ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

SYS800

Reconnaissance de formes et inspection

Heuristic Anchor Box Selection for the Single Shot Detector Algorithm

PRÉSENTÉ À: Mohamed Cheriet

PAR

Bozan XU - XUXB02079305

MONTRÉAL, 12 Décembre 2019

Summary

For this project, I propose a solution to optimize anchor boxes for state-of-the-art object detection algorithm Single Shot Detector.

Nowadays, anchor boxes are pre-defined in a simple hand-picked fashion. Wei et al. chose anchors with the ratio of 1:1, 1:2, 2:1, 1:3, and 3:1. In their original work, Single Shot Detector 300x300 achieves 74.3% mAP on VOC2007 testing set.

To improve accuracy and reduce the number of anchor boxes needed, I propose to run k-means on the ground truth boxes of the VOC2007+VOC2012 training set to adapt the anchors to the data distribution. It was found that the number of anchors per grid cell can be dropped from 6 to 4, and still observe an increased 0.7% mAP.

Table of Contents

Table of Figures

Introduction

In recent years, deep learning based object detection has seen great improvement in speed and accuracy. One such algorithm is Single Shot Detector (SSD) which has shown to be as accurate as Faster-R-CNN [1] and runs at over than twice the speed.

Motivation

Today, objection detection algorithms have taken center stage especially as many big corporations race toward building the first generation of autonomous vehicles. The applications of these algorithms range from autopilot systems to facial recognition and from anomaly detection to video tracking. As a result, many architectures of deep networks have been developed to solve the same problem. In order to compare these different detectors, we measure their performance in terms of mAP and speed. SSD is one such network. It is part of a subclass of detectors called single stage detectors. They are generally used for applications that require real-time processing, but they are not as accurate as algorithms belonging to the two-stage detector family, like Faster-R-CNN.

Current Challenges

Other from needing to process billions of image, lidar, radar, and map data, there is a critical need for faster and more robust algorithms that will improve the safety and reliability of autonomous systems. There are many concerns about how quickly they can come to market even if they work some of the time -- The technology has to work perfectly.

One obvious challenge arises when there is no large-scale labeled data. In the case for autopilot applications, amongst thousands of possible scenarios, let's imagine the instance for which, a traffic cop signals drivers to go around a construction site, into incoming traffic. The human driver will easily be able to break traffic rules and follow hand signs, but it will be hard for an autonomous car to make the same decision. The truth is, there is still a long way to go before robots can accurately differentiate complex social interactions.

Furthermore, from an analytical point of view, current leading algorithms are, while accurate, very computationally intensive, such that they are too slow for real-time applications, and simply do not run on embedded systems [1]. SSD is a state-of-the-art algorithm that balances speed and accuracy. But in the last two years, there have been algorithms like Retinanet that are still slower, but provides much higher accuracy.

Problem Statement

In this project, I aim to find a way to improve the accuracy of SSD. Even though it achieves a modest 74.3 mAP, some aspects of SSD can be reworked.

In its original paper, one point that left many people wondering is the simple way that Wei et al hand-picked their anchor boxes. The researchers do not provide any explanation on their choices. I think that if the anchor boxes follow the shapes of the data, they will make better predictions, hence improving the accuracy.

After going through class material, in Chapter 6, I came across a statistical method called k-means that classifies objects into clusters. If I apply k-means on the shape of the labeled data, will the generated anchor boxes improve detection for that specific dataset? We will find out in this project.

Baseline

Architecture

Developed in 2016 by Liu et al., SSD is a one-step approach, hence the name “single shot detector”. SSD comprises of two main components:

1) A backbone model, that functions as a feature extractor. It is usually a pre-trained image classification network like ResNet or Mobilenet, but whose final fully connected classification layer has been removed. In the original paper [1], Liu et al used VGG16.
2) And, an SSD head, made out of convolutional layers that progressively decreases in size to allow detection at multiple scales. It outputs the bounding boxes and classes of objects in the spatial location of the final layer activations.

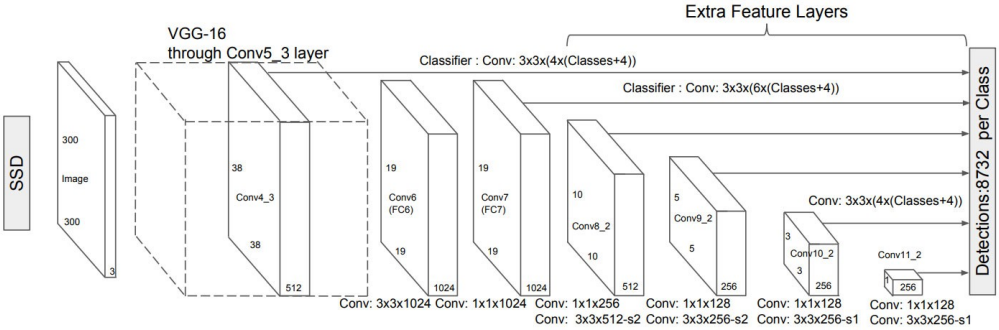

Figure 1: SSD architecture featuring VGG16 serving as backbone model.

As illustrated in Figure 1, the backbone network, VGG16, has had its final activation layer removed. On the VGG16 network, it means that we are only taking it up to layer Conv4_3, which happens to be a size 38x38x512. To this layer, we apply a 3x3 filter in the hopes to extract some kind of feature from the image. It also reduces the spatial dimension of the feature map while retaining only the most important information.

Connected to Conv4_3 is the SSD head. The SSD head consists of convolutional layers Conv7 through to Conv11_2, with a couple of 1x1 or 3x3 filters connecting each layer one another.

Multi-Scale Feature Maps

The application of 1x1 and 3x3 filters reduces the spatial dimension gradually which means that the resolution of the feature maps also decreases, but in the SSD framework, lower resolution layers are used to detect larger objects, as shown in Figure 2 (c).

SSD works by dividing the input image into a grid of cells. Each grid cell is responsible for predicting the class and position of objects within it. If there is no object, it will predict the background class, and no location will be returned.

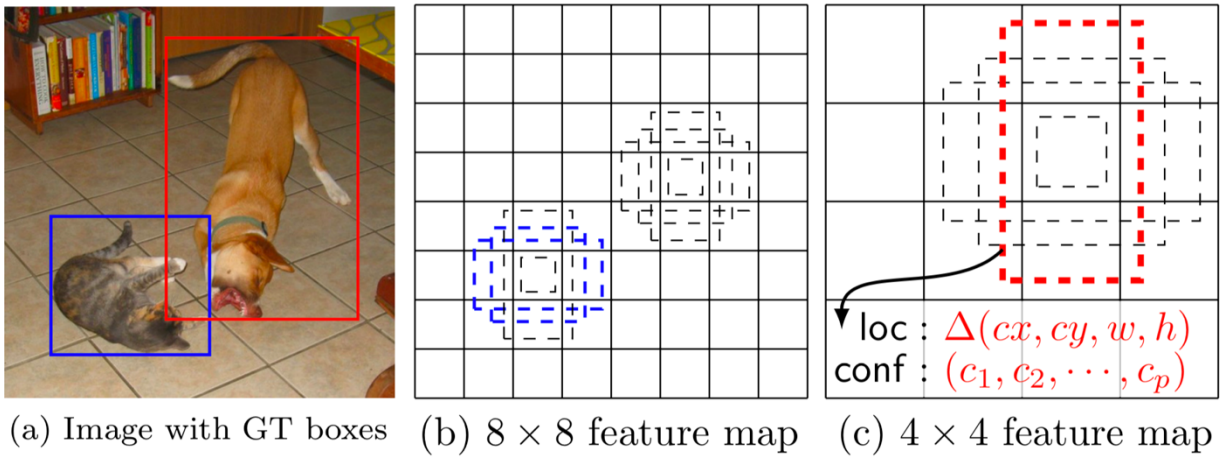

Figure 2: (a) A training data containing ground truth boxes for the cat and the dog. (b) In a feature map that is divided into an 8x8 grid, the anchor boxes of different aspect ratios cover the smaller areas of the raw input. (c) In a coarser grid, the anchor boxes detect objects in larger areas of the input.

Anchor boxes are chosen manually. In the original paper, Wei et al chose six anchor boxes of ratio $\left\{1, 2, 3, \frac{1}{2}, \frac{1}{3}\right\}$. They seem like 5 right now, but we will get to the 6th box.

SSD defines a scaling factor for each of the 6 feature maps. We define index $k \in [1, 6]$ to represent the k-th feature map. Then,

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{6 - 1}(k - 1)$$

where $s_{min} = 0.1$, $s_{max} = 0.7$, and $s_{m+1} = 1$. This means that starting from the left, the layers in the SSD head detect objects from the smallest scale of 0.1 all the way up to a scale of 0.7.

We can then compute the width and height of the anchor boxes:

$$w = s * \sqrt{aspect\ ratio}$$
$$h = s / \sqrt{aspect\ ratio}$$

For the aspect ratio of 1, the SSD adds an extra anchor box with scale:

$$s_k{}' = \sqrt{s_k * s_{k+1}}$$

Every anchor box is also centered about the grid cell. Its center coordinates are:

$$\left(\frac{i + 0.5}{f_k}, \frac{j + 0.5}{f_k}\right)$$

where f_k is the size of the k-th feature map and $i, j \in [0, f_k]$.

To calculate the total number of bounding boxes that will be predicted, we combine the predictions from every anchor box of different scales and aspect ratios coming from every grid cell. This number is usually in the thousands, and as for larger models like SSD512, the output makes 24564 predictions[1]. For SSD300, the number of output boxes per feature map are listed below.

Convolutional Layer	Dimension	Number of Anchor Boxes
Conv4_3	38x38	8664
Conv7	19x19	2166
Conv8_2	10x10	600

Conv9_2	5x5	150
Conv10_2	3x3	54
Conv11_2	1x1	6

Matching

During training, each ground truth box is first matched with an anchor box with which it gets the highest Jaccard overlap. Then, any anchor box that gets an overlap higher than the threshold of 0.5 is also matched to the corresponding ground truth box. In Figure 9 (b), there are two anchor boxes matched with the cat. These anchor boxes are called positive matches, whereas the two other anchors are negative matches. The training objective is then to minimize a loss function.

Loss Function

The loss function for SSD is a weighted sum of classification loss and localization loss:

$$L = \frac{1}{N}(L_{classification} + \propto L_{localization})$$

where N is the number of matched anchor boxes and $\propto$ balances the weights between the losses.

The localization loss is a loss between the predicted box correction (d_m^i) and the true values g_m^j:

$$L_{localization} = \sum_{ij} \sum_{m \in \{x,y,w,h\}} \mathrm{I}_{ij}^k \, L_{smoothL1}(d_m^i - t_m^j)^2$$

where I_{ij}^k represents the matching between i^{th} anchor box with coordinates $(p_x^i, p_y^i, p_w^i, p_h^i)$ and j^{th} ground truth box with coordinates $(g_x^i, g_y^i, g_w^i, g_h^i)$ for an object in class k. It is 1 if if $IoU > 0.5$, and 0 otherwise.

$L_{smoothL1}$ is a Smooth L1 loss defined as:

$$L_{smoothL1}(s) = \begin{cases} 0.5s^2 & when\ |s| < 1 \\ |s| - 0.5 & otherwise \end{cases}$$

t_m^j represent the offsets for the center, width, and height of the anchor boxes. They are defined as:

$$t_x^j = (g_x^j - p_x^i)/p_w^i$$
$$t_y^j = (g_y^j - p_y^i)/p_h^i$$
$$t_w^j = \log\left(\frac{g_w^j}{p_w^i}\right)$$
$$t_h^j = \log\left(\frac{g_h^j}{p_h^i}\right)$$

As for the classification loss, we apply a softmax to make it into a nice probability function:

$$L_{classification} = -\sum_{i \in Pos}^{N} \mathrm{I}_{ij}^k \log(\hat{c}_i^k) - \sum_{i \in Neg}^{N} \log(\hat{c}_i^0)$$

where $\hat{c}_i^k = softmax(c_i^k) = \frac{exp(c_i^k)}{\sum_k c_i^k}$

For positive matches, the classification loss is penalized according to the confidence score of class k. For negative matches, SSD predicts class 0, or background, meaning no object was detected.

As mentioned in the previous section, there is a very large number of predictions that are made for the number of objects present in a given image. This means that there will be many times more negative matches than positive matches. The imbalance causes the model to learn about the background space rather than detecting objects. To remedy this problem, SSD only keeps the matches with highest confidence (false positives) as to keep a ratio of 3:1 between negatives and positives. Apparently, this ratio leads to faster optimization and even makes the training process more stable[1].

Methodology

SSD predicts 6 bounding boxes for every grid cell on a feature map. During training, these anchor boxes compete with each other to score the highest Jaccard Index to predict objects. In the original paper, Wei et al handpicked the boxes in function of what they thought would work with the dataset.

Proposed Approach

To improve the accuracy of this detector, I propose a heuristic way to design the anchor boxes.

K-means clustering is one of the most popular unsupervised machine learning algorithms. In k-means clustering, we define k centroids where each centroid corresponds to the center of a non-overlapping cluster. K-means aims to minimize the following function:

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} w_{ik}(x^i - \mu_k)^2$$

where $w_{ik} = 1$ if data point x^i belongs to cluster k, and $w_{ik} = 0$ otherwise.
μ_k is the k-th centroid defined by:

$$\mu_k = \frac{\sum_{i=1}^{n} w_{ik} x^i}{\sum_{i=1}^{n} w_{ik}}$$

K-means minimizes the objective function in two parts [2]. First, it allocates every data point to the nearest cluster by computing the sum of the squared distances between the data points and each centroid to find the minima. This is reflected by taking the derivative of J with respect to w_{ik}.

$$\frac{\partial J}{\partial w_{ik}} = \sum_{i=1}^{n} \sum_{k=1}^{K} (x^i - \mu_k)^2$$

Then, it averages each data point in a cluster to find a new centroid. This is done by taking the derivative of J with respect to μ_k.

$$\frac{\partial J}{\partial \mu_k} = 2 \sum_{i=1}^{n} w_{ik} \parallel x^i - \mu_k \parallel = 0$$

As we have seen in Chapter 6 of the class, the k-means algorithm starts with randomly selected centroids and it performs iterative calculations to optimize the positions of the centroids until the centroids are stabilized.

In order to establish a controlled environment to compare the accuracy of SSD and SSD with k-means, I plan on dividing the workload in three.

I will start by creating a custom training and validation set from the PASCAL VOC2007+VOC2012 dataset. I will use the VOC2007 test set for testing.

The second step is to implement an SSD model. For that, I used OpenCV library with Python bindings for image processing, and a TensorFlow backend to build and train the network. Due to limitation on time, I could not train the network in the same way as was described in the original paper [1]. Once trained, I will test my version of SSD to see if I can get the performance from the original paper.

Finally, I will run k-means clustering on the shapes of the ground truth boxes in the training set and compare the change in accuracy on the testing set.

Validation and Results

PASCAL VOC

The PASCAL VOC 2007 dataset contains box annotations for 20 classes. The training and validation set consists of 9,963 images with 24,640 annotated objects.

The PASCAL VOC 2012 dataset contains box annotations for 20 classes. The training and validation set consists of 11,530 images with 27,450 annotated objects.

The training and validation set I am using consists of a mix of the VOC2007 and VOC2012 training sets. The testing set is the VOC2007 test set that contains 4952 images.

The training was done in 30,000 iterations of Stochastic Gradient Descent with a scheduled learning rate.

Iterations	Learning Rate
0~100	10^{-4}
101~15,000	10^{-3}
15,001~27,000	10^{-4}
27,001~30,000	10^{-5}

I rented a TPU provided by Google Colab to complete the training. It took over 6 hours to complete.

K-means Implementation Details

Since k-means uses distance based measurements to determine the similarity between data points, I needed to prepare data containing the standardized height and width. The shape of the clustering feature data is then

$$number\ of\ objects\ \times\ (width, height)$$

To visualize the clusters, I plot the clustering data.

Figure 3: Visualizing anchor box data.

The data does not seem easily separable at first sight. To choose the ideal number of anchor boxes k, I ran k-means on $k = 2, 3, \ldots, 7$.

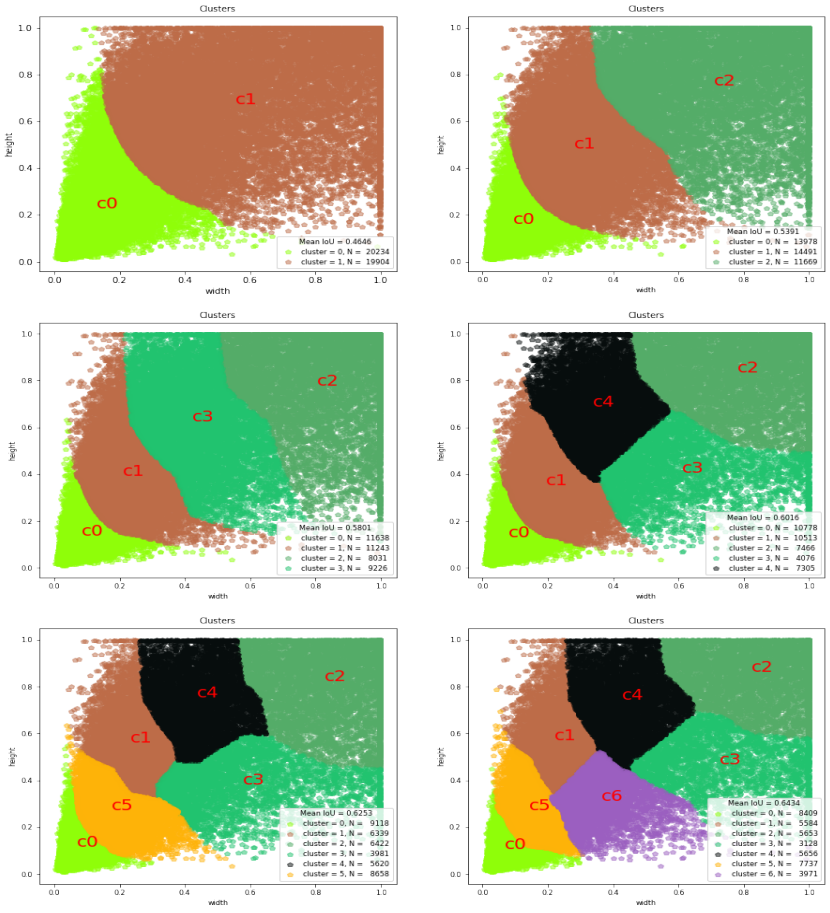

Figure 4: Plots of the clusters calculated by k-means for different k.

Number of Clusters	Mean IoU
2	0.4646
3	0.5391
4	0.5801
5	0.6016
6	0.6253
7	0.6434

The trend we observe is that the more clusters there are, the more overlap there will be between the generated anchor boxes and the ground truth boxes. It makes sense because if we had $k = number\ of\ objects$, we would expect a mean IoU of 1.

In Figure 5, we used the elbow method to find that the optimal $k = 4$ for the PASCAL VOC2012 dataset. This might be good because SSD predicts a large number of background class objects, so with fewer anchor boxes, SSD learns faster and makes training more stable in the early stage.

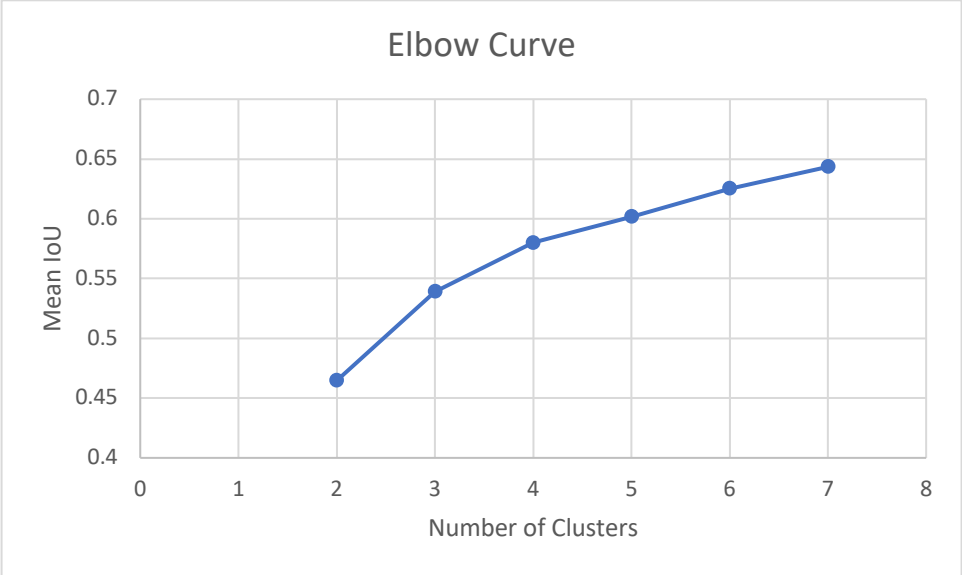

Figure 5:Clustering box dimensions on VOC.

And according to k-means, the four ideal centroids are:

Aspect Ratio of Anchor Boxes
1 : 1.654
1 : 1.891
1.037 : 1
1 : 1.478

This means that with only these four anchor boxes, we get a better representation of the data and make the training more robust. These anchor boxes also tell us that we aspect ratios of $\left\{\frac{1}{3}, 3\right\}$ might not be very important in detecting objects in the PASCAL VOC datasets.

Experimental Results

Method	mAP@0.5
SSD300 (baseline)	74.3 [1]
SSD300 without $\left\{\frac{1}{3}, 3\right\}$	73.7 [1]
SSD330 without $\left\{\frac{1}{3}, \frac{1}{2}, 2, 3\right\}$	71.6 [1]
my_SSD300	74.2
my_SSD300 + k-means	74.9

In the original paper, SSD achieves an accuracy of 74.3 when using all 6 hand-picked anchor boxes. When using only 4 anchors, it was shown that it only achieved an accuracy of 73.7. The table above shows that my implementation of SSD achieves a slightly lower accuracy of 74.2 when using the 6 anchor boxes chosen by the authors, but it is able to achieve an accuracy of 74.9 when the anchors are determined by k-means. From the results, the proposed anchor box optimization method boosts the performance by 0.7% on my version of SSD or a 0.6% boost compared to the baseline.

Conclusion

In this project, I introduced k-means as a method for anchor optimization for SSD. The anchors produced by k-means are better adapted for any custom dataset, which in turn, lead to an increase in accuracy. We can imagine that detecting characters will need very different anchor boxes than detecting wildlife. The proposed method demonstrated a 0.7% increase in performance on the popular SSD detector. In fact, this method is general and can be extended to any object detection algorithm that makes use of anchor boxes. Finally, this project solves the problem of choosing the best shapes of anchor boxes, but given more time, I would test the performance of SSD for different values of k to find the optimal number of anchor boxes, which in itself would be an interesting topic to study.

References

1. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. SSD: Single Shot MultiBox Detector. *arXiv preprint arXiv:1512.02325, 2016.*

2. T.Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE, 2002.*