Fine-tuning for Image Style Recognition

Yinghui Xia
Stanford University
450 Serra Mall, Stanford, CA 94305
yinghui@stanford.edu

Abstract

The style of an image contains some features and information about the image that facilitates object recognition and classification. A convolutional neural network model is described to predicting style of images, and perform an evaluation of the overall style classification accuracy. My work is based on the labeled Flickr dataset - photos extracted from 100 Million Flickr photographs annotated with 20 style labels. The result will be compared with baseline Fine-tuning CaffeNet for Style Recognition on "Flickr Style" Data and previous image style recognition work.

1. Introduction

There is a lot of research related genre recognition. My focus is the recognition of photographs style recognition with a test case on photographs style recognition. Since representations are key to effective perception, style classification is useful for aesthetic purpose, photoshopped photos, and telling the story of an image. A good style classifier takes advantages of the intrinsic elements of the image formation process, for example depth, color and paint.

Existing implementations of image style recognition use deep convolutional neural network as well as other image featuring techniques. This paper focuses on improving the convolutional neural network layers with batch normalization, different learning rate, and activation functions.

2. Background

Ioffe el at. [4] brought up the idea of batch normalization to address the internal covariate shift issue in training deep neural network. Their work makes normalization a part of the model architecture and performs the normalization for each training batch. Their work shows that batch normalization accelerate the convergence of deep neural network significantly.

Karayev et al. [3] tests several image features, i.e. color histogram, GIST, graph-based visual saliency on these styles and finds that deep convolutional neural network (CNN) features perform best for the image style classification task. This paper tries to examine whether the batch normalization is generic for style classification.

2 Methods

2.1 Batch Normalization [4]

Normalize x in each layer by

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Perform Batch Transformation following

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

See the pseudo-code of Training a Batch-Normalized Network in Appendix.

Back-propagation through a BN layer is unaffected by the scale of its parameters. The intuition of BN is to prevent model explosion as the gradient increased with large learning rate.

2.2 Datasets

On Flickr, there are many styles, including HDR, Macro, Vintage, etc. To line up with previous research [2], the same dataset of 80K images is used.

2.3 Baseline

80,000 images is a small dataset to train on, so starting with the parameters learned on the 1,000,000 ImageNet images will converge much faster.

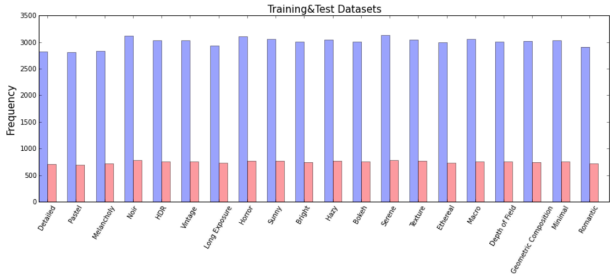
The baseline for a 20-class model is 0.05. The Flickr images of the Style dataset are similar to the ImageNet

dataset, on which the bvlc_reference_caffenet was trained. Since that model works well for object category classification, it is used as the pretrained model. Caffe model trained on 2,000 images gives an accuracy of 23.5% over 1,000 iterations and 80,000 images with 39.16% validation accuracy over 100,000 iterations.

2.4 Assembly the data

For each Flickr group of a certain style, we extract around 4,000 photographs. A total number of 79,990 (4,000*20 classes) photographs are extracted. 5,028 (6.25%) images are deprecated or removed by user. The remaining images are separated into training set (60013, 80%) and test set (14949, 20%).

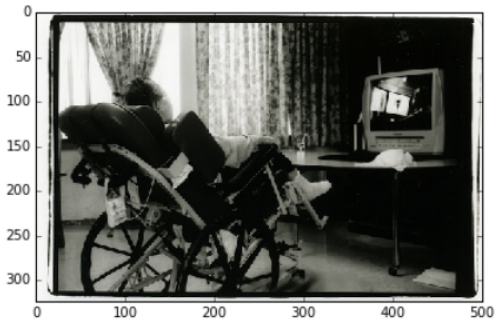
See the histogram of each image style.



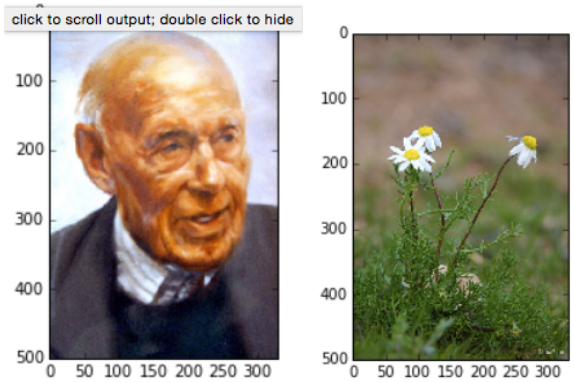
2.5 Data preprocessing and Augmentation

Certain Flickr styles strongly depend on color in formation (this is true for high-art painting since different paintings in the history display very different choices of color).

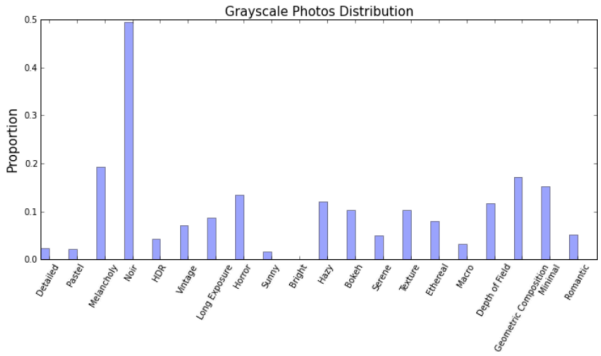
One simplest observation is that grayscale photos do not show up in Bright style, and there are very few in Detailed, Pastel, Sunny, and Macro in the training set. The number of grayscale photos that belong to Noir is substantially high (around 50%). The following photo is an example of Noir.



Examples of Bright:

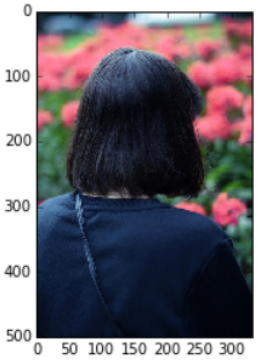


Therefore, the posterior Bayesian distribution of grayscale photos is assigned to each style when classifying test files.



The experiments before and after data preprocessing show a boost of performance after data preprocessing (the implementation of evaluating grayscale photos and random cropping&flip). See Figure 3.4.

Some figures are hard to classify because they have multiple labels. Here is an example of Romantic. It is reasonable to claim this is a Bright style.



2.6 Initial Layers

This structure is used in model a in section 2.8.2. As for convolutional neural network, DeCAF6, DeCAF7 models from Caffe are used. Since this is a small training set,

fine-tuning Linear Classifier on top layer of DeCAF7 is more promising.

Because we are predicting 20 classes, the top layer fc8_flickr begins training with random weights and boosted blobs_lrs. The idea is to have the rest of the model change slowly with new data, but let the fully-connected layer learn fast.

Chart 2.1 Initial Model

Layer	**Dimension**
Training Data	256*256*3
Training Data (Data Aug)	227*227*3
1. Conv11-96	55*55*96
Max Pooling1	27*27*96
LRN	27*27*96
2. Conv5-256	27*27*256
Max Pooling2	13*13*256
LRN	13*13*256
3. Conv3-384	13*13*384
4. Conv3-384	13*13*384
5. Conv3-256	13*13*256
Max Pooling5	6*6*256
6. FC6	1*1*4096
Dropout	1*1*4096
7. FC7	1*1*4096
Dropout	1*1*4096
8. FC8-Flickr	1*1*20
Softmax	

2.7 The Performance of Batch Normalization Networks.

Three Batch-Normalization(BN) layers are added to the model. The following layers are adjusted.

a. Two dropout layers removed
b. Local Response Normalization(LRN) layers replaced by BN layers.
c. Activation function ReLU replaced by others.

When trained with BN, the step size as well as the learning rate in the solver are set to a lower value than if we do not use it since training speedup from higher learning rates with comparable performance. Batch Normalization functions similar to Dropout. Removing Dropout speeds up training, without causing much overfitting. BN is a good substitute of Local Response Normalization. LRN is removed for the same reason as dropout to speed up the training. The following parameters are used in model b and c in section 2.8.2.

Chart 2.2 Final Model

Layer	**Dimension**
Training Data	256*256*3
Training Data (Data Aug)	227*227*3
1. Conv11-96	55*55*96
Max Pooling1	27*27*96
Batch Normalization	27*27*96
2. Conv5-256	27*27*256
Batch Normalization	27*27*256
Max Pooling2	13*13*256
3. Conv3-384	13*13*384
4. Conv3-384	13*13*384
5. Conv3-256	13*13*256
Batch Normalization	13*13*256
Max Pooling5	6*6*256
6. FC6	1*1*4096
7. FC7	1*1*4096
8. FC8-Flickr	1*1*20
Softmax	

2.8 Transfer Learning and Fine-tuning

2.8.1 Experiments

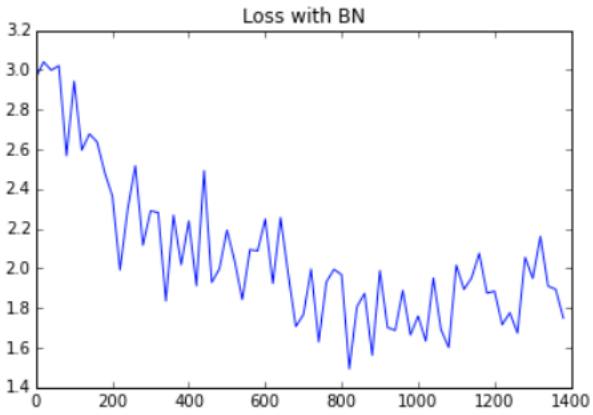
At the beginning of the training, the learning rates are fixed to 1e-3 for all conv layers. We train 1k mini-batches using the learning rate 1e-3, and then using 1e-4 for the next 4,000 iterations, and the rest (if any) of the batches with 1e-5.

2.8.2 Model Solver

a. Pretrained model: Imagenet
   Data augmentation: Yes
   BN: No
   Activation functions: ReLU
   Learning rate: 0.001
   Stepsize: 5000
   Weight decay: 0.0005

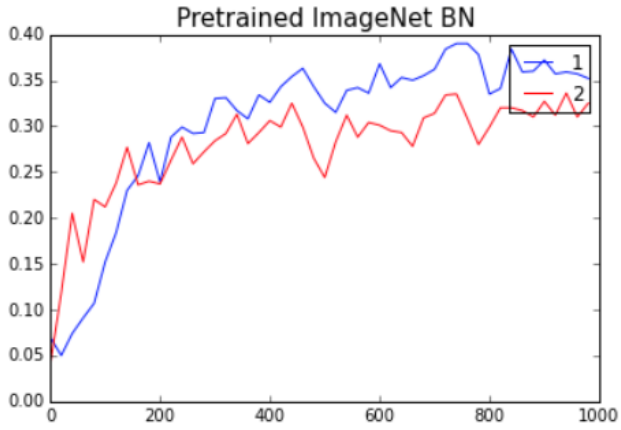
b. Pretrained model: Imagenet
   Data augmentation: Yes
   BN: Yes
   Activation functions: Leaky-ReLU (0.01)
   Learning rate: 0.001
   Stepsize: 5000
   Weight decay: 0.0005

Figure 3.1



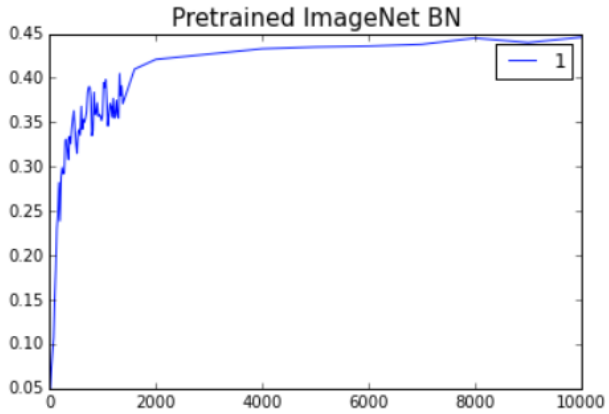
After changing learning rate at 1,400, the loss continues to converge after 1,400 iterations and reduced to around 0.8 after 10,000 iterations. It should reduce more if the maximum iteration is increased to 100,000 iterations.

Figure 3.2 Model a and b.
blue-with BN, red-without BN



Set the learning rate and step size lower because the pretrained model is closer to the ground truth. After 1,500 iterations, the accuracy is taken at 2,000, 3,000, 4,000, 5,000, 6,000, 7,000, 8,000, 9,000, 10,000 iterations.

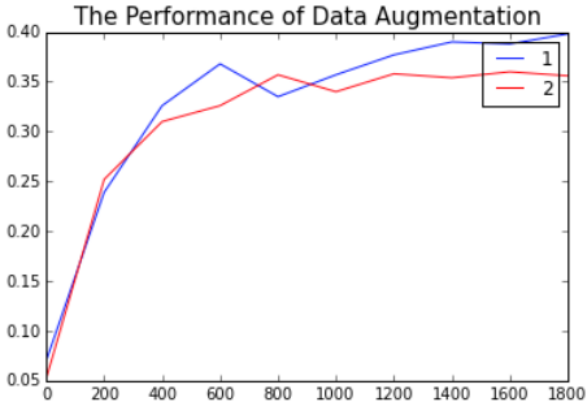
Figure 3.3



c. Pretrained model: Fine-tuned Flickr
Data augmentation: No
BN: Yes
Activation functions: Leaky-ReLU (0.01)
Learning rate: 0.0005
Stepsize: 1000
Weight decay: 0.0005
Set the learning rate and step size lower because the pretrained model is closer to the ground truth.

Compare model b and c. We can see that the data augmentation slightly boost the performance.

Figure 3.4 blue-with Data Augmentation,
red-without Data Augmentation



3 Result

3.1 Finished work

Transfer learning the new network from the ImageNet pretraining and the pre-trained Flickr model requires much less data than training from scratch. It allows the loss to go down faster than when we do not start with a pre-trained model.

Batch Normalization indeed accelerates the decreasing of the loss. However, the loss becomes bumpy and less stable. There is also possibility that it will not achieve the same accuracy as the model without Batch Normalization. In my experiments, this issue does not come up. Batch normalization with pretrained ImageNet achieves 39.16% accuracy within 10,000 iterations compared to the 100,000 iterations in the baseline model.

Batch Normalization preserves the representation ability of the network. The resulting networks are more tolerant to increased learning rates, and classification capability without Dropout for regularization. It works well with image style classification task. Some validation scores for Flickr data are shown in section 3.2.

3.2 Test Accuracy on Flickr data

Selected validation results. Three styles with the highest test accuracy.

Style		precision
Noir	False	0.53
	True	0.59
Macro	False	0.48
	True	0.47
Long Exposure	False	0.43
	True	0.47

4. References

[1] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, arXiv:1408.5093, 2014, Caffe: Convolutional Architecture for Fast Feature Embedding.

[2] Yangqing Jia, Fine-tuning CaffeNet for Style Recognition on "Flickr Style" Data.

[3] Karayev, S., Trentacoste, M., Han, H., et al. 2013, arXiv:1311.3715, Recognizing Image Style.

[4] Ioffe, Szegedy, 2015, arXiv:1502.03167, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[5] Batch normalization layer with test and examples, https://github.com/BVLC/caffe/pull/1965

5. Appendix

5.1 Detailed labels

Flickr labels	**Class labels**
Bokeh	11
Bright	9
Depth of Field	16
Detailed	0
Ethereal	14
Geometric Composition	17
Hazy	10
HDR	4
Horror	7
Long Exposure	6
Macro	15
Melancholy	3
Minimal	18
Noir	3
Romantic	19
Serene	12
Pastel	2
Sunny	8
Texture	13
Vintage	5

5.2 Pseudo Code for Training a Batch-Normalized Network (Actual code implemented in C++ in Caffe) [4]

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network

2: **for** $k = 1 \ldots K$ **do**

3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)

4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters

8: **for** $k = 1 \ldots K$ **do**

9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$