Image Classification using Deep Learning
CUNY 698 - MS in Data Science
Final Research Project

Walt Wells, 2018

Abstract

A significant byproduct of Moore's law is the democratization of large scale computation and storage resources. Modern machine learning techniques like deep learning that can accurately model highly dimensional data like images, video, and audio are now considerably less temporally and economically expensive and are accessible to anyone with a credit card and a laptop. This paper will demonstrate modern methods used to architect and train a simple convolutional neural network for image classification.

Keywords: CloudML, Convolutional Neural Networks, Deep Learning, ImageNet, Keras, R,

1. Introduction

Deep learning techniques are one of the most exciting and effective new approaches to building a machine learning system that can operate with accuracy and precision in a complex real-world environment [1]. This is important because for difficult problems with highly dimensional data like image classification, deep learning models like Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) are among the best performers in benchmarked competitions and are even outpacing their human subject matter expert counterparts at image classification tasks [2]. But how do they work?

This study will seek to explore a variety of deep learning techniques by building a convolutional neural network and conducting image classification over a popular benchmarking dataset (ImageNet) [3] and it's smaller sister (TinyImageNet) [4]. Our focus will be split on two tracks: a) data engineering, or managing the storage, computational infrastructure and backend for handling the dataset and models; and b) deep learning modeling, or training and optimizing a convolutional neural network to make classification predictions, and exploring methods for measuring, comparing, and improving model performance [5, 6, 7, 8, 9]. The approach will lean heavily on tooling released or updated in 2018 like the keras and cloudml packages in R [10, 11].

2. Literature Review

Our literature is divided along two lines - a) textbooks, MOOCs, and articles that facilitate comprehension of deep learning techniques at large, and b) research articles that are concerned with image classification using deep learning techniques.

2.1. Deep Learning Overview

Resources like Make Your Own Neural Network [5] provide a very basic entry point to deep learning, walking through the basics of creating a single layer ANN in Python and introducing key training techniques like gradient descent. On the other end of the spectrum lies Deep Learning [1], which provides a strong mathematical and theoretical framework for a wide variety of deep learning techniques and applications. This includes a broad survey of model types (ANNs, CNNs, RNNs, DNNs, etc), optimization and learning techniques, regularization, and applications for deep learning architecture.

The other range of overview literature showcases hands-on training using multiple languages. Hands on Machine Learning with Scikit-Learn & TensorFlow [8] teaches deep learning techniques in Python and provides an excellent Github repo with Jupyter Notebooks for each chapter. For an overview of deep learning in R, we utilize Machine Learning with R [6] and Introduction to Deep Learning Using R [7]. For a fuller representation on how to use Keras in R, we rely on the newly published (2018) Deep Learning with R [10]

Finally, the Coursera class on Neural Networks for Machine Learning [9], originally offered in 2013 by one of the "godfathers" of the field, Geoffrey Hinton, provides excellent videos, quizzes, and homework assignments in Octave (an open source sister to the commercial statistical software Matlab).

2.2. Image Classification Using Deep Learning

In 1988 Yann LeCun introduced the architecture for LeNet-5 in Gradient-Based Learning Applied to Document Recognition [12], which set major new performance benchmarks in image classification using the MNIST dataset.

The author introduces alternating convolution and pooling layers and set the groundwork for future CNN research.

ImageNet Classification with Deep Convolutional Neural Networks [13] is a landmark paper describing the methods used to expand on LeNet-5 and create AlexNet, which excelled in the 2012 ImageNet competition. The authors introduce important techniques like using ReLUs (Rectified Linear Units) as an activation function, reducing overfitting using regularization techniques and dropout, and expanding the training dataset through data augmentation techniques.

Visualizing and Understanding Convolutional Networks [14] goes a step further by introducing a visualization technique called Deconvnet to gain insight into what is happening in the hidden middle layers of a neural network with multiple layers. The authors leverage the ImageNet 2012 dataset and their ZFNet Architecture to showcase the power of this technique.

Multi-column Deep Neural Networks for Image Classification [2] provides a framework for introducing transfer learning techniques by leveraging models trained on multiple image classification datasets. The authors also provide a strong framework on the utilization of GPUs for deep learning.

3. Material and Methods

It is important to note that while training an image classifier over the ImageNet and TinyImageNet datasets is a non-trivial task, it has been done before and pre-trained models readily available to data scientists can be used for both image classification and other transfer learning tasks. We undergo the exercise of building our own model to gain knowledge about techniques for building and training a convolutional neural network and modern data engineering. In this section we will explore the dataset, the architecture and hardware techniques required to support this effort, and our approach to modeling and training.

3.1. Data Engineering

3.1.1. The Dataset

The ImageNet dataset is a major touchstone for image classification and deep learning benchmarking. This project explores the 2012 release and only utilizes subsets related to classification, ignoring tasks like edge detection or bounding boxes. The training set consists of 10,000,000 labeled images depicting more than 1,000 classes. The validation dataset has 50,000 images, 50 from each class [3]. We will not utilize the test dataset, as the ground truth is not provided.

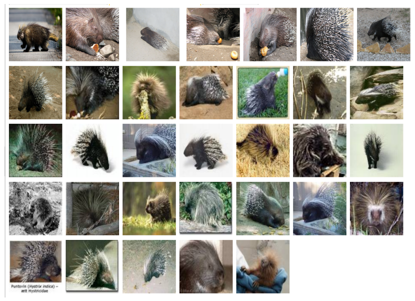


Figure 1: ImageNet Data, Class = Porcupine.

Managing the compute and storage used to explore deep learning techniques over the ImageNet dataset requires the use of modern big data techniques. Keras is a user-friendly back-end library with an R interface that can be built on top of Tensorflow. Using Keras will allow for a computational job to be broken out across multiple compute nodes and can leverage hardware like GPUs to cut down on training time.

3.1.2. Dataset Management

Since it is not efficient to manage the entire dataset in a VM or cluster, we will use batch processing to pull samples from the dataset and feed them to our CNN for training. Any techniques to modify, pad, scale, downsample, or augment the dataset will be initialized during this batch sampling procedure. The training, test and validation sets were served for research purposes only by the Stanford Vision lab as large .tar files [3]. These were transferred into a bucket in Google Object Storage by running curl commands with resuming and retry flags, checking md5sum to ensure integrity, unarchiving the tar file, and then using Google's gsutil rsync command to move the files to object storage. The primary bottleneck was the Vision lab server, where transfer speeds maxed out around 300 Mbps. A Google Compute VM with expanded disk was used to execute the transfers. In addition, the Keras workflow utilized required images to be in distinct directories representing their classes. While the training sets were already organized in this fashion, a script was created to organize the validation sets accordingly.

Table 1: ImageNet 2012 Preparation

Type	Size	Setup Time
Training	138 GB	152 hrs
Validation	6.3 GB	14 hrs
TinyImageNet	1 GB	2 hrs

3.1.3. Establishing a Pipeline and Workflow

Since training a CNN can be a temporally and economically expensive endeavor a pipeline was established over the TinyImageNet dataset. The pipeline could be run locally for an epoch or two to confirm it works, then run in the Google CloudML service over the TinyImageNet dataset loaded into Google object storage. The pipeline was designed to a) transfer data from our raw object storage to our cluster, b) train the model, c) provide helpful feedback on workflow progress, d) help determine bottlenecks e) make it easy to visualize the model's worst errors to help with debugging and f) be able to save, checkpoint, and export a model for later use [1, Chap. 11].

For the development and testing phase, the TinyImagenet [4] dataset was pushed into a separate bucket in object storage and organized in an equivalent way to the full Imagenet. TinyImagenet has 200 classes, downsampled to 64 * 64 pixels with 500 training images and 50 validation images for each class where the ground truth is known. Overall, the dataset is approximately .8GB and the bucket structure was laid out to mirror that of the full Imagenet Dataset. The pipeline was built in such a way that there are adjustable parameter flags so that once the pipeline was deemed mature, the flags could be swapped to train over the full dataset.

3.1.4. R - Keras and CloudML

We used relatively new tools and libraries in R (keras and cloudml) released or updated in 2018. The keras package in R provides a method to leverage deep learning techniques using Keras (in our case, with Tensorflow as the backend) and has excellent and intuitive functions for designing model architecture, training and optimizing models, data generation, data augmentation, and hyper parameter tuning [10]. The cloudml package provides a simple way to submit R keras jobs to the Google ML Engine for training, testing, and prediction [11]. Using this package abstracts away the need for a data scientist to manage and tune the hardware required for deep learning, making it as simple as submitting a job to an engine API with a preference for the hardware type. While Google has recently made their TPUs available to the public for rental [15], our training will rely on a maximum of 4 NVIDIA K80 GPUS.

3.2. Deep Learning Modeling

3.2.1. Classification Task

We focus on building a model that can accomplish the following image classification task as defined by the ImageNet 2012 challenge:

"For each image, algorithms will produce a list of at most 5 object categories in the descending order of confidence. The quality of a labeling will be evaluated based on the label that best matches the ground truth label for the image. The idea is to allow an algorithm to identify multiple objects in an image and not be penalized if one of the objects identified was in fact present, but not included in the ground truth [9]."

3.2.2. Performance Metric

To compare model performance, we use a top 5 accuracy rate metric based on the classification task. This is defined as the percentage for which the actual image class is among the top 5 predictions in the model's predicted answer. We will look at this metric over both the training and validation sets, and also review top-1 accuracy and loss metrics over both sets.

3.2.3. Convolutional Neural Net Architecture

Convolutional neural networks are a feed-forward deep learning technique that excels at image classification tasks. A feed-forward neural network is a series of interconnected layers that pass information forward from the input using a series of activation functions (relu, sigmoid, etc.). The connections between layers are weights that determine the strength of the values passed forward from the previous layer.

The essence of what makes a feed-forward neural network work is the weights connecting each layer where features are learned. When the model is trained, the inputs are pushed forward through the network and the output compares the prediction against the ground truth using the desired loss function. Adjustments to the weights are then backpropagated through the network using an optimization algorithm that bases adjustments on the loss function. The deep learning community has developed a number of algorithms to update the weights by minimizing the loss function (in this case, our classification accuracy rate) using an algorithm like Gradient Descent, Stochastic Gradient Descent, ADAM, or RMSProp [1].

For images, convolutional layers are excellent ways to learn the local features like the ear or nose of a dog in an image, and subsequently recognize those features anywhere in any new image. Pooling is used to downsample and group features learned by convolutional layers, so that the overall model parameters stay within a manageable framework. By alternating convolutional layers with pooling layers we can learn and discover spatial hierarchies of patterns. After our convolutional and pooling layers, we can flatten our feature output and augment the network with a softmax classifier that will provide a predicted probability for each of the dataset image classes, which we can compare against the actual class, allowing us to determine the top 5 accuracy rate.

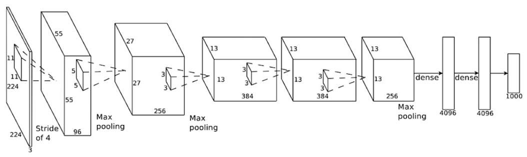


Figure 2: AlexNet Architecture

We will train and compare architectures that are roughly based on the AlexNet architecture that won the ImageNet 2012 challenge and relied on 5 convolutional layers, 3 max pooling layers, and 3 dense and flattened layers that output to a 1000 way softmax classifier. At the time, AlexNet was able to achieve a top-5 classification error of 15%, a major improvement on prior benchmarks around 26% [13]. To date, CNN architectures like ResNet and VGG Inception have achieved Top 5 error rates under 5%.

3.2.4. Model Tuning

The essence of training a network lies in tuning the backpropagation algorithm by finding the right learning rates, batches, epochs, and activation functions and preventing overfitting against the training and validation datasets. In addition, regularization and dropout techniques can be implemented to add noise or information loss to the network to help prevent overfitting. We will use a variety of search tools in the cloudml and the keras packages to find the optimal tuning parameters and improve and compare model performance.

3.2.5. Data Augmentation

When training CNNs, data augmentation is a relatively inexpensive way to improve model accuracy. Essentially, you update or modify the existing training set by shifting, rotating, shrinking, or flattening the existing images so that the model has more aspects to generalize. For each of our model architectures, we will implement data augmentation for comparison.

4. Complications

Since the technology leveraged for this research project is new to the public, we encountered a few issues someone approaching this tech in a few months or years likely would not. All issues would ultimately prove surmountable with a larger budget and an expansion of toolset range, but as outlined below, kludge solutions were chosen instead.

4.1. Training using Full Imagenet Dataset

The R CloudML package is very new. While the package is well supported by the Rstudio team, we experienced a problematic issue pairing the data generator function with our full Imagenet dataset. The function is a wrapper for gsutils object storage access and a data generator which essentially serves to batch format and feed images into the model. On multiple occasions, submitted jobs ran successfully over the TinyImageNet dataset, but stalled when tasked with the full dataset. Tickets were logged with Google Compute and with the Rstudio CloudML library team (which have overlapping members). Both teams were very responsive. While we worked together to hypothesize, diagnose, and troubleshoot, there was no solution found using the existing toolkit.

A potential solution is to move away from Google's CloudML service, to manage and configure our own GPU cluster, and build our own data generator functions to batch process images from Object Storage. Alternatively, we could to move the dataset into a large SSD directly mounted to the GPU cluster. Engineering either of these solutions would have taken our project over budget. Instead, since the pipeline ran successfully over the smaller TinyImageNet dataset, we opted to instead train and model using it. Since the TinyImageNet dataset is much smaller and more pixelated (64 * 64 images), our performance metric results are poorer than the benchmarks attained by researchers training on the full ImageNet Dataset. While undesirable, we can still effectively implement and demonstrate CNN training methods using the tools and pipeline that were built.

5. Results

As noted above, from this point forward, results reported were trained on the TinyImageNet dataset, entirely using the CloudML and Keras tools supported by the Google teams.

5.1. Model Architectures

As seen in Table 2, two different simple CNN architectures were trained, each basic, but similar in structure to AlexNet. In all, we used ReLU as an activation function. Each featured Convolutional layers interspersed with pooling layers. To prevent overfitting, these were periodically woven with layers that performed batch normalization and dropout. Finally, our CNN was flattened and connected to a softmax activated classifier that assigned probabilities to each of the available classes.

Model 1	**Model 2**
CNN Layers	**CNN Layers**
$InputImage = 64 * 64 * 3$	$InputImage = 64 * 64 * 3$
$ConvLayer, Filters = 32, Kernel = 3$	$ConvLayer, Filters = 32, Kernel = 3$
$MaxPoolingLayer, Size = 2$	$ConvLayer, Filters = 32, Kernel = 3$
$ConvLayer, Filters = 64, Kernel = 3$	$MaxPoolingLayer, Size = 2$
$MaxPoolingLayer, Size = 2$	$BatchNormalizationLayer$
$ConvLayer, Filters = 128, Kernel = 3$	$DropoutLayer$
$MaxPoolingLayer, Size = 2$	$ConvLayer, Filters = 32, Kernel = 3$
$ConvLayer, Filters = 128, Kernel = 3$	$ConvLayer, Filters = 64, Kernel = 3$
$MaxPoolingLayer, Size = 2$	$MaxPoolingLayer, Size = 2$
	$BatchNormalizationLayer$
	$DropoutLayer$
	$ConvLayer, Filters = 128, Kernel = 3$
	$MaxPoolingLayer, Size = 2$
	$ConvLayer, Filters = 128, Kernel = 3$
	$MaxPoolingLayer, Size = 2$
	$BatchNormalizationLayer$
Classifier Layers	**Classifier Layers**
$FlattenLayer$	$FlattenLayer$
$DropoutLayer$	$DropoutLayer$
$DenseLayer, Units = 512$	$DenseLayer, Units = 512$
	$DenseLayer, Units = 1024$
$DenseLayer, Softmax$	$DenseLayer, Softmax$

Table 2: Trained CNN Architectures

Since we are dealing with batches of images, our model inputs are 4 dimensional tensors. 2 of the dimensions are simply image height and image width. The next dimension splits the image into 3 parts along the RBG color channels. The final dimension represents the batch of images. So a batch of 32 images at 64 pixels squared would be fed to the network in a tensor with dimensions of [64, 64, 3, 32].

5.2. Tuning and Performance

For all models, we used the Adam algorithm for our optimization function with a learning rate at 1e-4 and decay set at 1e-6. To both avoid overfitting and keep costs down, we implemented an early stopping function that would stop training when the validation loss metric did not improve after 30 epochs. The Adam algorithm monitors our loss metric ("Categorical CrossEntropy") and then adjusts the network weights.

Keras offers a nice flag feature for tuning model hyperparameters. When you have model parameters defined as variable flags, instead of hardcoded values, a separate "tuning" yaml file can be submitted with the job. This file contains a range over which to search for the optimal parameters for a given variable. We played a bit with creating these flags and managing our tuning search range, but ultimately had relatively satisfactory performance over the TinyImageNet dataset so did not go further down this path. As computation becomes more democratized through increased availability of on-premise compute, commercial cloud compute, or commercial ML as a service offerings, tools like automated hyperparameter tuning (and even automated model or model architecture selection!) are becoming more available to the public.

In all, about 25 training and hyperparameter tuning jobs were submitted to CloudML, of which 17 completed successfully. We were able to leverage the models trained to make class predictions on new images using calls to the CloudML API.

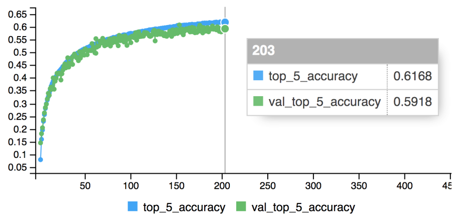
Table 3 shows four of the best overall performers trained over the TinyImageNet dataset. There were a few models not listed here that achieved > 90% top 5 accuracy on the training set, but were overfitting badly and did not generalize well, performing poorly on the validation dataset. As a baseline for comparison, randomly guessing an image class from the TinyImagenet Dataset amounts to .5 top 1% accuracy, and 2.5% top 5 accuracy.

Figure 3 b) compares their Top5 Accuracy on the training data across epochs and Figure 3 a) shows the relationship between the validation and

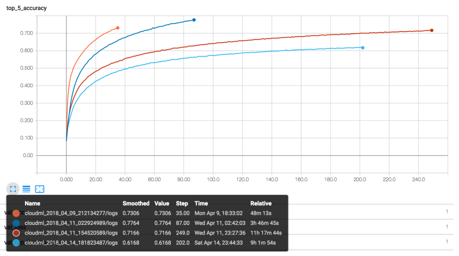
	Model 1	**Model 1**	**Model 2**	**Model 2**
$RunID$	212134277	181823487	022924989	154520589
$TrainAcc$	.4384	.3450	.4872	.4300
$TrainTop5Acc$	.7306	.6168	.7764	.7166
$ValAcc$	.1767	.3200	.3475	.2981
$ValTop5Acc$	.3974	.5918	.6086	.5674
$Dropout$	.5	.5	.5, .5, .75	.5, .5, .5
$DataAugmentation$	$FALSE$	$TRUE$	$FALSE$	$TRUE$
$EarlyStoppingEpoch$	38	203	88	250

Table 3: TinyImageNet Model Performance

training accuracy for our run with model 1 with ID 18123487. While its top5 accuracy maxes out around 62%, this trained model achieves a very similar performance over our validation set, indicating it is unlikely to be overfitting to the training data. Overall, the more complex model (model 2) w/o data augmentation has the best performance metrics, but it is worth noting that there is a 16% difference in top 5 accuracy between the training and validation datasets indicative of overfitting. What may be preferred about this model is that the top 1 accuracy is also very high, indicating the model is generally identifying classes correctly.



(a) Top 5, Model 1 w/ Data Aug.



Name	Smoothed	Value	Step	Time	Relative
cloudml_2018_04_09_212134277/logs	0.7306	0.7306	35.00	Mon Apr 9, 18:33:02	48m 13s
cloudml_2018_04_11_022924989/logs	0.7764	0.7764	87.00	Wed Apr 11, 02:42:03	3h 46m 45s
cloudml_2018_04_11_154520589/logs	0.7166	0.7166	249.0	Wed Apr 11, 23:27:36	11h 17m 44s
cloudml_2018_04_14_181823487/logs	0.6168	0.6168	202.0	Sat Apr 14, 23:44:33	9h 1m 54s

(b) Top 5, 4 Model logs in Tensorflow

Figure 3: Visualizing CNN Model Performance

6. Conclusion

Working with new open source tools to implement Convolutional Neural Networks for image classification, we were able to achieve a reasonable predictive classification accuracy using a small developmental dataset. Since deep learning techniques excel with large and complex datasets, it is reasonable to infer that were we able to train using the full ImageNet dataset, we would have seen further improvements in our performance metrics.

An interesting tertiary lesson reinforces a point we've learned again and again working with data: different and unique problems appear at different magnitudes of data. Depending on the a) size or velocity of one's data, b) one's environment for model deployment or c) one's appetite for model complexity, or d) some combination of all these factors, a completely differently engineered solution may be required for the same problem. An engineered pipeline that works well at one level may not scale as the complexity of the problem space increases.

7. Appendix

All code for data engineering, model architecture, training and hypertuning can be found at:

https://github.com/wwells/CUNY_DATA_698/tree/master/ImageNetCNN

8. References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[2] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, IEEE Conference on Computer Vision and Pattern Recognition (2012) 3642–3649.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, 2012.

[4] S. Y. L. Fei-Fei, J. Johnson, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, 2017.

[5] T. Rashid, Make Your Own Neural Network, CreateSpace Independent Publishing Platform, Seattle WA, 1 edition, 2016.

[6] T. B. III, Introduction to Deep Learning Using R: A Step-by-Step Guide to Learning and Implementing Deep Learning Models Using R, Apres, New York, NY, 2 edition, 2017.

[7] B. Lantz, Machine Learning with R - Second Edition: Expert techniques for predictive modeling to solve all your data analysis problems, Packt Publishing, Birmingham, UK, 2 edition, 2015.

[8] A. Gèron, Hands on Machine Learning with Scikit-Learn & TensorFlow, OReilly Media, Sebastopol, CA, 1 edition, 2017.

[9] G. Hinton, Neural Networks for Machine Learning, Coursera MOOC, 2013.

[10] F. Chollet, J. Allaire, Deep Learning with R, Manning Publishing, Shelter Island, NY, 1 edition, 2018.

[11] J. Allaire, R Interface to Google CloudML, RStudio, 2018/01/10.

[12] Y. Le Cun, L. Bottou, Y. Bengio, P. Haffner, Gradient based learning applied to document recognition, Proceedings of IEEE 86 (1998) 2278–2324.

[13] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[14] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, CoRR abs/1311.2901 (2013).

[15] F. Lardonis, Googles custom TPU machine learning accelerators are now available in beta, TechCrunch, 2018/02/12.