MiniPlaces Challenge
6.869 - Fall 2017
Team: ConvoRapt0r

Anelise Newman
`apnewman@mit.edu`
`kerberos: apnewman`
`6.869`

Nathan Landman
`landmann@mit.edu`
`kerberos: landmann`
`6.869`

Abstract

In recent years, neural networks have made incredible strides in their ability to identify and classify natural images. In this project, we take on the challenge of training a neural network from scratch to classify scene photographs using only limited training data. We make use of the MiniPlaces dataset, a collection of 120,000 low-resolution scene images divided into 100 categories, with the aim of maximizing our top-5 classification accuracy. We explore multiple network architectures, including AlexNet, ResNet-18, ResNet-34, and ResNet-50. Given ResNet-18's high initial performance and relatively quick training time, we performed further tests on this architecture to identify techniques that would boost our accuracy. We saw performance improvements from manual learning rate adjustment and extensive data augmentation, while adding gradient noise yielded no significant boost. We applied what we learned to the training of our final submission, ResNet-34 with an attenuated learning rate, batch size of 65, and extensive data augmentation. We achieved top-1 accuracy of 50.2% *and top-5 accuracy of* 78.8%.

1. Introduction

The objective of the MiniPlaces challenge is to design a network that takes a scene image as input and correctly identifies the correct scene category. Since an image can be described by multiple labels (a supermarket can also be a store), we aim to maximize the top 5 accuracy: the percentage of test images for which the correct ground truth label is within the top 5 predictions returned by the model. Additionally, we report our accuracy in returning the highest confidence label, but we do not attempt to specifically optimize on this accuracy.

Figure 1. A sample of images taken from the MiniPlaces dataset. Clockwise from the top right, the ground truth classification of the images are 'bedroom', 'volcano', 'golf course', and 'supermarket.'

1.1. Dataset

The MiniPlaces dataset is a subset of the Places Challenge dataset, a collection of scene images across a variety of categories [8]. The sample data contains $100,000$ images for training, $10,000$ images for validation, and $10,000$ images for testing across 100 scene categories. Every image is resized to 128×128 pixels, and each is given with a single ground-truth category label.

1.2. Division of work

We attempted to work together the majority of the time. Because we started early, we were able to find several time windows to do so. Therefore, most, if not all work was done collaboratively and evenly.

2. Related Work

Using deep neural networks is an increasingly common approach for solving classification problems in computer vision owing to the powerful classifiers that can be learned by these models. We review some of the architectures relevant for our project.

AlexNet [3] is a CNN containing 8 layers (5 convolutional and 3 fully-connected) separated by max-pooling layers. This network also makes use of the ReLU activation function to enable training of deep networks without reaching saturation. At the time of its debut, this architecture achieved top-1 and top-5 error of 37.5% and 17.0% respectively on the ImageNet LSVRC-2010 image classification contest.

More recently, Residual Neural Networks [1] have made it possible to train even deeper and more complex architectures. These feed-forward networks contain shortcut connections that provide a direct path from shallower layers (including the input) into deeper layers of the network. This construction allows for deeper layers to approximate a residual function, making learning quicker and more tractable. An implementation of ResNet won 1st place in the ILSVRC 2015 classification competition with 3.57% top-5 error.

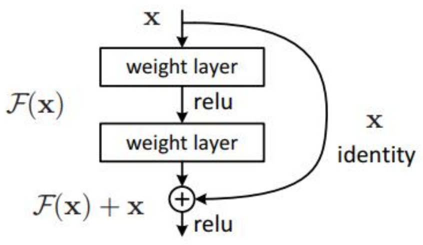
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. A building block of a ResNet, including a shortcut connection. [1]

Due to ResNet's ability to effectively train very deep networks, several architectures have been studied varying the number of layers. These include ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, among others.

3. Approach

All of our code was written in PyTorch. We developed our own dataloaders and wrote starter code for training and evaluating our network with the help of a couple of PyTorch tutorials [7, 6]. As the network trained, we observed the training accuracy every 100 iterations and the validation accuracy every epoch in order to keep track of our network's progress and tweak parameters as needed. We also saved copies of our networks' parameters after each epoch, which let us restart partially trained networks with different hyperparameters.

We experimented with several different types of data augmentations and hyperparameters to get the best performance.

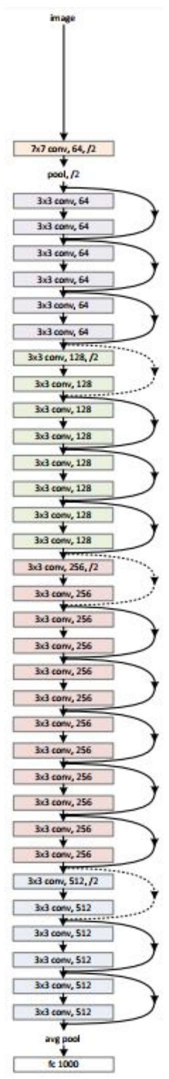

Figure 3. The architecture of ResNet-34. [1]

3.1. Network Architecture

We tried four different architectures to solve this challenge. To establish a benchmark, we first trained AlexNet as suggested in the starter code [3]. We later trained ResNet-18, and then ResNet-50 [1]. ResNets with more layers were not implemented due to potential memory constraints, and we skipped ResNet-34 at first due to time constraints.

We ran each network with a shuffled batch size of 10 due to memory constraints, and a learning rate of 0.001 using CUDA on one GPU. We ran the nets until the training/validation accuracy stopped increasing. ResNet-18 had the best performance of the three by a significant amount, so we used ResNet-18 in our other experiments. (Note that for our final model, we trained ResNet-34 from scratch to try to get more power from the increased layers, using parameters and techniques investigated in the rest of this report).

3.2. Data Augmentation

Since the MiniPlaces dataset is small, containing only 100,000 training images, we used data augmentation to increase our data set and reduce overfitting. Each time a training image was fed into the network, it was randomly cropped and subsequently upscaled to 224×224 pixels. We flipped the image horizontally with probability 0.5 and normalized according to the mean and standard deviation values from the ImageNet dataset [1].

In order to further boost our training samples, we experimented with adding even more random transformations. [2]. The updated training data processing pipeline was as follows: a random crop of an image was scaled up to 224×224. The resized crop was then transformed by both a random horizontal flip and a rotation ranging from -30 to 30 degrees. Random transformations were applied on some features as well: gamma ranging from 0.5 to 1.5, saturation ranging from -0.8 to 0.8, and brightness ranging from -0.3 to 0.3. Finally, the randomly transformed images were per-pixel normalized as described before.

The validation and test images also undergo transformations before prediction. They are first resized to 256×256 and center cropped to 244×244, with per-pixel transformations done exactly as in the training set.

3.3. Optimizers and Learning Rate Scheduling

We used cross-entropy loss as our loss function and root mean squared propagation (RMSProp) as our optimization method. RMSProp divides the gradient by a running average of its recent magnitude as we update the weights of the network.

We also did some manual learning-rate tuning. The starting learning rate of .001 was reduced by one tenth when learning plateaued, dropping the rate to .0001 and then again to .00001 until learning slowed to a negligible amount. This manual tuning of learning rate, as opposed to a scheduler, gave us more fine grained control and further understanding of the parameters and their performance.

3.4. Batch Normalization

Our ResNet architecture uses He batch normalization [2]. Here, weights are initialized according to a Gaussian distribution with mean zero and standard deviation $\sqrt{2/n}$, where n is the number of "connections" to the output of a convolutional layer, which is equal to the size of the filter times the number of color channels.

[1]ImageNet mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

[2]Code for our additional PyTorch transformations from [4]

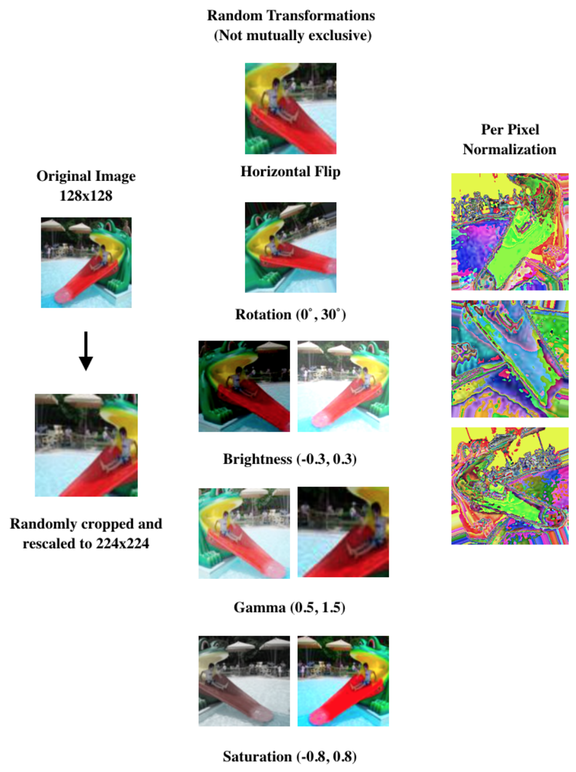

Figure 4. All training data is first cropped and resized to 224x224. Transformations are all randomized within the range described, and not mutually exclusive. Per Pixel normalization is done on every single transformed image.

3.5. Gradient Noise

Prior work [5] has proposed adding a small amount of noise to the gradient while performing the weights update during back-propagation. This change can help encourage exploration of the feature space (especially for complex networks) and has been shown to improve performance for several different network architectures and initializations. We tried adding gradient noise using the equations proposed in [5], using a weight update of the form:

$$g_t = g_t + N(0, \sigma^2)$$

where

$$\sigma = \eta/(1+t)^{.55}$$

and t is the epoch number.

4. Experiments

4.1. Architecture Selection

As in the TensorFlow starter code provided by the 6.869 staff, we originally used AlexNet as our model. This network quickly plateaued at a validation top-5 accuracy of around 48%.

Model	Epoch	Top-1 Acc. (Train)	Top-5 Acc. (Train)	Top-1 Acc. (Val)	Top-5 Acc. (Val)
ResNet-18	20	46.03	75.65	46.030	75.650
ResNet-50	35	42.22	73.323	42.440	69.030
AlexNet	38	14.558	37.894	20.330	47.980

Table 1. Our initial accuracy data for ResNet-18 vs ResNet-50 vs AlexNet. ResNet-18 was a good balance between speed and accuracy for running further experiments. Training accuracy is lower than validation accuracy as training accuracy has the images undergo transformations that make it harder for the machine to learn; transformations the network could possibly have never seen yet, whereas the validation set are more standard images whose features the network has most likely seen before.

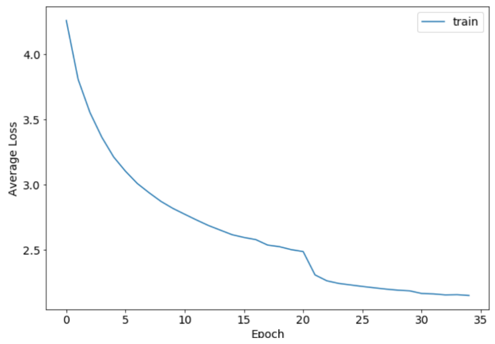

Figure 5. ResNet-50 training data loss as a function of epoch. The dip around epoch 20 is occurs when we manually decreased the learning rate from .001 to .0001. The model plateaus shortly thereafter.

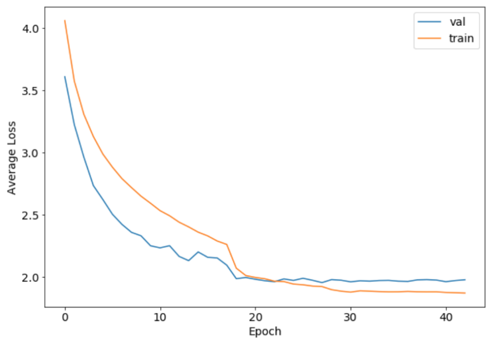

Figure 6. Training and validation loss for ResNet-18. The dip around epoch 20 is owing to the decrease in learning rate from .001 to .0001. Gradient noise was added around epoch 39; there is little seen benefit from doing so.

We compared AlexNet and two different ResNet architectures. A summary of the results is show in Table 1. ResNet-50 took considerably longer to train than ResNet-18, demonstrated relatively poor accuracy, and began to overfit at a lower accuracy than ResNet-18.

We decided to proceed with using ResNet-18 for further experiments, as it gave us good results with a small training time.

4.2. Learning Rate Adjustment

After achieving the above accuracy for the ResNet-18 network, we manually decreased our base learning rate from .001 to .0001 and trained for another 19 epochs until no further improvement was seen, which improved our results. The accuracy of the network before and after the learning rate decrease is shown in 2.

End Epoch	Base LR	Acc (train)	Acc (val)
20	.001	75.65	75.650
39	.001, .0001	78.63	78.630

Table 2. ResNet-18 top-5 performance before and after decreasing the learning rate for further training.

4.3. Effects of Gradient Noise

In general, adding gradient noise to ResNet-18 did not show significant improvement in the accuracy attained by the network. We tried adding gradient noise in two contexts: starting at epoch 20 using the weights obtained in part 4.1 and starting at epoch 39 using the weights obtained in part 4.2. In both experiments we used a base learning rate of .0001. However, there was no evidence to suggest that this technique improved accuracy of the network or helped move to a better minimum. See Table 3 and Figure 6.

ResNet-18 with added gradient noise		
Starting epoch	Train Acc	Validation Acc
20	78.42	77.980
39	78.64	78.43

Table 3. ResNet-18 top-5 performance after adding gradient noise. These models were initialized with weights from part 4.2 after ”starting epoch” epochs and trained until no further improvement was seen. Compare to the value seen without gradient descent, a top-5 accuracy of 78.63%.

4.4. Further Data Augmentation

Updating the training data processing pipeline to include more transformations (changing the brightness, saturation, etc.) slightly increased our accuracy.

Model	Train Acc.	Val Acc.
With Augmented Data	79.73	79.51
Without Augmented Data	78.63	78.63

Table 4. Top-5 accuracy acheived on ResNet-18 with and without additional data augmentation.

5. Final Results

Given the empirical results, our best model was built manually tuning the learning rate, using additional data augmentation, and to foregoing adding gradient noise. We trained a ResNet-34 with these parameters and achieved a slight validation boost over ResNet-18. We used a batch size of 60; data transforms in the form of random crops, horizontal flips, rotations from 0 to 30, brightness transformations from -0.3 to 0.3, gamma transformations from 0.5 to 1.5, and saturation levels form -0.8 to 0.8, as well as per-pixel normalization on the training set; and no gradient noise. All batches were randomly shuffled when training the network to improve generalization. The validation and training set were center cropped and resized to 224×224 with similar per-pixel normalization as in the training set. The model ran with a learning rate of 0.001 until epoch 29 and then learning rate of 0.0001 until epoch 70, when learning plateaued. This helped us achieve top-5 accuracy on our validation set just shy of 80%

	Top-1 Acc	Top-5 Acc
Train	53.399	80.268
Val	50.640	78.950
Test	50.2	78.8

Table 5. Final results - ResNet-34

6. Acknowledgements

A big thank you to the 6.869 staff, who conducted this experiment, as well as Spandan Madan, a humble code wizard who helped us debug some gnarly code.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv e-prints*, Feb. 2015.

[3] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[4] ncullen93. ”high-level training, data augmentation, and utilities for pytorch”. *GitHub repository*.

[5] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. *ArXiv e-prints*, Nov. 2015.

[6] pytorch. ”imagenet training in pytorch”. *GitHub repository*.

[7] M. Spandan. Pytorch Tutorial for Fine Tuning/Transfer Learning a Resnet for Image Classification.

[8] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.