Bug Brain: Robust Insect Image Classification via Convolutional Neural Network

Geoffrey Shimotsu
Computer Science
UTRGV 2019
Edinburg, TX, United States
geoffrey.shimotsu01@utrgv.edu

***Abstract*—This project presents the classification accuracy results from training the ResNet18 machine learning algorithm on several image data sets, subsequently running sets of classification tasks; in this case insect images comprised the training and testing data. To determine robustness to poor input quality (photographic conditions), I applied image noise through gamma correction and repeated experiments. To mitigate decreased performance, I used an bootstrap aggregation ensembled model approach.**

***Index Terms*—machine learning, learning systems, artificial intelligence, supervised learning, convolutional neural network, machine learning algorithms, image classification, image processing**

I. Introduction

This project investigates the robustness of image classification to light changes. Entomological taxonomic classification proves vital in a number of academic disciplines and industries, especially agriculture [1]. This is an interesting supervised multiclass classification problem approachable by employing machine learning algorithms. The feasibility of generalized classification beyond taxonomic order is questionable, a challenge even for human experts; but for instance, automatic monitoring of bug traps seems plausible where high specificity is not required [2] However, any algorithm would need to address images gathered under varying lighting conditions to account for weather, diurnal cycle, mechanical issues, etc.

II. Methodology

I used Python and the Pytorch to complete experiments due to convenient abstraction, numerous useful libraries, and active online communities for machine learning [3].

A. Dataset

I composed the raw data set by scraping images posted to Bugguide.net and InsectImages.org using Selenium and Google image search [4]. Images gathered included 7 classes of insects: ants, bees, beetles, butterflies, millipedes, scorpions, and spiders. Approximately 100 images for each class split into training and testing sets, 80% allocated for training and 20%for testing.

B. Machine Learning Model

I used ResNet18 to generate classification predictions [5]. Fig. 1 illustrates the architecture.

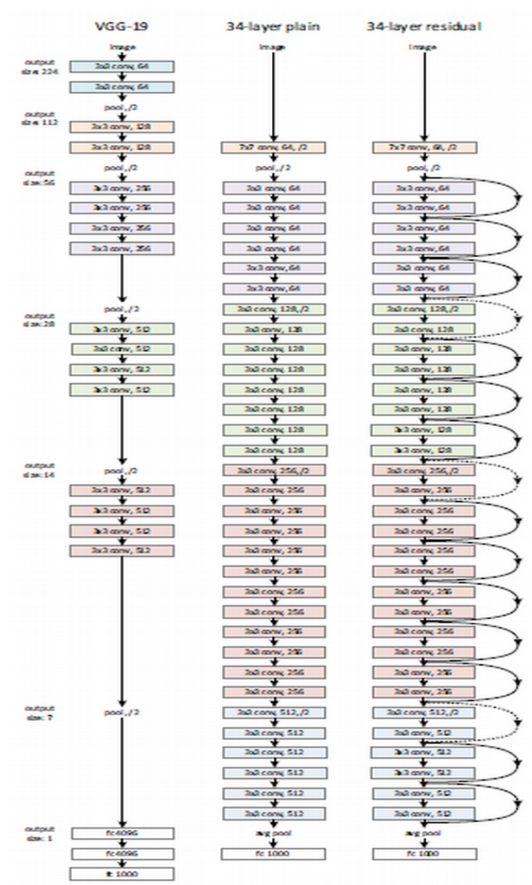

Fig. 1. ResNet34 architecture

III. Experiments

All experiments ran on the following system configuration: Ubuntu 16.04 OS, Intel Core i7-2600 CPU @ 3.40GHz, 16GiB DDR3 RAM @ 1333MHz, NVIDIA Titan Xp GPU w/12GiB GDDR5X. I used Python 3.6, Pytorch 0.4.1, and Torchvision 0.2.1.

A. Model Initialization

The ResNet18 model initialized with `pretrained=True` for easier generalization from the small dataset [6].

B. Feature Extraction/Fine Tune

Later in the ensembled model, the last fully connected (FC) layer is removed to use the model as a fixed feature extractor [7] and `requires_grad=False` for all model parameters to prevent finetuning i.e. gradient calculations during backpropagation [8].

C. Loss Function

Cross entropy loss commonly serves as the loss function for classification problems. Fig. 2 shows documentation details.

CLASS `torch.nn.CrossEntropyLoss`(*weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean'*) [SOURCE]

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain scores for each class.

input has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 2$ for the *K*-dimensional case (described later).

This criterion expects a class index (0 to *C-1*) as the *target* for each value of a 1D tensor of size *minibatch*

The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

or in the case of the *weight* argument being specified:

$$\text{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

The losses are averaged across observations for each minibatch.

Can also be used for higher dimension inputs, such as 2D images, by providing an input of size $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 2$, where K is the number of dimensions, and a target of appropriate shape (see below).

Fig. 2. Implementation details of the loss function [9]

D. Optimizer

I used the common stochastic gradient descent function to optimize the model [10]. Fig. 3 shows documentation details.

CLASS `torch.optim.SGD`(*params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False*) [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

Nesterov momentum is based on the formula from On the importance of initialization and momentum in deep learning.

Parameters:
- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate
- **momentum** (*float, optional*) – momentum factor (default: 0)
- **weight_decay** (*float, optional*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float, optional*) – dampening for momentum (default: 0)
- **nesterov** (*bool, optional*) – enables Nesterov momentum (default: False)

Fig. 3. Implementation details of the SGD optimizer [11]

E. Learning Rate

The step scheduler decays the learning rate according to user defined frequency per epoch. Fig. 4 shows documentation details.

CLASS `torch.optim.lr_scheduler.StepLR`(*optimizer, step_size, gamma=0.1, last_epoch=-1*) [SOURCE]

Sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size epochs. When last_epoch=-1, sets initial lr as lr.

Parameters:
- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **step_size** (*int*) – Period of learning rate decay.
- **gamma** (*float*) – Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** (*int*) – The index of last epoch. Default: -1.

Fig. 4. Implementation details of the learning rate scheduler [12]

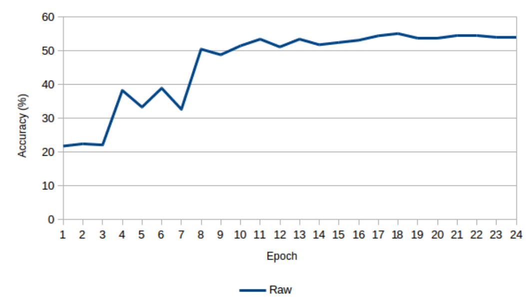

Fig. 5. Testing accuracy for ResNet18 trained and tested on raw images

F. Train/Test on Raw Images

Fig. 5 shows the results of testing the ResNet18 model trained on unaltered images.

G. Image Noise

I used gamma correction to change color saturation simulating under/over exposure of the image, and created 2 additional sets: oversaturated Γ = .25, and undersaturated Γ = 400 [13]. Fig. 6 compares an original image with gamma corrected.

Fig. 6. Gamma correction, Γ = 200

H. Train/Test on Undersaturated Images

Fig. 7 shows results of testing a raw-image-trained ResNet18 and undersaturated-image-trained ResNet18 on undersaturated images.

I. Train/Test on Oversaturated Images

Fig. 8 shows results of testing a raw-image-trained ResNet18 and oversaturated-image-trained ResNet18 on oversaturated images.

J. Ensembled Architecture

Previous research on ant classification yielded promising results with ensembled model architecture [14]. Here, I employ a bagging approach with 3 finetuned ResNets as fixed feature extractors, trained on raw, undersaturated, and oversaturated images respectively. For simplicity, the ensembled model uses the same model parameters as the single ResNet18 models. Fig. 9 roughly illustrates the architecture.

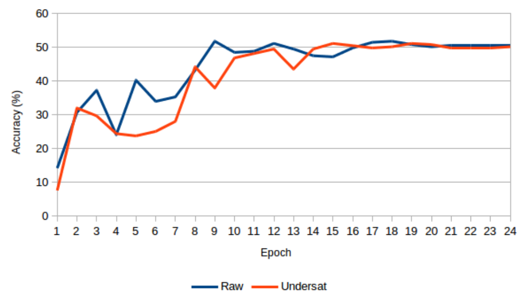

Fig. 7. Testing accuracy for ResNet18 trained on raw images, ResNet18 trained on undersaturated images, tested on undersaturated images

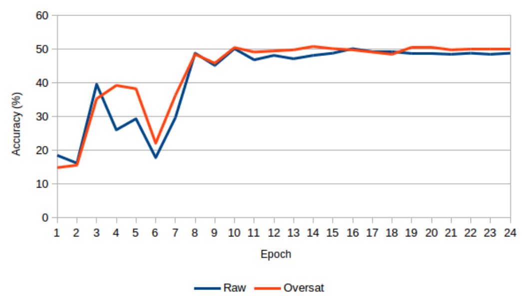

Fig. 8. Testing accuracy for ResNet18 trained on raw images, ResNet18 trained on oversaturated images, tested on oversaturated images

K. Train/Test on Mixed Images

I created one more subset of images from the 3 other subsets, making sure not to use gamma corrected versions of the same image. Fig. 10 shows results for the trained single models and ensembled model tested on mixed images.

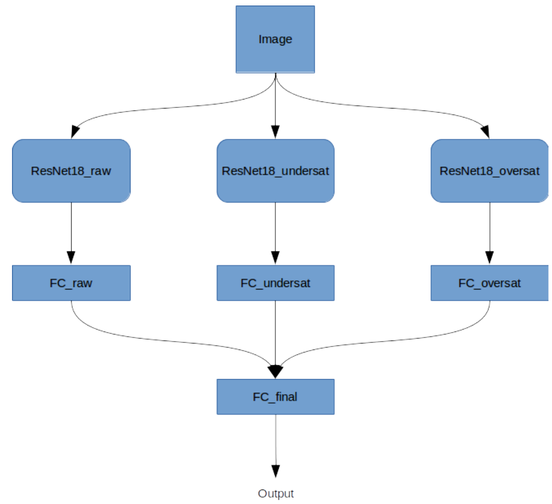

Fig. 9. Architecture of ensembled model

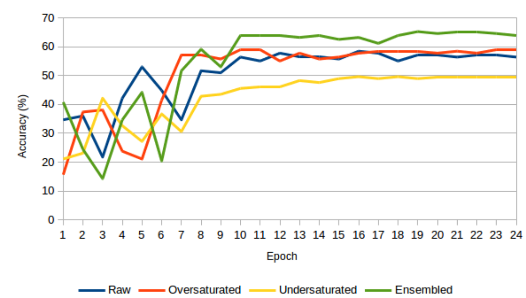

Fig. 10. Testing accuracy for single trained ResNet18 models and ensembled ResNet18 model tested on a mixed data set

IV. Discussion

While the ensembled model clearly outperforms the trained single models, the results are far below state of the art classification accuracy. This suggests several avenues of improvement: individual model architecture, ensemble strategy (boosting), hyperparameter tuning. The aforementioned tweaks mitigate performance drops in similar scenarios. Expanding the dataset with high resolution, preprocessed images might also prove effective [15].

Acknowledgment

Thanks to Dr. Dong-Chul Kim and Dr. Liyu Zhang for assistance starting, and to Dr. Hongkai Yu for advisement and support throughout.

References

[1] X. Cheng, Y. Zhang, Y. Chen, Y. Wu, and Y. Yue, “Pest identification via deep residual learning in complex background,” *Computers and Electronics in Agriculture*, vol. 141, pp. 351–356, 2017.
[2] S. N. A. Hassan, N. Rahman, and Z. Zaw, “Vision based entomology: a survey,” *International Journal of Computer science and engineering Survey (IJCSES)*, vol. 5, no. 1, 2014.
[3] A. Karpathy. Cs231n convolutional neural networks for visual recognition. [Online]. Available: http://cs231n.github.io/
[4] aaronsherwood. scrapeimages.py. [Online]. Available: https://gist.github.com/genekogan/ebd77196e4bf0705db51f86431099e57\#gistcomment-2007870
[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
[6] S. Chilamkurthy. (2018) Transfer learning tutorial. [Online]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html\#sphx-glr-beginner-transfer-learning-tutorial-py
[7] N. Inkawhich. Finetuning torchvision models. [Online]. Available: https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html\#alexnet
[8] V. Subramanian, *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch.* Packt Publishing, 2018. [Online]. Available: https://books.google.com/books?id=DOlODwAAQBAJ
[9] pytorch. torch.nn. [Online]. Available: https://pytorch.org/docs/stable/nn.html?highlight=cross_entropy\#torch.nn.functional.cross_entropy
[10] L. Bottou, *Stochastic Gradient Descent Tricks*, ser. Lecture Notes in Computer Science (LNCS). Springer, January 2012. [Online]. Available: https://www.microsoft.com/en-us/research/publication/stochastic-gradient-tricks/
[11] pytorch. torch.optim. [Online]. Available: https://pytorch.org/docs/stable/optim.html
[12] ——. torch.nn. [Online]. Available: https://pytorch.org/docs/stable/optim.html?highlight=steplr\#torch.optim.lr_scheduler.StepLR
[13] R. Szeliski. Changing the contrast and brightness of an image! [Online]. Available: https://docs.opencv.org/trunk/d3/dc1/tutorial_basic_linear_transform.html
[14] A. C. R. Marques, M. M. Raimundo, E. M. B. Cavalheiro, L. F. Salles, C. Lyra, and F. J. Von Zuben, “Ant genera identification using an

ensemble of convolutional neural networks," *PloS one*, vol. 13, no. 1, p. e0192011, 2018.
[15] K. Asefpour Vakilian and J. Massah, "Performance evaluation of a machine vision system for insect pests identification of field crops using artificial neural networks," *Archives of phytopathology and plant protection*, vol. 46, no. 11, pp. 1262–1269, 2013.