Transfer Learning for Sketch Recognition using PyTorch

Alyssa Chen

August 16, 2019

Abstract

In this project, I will discuss the state of sketch recognition today. Much of my work is motivated by two previous works done in the field, which I mentioned in my midterm review and which are still relevant: Sketch-a-Net [2] and On the Performance of GoogLeNet and AlexNet Applied to Sketches [1]. I attempt to approach the answer using various transfer learning methods in PyTorch using both hand-drawn and real image datasets.

1 Introduction

Sketch-a-Net [2] challenged photo-oriented neural networks at the time as one of the more successful networks able to recognize hand-drawn images. The researchers involved listed architectural designs as well as algorithmic changes from normal successful photo-oriented neural networks as the cause to a 75% recognition rate on the TU-Berlin Sketch Dataset, compared to the next best (Human, 73%, and FV-SP, 68.9%). In 2016, Ballester and Araujo released a small experiment on testing the performance of common neural nets on hand-drawn images, and illustrated their complete lack of ability to recognize hand-drawn images. Unsurprisingly, many images were incorrectly identified as paper clips or safety pins, presumably for the fact that both resemble lines, and most sketches by humans are line-drawings that provide little shading or detail. Interestingly, they also provided information on which images were most recognized, those being the most detailed or shaded.

Both papers lead to the question of why photo-oriented neural nets are bad at recognizing hand-drawn images, and what could possibly improve them. The question of how humans process hand-drawn images relative to real images (i.e. representation vs. real life) is also a more theoretical end goal to much of this research. Humans do not seem to have to be trained to recognize representations, and if they do, they do not seem to require retraining for a new object: Why is that? I discussed different theories and papers in my earlier midterm presentation, but will be focusing more on the computational aspect here.

(a) Hand-drawn ant (b) Hand-drawn bee (c) Photo of ant (d) Photo of bee

Figure 1: Examples of images from TU Berlin and ImageNet.

In this vein of thought, the main purpose of my project is to attempt to use transfer learning on a pretrained network and learn in what ways can we modify it to apply to sketches to hopefully gain an improvement without having to change the entire structure or algorithm.

2 Theory and Hypothesis

Understanding of convolutional neural networks, first introduced by LeCun, is not fully understood. Indeed, it is difficult to tell which aspects of a neural network must be changed in order to properly complete a transfer learning task from photo-oriented neural networks to hand-drawn images. To this end, I posit the question of how to carry out a transfer learning task. I hypothesized, however, that it would be difficult to modify a photo-oriented neural network without major changes to architecture and potentially damaging its ability to recognize real images.

3 Procedure and Results

Overall. For the computational portion, I will be focusing on classifying ants and bees using a pretrained VGG16 model. In each training and evaluation sequence, I use 25 epochs. In the paper, I will discuss the methods used; however, please see the code for specifics. Note also that I cited code used in the code itself (first line).

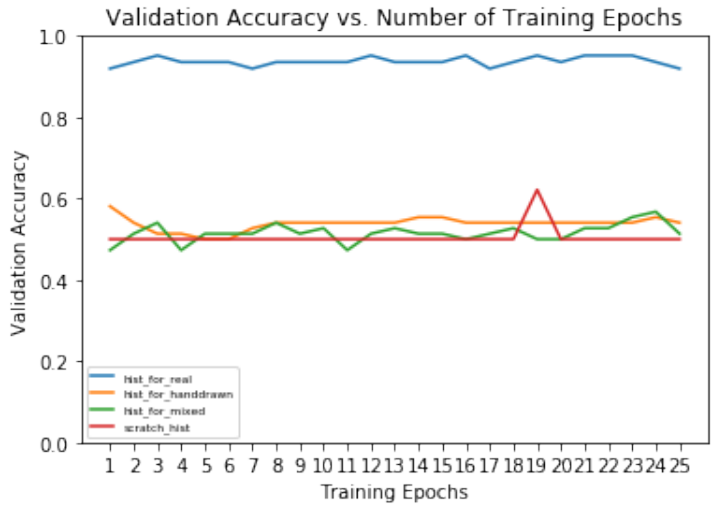
Datasets. We will be using three main datasets, with one additional dataset which will be specified in a later section.

TUBerlin. Train: 43 ants, 43 bees Evaluation (Val): 31 ants, 31 bees.
Mixed, i.e. training set from Hymenoptera dataset, but Evaluation set from TU Berlin. Train: 123 ants, 123 bees. Val: 31 ants, 31 bees.
Hymenoptera. (Photos from ImageNet.) Train: 123 ants, 123 bees. Val: 70 ants, 70 bees.

See Fig. 1 for examples from both datasets.

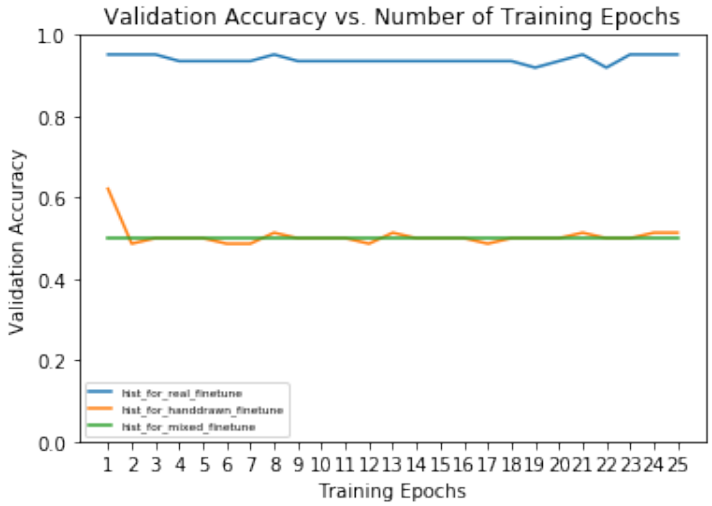
First Modification of VGG16. In order to classify 2 classes as opposed to thousands, we change the last classifier layer to have 2 outputs. See code for details on model architecture.

Fixed Feature Extraction. The first optimization for transfer learning was to fix/freeze most of the network so that only one layer of parameters needed to be optimized. We specified this in the `feature_extract` boolean (which was set to False for this part of the process) as well as in the `set_parameter_requires_grad` function, which specifies which parameters require computing of the gradient.



The red line, scratchhist, is simply the completely untrained model now trained on the real set and tested on the hand-drawn set. We can see that histforreal, which is the validation accuracy using the pretrained model with fixed feature extraction trained on real images and tested on real images, is the most successful in validation accuracy. This is unsurprising since VGG16 is optimized for and trained on photos. However, both models tested on handdrawn images (mixed and handdrawn) remain close to random. We can note, however, that handdrawn does hover above .5 and indeed achieves a best validation accuracy of .58 in the run (see code for more specifics regarding epochs and numbers for each model).

Finetuning the whole model. Next, instead of freezing the parameters of the layers of the pretrained VGG16 model, we set out to optimize the entire model in training. After setting up the proper Booleans (i.e. setting `feature_extract` to True) and functions (see code for more details), we train the same models on the same datasets again, to see if there are any improvements.

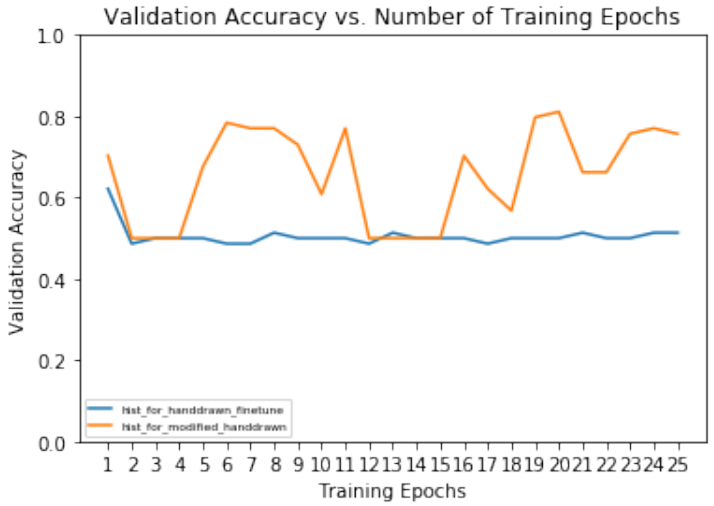


Surprisingly, the model on the mixed dataset ended up testing at exactly the expected random validation accuracy. Furthermore, the model on the hand-drawn dataset did not remain above .5 as it did previously. Finally, again, we can note the unsurprising good performance of the model tested on real images. The reason for this is unclear, as it would be most intuivitive to assume that finetuning the whole model would improve validation accuracy, especially with the model using the hand-drawn test set and hand-drawn val set.

Modifying the neural network a la Sketch-a-Net. Sketch-a-Net [2] provides a completely different neural network architecture and algorithm for sketch recognition. We will incorporate some architectural changes into our own model based on the research done in that paper. However, in the spirit of not modifying the model too much, as well as my own failure to properly manipulate MatConvNet (see appendix), we will not implement all optimizations listed.

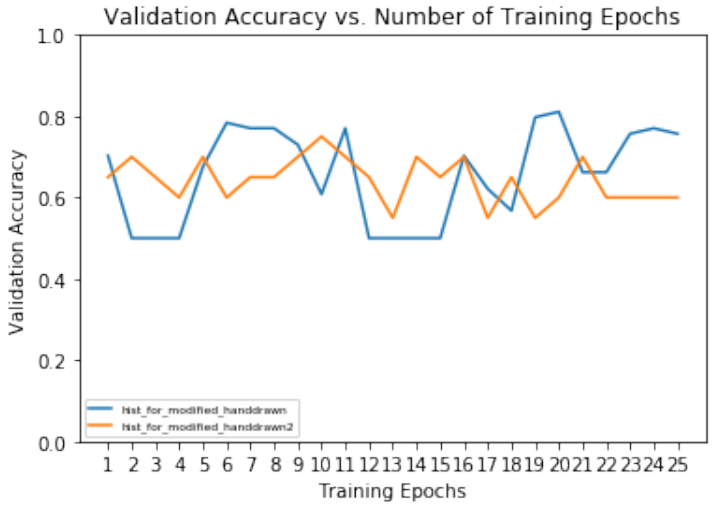
The aspects implemented are inspired by Section 3.1: Unique Aspects in our Sketch-a-Net Architecture. Namely, I modified the first layer to have a larger filter and increased the pooling size to 3 by 3 with stride 2. See code for printout of architecture.

At this point, I focus on optimizing the model tested on hand-drawn images and trained on hand-drawn images. Thus, we can graph a comparison of the performance of the modified model (using Sketch-a-Net optimizations) and the unmodified model, both of which are tested and trained on the hand-drawn set.



Interestingly enough, modifying the neural network after Sketch-a-Net led to a a higher variance in validation accuracy, and the model attains a best validation accuracy of 0.81. The reasoning for both optimizations [2] are that (1) sketches lack texture information, and thus a larger filter would be more appropriate, and (2) a larger pooling size brings more spatial invariance. However, for the second optimization, the authors also added that a reason was that the optimization brought a one-percent improvement without any difficulty in computation. Indeed, the uncertainty of why these optimizations work exactly the way they do is an interesting aspect of neural network research we will not fully address here. However, we can note that it is difficult to understand why our modifications led to the shown improvements in accuracy.

Modifying the hand-drawn dataset. One last optimization that I attempted was changing the proportion of no. of test images to no. of val images in the hand-drawn dataset. Instead of having 43 train images and 31 val images, the new dataset has 64 train images and 10 val images (for bees and ants alike). This is done in hopes of increasing accuracy with a larger training set (the val set was decreased in size only because of the limited number of ant/bee sketches in TU Berlin).



Again, it is difficult to conclude anything from the graph shown. However, we might be able to say that the network is more consistent in validation accuracy with the new dataset, presumably due to a larger training set.

4 Conclusion and Discussion

As we have found, although finetuning the whole model did not seem to do much, optimizing according to some of the Sketch-a-Net optimizations did seem to help the VGG16 model, pretrained on real images, recognize hand-drawn images. However, the datasets used to train were admittedly small (approx. 100 for the hand-drawn images, which had to be split into train and val sets). Furthermore, I limited the number of epochs to 25, and many of the models seemed to reach a set validation accuracy immediately, but it is hard to say where the later models (i.e. the modified models) might go if given more epochs to train. Thus, there is much uncertainty regarding the results of this experiment.

However, it is interesting, again, to note that providing seven simple modifications to the architecture in modifying using Sketch-a-Net did cause a drastic change in recognition without having to change the entire architecture or even the algorithms (i.e. loss functions or back-propagation methods) too much. This result hints at the possibility of not having to create an entirely different neural network to recognize sketches, but rather being able to build sketch regonition on top of photo recognition, as humans seem to do.

Further work can be done in further modifying the network in addition to testing the modified network on real images as well, to see if it retains its "ability" to recognize photos. Additionally, more time and data is needed to reach more conclusive results.

5 Datasets

(see links to both datasets below)

TU Berlin Dataset for hand-drawn images. Eitz, Mathias and Hays, James and Alexa, Marc. 2012. How Do Humans Sketch Objects? ACM Trans. Graph. (Proc. SIGGRAPH). Volume 31, Issue 3. Pgs. 44:1–44:10.

ImageNet/PyTorch Tutorials for real images. See https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html.

6 Links

Link to PyTorch Code in Google Colaboratory (along with code being run): https://colab.research.google.com/drive/1Dcs4M-iUcMzv6nwIuliYMmtUi8_afasW.

Link to datasets: https://drive.google.com/open?id=1kyh2-gOIfy4CvGFTbk8-KgH998etG9hq.

References

[1] Ballester, Pedro and Araujo, Ricardo. 2016. On the performance of GoogLeNet and AlexNet applied to sketches. Thirtieth AAAI Conference on Artificial Intelligence.

[2] Yu, Qian and Yongxin Yang, Yi-Zhe Song, Tao Xiang, Timothy Hospedales. 30 Jan 2015. Sketch-a-Net that Beats Humans . *arXiv:1501.07873*.