Multitask Object Detection with Dropout and Neural Code Metric Learning

Philip Pham
University of Washington
pmp10@uw.edu

Abstract

Using techniques and theory discussed in CSE 547, we try to detect objects in images with classification model and retrieve images with metric learning. The object detection task was seen as multi-label classification problem. Using multitask learning and dropout regularization, a neural network was trained to detect objects in patches. The last hidden layer of this network was used for neural code metric learning. Neighbors were retrieved with a ball tree.

1 Introduction

The COCO dataset consists of 330,000 images. Each image is annotated with bounding boxes which contain objects belonging to one of 80 categories (Lin et al., 2014). Using this dataset, I explored several techniques covered in CSE 547 including neural networks, non-convex optimization, metric learning, and nearest neighbor methods.

My objectives where:

- (*object detection*) given an image, extract patches that contain an object and classify the object;
- and (*metric learning*) define a distance metric that compares patches such that patches of the same categories are close together, and patches of different categories are far apart.

In particular, I explored different neural network architecures with various learning strategies before settling on a multi-layer perceptron model with dropout. For the metric learning, I attemped to use neural codes (Babenko et al., 2014). Then, the nearest neighbor search is done using a ball tree.

2 Methods and Materials

For object detection, regions were proposed with selective search (Uijlings et al., 2013). Features for each image were given based on Ren et al. (2015). A fixed length feature vector was extracted for each patch using pooling (He et al., 2014). For an example of region proposals by selective search, see Figure 1.

Using annotations provided by the COCO API, training, validation, and test datasets were constructed from the patches from the region proposoals with intersection over union (IoU) over 0.5 with a bouding box from an annotation. That patch was then labeled with the categories from the annotated bounding box.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Figure 1: Top 128 region proposals by selective search for COCO image 364814.

To simplify the problem, we were provided with two subsets of the COCO dataset: `tiny` and `small`. The `small` dataset was used. In this dataset, for each image a $256 \times 13 \times 13$ array of features was provided from a convolution neural network like the one described in Ren et al. (2015).

The training set consisted of 375,453 patches from 10,000 images. The test set was 74,276 patches from 2,000 images, and the validation set was 75,620 patches from 2,000 images. Each patch had 11,776 features. From here, the problem can be seen as a multi-label classification problem. Models were trained with PyTorch (Paszke et al., 2017).

The neural codes for the metric learning were extracted from the last hidden layer of the multi-layer perceptron used for object detection. To find nearest neighbors, a ball tree from `sklearn` was used (Pedregosa et al., 2011).

3 Results

The best unweighted mean average precision on the test set was 0.254112. With the learned metric, the nearest neighbor was of the same category 24% of the time.

3.1 Object Detection

The multi-layer perceptron with dropout performed best. It consisted of two hidden layers with 1,024 and 256 units, respectively. At each layer, dropout with $p = 0.5$ and ReLu activation functions were used.

3.1.1 Models

Three types of models were trained for object detection. You can see the results in Table 1. $L2$-regularization parameters and the number of hidden units were tuned to produce the smallest loss on the validation dataset. The idea behind using an MLP is that it enables weight-sharing in the hidden units. One might imagine certain animals or vehicles have common features that the model could learn together.

To use multitask learning, multi-label cross entropy was used for the loss function. The $L2$-regularization parameter λ by looking at the difference between validation and training loss across various training runs. While there were large differences, regularization was increased. Eventually, validation loss stopped improving. See Figure 2 for a plot of training and validation loss as function of training steps. See Figure 3 for a similar plot regarding the average precision score computed with macro averaging.

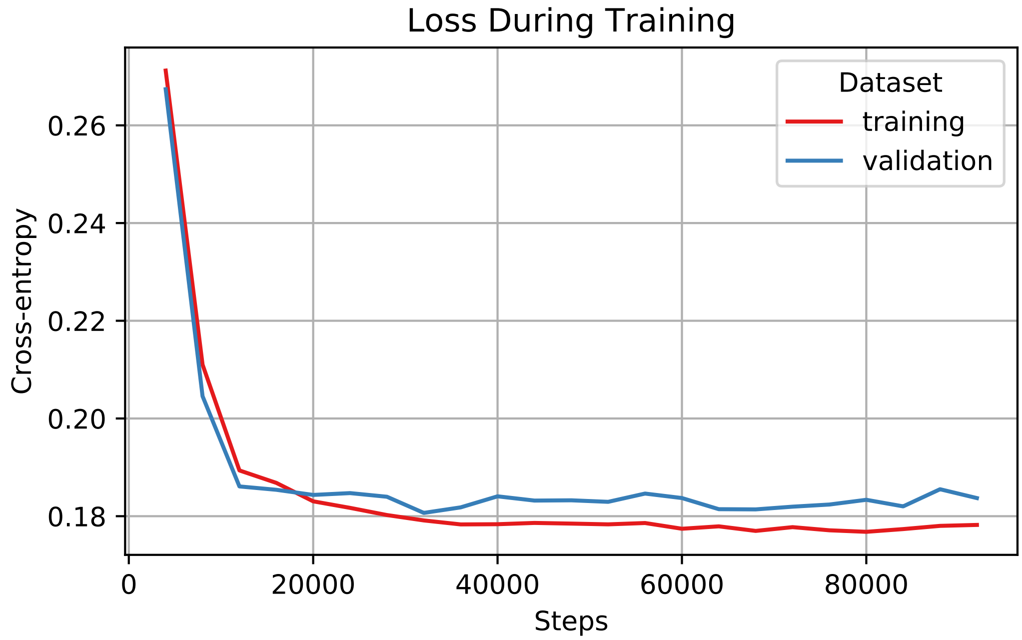

Figure 2: Loss was calculated after every 4,000 steps.

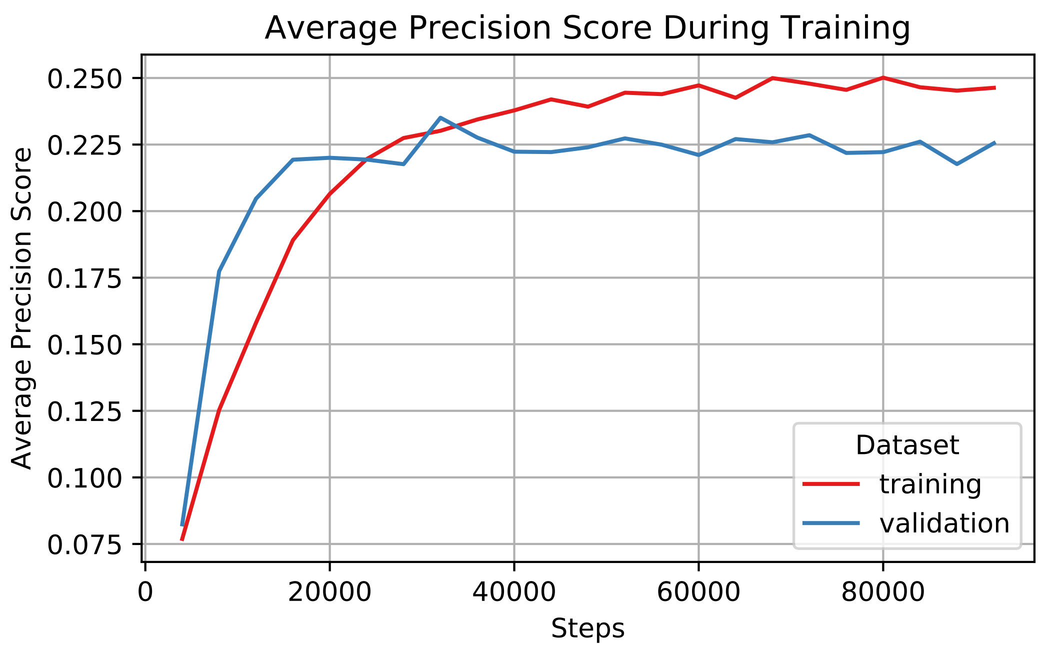

Figure 3: Mean average precision score was calculated after every 4,000 steps with a macro averaging strategy across categories.

Model	Average Precision Score	Loss
Linear	0.1750	0.1946
2-layer MLP	0.2198	0.1824
MLP with Dropout	0.2351	0.1807

Table 1: Metrics are computed against the validation dataset.

Label	Training Observations	Test Observations	Average Precision Score
bicycle	9950	1847	0.027483
car	56694	11010	0.362490
motorcycle	13163	2757	0.040598
airplane	15830	2529	0.077730
bus	16103	2985	0.087326
train	4568	873	0.040963
truck	29670	6045	0.117114
boat	15037	3179	0.057605
bird	28911	6232	0.314083
cat	12070	2778	0.500823
dog	15957	2629	0.240831
horse	22997	4966	0.272576
sheep	34010	6046	0.421672
cow	34359	7271	0.318933
elephant	23508	3972	0.454776
bear	3827	578	0.018140
zebra	25923	5720	0.578105
giraffe	19805	4243	0.642767

Table 2: Class breakdown of average precision score.

To obtain a further improvement, dropout regularization was applied. Dropout is a form of regularization that randomly zeroes outs a hidden unit during training (Srivastava et al., 2014). Intuitively, this prevents the network from seeing the same examples and avoids memorizing the training data.

3.1.2 Model Evaluation

When evaluated against the test dataset, the mean average precision was 0.254112 taken as unweighted average over the classes.

If one looks at the class breakdown in Table 2, one sees that have under-represented classes tend to have lower scores, for getting them wrong has less impact on loss.

3.2 Metric Learning

Babenko et al. (2014) and Krizhevsky et al. (2012) suggest that the top layer of a network may summarize an image, so I thought to try extract that layer to use as an embedding. Let f be my neural network. We can write $f = g \circ h$, where h takes the patch features and outputs 256-dimensional vector that forms the last hidden layer. Then, the distance metric for two patches P and P' is $d\left(P, P'\right) = \left\| h\left(P\right) - h\left(P'\right) \right\|_2$.

Once each patch was embedded into $\mathbb{R}^{256}$, a ball tree was used to enable fast nearest neighbor retrieval. If one fixes $K \in \mathbb{N}$, one has a classifier for category C by taking the percentage of the K nearest neighbors that belong to category C.

In light of this, we have that the average precision score for a class C is

$$\frac{1}{\#\text{ of patches of class } C} \sum_{\{P \,:\, \text{P has label } C\}} \frac{\#\text{ of } K \text{ nearest neighbors of } P \text{ category } C}{K}. \quad (1)$$

We take the unweighted average of this across categories at different values of K to obtain Figure 4.

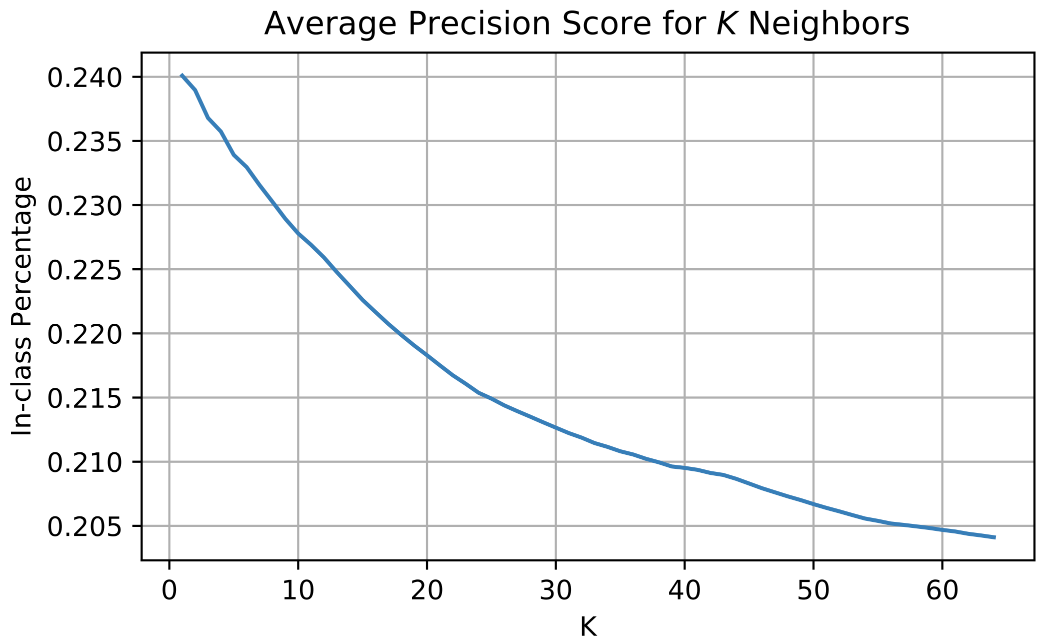

Figure 4: For K up to 64, the average precision score was calculated.

The nearest neighbor is most likely to be of the same category. Increasing K does not seem to help as the additional neighbors do not appear to more likely to be of the correct category.

Table 3 shows these results broken down by class. The nearest neighbor is of the same category about a quarter of time overall, but varies for some categories. These results roughly mirror those in Table 2 with categories that occur more frequently in the training data doing better. In some rare instances, like `bird`, `cat`, `dog`, and `cow`, the next few neighbors are more likely to be of the same category, too, so average precision score for these classes increases with K to a point.

4 Discussion

My model does not perform nearly as well as other models in the literature. For example, He et al. (2015) achieve a baseline mean average precision score of 0.415 with VGG-16. Using residual networks (ResNet-101), they achieved 0.484. With further refinements around input processing, they achieve 0.538.

Some possible improvements to improve object detection are harnessing more of the COCO dataset and exploring more complicated network architecures like convolutions and residual networks. These techniques would require more compute and memory.

My neural network currently generates the metric without any knowledge that the last hidden layer will be used as a metric. One promising way to incorporate this knowledge is a *siamese architecture* (Chopra et al., 2005). This architecure shares the weights between two identical networks. Each network takes a patch as input and our loss function is constructed such that small differences between patches of the same category and large differences between patches of different categories are favored.

It may be worth incorporating this loss function into a multitask learning framework, so weights are shared for the object detection model, too. One can image there are features that could be learned that are both useful for both retrieving similar images and classifying patches.

Label	Number of Neighbors K = 1	K = 3	K = 5	K = 10	K = 15
bicycle	0.042793	0.046547	0.044257	0.043187	0.043919
car	0.314220	0.309779	0.307115	0.291930	0.287312
motorcycle	0.051215	0.049844	0.049047	0.046280	0.045134
airplane	0.061684	0.063662	0.062554	0.058600	0.058521
bus	0.106321	0.100110	0.094995	0.095543	0.091804
train	0.044499	0.040379	0.037824	0.034981	0.037165
truck	0.147673	0.134968	0.126489	0.121318	0.120335
boat	0.098957	0.088207	0.083908	0.075624	0.075603
bird	0.208440	0.216303	0.214121	0.208521	0.197925
cat	0.287394	0.301850	0.301360	0.302536	0.289599
dog	0.289124	0.296166	0.284507	0.271753	0.266249
horse	0.242839	0.223252	0.217355	0.206529	0.198905
sheep	0.296334	0.295381	0.294618	0.289876	0.291456
cow	0.285961	0.286681	0.288553	0.289590	0.279952
elephant	0.256494	0.251366	0.255687	0.261892	0.267474
bear	0.127273	0.101818	0.097455	0.082364	0.073576
zebra	0.379962	0.376348	0.370869	0.351582	0.333310
giraffe	0.350071	0.340798	0.336445	0.339981	0.335691

Table 3: For each patch, the percentage of neighbors belonging to the same one was calculated and then averaged.

References

Babenko, A., A. Slesarev, A. Chigorin, and V. S. Lempitsky (2014). Neural codes for image retrieval. *CoRR abs/1404.1777*.

Chopra, S., R. Hadsell, and Y. LeCun (2005). Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, Washington, DC, USA, pp. 539–546. IEEE Computer Society.

He, K., X. Zhang, S. Ren, and J. Sun (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR abs/1406.4729*.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *CoRR abs/1512.03385*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, USA, pp. 1097–1105. Curran Associates Inc.

Lin, T., M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014). Microsoft COCO: common objects in context. *CoRR abs/1405.0312*.

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

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

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*(1), 1929–1958.

Uijlings, J. R. R., K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders (2013, Sep). Selective search for object recognition. *International Journal of Computer Vision 104*(2), 154–171.