InstaFashion: Clothing Detection and Classification with YOLO

Kevin Fry
Stanford University
kfry@stanford.edu

Xianming Li
Stanford University
xmli@stanford.edu

Vivian Yang
Stanford University
vivianca@stanford.edu

Abstract

In this paper, we develop an algorithm that allows users to detect and classify items of clothing within an image. Our algorithm, InstaFashion, is a deep neural network model built on the You Only Look Once (YOLO) version 1 architecture with 24 convolutional layers followed by 2 fully connected layers. We pretrain the convolutional layers on the Street2Shop open source images at half the resolution (224 $\times$ 224 input image) and then double the resolution for detection. For comparison, we use the VGG-16 pre-trained on ImageNet as a benchmark to assess performances on loss and mean average precision (mAP). Our best YOLO model achieves a test mAP of 0.719,outperforming the baseline sliding window model, which only achieves a mAP of 0.559. Further validating the correctness of our YOLO implementation, our test mAP is similar to what the original YOLO paper achieved on the PASCAL VOC dataset.

1. Introduction

Online shopping and e-commerce is an exponentially growing market. Retail sales world-wide, including both in-store and internet purchases, totaled more than $23.4 trillion in 2017. By 2021, e-commerce retail spending is projected to increase to nearly $4.8 trillion, more than double of what it currently is in 2018 at $2.8 trillion. Much of this purchasing is related to shopping for clothing items. However, finding exactly what you want from online shops is still not a solved problem. In this paper, we look at one task related to online shopping, the street-to-shop problem. Given a real-world photo of a clothing item, e.g. taken on the street, the goal of this task is to find that clothing item in an online shop.

We aim to build an object detection network using the YOLO architecture that takes a real-world image and detects as well as classifies the pieces of clothing. Such an application could be useful today, as consumers seek to conveniently locate items similar to what they see in images within online marketplaces. This task is challenging because of the discrepancies between real-world images that have noisy backgrounds with various perturbations of the clothing items and the stock-photo images of clothing items that exist online. For example, clothing will be worn on a person in street photos, whereas in online shops, clothing items may also be portrayed in isolation or on mannequins. Shop images are professionally photographed, with cleaner backgrounds, better lighting, and more distinctive poses than may be found in real-world, consumer-captured photos of garments. To deal with these challenges, we introduce a deep learning based methodology to learn a similarity measure between street and shop photos.

2. Literature Review

The machine learning community has created a wide body of work on image detection and classification. You Only Look Once was first introduced in 2015 as the state of the art real-time object detection algorithm. Prior work on object detection had repurposed classifiers to perform detection. YOLO approached detection as a regression problem optimizing for bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. YOLOs unified architecture is extremely fast and is able to achieve double the mean average precision (mAP) of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. By learning very general representations of objects, YOLO is able to outperform many other detection methods, including DPM and R-CNN.

3. Dataset

We adapted the dataset from the Street to Shop model (Kiapour et al., 2015) consisting of images of people photographed in everyday settings with bounding boxes (t, l, w, h) around items of clothing (Figure 1). The dataset has 18,000 images, 31,000 bounding boxes (averaging 2 bounding boxes per image, with a maximum of 17 bounding

Figure 1. Example image

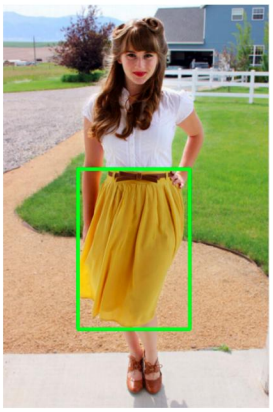

Figure 2. Example image

boxes for one image), and 11 classes: bags, belts, dresses, eyewear, footwear, hats, leggings, outerwear, pants, skirts, tops. Content wise, the Street to Shop dataset was reasonably appropriate for our needs; however, extensive preprocessing of the data was required to fit our specific application.

A sample data point from the dataset is as follows: {"photo": 2281, "product": 7871, "bbox": {"width": 112, "top": 335, "height": 204, "left": 59}}

4. Preprocessing

We conducted extensive preprocessing on the dataset to fit it to our novel application. Because the original dataset was intended to be used in a retrieval task, preprocessing was necessary in order to consolidate the images and bounding boxes into a tensor that could be used as input into the YOLO architecture.

4.1. Conversion

We first converted the training dataset into 3D tensor of size 18397 x 15 x 6 [number of images $\times$ max number of bounding boxes per image $\times$ (id, class, x_c, y_c, w, h)].

4.2. Standardization

After comparing YOLO, VGG-16, and RCNN architectures for our model and deciding to implement YOLO and VGG-16, we standardized the sizes of the images in our dataset to fit the needs of each particular architecture. YOLO v1 required size 448 x 448, and VGG-16 required 224 x 224. We cropped images to bounding boxes and then resized appropriately. In addition, we also filled in black margins vertically and horizontally as needed.

4.3. Alignment

In alignment with the amount of data we had as well as our computational resources, we allocated the dataset according to an approximately 90-5-5 split (Figure 2).

Partition	Number of Images
Training	16439
Validation	979
Test	979
Total	18397

Figure 2: Data Splits

5. The YOLO Model

We implemented YOLO v1 with a 7 x 7 grid and 5 bounding boxes per grid as in Redmon et al. We use pretrained weights trained on ImageNet for the first 20 layers of the model, and train the final four convolutional and two fully-connected layers on our dataset to fine-tune our model. We trained two models following this architecture: one with a static learning rate (1e-5), and one with a dynamic learning rate that changed over the course of training.

5.1. Hyperparameter Tuning

It is known that YOLO v1 suffers from unstable gradients early on in the training. This is due to the fact that it is trying to predict (x,y) coordinates for the boxes (this is remedied in YOLO v2 by instead calculating offsets from predetermined anchor boxes). As a result, Redmon et al. trained YOLO using a specific learning rate schedule with a smaller learning rate for the first few epochs, then increasing the learning rate in the middle of training, and then reducing the learning rate again towards that latter part of training.

Initially we used the same learning rate schedule, but found the gradients were still too unstable to train properly. We first tried lowering the learning rate (from the 1e-3 in Redmon et al. to 1e-4 and then 1e-5), but it was still too unstable to train more than a few dozen batches before getting NaNs . Then we experimented with two other hyperparameters: the l2 penalty and gradient clipping.

We tried several values of the l2 penalty (0.01, 0.1, and 0.2) and gradient norm clipping (0.25, 0.5, and 1) in combination with the lowered initial learning rates of 1e-4 and 1e-5. With a learning rate 1e-4 we got NaN losses for all values of the l2 and gradient clipping parameters. We also found that the l2 penalty was not enough to stabilize the training, even with the 1e-5 learning rate. With the lower learning rate we found that gradient clipping at any value stabilized our training, with or without any l2 penalty.

Not wanting to add unnecessary hyperparameters to our model, we decided to use only gradient clipping with norm 1 and discard the l2 penalty. We then experimented with several learning rate schedules. We first describes the ones that did not work:

- 1e-4 for the first 5 epochs, then raise it to 1e-3 over the next 5 epochs, then to 1e-2 over the following 5, training at 1e-2 for 60 epochs, 1e-3 for 30, then 1e-4 for the final 30.
- 1e-5 for the first 5 epochs, then raise it to 1e-4 over the next 5 epochs, then to 1e-3 over the following 5, training at 1e-3 for 60 epochs, 1e-4 for 30, then 1e-5 for the final 30.
- 1e-5 for the first 5 epochs, then raise it to 1e-4 over the next 10 epochs, then to 1e-3 over the following 10, training at 1e-3 for 50 epochs, 1e-4 for 30, then 1e-5 for the final 30.
- Finally we found a learning rate schedule that worked: 1e-5 for the first 15 epochs, then raise it to 1e-4 over the next 10 epochs, then to 1e-3 over the following 10, training at 1e-3 for 50 epochs, 1e-4 for 25, then 1e-5 for the final 25.

5.2. Training

We trained two versions of our YOLO model, one with a static learning rate, another with a dynamic learning rate schedule described above with the Adam optimizer. The static learning rate model had a learning rate of 1e-5 for the entire 135 epochs of training. This model was chosen partly for its simplicity and partly because we were more confident it would train fully without failing. We also trained the dynamic learning rate model because the more aggressive learning rate in the middle portion of training makes it more likely to find a better local optimum.

5.3. Baseline Model

For the baseline we implemented a simple sliding window model that performs image classification on each window using VGG16 model. We take the feature extractions from the KERAS pretrained VGG16 model and then train two fully-connected layers with 64 and 11 units, respectively, to learn class probabilities on our dataset. We trained this model for 100 epochs on the training dataset with the standard Adam optimizer. This very crude model is not very good, but is a decent baseline to compare our YOLO models against.

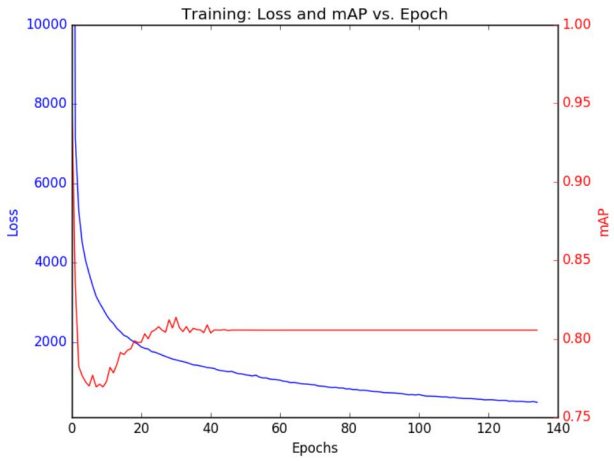

Figure 3. Static Training

6. Results

We wrote our own implementation of the YOLO loss function and the mean average-precision metric (mAP), and tracked them throughout the training of our models. Counter to our intuitions that the dynamic learning rate model, with its larger step size, would find a better optimum, it turns out that the static learning rate model performed better, achieving a test mAP of 0.72, while the dynamic learning rate model only achieved a test mAP of 0.69. They both significantly outperformed the VGG16 sliding window baseline, which only got 0.56 test mAP. Further validating the correctness of our YOLO implementation, our test mAP is similar to what the original YOLO paper achieved on the PASCAL VOC dataset.

6.1. Static

The static learning rate model achieved an mAP of 0.72 on the validation set after training for 135 epochs. We thought that would be the baseline for comparison with the dynamic learning rate model, which we trained next.

mAP: 0.7197916666666667

Loss: 5018.02041015625

6.2. Dynamic

Contrary to our expectations, after training for the same number of epochs the dynamic learning rate model achieved an mAP of 0.69 on the validation set, which is not as good as the static learning rate model.

mAP: 0.6895833333333333

Loss: 7850.24677734375

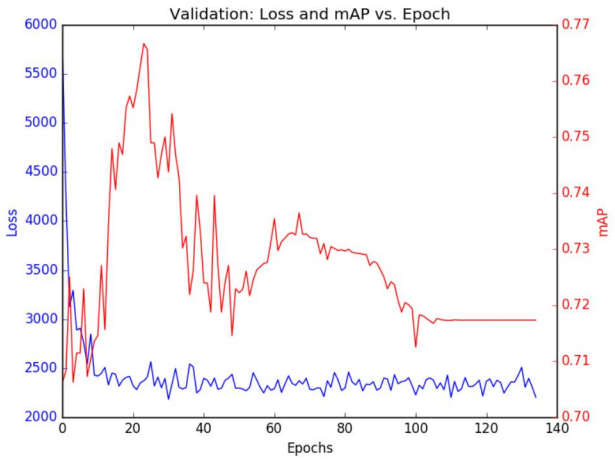

Figure 4. Static Validation

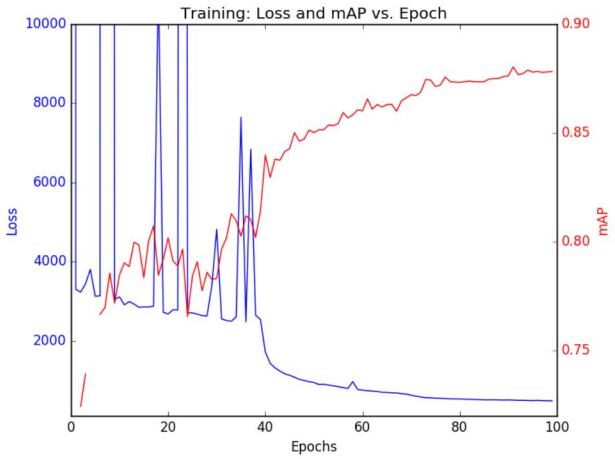

Figure 5. Dynamic Training

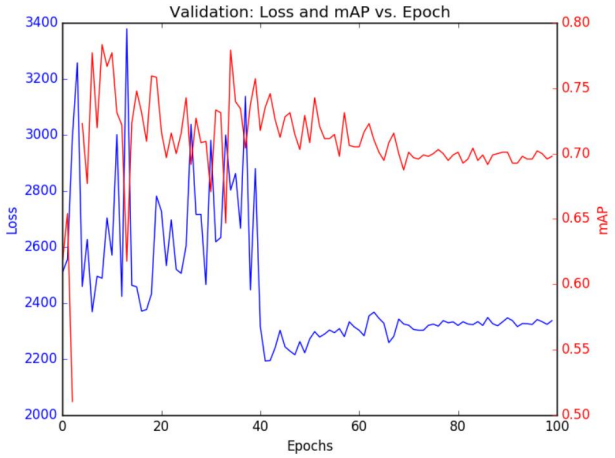

Figure 6. Dynamic Validation

7. Discussion

In this paper we implemented the YOLO v1 model in Keras/Tensorflow from scratch, and trained it to detect and classify clothing in images usinng the Street2Shop dataset. We used initial weights pre-trained on ImageNet and trained the last 6 layers of the network. We experimented with various types of regularizers and hyperparameter values to overcome instability in early training. We showed it learned to detect and classify clothing at a level far better than the sliding window baseline and comparable to Redmon et al.s performance on the PASCAL VOC dataset.

There are some fairly obvious next steps: upgrade the network architecture to YOLO v2. The batch-normalization, use of anchor boxes, and bounding box specific classification produce significant performance improvements in v2 provide significant improvements. We would also like to compare it against other real-time detection and classification architectures such as Fast(er)-RCNN and Mask-RCNN.

However, in our opinion, the biggest barrier to turning this model into something practical, something that people would actually use to identify clothing in images, is a better dataset. Our dataset had very crude class labels such as bag and dress. A dataset with thousands of labels specifying the style, color, and even brand of the clothing would allow us to train a model that could detect and classify images at a level granular enough to be truly useful to people.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.

[2] M. Hadi Kiapour, Xufeng Han, Svetlana Lazebnik, Alexander C. Berg, and Tamara L. Berg. 2015. Where to Buy It: Matching Street Clothing Photos in Online Shops. In Proc. ICCV

[3] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In CVPR, 2016.

[4] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition