Fine Tuning Convolutional Neural Nets *

Jordan Hoehne †

May 8, 2018

Abstract

First introduced in 1998 by Yan LeCun, Convolutional Neural Nets have received noticeable attention in recent years. With the rate at which data is being generated and with research and development increasing in this field, Convolutional Neural Nets have seen wider use cases and integration into our lives. I experiment with different parameters in my model in order to fine tune a Convolutional Neural Net to classify different images of cats and dogs. I configured the model based off of Siraj Raval's Convolutional Neural Net and the Keras sequential model documentation. In my paper, I explore: (i) different optimization techniques; (ii) varying dropout layers; (iii) varying the amount fully-connected layers and (iv) the models accuracy after data augmentation. The final results lead me to conclude that the best parameters are Adaptive Moment Estimator, Adam, for the optimizer, three convolutional layers activated with ReLU, followed by a pooling layer, and two fully connected layers with a dropout layer set to 0.5.

*Thank you to Siraj Raval and the Keras documentation for teaching me the fundamentals of Convolutional Neural Nets.

†Department of Economics, University of Oklahoma. E-mail address: jhoehne@ou.edu

1 Introduction

Artificial Intelligence, AI, and Machine Learning, ML, are perhaps some of the greatest innovations human-kind will have created and explored in the last century. Research and development in these fields is occurring at astonishing rates as the cost to run complex models has gone down and the access to big data has increased. AI is best categorized into three categories, the last being more theoretical. The first, narrow AI, refers to AI that can accomplish a narrow task, such as image recognition. The second, general AI, is likely what many of us think of when picturing AI. This is the point at which AI is functioning in the human environment. The third, superior AI, exists in a theoretical realm where AI has surpassed human control.

I built a form of narrow AI by building a Convolutional Neural Net, CNN, which is capable of detecting and classifying different images of cats and dogs. The CNN falls into the Computer Science field of natural language processing, nlp, among other areas of study such as language translation, sentiment analysis, predictive text, and text summarizing. CNNs have had success in a variety of different industries and have had successful applications such as detecting new planets in the solar system, identifying human sentiment from facial expressions, as well as detection of abnormalities from image scans in the human body and more.

Within Machine Learning, ML, there are three classes of learning, supervised learning, unsupervised learning, and reinforcement learning. CNNs are supervised learners, meaning their learning stems from the labeled data they process. In short, CNNs take images from data and process the pixels of the image to detect similarities in the images corresponding to the values for the objects of interest. Thus, when the model processes unseen data, it can make a prediction given the features in the pixels of the images the CNN was trained on. My CNN was built to classify

different images of cats and dogs, however CNNs are capable of detecting much more.

2 Literature Review

Research in neural networks and AI have similar interests of neuroscience, as both fields seek to understand how the brain functions. Before neural networks and natural language processing, neuroscience researchers were interested in the part of the brain that allowed animals and other primates to see. In the mid 20th century, researchers Hubel and Wiesel took an interest into macaques and spider monkeys revealing the parts of the brain and processes responsible for the visual field (Hubel and Wiesel, 1968). Years of research from Hubel and Wiesel among other researchers inadvertently laid out the framework for future research in the field of computer vision. Hubel and Wiesels research of the visual cortex inspired LeCuns Convolutional Neural Net. The immense neural makeup of the human visual cortex is built off of the connections of millions of neurons in the brain that make connections and store information. The relationships of these neurons and processes that allow humans to see set the foundations of CNN architecture.

Convolutional Neural Nets were first introduced by LeCun and a team of researchers in 1998. The innovation of convolutional neural nets stemmed from the development of other neural networks prior to 1998, however convolutional neural nets were designed to recognize objects in images. LeCun and his team of researchers created LeNet, which was combined recent innovations in machine learning techniques to build a more efficient image recognition model (LeCun et al., 1998). Since the inception of convolutional neural nets, other researchers have pushed the boundaries of convolutional neural nets. Since 1998, researchers and developers have made significant strides past the realm of character recognition. Currently, ImageNet is the leading research

project dedicated to a large-scale open source image database. ImageNet hosts yearly contests, thus fostering research and development in image recognition. Since then, other researchers and developers have challenged themselves furthering development in convolutional neural nets. The architecture of convolutional neural nets can be diverse, yet they do not change drastically from model to model. Convolutional Neural Nets have seen advancements with the help of ImageNet as well as other innovations in their architecture as Zeiler and Fergus state, "(i) the availability of much larger training sets (ii) powerful GPU implementations (iii) better model regularization strategies, such as Dropout" (Zeiler and Fergus, 2013).

The main difference between any other neural net and a convolutional neural net is that not every layer is connected to the next, thus they take less memory and train quickly (LeCun et al., 1998). While a normal neural net is fully connected at each layer within the model giving weight to the next layer, CNNs take features from one layer and pass it on to the next layer where they are able to detect more abstract features along the way. Each pooling layer identifies more abstract features until the final fully connected layer offers a prediction on the image fed into it. The number of layers a CNN contains is largely dependent on the developers discretion, however popular models like AlexNet, Googles Inception, and VGG16 among others vary in the number of layers they contain, each adapted to the needs of the model. LeCun et al. built several versions of LeNet, however LeNet-4 outperformed LeNet-5 by 0.1% (LeCun et al., 1998). Their convolutional neural nets architecture consisted of the input layer, two convolutional layers, each followed by a pooling layer with three fully connected layers and their model was able to reach an error rate of 0.7% (LeCun et al., 1998).

In 2014, researchers Zeiler and Fergus, attempted to replicate the results of Krizhevsky et al. achieving results within 1% of the Krizhevsky et al. model (Zeiler and Fergus, 2013). After repli-

cating the model, Zeiler and Fergus then adjust the several parameters of the model. Zeiler and Fergus ran several experiments on what happens to a convolutional neural nets accuracy when layers are omitted. Zeiler and Fergus report, "removing the fully connected layers (6,7) only gives a slight increase in error" and "removing two of the middle convolutional layers also makes a relatively small difference", yet "removing both the middle convolutional layers and the fully connected layers yield in a model with only four layers whose performance is dramatically worse" (Zeiler and Fergus, 2013). Zeiler and Fergus also find, "increasing the size of the middle convolutional layers goes give a useful gain in performance" (Zeiler and Fergus, 2013).

Researchers Krizhevsky et al. built several convolutional neural nets. The researchers latest model consisted of five convolutional layers and three fully-connected layers (Krizhevsky, Sutskever, and Hinton, 2012). Additionally, Krizhevsky et al. included Rectified Linear Units, ReLU, as well as augmenting their data, overlapping their pooling layers, stochastic gradient descent, and incorporated a dropout layer of 0.5 (Krizhevsky, Sutskever, and Hinton, 2012). Overall, their convolutional neural net achieved a top-5 error rate of 18.2% on the ImageNet Large Scale Visual Recognition Challenge 2012, ILSVRC-2012 (Krizhevsky, Sutskever, and Hinton, 2012). Additionally, Krizhevsky et al. state, "our networks performance degrades if a single convolutional layer is removed" which is peculiar as Zeiler and Fergus model did not suffer when convolutional layers were removed (Krizhevsky, Sutskever, and Hinton, 2012).

One such innovation was applying optimization techniques such as gradient descent to these models. Gradient descent tunes models and is best described by Rumelhart et al. as, "the procedure repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector" (Rumelhart, Hinton, and Williams, 1988). LeCun et al. simply state, "back-propagation is that gradients

can be computer efficiently by propagation from the output to the input" (LeCun et al., 1998). Essentially, optimization seeks to minimize the models loss function as, "estimating the impact of small variations of the parameter values" (LeCun et al., 1998). Gradient descent adjusts the weights to minimize the loss function by taking the derivative of the loss function with respect to the parameters of the model (LeCun et al., 1998). CNNs learn using gradient based learning from back-propagation of the images they train on. Previous models used for image recognition performed poorly, however by incorporating gradient based optimization techniques, CNNs were able to make more accurate predictions.

Beyond the level of layers, various optimization techniques can impact the performance of CNN models. When CNNs were first introduced, they were limited to early versions of gradient descent, while newer models have the advantage of newer optimization techniques. The evolution of gradient descent has produced multiple variations of optimization with slight changes in the math. Innovations in gradient descent have led to improved learning times and greater accuracy in convolutional neural nets. While there are numerous optimization techniques that can be used in tuning a model, I tested: sgd, adam, rmsprop, adamax, and nadam. The varying forms of gradient descent have minor changes in their formulas allowing for varying learning rates.

Beyond the increasing availability of data, architectural changes, and optimization techniques, convolutional neural nets have seen improvement of regularization strategies. Hinton et al. claim, "large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data" (Hinton et al., 2012). Thus, Hinton et al. proposed adding in a dropout layer to convolutional neural nets which randomly omits channels in the model which they claimed, "gives big improvement on many benchmark tasks and set new records for speech and object recognition" (Hinton et al., 2012).

3 Data

The data used in the convolutional neural net consisted of 25,000 images. There were 10,000 images of dogs and 10,000 images of cats in the training data and 2,500 images of cats and 2,500 images of dogs in the testing data (Kaggle.com, 2013). As per the data from the analysis, the data collected was from different variations of the model resulting in several conclusions. The data was obtained from the different iterations of the models and compiled into csv. files for analysis.

4 Methodology

The first step for building the CNN was to load in the necessary packages. In my model, I primarily relied on pandas, NumPy, Tensorflow, and Keras. Pandas stores and handles data, NumPy functions as the math operator, and Keras is a machine learning library built on top of Tensorflow. I built the CNN in a jupyter notebook, which uses Python. I used a sequential model for my CNN. A sequential CNN can be thought of as linear in its flow. The CNN consisted of three convolutional layers, each activated by a rectified linear function, ReLU, to detect non-linear features. After each convolutional layer, I included a max pooling layer which compiles different features detected in the image. The image data is imported into the first convolutional layer then passed to a pooling layer, then another convolutional layer then passed to another pooling layer, and again to another convolutional layer, which is then passed to another pooling layer. After the convolutional layers, the model passes data into the fully connected layers. The fully connected layers require a different set of parameters which were set to: flatten, dense(64), activation(relu), dropout(0.5), dense(1), and activation(sigmoid). This series of commands constructs the 2 fully connected layers by condensing the detected features then provides a prediction from the sigmoid function. The

series of commands seem simple, they are not trivial. After setting the parameters, the last step is to compile the model. I set the loss function to binary crossentropy, set the optimizer which varied through the data collection process and added a metric for accuracy.

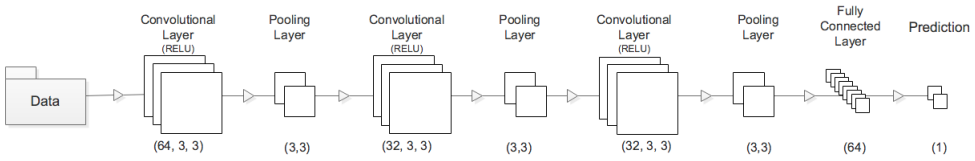

Figure 1: Model Architecture

After the CNN was built, I imported the data that was collected from Kaggle. I set the directories to flow from a training folder and testing folder within a data folder that I set up in the jupyter notebook. If done properly, the jupyter notebook should print out, "Found 20000 images belonging to 2 classes. Found 5000 images belonging to 2 classes". I then set several constraints on our images setting the desired pixel width and pixel height to 150x150. From there, I re-scaled the data from 0 to 255 for Red, Green, and Blue.

Once the model architecture is defined and the parameters are set, I defined the training strategy. First, I set the number of epochs or iterations that the model will train for. Then I set the input size of our training and testing images and describe the steps per epoch. After the parameters are set, we can then train our neural network and evaluate its efficiency. The sample models that I used to test the different parameters consisted of 2000 images in the training set and 400 images in the validation set. Additionally, each model was set to 10 epochs, with 125 steps per epoch. These constraints were set due to a constraint on time as I was using a 2017 MacBook Pro with an Intel core i5 processor to train the model.

The experiments that I used to collect my data resulted from: (i) different optimization tech-

niques; (ii) varying dropout layers; (iii) varying the amount fully-connected layers and (iv) the models accuracy after data augmentation.

The optimizers used to analyze my CNN are different variations of gradient decent. I optimize my model to determine the weights of the hidden layer in our model. Weights too big or too small will have non-optimal weights leading to a large margin of error from our predicted value and our expected value.

5 Findings

The results are listed in Tables 1-4.

Each experiment was designed to provide insight to the best parameters of the model. As the training time took a considerable amount of time, the experiments were run on smaller amounts of data. After determining the best parameters, I chose to finely tune the model and let it train for 30 epochs and then augmenting the data for an additional learning step increasing the models accuracy.

After running numerous models, the varying parameters of each model resulted in the following findings. The first experiment indicates that the best optimizer was Adam, adaptive moment estimation. The data collected is presented in the tables following the paper. As for the second experiment, I found that the models with a dropout layer of 0.5 and 0.000001 reached nearly the same accuracy. However, for any valuable conclusion to be made, I need to research this further. Since dropout is meant to regularize the model, the reported accuracy may not perform as well with the testing data. I strongly recommend that dropout be set to 0.5 according to my findings and the preceding literature review. The third experiment, tested the impact of varying fully connected

layers after the convolutional layers. The model that performed the best consisted of two fully connected layers and one dropout layer and the worst performing model had five fully connected layers and four dropout layers. I find that too many fully connected layers with a dropout layer set to 0.5 weakens the models performance. The fourth and final experiment was withheld until the prior experiments were performed, as the literature review led me to believe that data augmentation was likely to increase the models accuracy.

After the previous experiments, I built a final model which trained over the full 20,000 training images and tested on 5,000 test images for 30 epochs set to 1250 steps per epoch. I set the parameters of the final model according to the best performing results in the experiments. I used adaptive moment estimation, Adam, for my optimizer, and used two fully connected layers with one dropout layer set to 0.5. The model results in a near 76.4% accuracy after the first epoch, 86.4% accuracy by the 10th epoch, 86.2% accuracy by the 20th epoch, and 87.5% accuracy by the 30th and final epoch. As the model iterates through different images in each epoch, the model learns and is able to classify images with greater accuracy.

Without any data augmentation and training on the stock photos of cats and dogs, the models accuracy was 87.5%. After completing a run on the training images, the next step was to augment the data. The images were horizontally flipped, and their angles were adjusted so that the model could learn different patterns for dogs and cats. After the data augmentation, the model reaches an accuracy of 91.5%.

6 Conclusion

Convolutional Neural Nets have come a long way since their inception in 1998 and are now being used in a variety of commercial applications. The use case of large scale image detection beyond current use cases, requires more research and development. If researchers and developers continue to work on CNNs, we will see strides in new applications of this technology and they will be able to recognize more objects of the human environment. Further innovation will improve CNN performance and keeping them open-source will allow new industries to adopt and harness the power of CNNs allowing for wider application and further exploration of computer vision.

References

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *CoRR* abs/1207.0580. URL `http://arxiv.org/abs/1207.0580`.

Hubel, David H. and Torsten N. Wiesel. 1968. "Receptive Fields and Functional Architecture of Monkey Striate Cortex." *Journal of Physiology (London)* 195:215–243.

Kaggle.com. 2013. "Dogs vs. Cats." `https://www.kaggle.com/c/dogs-vs-cats`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 1097–1105. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86 (11):2278–2324.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1988. "Neurocomputing: Foundations of Research." chap. Learning Representations by Back-propagating Errors. Cambridge, MA, USA: MIT Press, 696–699. URL `http://dl.acm.org/citation.cfm?id=65669.104451`.

Zeiler, Matthew D. and Rob Fergus. 2013. "Visualizing and Understanding Convolutional Networks." *CoRR* abs/1311.2901. URL `http://arxiv.org/abs/1311.2901`.

Optimizer	Accuracy
rmsprop	0.72631744
sgd	0.630703011
adam	0.738472396
adamax	0.716436637
nadam	0.731649937
adagrad	0.720671267

Table 1: Accuracy By Optimizer

Fully Connected Layers	Dropout Layers	Accuracy
2	0	0.722474906
5	4	0.4981179422835634
2	1	0.738472396

Table 2: Using Adam

Dropout	Accuracy
0.5	0.7206712672521958
0.00001	0.7527446675031367

Table 3: Using Adagrad

Model Type	Accuracy
Training	0.974
Testing	0.875
Augmented	0.9111

Table 4: Model Accuracy