Cats vs. Dogs Classification Report

1. Definition

Project Overview

The project is to write an algorithm to classify whether images contains either a dog or a cat. This is easy for humans, dogs, and cats. However, the computer will find it a bit more difficult. Such a challenge is often called a CAPTCHA[1] (Completely Automated Public Turing test to tell Computers and Humans Apart) or HIP (Human Interactive Proof). HIPs are used for many purposes, such as to reduce email and blog spam and prevent brute-force attacks on web site passwords.

This project will focus on algorithm approaches, from classic Convolutional Neural Network (CNN) to state-of-the-art models (e.g. VGGNet, ResNet, Inception, Xception). I will compare the accuracy and performance of all of these algorithms.

There are many public image datasets online for this challenge. Here, I will use the dataset from Kaggle’s Dogs vs. Cats Redux: Kernels Edition competition[2]. It has two folders: train and test. The train folder contains 25,000 images of dogs and cats. Each image in this folder has the label as part of the filename. The test folder contains 12,500 images, named according to a numeric id. For each image in the test set, I should predict a probability that the image is a dog (1 = dog, 0 = cat).

Project Statement

The “Cats vs. Dogs” competition is a supervised binary classification problem.There are two types of images: cats and dogs. The goal is to extract proper features and build an effective model to classify each image contains either cat or dog.

By exploring the dataset, basic information about the dataset can be obtained. The size of each image could be different and have to be resized as each model has a standard for input image size (e.g. 224\*224 for VGGNet and 299\*299 for Inception). After that, the full dataset (under train folder) should be split into training set and validation set. When the models are being trained, the performance is also evaluated on the testing set. By checking the loss and accuracy on training and validation sets, I can know if the model built is correct and if the model is under- or over-fitting. Finally, I will run prediction on test set and upload the result to Kaggle to see what position I can occupy.

Mertics

Usually we use ROC curve (receiver operating characteristic curve) and PR curve (Precision-Recall curve) to evaluate models for binary classification problem. However, in order to comply with Kaggle’s rule, I will use log loss instead.

where

n is the number of images in the test set

is the predicted probability of the image being a dog

is 1 if the image is a dog, 0 if cat

is the natural (base e) logarithm

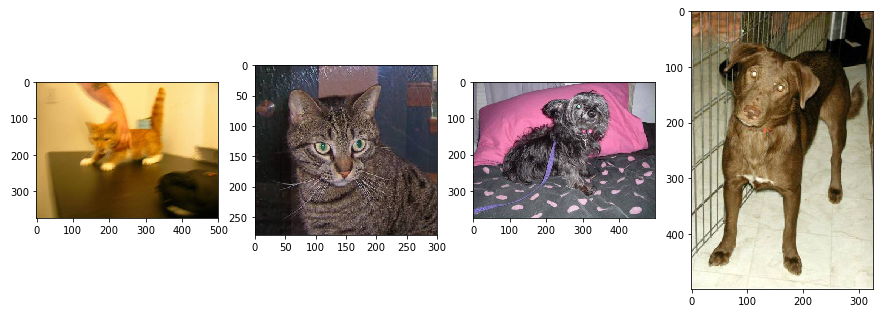
A smaller log loss is better.

1. Analysis

Data Exploration

The dataset is downloaded from <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>. There are two folders: train and test. The train folder has 25,000 images with names like dog.1.jpg, dog.2.jpg, cat.1.jpg, etc. The test folder has 12,500 images with names like 1.jpg, 2.jpg, 3.jpg, etc.

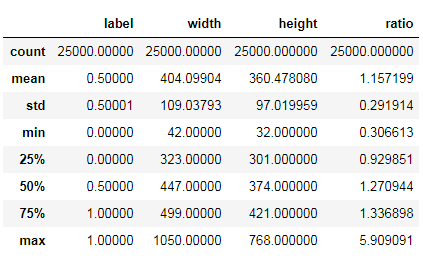
Here are some sample images with associated width and height.



Here are a few things we can notice directly by looking at above images:

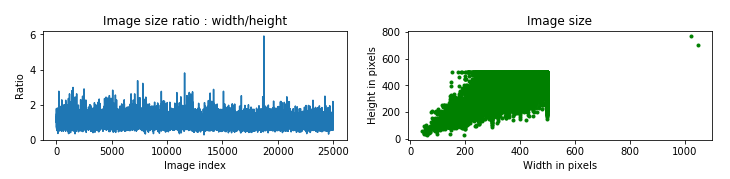
1. Images are generally well centered on the animal.
2. Images have different sizes.
3. The quality of some images are poor (e.g. 1st image).
4. Background images seem to vary a lot and be independent from the categories (the human hand in 1st image and pink pillow in 3rd image).

The following table shows the summary of image info: width, height and ratio (width / height).You can neglect “label” as it is a categorical value. The table clearly shows the sizes of images are varied.



Exploratory Visualization

We have noticed the dataset contains different sizes of images. The following plots show the distribution of each image’s width, height and ratio (width / height).



The average ratio is 1.157.

We can see two outliers in the “Image size” graph which have far too high height and width compared to the rest of the data points. On the “ratio” graph, we can also notice an outlier, which has a ratio of 6 whereas most other images have a ratio below 2.

Since most of models (e.g. VGGNet, Inception) require fixed image size, these graphs can help us determine if we need to resize or crop our images before using them as input of the models.

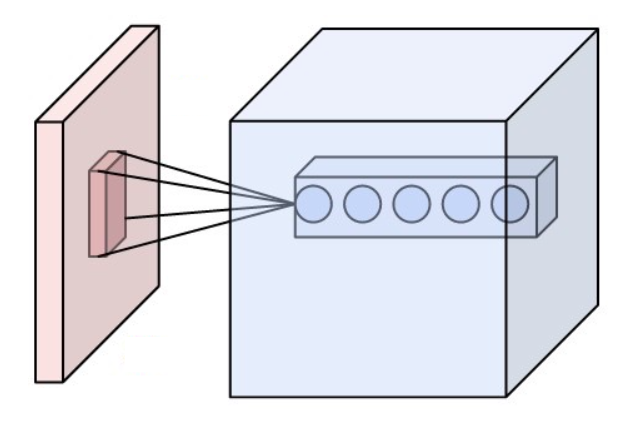
Algorithm and Techniques

For classification problem, the common method we will use is logistic regression or more generally, softmax regression. However, they have limited efficiency when the problem it tries to solve is using data with a high variability (many different inputs but have the same label), which is the case here: two images of cats can be extremely different. Images represent animals in different positions and background make it even harder for this method.

Recently a popular tool used for image classification is Convolutional Neural Network (CNN). It is very useful for images where important features (like a portion of a cat ear) can be anywhere, which is the case here as animals are in different position and sometimes more than one per image.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typical consist of convolutional layers, pooling layers, fully connected layer, ReLu layers and sometimes dropout layers.[2]

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.



*Neurons of a convolutional layer (blue), connected to their receptive field (red)*

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.



Max pooling with a 2\*2 filter and stride = 2

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function . It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent , , and the sigmoid function . ReLU is often preferred to other functions, because it trains the neural network several times faster without a significant penalty to generalisation accuracy.

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Sometimes there is a dropout layer. Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability or kept with probability , so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

Since the image is labeled (let’s say it is a cat) and we know the influence of each neuron’s weight, we automatically tweak these weights to make our model closer to a 100% confidence it is a cat. This adjustment is called “backpropagation”. To do this, we compute a loss function and find its lowest value with an optimizer to find the best combination of these weights.

Besides the CNN model, I will also use some pre-trained models for transfer learning. They are VGGNet, ResNet and Inception/Xception.

VGGNet comes from the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” authored by Karen Simonyan and Andrew Zisserman [3]. In this paper, they investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Their main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers. These findings were the basis of their ImageNet Challenge 2014 submission, where their team secured the first and the second places in the localization and classification tracks respectively. They also show that their representations generalize well to other datasets, where they achieve state-of-the-art results.

ResNet is created by Kaiming He, etc [4] and they present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. They explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. They provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task.

InceptionV3 is released by Google and their work has been recorded in paper “Rethinking the Inception Architecture for Computer Vision”[5]. They explore ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. They benchmark their methods on the ILSVRC 2012 classification challenge validation set demonstrate substantial gains over the state of the art: 21.2% top-1 and 5.6% top-5 error for single frame evaluation using a network with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. With an ensemble of 4 models and multi-crop evaluation, they report 3.5% top-5 error on the validation set (3.6% error on the test set) and 17.3% top-1 error on the validation set.

Benchmark

The Kaggle leaderboard shows data scientists managing to reach a log loss of 0.05629 as top 100. The method to calculate log loss I have described in the “Metrics” part above. Let’s see if the model I have built can beat this score.

1. Methodology

Data Preprocessing

Ideally, the model needs some good quality input but the image size could not be too big so that the model cannot handle. To find patterns among high volumes of information, we need large models, which cannot be the case due to our limited computation capabilities. In addition, it is easier to understand the image content if there is less information to filter. So it is important to have lower resolution images and we can achieve this by resizing them.

We can see there are two outliers images with largest width and height mentioned in “Exploratory Visualization” part. However, I would not like to delete them as the model is focused on understanding the content of image while the size doesn’t impact the result. I will simply resize to make them the same as other images.

I will then split the images under “train” folder into training and validation set. The validation set is used to verify if the model has under- or over-fit the training set. I will extract 20% data as validation set. Since the entire test data is in “test” folder, I can treat it as test set.

|  |  |  |
| --- | --- | --- |
|  | Cats Number | Dogs Number |
| Training Set | 10000 | 10000 |
| Validation Set | 2500 | 2500 |

All the data sets are stored as numpy arrays. I have tried to store as pickle file so that I can load the data back easily when I want to use it. However, due to limit storage on AWS, I have to clean them up.

Implementation

The data exploration is saved in a separate notebook. I list the basic statistics of the dataset and plot some graph to show the data distribution.

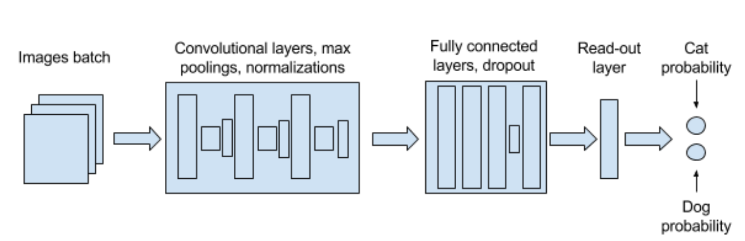
Then I have a separate notebook to illustrate how to build a CNN from scratch. Here, I refer the structure of VGG16. The CNN model has 4 connected component method (CCM). Each CCM has the following structures:

Conv -> Batch Normalization -> Conv -> Batch Normalization -> MaxPooling-

Batch Normalization is a method to reduce internal covariate shift in neural networks, first described in [6], leading to the possible usage of higher learning rates. In principle, the method adds an additional step between the layers, in which the output of the layer before is normalized.

After 4 CCM, the last 3 layers are Flatten, Dropout and Dense layers.

The entire model is like this:



While the number of each layer type was determined iteratively, based on the performance of each tried model architecture.

To easily iterate over different algorithms architecture and parameters combinations to find proper model structure, we can leverage the function of saving and loading trained models from checkpoints as it is easy to add training steps to already trained model. Successful training sessions’ model-performance pairs were stored by saving copies of Jupyter notebook to for later analysis.

I choose Keras to build the model. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. So that I don’t need to build the model from scratch but focus on the model structure.

The main steps of the algorithm are:

1. Define the model structure as I have described above.
2. Given the input parameters (e.g. the number of CCM and kernel depth) to create model instance.
3. Compile the model with proper loss function, optimizer and performance metric.
4. Fit the model with training data and test with validation data parallel. We check if the loss (both training and validation) is decreasing and accuracy (both training and validation) is increasing which can tell us if the model is under or over fitting.
5. Last we can apply the model to prediction.

Images are randomly grouped in batches of 32 (small batches result in quick convergence), 10 epochs (10 iterations on the entire dataset).

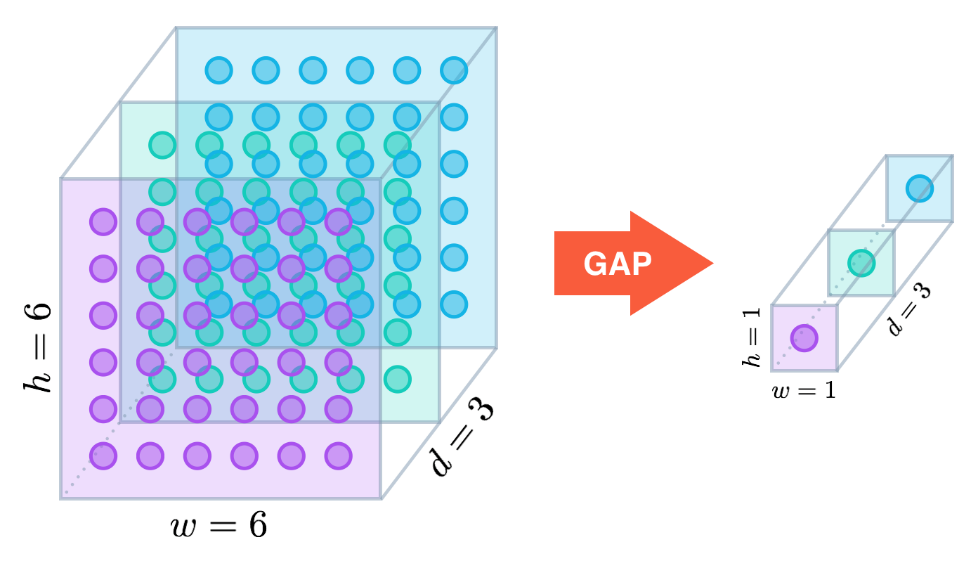
Some problems occurred during the iterations over different models due to the laptop used for training. When convolutional and fully connected layers had too many neurons, or when images resolution were too high, the python kernel regularly crashed, probably because of insufficient available memory.

So I move the training process to AWS which can provide large RAM and GPU supported. Then each epoch reduced to around 130 seconds (although the 1st epoch took a little longer). This is acceptable as the entire training process just took 20 minutes. Finally, my best training accuracy is 0.9381 and validation accuracy is 0.8940, which is good, but I think there is still room to improve.

Refinement

Transfer learning is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

There are several approaches to implement transfer learning. One way is to freeze all the convolution layers in the pre-trained model then replace fully connected layers with your own classifier. The pre-trained models I have used are VGG16 (22 layers) and ResNet50 (178 layers). Both models have been froze all the convolution layers, then I apply global average pooling (GAP) to the output of the last convolutional layer. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions is reduced in size to have dimensions . GAP layers reduce each feature map to a single number by simply taking the average of all values.[7]



Then I add a dropout layer with rate of 0.5, which means half of input units will be dropped. Last is the dense layer using sigmoid as activation function. The advantage is to reduce the number of trainable parameters that results in the reducing of total training time.

Here are the results when setting batch size = 16, epochs = 5:

|  |  |  |
| --- | --- | --- |
|  | VGG16 | ResNet50 |
| Total params | 14,715,201 | 23,589,761 |
| Trainable params | 513 | 2,049 |
| Non-trainable params | 14,714,688 | 23,587,712 |
| Average time per epoch | 484 s | 296 s |
| Training loss | 0.2120 | 0.0868 |
| Training accuracy | 0.9415 | 0.9657 |
| Validation loss | 0.0973 | 0.0566 |
| Validation accuracy | 0.9680 | 0.9772 |

We can also “fine tune” the last several layers based on above method. Here I use ResNet50 as pre-trained model. Since several layers form a component (e.g. convolution + batch normalization + activation), I have blocked/unblocked them together. I start from high (output) to low (input) layers and train 5 epochs when unblocking one component. Then I will check if the loss and accuracy have improved. If the answer is yes, then we can continue training. Otherwise, we may need to unblock more layers until the training result has improvement. However, even I unblock the last 35 layers, the improvement from fine tuning is still not obvious. The training loss is 0.081 with accuracy of 0.9698 while the validation loss is 0.0496 with accuracy of 0.9794.

The last way I have tried is to ensemble multiple models. I calculate the feature vectors through different models and then merge these vectors. After that, I train the classifier based on these vectors. Since we have read the training and test datasets into memory we can use model.predict() to get feature vectors directly.

1. Results

Model Evaluation and Validation

The used images’ height, width and ratio are good enough to let the algorithm detect interesting patterns but small enough not to embarrass the algorithm with too much information or simply make it crash.

I managed to build a model that learnt to distinguish images of cats from images of dogs. Indeed, my final model (the one after merged feature vectors from VGG16, ResNet50 and InceptionV3) has reached validation loss = 0.0340 with accuracy = 0.9888.

The original split of the labeled dataset into training, validation and test set gave us an appropriate way to evaluate the model. Using only the training model may mislead us since we would not have been able to distinguish seen images from new images. We can see when there is overfitting and check that our model is indeed generalizing.

With the different approaches: CNN, transfer learning (Freezing, Fine Tune and Feature Vector), the validation loss continues decreasing while accuracy is raising up.

Justification

An accuracy of 0.9888 is good but we cannot say the model is trust since the model has not been verified through test set. Last, I use the final model to predict the probability of the image that contains dog and save the result as a csv file. Since Kaggle's scoring system will adjust the probability between 10^-15 and 1 - 10^-15), I change the output from 0/1 to 0.005/0.995 when the output is close to 0 or 1. After uploading the test result to Kaggle leaderboard, I can see the score (log loss) is 0.04944, which is at 57th place of public leaderboard. This has achieved my goal (enter the top 100).

Conclusion

It’s the first project that I apply various deep learning techniques to solve an actual problem. I have learnt how to combine theoretical and practical parts.

On the technical part, I have practiced coding with python and using various packages including numpy, matplotlib, opencv, tqdm and keras through data exploration, manipulation and visualization. Although I do not use Tensorflow api directly in this project, I have still learnt a lot when I reading the keras document to understand how keras uses tensorflow as backend. More importantly, I have learnt a complete process from loading data, resizing data and building model layers by layers, training the model and evaluating it on different datasets.

Choosing different parameters (e.g. filter size, stride size, optimizer) and optimizing its architecture also help me understand how each layer works and its role in the neural network architecture. During the training process, I have seen both scenarios of under and over fitting. I face the trade-offs and see the reality of improving a model’s accuracy due to metrics analysis and different techniques.

In terms of possible improvements, integrating feature vectors from more models and some other solutions like image enhancement (e.g. shift the image position) may be helpful. In addition, I can try some wider and deeper models that can learn more details of the images. This may give us results that are more accurate but spend more time to train the model.

References

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

[2] <https://en.wikipedia.org/wiki/Convolutional_neural_network>

[3] <https://arxiv.org/abs/1409.1556>

[4] <https://arxiv.org/abs/1512.03385>

[5] <https://arxiv.org/abs/1512.00567>

[6] https://arxiv.org/abs/1502.03167

[7] <https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/>