Exploring Box Convolutional Layers

David Kang
Stanford University
dwkang@stanford.edu

Harry Emeric
Stanford University
harryem@stanford.edu

Abstract

Many current computer vision algorithms achieve high accuracy metrics, but require considerable computation to train. This paper will implement the Box Convolutional Layer into several state of the art architectures on various problems, to see whether there is an improvement in efficiency of computation and accuracy. We will investigate three tasks: classification through ResNet, semantic segmentation through PSPNet, and object detection through SSD. In our results, we found that the Box architectures produced relatively similar results in terms of accuracy but was more computationally expensive as epochs took longer to train than their non-Box counterparts. For object detection however, we noticed significant gains in computational efficiency with quicker descending loss curves and similar performance relative to accuracy, and the box implementation of other object detection models could be a topic of further study.

1. Introduction

In 2012, the introduction of AlexNet in the ImageNet competition opened a world of possibilities in the field of computer vision and deep learning. The idea of convolving around images using filters dramatically increased the ability of a neural network to understand the structure of images, and outperformed the previously state-of-the-art algorithms. As the convolutional layer began to be used more widely, people have recognized one significant limitation with convolutional neural networks, which is that they require a significant number of parameters depending on the size of the filter. As one increases the dimensions of the convolution, the computational expenses increase quadratically and the network becomes prone to overfitting. Thus, most applications limit the size of the average filter to around a 3x3 spatial dimension.

With Egor Burkov and Victor Lempitsky's 2018 paper Deep Neural Networks with Box Convolutions,[2] the idea of a box filter was reintroduced into the computer vision world. Unlike the rather static square filter of traditional convolutional layers, the box convolution layer allows filters of learnable dimensions to pass through the images. This might seem even more computationally expensive, but through the use of integral images and using the average pixel value of a convolution instead of a dot product, the box convolution is significantly more computationally efficient.

Due to the relative recency of this idea, architectures utilizing box convolutions have not been widely explored. In their paper, Burkov and Lempitsky implemented their box convolution layer on the Cityscapes dataset combined with the ENet [13] and ERFNet [16] architectures, but there is not much else aside from this.

The goal of this paper is to implement the Box Convolution layer in various computer vision problems using a number of architectures. We will start off with a basic image classification problem on the Tiny Imagenet dataset, using a standard ResNet architecture as the base model. We will then see how the Box Convolution layer compares in the semantic segmentation problem with PSPNet [22] using the PASCAL VOC 2012 dataset [5]. Lastly, we will see how the Box Convolution layer deals with the problem of object detection, using SSD [11], on the PASCAL VOC 2007/2012 dataset [5]. With the latter two problems of object detection and semantic segmentation segmentation, on a higher level, one would expect that the differences would be more palpable between the Box Convolutional layer and the regular Convolutional layer. The dynamic structure and size of the filters being able to convolve around specific parts of the image seems perfectly suited for the segmentation and object detection problem.

2. Related Work

2.1. Alternative Convolutions:

For work related to the box convolution idea, we will look at alternatives of the standard square convolutions that are typical in image related problem as described by Dumoulin and Visin in [4], in particular, dilated, transpose, and separable convolutions. A common link between all these alterations, including the Box Convolution, is that they all

try to deal with the problem with excess computational cost of dealing with high resolution or dense images through reducing the number of parameters or the computation required.

2.1.1 Dilated/Atrous Convolutions

With the introduction of the problem of semantic segmentation, there has been a difficulty balancing out the ideas of integrating contextual information while still providing dense predictions for the outputs in full resolution. The two ideas are contradictory at hand because the former involves either successive pooling or subsampling layers. Originally designed for image classification, the standard square convolution was not suited for dense prediction of each pixel for the segmentation problem. A new approach of tackling this problem was introduced in 2015 by Yu and Koltun called the dilated convolution [20]. The dilated convolution is similar to the regular convolution except that for any size kernel to convolve, there are only 9 parameters. For example, for a 7x7 dilated convolution, the parameters to be element-wise multiplied to the image are the pixels in the corner, center, and the midway point between the corners on the edges. A result of this design is that the dilated convolution will have a larger effective receptive field due to the dilation while having much fewer parameters than a standard convolution. This results in better understanding of contextual information at a fraction of the cost, which allows for high-resolution opertion throughout the network.

2.1.2 Transposed Convolutions

Again due to the nature of the semantic segmentation problem, the images at hand are likely to be very high resolution, and as stated before, since we are predicting the class of each pixel, the problem requires dense calculation of each pixel in a high resolution picture. In order to deal with this problem, Long, Shelhamer, and Darrell came up with the Fully Convolutional Network [17]. The FCN performs both downsampling and upsampling of a given feature mapping in order to reduce computation and the number of parameters while also maintaining the same effective receptive field. In order to do the upsampling, they used the idea of a transposed convolution. The transpose convolution was first introduced in the literature as Deconvolution by Zeiler et al. in 2011 with the paper "Deconvolution Networks" [21]. Instead of a simple max unpooling layer, which does not involve any learnable parameters, the transpose convolution is a learnable upsampling method. For a given input image, each pixel is scalar multiplied to a filter of a certain size in order to increase its output size with any overlaps being summed together.

2.1.3 Separable Convolutions

Separable convolutions is another way to reduced the computational cost of operations while maintaining the same output dimensions. First introduced in by Sifret and Mallat in 2014 [18], separable convolution as the name implies separates a typical convolution into to separate convolutions that involve fewer operations. Although there are other variations of separable convolutions, Sifret and Mallat's implementation involves a depth-wise convolution and then a point-wise convolution. The depth-wise convolution consists of having a filter for each filter dimension, which simply alters the image dimension based on the stride. The point-wise convolution will then take a 1x1xC with C being the number of filters of the output of the depth-wise convolution. By choosing the number of point-wise filters, the number of output channels can be chosen as with a normal convolution. By separating out these two operations, this method reduces the number of operations by a significant magnitude.

3. Methods

We will now go in depth with the workings of the Box Convolution starting with the idea of an integral image.

3.1. Integral Image

As stated in the introduction, the Box Convolution relies heavily on the concept of the integral image that has been around for a while in the field of computer vision. First introduced in 1984, it was popularized by the famous 2001 paper by Paul Viola and Michael Jones: "Rapid Object Detection using a Boosted Cascade of Simple Features," [19] which is one of the landmark papers on facial recognition. The integral image is one of the "simple features" question, and it is computed by summing over pixel values at the left, above, or both of a given pixel. For example, for a pixel at location x, y,

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1}$$

where $ii(x, y)$ is the value of a pixel at x, y for an integral image and $i(x, y)$ is the pixel value of the original image. Thus, each pixel value of the integral image represents the total sum of pixel values for all pixels to the left and above it including itself.

From the integral image, the area of any rectangular section of the integral image can be calculated easily by four array references. As you can see in Figure 1, the area for box D can be calculated through 3 addition/subtraction operations: the value of the integral image at 4 plus the value at 1 then subtracted by the values at 2 and 3. Alternatively, this operation can be thought of as $(A + B + C + D) + (A) - (A + B) - (A + C) = D$. Thus, the average pooling

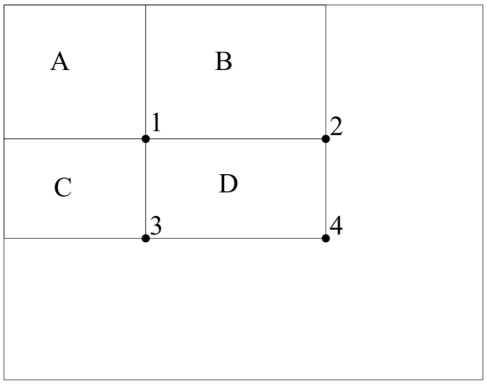

Figure 1. Example of an integral image. The value at 1 represents the area of the box A, 2 for the area $A + B$, and 3 for $A + C$, and 4 for $A + B + C + D$. [19]

operation, of which the Box Convolution layer is built off of, is a relatively inexpensive operation.

3.2. Box Convolution

The Box Convolution layer works by creating filters from this box averaging filter or kernel. Thus, each filter only consists of 4 parameters $\theta = (x_{\min}, x_{\max}, y_{\min}, y_{\max})$. These parameters represent the offsets from a given pixel location (let's say (x, y)), which together represent the box of pixels to be averaged for the output pixel location. In math notation as per Burkov and Lempitsky [2], an input map or one input filter is defined over a discrete lattice $(\hat{I}_{i,j})_{i=1,j=1}^{w,h}$, where h and w are the dimensions of the image (thus, for a 64x64 image, $h = w = 64$). The box averaging kernel [2] with parameters θ is the following function over 2D plane $\mathbb{R}^2$:

$$K_\theta(x, y) = \frac{\mathbb{I}(x_{\min} \le x \le x_{\max})\,\mathbb{I}(y_{\min} \le y \le y_{\max})}{(x_{\max} - x_{\min})\,(y_{\max} - y_{\min})} \tag{2}$$

A pixel value within a given input map is defined as $I(x, y)$ with x and y denoting the pixel location. Since continuous values for x and y are extremely frequent through our parameters being non-integer numbers, Burkov and Lempitsky chose to round these values to the nearest integer or closest pixel:

$$\mathbf{I}(x, y) = \begin{cases} \hat{\mathbf{I}}_{[x],[y]}, & 1 \le [x] \le h \text{ and } 1 \le [y] \le w \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $\hat{I}$ denotes the input channels and I denotes its extension to the closest pixel.

The output mapping $\hat{O}$ of a given input pixel is thus given by

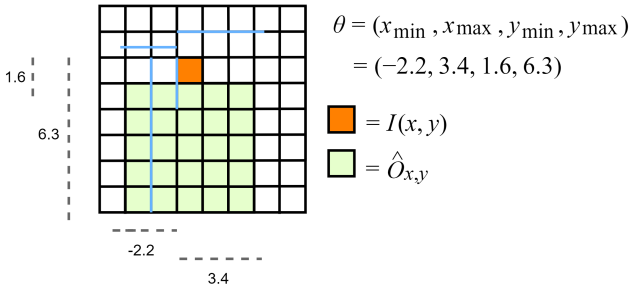

Figure 2. Calculation of a box convolution kernel with example parameters. The red area represents the target pixel, and the green area represents where the average pooling will take place [2]

$$\hat{\mathbf{O}}_{x,y} = \mathbf{O}(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \mathbf{I}(x + u, y + v) K_\theta(u, v) \mathrm{d}u\mathrm{d}v \tag{4}$$

$$= \frac{1}{(x_{\max} - x_{\min})(y_{\max} - y_{\min})} \int_{u+x_{\min}}^{x+x_{\max}} \int_{y+y_{\min}}^{y+y_{\max}} \mathbf{I}(u, v) \mathrm{d}u\mathrm{d}v \tag{5}$$

where $O(x, y)$ denotes the continuous result of the convolution.

The number of box convolutions per input mapping is a modeling hyperparameter. If there are M box convolution kernels per N input kernels, then the output dimension would be MN with $4MN$ parameters.

3.3. BoxResNet

For our image classification model, we implemented a modification of a standard 50 layer ResNet [9]. In a typical 50 layer ResNet, the bottleneck layer is the module that composes the architecture. It is composed of a 1x1 convolution layer, a 3x3 convolution layer, and another 1x1 convolution layer, with batch normalization after each convolution layer and ReLU output for the first two convolutions. The original input is then added to this output, so that the residual can be estimated before a final ReLU output. For our BoxResNet, the only thing that is changed is the number of output filters for the first 1x1 convolution and the replacement of the 3x3 convolution with the Box Convolution layer. For the latter, there are 4 box sizes for each output filter of the previous convolution, and the output of the first 1x1 convolution will be divided by four to match the number of output filters of the ResNet model. The rest of the architecture remains exactly the same and starts off with the same 7x7 convolution and a Max Pooling layer as that of a normal ResNet model.

3.4. BoxPSPNet

For the semantic segmentation model, for our baseline to compare to, we have used the PSPNet model [22]. Its ar-

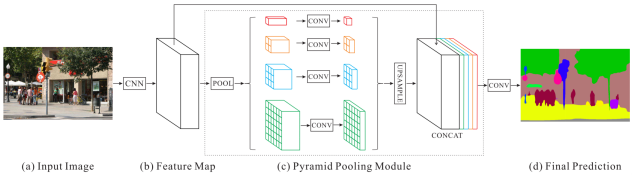

Figure 3. A pictorial representation of PSPNet as provided by [22]

chitecture is relatively straightforward. Given an input image, a ResNet module is used to get a feature map from the last convolution layer. This feature map is then fed through the Pyramid Pooling module, which is explained as the following. Through average pooling, four different sub-region representations are generated with dimensions of 1x1, 2x2, 3x3, and 6x6. Then these sub-region representations are run through a 1x1 convolution to condense into a single filter. Then the regions are upsampled back to the original pixel dimension of the feature mapping. The original feature mapping is then concatenated on top of the upsampled four regions, which is then fed through a final 3x3 convolutional layer to cut down the filter size and is followed by batch normalization, ReLU, and dropout with p = 0.1. The output is then put through a fully connected layer to generate prediction probabilities for each of the classes. The entire architecture can be seen in figure 3.

The Box Convolution analogue of the PSPNet will be relatively straightforward. The ResNet module is simply just replaced with our BoxResNet module. Thus, only the feature map will be altered as there is no way to downsample the number of filters through the Box Convolutional Layer like there is with a regular convolution. Thus, the rest of the architecture will be exactly the same, including the final 3x3 convolutional layer at the end of the Pyramid Pooling Module.

3.5. BoxSSD

Given that the focus of this paper is on a low number of parameters and computational efficiency, SSD [11] was chosen over Mask R-CNN [8] and Faster R-CNN [15]. SSD uses a base network, VGG here, as a feature extractor. Then extra layers are added: multi-scale feature maps, which decrease in spatial dimension and allow detection across a range of sizes, and convolutional predictors to produce a fixed set of predictors. These are based on a default set of predictor bounding boxes. The scores from each of these, which total 8732 detections over the 22 classes, are input into a multibox loss function which is a weighted average of confidence and localization loss.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Here the localization loss $L_{loc}(x, l, g)$ is a smooth L1 loss on the predicted (l) and ground truth (g) boxes, and the confidence loss $\frac{1}{N}(L_{conf}(x, c)$ is the multiclass softmax loss. These are averaged over matching default boxes, setting loss = 0 if there are no matches. We use $\alpha = 1$ from [11].

To create the BoxSSD architecture, we replace all convolutional layers with same padding in the VGG base network with the Box Convolution layer. A total of 12 layers are replaced, leading to a 55% reduction in parameters.

3.6. Coding Implementation Details

Our coding implementation relies upon the box_convolution package [1] in Pytorch [14]. For classification and segmentation, we use the code of resnet50 in Pytorch [14] as the baseline encoder network. The main change was to create our own BoxBottleneck layer and then incorporating it in the overall BoxResNet model. For object detection, the VGG backbone in SSD [3] was adapted to BoxSSD as explained previously.

We have used Pakhomov et al's implementation of PSPNet [12] on PyTorch, and our implementation of BoxPSPNet is based off of their implementation. The main changes in the model come from just swapping out the ResNet feature mapping layer with our own BoxResNet model and then making sure the dimensions match accordingly.

4. Dataset

For image classification, we used the Tiny Imagenet dataset, which consists of 200 classes. The training set consists of 100,000 images with 200 classes with 500 images each. We split this into the train and validation set of 90,000 and 10,000 images. Thus, the train set consists of 200 classes of 450 images, and the validation set consists of 200 classes of 50 images each. Moreover, each image is of dimension 3x64x64. Since we split the training set of the typical Tiny Imagenet challenge, it is likely that we are going to see worse performance than most standard ResNet results. Moreover, no data augmentation was done as our goal is to compare performance rather than to maximize performance.

For both semantic segmentation , we used the PASCAL VOC 2012 dataset annotated as outlined in [7]. The training set thus consists of 8498 images while the validation set consists of 2857 images. To create as test set, we divided the validation set into two to make 1428 for the validation set and 1429 for the test set. There are 21 classes total including the background class, and a 22nd class of borderline pixel labels ('void' per the dataset) that is ignored when calculating the loss. All the images have been normalized by the mean and standard deviation of the pixels in the training set. To make sure we have consistent mini-batches all the images were standardized to dimension 300x500 with black padding when necessary. To further prevent overfitting, we

chose to augment the data with random horizontal flips with probability of .5 and random color jitters for each image. With how PyTorch [14] approaches data augmentation, every original training image's brightness, contrast, gamma, and saturation is slightly altered every time it is included in the minibatch.

For Object Detection we also used Pascal VOC, here using the full 2012 train/val set of 11,530 images to train, containing 27,450 ROI annotated objects. The 2007 train and test set of 4952 and 2510 images were used for our validation and testing sets respectively.

5. Results

5.1. Classification Results

To evaluate performance, the output of the fully connected layer will be outputted with a softmax function, and then cross entropy loss is used to evaluate the result. Functions for both are defined as [6]:

$$P\left(Y=k|X=x_i\right)=\frac{e^{s_k}}{\sum_j e^{s_j}} \tag{6}$$

$$L_i=-\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right) \tag{7}$$

where s denotes the output of the fully connected layer, and s_i denotes the ith row/class of the output.

Stochastic Gradient Descent is used optimize with a batch size of 64. Stochastic Gradient Descent was used as it was the optimization algorithm used in the original ResNet architecture [9]. In terms of hyperparameters, for regular ResNet, we will use the standard learning rate of .025, momentum of .9, and an L2 regularization factor of 5e-4. We will also utilize step decay for the learning rate for every 20 iterations, with the learning rate decreasing by a factor of 10 every step. For BoxResNet, we will use a learning rate of .001, momentum of .9, and an L2 regularization factor of 1e-4 and will not use any step decay as it results in worse performance.

To compare the two models, we will use both Top-1 and Top-5 test accuracy to assess prediction and the average time to complete one epoch for each model to test efficiency of the algorithms.

Looking at the loss curves, we can see that initially, the BoxResNet performs slightly better and decreases both training and validation loss slightly faster than regular ResNet. In terms of training loss, the vanilla ResNet eventually catches up to the BoxResNet to plateau at around epoch 30. For the validation loss however, we can see that the loss becomes much more volatile in the later iterations.

The accuracy curves tell a similar picture. BoxResNet performs slightly better at the onset of training compared to vanilla ResNet, but they eventually converge to about

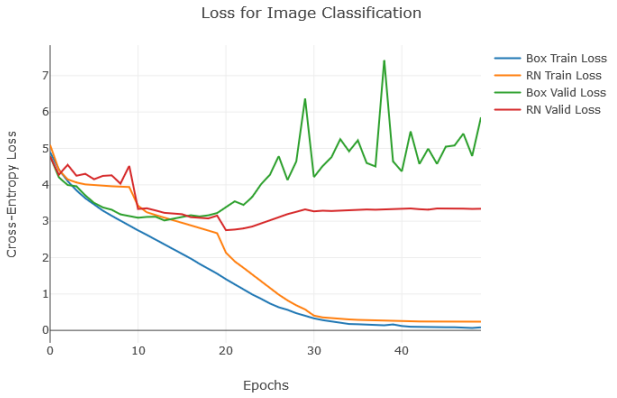

Figure 4. Loss Curves for Both HalfBox and ResNet50

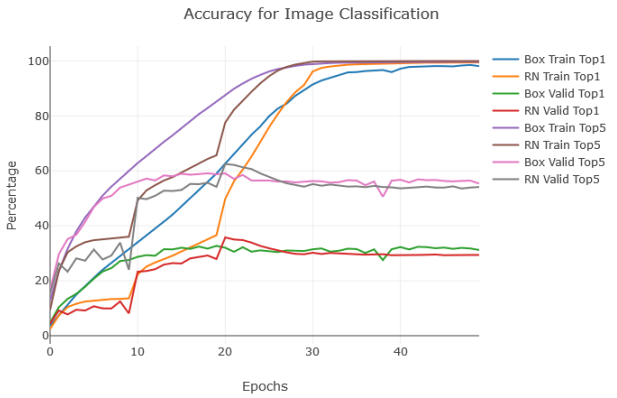

Figure 5. Accuracy Curves for Both HalfBox and ResNet50

the same accuracy at around iteration 30. We do note here that vanilla ResNet had a slightly higher validation accuracy peak than did BoxResNet, but this was a mostly negligible difference.

A summary of the main differences can be seen in Table 1. As discussed earlier in the paper, BoxResNet requires significantly less parameters than does vanilla ResNet. However, we can see that in terms of actual computation time, ResNet performed slightly better than BoxResNet. We suspect that this is because the Box Convolution layer requires the image to be converted into an integral image before calculation, which could be somewhat computationally expensive. Moreover, we can see that with respect to test accuracy, the BoxResNet did slightly better than vanilla ResNet. However, the difference is not very significant considering its relative absolute amount.

In addition, saliency maps of both models can be seen in Figure 6 and 7. We can see that compared to that of the vanilla ResNet, BoxResNet's saliency maps are more sparse in terms of allocating weights to pixels. In other words, BoxResNet doesn't place extreme emphasis on any

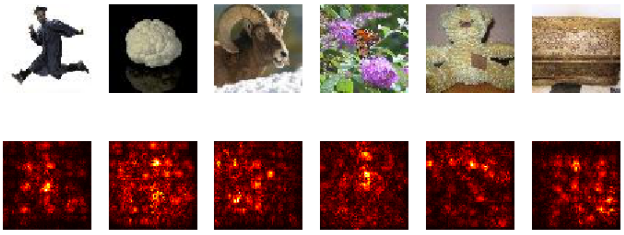

Figure 6. Saliency Map of Vanilla ResNet on Test Data

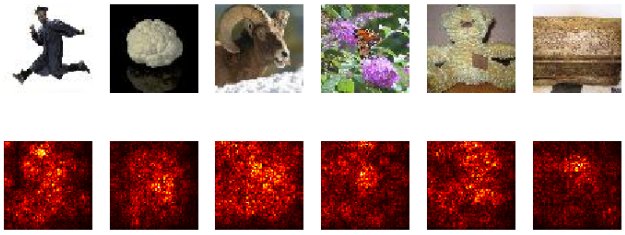

Figure 7. Saliency Map of BoxResNet on Test Data

Table 1. Classification Model Comparison

	ResNet	BoxResNet
# of Params	23,917,832	13,013,128
Epoch time, s	100.4	104.7
Test Top-1 Accuracy	29.34 %	31.54 %
Test Top-5 Accuracy	53.88%	57.62 %

given pixel or small condensed group of pixels in terms of class identification. This is likely due to the average pooling nature of the box convolutions instead of the element-wise multiplication done in a regular convolution layer. One could argue that this results in better generalization of images and acts as sort of a regularization on determining image shapes. This can be seen in the saliency maps. For the first picture of the man jumping in the air, the BoxResNet does a better job of capturing the man's overall shape compared to vanilla ResNet. One could argue this for all the pictures. In terms of actual prediction however, the results above show that the saliency map interpretation might not be very relevant all in terms of image classification prediction.

5.2. Semantic Segmentation Results

To evaluate performance, the output of the fully connected/spatial upsampling layer is outputted through a softmax function, which is calculated for each pixel in the input image, and the loss is calculated from a simple 2-dimensional cross entropy loss. To evaluate accuracy we will utilize both total pixel accuracy and the Intersection-Over-Union metric as did Zhao et al. in their PSPnet paper [22]. The mean IoU along with the IoU for the individual classes will be reported. IoU stands for Intersection over Union, where accuracy is determined by the number of pixels in the intersection of the predicted and target image is divided by the union of the two for each class. In terms of optimization, we used Adam Optimizer [10] with a batch size of 16. This is because of the CUDA limitations of the Box Convolution code as the output pixel amount is limited to $2^{31} - 1$ pixels.

For the hyperparameters of our base PSPNet model, we found that both models performed the best with the same hyperparameters, specifically a learning rate of 3e-4 and a weight decay factor of 1e-4 with β_1 and β_2 equaling .9 and .999 for the exponential decay rate for both the first and second moment estimates.

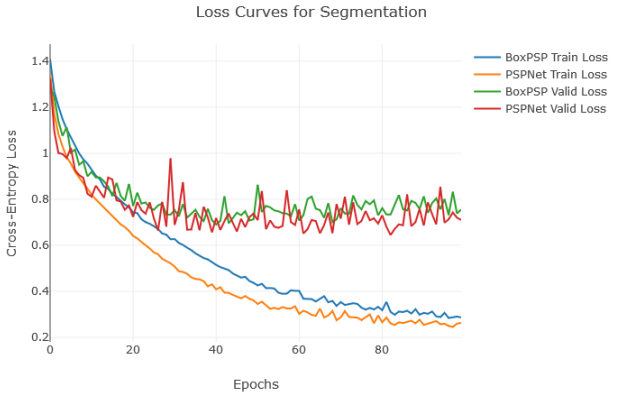

Figure 8. Loss Curves for Both BoxPSPNet and PSPNet

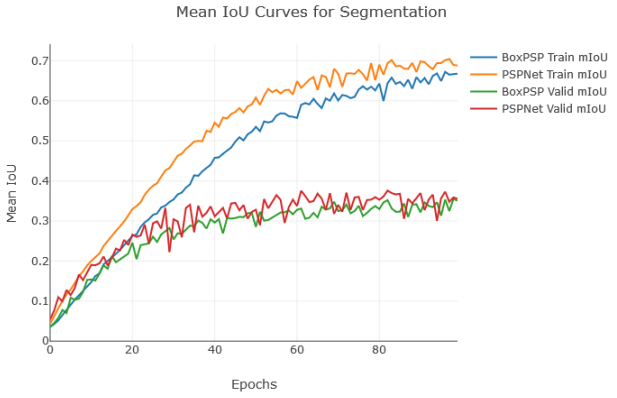

Figure 9. Mean IoU

As we can see from both the loss and the mean IoU curves for both BoxPSPNet and PSPNet models in Figures 11 and 9, the two perform relatively the same with similar behavior. However, it is evident that the baseline PSPNet model performs slightly better in terms of the speed at

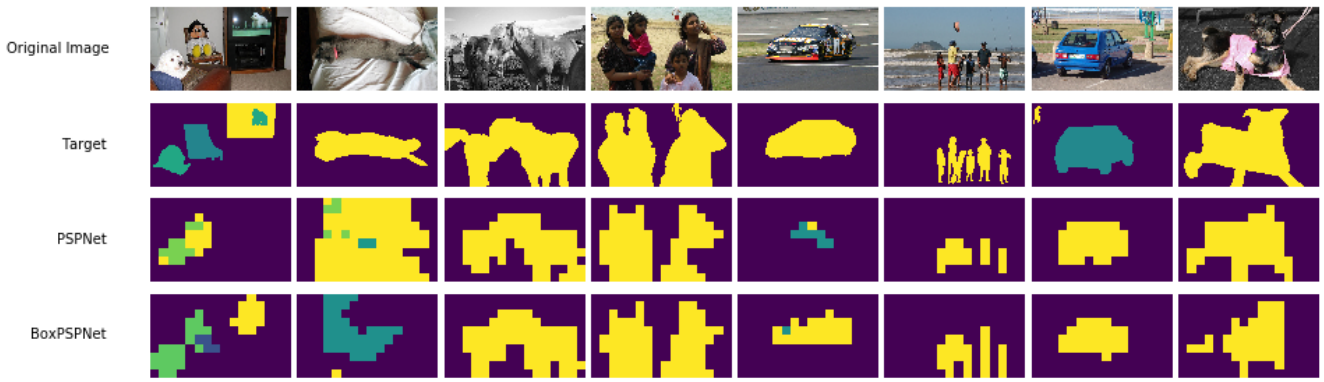

Figure 10. Segmentation Prediction Images

Table 2. Segmentation Model Comparison

	PSPNet	BoxPSPNet
# of Params	46,591,573	35,686,869
Epoch time, s	457.9	498.8
Test Total Pixel Accuracy	81.9	80.1

which the loss decreases. Looking at the class IoU values in 3, we can see the differences in where the BoxPSPNet performs better than PSPNet and vice versa. For example, we can see that PSPNet performs better across the board for predicting classes related to automobiles such as planes, buses, and cars while BoxPSPNet performs better at classifying chairs and cows, and the rest of the classes are relatively close in magnitude.

We can see some of the predictions of both the BoxPSPNet and PSPNet on some samples of the test set in Figure 10. We can see that the performance of the two are relatively close with minor differences. One possible conclusion we can derive from some of these differences in prediction is that the BoxPSPNet performs better in ascertaining certain shapes of the objects. Although both models struggled to strongly fit any of the predicted images, it seems as if BoxPSPNet did slightly better in determining more of the shapes of the objects than did PSPNet in this sample. It is also interesting to note that BoxPSPNet was able to parse out the three objects in the first picture in Figure 10 while PSPNet was not able to.

With prediction on the test set, we can see that BoxPSPNet performed slightly worse with both mean IoU and total pixel accuracy with 39.2 vs 37.1 and 80.9 vs 81.1 respective as seen in Tables 2 and 3. With such similar performance in accuracy, the deciding factor of the two should be its computationally efficiency. Despite the number of parameters being significantly reduced in BoxPSPNet versus PSPNet as was the case in BoxResNet vs ResNet, Box version of the two again performed slightly poorer in computation efficiency with the BoxPSPNet being slower by an average of 31 seconds. This is not too big of a surprise considering the performance of BoxResNet vs ResNet as outlined previously and the architecture design of PSPNet.

Figure 11. Loss Curves for Both BoxSSD and SSD

Moreover, we provided visualization of the learning process of the boxes over time for the first Box Convolution layer in BoxPSPNet. The boxes in the image show the size of the box relative to the center point of the pixel, and the colors indicate the importance of a particular box. The more important boxes are given a brighter color while the unimportant boxes will be almost transparent. In the first epoch, you can see that the size of the boxes are rather sparse and random, but by the 100th epoch, the boxes are of a much smaller size as noted by the concentration of bright colors near the center of the image.

5.3. Object Detection Results

For object detection, similar training and validation loss curves were observed, but BoxSSD achieved these results in less time. A learning rate of 0.001 was used to train both architectures, with weight decay of 0.0008 for SSD

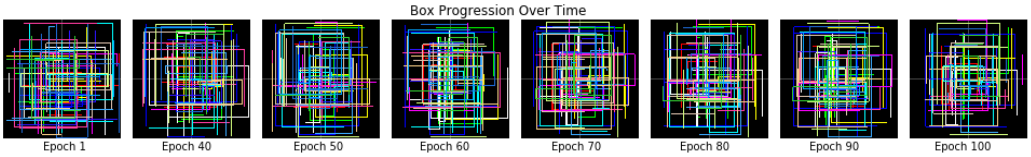

Figure 12. Segmentation Boxes Learning

Table 3. Segmentation IoU Results for the Test Set

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	bike	person	plant	sheep	sofa	train	tv	mIoU
PSPNet	48.5	54.4	32.3	31.3	21.5	62.9	49.0	53.9	5.8	30.2	11.2	43.8	25.1	47.8	58.3	21.2	38.9	20.1	50.9	33.0	39.2
BoxPSPNet	44.6	47.6	32.8	28.5	15.8	54.9	45.1	41.3	17.9	36.2	14.9	35.6	25.3	45.3	58.7	17.5	31.2	20.9	45.1	38.7	37.1

Table 4. Object Detection Model Comparison

	SSD	BoxSSD
# of Params	26,285,486	11,589,230
Epoch time, s	70.71	67.15
Train Loss	7.387	7.130
Validation Loss	7.308	7.318

and 0.0005 for BoxSSD, and training was run for 5000 iterations. Weight decay with gamma = 0.1 around iterations 1000 and 2000 did not significantly improve the results.

Although these experiments did not yield meaningful accuracy metrics, the fact that the models trained faster with comparable losses suggests that with further tuning, the box convolutional layer could cause a material improvement in efficiency in object detection.

6. Conclusion

Although the purpose of the Box Convolutional layer was to achieve greater computational efficiency while still maintaining the accuracy, our results in the above show more of the latter and less of the former. For both our image classification and semantic segmentation problems, we did see relatively similar performance in terms of convergence and accuracy, but we failed to see the computational efficiency that was promised in its design. This could be due to the creation of the integral image as mentioned before or could relate to the Box Convolution layer's relationship with residual connections found in both ResNet and PSPNet, which has ResNet embedded in its architecture. However, in image detection we did see fairly good gains in both computation time and training loss convergence with similar performance in accuracy, so future work could be done in implementing Box Convolution layers in other object detection models.

7. Contributions & Acknowledgements

In terms of what each person did for the project, David worked with the image classification and semantic segmentation problems outlined in the paper while Harry primarily worked with the object detection problem. David coded the model architecture for BoxResnet and BoxPSPNet while Harry coded the architecture for BoxSSD. We both wrote each part of our problems in this report while collectively writing all the other parts. In terms of GitHub repositories used, the Box Convolution functions were all taken from Burkov's Github repository [1] https://github.com/shrubb/box-convolutions/tree/master/box_convolution, the BoxResNet architecture was heavily modeled after the source code in PyTorch [14] https://github.com/pytorch/vision, and the BoxSSD net architecture is based on the source code in https://github.com/amdegroot/ssd.pytorch/. Thanks to Jim Fan for his guidance on this project of what kind of experiments to run, and why this is an important area of computer vision.

References

[1] E. Burkov. Pytorch code for the "deep neural networks with box convolutions" paper. `https://github.com/shrubb/box-convolutions`, 2019.

[2] E. Burkov and V. Lempitsky. Deep neural networks with box convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6211–6221. Curran Associates, Inc., 2018.

[3] M. deGroot. A pytorch implementation of single shot multibox detector. `https://https://github.com/amdegroot/ssd.pytorch`, 2019.

[4] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[6] J. J. Fei-Fei Li and S. Yeung. Lecture 3: Loss functions and optimization, April 2019.

[7] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011.

[8] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[12] D. Pakhomov, V. Premachandran, M. Allan, M. Azizian, and N. Navab. Deep residual learning for instrument segmentation in robotic surgery. *arXiv preprint arXiv:1703.08580*, 2017.

[13] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[15] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[16] E. Romera, J. M. Alvarez, L. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, PP:1–10, 10 2017.

[17] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, Apr. 2017.

[18] L. Sifre and P. S. Mallat. Ecole polytechnique, cmap phd thesis rigid-motion scattering for image classification author:, 2014.

[19] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[20] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[21] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, June 2010.

[22] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.