  reader = tf.WholdeFileReader()

  # read a file
  key, record_string = reader.read(filename_queue)

  # decode the png and convert into float
  image = tf.image.decode_png(record_string, channels = 3)
  image = tf.cast(image, tf.float32)

with tf.name_scope("pre_process_image"):
  #Apply the pre_processing
  image = pre_process(image) # preprocess using tensorflow functions
```

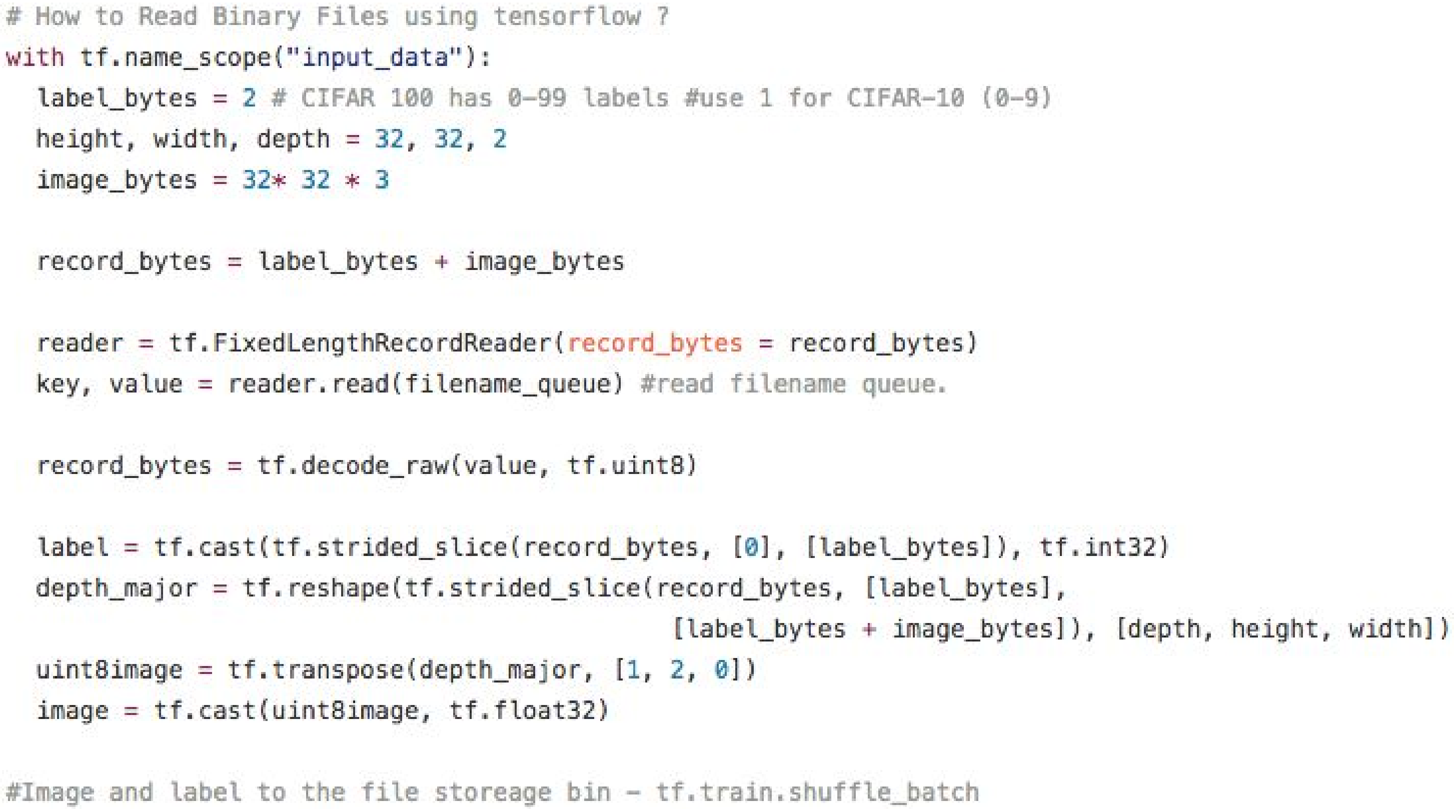
Data Pipelines in TensorFlow

Enqueue the images and feed to the network batch_wise



```
with tf.name_scope("pre_process_image"):
  #Apply the pre_processing
  image = pre_process(image) # preprocess using tensorflow functions

# Queue the images and feed the network with a batch of images
with tf.name_scope("queue_images"):
  #Establish a queue (An Image storer, which stores images and feed to the network when required)
  min_after_dequeue = 100 # minimum number of images in the bin after dequeue
  capacity = min_after_dequeue + 3* batch_size # max number of images which can be stored in bin
  image.set_shape([36, 36, 3]) # set the shape of images in the queue
  key.set_shape([1]) # set the key(label) shape
  example_batch, label_batch = tf.train.shuffle_batch([example, label], batch_size = batch_size, num_thread =4,
                                                       capacity = capacity, min_after_dequeue = min_after_dequeue)
```

Data Pipelines in TensorFlow



```
# How to Read Binary Files using tensorflow ?
with tf.name_scope("input_data"):
  label_bytes = 2 # CIFAR 100 has 0-99 labels #use 1 for CIFAR-10 (0-9)
  height, width, depth = 32, 32, 2
  image_bytes = 32* 32 * 3

  record_bytes = label_bytes + image_bytes

  reader = tf.FixedLengthRecordReader(record_bytes = record_bytes)
  key, value = reader.read(filename_queue) #read filename queue.

  record_bytes = tf.decode_raw(value, tf.uint8)

  label = tf.cast(tf.strided_slice(record_bytes, [0], [label_bytes]), tf.int32)
  depth_major = tf.reshape(tf.strided_slice(record_bytes, [label_bytes],
                                             [label_bytes + image_bytes]), [depth, height, width])
  uint8image = tf.transpose(depth_major, [1, 2, 0])
  image = tf.cast(uint8image, tf.float32)

#Image and label to the file storeage bin - tf.train.shuffle_batch
```

AlexNet

Image Input (224*224*3)	Kernel size = 11 * 11 * 3, stride = 4 pixels	Output size? (55 * 55)
Conv_1 (96 filter Maps)		
Local response normalization	Kernel size = 3 * 3, stride = 2 pixels	
Max Pooling	Kernel size = 5 * 5 * 96, stride = 3 pixels	Output size? (27 * 27)
Conv_2 (256 filter Maps)	Kernel size = 3 * 3, stride = 2 pixels	Output size? (13 * 13)
Max Pooling	Kernel size = 3 * 3 * 256, stride = 1 pixels	Output size? (13 * 13)
Conv_3 (384 filter Maps)	Kernel size = 3 * 3 * 384, stride = 1 pixels	Output size? (13 * 13)
Conv_4 (384 filter Maps)	Kernel size = 3 * 3 * 384, stride = 1 pixels	Output size? (13 * 13)
Conv_5 (256 filter Maps)	Kernel size = 3 * 3 , stride = 2 pixels	Output size? (6 * 6)
Max Pooling	Total Features = 6*6*256 = 9216	
Fully Connected - 4096		
Fully Connected - 4096		
Output - 1000		

Neurons to train in Layer-1 = ((11*11*3)+1)*96 = 34, 944

Fractional Max Pooling

General Pooling reduces the size of feature map by 1/4.

Nin = 2 * Nout

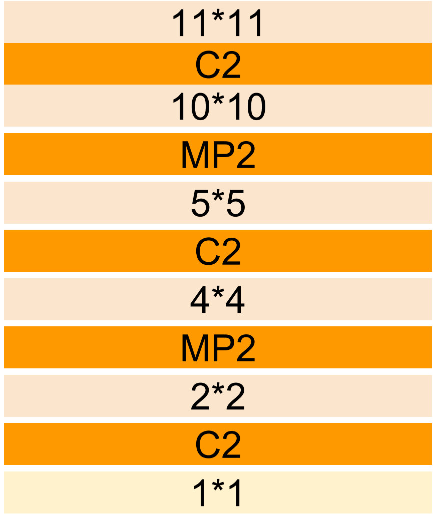
It Quickly reduces the size of feature map thereby not allowing to go deep and learn more features

Nin = 1.414 *Nout (Nearest integer)

Random or Pseudo Random

Disjoint or Overlapping

What happens to the network when using FMP ?



11*11

C2

10*10

MP2

5*5

C2

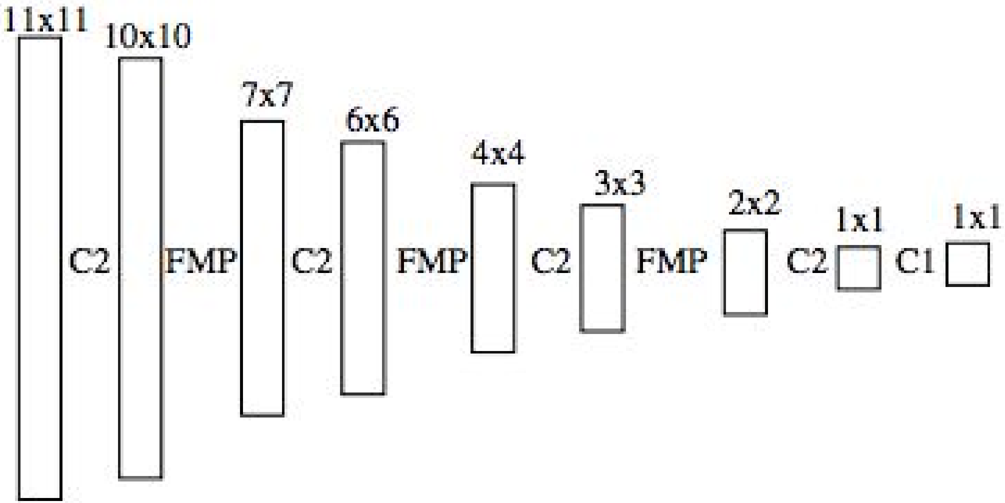
4*4

MP2

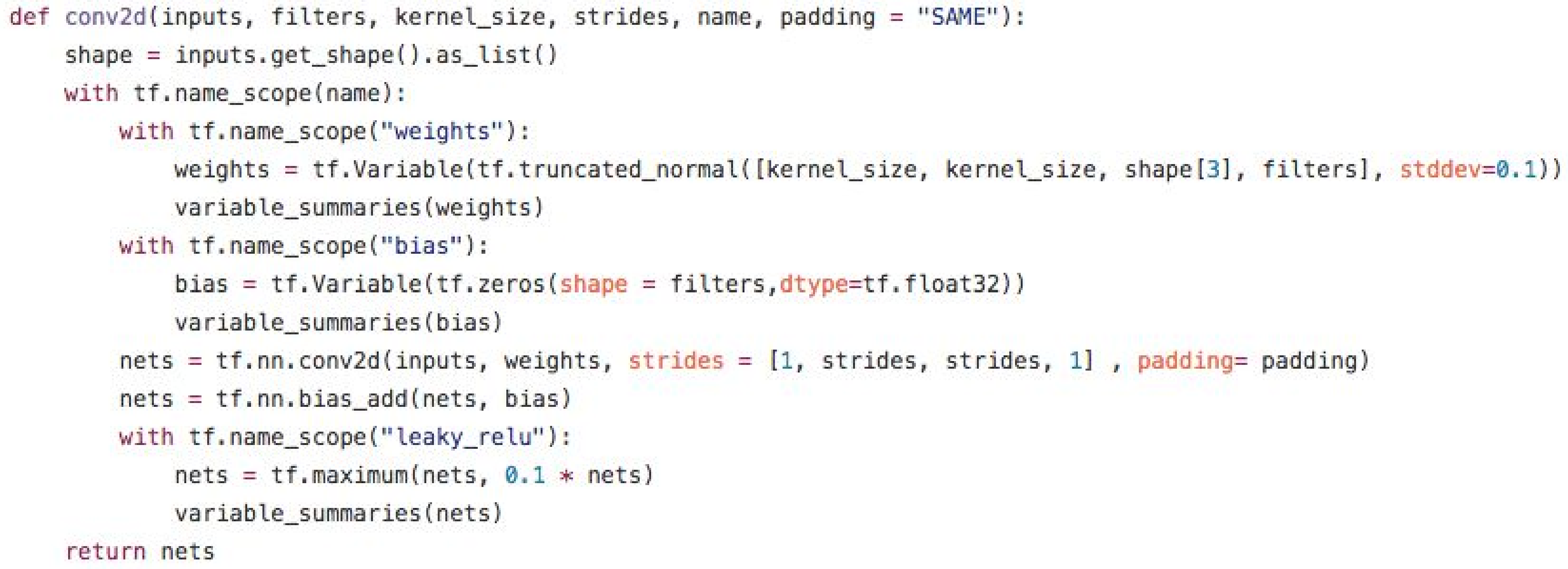
2*2

C2

1*1

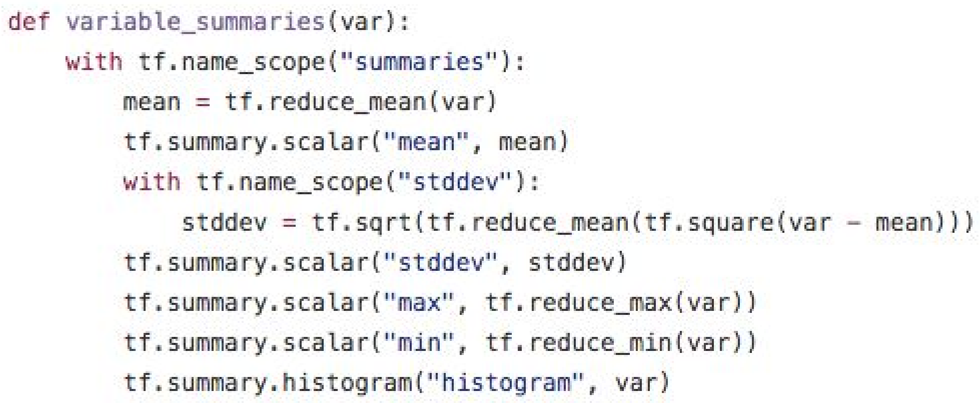


Design Architecture with less verbose code - code reusability



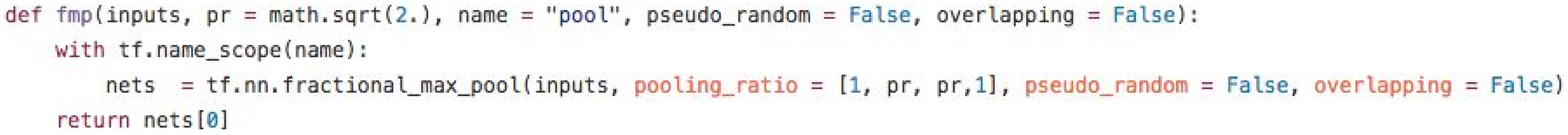
```
def conv2d(inputs, filters, kernel_size, strides, name, padding = "SAME"):
    shape = inputs.get_shape().as_list()
    with tf.name_scope(name):
        with tf.name_scope("weights"):
            weights = tf.Variable(tf.truncated_normal([kernel_size, kernel_size, shape[3], filters], stddev=0.1))
            variable_summaries(weights)
        with tf.name_scope("bias"):
            bias = tf.Variable(tf.zeros(shape = filters,dtype=tf.float32))
            variable_summaries(bias)
        nets = tf.nn.conv2d(inputs, weights, strides = [1, strides, strides, 1] , padding= padding)
        nets = tf.nn.bias_add(nets, bias)
        with tf.name_scope("leaky_relu"):
            nets = tf.maximum(nets, 0.1 * nets)
            variable_summaries(nets)
    return nets
```

Visualization using TensorBoard



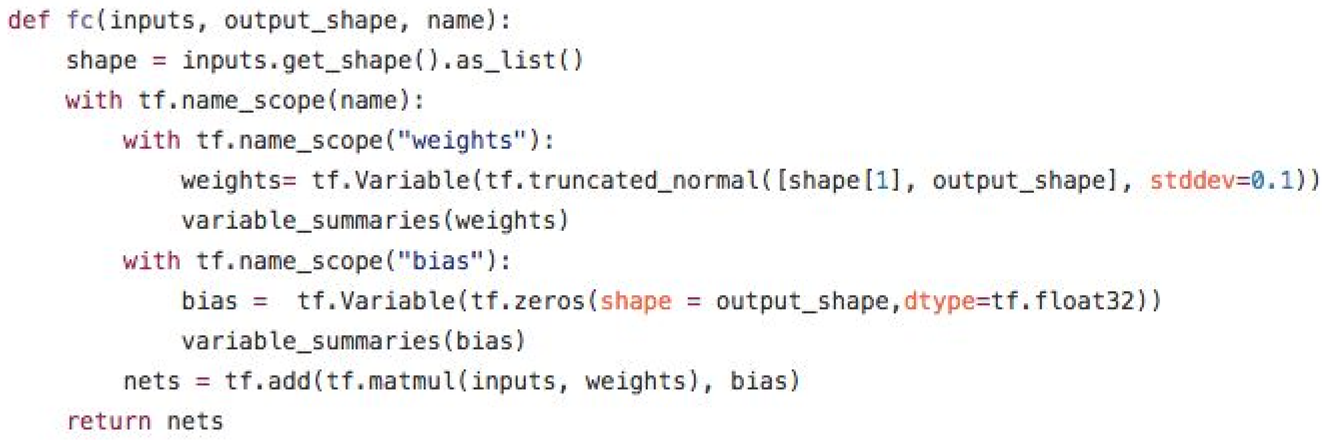
```
def variable_summaries(var):
    with tf.name_scope("summaries"):
        mean = tf.reduce_mean(var)
        tf.summary.scalar("mean", mean)
        with tf.name_scope("stddev"):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
        tf.summary.scalar("stddev", stddev)
        tf.summary.scalar("max", tf.reduce_max(var))
        tf.summary.scalar("min", tf.reduce_min(var))
        tf.summary.histogram("histogram", var)
```

Design Architecture - FMP



```
def fmp(inputs, pr = math.sqrt(2.), name = "pool", pseudo_random = False, overlapping = False):
    with tf.name_scope(name):
        nets  = tf.nn.fractional_max_pool(inputs, pooling_ratio = [1, pr, pr,1], pseudo_random = False, overlapping = False)
    return nets[0]
```

Design Architecture - Fully connected

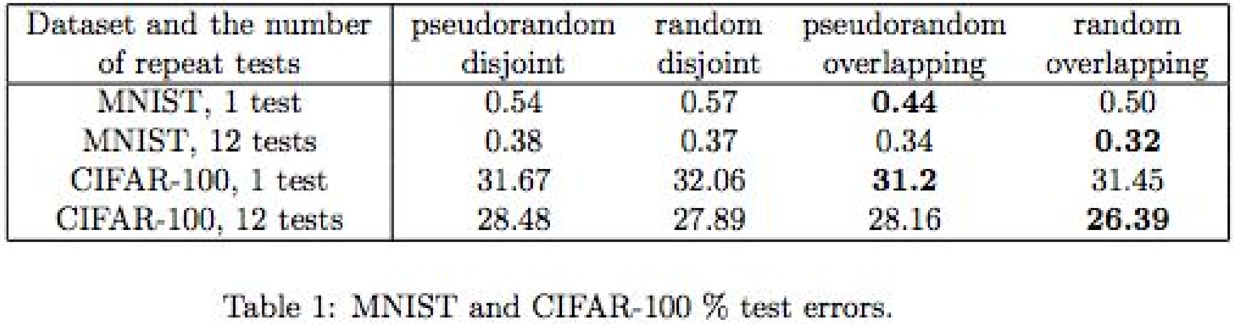


```
def fc(inputs, output_shape, name):
    shape = inputs.get_shape().as_list()
    with tf.name_scope(name):
        with tf.name_scope("weights"):
            weights= tf.Variable(tf.truncated_normal([shape[1], output_shape], stddev=0.1))
            variable_summaries(weights)
        with tf.name_scope("bias"):
            bias =  tf.Variable(tf.zeros(shape = output_shape,dtype=tf.float32))
            variable_summaries(bias)
        nets = tf.add(tf.matmul(inputs, weights), bias)
    return nets
```

Train the network with one line of code.

FMP on MNIST Dataset

FMP on CIFAR-100 Dataset

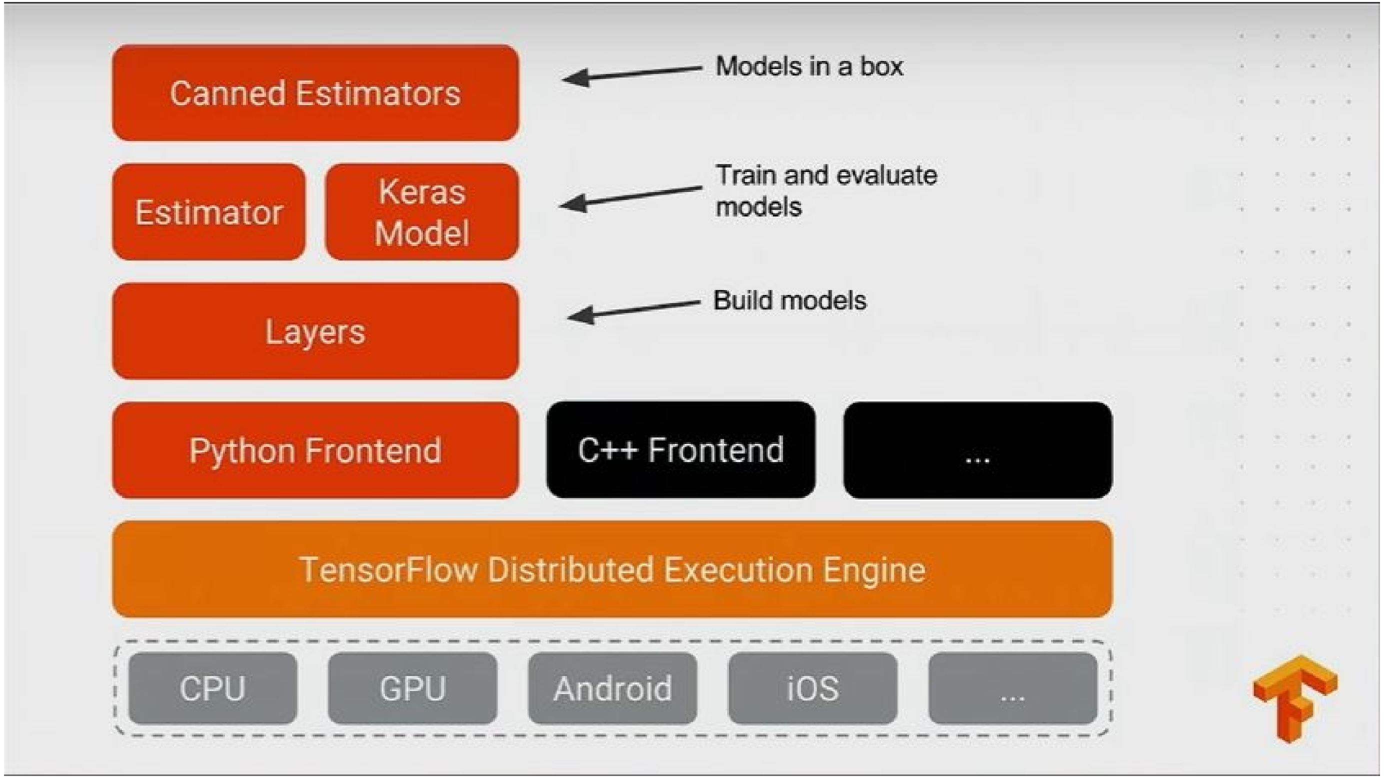


Dataset and the number of repeat tests	pseudorandom disjoint	random disjoint	pseudorandom overlapping	random overlapping
MNIST, 1 test	0.54	0.57	**0.44**	0.50
MNIST, 12 tests	0.38	0.37	0.34	**0.32**
CIFAR-100, 1 test	31.67	32.06	**31.2**	31.45
CIFAR-100, 12 tests	28.48	27.89	28.16	**26.39**

Table 1: MNIST and CIFAR-100 % test errors.

https://arxiv.org/pdf/1412.6071.pdf

Everything under one hood



Canned Estimators

Models in a box

Estimator

Keras Model

Train and evaluate models

Layers

Build models

Python Frontend

C++ Frontend

...

TensorFlow Distributed Execution Engine

CPU

GPU

Android

iOS

...

What we do at AthenaSight …

Questions ?

Reach me at:
Email - vanapaliprakash@gmail.com
Twitter - @14prakash

Thank You