

# Image Classification: Dogs vs. Cats (Kaggle Competition)

Wei Sun

Aug 4, 2017

## Definition

### Project Overview

Image classification is an important technique in computer vision, and has a wide variety of applications. For example, it has been applied to detect foreign species that invade the Georgia forest [4] so as to protect the ecosystem, to detect distracted driver so as to promote road safety [5], and even to study the galaxy so as to push the boundaries of astronomy research [6].

In this capstone project, I am interested in a specific image classification problem: distinguish dogs from cats. This problem originates from a Kaggle machine learning competition: “Dogs vs. Cats Redux: Kernels Edition” [2], which provides the relevant dataset to participants.

My interest in this problem is personal and academic, rather than for practical purpose. Before I have enrolled in this nanodegree program, I have taken an online course [1] taught by Jeremy Howard, former president of Kaggle. In that course, Jeremy taught how to fine-tune the VGG-16 model to solve this problem. In the homework, he has asked us to get into top 50% of the competition. Back then, the competition was still open to submissions, and I have tried hard to score high on the leaderboard. After 23 submissions, I have finally ranked No. 128 out of 1314 teams. Guess what? Jeremy was immediately below me on the leaderboard, ranked No. 129. I felt excited and surprised, and motivated to learn more about machine learning.

In this project, my goal is to beat my previous record, with the new knowledge I have learned in this nanodegree program. Although I cannot think about any direct practical value to distinguish dogs from cats, I believe that what I learned from this project – for pushing my personal limit – can be applied in other more valuable computer vision projects in the future.

### Problem Statement

The problem is to distinguish images of dogs from cats. To elaborate, we want to train a deep neural network to tell whether any given image is a dog or a cat, assuming that the input image is always either a dog or a cat, but not both at the same time. Further details are explained in the competition homepage [2].

## Metrics

Two metrics are used to evaluate and compare the machine learning models: LogLoss for testing dataset and Accuracy for validation dataset.

Following the official evaluation standard for this Kaggle competition [3], LogLoss on the testing dataset will be used as the primary evaluation metrics. Its formula is given below, and further explanation can be found in [3].

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

We can use the two models to predict the probabilities of the testing dataset, and submit the prediction results to Kaggle to calculate the LogLoss metric. The goal is to minimize this metric.

Although LogLoss for testing dataset is what ultimately matters in this competition, it cannot be easily interpreted. Therefore, we also introduce Accuracy for the validation dataset as a secondary evaluation metrics that is more intuitive to understand. The formula is given below.

$$\text{accuracy} = \frac{\text{number of correctly classified cases}}{\text{number of total cases}}$$

Note: clearly, Kaggle has the labels of the testing dataset so that they can compute LogLoss on it. However, these labels are not given to competition participants. Therefore, participants can only compute accuracy for the training and validation datasets, but not the testing dataset.

# Analysis

## Data Exploration

**Training and Validation Set:** Kaggle provides a labeled dataset for training, including 12500 dogs images and 12497 cats images. The images are RGB color images. I will randomly divide the images in each class so that 80% are used as training dataset and the other 20% are used as validation dataset, using stratified random sampling method.

**Testing Set:** Kaggle also provides an unlabeled dataset of 12500 images of dogs and cats.

## Explorative Visualization

Given the simplicity of this dataset, there is not much to visualize. By browsing the images, I found that the images have different sizes and aspect ratios. Some images contain the dog or cat only, while others also contain human. Below are a few representative sample images from the dataset.

Sample Dog Images



Sample Cat Images



## Algorithms and Techniques

Since the images have labels, it is appropriate to apply supervised learning methods. Since most images are similar to ImageNet images, it is preferable to apply transfer learning, rather than designing my own neural network architecture from scratch. In fact, as I explained in project overview, the first time I attempted this competition, I have already applied transfer learning with the VGG-16 model, by following Jeremy's course. In this project, I want to use more powerful models to achieve better performance. The details will be explained in the subsequent methodology section.

## Benchmark

As explained in project overview, the benchmark model is my previous implementation based on fine-tuning the VGG-16 model, for the same task of distinguishing dogs from cats in images. Following Jeremy's course, the benchmark model uses not only the convolutional layers, but also the fully connected layers, except the output layer, of VGG-16.

# Methodology

## Data Preprocessing

The preprocessing consists of the following steps:

- A list of transfer learning models trained on ImageNet data are selected. After a few iterations, the final list consists of ResNet-50, VGG-19, Xception, InceptionV3, and MobileNet. (The selection process will be explained in the refinement subsection below.)
- Following instructions in the deep learning project of the nanodegree program, I use the convolutional layers – followed by global average pooling – of each transfer learning model to preprocess the images separately, to turn each image into a feature vector. Thus, for each pair of model and image, a feature vector is extracted.
- For each image in both labeled and unlabeled dataset, their feature vectors from different models are concatenated together into a combined feature vector. (The order of concatenation is the same for all images.)
- The labeled dataset is randomly divided into 80% training set and 20% validation set using stratified sampling method.

## Implementation

A neural network (visualized in the right figure) is constructed that takes the preprocessed image features as input, and produces a two-way softmax<sup>1</sup> as output. In between, there are two hidden layers (with leaky relu activation) of 1024 neurons each.

- To make the neural network generalize better, there is a dropout layer (with dropout rate of 60%) after the input and each hidden layers.
- To speedup training, there is a batch normalization layer after each hidden layer.
- In the subsequent refinement subsection, I will explain how this architecture has been chosen though hyper-parameter tuning.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 7680)	0
dropout_1 (Dropout)	(None, 7680)	0
dense_1 (Dense)	(None, 1024)	7865344
dropout_2 (Dropout)	(None, 1024)	0
leaky_re_lu_1 (LeakyReLU)	(None, 1024)	0
batch_normalization_1 (Batch Normalization)	(None, 1024)	4096
dense_2 (Dense)	(None, 1024)	1049600
dropout_3 (Dropout)	(None, 1024)	0
leaky_re_lu_2 (LeakyReLU)	(None, 1024)	0
batch_normalization_2 (Batch Normalization)	(None, 1024)	4096
dense_3 (Dense)	(None, 2)	2050
Total params: 8,925,186		
Trainable params: 8,921,090		
Non-trainable params: 4,096		

The preprocessed labeled training and validation dataset are then used to train the neural network using standard supervised learning methods for 50 epochs. The network weights that results in smallest validation set loss (i.e., validation set Log Error) are saved and later used to make predictions of the test set.

Once training is finished, this neural network is applied to make predictions using the preprocessed feature vectors of the test set images. The predicted probabilities that each test set image is a cat is then collected into a CSV file, and finally submitted to Kaggle to be scored.

## Refinement

**On Selection of Transfer Learning Models:** In my historical attempt to the problem (i.e., the benchmark model), I have used VGG-16 as the transfer learning model, according to the course I took back then. In capstone proposal of this nanodegree program, I proposed to use ResNet-50, which is a natural choice since it is a successor to VGG-16, and is something new that I have learn in this nanodegree program. On second thought, I want to be more creative than applying ResNet-

---

<sup>1</sup> This is equivalent to a single sigmoid output neuron. Although I understand that a two-way softmax results in a few more parameters to train, I have chosen this more general architecture because it applies to not only binary but also multimodal classification. There is no noticeable difference if a single sigmoid output neuron is used. The choice is just personal preference. Why do I prefer softmax? The last time I tried Keras with the Theano backend with GPU, using a single sigmoid output neuron results in NVidia compiler errors. This is likely due to bug of Theano because such problem does not exist when using the Tensorflow backend.

50 alone, which has already been taught in the deep learning project of this nanodegree program. Hence, I have decided to apply transfer learning by combining multiple relevant models together – this is a new idea that the nanodegree program did not teach. The intuition is natural. If having one expert (e.g., VGG-16) on the decision (i.e., classification) committee improves performance, having multiple experts (e.g., both VGG-16 and ResNet-50 together) is likely to improve the performance even more. Finally, I have decided to use a combination of ResNet-50, VGG-19, Xception, InceptionV3, and MobileNet. (These – plus VGG-16 – constitutes a complete list of models that are implemented in the Keras API v2.0.2. VGG-16 is not included in my list because it is used in the benchmark model, and the similar but more powerful VGG-19 is included already.) To combine them in transfer learning, we use the convolutional part – followed by global average pooling – of each of them to extract features from an image, and concatenate feature vectors from different models together into a big feature vector. The combined feature vectors of the images are then used as preprocessed input to train a neural network for classification.

**On Neural Network Architecture:** The neural network architecture has been chosen through extensive hyper-parameter tuning. The combination that results in lowest validation loss is chosen. The hyper-parameters include dropout rate (ranging from 0.1 to 0.8), the number of hidden layers (0, 1, 2, 3, or 4), and number of neurons in the hidden layers (256, 512, 1024, 2048, or 4096). Note: the dropout rate is chosen so that the training and validation loss stay close to each other towards the end of training – indicating that the model is likely to generalize well.

# Results

## Model Evaluation and Validation

Using stratified random sampling method, 20% of the labeled dataset are selected as validation dataset. During development, the model is evaluated using the validation dataset. Extensive hyper-parameter tuning has been used during development, the final neural network architecture and its hyper-parameters are chosen because they minimize validation loss (i.e., LogLoss for the validation dataset) among all the tried combinations.

The hyper-parameter tuning process has been explained in detail in the “refinement” subsection of the previous section. The final architecture and hyper-parameters has been shown in the “implementation” subsection of the previous section.

Some additional technical details are outlined below:

- The training ran for 50 epochs.
- Parameter initialization are done per the default settings of Keras.
- The initial learning rate is  $2e-4$ . “Adam” is chosen as the optimizer, and it adapts the learning rate during the training process.

The model is concluded to be robust because it has achieved a very high validation set accuracy of 99.8% (It performed the same on the training dataset as well.) In addition, it has also performed very well on the testing dataset. More details will be given in the next subsection.












## Justification

The evaluation metrics for the solution and benchmark models are compared in the table below. This indicates that *the solution model significantly outperforms the benchmark model*.

	Testing Dataset LogLoss	Kaggle Leaderboard Rank	Validation Dataset Accuracy
<b>Solution Model</b> (Implemented In This Capstone Project)	0.04080	#18 Top 2%	99.8%
<b>Benchmark Model</b> (Implemented In A Previous Course Project)	0.06083	#128 Top 10%	98.5%

Note: Although testing dataset are provided to public without labels, competition participants can submit the prediction result to Kaggle to compute the LogLoss metrics. Obviously, Kaggle has the labels of the testing dataset so that they can do such computation.

A screenshot of my score (both old and new) of this Kaggle competition [2] is provided below for reference. *I am “Wilson Sun” on the leaderboard.* Since the competition deadline has passed, my new submission does not change the publicly visible score and rank.

16	—	yangpeiwen		0.04009	27	5mo
17	—	Anjith George		0.04078	46	7mo
18	—	ibless12345678		0.04128	13	5mo
19	—	NK255		0.04134	6	7mo
20	—	nash		0.04184	5	6mo
... ..						
125	—	John Vial		0.06038	18	5mo
126	—	tmuzap		0.06054	21	7mo
127	—	icodingc		0.06068	12	7mo
128	—	Wilson Sun		0.06083	23	5mo
<b>Your Best Entry</b>  Your submission scored 0.04080, which is not an improvement of your best score. Keep trying!						
129	—	Jeremy Howard		0.06086	11	9mo

Note: The above screenshot says, “*not an improvement of your best score*”. This is obviously a bug of Kaggle because the competition goal is to minimize the LogLoss score, and the new submission achieved lower LogLoss score.



# Conclusion

## Free Form Visualization

This project has a very simple goal: to distinguish images of dogs from cats. Thus, the most interesting visualization for this project is to investigate what images are incorrectly classified by the neural network. Hence, I have identified and studied these images. There are approximately 20 incorrectly classified images for dogs and another 20 for cats (Note: The exact set of images differs across runs because the neural network weights are randomly initialized and thus the training result differs slightly across runs.)

I have classified these images into a few categories and subcategories. The taxonomy is presented as follows along with example images from the labelled dataset. *For some of them, I have also provided discussions regarding how the neural network and its training process could be improved to obtain better performance for similar images. Such discussions are highlighted in italic font.*

The first category are images that are given incorrect labels in the dataset. (i.e., These incorrect labels are given as incorrect facts rather than computed or predicted.) *It seems beneficial to exclude them from the labeled dataset. In addition, this also seems to indicate that Kaggle's labeled dataset may be obtained from image search, and there is a small chance that the search return images that do not match the search keyword. This inspires us that we can obtain more labelled images via Google image search, such that our neural network training is not limited to the images provided by Kaggle.* There are three subcategories.

First subcategory are images that contain both dogs and cats.



Second subcategory are images that contain neither dogs nor cats.



Third subcategory are dog images incorrectly labeled as cats, or vice versa.



The second category are images with distracting background or foreground.

First subcategory are images with distracting background (e.g., people, bookshelf), while the animal is typically very small compared to the background. Perhaps, in the “eye” of the neural network, a person looks more like a dog than cat.



Second subcategory are images with distracting foreground, such as when an animal is inside a cage. *To improve classification performance for these images, we can try augmenting the existing labeled dataset by photo-synthesizing the foreground (e.g., cage).*



The third category are images with insufficient clues such that even a human would have at least some difficulty labeling them. There are three subcategories.

First subcategory are images taken at weird angles.



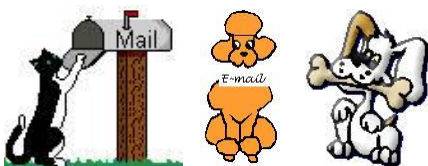
Second subcategory are images with low resolution or taken under poor lighting conditions.



Third subcategory are images where the animals are black. *I theorize that, in this case, lots of image details are lost in preprocessing, such that it is difficult for the neural network to get enough useful features for classification, especially when the image is zoomed out before feed to the transfer learning models. These images may benefit from different ways of preprocessing.*



The fourth and final category are cartoons. *Since not many cartoons are found in the training dataset as I browsed, it is likely that if we include more cartoons in training, the neural network would then perform better for cartoons. In addition, note that the transfer learning models are trained on real ImageNet photos rather than cartoons. Therefore, the neural network is likely to benefit from adding a transfer learning model that is trained on cartoons.*





## Reflection

This project consists of the following steps.

1. A Kaggle competition is selected as the project goal.
2. Datasets are obtained from the Kaggle competition.
3. Transfer learning is applied to preprocess the images to extract features. This project combines quite a few transfer learning models together, including ResNet-50, VGG-19, InceptionV3, Xception, and MobileNet.
4. A relatively simple neural network is constructed, to take the combined features as input, and predict whether the corresponding image is a dog or a cat as output.
5. The combined feature vectors of the images are used to train the neural network.
6. A few different combinations of architectures and hyper-parameters are tried in hyper-parameter tuning. Steps 4 and 5 are repeated for these combinations. The best combination that minimizes validation loss is chosen to finalize the neural network.
7. The finalized neural network is used to make predictions on the unlabeled testing dataset. The prediction results are submitted to Kaggle to compute the LogLoss score, and get ranked on the competition leaderboard.
8. The finalized neural network is used to make predictions on the labeled dataset to identify which images are incorrectly classified. These images are then analyzed in the previous subsection titled “free form visualization”.

Among the above, I have found selecting the transfer learning models in step 3 to be most interesting. This is the key step that enables me to construct a model that significantly outperforms the benchmark (i.e., my previous personal record). Once the idea in this step has been conceived, outperforming the benchmark seems quite easy.

I have found hyper-parameter tuning in step 6 to be difficult, because there is no standard way to decide what exact combinations to try in hyper-parameter tuning. This process heavily relies on experience and trial-and-error. As mentioned in the course materials for this nanodegree program, the practice is way ahead of the science in deep learning.

Last but certainly not least, I have obtained many very useful insights in step 8, i.e., analyzing the incorrectly classified images. It seems beneficial to do so for all my future projects.

## Improvement

The implementation does not use any data augmentation techniques (e.g., randomly rotate the image before using it as input). These techniques are known to be helpful for training neural network for image classification. Therefore, one potential improvement is to apply data augmentation. This effectively increases the size of the labeled dataset, and is likely to be useful for achieving better training result.

While the above are very general discussions, *some specific improvements* tailor to this Kaggle competition have been proposed and discussed in the “free-form visualization” subsection, *and are highlighted in italic font*. I hope the readers of this report will find that discussion interesting.

## Appendix

### Software Development Environment

- Amazon EC2 p2.xlarge instance with NVidia K80 GPU
- Ubuntu Linux 16.04.2 LTS x86\_64
- NVidia CUDA 8.0 with CUDNN 5.1
- Anaconda 4.4 with Python 2.7
- Keras 2.0.2 with Tensorflow GPU 1.2 Backend

### References

- [1] Jeremy Howard. Practical Deep Learning for Coders. 2016. <http://course.fast.ai/>
- [2] Kaggle. Dogs vs. Cats Redux: Kernels Edition. 2017. <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition>
- [3] Kaggle. Evaluation of Dogs vs. Cats Redux: Kernels Edition. 2017. <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition#evaluation>
- [4] Kaggle. Invasive Species Monitoring. 2017. <https://www.kaggle.com/c/invasive-species-monitoring>
- [5] Kaggle. State Farm Distracted Driver Detection. 2016. <https://www.kaggle.com/c/state-farm-distracted-driver-detection>
- [6] Kaggle. Galaxy Zoo – The Galaxy Challenge. 2014. <https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge>