Where'd You Get That? An Efficient Deep Learning Approach to Fashion Identification (Computer Vision)

Varun Tandon
Stanford University
varunt@stanford.edu

Ece Korkmaz
Stanford University
ekorkmaz@stanford.edu

Beri Kohen Behar
Stanford University
bkohen@stanford.edu

Abstract

Many fashion trends find their birthplace on TV and in movies. In fact, searching online, one can find hundreds of sites devoted to analyzing the fashion found in popular TV shows and movies. We present an automated approach to compiling this data, a deep learning framework for detecting clothing, and identifying the store it comes from. Furthermore, we developed our approach with an emphasis on low runtime inference, to allow our system to be run on video in realtime. Our system consists of a Match RCNN neural network based on the R101-FPN architecture for clothing localization, a CNN for rapid classification into clothing type, an embedding layer based on a CNN optimized on triplet loss and a final deep neural network for measuring embedding similarity.

1 Introduction

The problem we consider is the following: given an image of someone wearing some clothing, determine the exact shop listing for that clothing (assuming that the shop listing exists in a precompiled database). In particular, we set a target inference time of 33 milliseconds (computed from running our algorithm on every frame of a 30 FPS video). Our process is divided into two operations: 1) compilation of a shop item database and 2) inference on an image in a video. For our database compilation, we define the input as a shop image and the output as a vector embedding representing the image. We use a Match RCNN to perform item localization within the shop image, a softmax-based deep neural network for item categorization, and finally a CNN for embedding generation. For our video frame inference, we define the input as a frame of a video and the output as a list of shop items in our database which correspond to the clothes in the video frame. We use a Match RCNN to perform item localization within the video frame, a softmax-based deep neural network for item categorization, a CNN for embedding generation, and finally a deep neural network to perform a comparison of the video frame embedding to the embeddings in our database.

2 Related Work

Related work in this space involves clothing localization, clothing type classification, image embedding generation, and embedding comparison. Object localization is a very well studied field, and a variety of techniques exist including Viola-Jones object detection[10], YOLO[11], and RCNN[13]. RCNN in particular was promising, since it is discussed in the paper DeepFashion2[6], which presents an object localization technique specific to fashion images. Furthermore, a particular variant of the RCNN, the Faster R-CNN[13], presents an object localization algorithm with the intention of real time detection. Shifting focus to our clothing type classification algorithm, we want to implement a simple multi-class classifier. From the CS230 lectures, we know that a common approach here is a softmax classifier, and specific architectures for clothing classification exist in the form of classifiers

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

implemented to classify the famous Fashion MNIST dataset [14]. We decided to use one of these classifiers on Github[8] that achieved 91% accuracy on MNIST training set using 10 different classes. For image embedding generation, we looked to facial recognition techniques since they embeddings are a common technique used for low runtime facial recognition. Specifically, we referred to the FaceNet paper [9], which describes a CNN architecture and the triplet loss function which are used in tandem to generate accurate embeddings. We also referred to the Street2Shop paper [5] for its approaches for clothing identification, and Keras' implementation of the variational autoencoder[15]. Finally, for embedding comparison, we once again referred to Street2Shop and FaceNet, specifically considering cosine similarity and deep neural network architectures with embedding inputs.

3 Dataset and Features

For our paper, we considered a few key fashion-related datasets. An extensive search of the literature yielded Fashion MNIST, DeepFashion2, and Street2Shop. The Fashion MNIST database contains a mapping of fashion images to clothing type. DeepFashion2 contains a dataset of fashion images that have labeled bounding boxes for clothing, as well as the clothing type. Street2Shop contains a dataset of fashion images paired with shop images of the products shown in the fashion images, as well as bounding boxes for the fashion images shown. While the Fashion MNIST dataset seemed initially promising for creating a model for classifying images of clothing into clothing type, unfortunately, the images in the dataset do not mimic the fashion images captured in the real world, and the image sizes are much smaller than desired (28 pixels by 28 pixels). Furthermore, while DeepFashion2 was useful for object localization training since it lacked a mapping from the fashion image to a store image, it's usefulness was restricted to the localization task. For features, we decided to forego manual feature generation, and instead leverage CNNs on the raw pixel data.

4 Methods

4.1 Object Localization

As previously discussed, for clothing localization, we leveraged Region Based Convolutional Neural Networks (RCNN). The RCNN extracts region proposals prior to computing CNN features on the image. The input to this network is a fashion image. The network then identifies some clothing and outputs a bounding box for that clothing. To measure error, we use the standard metric of the average precision (AP) for varying intersection over union (IoU). For our implementation, we made use of Facebook's detectron2 package, which provides access to common architectures, including the R50-FPN, R101-FPN, X101-FPN Faster RCNNs[16].

4.2 Clothing Classification

For clothing classification, we used a deep CNN with a softmax output. For our loss function, we used categorical cross entropy loss (as is common with a softmax output), and for our optimization algorithm, we used the Adam optimizer. We also employed dropout regularization to reduce overfitting.

4.3 Embedding Generation

For embedding generation, we considered three main techniques. The first was using embeddings from the VGG16 and VGG19 models[12], pretrained on Imagenet. We generate these embeddings by propagating the image through the pretrained model and then storing the data of the layer prior to the output. The second approach we considered was via variational autoencoders. In this approach, our input and output layers have the same shape. We pass in the input image and train the model to have the output image match the input. However, we include a bottleneck within the model architecture, forcing the model to represent the input image by first encoding the image as some vector. This vector can then be used as our embedding for the input image. Finally, we attempted to train a deep CNN based on the same architecture as the FaceNet paper. Specifically, this CNN takes an image as input and outputs an embedding. We train this model by passing in a set of anchor images, positive examples, and negative examples, where the anchor images are the fashion images capture in "the wild", the positive examples are the matching shop image, and the negative examples are

non-matching shop images. Per the FaceNet paper, we used triplet loss to train these embeddings. Furthermore, in order to generate positive and negative examples, we used both random sampling for negative examples, as well as the approach described in the FaceNet paper. Specifically, we find shop images satisfying

$$argmin_{x_i^n} ||f(x_i^a) - f(x_i^n)||_2^2$$

where x_i^n represents a negative image, x_i^a represents an anchor image, and f represents the function mapping from the image to the embedding. In addition to the architecture proposed in the FaceNet paper, we attempted to tune our own deep CNN architecture.

4.4 Embedding Comparison

For embedding comparison, we consider two approaches: cosine similarity and deep neural networks. Cosine similarity is a common metric for comparing the similarity of two vectors, whereas the deep neural network can learn the values within the embedding that are significant for classification. Specifically, our deep neural network takes as input two embeddings in concatenated form and outputs via a sigmoid the probability that the two embeddings represent the same shop item.

5 Experiments/Results/Discussion

5.1 Bounding Boxes

We trained our RCNNs on the DeepFashion2 dataset. We trained for 5000 iterations with 1500 warmup iterations, four images per batch, a base learning rate of 0.001, and a gamma of 0.05. A table of our results is shown below.

RCNN Architecture	AP	Inference Time (s)
R50-FPN	35.1	0.042
R101-FPN	39.3	0.061
X101-FPN	40.7	0.093

Given our emphasis on runtime, we observe the R101-FPN provides the best tradeoff betweden AP and inference time.

5.2 Clothing Classification

Initially, we attempted to classify images into the 11 clothing categories that exist in our data set. We started by trying a network that was used to apply the same task to the Fashion-MNIST dataset on Github [7]. This CNN accepted grayscale input images of size 28 by 28. It consisted of three identical blocks, followed by a fully connected layer with ReLU activation and a final fully connected layer with softmax activation. Each building block consisted of a convolutional layer with ReLU activation, max pooling layer, and dropout. Categorical cross entropy loss and Adam optimization were used. Our first training of this CNN (without any adjustments to the network) resulted in a training error of 77% and a validation error of 75%.

From these results, it seemed like we had a high bias problem. Adding further blocks to the CNN did not lead to any improvements, and tuning hyperparameters such as batch size and dropout rate led to minor differences (not more than 1% increase in validation accuracy). After visualizing some images, we realized that the default 28*28 input image size might not be ideal for our dataset, especially considering that a lot of the images had to be squeezed disproportionately in one direction since many bounding boxes were rectangular.

Using a similar architecture (with an additional 4th convolution + pooling layer) with 64*64 input size did not lead to any significant improvements in the results (validation accuracy went up to 77%). We then decided to narrow down our problem by eliminating classes that had much fewer data points. Around 8% of the dataset was dropped in this process. We ended up with 6 classes: footwear, leggings, pants, tops, dresses, and skirts. Furthermore, after some manual inspection of the results of our previous trials, we decided to merge the leggings and pants classes, since the classifier often could not distinguish between the two (and some examples were even indistinguishable to the human

eye). These changes, along with hyperparameter tuning, increased the validation accuracy to 85%.

Furthermore, we also used a top-2 accuracy metric, which measured the percent of the predictions were correct if we used the top 2 results from the softmax output, and was 95% in this case (compared to 85% before we reduced the number of classes). This metric is useful in the scope of the whole project because predicting 2 classes still helps lower the inference time. Instead of comparing the embedding of an image to the whole database, we can instead compare it to the top 2 classes we predict. Considering that the task of matching a street image to a shop image is expected to have low accuracy, 95% is sufficient, especially considering that we would cut down the embedding comparison time by about 60% (2 classes instead of 5) if we had a balanced dataset. The final results on the test set were an 83% accuracy and 93% top-2 accuracy. Moreover, the inference time on a single frame was measured to be on average less than 0.02s, which is well within our runtime limitations.

Model Type	Train Acc.(%)	Val. Acc.(%)	Val. top-2 Ac(%)
Original Network with 28*28 input	77	75	85
Deeper Network with 64*64 input	78	77	87
64*64 input with fewer classes	90	85	95

5.3 Embedding Generation and Embedding Similarity

5.3.1 Trial 1: Face Recognition CNN

We tried a face recognition CNN [8] with two CONV2D layers with 'relu' activation followed by max pooling layers. We flattened the output and used 3 dense layers followed by dropout layers. Our final dense layer used a 'softmax' activation. We trained our model with tops and pants with a batch size of 128 and 150 epochs. This was a common architecture used in face recognition algorithms that we found online. We used tops and pants for our training to see how this embedding performs on a smaller subset of our data. Our input was a user image bounding box and output was an e-commerce product image. We had 1799 pairs in our training set and 300 each for our validation and test sets. We were planning to use the face recognition algorithm to get image embeddings by dropping the last softmax output layer. Thus, we trained our dataset with the user images and treated all of them as separate classes. We used categorical cross entropy and Adam optimizer. We reached to a 95.38% accuracy with this data.

We dropped the last layer and used this model to generate embeddings. We used a similar logic to the Coursera face recognition lab by finding every image's embedding and calculating its cosine similarity to the shop images in order to find pairings. Yet, we had 0.015% accuracy in our test set. Since this accuracy was very low, we decided that Face Recognition structure is not feasible with our dataset since our input and output images give very different embeddings.

Then, we decided to train our model with both user and shop images as the input and treated every pairing as one category. Thus, when we are training our CNN, it has 2 different images for one product and might give better embeddings with lower cosine distances for the correct pairings. When we expanded our training set, we got 35% accuracy. We decided to train longer with 500 epochs to reduce bias. Then, we got 60% accuracy and added batch normalization to our model and reached 75%. However, when we ran the embeddings while using cosine similarity we still got only one correct match. We decided that face recognition CNN's doesn't work and decided to pursue a different algorithm.

5.3.2 Trial 2: Using Pre-Trained VGG-16

Since our face recognition CNN failed, we decided to try a pre-trained model to get our embeddings. We used VGG16 convolutional network trained with ImageNet[12] and changed the input tensor to 128x128x3. We tested this network our test set and got 0.015% accuracy via both cosine similarity and the deep learning system.

5.3.3 Trial 3: Training VGG16 on Fashion Data

We attempted to train VGG16[12] on our fashion data as a classifier, then to remove the output layer and use this model for embeddings. Once again, our embeddings resulted in a 0.015% accuracy via both cosine similarity and the deep learning system.

5.3.4 Trial 4: FaceNet Architecture via Triplet Loss

We copied the FaceNet architecture and then trained this architecture on our images via the triplet loss function. The FaceNet architecture outputs a 4096 dimensional embedding which we then use for the triplet loss comparison as described. We settled on an iterative approach involving training for 15 epochs on the full batch, followed by a regeneration of the negative and positive examples. We repeated this iterative process 100 times for training. Through this process, we observed that the FaceNet embeddings were converging to 0, meaning that all of the values of every embedding were 0. This is a common problem when training embeddings, and we attempted to both lower the training rate and modulate the batch size; however, these did not have an impact on providing meaningful embeddings. We considered that this might be a problem with network architecture, leading us to our next approach.

5.3.5 Trial 5: Custom Neural Network Architecture via Triplet Loss

We attempted to create our own CNN for generating embeddings to be used for triplet loss optimization. We experimented with between 1 - 5 convolutional layers with filter quantities ranging from 4 to 128, and strides of between 1 and 3. After our convolutional layers, we performed a flatten, and then experimented with a deep neural network with between 1 - 5 layers, with between 8 - 4096 hidden units. Despite all these experiments, we either got embeddings that would have an accuracy of 0.015 on cosine similarity and the deep similarity metric, or we would encounter the afforementioned issue of embeddings converging to 0.

6 Conclusion/Future Work

In this project, we demonstrated that we are able to classify clothing by object localization followed by a deep CNN classifier which respectively achieved an AP of 39.3 and 83 percent accuracy. However, we couldn't achieve substantial accuracy with any of our user-shop image matching algorithms. All of our image embedding networks gave very different results for shop and user image pairs. We believe that one of our main constraints was the quality of our dataset. We decided to use Street2Shop instead of the DeepFashion2 dataset because DeepFashion2 doesn't specify which bounding box in the user image corresponds to the shop image. However, Street2Shop has poorly cropped product images. Some of the store images are very zoomed in and only displays a small texture of the product. Yet, the crops are not consistent so while user images display the full product, shop images can vary from a close-up to a small button on a top to a full picture. Thus, none of our embedding networks gave similar results for the user input and shop image output since it requires not only product matching but also texture matching which is a harder problem to solve.

The main existing study using the Street2Shop dataset achieves 15.6% accuracy on matching the user image of dresses with the shop images [5]. Their algorithm reaches to 33.5% accuracy when they use the top 10 guesses. They also state in their human experiments, humans chose the correct item out of 10 choices with 87% accuracy. This further supports that the available datasets in this domain are not sufficient to train this network. For future work, we observe three avenues for improvement. The first is to contact the authors of the Street2Shop paper to clarify the code they used to obtain their results and ensure reproducibility. The second is to determine the end to end inference time of our models in tandem. Due to the limitations of our model accuracy we were unable to determine an end to end runtime. Finally, in order to improve our embedding networks, the best next step is to create a new dataset that maps user inputs to appropriately cropped shop images which display the full product and have sufficient lighting. This work would involve scraping data from sites that describe fashion trends in movie and television, and then generating a set of images from screenshots of those movies.

7 Contributions

Every member of our team contributed to this project equally. Varun implemented our object localization network and the FaceNet architecture; Ece and Beri implemented the clothing classifier, facial recognition CNN's, and pre-trained networks for embedding generation. After we got our initial results, we improved our models all together by trying different suggestions to increase our accuracy.

References

[1] J. Huang, R. S. Feris, Q. Chen and S. Yan, "Cross-domain image retrieval with a dual attribute-aware ranking network", IEEE International Conference on Computer Vision, 2015.

[2] Q. Chen, J. Huang, R. Feris, L. M. Brown, J. Dong and S. Yan, "Deep domain adaptation for describing people based on fine-grained clothing attributes", IEEE Conference on ComputerVision and Pattern Recognition, pp. 5315-5324, 2015

[3] Dong, Qi, Shaogang Gong, and Xiatian Zhu. "Multi-task curriculum transfer deep learning of clothing attributes." 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2017.

[4] Y. Ge, R. Zhang, L. Wu, et al. A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images. In CVPR, 2019.

[5] M. H. Kiapour, X. Han, S. Lazebnik, A. C. Berg and T. L. Berg, "Where to Buy It: Matching Street Clothing Photos in Online Shops," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 3343-3351, doi: 10.1109/ICCV.2015.382.

[6] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.

[7] https://github.com/sandeep201451066/Fashion-Wear-Classification-

[8] https://github.com/Fatemeh-MA/Face-recognition-using-CNN

[9] F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 815-823, doi: 10.1109/CVPR.2015.7298682.

[10] Viola, P., Jones, M.J. Robust Real-Time Face Detection. International Journal of Computer Vision 57, 137–154 (2004). https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[11] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[12] Simonyan, K. Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

[13] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[14] Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.

[15] https://keras.io/examples/generative/vae/

[16] https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md