In [1]:

```
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
from random import shuffle
import tensorflow as tf
import pandas as pd
from sklearn.metrics import roc_auc_score
```

In [2]:

```
epochs = 50
step_size = 8
IMG_SIZE_ALEXNET = 227 # image size
validating_size = 20 # while cross validating, we are evaluating batch by batch
nodes_fc1 = 4096 # no of nodes on fc layer 1
nodes_fc2 = 4096 # no of nodes on fc layer 2
output_classes = 3 # three classes: eggplant,
output_locations = 4 # minx, miny, maxx, maxy

TRAIN_DIR = os.getcwd()
```

In [3]:

```
data = np.load('object_localization.npy')
train = data[:int(len(data)*0.8)]
cv = data[int(len(data)*0.8):]

#print(train)
# X for train input
X = np.array([i[0] for i in train]).reshape(-1,IMG_SIZE_ALEXNET,
IMG_SIZE_ALEXNET,3)
#X = np.array([i[0] for i in train])
#X = Xa.reshape(-1,IMG_SIZE_ALEXNET,IMG_SIZE_ALEXNET,3)
# Y1 for classification head
Y1 = np.array([i[1] for i in train])
# Y2 for regression head
Y2 = np.array([i[2] for i in train])

# cv_x for train input
cv_x = np.array([i[0] for i in cv]).reshape(-1,IMG_SIZE_ALEXNET,
IMG_SIZE_ALEXNET,3)
# cv_y1 for classification head
cv_y1 = np.array([i[1] for i in cv])
# cv_y2 for regression head
cv_y2 = np.array([i[2] for i in cv])

print(cv_y1[:10])
print(X.shape)

print(Y1.shape)

print(cv_x.shape)

print(cv_y1.shape)
```

```
[[0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]
 [0 0 1]]
(21196, 227, 227, 3)
(21196, 3)
(5300, 227, 227, 3)
(5300, 3)
```

In [4]:

```
steps = len(train)
print(steps)
remaining = steps % step_size

#Resetting graph
tf.reset_default_graph()

#Defining Placeholders
x = tf.placeholder(tf.float32,shape=[None,IMG_SIZE_ALEXNET,IMG_SIZE_ALEXNET,3])
y_true_1 = tf.placeholder(tf.float32,shape=[None,output_classes])
y_true_2 = tf.placeholder(tf.float32,shape=[None,output_locations])

##CONVOLUTION LAYER 1
#Weights for layer 1
w_1 = tf.Variable(tf.truncated_normal([11,11,3,96], stddev=0.01))
#Bias for layer 1
b_1 = tf.Variable(tf.constant(0.0, shape=[[11,11,3,96][3]]))
#Applying convolution
c_1 = tf.nn.conv2d(x, w_1,strides=[1, 4, 4, 1], padding='VALID')
#Adding bias
c_1 = c_1 + b_1
#Applying RELU
c_1 = tf.nn.relu(c_1)
```

```python
print(c_1)
##POOLING LAYER1
p_1 = tf.nn.max_pool(c_1, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1
], padding='VALID')
print(p_1)

##CONVOLUTION LAYER 2
#Weights for layer 2
w_2 = tf.Variable(tf.truncated_normal([5,5,96,256], stddev=0.01)
)
#Bias for layer 2
b_2 = tf.Variable(tf.constant(1.0, shape=[[5,5,96,256][3]]))
#Applying convolution
c_2 = tf.nn.conv2d(p_1, w_2,strides=[1, 1, 1, 1], padding='SAME'
)
#Adding bias
c_2 = c_2 + b_2
#Applying RELU
c_2 = tf.nn.relu(c_2)

print(c_2)
```

```
21196
WARNING:tensorflow:From /anaconda3/lib/python3.7/sit
e-packages/tensorflow/python/framework/op_def_librar
y.py:263: colocate_with (from tensorflow.python.fram
ework.ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Colocations handled automatically by placer.
Tensor("Relu:0", shape=(?, 55, 55, 96), dtype=float3
2)
Tensor("MaxPool:0", shape=(?, 27, 27, 96), dtype=flo
at32)
Tensor("Relu_1:0", shape=(?, 27, 27, 256), dtype=flo
at32)
```

In [5]:

```
p_2 = tf.nn.max_pool(c_2, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1
], padding='VALID')
print(p_2)

##CONVOLUTION LAYER 3
#Weights for layer 3
w_3 = tf.Variable(tf.truncated_normal([3, 3, 256, 384], stddev=0
.01))
#Bias for layer 3
b_3 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 256, 384][3]]))
#Applying convolution
c_3 = tf.nn.conv2d(p_2, w_3,strides=[1, 1, 1, 1], padding='SAME'
)
#Adding bias
c_3 = c_3 + b_3
#Applying RELU
c_3 = tf.nn.relu(c_3)

print(c_3)
```

```
Tensor("MaxPool_1:0", shape=(?, 13, 13, 256), dtype=
float32)
Tensor("Relu_2:0", shape=(?, 13, 13, 384), dtype=flo
at32)
```

In [6]:

```
w_4 = tf.Variable(tf.truncated_normal([3, 3, 384, 384], stddev=0
.01))
#Bias for layer 4
b_4 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 384][3]]))
#Applying convolution
c_4 = tf.nn.conv2d(c_3, w_4,strides=[1, 1, 1, 1], padding='SAME'
)
#Adding bias
c_4 = c_4 + b_4
#Applying RELU
c_4 = tf.nn.relu(c_4)

print(c_4)
```

```
Tensor("Relu_3:0", shape=(?, 13, 13, 384), dtype=flo
at32)
```

In [7]:

```
w_5 = tf.Variable(tf.truncated_normal([3, 3, 384, 256], stddev=0
.01))
#Bias for layer 5
b_5 = tf.Variable(tf.constant(0.0, shape=[[3, 3, 384, 256][3]]))
#Applying convolution
c_5 = tf.nn.conv2d(c_4, w_5,strides=[1, 1, 1, 1], padding='SAME'
)
#Adding bias
c_5 = c_5 + b_5
#Applying RELU
c_5 = tf.nn.relu(c_5)

print(c_5)
```

```
Tensor("Relu_4:0", shape=(?, 13, 13, 256), dtype=flo
at32)
```

In [8]:

```
p_3 = tf.nn.max_pool(c_5, ksize=[1, 3, 3, 1],strides=[1, 2, 2, 1
], padding='VALID')
print(p_3)

#Flattening
flattened = tf.reshape(p_3,[-1,6*6*256])
print(flattened)
```

```
Tensor("MaxPool_2:0", shape=(?, 6, 6, 256), dtype=fl
oat32)
Tensor("Reshape:0", shape=(?, 9216), dtype=float32)
```

In [9]:

```
input_size = int( flattened.get_shape()[1] )
#Weights for FC Layer 1
w1_fc = tf.Variable(tf.truncated_normal([input_size, nodes_fc1],
stddev=0.01))
#Bias for FC Layer 1
b1_fc = tf.Variable( tf.constant(1.0, shape=[nodes_fc1] ) )
#Summing Matrix calculations and bias
s_fc1 = tf.matmul(flattened, w1_fc) + b1_fc
#Applying RELU
s_fc1 = tf.nn.relu(s_fc1)

#Dropout Layer 1
hold_prob1 = tf.placeholder(tf.float32)
s_fc1 = tf.nn.dropout(s_fc1,keep_prob=hold_prob1)

print(s_fc1)
```

```
WARNING:tensorflow:From <ipython-input-9-c7b7cb0b8f1
b>:13: calling dropout (from tensorflow.python.ops.n
n_ops) with keep_prob is deprecated and will be remo
ved in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate shoul
d be set to `rate = 1 - keep_prob`.
Tensor("dropout/mul:0", shape=(?, 4096), dtype=float
32)
```

In [10]:

```
w2_fc = tf.Variable(tf.truncated_normal([nodes_fc1, nodes_fc2],
stddev=0.01))
#Bias for FC Layer 2
b2_fc = tf.Variable( tf.constant(1.0, shape=[nodes_fc2] ) )
#Summing Matrix calculations and bias
s_fc2 = tf.matmul(s_fc1, w2_fc) + b2_fc
#Applying RELU
s_fc2 = tf.nn.relu(s_fc2)
print(s_fc2)
```

```
Tensor("Relu_6:0", shape=(?, 4096), dtype=float32)
```

In [11]:

```
hold_prob2 = tf.placeholder(tf.float32)
s_fc2 = tf.nn.dropout(s_fc2,keep_prob=hold_prob1)

##Fully Connected Layer 3 -- CLASSIFICATION HEAD
#Weights for FC Layer 3
w3_fc_1 = tf.Variable(tf.truncated_normal([nodes_fc2,output_clas
ses], stddev=0.01))
#Bias for FC Layer 3b3_fc = tf.Variable( tf.constant(1.0, shape=
[output_classes] ) )
b3_fc_1 = tf.Variable( tf.constant(1.0, shape=[output_classes] )
)
#Summing Matrix calculations and bias
y_pred_1 = tf.matmul(s_fc2, w3_fc_1) + b3_fc_1
#Applying RELU
print(y_pred_1)
```

```
Tensor("add_7:0", shape=(?, 3), dtype=float32)
```

In [12]:

```
w3_fc_2 = tf.Variable(tf.truncated_normal([nodes_fc2,output_locations], stddev=0.01))
#Bias for FC Layer 3b3_fc = tf.Variable( tf.constant(1.0, shape=[output_classes] ) )
b3_fc_2 = tf.Variable( tf.constant(1.0, shape=[output_locations] ) )
#Summing Matrix calculations and bias
y_pred_2 = tf.matmul(s_fc2, w3_fc_2) + b3_fc_2
#Applying RELU
print(y_pred_2)
```

```
Tensor("add_8:0", shape=(?, 4), dtype=float32)
```

In [13]:

```
cross_entropy = tf.multiply(tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true_1,logits=y_pred_1)),10)

#Defining Regression Loss
regression_loss = tf.multiply(tf.reduce_mean(tf.square(y_pred_2 - y_true_2)),1.0)

#Defining total loss
final_loss = cross_entropy + regression_loss

#Defining objective
train = tf.train.AdamOptimizer(learning_rate=0.00001).minimize(final_loss)

#Defining Accuracy
matches = tf.equal(tf.argmax(y_pred_1,1),tf.argmax(y_true_1,1))
acc = tf.reduce_mean(tf.cast(matches,tf.float32))

#Initializing weights
init = tf.global_variables_initializer()

#Starting Empty lists to keep results
acc_list = []
auc_list = []
loss_list = []
regression_list = []
#In order to save, creating a tf.train.Saver() object.
saver = tf.train.Saver()
```

In [14]:

```
config = tf.ConfigProto(allow_soft_placement=True)
config.gpu_options.allow_growth = True
config.gpu_options.allocator_type = 'BFC'

tf.add_to_collection("classification_head", y_pred_1)
tf.add_to_collection("regression_head", y_pred_2)
```

In [15]:

```
def main():
    with tf.Session(config=config) as sess:
        sess.run(init)

        for i in range(epochs):
            for j in range(0,steps-remaining,step_size):
                #Feeding step_size-amount data with 0.5 keeping probabilities on DROPOUT LAYERS
                _,c = sess.run([train,final_loss],
                feed_dict = {x:X[j:j+step_size], y_true_1:Y1[j:j+step_size],
                y_true_2:Y2[j:j+step_size],hold_prob1:0.5,hold_prob2:0.5})

            #Writing for loop to calculate test statistics. GTX 1050 isn't able to calculate all cv data.
            cv_auc_list = []
            cv_acc_list = []
            cv_loss_list = []
            cv_regression_list = []
            for v in range(0,len(cv_x)-int(len(cv_x) % validating_size),validating_size):
                acc_on_cv,loss_on_cv,preds,coordinates = sess.run([acc,cross_entropy,tf.nn.softmax(y_pred_1),y_pred_2],
                feed_dict={x:cv_x[v:v+validating_size], y_true_1:cv_y1[v:v+validating_size], y_true_2:cv_y2[v:v+validating_size],
                  hold_prob1:1.0,hold_prob2:1.0})

                auc_on_cv = roc_auc_score(cv_y1[v:v+validating_size],preds,average='micro')

                regression_loss = np.mean(pow(cv_y2[v:v+validating_size] - coordinates , 2 ) )
                cv_acc_list.append(acc_on_cv)
                cv_auc_list.append(auc_on_cv)
                cv_loss_list.append(loss_on_cv)
                cv_regression_list.append(regression_loss)
            acc_cv_ = round(np.mean(cv_acc_list),5)
            auc_cv_ = round(np.mean(cv_auc_list),5)
            loss_cv_ = round(np.mean(cv_loss_list),5)
            regression_loss_cv_ = round(np.mean(cv_loss_list),5)
            acc_list.append(acc_cv_)
            auc_list.append(auc_cv_)
            loss_list.append(loss_cv_)
            regression_list.append(regression_loss_cv_)
            print("Epoch:",i,"Accuracy:",acc_cv_,"Loss:",loss_cv
_ ,"AUC:",auc_cv_)
        print("Training has finished and model is saved")
        saver.save(sess, os.path.join(os.getcwd(),"CNN_OL.ckpt")
)
```

In [ ]:

```
if __name__ == "__main__":
    main()
```

```
Epoch: 0 Accuracy: 1.0 Loss: 0.0 AUC: 1.0
```

In [ ]:

In [ ]: