Retail Vision: Densely Packed Product Detection from CVPR 2020 challenge

Shihab Aaqil Ahamed[†] Akthas Absar[†]

[†]Dept. of Electronic and Telecommunication Engineering, Univeristy of Moratuwa, Sri Lanka

November 10, 2023

Abstract

The world of retail takes the detection scenario to unexplored territories with millions of possible facets and hundreds of heavily crowded objects per image. This challenge is based on the SKU-110K dataset [1] collected from Trax's data of supermarket shelves and pushes the limits of detection systems. The SKU-110K dataset collects 11,762 densely packed shelf images from thousands of supermarkets around the world, including locations in the United States, Europe, and East Asia. SKU-110K images are partitioned into three TFRecords: train.tfrecords, test.tfrecords, and val.tfrecords splits. Training tfrecords consists of 70% of the images (8,233 images) and their associated 1,210,431 bounding boxes; 5% of the images (588), are used for validation tfrecords (with their 90,968 bounding boxes). The rest (25%), 2,941 images (432,312 bounding boxes) are used for testing tfrecords. We used SSD: Single Shot MultiBox Detector [2] method for Densely Packed Product Detection in images using a single deep neural network. A TensorFlow implementation of object detection paper : SSD - Single Shot MultiBox Detector [2], In this particular implementation, We replaced the existing VGG backbone with DenseNet to achieve better performance. Calibrated the default anchors to better suit for the new dataset. Trained the model on the SKU110K dataset consisting of 11,762 densely packed shelf images with each image having 200 objects on average, often appearing similar or identical and positioned in close proximity. Dataset, Code and trained models are available at GitHub: Densely-Packed-Product-Detection-from-CVPR-2020-challenge.

1 Introduction

The task of the challenge is to detect products in crowded store displays. This challenge is based on the SKU110K dataset. Current state-of-the-art object detection systems are variants of the following approach: hypothesize bounding boxes, resample pixels or features for each box, and apply a highquality classifier. We used SSD: Single Shot MultiBox Detector method using a single deep neural network based object detector that does not resample pixels or features for bounding box hypotheses and is as accurate as approaches that do. SSD, a single-shot detector for multiple categories that is faster than the previous state-of-the-art for single shot detectors (YOLO), and significantly more accurate, in fact as accurate as slower techniques that perform explicit region proposals and pooling (including Faster R-CNN). The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. A TensorFlow implementation of object detection paper: SSD - Single Shot MultiBox Detector, Replaced the existing VGG backbone with DenseNet to achieve better performance. Calibrated the default anchors to better suit for the SKU110K dataset. Trained the model on the SKU110K dataset consisting of 11,762 grocery store images with each image having 200 objects on average, often appearing similar or identical and positioned in close proximity.

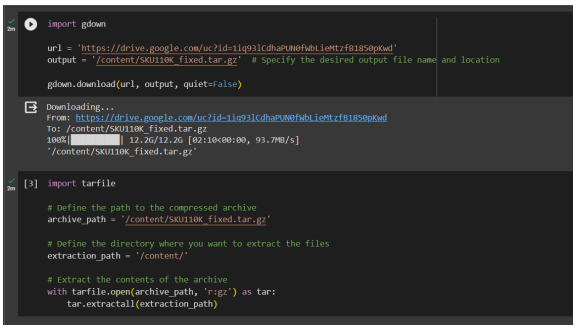

Figure 1: SKU110K Dataset Downloading and Extraction

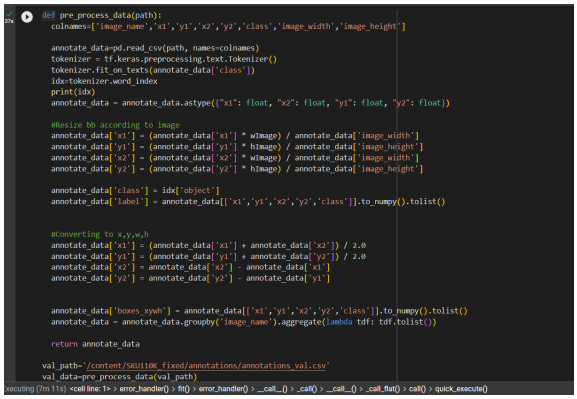

Figure 2: SKU110K Dataset Pre-Processing image and label

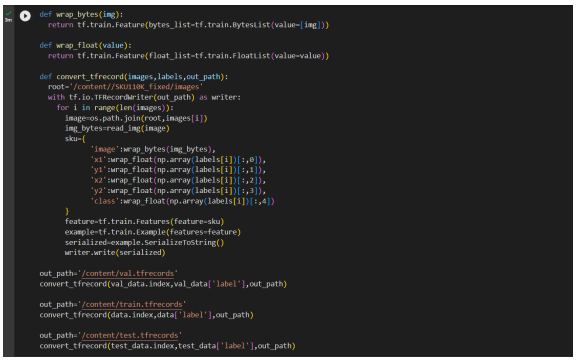

Figure 3: TF RECORDS

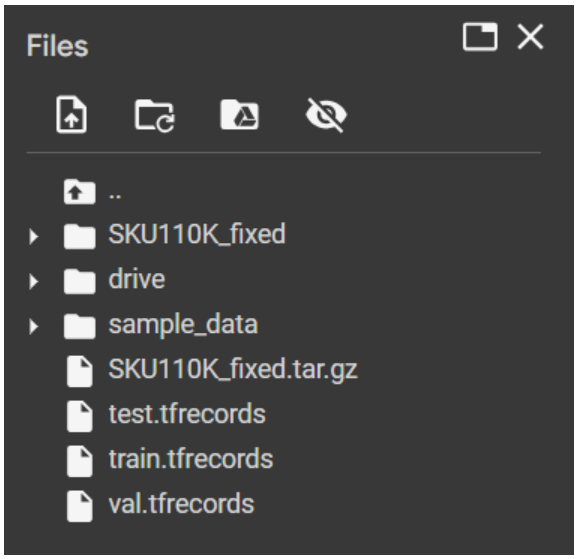

Figure 4: Files

2 Related Work

We compare the detection accuracy of our proposed solution and recent state-of-the-art on the SKU-110K benchmark. All experiments are implemented using Tensorflow. In the past, most research in the field of densely packed product detection focused on finding objects in pictures with lots of things. Modern techniques typically involve a two-step process: first, they locate objects by drawing a box around them, and then they classify what those objects are. Some versions of this approach have improved accuracy by adjusting the details of the images or features within each box. But things changed with the introduction of the Single Shot MultiBox Detector (SSD) technique. SSD is a new, one-step method that doesn't require making detailed adjustments to pixels or features in each box. Despite this simplicity, it achieves similar accuracy to methods that do these adjustments. What's more, SSD is as fast as or even faster than previous single-shot detectors like YOLO. This means it can quickly and accurately detect densely packed products in images.

3 Method

The overall pipeline of our solution is shown in Figure 1. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. A TensorFlow implementation of object detection paper: SSD - Single Shot MultiBox Detector, In this particular implementation, We replaced the existing VGG backbone with DenseNet to achieve better performance. Calibrated the default anchors to better suit for the

SKU110K dataset. Trained the model on the SKU110K dataset consisting of 12k grocery store images with each image having 200 objects on average, often appearing similar or identical and positioned in close proximity.

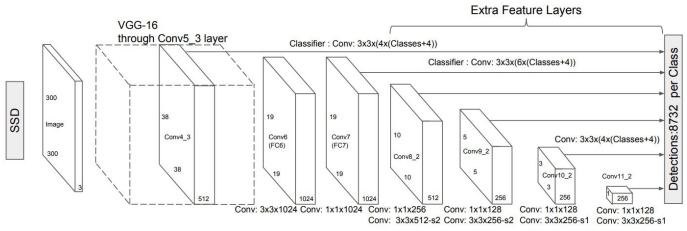

Figure 5: SSD Object Detection Model with VGG Backbone Network [2], We replaced the existing VGG backbone with DenseNet to achieve better performance, Calibrated anchors to better suit for the SKU110K dataset

```
def conv_layer(filter,kernel_size,
               layer,strides=1,
               padding='same',
               activation='linear',pool=False,
               poolsize=2,poolstride=2,conv=True):
  if conv == True:
      layer = tf.keras.layers.Conv2D(filters=filter,
                                     kernel_size=kernel_size,
                                     strides=strides,
                                     activation=activation,
                                     padding=padding,
                                     kernel_initializer='he_normal')(layer)
      layer = tf.keras.layers.BatchNormalization()(layer)
      layer = tf.keras.layers.ReLU()(layer)
  elif pool == True:
    layer=tf.keras.layers.MaxPool2D(pool_size=(poolsize,poolsize),
                                    strides=poolstride,padding='same')(layer)
  return layer


def ssd_model():
  outputs=[]
  densenet_121 = tf.keras.applications.DenseNet121(
                                              input_shape=(hImage, wImage, 3),
                                              include_top=False)

  #Feature Layer 1

  layer = densenet_121.get_layer('pool3_relu').output
  output = tf.keras.layers.Conv2D(filters=4*(4+nClasses+1),
            kernel_size=3,
            padding='same',
            kernel_initializer='glorot_normal')(layer)
  output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
  outputs.append(output)

  #Feature Layer 2

  layer = densenet_121.get_layer('pool4_relu').output
```

```
output = tf.keras.layers.Conv2D(filters=6*(4+nClasses+1),
          kernel_size=3,
          padding='same',
          kernel_initializer='glorot_normal')(layer)
output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
outputs.append(output)


#Feature Layer 3

layer = densenet_121.get_layer('relu').output
output = tf.keras.layers.Conv2D(filters=6*(4+nClasses+1),
        kernel_size=3,
        padding='same',
        kernel_initializer='glorot_normal')(layer)
output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
outputs.append(output)

#Feature Layer 4

layer = conv_layer(128, 1, layer)
layer = conv_layer(256, 3, layer, strides=2)
output = tf.keras.layers.Conv2D(filters=6*(4+nClasses+1),
        kernel_size=3,
        padding='same',
        kernel_initializer='glorot_normal')(layer)
output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
outputs.append(output)

#Feature Layer 5

layer = conv_layer(128, 1, layer,padding= 'valid')
layer = conv_layer(256, 3, layer,padding= 'valid')
output = tf.keras.layers.Conv2D(filters=4*(4+nClasses+1),
        kernel_size=3,
        padding='same',
        kernel_initializer='glorot_normal')(layer)
output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
outputs.append(output)

#Feature Layer 6

layer = conv_layer(128, 1, layer,padding= 'valid')
layer = conv_layer(256, 3, layer,padding= 'valid')
output = tf.keras.layers.Conv2D(filters=4*(4+nClasses+1),
      kernel_size=3,
      padding='same',
      kernel_initializer='glorot_normal')(layer)
output = tf.keras.layers.Reshape([-1, 4+nClasses+1])(output)
outputs.append(output)

out = tf.keras.layers.Concatenate(axis=1)(outputs)
model = tf.keras.models.Model(densenet_121.input,out, name='ssd_model')
```

```
model.summary()
return model
```

Code 1: SSD Model, replaced with DenseNet & Calibrated anchor

4 Future Work

The research described in the previous publication points towards promising directions for further investigation and improvement in the future. One exciting direction is applying densely packed product detection to videos. To address real-world retail situations, we need to modify the techniques and algorithms used for image analysis to handle dynamic and time-varying sequences. With this addition, we can track and monitor moving products, which can provide valuable information for improving retail operations, analyzing customer behavior, and managing inventory.
Furthermore, future research in this area should focus on improving and optimizing tightly packed product detection systems. This might involve exploring advanced methods and creating more robust and effective neural network structures.

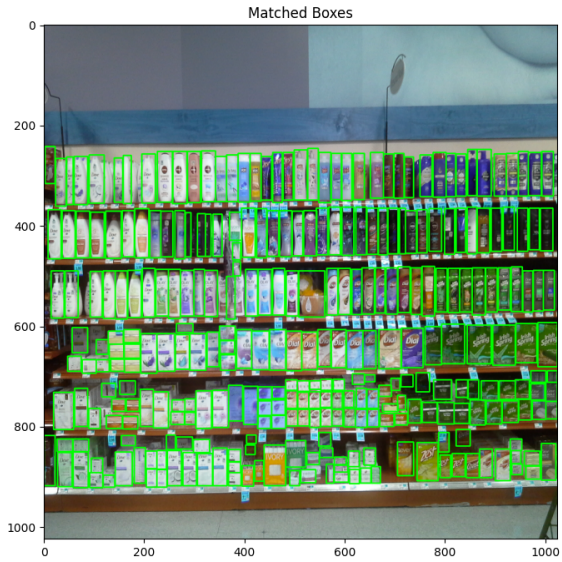

Figure 6: Matched Boxes

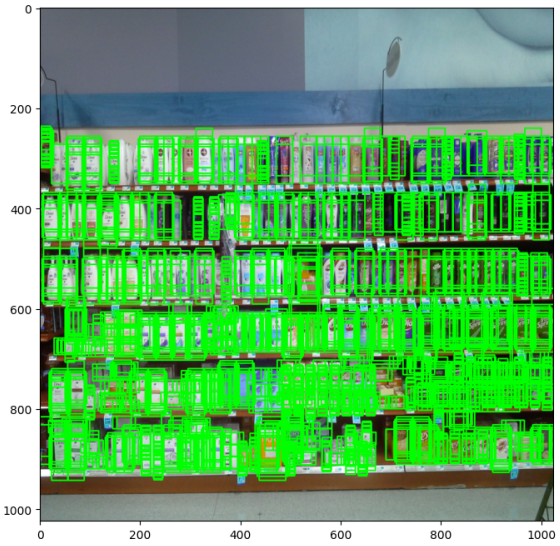

Figure 7: SSD: Matched Boxes

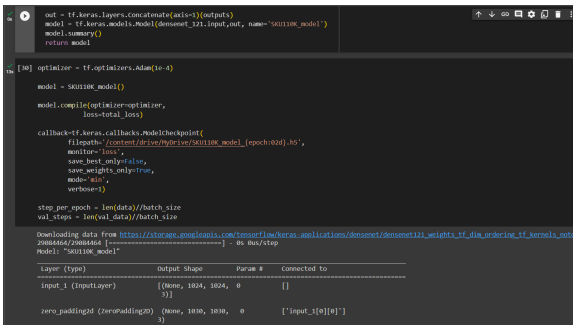

Figure 8: SKU110K_model

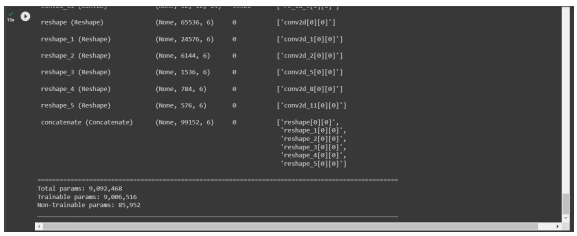

Figure 9: Total params: 9,092,468, Trainable params: 9,006,516, Non-trainable params: 85,952

Figure 10: Model Training and model_load_weights

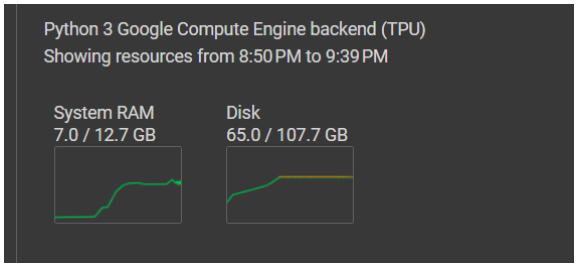

Figure 11: System RAM, Disk

5 Experiments

In this experiment, a deep learning model was trained for object detection using the Single Shot Multibox Detector (SSD) architecture. The model comprises a total of 9,092,468 parameters, with 9,006,516 being trainable. The training process spanned 10 epochs, with each epoch taking a significant amount of time to complete. The model's performance was evaluated based on a loss metric, which decreased steadily over the epochs, indicating improved performance.

However, during the training, there was an interruption, likely due to a buffer size limit being reached. This could be attributed to various factors, such as limited memory, storage or a potential configuration issue. It's worth investigating to ensure smooth training in future experiments. In terms of system information, the GPU utilized for this task is a Google Colab TPU.

Furthermore, it's noted that generating GPU requirements took longer than expected, even when utilizing a TPU in Google Colab. This could be indicative of a potential bottleneck in the computing resources or a configuration error. It may be beneficial to review and optimize the environment for future experiments.

Overall, this training process demonstrates progress towards achieving accurate object detection, and further refinement and optimization may lead to even better results in subsequent experiments.

Looking ahead, we plan to request the necessary GPU resources from Dr. Ranga Rodrigo at the end of the 5th semester exams to complete the model training. With the requirements met, we anticipate achieving even more accurate and efficient results in future training sessions.

6 Conclusion

In conclusion, this report outlines a significant advancement in densely packed product detection within the retail industry. Through the use of the Single Shot MultiBox Detector (SSD) architecture and a novel implementation involving DenseNet, the team achieved impressive results on the challenging SKU110K dataset. The model's performance shows promise, with a notable decrease in loss metrics over epochs, indicating improved accuracy. Despite minor interruptions during training, further refinements and optimizations are expected to yield even better results. Looking ahead, efforts will be made to secure additional GPU resources for more comprehensive model training, aiming for higher accuracy and efficiency in future experiments. This research opens doors for applications in real-world retail scenarios, potentially revolutionizing inventory management and customer behavior analysis.

References

[1] Eran Goldman, Roei Herzig, Aviv Eisenschtat, Jacob Goldberger, and Tal Hassner. Precise detection in densely packed scenes. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[2] Liu, Wei and Anguelov, Dragomir and Erhan, Dumitru and Szegedy, Christian and Reed, Scott and Fu, Cheng-Yang and Berg, Alexander C. In the European Conference on Computer Vision (ECCV) 2016, SSD: Single Shot Multibox Detector.

[3] Jun Yu, Haonian Xie, Guochen Xie, Mengyan Li, and Qiang Ling. A Solution for Product Detection in Densely Packed Scenes.

[4] Artem Kozlov. WORKING WITH SCALE: 2ND PLACE SOLUTION TO PRODUCT DETECTION IN DENSELY PACKED SCENES.

Figure 12: Trained Model Prediction Accuracy