Visual Recognition with Deep Neural Networks

Final Report

UY, Mikaela Angelina Chan
mauy@connect.ust.hk

Abstract

This project aims to be able to develop a model that can recognize images, specifically, cats and dogs. The goal is to have an accurate model that can classify cat and dog images on a Kaggle challenge using the training images provided. A deep learning approach using Tensorflow was used to achieve this task, which was incorporated with image processing techniques in order to achieve higher accuracy. Various deep learning architectures were studied and tested in order to achieve a model with the best possible performance in the Kaggle challenge. Models were all trained from scratch using the 25,000 training images provided. The best performing model was a deep network that had 6 convolutional and 2 fully-connected hidden layers, and it achieved a loss of 0.3355 in the public leaderboard, which places at rank 874 out of 1314.

1. Introduction

When humans look at any given image, they can instantly classify the objects it contains. They can recognize images into multiple classes from people, animals, plants and things. The human brain is able to recognize objects in images accurately, even with object obstructions and various orientation and illumination variances. Accurate algorithms for object classification would allow computers to do the same, which would lead to various autonomous systems such as self driving cars and robot assistants.

Deep learning enables computers to mimic the thinking process of the human brain. Deep learning models have multiple layers of various architectures, and the idea is each layer would extract relevant features that would aid in classifying the image. These layers include convolution layers, fully connected layers, activation layers, etc., each containing weights that will be tuned during the successive iterations of model training.

In each of the iterations, a number of training images (specified by the batch size) are passed through the model and the loss between the predicted and expected output is obtained. Using the obtained loss, the weights in each layer are modified or tuned in a process called backpropagation, which aims to minimize future losses. Hence, the model "learns" using the deep network architecture through multiple iterations, and thus resulting in a trained model.

The goal of this project is to be able to design and train a network that can accurately classify cat and dog images in the Kaggle competition. Tensorflow is the library that is being used with GPU acceleration, and OpenCV is also used for image processing to aid the model training. The hardware used is a Lenevo Ideapad Y700 with 16GB of RAM and a NVIDIA GTX960M graphics card with 4GB of VRAM.

For learning purposes, it was decided that the model was to be trained from scratch instead of using transfer learning, which means fine-tuning the last few layers of an existing pre-trained network. Two model architectures were tested: one smaller network and one bigger/denser network. Proper data set-up, image pre-processing and weight initializations were done, and moreover, different optimizers, batch sizes, dropout rates and number of epochs were studied. All of these will be further discussed in the sections to follow.

2. Data set-up

To achieve a successfully trained model, proper data handling was needed, which includes handling the transfer of data from disk to memory and handling the split of training and validation sets.

The first issue needed to be solved was how to utilize the 25,000 training images to be able to train over all of them. Due to the lack of memory, not all 25,000 images could be read from the disk at once. Hence they had to be read in "loading batches", and each time a batch is loaded, it had to have an equal number of cat and dog images in order to prevent classification bias. Initially, a fix set of 1000 dog and 1000 cat images were used for training over multiple epochs and iterations. However, this does not utilize the whole set of training images which inhibits the model from achieving better robustness and accuracy. Thus, the improved implementation read different loading batches of images on the fly, each with 625 dog and 625 cat images, and hence each epoch would include training the model in 25,000/1250 = 20 different loading batches.

In each loading batch, the 1250 images were split into a training set and a validation set. The model was trained in successive iterations, each containing a certain number of images (specified by batch size) from the training set, and the training loss was recorded in each iteration. The validation set was used to access the current performance of the model by calculating the validation loss and accuracy, which was done approximately once every 20 training iterations. Training was stopped whenever the training loss was significantly lower than the validation loss as this signaled overfitting.

Additionally, the list of training images were shuffled after every epoch in order to generate different sets of loading batches each time. Moreover, the validation set was also randomly selected from the loading batch each time, and this mimics a similar idea to cross validation. It contained 240 and 50 images in the smaller and denser networks, respectively, due to limits in memory.

3. Image pre-processing

The training dataset included dog and cat images of different sizes, styles and brightness. Thus, proper pre-processing was needed to be done before training started. All the images were resized to 224x224 pixels because it is the usual size used in ImageNet. The original aspect ratio was preserved and excess borders were padded black.

Additionally, as recommended in Stanford's Visual Recognition course (CS231n), mean subtraction was done in order to center the data about the origin. This was done separately across the 3 color channels. After mean subtraction, the values were also dived by pixel depth, which is equal to 255.

The first models trained did not use mean subtraction to process the images, which was later concluded to be an essential process that improved the performance of the trained model.

4. Model Architecture

Stanford's Visual Recognition (CS231n) lecture notes were thoroughly studied before designing the model architecture. For this project, two different architectures were used and studied, a smaller network and a denser network.

The smaller network had architecture as follows:

INPUT →[CONV→RELU→POOL]*3→FC→ RELU→FC

The denser network had the architecture shown below:

INPUT→[CONV→RELU→CONV→RELU→POOL]*3→ [FC→RELU]*2→FC

Here, CONV, RELU, POOL and FC refer to a convolutional layer, rectified linear unit, max pooling layer and fully-connected layer, respectively.

Both of these architectures are similar to the common network patterns discussed in CS231n, which is why these are the architectures that were chosen to be tested and studied.

5. Discussion

It was hypothesized that the denser architecture would significantly outperform the smaller network as more complex features could be extracted from a more sophisticated model. The denser network did perform better than the smaller, however, the difference in performance was not as significant as initially hypothesized. Moreover, it took significantly more time to train the denser network compared to the smaller one as there are more parameters involved.

Moreover, the rectified linear unit (RELU) was used as the activation function throughout the networks because it is known to outperform other activation functions such as sigmoid and tanh. The activation function is defined as f(x)=max(0,x), which provides non-linearity to the model.

Other factors that were studied during the process of model training were optimizers, dropout, batch size and number of epochs, which will all be discussed in the sections to follow.

5.1. Optimizers

Two optimizers were studied, namely the RMSProp Optimizer and the Adam Optimizer. According to Ruder[3], the Adam is the one to use as it adds momentum and bias correction to RMSProp, while RMSProp well divides the learning rate by exponentially decaying average of squared gradients. Both were tested on the small dataset of a 1000 dog and 1000 cat images on the smaller architecture to see their performance, and the accuracy on 400 validation images were recorded across multiple iterations as shown below.

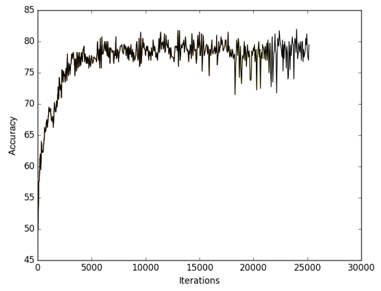


Figure 1: Accuracy of the validation set using the Adam Optimizer

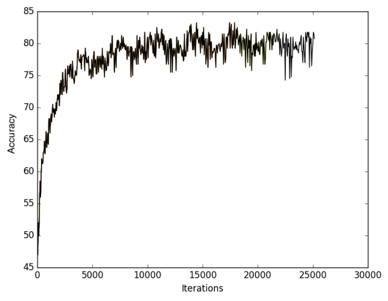


Figure 2: Accuracy of the validation set using the RMSProp Optimizer

It is seen that the performance of both optimizers are similar with the RMSProp Optimizer performing slightly better. Thus, the RMSProp was decided to be used for this study.

5.2. Dropout

Dropout is used in order to prevent overfitting the model to the training images being fed in each iteration. It was initially studied whether dropout does indeed help in the overall performance of the model, and after running the same model with and without the implementation of dropout, it was concluded that dropout does improve the final performance, which was done by submitting two different sets of predictions to Kaggle with the prediction trained by the model implementing dropout achieving a lower loss.

Dropout essentially is a form regularization as it constraints the network from highly depending on certain nodes. This works by switching off some of network nodes during model training, and thus making the learned weights more insensitive to the weights of the other nodes. The switching off of the nodes is determined by an input parameter called "keep probability".

For the models used in this project, a keep probability of 0.5 and 0.9 were used during dropout after the fully-connected and convolutional layers, respectively.

However, because some nodes are randomly dropped during training, the process will take longer as only a fraction of the nodes learn during each iteration. This phenomena is clearly shown in the graphs below.

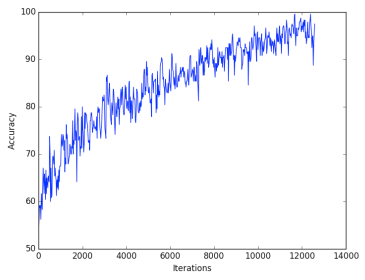


Figure 3: Validation accuracy across training iterations of a model not implementing dropout.

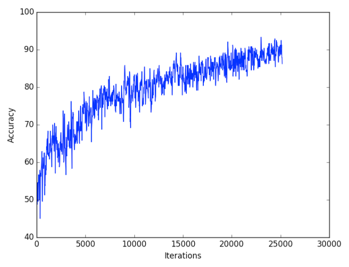


Figure 4: Validation accuracy across training iterations of a model implementing dropout.

As shown on the graphs above, the accuracy of the validation set already peaked to around 90% by the 12000th iteration, however, the model with dropout did not reach the same accuracy until the 25000th iteration. These graphs were constructed during the training of the smaller architecture. Hence, it was concluded that models, which implement dropout, converge slower that models that do not.

It is also important to mention that dropout is only used during model training but not during testing nor during validating. That is, all nodes must be used in making predictions with the final model.

5.3 Batch size

Batch size refers to the number of images that is fed into the model per iteration. This means that in each iteration, the model is minimizing the average loss garnered by the images in the batch. For example, if the batch size were 1, then the model would correct itself based on the loss of that one image, whereas if the batch size were 64, then the model would correct itself in the direction that minimizes the average error of the 64 images. Thus naturally, it is better to have a larger batch size, as the learning of model would be more stable. However, a larger batch size would lead to a longer processing time for each iteration. Moreover, batch size is also restrained by the memory of the system. For this project batch sizes of 16 and 32 were used depending on the model's demand on memory.

The graphs below show that larger batch size achieves a smoother learning curve. The first and second graphs show the learning curves of the smaller network using batch sizes of 8 and 16, respectively.

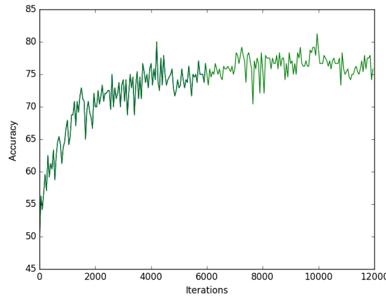


Figure 5: Learning curve of a model with batch size of 8.

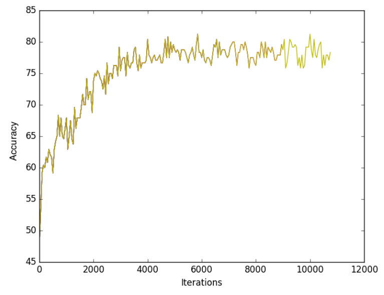


Figure 6: Learning curve of a model with batch size of 16.

5.4 Epochs

The number of epochs refers to the number of times the model is trained across the whole training dataset. Thus, the longer the training process is, the more the model learns about the training dataset, which leads to a lower training loss since at every iteration in each epoch, the model is changed in the direction that minimizes the training loss.

A problem arises with when the model is trained over too many epochs that leads it to overfitting to the training data. This means that the model performs exceptionally well on the images it has trained on, but it fails to generalize to images it has not encountered before. Thus, training has to be stopped before overfitting happens, and a signal of overfitting is when the training loss is significantly lower than the validation loss. Ideally, the desired outcome is a slightly lower training loss compared to validation loss. For this project, both training and validation losses were graphed in order to check for overfitting. Below are graphs produced by an overfitted model, which as predicted did not perform well on the test dataset.

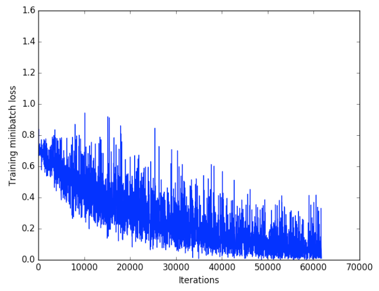


Figure 7: Training loss of an overfitted model

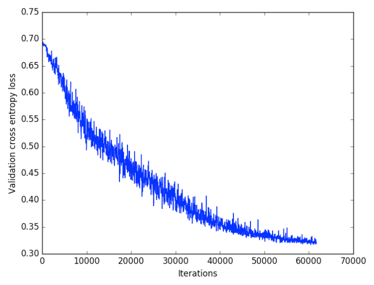


Figure 8: Validation loss of an overfitted model

As shown on the graphs above, the training loss of the model was already very close to 0.0 while the validation loss was plateauing at around 0.33. This showed signs of over fitting, and when it was run on the test set, it generated an error of around 0.70.

Additionally, larger networks and networks implementing dropouts were run over more epochs as they are less prone to overfitting and take more iterations to optimize. For this project, the smaller network was trained on over 15 epochs while the larger network was trained on over 25-30 epochs.

6. Results: Best Model

The model that achieved the best result is the network with a denser architecture. Three sets of two-stacks of CONV-RELU each followed by a max-pooling layer were first done. This totaled to 6 convolutional layers all of which used a 3x3 convolution kernel. A 2x2 max-pooling layer followed every two convolutional layers. After this, the resulting tensor was then flattened and fed into two fully-connected layers with output dimensions of 1024 and 64. And finally it was flattened into a 2-dimensional output tensor contained the probabilities of it being a cat and a dog.

The weights of these hidden layers were initialized through a normal distribution with a mean of 0 and a standard deviation of 0.1.

In order to optimize the parameters on these hidden layers in each iteration, a loss function had to first be defined, and the softmax cross-entropy loss was used, which is also the loss that is being minimized in the Kaggle competition.

The graphs below show the training and validation loss of the model after about 20,000 iterations.

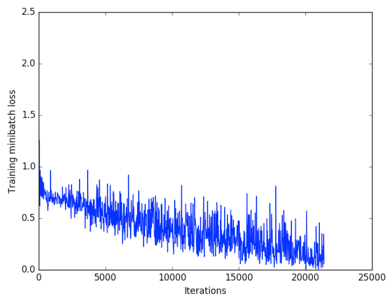


Figure 9: Training loss of the best-trained dense network

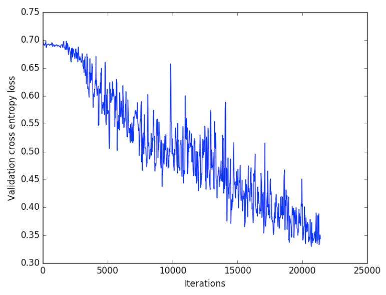
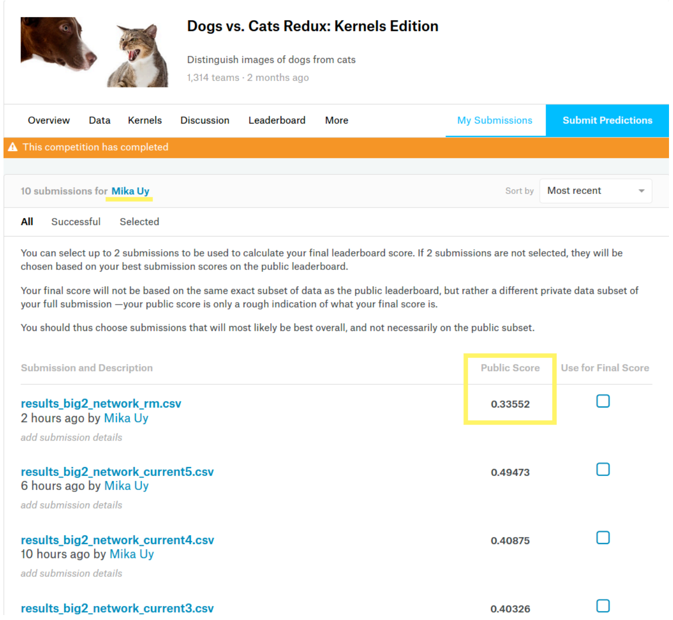


Figure 10: Validation loss of the best-trained dense network

As seen, the training error is averaging at around 0.25 while the validation loss is at about 0.35. It can be concluded that the model has not yet overfitted itself to the training data. It was then run over a few more epochs, which resulted in a final model that achieved a test loss of 0.3355 on Kaggle's public leaderboard that places it at rank 874. The reported loss is shown in the screenshot as seen below.



Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · 2 months ago

This competition has completed

10 submissions for Mika Uy

Sort by Most recent

All Successful Selected

You can select up to 2 submissions to be used to calculate your final leaderboard score. If 2 submissions are not selected, they will be chosen based on your best submission scores on the public leaderboard.

Your final score will not be based on the same exact subset of data as the public leaderboard, but rather a different private data subset of your full submission —your public score is only a rough indication of what your final score is.

You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.

Submission and Description	Public Score	Use for Final Score
results_big2_network_rm.csv 2 hours ago by Mika Uy add submission details	0.33552	☐
results_big2_network_current5.csv 6 hours ago by Mika Uy add submission details	0.49473	☐
results_big2_network_current4.csv 10 hours ago by Mika Uy add submission details	0.40875	☐
results_big2_network_current3.csv	0.40326	☐

Figure 11: Screenshot of Kaggle public score.

7. Code Implementation

An initial tensorflow starter kit template downloaded from the kernels publicly available in the Kaggle competition. This kernel used 150x150 input images that is trained over a fixed set of 1000 cat and 1000 dog images with a fixed split of 1600 training and 400 validation images. The network architecture of this initial model is similar to the smaller network architecture discussed in this report except that it used smaller input images, which made the dimensions of some layers different and it did not implement dropout. Image pre-processing that was done in this initial template was a simple normalization that scaled down pixel values in each color channel from [0, 255] to [-0.5, 0.5]. And the accuracy achieved by this model is around 65%.

The rest of the code implementation was written by the student, herself. This includes taking in images of size 224x224 instead of 150x150, building the architecture of the denser network, mean subtraction for image pre-processing, implementing dropout during training, reading different loading batches, random splitting of training and validation sets, plotting graphs, etc. Separate scripts were also written for model training and the generation of test predictions in order to control the amount of csv files written. Trained models were also outputted and saved so that it can be loaded on the fly for future use.

8. Future Improvements

For better performance and higher ranked models, more sophisticated architectures can be further studied. Combinations of different approaches can be looked into to achieve more successful feature extraction methods, as input images are not consistent. Moreover, the model can be trained on a more powerful machine with better memory so larger batch sizes can be used.

References

[1] Stanford University. CS231n: Convolutional Neural Networks for Visual Recognition, 2017. Retrieved from http://cs231n.stanford.edu

[2] Stanford University. CS20SI: Tensorflow for Deep Learning Research, 2017. Retrieved from http://web.stanford.edu/class/cs20si/

[3] Ruder, Sebastian. An overview on gradient optimization algorithms, 2016. Retrieved from http://sebastianruder.com/optimizing-gradient-descent/

[4] Tensorflow Tutorial, 2017. Retrieved from https://www.tensorflow.org/tutorials/

[5] McCaffrey, James D. Why you should use cross-entropy error instead of classification error or mean squared error for neural network classification training, 2013. Retrieved from https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/

[6] Loher, Phillipe. TensorFlow Starter Kit (Fixed!), 2017. Retrieved from https://www.kaggle.com/kbhits/tensorflow-starter-kit-fixed