ARTIFICIAL INTELLIGENCE IN MANUFACTURING ME6324 FINAL REVIEW

Computer Vision based sorting system for manufacturing industries

Frameworks Used: OpenCV, Python Dependencies (numpy & pil), TensorFlow

Tools: Feed Forward Neural Network and Image Augmentation

Languages : Python

Ridhi Puppala (ME15B133)

Vishal Chandrahas (ME15B148)

COLLECTION OF DATA

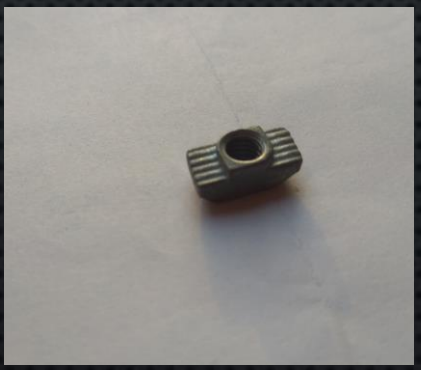
- Due to the novelty of the chosen problem statement there is no available data both in research journals and online platform
- The image data set has been generated by taking sample images of fasteners and other mechanical components using smart phone cameras and the data is populated using image augmentation techniques for training and testing over Neural Networks
- Images were taken in different perspectives, lighting conditions and orientation of the components to simulate the dynamic orientations and lighting conditions of components moving on a conveyor belt

Hex Bolt

Countersunk Bolt

Hex Nut

Bearing



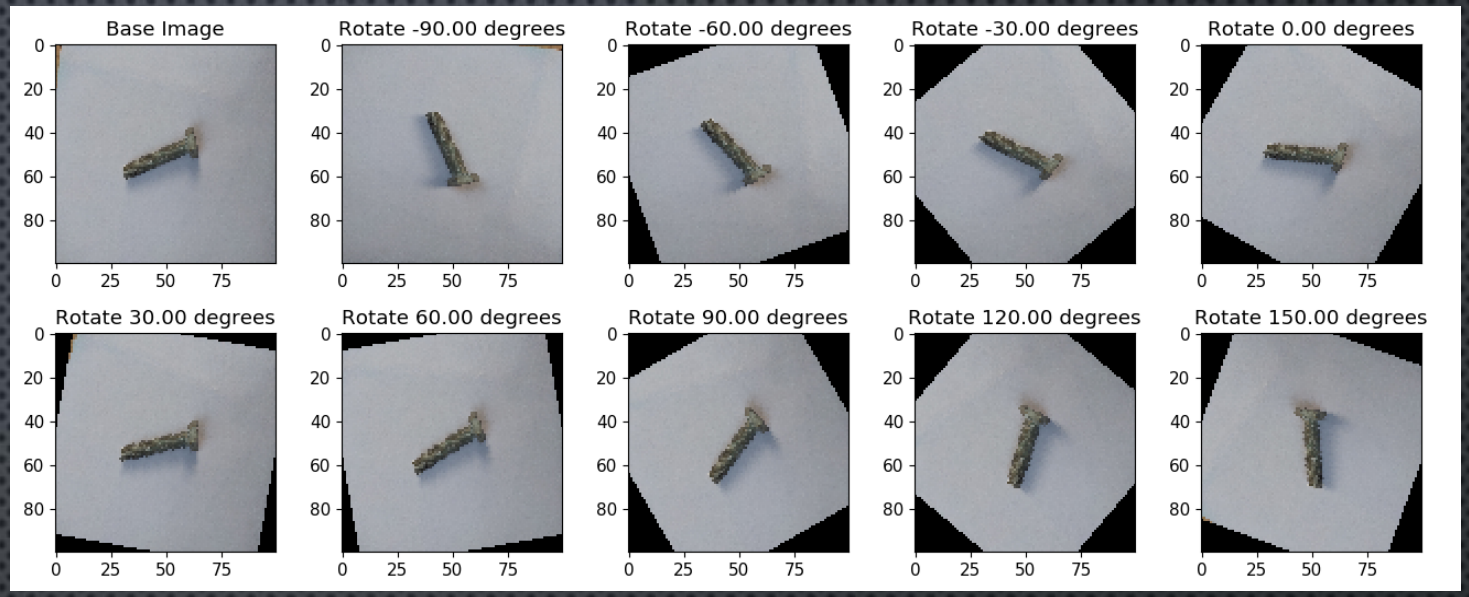
T nut

AUGMENTATION OF IMAGE DATA

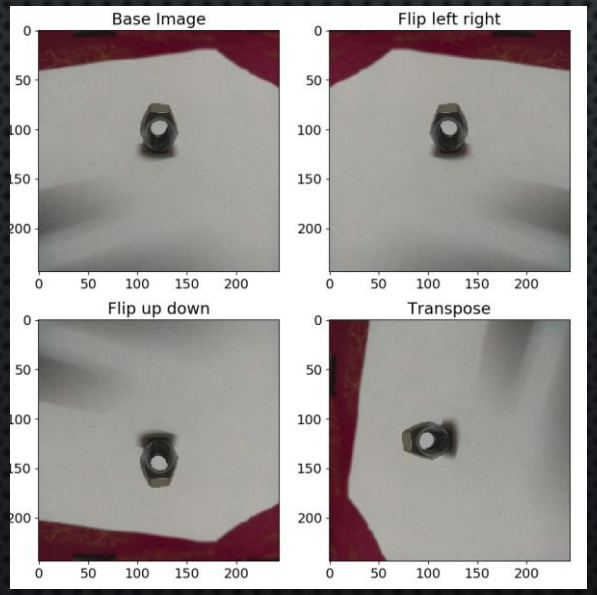
- Tensor Flow and OpenCV (Python) were used to perform image augmentation
- Augmentation involves a combination of affine transformations and addition of white and gaussian noise, blurring and other CV operations
- Images are resized from 4160 X 3120 to 100 X 100 pixels
- Then following Augmentation Operations are applied:
  - Scaling
  - Translation
  - Rotation by coarse and fine angles
  - Flipping
  - Adding Salt and Pepper Noise
  - Lighting Condition (Add Gaussian Noise)
  - Perspective Transform

Affine Transform	Example	Transformation Matrix
Translation		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$
Scale		$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Shear		$\begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation		$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

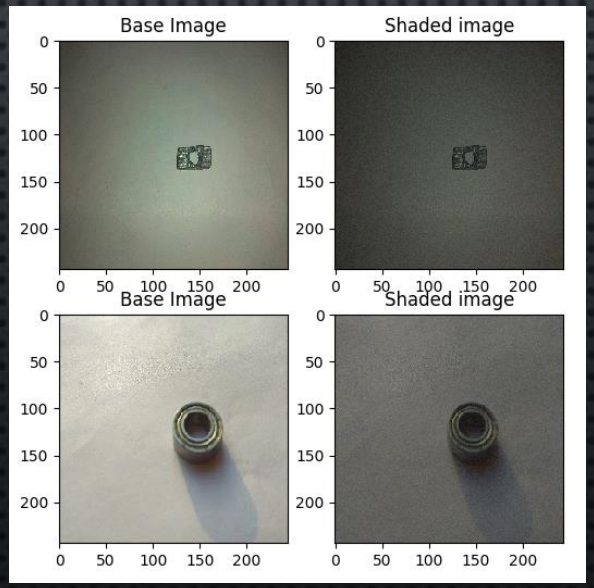
Resized Images (100X100)



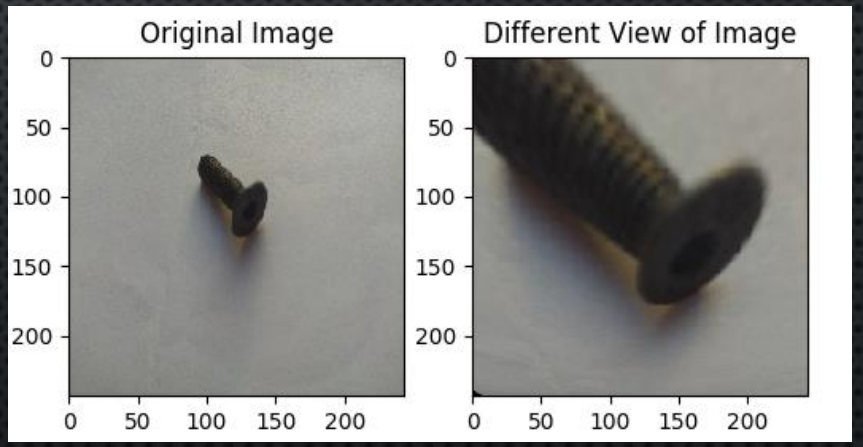
Rotation Transformation through multiples of 30°



Flip Up, Down, and Transpose Transformation



Gaussian Noise for simulating different Lightning conditions



Perspective Transformation to account for image distortion during movement on conveyors

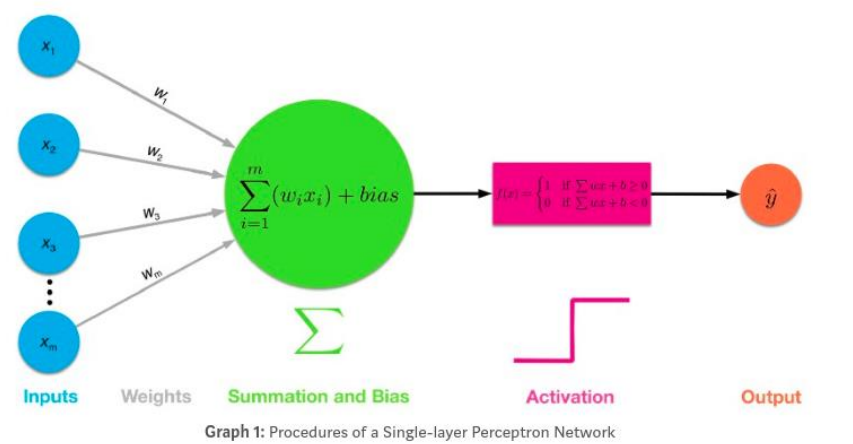
TRAINING OF THE NEURAL NETWORK

- The neural network propagates through a weighted transformation with a bias represented as: y = **W**x + **b** ,where W is weight matrix while x is an array of input data with 784 X 1 dimension

- Back Propagation algorithm is implemented for training the chosen feed forward neural network
- The algorithm calculates the new weights based on the errors in each of the layers by calculating the gradients in weights and biases

```
def back_prop(self,Y3,Y4,H,A,W,B):
    da = [0]*(layers+1)
    db = [0]*(layers+1)
    dw = [0]*(layers+1)
    dh = [0]*(layers+1)
    da[layers] = (Y3-Y4)
    for i in range(layers,-1,-1):
        dw[i] = np.transpose(np.matmul(da[i],np.transpose(H[i])))
        db[i] = da[i]
        dh[i] = np.matmul(W[i],da[i])
        if(i!=0):
            da[i-1] = np.multiply(dh[i],self.grad_sig(A[i-1]))
    return dw,db
```

Summary of Single perceptron NN



Graph 1: Procedures of a Single-layer Perceptron Network

- Then the new weights and biases are updated after the back propagation of error signals

```
for k in range(0,layers+1):
    W[k] = W[k] - eta*dw[k]
    B[k] = B[k] - eta*db[k]
```

CUSTOM DEFINED PYTHON FUNCTIONS

```
def makeNN(self,layers): #call with layers =2
    self.layers =layers
    instancenames = []
    #size =30
    for i in range(0,layers+2):
        instancenames.append(i)
    layer = {name: makelayer(name=name,size=100) for name in instancenames}
    e = []
    f = []
    layer[0].size = 784
    layer[layers+1].size = 5
    for i in range(1,layers+2):
        a =layer[i].size
        c= layer[i-1].size
        e.append(0.01*np.random.randn(c,a))  #weight matrix of dim(784,100)
        f.append(np.zeros((a,1))) #bias
    return e,f
```

```
def sigmoid(self,Q):
    M = []
    for i in Q:
        try:
            res = 1.0 /(1.0 + math.exp(-i))
        except OverflowError:
            res = 0.0
        M.append(res)
    return np.transpose(np.asmatrix(M))

def softmax(self,U):
    return np.exp(U)/float(np.sum(np.exp(U)))
```

```
LOSS = np.sum((Y1-Y2).^2)/2*1500
```

Fig. Mean Squared Error (MSE) is chosen as LOSS function for tuning alpha

```
def feature_normalize(self,R):
    mean = np.mean(R)
    range_val = np.amax(R)-np.amin(R)
    R = (R-mean)/float(np.sqrt(np.var(R)))
    return R

def preprocessing(self,Y):
    #convert to one hot vector for prediction
    Y2 = (np.arange(0,5)==Y).astype(float)
    return np.transpose(np.asmatrix(Y2))

def sig(self,z):
    try:
        res = 1 / float(1 + math.exp(-z))
    except OverflowError:
        res = 0.0
    return res

def grad_sig(self,S):
    L = []
    for i in S:
        L.append(self.sig(i)*(1 - self.sig(i)))
    return np.transpose(np.asmatrix(L))
```

FEEDFORWARD FUNCTION

- Feedforward function is the heart of the neural network
- It takes input images, weight and bias as numpy arrays
- It operates over the network generated using makeNN() function
- H is activation while A is pre activation function for the neurons

```
def feedforward(self,K,W,B):
    #print(K)
    instancenames = []
    for i in range(0,layers+2):
        instancenames.append(i)
    layer = {name: makelayer(name=name,size=100) for name in instancenames}
    H = []
    A = []
    layer[0].size = 784
    layer[layers+1].size = 5
    for i in range(1,layers+2):
        a =layer[i].size
        c= layer[i-1].size
        H.append(np.zeros((c,1)))
        A.append(np.zeros((a,1)))
    H.append(np.zeros((5,1)))
    H[0] = np.transpose(np.asmatrix(K))
    for i in range(0,layers):
        A[i] = B[i] + np.matmul(np.transpose(W[i]),H[i])
        H[i+1] = self.sigmoid((A[i]))
    A[layers] = B[layers] + np.matmul(np.transpose(W[layers]),H[layers])
    Y1 = self.softmax((A[layers]))
    return Y1,H,A
```

OPTIMIZATION AND MAIN FUNCTION

Main function: Loads weights and bias from training sessions and runs feedforward to predict the label of the test data and compute accuracy based on correct labels

```
lays = NN()
layers = 2

W,B = lays.optimization(eta=0.01,epochs=50,batch_size=1500)
np.save('weights.npy',W,allow_pickle = True,fix_imports = True)
np.save('bias.npy',B,allow_pickle = True,fix_imports = True)


for i in range(0,784):
    N[:,i] = lays.feature_normalize(N[:,i])


pred = []
for i in range(len(N)):
    P,Q,R = lays.feedforward(N[i,:],W,B)
    pred.append(np.argmax(P))
accur = lays.accuracy(M,np.asmatrix(pred))
print(accur)
```

Optimization Function: Performs several iterations(epochs) of feedforward, back propagation and weights update step to train the network with the input data

```
def optimization(self,eta,epochs,batch_size):

    X = np.load('dataset.npy')
    class_id = np.load('class_id.npy')
    Y = np.array(class_id)

    W,B = lays.makeNN(layers)
    for i in range(epochs):
        batches = int((1500/(batch_size))+1)
        for o in range(batches-1):
            db = [0]*(layers+1)
            dw = [0]*(layers+1)
            for j in range(o*batch_size,(o+1)*batch_size):
                Y2 = self.preprocessing(Y[j])
                Y1,I,J  = self.feedforward(X[j,:],W,B)
                print(np.argmax(Y1))
                dx,dy= self.back_prop(Y1,Y2,I,J,W,B)
                dw = [sum(x) for x in zip(dw,dx)]
                db = [sum(x) for x in zip(db,dy)]
                LOSS = np.sum((Y1-Y2).^2)/2*15
            for k in range(0,layers+1):
                W[k] = W[k] - eta*dw[k]
                B[k] = B[k] - eta*db[k]
        db = [0]*(layers+1)
        dw = [0]*(layers+1)
        for m in range(batches*batch_size,len(X)+1):
            Y2 = self.preprocessing(Y[m])
            Y1,I,J   = self.feedforward(X[j,:],W,B)
            dl,dk= self.back_prop(Y1,Y2,I,J,W,B)
            print(np.argmax(Y1))
            dw = [sum(x) for x in zip(dw,dl)]
            db = [sum(x) for x in zip(db,dk)]
        for k in range(0,layers+1):
            W[k] = W[k] - eta*dw[k]
            B[k] = B[k] - eta*db[k]
        print("epoch {}".format(i+1))
    return W,B
```

PYTHON NUMPY VS TENSORFLOW APPROACH

Comparision of the training, testing, data processing time for two different approaches

Operation	Python Numpy	Tensor Flow
Training	~350 seconds	~15 seconds
Data Processing	~ 30 seconds	~ 5 seconds
Testing & Implementation	~ 5 seconds	< 2 seconds
No. of Lines of Code	180	30

Conditions

300 Iterations (Epochs)
300 Hidden Layer Neurons
784 input layer Neurons
5 classes (5 output layers)
Learning rate (alpha) = 0.005

TENSOR FLOW CODE

```python
from __future__ import division, print_function, unicode_literals
import tensorflow as tf
from time import time
import numpy as np
from time import sleep

X = np.load('dataset.npy')
Y = np.load('class_id.npy')
y = []
for i in range(len(Y)):
    y.append((np.arange(1,6)==Y[i]))
a = np.reshape(np.asarray(y),(-1,5))

ILN = 784
HLN = 300
alpha = 0.005

x = tf.placeholder(tf.float32, [None, ILN])
W1 = tf.Variable(tf.random_normal([ILN,HLN]))
b1 = tf.Variable(tf.random_normal([HLN]))
W2 = tf.Variable(tf.random_normal([HLN, 5]))
b2 = tf.Variable(tf.random_normal([5]))

h = tf.nn.sigmoid(tf.matmul(x,W1)+b1)
y = tf.nn.softmax(tf.matmul(h, W2) + b2)
y_ = tf.placeholder(tf.float32, [None, 5])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
regularizers = tf.nn.l2_loss(W1)+tf.nn.l2_loss(W2)
train_step = tf.train.AdamOptimizer(alpha).minimize(cross_entropy)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

yt = []
for i in range(len(testY)):
    yt.append((np.arange(1,6)==testY[i]))
at = np.reshape(np.asarray(yt),(-1,5))

h = tf.nn.sigmoid(tf.matmul(x,W1)+b1)
y = tf.nn.softmax(tf.matmul(h, W2) + b2)
for i in range(12):
    sess.run(train_step, feed_dict={x: testX, y_: at})

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))

prediction=tf.argmax(y,1)
print("Predictions: ", prediction.eval(feed_dict={x: testX}, session=sess))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Complete Testing Accuracy: %.4f" % (sess.run(accuracy, feed_dict={x: testX, y_: at})))
```

- Open source machine learning framework
- Flexible architecture allows easy deployment of computation across variety of platforms with inter-operability
- CPUs, GPUs, TPUs ,desktops, clusters of servers, mobile and edge devices
- Inbuilt functions with highly optimized performance and less computation time
- Save and Restore sessions of training allow us to keep training and testing at any time

After rigorous tuning of the parameters, the following were chosen:

- Learning Rate = 0.005
- No. of hidden layers = 1
- Hidden layer Neuron = 300
- Iteration (Epochs) = 300
- Input dataset size = 1500
- Input layer Neurons = 784

PERFORMANCE STATISTICS

```
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-04-20 18:26:41.548210: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that
Hidden Layer Neurons: 300       Learn Rate: 0.0010      Epochs: 100  Train Time: 3.0303         Accuracy: 0.5493
Hidden Layer Neurons: 300       Learn Rate: 0.0020      Epochs: 100  Train Time: 2.9282         Accuracy: 0.6073
Hidden Layer Neurons: 300       Learn Rate: 0.0030      Epochs: 100  Train Time: 2.8694         Accuracy: 0.6273
Hidden Layer Neurons: 300       Learn Rate: 0.0040      Epochs: 100  Train Time: 3.0239         Accuracy: 0.6720
Hidden Layer Neurons: 300       Learn Rate: 0.0050      Epochs: 100  Train Time: 2.9105         Accuracy: 0.7600
Hidden Layer Neurons: 300       Learn Rate: 0.0060      Epochs: 100  Train Time: 3.0385         Accuracy: 0.6827
Hidden Layer Neurons: 300       Learn Rate: 0.0070      Epochs: 100  Train Time: 2.9022         Accuracy: 0.6513
Hidden Layer Neurons: 300       Learn Rate: 0.0080      Epochs: 100  Train Time: 3.0124         Accuracy: 0.6640
Hidden Layer Neurons: 300       Learn Rate: 0.0090      Epochs: 100  Train Time: 2.8774         Accuracy: 0.6447
Hidden Layer Neurons: 300       Learn Rate: 0.0100      Epochs: 100  Train Time: 3.0163         Accuracy: 0.5927
Hidden Layer Neurons: 300       Learn Rate: 0.0110      Epochs: 100  Train Time: 3.0046         Accuracy: 0.6160
Hidden Layer Neurons: 300       Learn Rate: 0.0120      Epochs: 100  Train Time: 2.8823         Accuracy: 0.6533
Hidden Layer Neurons: 300       Learn Rate: 0.0130      Epochs: 100  Train Time: 3.0433         Accuracy: 0.4967
Hidden Layer Neurons: 300       Learn Rate: 0.0140      Epochs: 100  Train Time: 3.0313         Accuracy: 0.4773
Hidden Layer Neurons: 300       Learn Rate: 0.0150      Epochs: 100  Train Time: 2.8625         Accuracy: 0.5220
```

Epochs: 100
Alpha: Variable
HLN : 300
Train Time : ~2.8s

```
2018-04-20 18:28:44.632054: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that
Hidden Layer Neurons: 500       Learn Rate: 0.0010      Epochs: 100  Train Time: 4.6387         Accuracy: 0.7327
Hidden Layer Neurons: 500       Learn Rate: 0.0020      Epochs: 100  Train Time: 4.6614         Accuracy: 0.7260
Hidden Layer Neurons: 500       Learn Rate: 0.0030      Epochs: 100  Train Time: 4.5195         Accuracy: 0.7407
Hidden Layer Neurons: 500       Learn Rate: 0.0040      Epochs: 100  Train Time: 4.6522         Accuracy: 0.8233
Hidden Layer Neurons: 500       Learn Rate: 0.0050      Epochs: 100  Train Time: 4.5327         Accuracy: 0.8367
Hidden Layer Neurons: 500       Learn Rate: 0.0060      Epochs: 100  Train Time: 4.5542         Accuracy: 0.8767
Hidden Layer Neurons: 500       Learn Rate: 0.0070      Epochs: 100  Train Time: 4.6766         Accuracy: 0.7520
Hidden Layer Neurons: 500       Learn Rate: 0.0080      Epochs: 100  Train Time: 4.5263         Accuracy: 0.7293
Hidden Layer Neurons: 500       Learn Rate: 0.0090      Epochs: 100  Train Time: 4.7056         Accuracy: 0.7567
Hidden Layer Neurons: 500       Learn Rate: 0.0100      Epochs: 100  Train Time: 4.7433         Accuracy: 0.6827
Hidden Layer Neurons: 500       Learn Rate: 0.0110      Epochs: 100  Train Time: 4.6867         Accuracy: 0.5747
Hidden Layer Neurons: 500       Learn Rate: 0.0120      Epochs: 100  Train Time: 4.5985         Accuracy: 0.6233
Hidden Layer Neurons: 500       Learn Rate: 0.0130      Epochs: 100  Train Time: 4.6968         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0140      Epochs: 100  Train Time: 4.5422         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0150      Epochs: 100  Train Time: 4.7464         Accuracy: 0.2000
```

Epochs: 100
Alpha: Variable
HLN : 500
Train Time : ~4.5s

EFFECT OF HLN AFTER SATURATION

```
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-04-20 18:39:15.901146: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that th
Hidden Layer Neurons: 500       Learn Rate: 0.0010      Epochs: 200  Train Time: 9.9356         Accuracy: 0.7907
Hidden Layer Neurons: 500       Learn Rate: 0.0020      Epochs: 200  Train Time: 9.3103         Accuracy: 0.8980
Hidden Layer Neurons: 500       Learn Rate: 0.0030      Epochs: 200  Train Time: 9.7607         Accuracy: 0.9827
Hidden Layer Neurons: 500       Learn Rate: 0.0040      Epochs: 200  Train Time: 10.4142        Accuracy: 0.9627
Hidden Layer Neurons: 500       Learn Rate: 0.0050      Epochs: 200  Train Time: 9.1304         Accuracy: 0.9880
Hidden Layer Neurons: 500       Learn Rate: 0.0060      Epochs: 200  Train Time: 9.5894         Accuracy: 0.9840
Hidden Layer Neurons: 500       Learn Rate: 0.0070      Epochs: 200  Train Time: 9.9635         Accuracy: 0.9720
Hidden Layer Neurons: 500       Learn Rate: 0.0080      Epochs: 200  Train Time: 10.0899        Accuracy: 0.9253
Hidden Layer Neurons: 500       Learn Rate: 0.0090      Epochs: 200  Train Time: 10.3131        Accuracy: 0.9500
Hidden Layer Neurons: 500       Learn Rate: 0.0100      Epochs: 200  Train Time: 9.3635         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0110      Epochs: 200  Train Time: 9.9588         Accuracy: 0.9067
Hidden Layer Neurons: 500       Learn Rate: 0.0120      Epochs: 200  Train Time: 9.8264         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0130      Epochs: 200  Train Time: 9.2552         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0140      Epochs: 200  Train Time: 9.0616         Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0150      Epochs: 200  Train Time: 9.3080         Accuracy: 0.2000
```

Epochs: 200
Alpha: Variable
HLN : 500
Train Time : ~9s

```
2018-04-20 18:45:06.828821: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions tha
Hidden Layer Neurons: 300       Learn Rate: 0.0010      Epochs: 200  Train Time: 6.8108         Accuracy: 0.6647
Hidden Layer Neurons: 300       Learn Rate: 0.0020      Epochs: 200  Train Time: 6.0033         Accuracy: 0.8693
Hidden Layer Neurons: 300       Learn Rate: 0.0030      Epochs: 200  Train Time: 5.7176         Accuracy: 0.8673
Hidden Layer Neurons: 300       Learn Rate: 0.0040      Epochs: 200  Train Time: 6.0015         Accuracy: 0.9587
Hidden Layer Neurons: 300       Learn Rate: 0.0050      Epochs: 200  Train Time: 5.8507         Accuracy: 0.9060
Hidden Layer Neurons: 300       Learn Rate: 0.0060      Epochs: 200  Train Time: 6.0151         Accuracy: 0.8513
Hidden Layer Neurons: 300       Learn Rate: 0.0070      Epochs: 200  Train Time: 6.6751         Accuracy: 0.8920
Hidden Layer Neurons: 300       Learn Rate: 0.0080      Epochs: 200  Train Time: 6.7715         Accuracy: 0.9360
Hidden Layer Neurons: 300       Learn Rate: 0.0090      Epochs: 200  Train Time: 8.3853         Accuracy: 0.9020
Hidden Layer Neurons: 300       Learn Rate: 0.0100      Epochs: 200  Train Time: 8.0361         Accuracy: 0.8107
Hidden Layer Neurons: 300       Learn Rate: 0.0110      Epochs: 200  Train Time: 6.1647         Accuracy: 0.7613
Hidden Layer Neurons: 300       Learn Rate: 0.0120      Epochs: 200  Train Time: 6.5554         Accuracy: 0.6947
Hidden Layer Neurons: 300       Learn Rate: 0.0130      Epochs: 200  Train Time: 6.1430         Accuracy: 0.7607
Hidden Layer Neurons: 300       Learn Rate: 0.0140      Epochs: 200  Train Time: 6.1905         Accuracy: 0.8133
Hidden Layer Neurons: 300       Learn Rate: 0.0150      Epochs: 200  Train Time: 6.1675         Accuracy: 0.6420
```

Epochs: 200
Alpha: Variable
HLN : 300
Train Time : ~6s

REDUNDANCY OF HIGHER HLN AFTER SATURATION

```
Hidden Layer Neurons: 300       Learn Rate: 0.0010      Epochs: 300  Train Time: 10.1654        Accuracy: 0.7773
Hidden Layer Neurons: 300       Learn Rate: 0.0020      Epochs: 300  Train Time: 10.6538        Accuracy: 0.9427
Hidden Layer Neurons: 300       Learn Rate: 0.0030      Epochs: 300  Train Time: 9.1578         Accuracy: 0.9553
Hidden Layer Neurons: 300       Learn Rate: 0.0040      Epochs: 300  Train Time: 11.3208        Accuracy: 0.9933
Hidden Layer Neurons: 300       Learn Rate: 0.0050      Epochs: 300  Train Time: 8.7323         Accuracy: 0.9753
Hidden Layer Neurons: 300       Learn Rate: 0.0060      Epochs: 300  Train Time: 10.0187        Accuracy: 0.9807
Hidden Layer Neurons: 300       Learn Rate: 0.0070      Epochs: 300  Train Time: 10.5762        Accuracy: 0.9773
Hidden Layer Neurons: 300       Learn Rate: 0.0080      Epochs: 300  Train Time: 11.7176        Accuracy: 0.9740
Hidden Layer Neurons: 300       Learn Rate: 0.0090      Epochs: 300  Train Time: 10.9246        Accuracy: 0.9113
Hidden Layer Neurons: 300       Learn Rate: 0.0100      Epochs: 300  Train Time: 11.5129        Accuracy: 0.9660
Hidden Layer Neurons: 300       Learn Rate: 0.0110      Epochs: 300  Train Time: 9.7121         Accuracy: 0.8587
Hidden Layer Neurons: 300       Learn Rate: 0.0120      Epochs: 300  Train Time: 9.7660         Accuracy: 0.8647
Hidden Layer Neurons: 300       Learn Rate: 0.0130      Epochs: 300  Train Time: 9.7152         Accuracy: 0.8113
Hidden Layer Neurons: 300       Learn Rate: 0.0140      Epochs: 300  Train Time: 10.0045        Accuracy: 0.7033
Hidden Layer Neurons: 300       Learn Rate: 0.0150      Epochs: 300  Train Time: 10.4263        Accuracy: 0.8393
```

Epochs: 300
Alpha: Variable
HLN : 300
Train Time : ~10s

```
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-04-20 18:31:49.209577: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that th
Hidden Layer Neurons: 500       Learn Rate: 0.0010      Epochs: 300  Train Time: 14.0821        Accuracy: 0.9547
Hidden Layer Neurons: 500       Learn Rate: 0.0020      Epochs: 300  Train Time: 14.0597        Accuracy: 0.9720
Hidden Layer Neurons: 500       Learn Rate: 0.0030      Epochs: 300  Train Time: 14.1055        Accuracy: 0.9840
Hidden Layer Neurons: 500       Learn Rate: 0.0040      Epochs: 300  Train Time: 13.5465        Accuracy: 0.9987
Hidden Layer Neurons: 500       Learn Rate: 0.0050      Epochs: 300  Train Time: 14.6362        Accuracy: 0.9927
Hidden Layer Neurons: 500       Learn Rate: 0.0060      Epochs: 300  Train Time: 14.9800        Accuracy: 0.9920
Hidden Layer Neurons: 500       Learn Rate: 0.0070      Epochs: 300  Train Time: 14.6432        Accuracy: 0.9967
Hidden Layer Neurons: 500       Learn Rate: 0.0080      Epochs: 300  Train Time: 14.9446        Accuracy: 0.9993
Hidden Layer Neurons: 500       Learn Rate: 0.0090      Epochs: 300  Train Time: 14.6024        Accuracy: 0.9927
Hidden Layer Neurons: 500       Learn Rate: 0.0100      Epochs: 300  Train Time: 15.6943        Accuracy: 0.9760
Hidden Layer Neurons: 500       Learn Rate: 0.0110      Epochs: 300  Train Time: 17.7786        Accuracy: 0.9527
Hidden Layer Neurons: 500       Learn Rate: 0.0120      Epochs: 300  Train Time: 17.8016        Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0130      Epochs: 300  Train Time: 16.4527        Accuracy: 0.9333
Hidden Layer Neurons: 500       Learn Rate: 0.0140      Epochs: 300  Train Time: 14.1160        Accuracy: 0.2000
Hidden Layer Neurons: 500       Learn Rate: 0.0150      Epochs: 300  Train Time: 15.6811        Accuracy: 0.2000
```

Epochs: 300
Alpha: Variable
HLN : 500
Train Time : ~16s

EFFECT OF TUNING PARAMETERS ON PERFORMANCE

Parameters : Learning Rate(alpha), HLN, ILN, No . Of classes, Train Dataset Size, Epochs

Epochs

- Testing accuracy improves, but saturates after a certain point
- Increases train time proportionally

HLN

- Testing accuracy improves, but saturates after a certain point
- Increases train time
- Capable of handling higher volume of data

No. of Classes

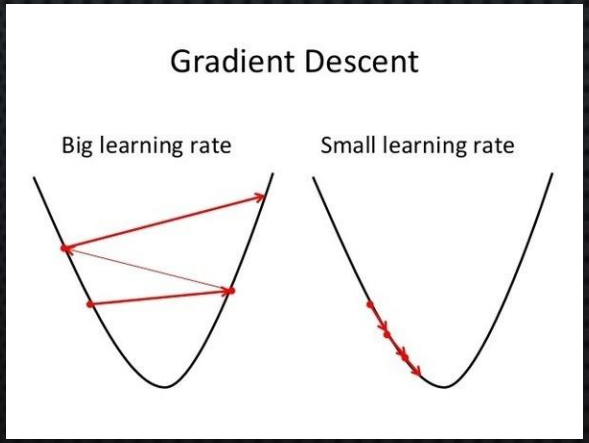
- Improves testing and training accuracy
- Increases train time and volume of input data
- Requires more number of hidden layers and efficient probability distribution function

ILN

- Better prediction against higher variation
- Requires more data for better performance
- Increases train time exponentially
- Requires more number of hidden layers

Learning Rate (alpha)

- Training accuracy and prediction are highly sensitive to alpha
- Small alpha implies slower gradient descent, large alpha leads to overshoot of minimum LOSS and divergence, the cost function chosen was MSE
- Training started from a relatively large alpha and then alpha was decreased during the training to allow more fine-grained weight updates.



CROPPING AND GRAYSCALE CONVERSION

Why Cropping ?

- Presence of too many white pixels in train and test images
- Resizing of image from (4160X3120) to (28,28) suppresses the features of the images making it difficult for the network to train and predict over the dataset
- When resized all the images except Hex bolt shrinks into a blob of grey pixels

Fig. Incorrect prediction of T - nut as a counter sunk bolt when not cropped

Why Grayscale Conversion ?

- Feedforward network operates over single layer of perceptron
- RGB images have 3 channels which are a 3D matrix with depth
- Input layer of this network has 784 neurons for each of the pixels
- Each pixels is fed in as a feature
- CNNs can be used for operating over higher volumes of colour image dataset for feature extraction

DISCUSSIONS ON PERFORMANCE RESULTS

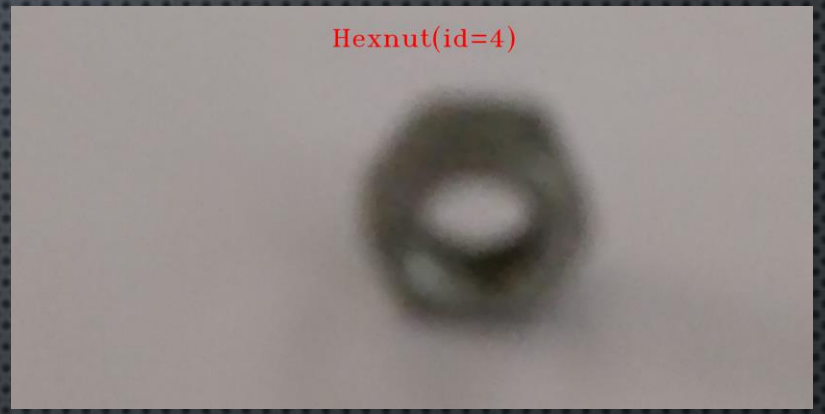
- Initialization of arrays to array of zeros led to degraded training and prediction
- randn() function generates a random array over gaussian distribution
- Prediction accuracy is not consistent for a for a few images, while it is consistent around 90% for large test image datasets
- Classification can be improved with higher volume of train data, better computational resources (GPUs), using CNNs for convoluting over test data for specific feature extraction

```
WARNING:tensorflow:From /usr/local/lib/python3.4/dist-packages/tensorflow/py
ize_all_variables (from tensorflow.python.ops.variables) is deprecated and w
Instructions for updating:
Use `tf.global_variables_initializer` instead.
================= Loading Weights and Bias ======================
================= Predicting the Classes for Test Images ==================
Predicted Labels:  [1 1 1 2 1 2 3 3 4 4 4 4 5 3]
Testing Accuracy based on Test input labels: 0.8571
ridhi@ridhi-Inspiron-5558:~/AI/final$ python3 resultdisplay.py
['./testimages/0.jpg', './testimages/1.jpg', './testimages/2.jpg', './testi
estimages/5.jpg', './testimages/6.jpg', './testimages/7.jpg', './testimages/
ages/10.jpg', './testimages/11.jpg', './testimages/12.jpg', './testimages/13
 ======================Cropping the Test Images ========================
========Generating a single array of Test images after Resizing and Grayscal
['./cropimgs_png/0.png', './cropimgs_png/1.png', './cropimgs_png/2.png', './
ng', './cropimgs_png/5.png', './cropimgs_png/6.png', './cropimgs_png/7.png',
g/9.png', './cropimgs_png/10.png', './cropimgs_png/11.png', './cropimgs_png/
2018-04-24 06:02:22.447591: I tensorflow/core/platform/cpu_feature_guard.cc:
t this TensorFlow binary was not compiled to use: AVX2 FMA
Dimensions of final array for Feedforward Input (Prediction)
(14, 784)
WARNING:tensorflow:From /usr/local/lib/python3.4/dist-packages/tensorflow/py
ize_all_variables (from tensorflow.python.ops.variables) is deprecated and w
Instructions for updating:
Use `tf.global_variables_initializer` instead.
================= Loading Weights and Bias ======================
================= Predicting the Classes for Test Images ==================
Predicted Labels:  [1 1 1 5 5 5 5 4 5 4 5 4 5 5]
Testing Accuracy based on Test input labels: 0.5000
ridhi@ridhi-Inspiron-5558:~/AI/final$
```

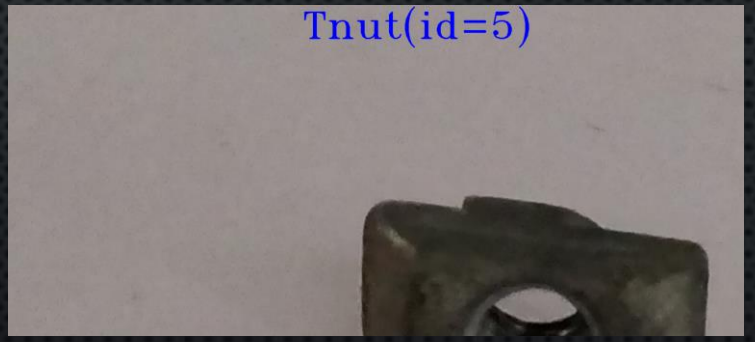
```
2018-04-21 00:39:20.303545: I tensorflow/core/platform/cpu_fea
t this TensorFlow binary was not compiled to use: AVX2 FMA
Starting the training...
The training took 11.0446 seconds.
Accuracy: 0.9933
training Done
Starting testing...
predictions [0 1 0 3 4 2 2 4 1 0 3 0 0 0]
Complete Testing Accuracy: 0.9527
ridhi@ridhi-Inspiron-5558:~/AI/main/testdata$ python3 testing.
WARNING:tensorflow:From /usr/local/lib/python3.4/dist-packages
ize_all_variables (from tensorflow.python.ops.variables) is de
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-04-21 00:39:53.844898: I tensorflow/core/platform/cpu_fea
t this TensorFlow binary was not compiled to use: AVX2 FMA
Starting the training...
The training took 10.9923 seconds.
Accuracy: 0.9933
training Done
Starting testing...
predictions [0 0 0 3 0 0 3 0 3 1 3 1 3 0]
Complete Testing Accuracy: 0.8800
ridhi@ridhi-Inspiron-5558:~/AI/main/testdata$
```

CLASSIFICATION OF FADED & CROPPED IMAGES

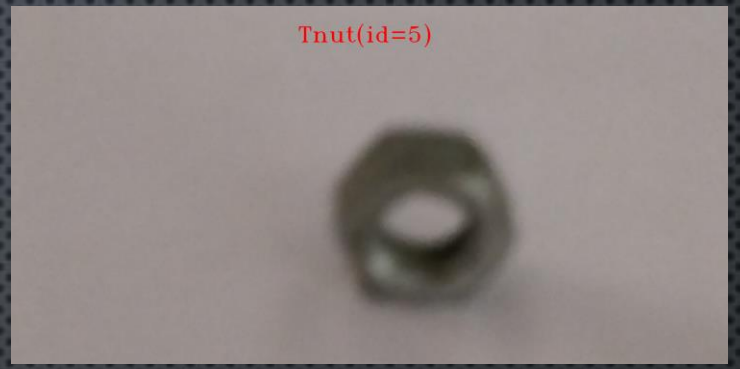
Faded test images



Test images with the component slightly cropped out

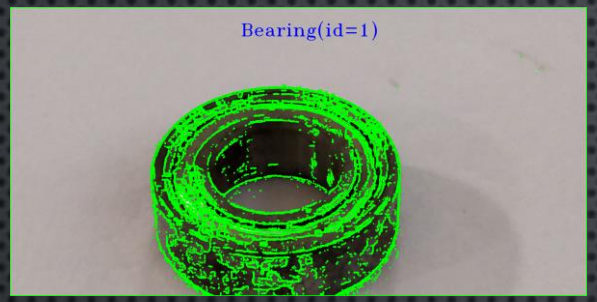


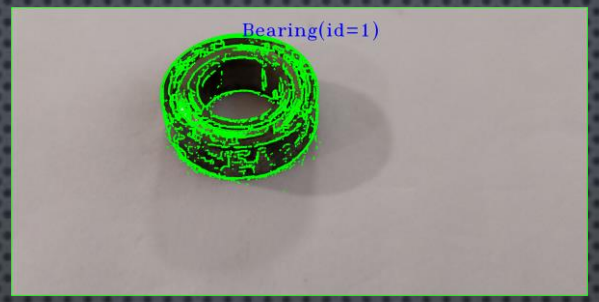
RESULTS WITH HLN = 100

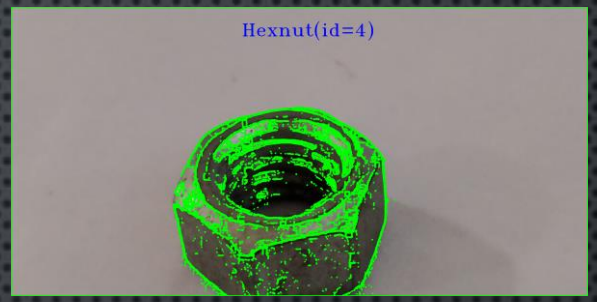


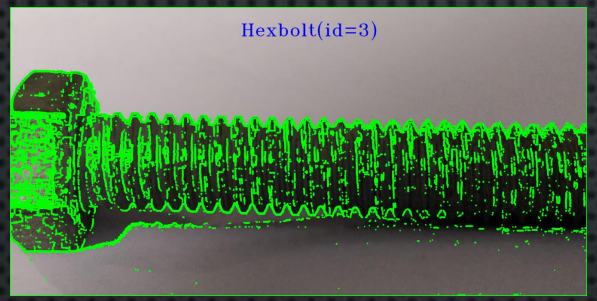
DEGRADED PERFORMANCE WITH LESS EPOCHS

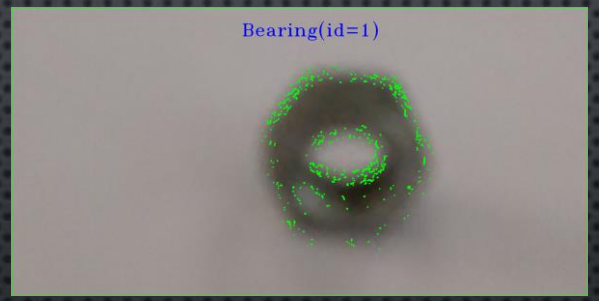
CONTOUR DETECTION & CLASSIFICATION

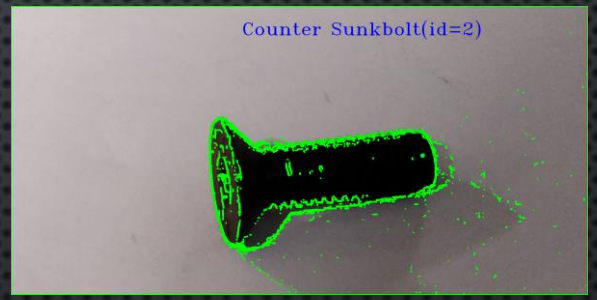


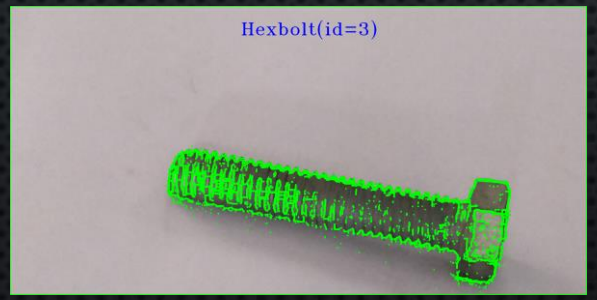


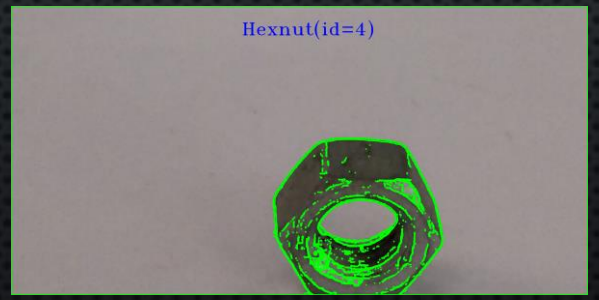


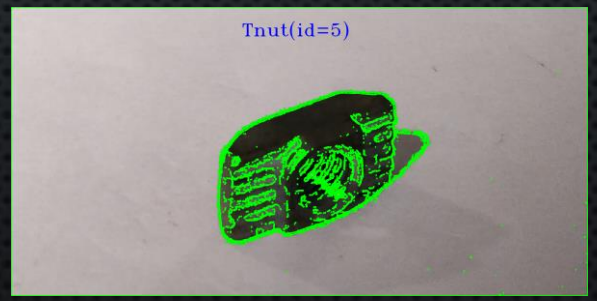




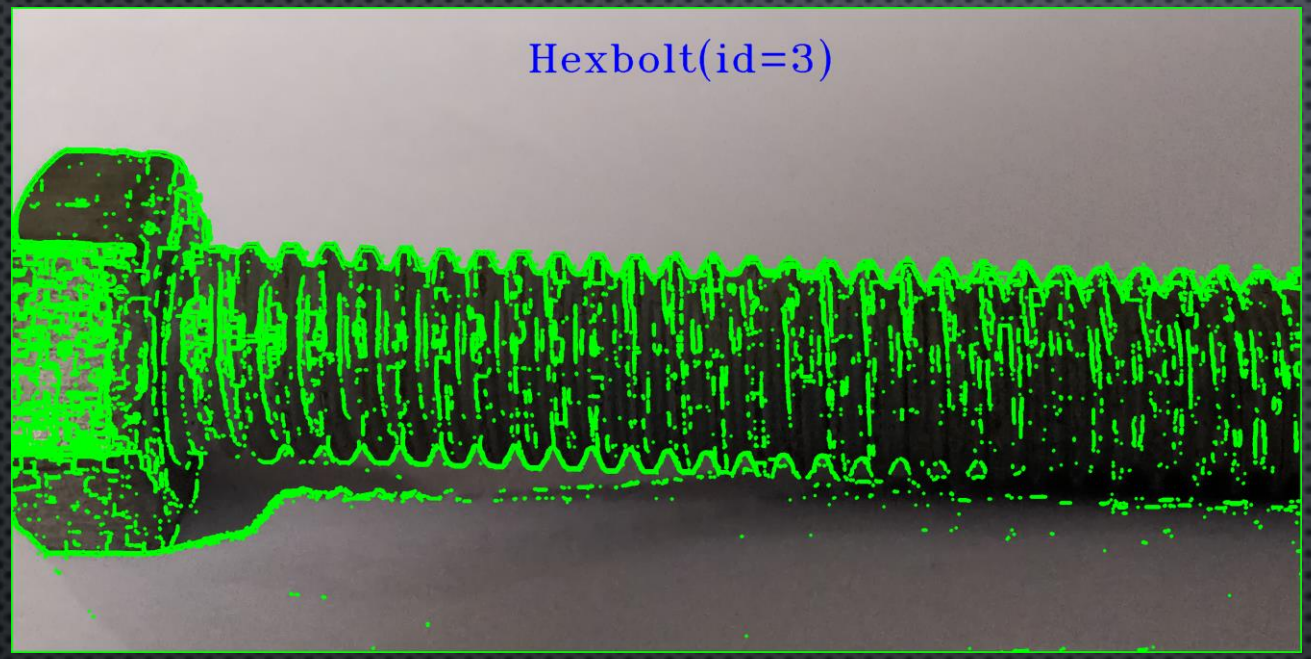


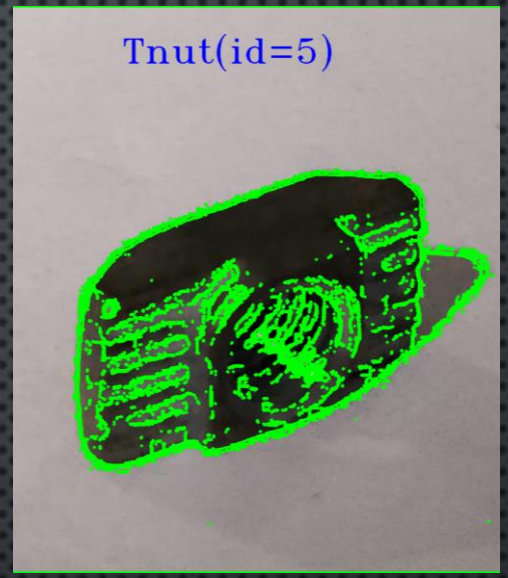


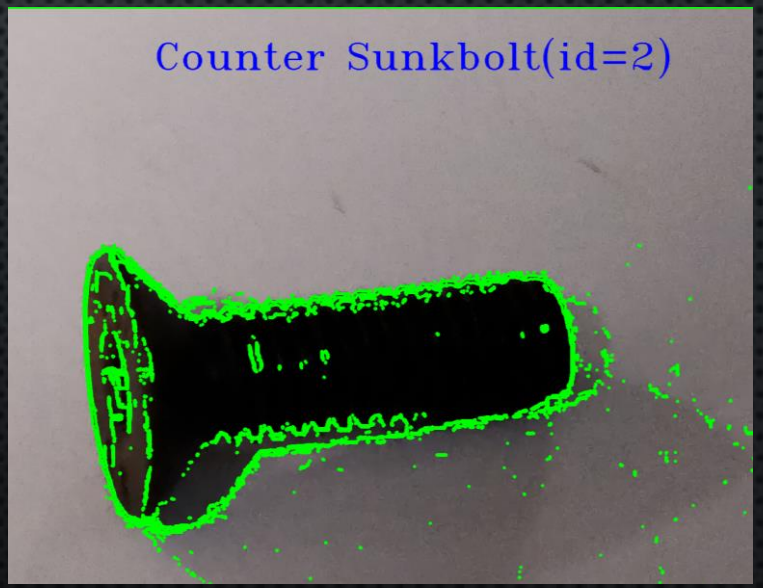




CONTOUR DETECTION - FEATURE EXTRACTION







- Contour Detection can be implemented post classification by running specific OpenCV scripts to search for features (feature extraction)
- Features can help identify the sub categories of the family of components (ex: types of bearings, fasteners)
- Canny Edge detection & contour detection can be deployed under calibrated environments for extracting simple dimensional information like bore, OD, thread length, flange dimensions and other dimensions

SCHEMATIC OF AN INDUSTRIAL SORTING SYSTEM

Training Data

Computer and UI

Training Data Processing, Compression and Labelling

Training Neural Network (CPU + GPU)

Dynamic Training

Data Processing

Feed Forward NN Classifier operating over Trained Model

High Quality Video Input

Classification & Feature Extraction

PLC system for automated tossing

Record dimensional, component and feature information

OUTCOMES

- Feed forward Neural Network Classifier for predicting the type of component
- Populated higher volumes of data through Image Augmentation
- Developed codes for Python Numpy and Tensor Flow approach
- Comparision of custom defined Python functions and Tensor Flow
- Possible methods for extracting features and dimensional information
- Proposed schematic for a complete setup of Industrial sorting system

LEARNING OUTCOMES

- Tensor Flow & Python (Numpy)
- Image Augmentation
- Structure of Neural Networks
- OpenCV Python