LaCroixNet: 6.819 MiniPlaces Challenge

Milo Knowles and Ayush Sharma
Massachusetts Institute of Technology
Cambridge, MA
{mknowles, ayushs}@mit.edu

Abstract

In this paper, we train multiple neural networks on the Miniplaces dataset, a collection of 100K labelled images of scene categories, with the goal of experimenting, comparing and evaluating convolutional neural networks (CNNs) for this task. We train variants of VGG, experiment with hyper-parameters, employ data-augmentation, and use this challenge to test the performance of Capsule Networks (CapsNets) on large scale image classification. This paper presents our findings. We conclude the importance of regularization and data-augmentation for smaller datasets, and explore the potential of future work with CapsNet for complex supervised learning tasks.

1. Introduction

Following is an introduction of the challenge, and our division of work.

1.1. MiniPlaces Challenge

Miniplaces challenge is an image classification challenge using neural networks with the explicit constraints of **a)** Not using pre-trained models and **b)** having limited labelled training examples (100K vs 15mn in ImageNet)

1.2. Division of work

Milo and Ayush both worked on implementing various experiments in Tensorflow and Pytorch, although we eventually settled on Pytorch. Ayush performed research into data augmentation techniques, while Milo spent additional time researching CapsNet. Writing was divided equally. Ayush also handled offloading our training to AWS.

2. Final Approach: VGG16

After experimenting with several deep convolutional neural network architectures, we achieved our best performance with a modified implementation of VGG16 [5]. Despite the relatively large number of parameters in this network and small size of our training set, we found that this larger network still performed better than the lightweights networks that we also experimented with. See Section 3 for more detail on the various approaches we tried before settling on VGG16.

2.1. Architecture

As its name implies, VGG16 has 16 weight layers. The first 13 layers are convolutional, and interleaved with max pooling layers, while the last 3 layers are fully connected. See Table 2.1 for the full architecture.

Our model is modified from the original implementation of VGG16 in several notable ways. Since our challenge has only 100 scene categories, our final fully-connected layer has only 100 units. Because the final output layer is smaller by a factor of ten, we also reduced the size of the hidden fully-connected layers roughly proportionally, from 4096 units to 512 units. Because the majority of weights in a deep CNN come from the fully-connected layers, this greatly reduces the number of trainable parameters in our model. Whereas the original implementation of VGG16 has approximately 138 million parameters, our model has only 19.2 million, accelerating training, and mitigating the effects of overfitting. Finally, we use an image size of 128x128 rather than 224x224, which was used by the original authors of VGG.

2.2. VGG16 Training Parameters

We experimented with several adaptive optimizers such as Adam and Adadelta, but found that stochastic gradient descent performed better for training VGG16. Our learning rate schedule is manually adjusted: we begin with a learning rate of 0.005 and reduce the learning rate by a factor of 5 whenever the validation accuracy of our model shows no improvement for two epochs. We found that lowering the learning rate during a training plateau always caused our model's accuracy to continue rising, although at a slower rate. In addition, we use Nesterov momentum, which improved our model's final accuracy.

We used PyTorch's built-in categorical cross entropy loss

ConvNet Configurations		
VGG11	VGG13	VGG16
Input image: 128 x 128		
conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool		
conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool		
FC-512		
FC-512		
FC-100		
soft-max		

Table 1. The 3 ConvNet configuration used in our experiments. Our final and best-performing model was VGG16. The convolutional layer parameters are denoted as conv(receptive field size)-(number of channels). All layers except the final logits have a ReLU activation function, which has been omitted from this diagram for brevity.

Training Parameters	
Algorithm	SGD
lr (until epoch 19)	0.005
lr (until epoch 32)	0.001
lr (until epoch 50)	0.0005
lr (until end)	0.0001
momentum	0.9
nesterov	True
dropout	0.5

Table 2.

function, and applied dropout with a probability of 0.5 to the final layer of the network during training. Although we experimented briefly with batch normalization, this initially had a detrimental effect on our accuracy, and we did not have time to fully explore this method of regularization.

VGG16 Performance	
Validation Accuracy (Top 1)	TODO
Validation Accuracy (Top 5)	TODO
Test Accuracy (Top 1)	TODO
Test Accuracy (Top 5)	TODO

Table 3.

2.3. Data Augmentation

Despite reducing the number of parameters in our implementation of VGG16 by an order of magnitude, we still had a large number of parameters relative to the number of examples in our training set. Consequently, we relied heavily on data augmentation to improve the validation accuracy of our model. We applied random horizontal and vertical flips, rotations, crops, and color jittering to the training data [1]. In addition, our training images were normalized such that each color channel had a mean close to 0.5 and a standard deviation close to 0.225. During validation and testing, data is normalized using the same parameters, but the training augmentations described above are not applied.

2.4. Model Performance

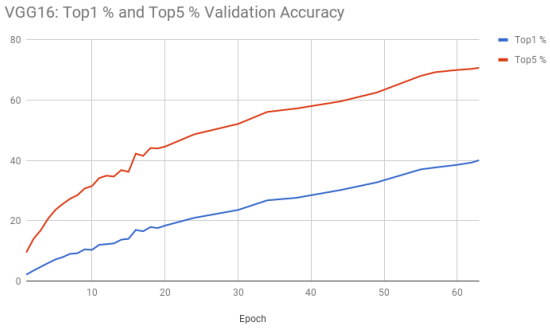

Figure 1. VGG16 accuracy

3. Experiments and Results

We experimented with various architectures, configurations, learning rates, optimizers, batch normalization and data-augmentation techniques. This section lists and details our experiments and their results as measured via several metrics.

3.1. VGG11

One of the first models we trained, a VGG11 architecture seemed a good choice to begin with. Given the effectiveness of VGG models [5], and the smaller size of an 11 layer VGG, we chose to train this first. See Table 2.1 for the full architecture of our variant. The table below lists our parameters for training –

VGG11: Training Parameters	
Algorithm	Adam
batch-size	200
loss function	Cross Entropy Loss
dropout	0.5

Table 4.

VGG13: Training Parameters	
Algorithm	SGD
batch-size	56
learning-rate	0.001
momentum	0.9
loss function	Cross Entropy Loss
dropout	0.5

Table 5.

In terms of performance, we got our top5 accuracy upto 50% with this architecture. See figure 2 for full plot –

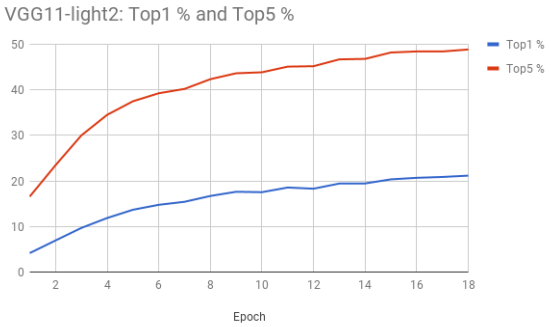

Figure 2. VGG11 accuracy

3.2. VGG13

Next, we experimented and switched to a deeper VGG13 model. Our initial results with VGG11 were promising but not necessarily the best, and we wondered about the effect of making it deeper. See Table 2.1 for the full architecture of our variant. Table 3.2 lists our parameters for training –

In terms of performance, we got our top5 accuracy up to 56% with this architecture.

4. Applying Capsule Networks to Large Scale Image Classification

4.1. Motivation

Given the recent success of Capsule Networks on digit classification as demonstrated by Hinton et. al. [2], we decided to experiment with this new paradigm of supervised learning on the more complex task of scene recognition. In their paper, they describe the concept of a capsule as follows:

> A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part. We use the length of the activity vector to represent the probability that the entity exists and its orientation to represent the instantiation parameters. Active capsules at one level make predictions, via transformation matrices, for the instantiation parameters of higher-level capsules.

Whereas traditional CNNs lose spatial information about an image through max pooling layers [3], capsule networks use a routing-by-agreement procedure which does not obscure spatial information. Because capsule networks explicitly model the geometric relationships between parts of an image and the whole, they seem intuitively useful for modeling a scene, which can be thought of as a hierarchical collection of objects.

The translational invariance of CNNs may miss the possibility to exploit the relative location of objects in a scene. For example, a wooden cross located on top of a building is a strong indicator of a church, but a wooden cross located on the ground may indicate that we are in a graveyard. We argue that a capsule network is better equipped than a CNN to understand this kind of relationship.

4.2. Architectural Modifications

The main difficulty we had to address while adapting CapsNet to the MiniPlaces challenge (PlacesCapsNet) was the computational difficulty of training the network. Our images are 128x128 rather than 28x28 (MNIST), and have three color channels instead of one. Rather than 10 digit capsules, we have 100 scene category capsules. The routing-by-agreement procedure inside of the CapsNet implementation [4] relies on a slow inner loop, which is not optimized as a tensor operation. This proved to be a bottleneck during training, and limited the size of the capsule network that we could feasibly train.

In order to extend the architecture used by Hinton et. al. [2] on MNIST, we added an additional convolutional layer and an additional capsule layer. By adding an additional convolution layer with a stride equal to 2, we reduce the size of the output volume from the convolution layers, while showing no drop in model performance. This greatly reduces the model parameters, the majority of which lie in the fully connected region between the convolution layers and the primary capsule layer.

Because a scene has a much more complex object hierarchy than a handwritten digit, we added another layer of capsules - we call this the secondary capsules - to allow the network to learn higher level objects. We found that adding

CapsNet Configurations	
Original CapsNet	PlacesCapsNet
24x24x1 Image	128x128x3 Image
conv9-256 (stride=1)	conv9-64 (stride=2) conv9-128 (stride=2)
32 PrimaryCaps stride=2 channels=8 kernel=9x9	32 PrimaryCaps stride=2 channels=8 kernel=9x9
- -	16 SecondaryCaps channels=12
10 DigitCaps channels=16	100 SceneCaps channels=16

Table 6. A comparison of our PlacesCapsNet architecture against that of Hinton et. al. [2], which was applied to the MNIST dataset. We deepen the architecture by adding an additional convolution layer and a secondary capsule layer. As in the original implementation, we use 3 routing iterations between adjacent capsule layers.

a third layer of capsules gave no additional improvement in performance, although this is mostly because more capsule layers made the network prohibitively slow to train. The secondary capsule layer has fewer units, but a higher dimensional vector of instantiation parameters. Our justification for this is that there should be fewer higher level objects in a scene, but it should require more parameters too effectively describe a higher level object.

4.3. Performance

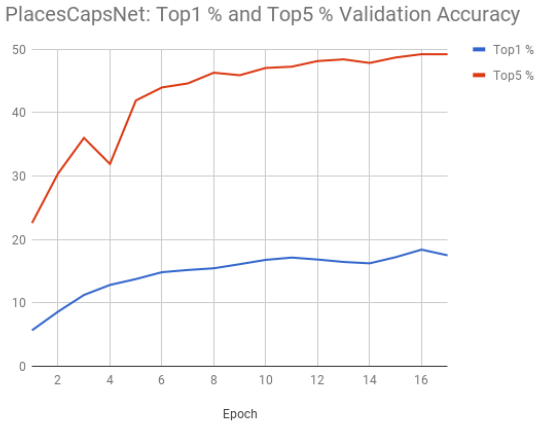

Figure 3. Validation accuracy of PlacesCapsNet.

We trained PlacesCapsnet in 3 different trials, while varying the architecture. All trials were conducted with PyTorch's built-in Adam optimizer, and its default parameters. Although Hinton et. al. use a reconstruction term in their loss function to improve their performance on MNIST and

PlacesCapsNet Experiments			
Model	**PCN1**	**PCN2**	**PCN3 (best)**
Parameters	16,192,768	44,520,960	7,622,272
f heightconv1 Filters	256	256	64
conv2 Filters	–	–	128
PrimaryCaps	32	64	32
SecondaryCaps	16	32	16
Batch Size	16	4	32
Time Per Epoch	2.5 hours	4.25 hours	30mins
Top5 Accuracy	46.9	30.52*	49.24
Top1 Accuracy	15.42	9.2*	18.39

Table 7. A comparison of our PlacesCapsNet architecture against that of Hinton et. al. [2], which was used on the MNIST dataset. We make the architecture deeper by adding an additional convolution layer and capsule layer. As in the original implementation, we use 3 routing iterations between adjacent capsule layers and the Adam optimizer. All convolution layers use a stride of 2 and a 9x9 kernel size. Note that PCN2's validation results, denoted with an asterisk, are not accurate because we stopped training this model before it could converge.

coerce capsules to encode physically sensible parameters, we did not include this in our model, as it greatly increased the size of our network. Where not specified, the parameters of the experiments are the same as described above.

We also experimented with changing the number of routing iterations from 3 to 5, but this adversely affected training. The model became stuck at a much lower accuracy during training. It is likely that using more routing iterations during early training causes the network to have a self-reinforcing behavior where certain capsules dominate routes to other capsules and the network favors exploitation over exploration. Varying the number of routing iterations over time may be an interesting hyperparameter optimization to explore in the future.

Finally, we point out the fact that our best implementation of PlacesCapsNet has only 7.6 million parameters, which is less than half the size of VGG16 (19.2 million). With proper tuning and more experimentation with architecture, we believe that this network could achieve comparable accuracy to a smaller CNN with roughly the same number of parameters.

References

[1] S. Dieleman. My solution for the galaxy zoo challenge. `http://benanne.github.io/2014/04/05/galaxy-zoo.html`, 2014.

[2] S. S. Geoffrey E Hinton, Nicholas Frosst. Dynamic routing between capsules. *NIPS*, 2017.

[3] G. E. Hinton. What is wrong with convolutional neural networks? `https://www.youtube.com/watch?v=rTawFwUvnLE`, 2017.

[4] K. Iwasaki. A pytorch implementation of the nips 2017 paper dynamic routing between capsules.

[5] A. Z. Karen Simonyan. Very deep convolution networks for large-scale image recognition. *ICLR*, 2015.