Team Scenic Recursion: 6.869 Final Project Report

Benjamin Schreck (with Kenneth Cooper and Sarah Liu)
6.869
bschreck@mit.edu

Abstract

Image classification by computers has surpassed human accuracy in the past year in terms of object recognition, but lags behind in the harder problem of scene recognition. In this paper we develop algorithms for scene classification. We attempt to use both a convolutional neural network and a more complex recurrent neural network/convolutional neural network combination in order to classify images into 100 scene categories using the Miniplaces dataset. Because of time limitations and framework difficulties, we were unable to produce any presentable results. However, we learned a lot about computer vision and deep learning in the process.

1. Introduction

Every moment we are conscious, we are aware of our surroundings. The ability to recognize our current location in the world is something so intuitive that we seldom even realize we are doing it. Furthermore, we find it almost as easy to recognize scenes from photographs using only visual information, even if we have never physically traveled to that location. However, training a computer program to do the same thing is incredibly difficult. The amount of processing our brains are doing to understand where we are can only be fully appreciated in the context of trying to replicate the result artificially.

In order to do that, computer vision researchers have taken an approach similar to how we as humans learn—using supervision. By providing enough training examples, researchers can instruct a machine to differentiate between them, which can internalize a model that generalizes to unseen examples. To do this we need tens of thousands of examples, and to do it well we need much more than that.

In this paper, we describe a model that uses the Miniplaces dataset, consisting of 100,000 128x128 labeled training images, and attempts to learn how to classify scenes. We implement both a deep convolutional neural network, which has become standard in the computer vision field, and a recurrent neural network with convolutional subnets that allows for a form of "visual attention". This means that the model can focus on different parts of the image sequentially before coming to a conclusion about its class. We believe that this is a natural approach to use, and is close to how humans classify scenes.

2. Related Work

Object-recognition classifiers have been steadily improving over the past few years due to the rise of deep convolutional neural networks, and are generally trained on the ImageNet 2012 database. Krizhevsky et. al kickstarted this new approach in 2012 with a model called AlexNet, which consisted of 3 convolutional layers followed by 2 fully connected layers and a pioneering activation layer called ReLu, for rectified linear unit, and achieved a top-5 error rate of 15.3%. This considerably advanced the state of the art. Many more models followed this one, with increasingly deep nets such as Google's Inception network[9].

Interesting additions to the standard convolutional model include the addition of a parameterized term into the ReLu activation unit, which allows different layers of the net to respond differently and decrease the loss of information due to saturation[5]. Graham et. al introduced the concept of Dropout, which randomly selects activations to ignore and reduces the chance of overfitting[4]. In a similar vein, Iioffe and Szegedy presented batch normalization, which reduced overfitting and accelerated training time by keeping the input to each activation unit to a fixed mean and variance, thus preventing excess saturation in the nonlinear regime of the activations[7]. Tieleman and Hinton presented a novel variant to traditional gradient descent called RMSProp, which increases performance and training time by keeping a running average of gradients[10]. Lastly, Glorot and Bengio showed how a novel initialization scheme could reduce over-saturation and thus allow for easier training and convergence[3].

Ba et al presented a combination of convolutional neural networks with recurrent neural networks to classify multiple objects per image[2]. This model extracts features from glimpses, or small slices of the input image, in a recurrent sequence.

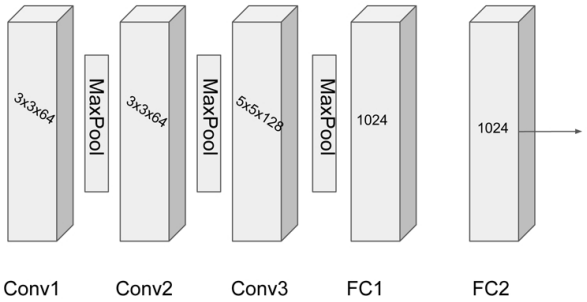

Figure 1. Our initial model used 3 convolutional layers interleaved with max-pooling layers, followed by 2 fully connected layers. The first two used a 3x3 kernel with 64 dimensions, the last used a 5x5 kernel with 128 dimensions, and the fully connected layers both used 1024 units.

Classifiers trained on the large Imagenet database perform poorly on scene recognition. Moreover, existing scene datasets like the Sun dataset are too small for the same types of deep neural network models to effectively train. To increase scene recognition performance, Zhou et. al. and the MIT vision group build a database of many millions of scene-labeled images, called the Places database. Zhou et. al then trained a deep convolution neural network similar to those used in object recognition on the Places dataset, to achieve better than state-of-the-art performance [11].

3. Approach

We opted to use the deep learning framework recently released by Google called Tensor Flow[1]. Tensor Flow was designed to abstract away many implementation details, and let users program a high-level graph description of a series of computations. This graph is then automatically built and run using lower level abstractions. The key advantages to us were its brevity and seeming simplicity in interface, usage of an extensive Python front-end, automatic generation of visualizations, and speed due to its optimized C++ back-end.

We started with a simple convolutional model provided in a Tensor Flow tutorial, and then added additions and modifications to it while simultaneously attempting to build a more complex recurrent model like that used for multiple object recognition in Ba et al.

3.1. Convolutional Neural Network

Our initial model was similar to that provided by Tensor Flow to classify images in the CIFAR10 dataset, and is based on Alexnet. It consisted of 3 convolutional layers, with a max-pooling layer after each one, and followed by 2 fully connected layers.

3.2. Data Augmentation

Because our dataset is much smaller than the actual Places dataset, our models are highly prone to overfitting, and would benefit from additional data. We can artificially produce data by randomly cropping and distorting images– as long as the information about the type of scene is preserved (judged by human eyes). In our model, we crop images from 128x128 to 100x100, flip images left to right, adjust the brightness, and adjust the contrast, all in a random fashion. The last two are non-commutative so we randomly select either brightness or contrast adjustment to perform first. Each image is fed through the distortion pipeline, so that after the image is slightly different every epoch. All of these operations are simple function calls in Tensor Flow.

3.3. Reducing Overfitting and Training Time

We attempted to apply Dropout, Batch Normalization, and Xavier Initialization to reduce overfitting and increase training time. Since Batch Normalization and Dropout do very similar things, we only do tried one of them at a time. Dropout is a simple function call in Tensor Flow. Batch Normalization requires a bit of tinkering, but the code on how to do it in Tensor Flow was posted by a Tensor Flow contributor to Stack Overflow. Xavier initialization consists of simply changing the initial distribution of variable weights based on the number of input nodes to and output nodes from each variable.

3.4. Visual Attention with Recurrent Neural Networks

Inspired by Ba et. als work on multiple object recognition with visual attention**??**, we propose a recurrent neural network model that incorporates a form of visual attention. The basic intuition is that scenes contain many smaller aspects that, when taken together as a whole, make up the scene category. This naturally leads to a model that is able to take into account multiple small portions of the original scene image and pool its different ideas of what is contained in the scene together in order to draw a conclusion about the overall scene category. One way to exploit multiple substructures is with a recurrent neural network, more commonly used for natural language processing. Recurrent neural networks, or RNNs, take as input a sequence, and for each step in the sequence, the input element is combined with the networks own internal state from the previous step to produce an input to the RNN. This way, the RNN acts as a state machine, and can understand time-dependent patterns.

In our case, the sequence is not well-defined, as all we are given is an image. However, we can attempt to allow the RNN to learn an order for looking at subimages, or glimpses. The RNN will keep track of its state as it captures information about these various glimpses, and then after some number of steps produce an output that depends

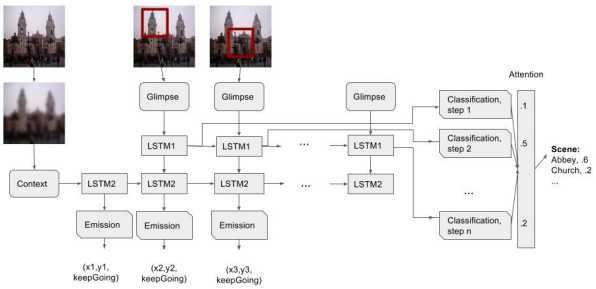

Figure 2. Using recurrent neural networks to capture visual attention. Images are downsampled and fed into the context network: a 3-layer convolutional network. This outputs the initial state for the second level of the recurrent network, using LSTM cells. The output of the second layer of LSTM feeds the emission network, a fully connected net, which outputs a location tuple and flag signifying whether to extract the next glimpse at that location. The glimpse network receives a slice of the image and its location, and outputs an embedding of the image as an input into the first LSTM cell. This LSTM cell then feeds its state into the classification network, which includes 1 fully connected layer and a softmax. The classifications at each step are linearly combined with an attention vector to produce a single output probability distribution over categories.

on all the previous glimpses it has seen. There are a few details to the model that are necessary because we are dealing with visual input. Fig. 3.4 shows the interconnections between the components in our recurrent model. The details are hashed out in the next sections.

3.4.1 Glimpse Network

The first detail specific to a visual recurrent model is that convolutional networks should be employed for each glimpse, because they are well-known as successful image feature extractors[8]. Each network shares weights, and consists of 2 separate subnetworks for taking into account both the what (the glimpse image), and the where (the location tuple). The what network consists of 3 convolutional layers, alternating with maxpools and feeding into a fully connected layer. The where network consists of a single fully connected layer. These final fully connected layers and multiplied together element-wise to produce a final output vector that is fed into the RNN. Fig. 3 contains a visualization.

To speed up training time and increase probability of convergence, it should be possible to initialize the glimpse network's weights with those of the first 4 layers of the convolutional model (before the final fully connected layer) described in 3.1.

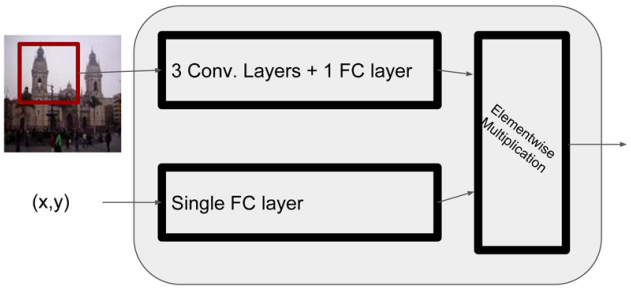

Figure 3. The glimpse network receives a section of image and a location tuple as input, sends them each through various layers, and multiplies a feature vector learned for each of them element-wise.

3.4.2 Recurrent Layers

If we only consider the output state of the RNN after it has run through some number of steps, the output will be skewed toward the information contained in these later glimpses. Instead, we output a probability distribution over scene class labels at every step along the way, and linearly combine all of these distributions using a learned attention weight vector. One can come up with ways to allow this attention vector to depend on the particular example and thus allow the model to dynamically generate attention weights based on how important the glimpse CNN layer or RNN layer believes the particular step is. However, given the complexity that would entail we decided to simply learn a single attention vector. Furthermore, we employ two layers of long short-term memory cells, or LSTM, inside the RNN, with the first responsible for classification and the second responsible for learning the sequence. LSTMs are known to have good gradient-preserving abilities- that is they are able to back propagate the gradient signal far back in time without it vanishing or exploding[6]. The first LSTM layer is initialized with a state of zeros, and its first input is the initial output from the glimpse network. The second is initialized with a state containing the output of the Context network, described in section 3.4.3, and an input of zeros, where in the future it will take as input the output from the first LSTM layer.

3.4.3 Context Network

The context network is responsible for providing an initial context to the RNN so that it knows where to start looking. It does this by taking as input a downsampled version of the original image (to 64x64 pixels) and using 3 convolutional layers and a single max-pooling layer. Similar to the glimpse network, this network could also be initialized with weights from a pretrained convolutional model.

3.4.4 Classification Network

At each step the first LSTM cell feeds its state into a Classification network, which is responsible for outputting a probability distribution over scene categories. It consists of a fully connected layer fed into a softmax. These probabilities are linearly combined by multiplication with the attention vector. One implementation detail is that the attention vector needs to be a fixed size, so we make it the size of the maximum allowable steps the RNN can take, and add in zero vectors to stand in for any unused steps.

3.4.5 Emission Network

The RNN needs to both learn where to look, and when to stop looking. For this we employ a fully connected network that takes in the output of a LSTM unit from the second layer of the RNN at a particular step, and spits out 3 values: an (x,y) coordinate pair that describes where in the image to extract the next glimpse, and a value called keepGoing that determines whether or not to actually perform another step. We keep extracting glimpses and progressing forward until either keepGoing drops below a threshold or we reach some max number of steps- both hyper-parameters set at 0.1 and 10, respectively.

4. Experimental Results

We initially tried to implement the more standard convolutional approach described in 3.1. Once we were able to produce output we split into two groups: I would try to implement the recurrent visual attention model, and the others would try to implement extra features to the convolutional model that attempt to reduce overfitting and lower training time. However, at the time of the split we had not evaluated the model yet, and soon realized that the model was not actually training. The measured cross-entropy loss would quickly drop in the first 200 steps to a number below 10 (typically 4.6) and then remain there for all time, as shown in Fig. 4. This is consistent with random chance, and indeed always produced a top-1 precision of .01 and a top-5 precision of .05, which for 100 class labels is exactly random guessing.

We were able to visualize the internal nodes in the network using Tensor Flow's Tensor Board visualization feature. We show histograms generated for the weights, activations, and gradients of convolutional layer 1 in our network and the tutorial's in Fig. 4. The tutorial network was the same model trained on the CIFAR10 dataset. In comparing the two, it is clear that our model is training. The activations are more uniform over time and have a larger percentage much closer to zero, the gradients are much smaller in absolute value, and the weights remain distributed according to the same distribution across all time. The other, trained

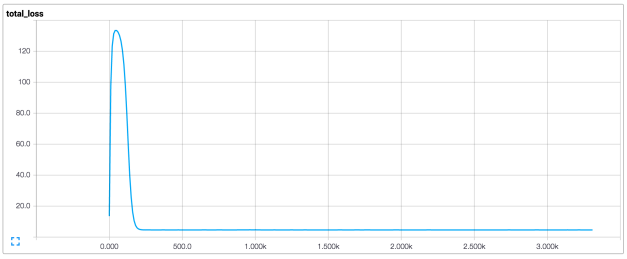

Figure 4. Cross-entropy loss while training convolutional model. This graph was almost identical with every parameter setting, and every addition/removal we made to the network.

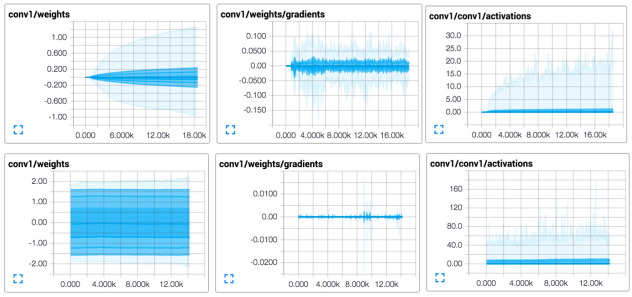

Figure 5. Histograms of the weights, gradients, and activations inside a trained network and our network. The top is the CIFAR10-trained network. X-axis is thousands of iterations, Y-axis is value of the variable, color indicates number of variables at that particular point. The deeper the color the more values at that location.

model's weights progressively grow larger with a larger deviation from the mean over time, and that deviation is significantly less than our untrained model's.

4.1. Issues

We (perhaps foolishly) opted to use the brand new Tensor Flow library to build our classifier, which was released by Google with lots of hype and enthusiasm earlier this semester. While it seems like a very easy, extensible, and fast library for building these kind of nets, it lacks the battle hardening of other systems like Caffe and Theano due to its novelty. As such we struggled to do very basic things like load images with labels correctly, and too much of our time was spent on implementation.

4.1.1 Implementation

We spent a long time aligning images with labels, and believed to have correctly done so, however as described above our models did not better than random chance. This was the case even for a simple convolutional model using exactly the same network as the CIFAR10 tutorial provided on Tensor Flows website (which correctly trains using their code for reading the CIFAR10 dataset). We thus conclude that we were never able to correctly align the images

with the labels even though Tensor Flows debugging tools seemed to say otherwise.

4.1.2 Theoretical

Another issue was discovered too late to be able to fix, and as such we would not have been able to get our recurrent model to train correctly even had we fixed the label alignment problem (Fig. 4.1.2 shows the loss as a function of iteration). The model runs but the gradient back-propagation stops at the beginning of each glimpse layer. This makes sense since the gradient of the glimpse-extraction is more or less meaningless, however it means that we have no way to train the emission network, which is the only way to learn the sequence of locations for extracting glimpses. Therefore, the emission network remains at its initialized weights and always outputs the same location. This leads to the Recurrent and Glimpse networks simply receiving the same input for all the steps, and only allows us to train on a random subset of each image. It reduces the model to an initial context network, an overly complex RNN that probably does not do anything useful, and a glimpse network that only learns features from a small crop of the image that most likely does not contain enough information to correctly classify the scene.

To fix this issue, we would have to provide updates to the emission network based on how effective each particular step was at classifying the image. This would consist of transforming the classification loss per step into some loss that depends on the location and keepGoing value that the emission network outputs. Ba et al.[2] accomplish this by effectively performing a reinforcement learning step. However, their work is slightly different in that they are trying to classify multiple objects, and each object is contained within a separate glimpse, so the classifier at each step should be outputting a different value. In ours, we want each classifier to classify its guess at the overall scene. Given more time, we think we would be able to derive a gradient update that would allow us to train this model, and believe that it would eventually perform well on the Miniplaces or Places datasets.

5. Conclusion

We attempted to build both a standard deep convolutional neural network as well as a more complicated recurrent neural network with visual attention to classify scenes from the Miniplaces dataset. In doing so, we experimented with lots of ideas from recent state-of-the-art classifiers, and learned a lot about computer vision in so doing. We also learned a lot about how to use the Tensor Flow framework, including initiating several pull-requests on Github for features that are missing and that we know how to build from reading the source code. Unfortunately, we could not ultimately produce any substantive results, because our models never trained to perform better than random chance. Given a little more time, we believe we could have resolved these issues and trained a high-performing scene classifier.

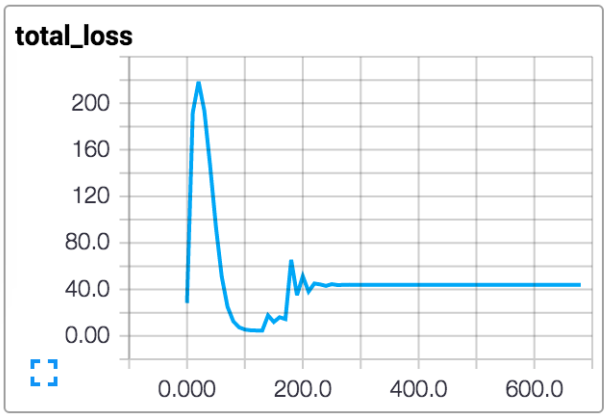

Figure 6. Cross-entropy loss while training recurrent model. This graph was almost identical with every parameter setting, and every addition/removal we made to the network.

References

[1] Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2

[2] J. L. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. 2015. Published as a conference paper at ICLR 2015. 2, 5

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256. 2010. 1

[4] B. Graham, J. Reizenstein, and L. Robinson. Efficient batch-wise dropout training using submatrices. 2015. Published as a conference paper at ICLR 2015. 1

[5] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015. Published as a conference paper at ICLR 2015. 1

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. Published as a conference paper at ICLR 2015. 1

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012. 3

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D.Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 1

[10] T. Tieleman and Hinton. Lecture 6.5 - rmsprop, 2012. 1

[11] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. pages 487–495, 2014. 2