Image Classification using VGG Network and Transfer Learning

Richard Anton rna63@drexel.edu

June 2021

Abstract

Using PyTorch and a VGG16 network architecture together with a dataset created from a subset of Tiny ImageNet, we train networks from scratch and compare against transfer learning results for the same dataset. The benefits of transfer learning are obvious since the results jump from an error rate of about 7% to about 30% at best without using a pretrained model, likely due to a larger dataset being needed for training the network effectively. We also compare for both from scratch and transfer learning different optimization methods, including stochastic gradient descent (SGD) with momentum, SGD with Nestorov momentum, Adadelta, and Adam.

1 Introduction

This work seeks to answer two questions. First, how much difference does training from scratch versus using a pretrained network for transfer learning affect the performance of image classification? And second, is 5,500 images for two classes enough data for training a VGG16 [SZ15] network without transfer learning from a pretrained model? Besides these two primary questions, two additional comparisons were made on the effect of using different optimizers during training.

1.1 GitHub Repository

The code for this work is available at `https://github.com/ranton256/cs583_final_project_classify`

2 Dataset

For the image dataset I used a subset of the Tiny ImageNet Challenge dataset [Ma15] by reducing it to 11 classes, which I then mapped into two classes, canine and feline. Six original classes mapped to the canine class(Chihuahua, Yorkshire_terrier, golden_retriever, Labrador_retriever, German_shepherd, standard_poodle). And five original classes mapped to the feline class (tabby, Persian_cat, Egyptian_cat, cougar, and lion). Originally the training data had 500 images per original class, but I used 10% of each for test data since the original dataset had training, validation, and test, but the test data was not labeled. This meant our training dataset has 2700 canine and 2250 feline images. Our validation dataset has 300 canine and 250 feline images. And our test dataset has 300 canine and 250 feline images. Each image is 64 by 64 pixels.

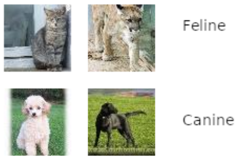


Figure 1: Dataset Examples

3 Related Work

VGG is a deep learning network architecture first published in 2014. It has variants like VGG 11, VGG 16, VGG 19 with varying complexity and network depths [SZ15]. A survey of deep learning network architectures and their uses can be found in *A Survey of the Recent Architectures of Deep Convolutional Neural Networks* by Khan et al [Kha+20].

Transfer learning uses the weights from a pretrained model to retrain the final layers and/or fine-tune the existing weights and is often used with CNNs trained on ImageNet dataset [Rus+15]. Transfer learning applied to neural networks dates back to work in the 1970's [Boz20; SF76].

The ImageNet dataset was used for training VGG for the pretrained model weights available in PyTorch. ImageNet is a very large image dataset frequently used for computer vision research labeled with many classes and synonym sets [Rus+15]. The Tiny ImageNet dataset [Ma15] comes from the ILSVRC [Rus+15] benchmark test but with fewer categories and lower resolution.

4 Network Setup and Training Framework: Requirement 2

Implement training using a modern framework (pytorch, mxnet, etc) for an architecture of your choice in google colab (or other GPU-accelerated environment)

PyTorch [Pas+19] was used for the machine learning framework and the VGG16 network architecture [SZ15] was used which is supported by the PyTorch torch.models module which contains prebuilt network models, which can optionally use pre-trained network weights. The vgg16_bn version was used which adds batch normalization. The final output layer of the VGG 16 architecture was replaced with a 2 output linear layer to match our number of classes. For all the results described below cross entropy loss was used. All training and evaluation was done using Google Colab [Col] notebooks plus additional utility code all of which is available in the project's GitHub repo

5 Training Algorithm Evaluation: Requirement 4

Evaluate the accuracy and speed using 3 different training algorithms (i.e., adam,momentum, etc)

During both transfer learning from a pretrained network and when training from scratch the network was trained using multiple algorithms for comparison. 1) with stochastic gradient descent (SGD) using momentum, 2) SGD with Nesterov momentum [Nes83], 3) with Adam optimizer [KB17], and 4) with Adadelta optimizer [Zei12]. All setups were trained for 30 epochs each. A learning rate schedule was used to lower the learning rate after every 9 epochs by a gamma(multiplicative factor) of 0.03.

```
exp_lr_scheduler = lr_scheduler.StepLR(
    optimizer_ft , step_size=9, gamma=0.03)
```

6 Data Augmentation

We used a number of data augmentation methods to improve generalization on our dataset, including random horizontal flips and random resized crops. Adding additional transforms, including random rotations and color jitter did not seem to further improve the performance by a measurable amount on either transfer learning or when training from scratch.

7 Transfer Training: Requirement 5

Compare your results to fine-tuning a pre-trained network on your same data.

For the transfer learning case, we used PyTorch's support for VGG 16 pretrained on the ImageNet dataset. All network layers prior to the last 6 sequential layers of the architecture had their weights frozen.

8 Challenges

It took quite a lot of time experimenting with dataset augmentation, learning rate schedule, optimzers, and hyperparameters used for optimizer algorithms such as Adam, to find effective settings. Since each test run takes 40 minutes or more this consumed much of the time available for the work. Initially training from scratch without transfer learning often failed to converge at all, returning results no better than random. This was finally overcome by better settings for the learning rate and learning rate scheduler.

One success was finding a method to search for an appropriate learning rate from *Programming PyTorch for Deep Learning* [Poi19] originally by Smith [Smi17]. This method works by calculating the loss for each learning rate over batches in an epoch while increasing learning rate in each mini-batch and examining the plot of learning rate versus loss for the steepest descent.

In the end, I was not happy with the results for Adam, which I think could be better given better setup on the optimizer parameters. Given how widely Adam is used, I expected better performance, and suspect the fault lies still in my own setup.

9 Visualization

To visualize training progress, I used a package called livelossplot [Pio] to display accuracy and loss graphs during training. The X axis is epoch of training in each graph. See Figure 2 for an example.

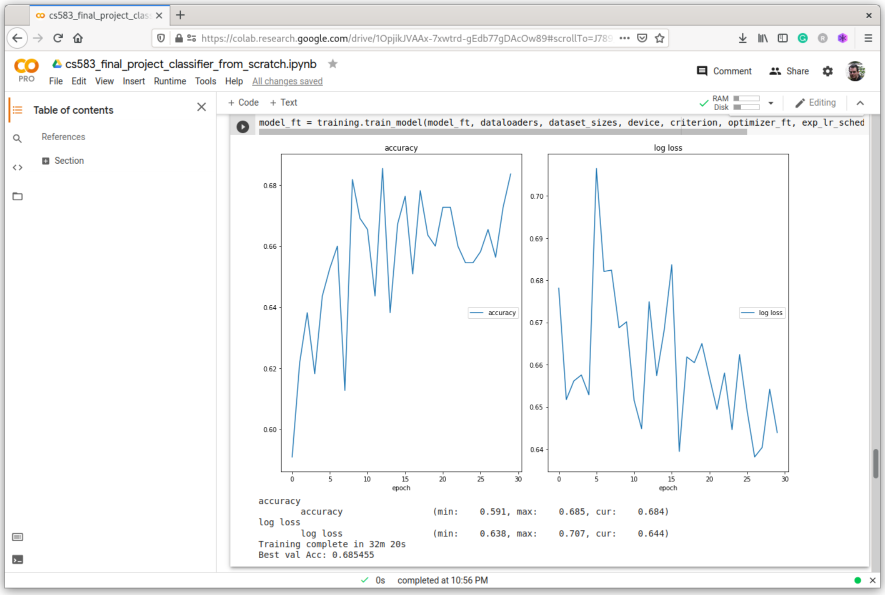
10 Results

The final results show reasonable performance for all optimization algorithms except for Adam which is significantly lower than the others. As described in the Challenges section, this is hypothesized to be due to parameters rather than Adam in general given its widespread adoption.

For the other optimization algorithms, accuracy ranged from around 68% to over 70% without transfer learning. For transfer learning using the pretrained model, accuracy jumps up to a range of around 89% to over 93%, excluding Adam which had approximately 65% accuracy on test data. The values for loss and accuracy during training validation generalize well to the test dataset in all cases. See Table 1 for more details on results.

11 Conclusion

Although it was possible to achieve reasonable performance training the model without transfer learning, the results were clearly better in terms of accuracy, loss, and consistency in converging when the pretrained model was used as the starting point. I suspect performance could be improved when not using



```
model_ft = training.train_model(model_ft, dataloaders, dataset_sizes, device, criterion, optimizer_ft, exp_lr_sched
```

```
accuracy
	accuracy             	 (min:    0.591, max:    0.685, cur:    0.684)
log loss
	log loss             	 (min:    0.638, max:    0.707, cur:    0.644)
Training complete in 32m 20s
Best val Acc: 0.685455
```

Figure 2: Adadelta Training From Scratch Visualization

Table 1: Validation and Test Results

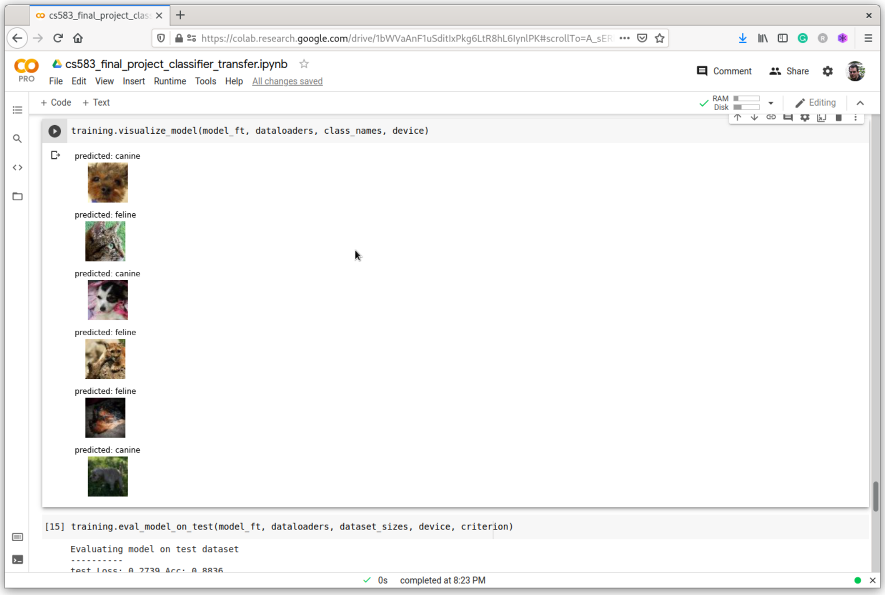
	SGD w/Momentum	SGD w/Nesterov	Adam	Adadelta
From Scratch				
Best Val. Acc.	0.7073	0.6818	0.5509	0.6855
Best Val. Loss	0.5670	0.6040	0.6920	0.6380
Test Acc.	0.7087	0.6893	0.5397	0.6655
Test Loss	0.5830	0.6086	0.6941	0.6922
Transfer Learning				
Best Val. Acc.	0.9309	0.9291	0.6382	0.8964
Best Val. Loss	0.1740	0.1710	0.6420	0.2660
Test Acc.	0.9309	0.9327	0.6509	0.8945
Test Loss	0.1632	0.1390	0.6931	0.2525

transfer learning and a pretrained model by using a simpler architecture or a larger dataset.

12 Future Work

Based on the difficulty getting a reasonably high accuracy for the network when training the model from scratch, the most interesting area of future work would be how to simplify the network to have fewer parameters and/or better augment or expand the dataset so that a more comparable outcome to transfer learning could be achieved when training from scratch.

Since the images are actually 64 by 64 pixels instead of 224 by 224 pixels


cs583_final_project_clas
https://colab.research.google.com/drive/1bWVaAnF1uSditIxPkg6LtR8hL6IynlPK#scrollTo=A_sER
cs583_final_project_classifier_transfer.ipynb
PRO
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing
0s completed at 8:23 PM

```
training.visualize_model(model_ft, dataloaders, class_names, device)
```

```
predicted: canine
predicted: feline
predicted: canine
predicted: feline
predicted: feline
predicted: canine
```

```
[15] training.eval_model_on_test(model_ft, dataloaders, dataset_sizes, device, criterion)
```

```
Evaluating model on test dataset
----------
test Loss: 0.2739 Acc: 0.8836
```

Figure 3: Adadelta Transfer Learning Example Classification Output

one way to reduce the network size and possibly improve results would be to change the architecture to match the input size for the input and early layers rather than resizeing the images. Another area would be to fine-tune the pretrained network parameters in the earlier layers of the VGG16 architecture during transfer learning instead of keeping them frozen.

I would also like to run this on AWS SageMaker to compare with Colab and make training and collecting results more automated.

References

[SF76] Bozinovskim S. and A. Fulgosi. "The influence of pattern similarity and transfer learning upon the training of a base perceptron B2." In: *Proceedings of Symposium Informatica* 3–121–5 (1976).

[Nes83] Yu Nesterov. "A method for solving the convex programming problem with convergence rate O(1/k^2)". In: *Proceedings of the USSR Academy of Sciences* 269 (1983), pp. 543–547.

[Zei12] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: 1212.5701 [cs.LG].

[Ma15] Yinbin Ma. *Tiny ImageNet Challenge*. 2015. URL: http://cs231n.stanford.edu/reports/2017/pdfs/935.pdf.

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (Dec. 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2015. arXiv: 1409.1556 [cs.CV].

[KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[Smi17] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks.* 2017. arXiv: 1506.01186 [cs.CV].

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Poi19] Ian Pointer. *Programming PyTorch for Deep Learning.* O'Reilly Media, Inc., Sept. 2019. ISBN: 9781492045359.

[Boz20] Stevo Bozinovski. "Reminder of the First Paper on Transfer Learning in Neural Networks, 1976". In: *Informatica* 44 (Sept. 2020). DOI: 10.31449/inf.v44i3.2828.

[Kha+20] Asifullah Khan et al. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8 (Apr. 2020), pp. 5455–5516. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09825-6. URL: http://dx.doi.org/10.1007/s10462-020-09825-6.

[Col] Google Colab. *Colaboratory.* URL: https://research.google.com/colaboratory/faq.html.

[Pio] Bartłomiej Olechno et al. Piotr Migdał. *livelossplot example: PyTorch.* URL: https://github.com/stared/livelossplot/blob/master/examples/pytorch.ipynb.