Hotel Image Classification using Deep Learning

Teja Kanchinadam
UNC Charlotte
tkanchin@uncc.edu

Abstract

Hotel Image Classification is of the real-world problems for object classification. Typically, to tackle this task, large datasets and domain-specific features are used to best fit the data. In this project, I propose to implement, train, and test one of the best state-of-the-art classifier, with the added constraint of limited data and time. I experiment this with transfer learning for fitting this model over our domain and report the loss and accuracy rates of this classifier.

1. Introduction

Object recognition has been very challenging because of the subtle differences between classes of images. For example, if we consider the 'Aerial View' class of Hotel Images which was used for this project, it is very difficult for the computers to classify the images correctly because each image is different from another.

The main challenge for the computers is to learn both the low-level latent features and high level features of the images. Even though, if we assume that the computers are capable of learning all those features, our model will definitely fail to generalize over test data. And this usually require tremendous amount of computation and with say, if we are using this on a very large dataset such as ImageNet, this approach would be no less than a herculean task.

One approach to this problem is to use deep learning. There has been a big shift of change in the object category classification especially in the last 5 years starting from ILSVRC 2012 when Alex Krizhevsky and Geoffrey Hinton used Convolutional Neural Networks for ImageNet Challenge and their top-5 error rate has beaten all the other teams by a far margin. Though, it's not a new idea, the things they did has made the world think about the forgotten CNN's and its importance in Computer Vision. Deep Learning has now become the state of the art for Object Classification because we don't have to figure out the features ahead of time, use same neural net approach for many different problems, it is fault tolerant and it scales really well. Therefore, in this project I've used deep learning for Image classification of hotel images.

I've used a model which was the winner of ILSVRC 2014 – Googlenet [8].

I conducted experiments on the Hotel Images dataset [9], which contained around 38000 images for training and over 19000 images for testing. The entire dataset has been spread across 8 different classes. Because of the limitations on the computation time and difficulty of the task, I wasn't able to rectify the errors I made during training and this resulted in low-quality results.

2. Approach

2.1. GoogleNet

GoogleNet was designed to be a direct improvement over AlexNet for Image Classification which was the winner of ILSVRC 2012[7] and GoogleNet was the winner of ILSVRC 2014[8]. It has 22 layers, compared to AlexNet's 8 layers, and the number of parameters is smaller than AlexNet because of the smaller number of weights in the fully connected layers.

GoogleNet has three classifiers and these classifiers generate three outputs for every image at various depths. Google has considered only the last classification layer in their approach; therefore in this project even though I've used three classifiers while training to calculate the loss, I've only considered the last classifier for testing because it usually gives the lowest loss.

2.2. Transfer Learning

Transfer learning is the main key to my approach for two reasons. Firstly, I don't have a large dataset to fully train my deep network and with smaller datasets the Convolutional Neural Nets will not work well. Secondly, in transfer learning, we will retrain a subset of the pre-trained network weights and features and the overall effect will result in a model that minimizes overfitting and significantly much lesser work than to train the complete network.

2.3. Inception Architecture Details

GoogleNet uses Inception architecture which is a combination of max-pooling and 1x1, 3x3 and 5x5 convolutions. The main idea of using inceptions in our nets is because using only Max-pooling or only convolutions will result is some amount of data loss between layers and our model will fail to recognize some of the images during our classification because of this loss. It is also implemented because of *statistical invariance*. Statistical Invariance means that when a cat is placed in the center of the image and when we train our classifier using this cat image, our classifier will predict the cat images only if they are centered. If the cat images are to the left or right side of the image, then our classifier will fail to identify them. Inception was a technique which was introduced by the Google for the first time in ILSVRC 2014.

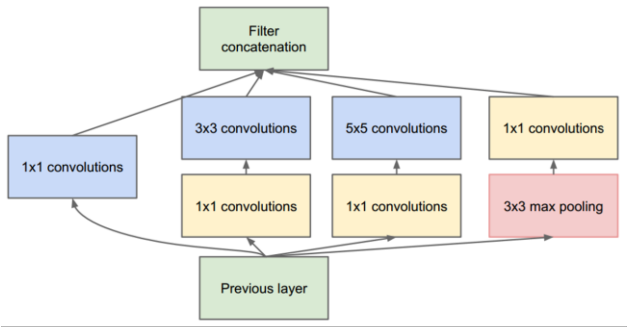

Figure 1: Construction of One Inception Layer
(Note, image is directly sourced from [4])

The inception modules are stacked on top of each other and as we move to the higher layers, the ration of 3 x 3 and 5 x 5 convolutions also increases.

2.4 Weight Initializations

Initially, during training I've used the weight initializations as random but given the depth of the network, some of the gradients were becoming zero while back propagating. I realized that this is *vanishing gradients*. After thorough research, I came to know about Xavier initialization [9]. Caffe deep learning framework has default Xavier initialization in its GoogleNet model.

2.5 Activation units

GoogleNet was made built on rectified linear units (RELU). But recently in 2014, leaky RELU [10] is said to have better performance on the model than a normal RELU.

2.6 Dropout

It is concept introduced by Geoffrey Hinton to avoid overfitting in the networks by preventing the neurons from co-adapting too much. It essentially forces the network to learn multiple independent representations of the same data by alternate random disabling of neurons in the learning phase.

2.7 Architecture

The input image of size 224x224x3 is passed to a convolutional layer and all the convolutions have stride sizes of 5x5 and 7x7 with zero mean. It is then followed by a max pooling layer and all the inceptions modules are stacked on each other with max pooling layers in between them. The output of the final inception layer is passed to a average pooling layer and its output is passed to a dropout. The output of dropout is our classifier and when the output of classifier is passes through a softmax layer, we get our eight probabilities.

3. Experiments

3.1 Software Library

I've used Caffe deep learning framework to implement this project. They've the best documentation among other software libraries and it was highly recommended by the Stanford 231n Computer Vision course instructor Andrej Karpathy in one of his lecture videos.

3.2 Preprocessing

I've divided my training set into training data and validation data, and then converted my training data and validation data into LMDB files. In LMDB files are stored as key-value pairs. To convert the data into LMDB, I've defined a text file and in the text file the input is stores as follows "path-to-the-image class". The class should be a numerical value and I've named my classes from 1 to 8 giving relevant image paths using train.csv file. While saving the data as LMDB file I've resized all the images to 256x256 and with zero mean and RGB as (2, 0, 1).

3.3 GoogleNet pre-trained weights

I've downloaded the GoogleNet pre-trained weights from GitHub. These pre-trained weights are a result of ImageNet training with over 15 million images and they it has a top-5 error rate of 6.1% and top-1 error rate of 18.1%.

3.4 Fine-tuning the network

I've downloaded the GoogleNet network model from GitHub. This network will be in the form Google's protobuf format and it comes with an extension "prototxt". The first layer of the network is data layer and I've defined my LMDB files in these layers. Also, in these layers you can define your network's pixel depth and pixel size. I've left it as default 224x224.

Later, I freeze all the other layers such as convolutions, activation layers, inception layers, max pooling layers, dropouts by changing hyper parameters 'lr_mult' and 'lr_decay' to '0'. By doing this, we are allowing the network not to back propagate and update its weights.

GoogleNet has three classifiers. I've changed the names of the three classifiers and left their hyper parameters 'lr_mult' and 'lr_decay' as it is. By doing this, the network will only update the weights of these three loss classifiers only. In short, only these classifiers will get trained.

There is a file called 'solver.txt' in which all the hyper parameters of the network will be defined such as the learning rate, growth decay, batch size, number of iterations, etc..,

I've defined the growth rate as 0.001 and weight decay as 0.96 and batch size as 32. Higher batch sizes result in higher efficiency and higher accuracy. I've given the number of number of iterations as 5000 and ran the experiment. The net with run for 40 hours. 30 hours for training and 10 hours on validation set.

A new file 'bvlc_googlenet_5000.caffemodel' will be created. This file has now the weights for our Hotel Image Classification and we are ready to test. To test the model, use the same network, but instead of training call the test parameter. Testing can be done in two ways. One is batch testing which is quick if we have a large set of images and the other is testing the image one by one. For this experiment, I've run batch testing, by creating a LMDB file for the test data and defining in our network file. Append the results of testing into a CSV file.

4. Results

Because of some mistake in creating the LMDB file, my network wasn't efficient. I've only got a loss of 2.34 and an accuracy of 27%. Given the amount of training time and difficulty of the task, this error was hard to see. Because of these results, I came 26th on Kaggle and was among the lowest performers in the class.

Almost every image was getting misclassified and images were getting classified at random.

5. Debugging and actual causes of Error

After the deadline for Kaggle, I've found that I made an error while creating the LMDB file. While creating the text file for the LMDB which I've mentioned earlier in the pre-processing step. The text file should have been "path-to-image class", where the class name should have started from '0-7'. But instead, I've made the class name start form '1-8'. Because of this, I was getting a random value at class 0 and my class 8 was completely omitted from the classification and the remaining classes have been shifted for one place. For example, the classification for class one for coming for class two and classification for class two was coming for class three and so on.

After, I've rectified my mistake. I've re-run my code for only 1000 iterations and **came third on Kaggle with a loss of 0.42 and an accuracy of 87%.** This is standard state-of-art accuracy for GoogleNet.

I've checked my own human-accuracy on some of the images and I can clearly see that I'm making an error of at least 5-6% on the images. This explains on how complex the images were and how hard it was to classify them.

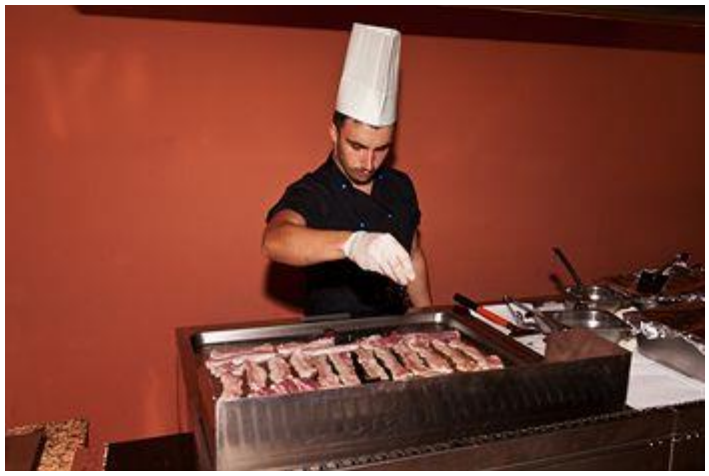

Figure 2: Misclassified Image

The above image in Figure 2 has been misclassified by new classifier (after Kaggle Submission). From the above picture we can see that, there are a lot of colors distributed throughout the screen. And the classifier gave equal probabilities for restaurant, guestroom and lobby for the above image. If you observe the images, you can realize that it was super hard for the classifier to classify this image as the colors were so equally distributed. Therefore, the classifier was confused.

6. Future Work

To train the classifier using different sets of models such as VGG Net, Res Net which are said to give higher accuracy rates for image classification. Of course, given the time and complexity I've only run GoogleNet for 1000 and 5000 iterations, if I would have run it for 10000 iterations, things would have been very different.

Acknowledgements

I would like to thank **Dr. Richard Souvenir** for giving this opportunity to work on real-world problem and his course slides and lectures have given me a start for this Project. **Stanford 231n Computer Vision,** this course was really helpful in understanding the hyper parameters of deep learning and the concept of transfer learning. **Caffenet** documentation was really helpful in writing the scripts and executing them.

References

[1] Derrect Liu: Image Classification of Vehicle Make and Model using Convolutional Neural Networks and Transfer Learning

[2] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/, 2014.

[3] Angelova, A. and Zhu, S.: Efficient object detection and segmentation for fine-grained recognition. In: CVPR 2013

[4] Szegedy, et al.: Going deeper with convolutions. In: CVPR 2014

[5] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: ICLR 2015

[6] Pan, S. and Qiang, Y.: A Survey on Transfer Learning. In: IEEE Transactions on Knowledge and Data Engineering 2010

[7] Krizhevsky, A., Sutskever, I. and Hinton, G.: ImageNet Classification with Deep Convolutional Networks. In: NIPS 2012

[8] Donahue, J. et al.: A Deep Convolutional Activation Feature for Generic Visual Recognition. 2014

[9] Xavier Glorot: Understanding the difficulty of training deep feedforward neural networks

[10] Bing Xu, Mu Li : Empirical Evaluation of Rectified Activations in Convolution Network