5-24-2019

An Evaluation of VGG16 and YOLO v3 on Hand-drawn Images

Lee Hoang
Portland State University

An evaluation of VGG16 and YOLO v3 on hand-drawn images

by

Lee Hoang

An undergraduate honors thesis submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Science

in

University Honors

and

Computer Science

Thesis Adviser

Melanie Mitchell, Ph.D.

Portland State University

Lee Hoang

10 January 2018

An evaluation of VGG16 and YOLO v3 on hand-drawn images

Abstract

This thesis evaluates the accuracy and performance of VGG16, a convolutional neural network (CNN), and YOLO v3, an object detector, on a dataset of 1000 hand-drawn images. Unlike with photographs, which possess high amounts of detail, sketches tend to lack much detail aside from the freehand lines that comprise them. This is further detailed in prior works about Sketch-based Image Retrieval (SBIR) [10], a classification task to map photographs to sketches; and SketchParse [10], a CNN that analyzes sketch features and assigns captions. In this paper, I show the differences in classification accuracy between VGG16 and YOLO v3. The former model, pretrained on ImageNet, showed a test accuracy as high as 79.6%. On the other hand, YOLO v3, pretrained on MS COCO, performed worse; it misclassified objects in the dog category and across all categories, made no detections on several images.

Introduction

Object recognition is a task in machine learning that involves identifying objects in images while object detection focuses on identifying and locating instances of an object in an image. Figure 1 shows an example of these two tasks using two sketches (dog and horse) from my dataset. With object recognition, a box is drawn around the image and a label is put on it, but

with object detection, boxes are drawn on all detected objects within the image and labels are added for each box.

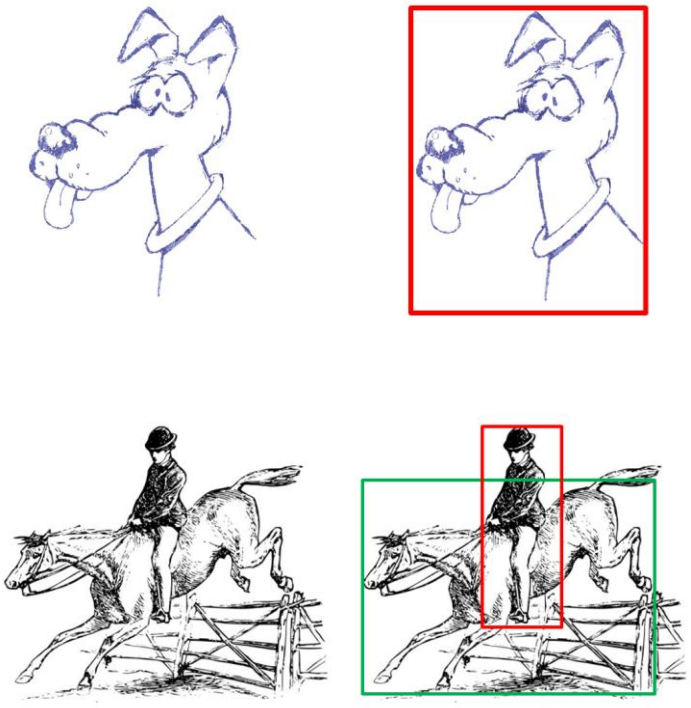

Figure 1: Object recognition (top) vs. Object detection (bottom). The original sketches are shown to the left and the results are shown to the right.

Photographs are usually filled with color [4] and detail that makes it easy for object detectors to locate objects such as dogs or cats. Hand-drawn images, unlike photographs, are composed only of black and white lines and lack much detail. Adding to that, the artistic quality of these sketches, including sketches drawn by observing photographs [6], varies depending on the skill level of the artist. This presents a unique problem for object detectors as they have mostly been trained on colored photographs.

Two well-known object detectors are YOLO (You Only Look Once) [8] and Faster-R-CNN [9]. Both have the same goal of detecting one or more objects in images, but they differ in

how they work to perform the tasks. While YOLO relies on a single convolutional layer to detect objects in one run [8], Faster R-CNN uses region proposal networks [9] combined with multiple convolutional neural networks to accomplish the task. In the associated papers [8, 9], both object detectors were trained on color photographs.

Other papers [10, 12] instead used datasets comprised of sketches in order to test new parsing frameworks. Like with Faster R-CNN, SketchParse utilized a dual-layer convolutional neural network, except the purpose of this two-layer network was to help classify sketches by category. In addition, the authors of SketchParse mentioned a procedure to turn photographs into sketches in order to create images for their dataset.

The goal of my research was to analyze the accuracy of VGG16, an object recognition network, and YOLO v3, an object detector, on a new dataset of hand-drawn images and compare the two. Before that, I carried out a small experiment with VGG16, an object recognition network, in order to further understand the mechanics. The results from this object recognition experiment were used to guide the object detection tests, which evaluated the test set accuracy. The final experiment used the 4096 outputs of VGG16's second-to-last fully connected layer to train and test ten support vector machines (SVMs). My hypothesis was that regardless of the method used to detect the hand-drawn images, VGG16 and YOLO v3 would perform poorly since they had been trained mostly on photographs.

Related Work

Sketches are abstract forms of artwork that lack detail and vary in appearance [10], causing difficulty for machine learning networks trained to recognize color photographs. There is, however, a body of literature describing how machine learning networks can be adapted to recognize and parse hand-drawn sketches. Sarvadevabhatla et al. describes in their paper how a convolutional neural network like SketchParse can be configured to detect and parse hand-drawn sketches by analyzing the salient features. Using two layers, one containing object categories and another containing expert sub-networks in super-categories, SketchParse can parse freehand sketches and auto-caption images. This contrasts with my choices of neural networks, which are by default trained on photographs from the ImageNet [11], MS COCO [5], and PASCAL VOC datasets.

In fact, Sarvadevabhatla et al. explained that to train SketchParse, they took images from PASCAL-Parts, a subset of the PASCAL VOC2010 dataset that provided "semantic part segmentation," and used a tool to transform the photographs into sketches so that the model could be trained on the "sketchified" images. To test their network, they used freehand sketches from two datasets, the TU-Berlin and Sketchy datasets, which consisted of 20,000 sketches across 250 object categories and 75,471 sketches across 125 categories, respectively. The networks that I used, however, differ in their training and testing methods; for example, the VGG16 network was retrained and tested on my new dataset of sketches, while the YOLO v3 network was directly tested on my dataset without retraining.

With Sketch-based image retrieval (SBIR), a classification task that matches photographs to sketches, Xu et al. [12] elaborated on how difficult it was to set up this field due to the "cross-modal gap" that exists between pictures and sketches. This gap refers to free-hand sketches in the

sense that they are abstract and lack a close resemblance to original objects shown in photographs. Prior work, as Xu et al. explained, had ignored the gap. In the text, Xu et al. conducted an experiment involving SBIR methods and cross-modal subspace learning on two sketch-based datasets, each one having sketches of shoes or chairs and their associated "ground-truth" photographs. According to the author, the shoe dataset had 419 photo-sketch pairs with three subclasses and the chair dataset had 297 photo-sketch pairs with six subclasses. Each of these datasets was created by pulling online photographs from sites such as Amazon and Ikea [13] and matching them with abstract, free-hand sketches created by volunteers [13]. Due to the abstract nature of these sketches, SBIR has many "barriers to entry" [13].

In contrast to Xu et al.'s work, my work includes not just free-hand sketches with sparse detail, but also contour-based sketches. The latter sketches possess a level of detail that allows them to resemble the original objects in a similar manner to a photograph. On another note, I am not performing SBIR on my dataset due to a lack of photographic look-alikes for the sketches; rather, I am performing an evaluation of two different types of machine learning networks to see how they would perform on hand-drawn images.

Background

Throughout this paper, I will be defining and using a range of terminology to refer to aspects of my experiments.

A convolutional neural network (CNN) is a neural network of "neurons" that consists of one input layer, one set of feature layers, and one set of classification layers [2]. Within the feature layers are convolutional layers, ReLU (Rectified Linear Unit) layers, and max pooling layers. The convolutional layers are responsible for passing the input images through filters in

order to obtain certain features from them, while the ReLU layers, also known as "activation" layers, map weight values to either 1 or 0 in order to maintain scaling. Max pooling layers are responsible for speeding up the processing time by decreasing the number of parameters the network needs to learn. Once the inputs are propagated through these hidden layers, they arrive at fully connected layers, which form part of the classification layers. From here, the network performs classification and the softmax layer, which is the final layer of the network, provides the output to the network.

When training a CNN, I train it for a specified number of epochs. An epoch is a single pass through the training set, which is a subset of a dataset used to train the CNN. For every epoch, the network will pass the input images, along with associated weights, through its hidden layers in a process called forward propagation. After this step, the network, based on the output that it gets, will proceed to adjust its weights and "back-propagate" the new weights to the beginning of the network. This process is repeated for as many epochs as required. Training from scratch will usually involve a large number of epochs in order to increase training accuracy.

Training a neural network can be done either with the entire training set or with a subset of the training set known as a “batch.” The latter method is called batch training. Changing the number of batches can affect the stability and the accuracy of detections/classifications.

After evaluating the network, I compile my results into a confusion matrix, which is an N x N table that shows, for each class N in my dataset, the number of correct predictions and the number of incorrect predictions. The diagonal from the top left to the bottom right shows the number of correct predictions for each class.

YOLO (You Only Look Once) and YOLO v3 are object detectors. This means that they take images and detect objects within the image instead of classifying the image according to a class. To be more specific, YOLO uses a single neural network pretrained on ImageNet [3] to predict bounding boxes, which define where the object is in the image, and class probabilities [8]. YOLO v3 is an improved version of YOLO that works like the original network, but it runs faster and detects small objects better [7] due to a new and improved network. During pretraining, the network was pretrained on 224 x 224 images, but during detection, YOLO works on images that are double the resolution.

A Support Vector Machine (SVM) is a classification model that assigns new datapoints to a class depending on their distance to a set of “support vectors.” These support vectors are datapoints that lie margins close to the separating line [1]. This line separates the two classes. More importantly, an SVM uses the "kernel method" to project data into higher dimensions (i.e. from 1-D to 2-D) in order to classify non-linear data. Once the separation line is drawn, the data is brought back to its original dimension. In my experiment, I am using a linear SVM, which acts like a linear classifier. Linear classifiers draw a straight line between datapoints. Typically, it is a binary classifier where it assigns a 1 or a 0 to a result, but here, I am using a multi-class classifier.

Methods

To prepare for the experiments, I prepared a custom dataset of one thousand hand-drawn images from open-source websites such as Openclipart.org and Pixabay. The dataset contained ten categories (dog, cat, bird, plane, ship, horse, bike, chair, car, elephant) and for each category there were one hundred images, each one in either the .JPEG or the .PNG file format. Prior to

running the experiments, I converted the image files to the non-transparent .JPEG format to remove transparency and make it easier for the machine learning networks to analyze the images.

The first experiment, which involved the VGG16 network [11], was to evaluate the network's accuracy on my dataset. VGG16 is an object recognition network that uses a convolutional neural network (CNN) with 16 layers [11] to recognize a 224 x 224 image as input. It possesses 3 fully-connected layers, the last of which is a softmax layer that contains the output features corresponding to each of the ten image classes. It is pretrained on ImageNet [3].

After collecting my images and organizing them into a dataset, I split the dataset into a training set and a test set of five hundred images each before rewriting parts of the code that I obtained from Kaggle to accommodate my dataset. Please see the original Kaggle code at https://kaggle.com/carloalbertobarbano/vgg16-transfer-learning-pytorch. To ensure that modified code behaved as expected, I conducted some test runs. While test runs were conducted on my CPU-based laptop, the final experiment was conducted on a machine with a Nvidia GTX 1070 Ti GPU in order to speed up test times. On a CPU-based machine, it took at least twelve hours to retrain VGG16 for fifty epochs, while on my GPU-based machine, it took only five minutes on average to retrain the network for the same number of epochs. The performance measure for this experiment is the number of correctly recognized images.

In the second experiment, I used the YOLO v3 object detector [7] and evaluated its accuracy on my test-set. For each class of images, I measured YOLO v3's detection accuracy based on two metrics: 1) the number of images with detections, and 2) whether or not the object detector made correct detections. YOLO v3 is an improved version of the original YOLO object detector. The version of YOLO v3 that I obtained had weights pretrained on MS COCO [5], a dataset with "91 common object categories" with a total of "328,000 images." Please see the

original code on https://github.com/ayooshkathuria/pytorch-yolo-v3. All detections were done on a CPU-based laptop, which had no problem completing fifty detections in the span of 140 seconds on average. The high detection speed was due in part to YOLO's single convolutional layer [8].

The final experiment returned to using the VGG16 object recognition network. Instead of using batch training like with the first experiment, I switched to using a multi-class SVM where the inputs were the 4096 outputs from the second-to-last fully connected layer. The intention was to determine, all other things equal, whether using the SVM instead of VGG16's softmax classification layer made any difference in accuracy over the standard batch training algorithm used for VGG16 [11]. As with the first VGG16 experiment, I ran my final experiments on a desktop machine with a Nvidia GTX 1070 Ti GPU.

Results

Batch training, 30 epochs

I ran an experiment with batch training (ten images) and for thirty epochs. After the training completed, the resulting accuracy over all of the training data was 98.8%, which was obtained by averaging the class accuracies. After the last epoch completed, I had a best training accuracy of 98.80%.

The initial test set accuracy, which was obtained from averaging class accuracies across each of the ten categories in the test set, was 77.0%. Following conclusion of training and a subsequent evaluation, VGG16 reported an average test set accuracy of 78.4%.

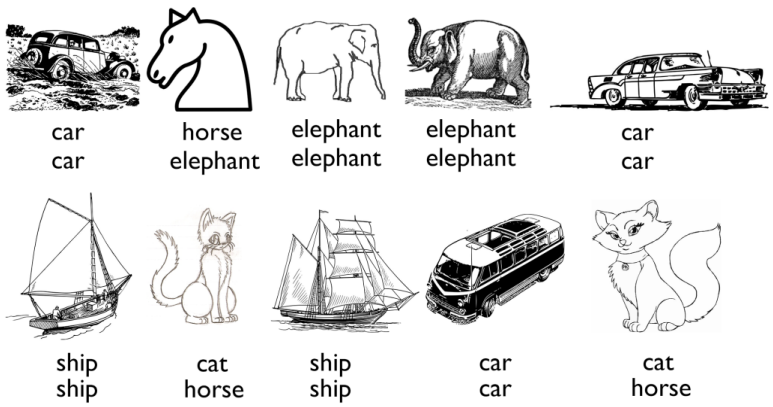

Figure 2.1: Ground truth (top text) and predictions (bottom text) for a sample batch of 10 images. All images are from the test set.

The figure above is an example of the classifications shown from evaluating VGG16 over my test set. It shows ten images, each one with correct labels and VGG16's predictions. Note how it misclassified a horse as an "elephant" and both cats as "horses."

Predicted class/ Actual class	**Bikes**	**Birds**	**Cars**	**Cats**	**Chairs**	**Dogs**	**Elephants**	**Horses**	**Planes**	**Ships**
Bikes	47	0	1	0	0	0	0	0	1	1
Birds	1	34	0	0	1	3	7	4	0	0
Cars	1	0	46	0	3	0	0	0	0	0
Cats	1	1	0	27	1	6	4	9	1	0
Chairs	1	0	2	3	38	0	2	2	2	0
Dogs	0	0	0	3	0	39	4	4	0	0
Elephants	0	0	2	2	1	1	41	3	0	0
Horses	1	0	1	2	0	3	3	40	0	0
Planes	2	0	4	1	2	0	0	2	36	3
Ships	0	0	2	0	3	0	0	0	1	44

Figure 2.2: Confusion Matrix for the test data of Experiment 1. Each category in the test set had 50 images.

Bikes	Birds	Cars	Cats	Chairs	Dogs	Elephants	Horses	Planes	Ships
94%	68%	92%	54%	76%	78%	82%	80%	72%	88%

Figure 2.3: Accuracies across the ten classes in the test data for Experiment 1

The confusion matrix, shown in Figure 2.5, is created such that the rows refer to predicted classes and the columns refer to actual classes. In each diagonal cell is the number of correct predictions for each class out of 50 images per category. Below the matrix is a table that shows the class-by-class accuracies, represented as Figure 2.6.

Batch training, 50 epochs

As with the first experiment, I used a batch size of ten, but instead trained the network for fifty epochs. Following fifty epochs of training, The final training accuracy was 99.2%. On my GPU-based machine, it took seven minutes and forty-seven seconds to finish this experiment. The initial test accuracy prior to training was 78.4%. Following 50 epochs, the final test set accuracy across all categories was 79.6%. As with the first experiment, the training accuracies were in the 90% to 99% range while the test accuracies were in the 75% to 78% range. Compared to the previous experiment, the test accuracy was 79.6% against 78.4%, which was an improvement of 1.2%. The accuracy shown here is the final average test accuracy after training the model for fifty epochs.

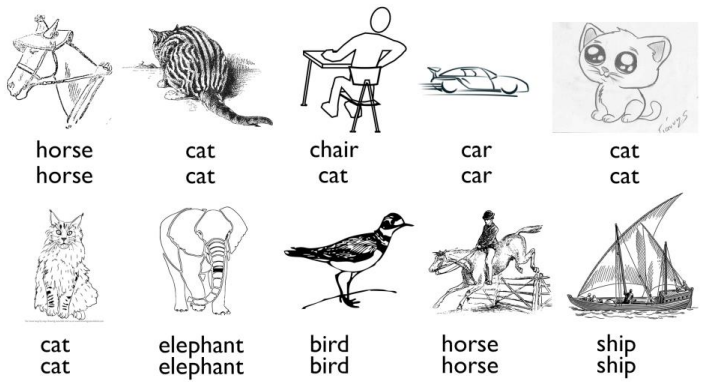

Figure 3.1: Ground truth (top text) vs. predictions (bottom text) for Experiment 2. This is a sample batch of images from the test set.

Predicted class/ Actual class	**Bikes**	**Birds**	**Cars**	**Cats**	**Chairs**	**Dogs**	**Elephants**	**Horses**	**Planes**	**Ships**
Bikes	44	0	1	0	0	0	1	1	3	0
Birds	0	39	0	2	0	2	3	3	0	1
Cars	1	1	47	0	1	0	0	0	0	0
Cats	0	0	0	38	0	1	1	8	1	1
Chairs	2	1	1	3	33	1	2	3	3	1
Dogs	0	2	0	5	0	39	1	3	0	0
Elephants	0	1	1	4	0	1	37	5	1	0
Horses	0	0	1	3	0	3	4	37	1	1
Planes	1	1	1	1	1	0	1	3	38	3
Ships	0	0	0	0	0	0	0	0	4	46

Figure 3.2: Confusion matrix on the test set of Experiment 2. Each category in the test set had 50 images each.

Bikes	Birds	Cars	Cats	Chairs	Dogs	Elephants	Horses	Planes	Ships
88%	78%	94%	76%	66%	78%	74%	74%	76%	92%

Figure 3.3: Accuracies across the ten classes in the test data for Experiment 2

As with the first experiment, the confusion matrix is shown before the per-class accuracy, which ranged from 66% to 94%. The average test accuracy over all of the class accuracies after 50 epochs was 79.6%, higher than the 78.4% after 30 epochs.

YOLO v3 Accuracy tests

After running a pretrained version of YOLO v3 (COCO) on my test set of 500 images, I noted each of the class-by-class accuracies as shown in Figure 4:

Bikes	Birds	Cars	Cats	Chairs	Dogs	Elephants	Horses	Planes	Ships
74%	46%	34%	10%	36%	8%	30%	12%	14%	8%

Figure 4: Class-by-class detection accuracy for YOLO v3

As with the VGG16 experiments, I ran the YOLO v3 detection experiments on the test set (500 images). Class-wise, YOLO v3 had the highest detection accuracy on bicycle pictures and the lowest detection accuracy on dog and ship pictures. For each category, YOLO v3 failed to detect some images and incorrectly labeled others. In the bicycle category, for example, one bike image was detected, but labeled as a "refrigerator." For dog pictures, only 8% of all dog pictures were correctly detected and labeled since most of them were either not detected or incorrectly labeled. The average test set accuracy over all ten categories was 27.2%.

SVM with VGG16's outputs

Predicted class/ Actual class	**Bikes**	**Birds**	**Cars**	**Cats**	**Chairs**	**Dogs**	**Elephants**	**Horses**	**Planes**	**Ships**
Bikes	35	2	3	1	0	0	4	2	2	1
Birds	1	35	0	3	0	5	1	5	0	0
Cars	3	1	33	0	1	0	0	0	8	4
Cats	0	13	0	19	0	10	4	4	0	0
Chairs	2	4	2	3	25	1	1	2	5	5
Dogs	0	7	0	2	0	25	5	11	0	0
Elephants	0	9	0	7	0	9	21	4	0	0
Horses	0	6	0	2	0	15	9	17	1	0
Planes	4	1	3	4	0	1	2	1	32	2
Ships	3	0	1	0	1	0	0	2	2	41

Figure 5.1: Confusion matrix on the test set of Experiment 3. Each category in the test set had 50 images each.

Bikes	Birds	Cars	Cats	Chairs	Dogs	Elephants	Horses	Planes	Ships
70%	70%	66%	38%	50%	50%	42%	34%	64%	82%

Figure 5.2: Accuracy across ten classes in the test data for Experiment 3.

Contrary to the first experiment, using a multi-class SVM with VGG16's outputs as its inputs resulted in a lower average test-set accuracy of 56.6%, although the classifier still achieved an accuracy of 82% when classifying ship pictures.

Discussion

Most of the dataset is comprised of images with a single object within them. This means that an object recognition model like VGG16 will perform better since it is trained to recognize pictures with a single object. On the other hand, I have discovered that while YOLO v3 was able to correctly detect some images in my dataset, it almost completely misidentified images in certain categories. For example, YOLO v3 misidentified objects in most of the detected dog images as “person” or “zebra,” and in one image containing a single dog, the object detector found a “person,” an “elephant,” and a “zebra.”

Since YOLO v3 has issues detecting certain objects such as dogs and ships, I am looking into ways to improve its accuracy. This includes tuning the learning rate and/or the momentum, both of which are hyperparameters. The same can be said for the SVM with VGG16's outputs. At the same time, I plan to add many more images to my dataset to see if more data can increase the accuracy of both VGG16 and YOLO v3 given the same parameters.

Conclusion

These experiments have shown that on a dataset comprised entirely of hand-drawn images, VGG16 and YOLO v3 differ in accuracy. While VGG16 performed relatively well on the test dataset for all categories, YOLO v3 ran into roadblocks detecting objects in the dog and ship categories; for each image, it either detected nothing or it detected the wrong type of object. This indicates that on hand-drawn images, object detectors like YOLO v3 have issues detecting objects within them. However, most of the images in my dataset had only one object within them, which could have affected the object detector's accuracy and the ability to detect within images. On the other hand, VGG16 performs well because of two factors: VGG16 is an object recognition model that recognizes images with one object and most of my dataset is comprised on images with a single object.

Acknowledgements

I would like to thank Dr. Melanie Mitchell, my advisor, and Li-Yun Wang, my teaching assistant, for their feedback and support throughout the project.

References

[1] A. Ben-Hur and J. Weston, "A User's Guide to Support Vector Machines," *Methods in molecular biology (Clifton, N.J.)*, vol. 609, pp. 223–39, Jan. 2010.

[2] "Convolutional Neural Network." [Online]. Available: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. [Accessed: 18-May-2019].

[3] "ImageNet." [Online]. Available: http://image-net.org/index. [Accessed: 18-May-2019].

[4] Y. Li, "Fine-Grained Sketch-Based Image Retrieval by Matching Deformable Part Models," p. 12.

[5] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, May 2014.

[6] S. Ouyang, T. Hospedales, Y.-Z. Song, and X. Li, "Cross-Modal Face Matching: Beyond Viewed Sketches," in *Computer Vision -- ACCV 2014*, vol. 9004, D. Cremers, I. Reid, H. Saito, and M.-H. Yang, Eds. Cham: Springer International Publishing, 2015, pp. 210–225.

[7] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767 [cs]*, Apr. 2018.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[10] R. K. Sarvadevabhatla, I. Dwivedi, A. Biswas, S. Manocha, and R. V. Babu, "SketchParse : Towards Rich Descriptions for Poorly Drawn Sketches using Multi-Task Hierarchical Deep Networks," *arXiv:1709.01295 [cs]*, Sep. 2017.

[11] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sep. 2014.

[12] P. Xu *et al.*, "Cross-modal Subspace Learning for Fine-grained Sketch-based Image Retrieval," *arXiv:1705.09888 [cs]*, May 2017.

[13] Q. Yu, F. Liu, Y.-Z. Song, T. Xiang, T. M. Hospedales, and C.-C. Loy, "Sketch Me That Shoe," p. 9.