|  |  |
| --- | --- |
| **Capstone Project**  **Machine Learning Engineer Nanodegree** | Priyanka Dwivedi  September 2016 |

**Definition**

**Project Overview**

The project I chose is an active Kaggle competition called Dogs vs Cats.

<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition>

The goal of the project is to use supervised learning to be able to distinguish images of dogs from cats. The training dataset for this project is a set of 25000 images that have been labelled as either a cat or a dog. The dataset available on Kaggle has 12,500 images for cats and 12,500 images for dogs. The images are very real life with photos having varying degree of lightening, other objects/people in the dataset along with cats or dogs and the animals are not always centered in the image.

The problem domain for this exercise is **computer vision** and **deep learning**. I am excited about this project as it gives me the chance to explore a field that is up and coming and out of my comfort zone. My background is in consumer finance and this gives me a chance to learn something new!

To succeed in this project I have leveraged several sources to build my understanding of Deep learning and also learn about implementation using TensorFlow. These sources include:

* Udacity Course: [Deep Learning](https://www.google.com/url?q=https://www.udacity.com/course/deep-learning--ud730&sa=D&ust=1473187237687000&usg=AFQjCNEh1l5QuBAKB9jBxl7CM_kpkv_tpw) by Vincent Vanhoucke (Google and Udacity).
* TensorFlow and TFLearn

<http://tflearn.org/getting_started>

<https://www.tensorflow.org/versions/r0.10/tutorials/index.html>

* Books on Deep Learning
  + Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks by Jeff Heaton
  + Getting started with TensorFlow by Giancarlo Zaccone
* [CS231n: Convolutional Neural Networks for Visual Recognition](https://cs231n.github.io/) — Andrej Karpathy's Stanford CS class

<http://cs231n.github.io/>

* Research papers on latest techniques in convolution neural networks
  + http://arxiv.org/pdf/1412.6806.pdf

**Problem Statement**

The problem here is to use Supervised Learning to train a computer to distinguish between images of dogs and cats. We have a dataset of 25000 images for cats and dogs with labels to accomplish this. Once the computer is trained, we have a test dataset of 12,500 mixed images for which a label (cat or dog) needs to be provided. The success of the project will depend on our accuracy in classifying cat or dog images in the test dataset.

This problem is a classic case of a computer vision challenge. The images are not centered and there are often other objects on the image. Due to variation in scale, rotation and noise between images, my approach would be to train a **deep convolution neural network (CNN)** implemented using TensorFlow to teach the computer to analyze various aspects of the image. My goal is to start with a simple classic implementation of CNN with several layers of convolution and max pooling. And gradually improve on this basic model by adding dropout, regularization and learning rate decay as well as optimizing hyper parameters and training parameters.

One challenge that I will likely encounter is processing power. I don’t have a GPU computer and I am not sure if my Mac with CPU will have enough processing power. If I do need access to GPU, then I will leverage Amazon Web Services, Elastic Cloud 2 GPU for training the model. This will be new for me and another learning experience. The challenge here is to approach the problem in a computationally efficient manner

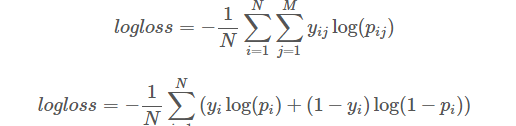
**Metrics**

For this project we have two datasets:

* Train dataset of 25,000 images with labels provided for each image
* Test dataset of 12,500 images. **No labels provided**. After the model has been trained, the test set will be scored on the best model and the output submitted to Kaggle. Kaggle will provide a log loss estimate on the submission

My plan is to divide the train dataset into 3 different sets – 1.Ttraining dataset of 20,000 images that will be used to train the model, 2. Validation dataset of 4,000 images for testing the performance of the CNN and 3. A test set of 1000 images that will be used to do a final evaluation before submission. The goal of creating this test set is that as the CNN is repeatedly trained and tested on validation set, it will eventually “see” the validation set. This test set will be a more objective evaluation of the accuracy of CNN on an out of sample. I will be using two metrics for measuring the success of the project:

1. Log Loss function – The goal of the convolution optimizer will be to minimum the cross entropy loss which is the same as log loss. The selected model would be the one with the lowest log loss on the validation and test sets. This is also the metric that will be used by Kaggle to rank contestants in this competition. The calculation of log loss is below:



Log loss will be computed on training and validation sets before submission. The Log loss for testing dataset will be provided by Kaggle.

2. Accuracy Score – Defined as % labels correctly classified when comparing model prediction vs actual. This metric will be computed on the training dataset and the validation set which is the population for which we have labels for the images. This is the secondary metric that will be computed and shared but the select of the optimal model will be based on minimizing log loss. It is expected that the model with the lowest log loss will also have one of the best accuracy score.

**Analysis**

**Data Exploration and Exploratory Visualization**

The data for this project can be found on Kaggle website and is a zip file of 25,000 images for training for which cat or dog label has been provided and a zip file of 12,500 images for testing with no labels for submission to the competition.

<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>

Each image in the training has the format “cat.250.jpg” which includes a label for cat or dog and an image sequence number. The data includes real images of cats and dogs. A few images are shared below:

Cats

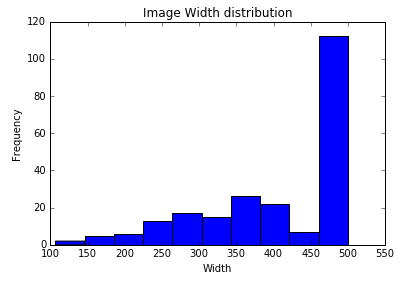
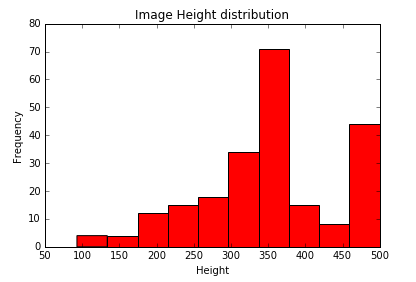
Dogs

Here are a few characteristics of the dataset that make this classification exercise challenging:

1. Presence of people and other objects in the image
2. Some images have multiple animals
3. Images are shot under different amount of light
4. Some images are blurry
5. Animals are not always centered in the image
6. Varying amount of color in the images
7. Images have different height and width

The histogram below takes a random sample of 225 images from the dataset and plots image width and height

We can observe that images have different width and heights. Most pictures have width around 500 pixels and height varies b/w 300-500 pixels.

I think this is a first challenge to solve for. I would like to

1. Have each picture be reduced to a uniform width and height
2. The images are big. To deal with processing efficiently, images should be smaller. Each pixel will be an input into a neural network so large images would require substantial computing power. Example each 56x56 gray image would be convereted to an array of 56x56x1 pixels = 3136 pixels. If instead we go with a colored image of 128x128 then it will have 128x128x3 = 49152 pixels i.e each image will have a dataset that’s 15 times bigger. Multiplying that with 25k images and we can see how this can quickly become a massive dataset
3. I would like to start with a dataset of all 25000 images resized to 56x56 pixels and reduced to grayscale.

**Algorithms and Techniques**

Given the nature of this problem – images with difference in scale, rotation and noise, I think a deep convoluted neural network is the best algorithm to choose for this problem.   Convolutional neural networks are biologically inspired variants of multilayer perceptrons (MLP), designed to emulate the behaviour of a visual cortex. These models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images Also all neurons in a given convolutional layer share the same weights and detect exactly the same thing. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting the property of translation invariance. Also sharing weights between neurons makes it computationally efficient.

Here is the approach I have in mind:

1. Start with a basic convolution neural network structure – A Convolution Layer with RELU activation function followed by a max pool layer. Multiple such convolution layers will be connected. Finally have one fully connected layer before the output layer. Stochastic gradient descent will be used to optimize this model.

Input Layer -> Conv Layer 1 with RELU -> Max Pool Layer 1 -> Conv Layer 2 with RELU -> Max Pool Layer 2 -> Fully Connected Layer with RELU -> Output Layer

Add regularization to the basic convolution neural network defined in step 1 above to solve for the problem of over fitting. The regularization techniques I will use are dropout and L2 regularization. I would also like to explore learning rate decay to further tune stochastic gradient descent optimization

1. Once a basic convolution neural network is set up, then I will move to the important task of tuning it to the dataset to improve on the results. The following will be done to further tune the basic model:
   1. Change in layer structure of the neural network – Adding more convolution and fully connected layers, replacing max pool layer with a convolution layer etc.
   2. Tuning the no of neurons in the all the different layers
   3. Optimization of regularization techniques – L2 regularization on just fully connected layer vs all layers
   4. Selecting the best optimization function – SGD, Adam etc
   5. Tuning the learning rate in the optimization function
   6. Image augmentation
   7. Increasing no of training epochs
   8. Using color images instead of grayscale images

**Benchmark**

Kaggle has defined a loss rate of **0.5** as the benchmark performance of this competition. Once the model is trained, I will submit my predictions on the test dataset and share the performance on the test dataset with Udacity.

**Methodology**

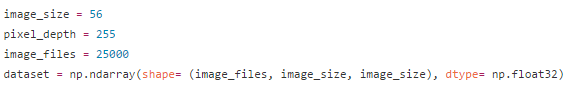
**Data Preprocessing**

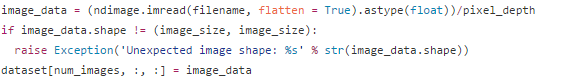
Here are the image processing steps that were undertaken:

1. Resize the images to similar width and height and reduce the size. All images were resized to 56x56 pixels. PIL Image library was used to do the resize and save images as Jpeg. Sample code below:

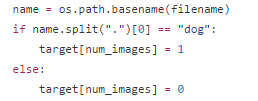


2. Use scipy ndimage to read the image files as a dataset. To begin with all images were read as grayscale images (Flatten = True). Also the resulting dataset was divided by pixel depth = 255 to scale it b/w 0 -1. Sample code below:

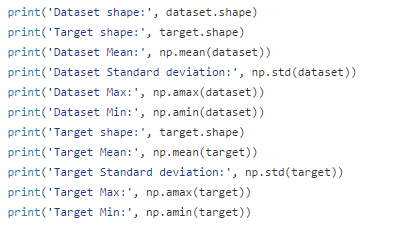




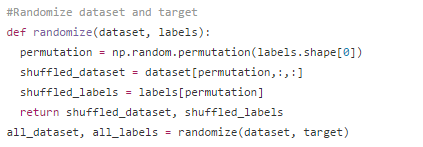
4. Create a dataset for labels. The Kaggle dataset has a jpeg labelled as either cat or dog. The target was extracted from the file name



At this stage sample statistics were calculated on the dataset and target set including shape, mean, standard deviation, min and max. See code below:



5. Shuffle the cat and dog images within the dataset since the pictures were read as first all cat pics and then all dog pics

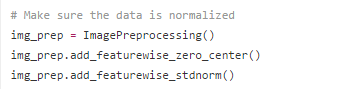


6. Use sklearn cross validation to split the overall dataset into 3 parts – 1. Train set of 20k images, 2. Validation set of 4k images and 3. Testing set of 1k images. This was done in two steps:





7. Before feeding this dataset into the convolution neural network. The dataset was normalized by mean and standard deviation evaluated over all the samples. This was implemented using TFLearn preprocessing module. See sample code below;



8. Reshape the dataset and labels into the proper format before feeding into Tensorflow. The dataset was reshape as a 4D tensor and the labels as two columns – one for cat and another for dog. See sample code below:





No further pre-processing was done for images since convolution neural networks work best when input images are kept close to natural state and the model is allowed to extract all relevant features. I did explore adding ZCA whitening as a preprocessing step however that reduced the log loss of the final model and was dropped. The full code for preprocessing can be found on my GitHub repository:

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/create_new_pickle_4.py>

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/catvsdog_vanilla.py>

**Implementation**

I started with implementing a simple convolution neural network in TFLearn. This neural network had the following structure

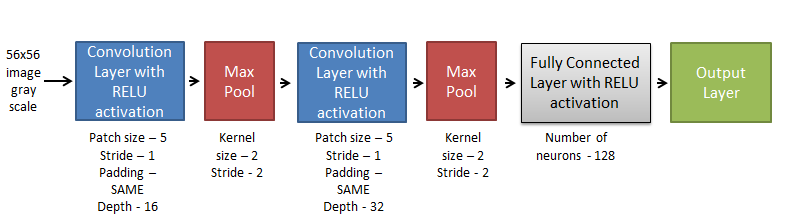


Figure 1: Simple Convolution Neural Network

The algorithm was created using TFLearn tutorial on their website

<http://tflearn.org/getting_started/>

Batch size of 96 neurons was used to train the model. I found the model very slow to train on my Mac CPU so I used Amazon Web Service Elastic Cloud 2 GPU 2.2x Large to train this model. After experimenting with several parameters and running the model for 15 epochs, I got the following results:

**Best Validation Accuracy Score: 78.78%**

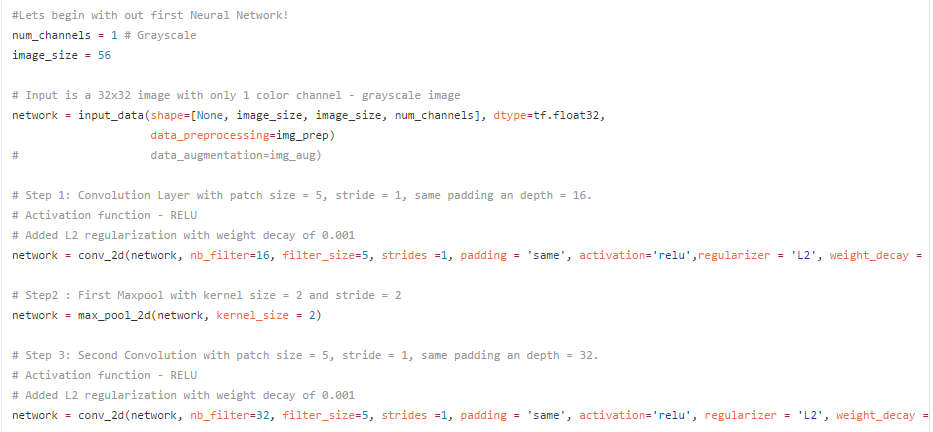
**Valid Log Loss: 0.5487**

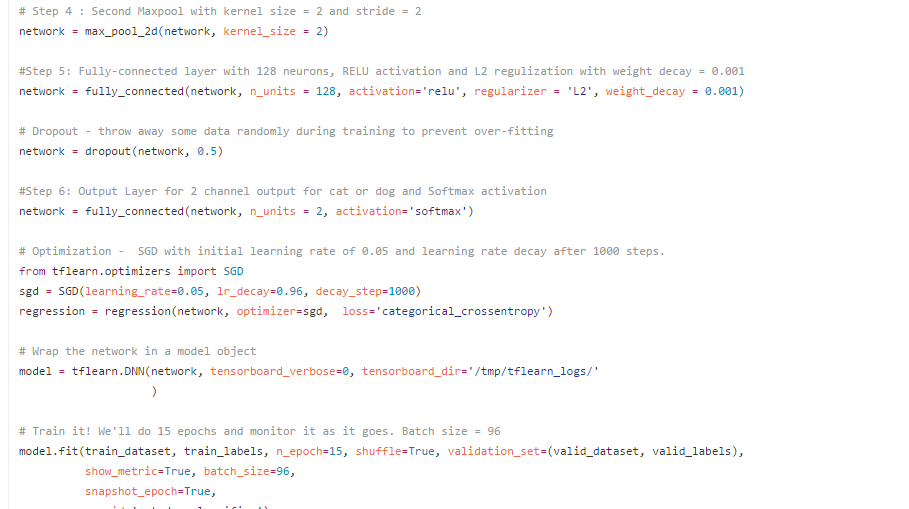
**Test Log Loss: 0.5306**

The full code for this model is shared at my GitHub repository:

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/catvsdog_vanilla.py>

Sample code is shared below:





**Refinement**

Bulk of my time in this project was spent on this stage – tuning of hyper parameters and training parameters. This section summarizes all the different steps I took and the results for the same. Use of Amazon EC2 GPU really helped at this stage in being able to experiment with different parameters efficiently. The following adjustments were tried to further refine the model:

1. Change in layer structure of the neural network – Adding more convolution and fully connected layers, placement of max pool layer
2. Tuning of neural network parameters – patch size and depth of convolution layers and number of neurons in the fully connected layer
3. Optimization of regularization techniques – L2 regularization on just fully connected layer vs all layers
4. Selecting the best optimization function – Stochastic Gradient Descent or Adam Optimizer
5. Tuning the learning rate in the optimization function
6. Image augmentation – Use of rotation, blur, flip left to right and crop to augment the training set
7. Increasing no of training epochs
8. Using color images instead of grayscale images

The above techniques were selected as I felt that they would influence the model output. I used my learning from the books, courses and research papers that I read for this project to select this list. The table below summarizes the key adjustments made and results of the same.

You can also see the excel file for this at my GitHub repository:

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/refinement_results.xlsx>

Table 1 – Results from refinement of model



The table covers most of the important adjustments to the model. The green cell highlights what was changed in that refinement from the row above. The 3 columns on the right summarize the results of the changes on the log loss and accuracy of the model. Please note the shorthand: conv – Convolution Layer, MP – Max Pool Layer and FC – Fully connected layer.

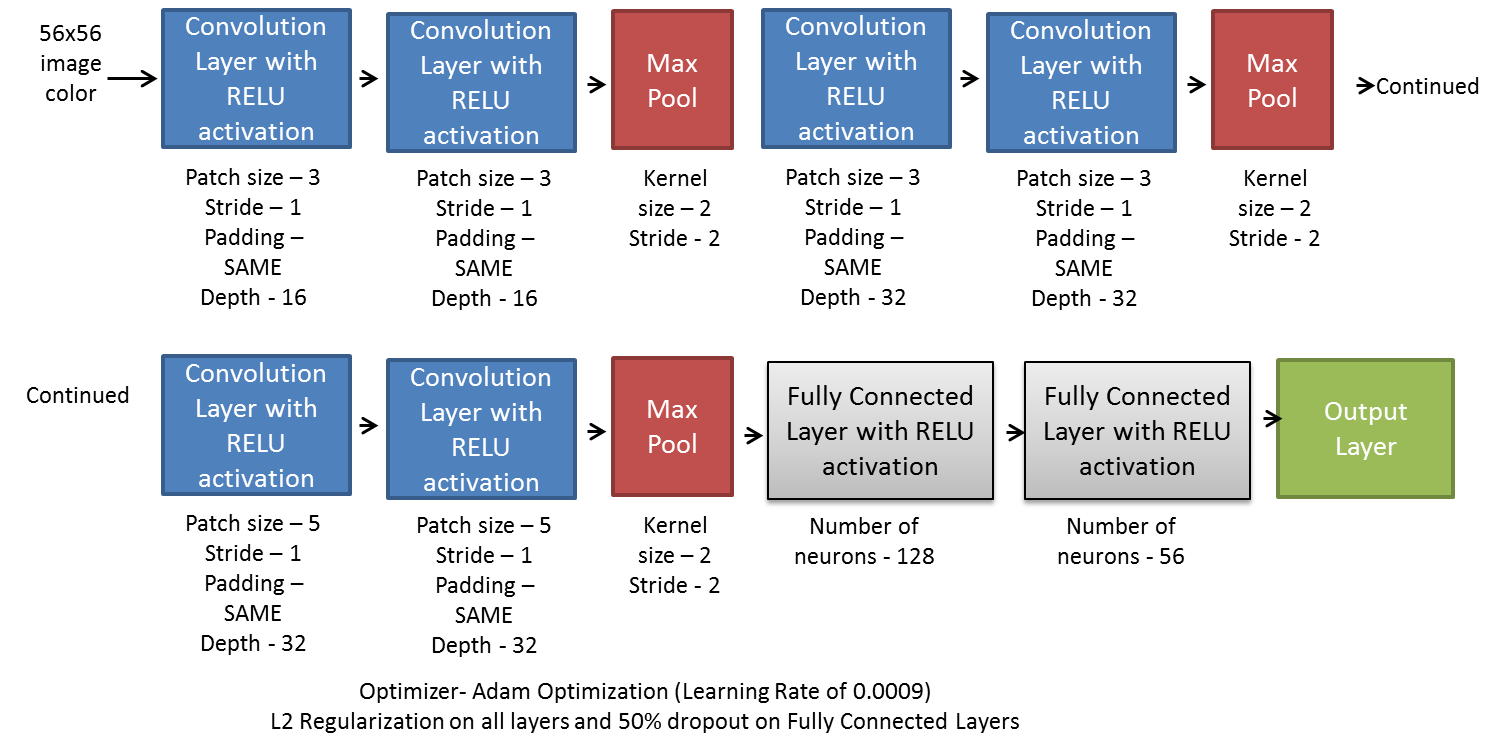
The initial model is highlighted in the top row of the table and the final model was the refinement 16 highlighted in yellow. We can see significant improvements in performance between initial and final model.



**Results**

**Model Evaluation and Validation**

The figure below shows the structure of the final model:



The code for this model is saved on my GitHub repository

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/final_code.py>

This model was chosen as the final model for several reasons:

1. I tried all the techniques I possibly could to refine the model. The final output seems reasonable in terms of parameters and layer structure.

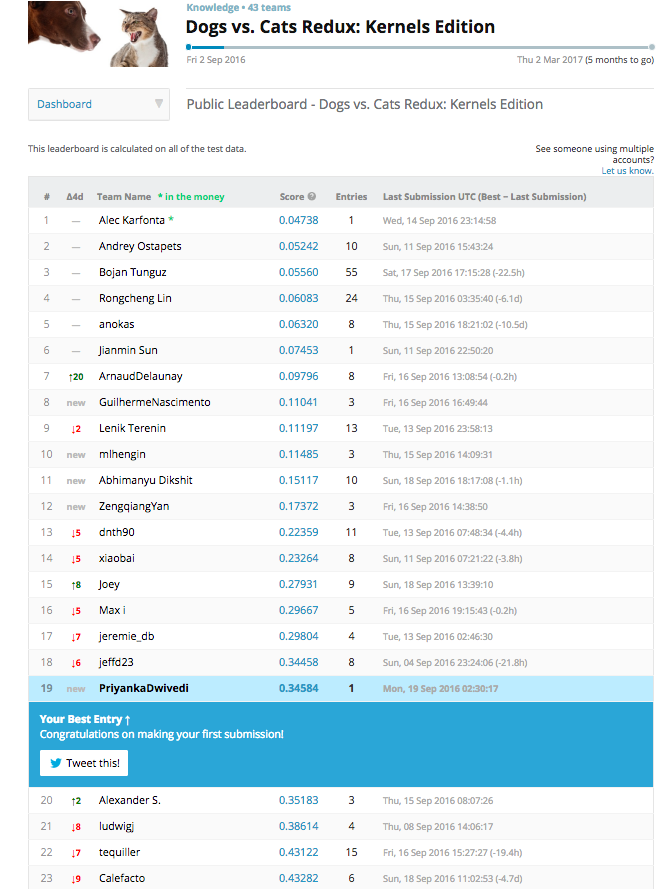
2. As can be seen from table 1, the incremental benefit of changes to the model was very small after a point. So theoretically refinement can continue but in my opinion the effect would be small and not worth the time

3. Log loss is lower than the benchmark established in the beginning (0.5) and the model accuracy score of 85% on validation set is reasonable

To evaluate the robustness of this model on data never seen before, I used the test set on Kaggle website to make a submission. The test set is a dataset of 12.500 images for which a label has not been provided. The trained model was used to score this dataset and the results were submitted to Kaggle. The code for this part is also on my GitHub repository:

<https://github.com/priya-dwivedi/Udacity-ML-Nano-Degree/blob/master/P5/kaggle_test_pred.py>

The Log loss on the test dataset at Kaggle was **0.3458.** This is in the ballpark we had with the final model on our validation and test sample. As expected performance is slightly weaker on the sample never seen before however the model seems robust. The screen shot below shows performance on Kaggle dataset.



You can also see the latest leaderboard on Kaggle to verify the same (ranking would fluctuate over time as the competition is still open).

<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/leaderboard>

**Justification**

The log loss on the final model is 0.3458 on the test dataset on Kaggle. This is significantly better than the benchmark of 0.5 defined for this project. The final model for this problem is a deep convolution neural network that utilizes 6 convolution layers, 2 MaxPool layers and 2 fully connected layers on colored images to predict probability of cat vs dog. I have tried many different combinations of network parameters, optimization function and regularization to tune the model to its final setting. I see little incremental benefit of continuing this optimization further and feel comfortable stopping at this point.

The model has an accuracy of 85.6% on validation set in its ability to correctly predict cat vs dog. This is still significantly below what a human would do and represents a limit on machine learning on real world images. I think an algorithm of this accuracy can still be used by say Facebook or Google to predict the probability of cat or a dog in an untagged image and suggest a description of the image to the user.

**Conclusion**

**Free-Form Visualization**

**Reflection**

I went through the following phases as I worked through this project

1. Understanding of deep learning, convolution neural networks and Tensorflow through courses and books. When I started I had little to no understanding of this field. This was the pre-phase to give me the basic tools to get started.

2. Pre-Processing of the dataset – Learning about image processing was the next phase. Deciding what pre-processing must be done with the first iteration (resize image, flatten to grayscale, normalize by mean and std) vs what can be tried later after I get comfortable (image augmentation, colored images)

3. Building the first model – Writing the code for this first model took time as I learnt how to code with TensorFlow and TFLean and then ran into the challenge that my CPU was too slow for processing. I didn’t want to invest in a GPU so again started learning about Amazon EC2. It was a huge relief moment once the first model ran. The structure of this first neural network was the basic structure suggested by courses and books and the goal of this phase was to just get something working.

4. Optimization and Refinement – Personally this was the best phase. To learn more what should be changed or matters really drove my learning of convolution neural network. I read scores of research papers, blogs and Andrej Karpathy’s Stanford course to understand different architectures that I have been tried, latest research in this field etc. and started to chalk out what I want to tune with my basic model. I found EC2 GPU super useful in this stage as the speed of processing allowed many iterations to optimize the model.

5. The final phase as documenting all my findings in this report and uploading the code to my GitHub repository

This whole project was a very interesting, engaging and deeply rewarding exercise for me. I now feel I have the toolkit to get started with problems of this kind involving image processing and prediction. I think every problem will have nuances but I know how to get started. As well as basics on tensorflow, neural network and GPU implementation gives me confidence to venture into other areas like use of recurrent neural networks for natural language processing.

I think there were several interesting learning for me through this exercise:

* The depth of neural network materially affects accuracy. As the network gets deeper, its ability to learn finer features improves. This is intuitive. However I found that increasing the number of neurons in each layer didn’t always have a positive effect on accuracy and sometimes decreased accuracy. The best setting for me was a deep network with small number of neurons in each layer
* The optimization function and its learning rate have a huge effect on the accuracy of the final model. This maybe the single most important setting
* Use of L2 regularization on all layers was powerful to reduce overfitting
* Image augmentation didn’t have as much positive affect as I thought it would. Maybe this particular problem had enough images and generating new ones materially different from existing decreased accuracy
* Addition of color channels also didn’t materially improve accuracy. A bit surprising since dogs and cats have different color and is a valuable input to human eye

**Improvement**

As I look at the leaderboard for this project, I see submissions with log loss rates significantly better than mine – lowest log loss rate is 0.047 compared to mine - 0.345. I definitely wonder how I can further improve my model to such high accuracy. Here are some steps that I think can be undertaken:

* Use of external data (other than the 25k images) from this project to further train the model on a wide sample. Maybe there is an external database of these images that is publically available and can be used
* Better input pre-processing to help the model – If there are multiple animals in the image, then remove all but one. If there are humans along with their pets, remove the humans from the pictures. Maybe just keep the face of the animal as that has the most valuable features in distinguishing b/w them.
* Use of other more modern techniques that can help reduce overfitting. One that I would have tried was Stochastic Pooling as some research papers had good reviews however TensorFlow currently doesn’t have support for this

Next Steps

1. Complete free-form visualization

2. Zip and submit. Include the following

* Capstone report in pdf
* Readme in pdf
* Final code.py
* Cd\_color pickle file