Avishek Kedia

Capstone Project
Machine Learning Engineer
Nanodegree

November 22, 2016

Distinguish images of dogs from cats

Definition

Project Overview

Image recognition have received a large amount of attention over the last few decades. We can see it through hundreds of books published on the subject and the growing of new big train in Machine Learning: Deep Learning. By using a hierarchy of numerous artificial neurons, deep learning can automatically classify images with a high degree of accuracy. Thus, neural networks can recognize different species of cats, or models of cars or airplanes from images.

In this project, we focus on classify whether images contain either a dog or a cat. We have built a cat/dog classifier using a deep learning algorithm called Convolutional Neural Network(CNN). The application uses a classifier trained using the Kaggle dataset. This project was inspired by Kaggle competition.

Problem statement

The goal is to classify whether a given image contain either a dog or a cat, the tasks involved are the following:

1. Download Dogs and Cats data.

First, we need to download the 2 datasets (train.zip and test.zip) from the competition page:

- train.zip contains labeled cats and dog's images that we will use to train the model.
- test.zip contains unlabeled cats and dog's images that we will use to classify to either dog or cat using the trained model.

2. Train the CNN on the training data
3. Make prediction on the testing data

Metrics

The accuracy metric for binary classifier with formula is used:

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

- True positives in this case is when the input is dog image and the CNN predict 1= dog as output; or the input animal is cat and our algorithm predict 0=cat.
- True negatives in this case is when the input is dog image and the CNN predict 0= cat as output; or the input animal is cat and our algorithm predict 1=dog.

Analysis

Data Exploration

The dataset (from Kaggle dataset) contains 25,000 labeled images of dogs and cats; and the testing data contains 12,500 not labeled images. The labels are not described here because only the images are used to predict if the corresponding one is cate or dog. The images are colored and have different pixels' size.

Figure 1: Images from the Kaggle dataset

Because of this difference, we have normalized our dataset as follow:

- Run histogram equalization on all training images. Histogram equalization is a technique for adjusting the contrast of images.
- Resize all training images to a 227x227 format.
- Divide the training data into 2 sets: One for training (5/6 of images) and the other for validation (1/6 of images). The training set is used to train the model, and the validation set is used to calculate the accuracy of the model.
- Store the training and validation in 2 LMDB databases. train_lmdb for training the model and validation_lmbd for model evaluation.

Exploratory visualization

The training set contains equals numbers of cats and dogs images i.e. 12,500 for each category. See the Data exploration section for more details.

Algorithms and techniques

The classifier is a Convolutional Neural Network, which is an algorithm for image processing tasks, including classification. CNN needs a large amount of training data; the cat's dog's datasets are big enough. The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positives using a threshold. The following parameters can be tuned to optimize the classifier:

- Classification threshold (see above)
- Training parameters
  - Maximum Iteration (number of epochs)
  - Solver mode (GPU or CPU)
  - Batch size (how many images to look at once during a single training step)
  - Solver type (what algorithm to use for learning)

- Learning rate policy (learning rate policy)
- Learning rate (how fast to learn; this can be dynamic)
- Weight decay (prevents the model being dominated by a few "neurons")
- Momentum (takes the previous learning step into account when calculating the next one)

- Neural network architecture
  - Number of layers
  - Layer types (convolutional, fully-connected, or pooling)
  - Layer parameters (see links above)
- Preprocessing parameters (see the Data Preprocessing section)
  - Image size (image width, image size)
  - Image, label

Benchmark

To create an initial benchmark for the classifier, I used the bvlc_reference_caffenet model which is a replication of AlexNet with a few modifications.The "standard" AlexNet architecture (Fig. 2) achieved the best accuracy, around 0.9.

Methodology

Data Preprocessing

The preprocessing done consists of the following steps:

1. The list of training images is randomized
2. Run histogram equalization (a technique for adjusting the contrast of images) on all training images.
3. Resize all training images to a 227x227 format
4. The images are divided into a training set and validation set
5. Store the training and validation in 2 LMDB databases. train_lmdb for training the model and validation_lmbd for model evaluation.

Implementation

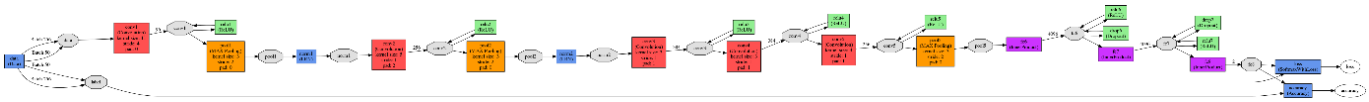
The implementation process is split into two main stages:

1- Model training
2- Model prediction

After defining the model and the solver, we started training the model .

During the training process, we monitored the loss and the model accuracy. We take a snapshot of the trained model every 5000 iterations, and store them under *caffe_model* folder. The snapshots have *.caffemodel* extension. For example, 5000 iterations snapshot will be called: *caffe_model_1_iter_5000.caffemodel*.
Training requires:

- *Caffenet_train_val_1.prototxt*: defines the network architecture(figure 2), initialization parameters, and local learning rates



- *Solver_1.prototxt*: defines optimization/training parameters and serves as the actual file that is called to train a deep network

  Some important parameters as described in Algorithm and techniques section are described as follow:

```
net: "/home/ubuntu/cats-dogs/caffe_model/caffenet_train_val_1.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.001  # begin training at a learning rate of 0.01 = 1e-2
lr_policy: "step" # learning rate policy: drop the learning rate in "steps"
                  # by a factor of gamma every stepsize iterations
gamma: 0.1 # drop the learning rate by a factor of 10
                  # (i.e., multiply it by a factor of gamma = 0.1)
stepsize: 2500 # drop the learning rate every 2500 iterations
display: 50
max_iter: 40000 # train for 40000 iterations total
momentum: 0.9
weight_decay: 0.0005
snapshot: 5000
snapshot_prefix: "/home/ubuntu/cats-dogs/caffe_model"
solver_mode: GPU
```

- *Caffenet_deploy_1.prototxt*: used only in testing. It must be exactly the same as *train_val.prototxtexcept* from the input layer(s), loss layer(s), and weights initialization (e.,g weight_filler) as the latter two do not exist in caffe_deploy_1.prototxt.

Figure 2: Caffe model Architecture

A learning curve(**Figure** 3) is a plot of the training and test losses as a function of the number of iterations. These plots are very useful to visualize the train/validation losses and validation accuracy.
We can see from the learning curve that the model achieved a validation accuracy of 90%, and it stopped improving after 3000 iterations.

The model prediction stage use the trained model to make predictions on images from test1. The code uses 4 files to run:

a) Test images: We have used test1 images.
b) Mean image: The mean image.
c) Model architecture file: We have called this file *caffenet_deploy_1.prototxt*. It's stored under /caffe_model folder. It's structured in a similar way to caffenet_train_val_1.prototxt, but with a few modifications. We have deleted the data layers, add an input layer and change the last layer type from SoftmaxWithLoss to Softmax.
d) Trained model weights: This is the file that we computed in the training phase. We use caffe_model_1_iter_10000.caffemodel.

The predictions will be stored under caffe_model/submission_model_1.csv.

Refinement

We use a solver for model optimization. We defined the solver's parameters in a .prototxt file. The solver is saved under /caffe_model with name solver_1.prototxt. Below is a copy of the same.

This solver computes the accuracy of the model using the validation set every 1000 iterations. The optimization process will run for a maximum of 40000 iterations and will take a snapshot of the trained model every 5000 iterations.

base_lr, lr_policy, gamma, momentum and *weight_decay* are hyperparameters that we need to tune to get a good convergence of the model. It is described in the previous section.

I chose:

lr_policy: "step" with

stepsize: 2500,

base_lr: 0.001 and

gamma: 0.1.

In this configuration, we started with a learning rate of 0.001, and we dropped the learning rate by a factor of ten every 2500 iterations.

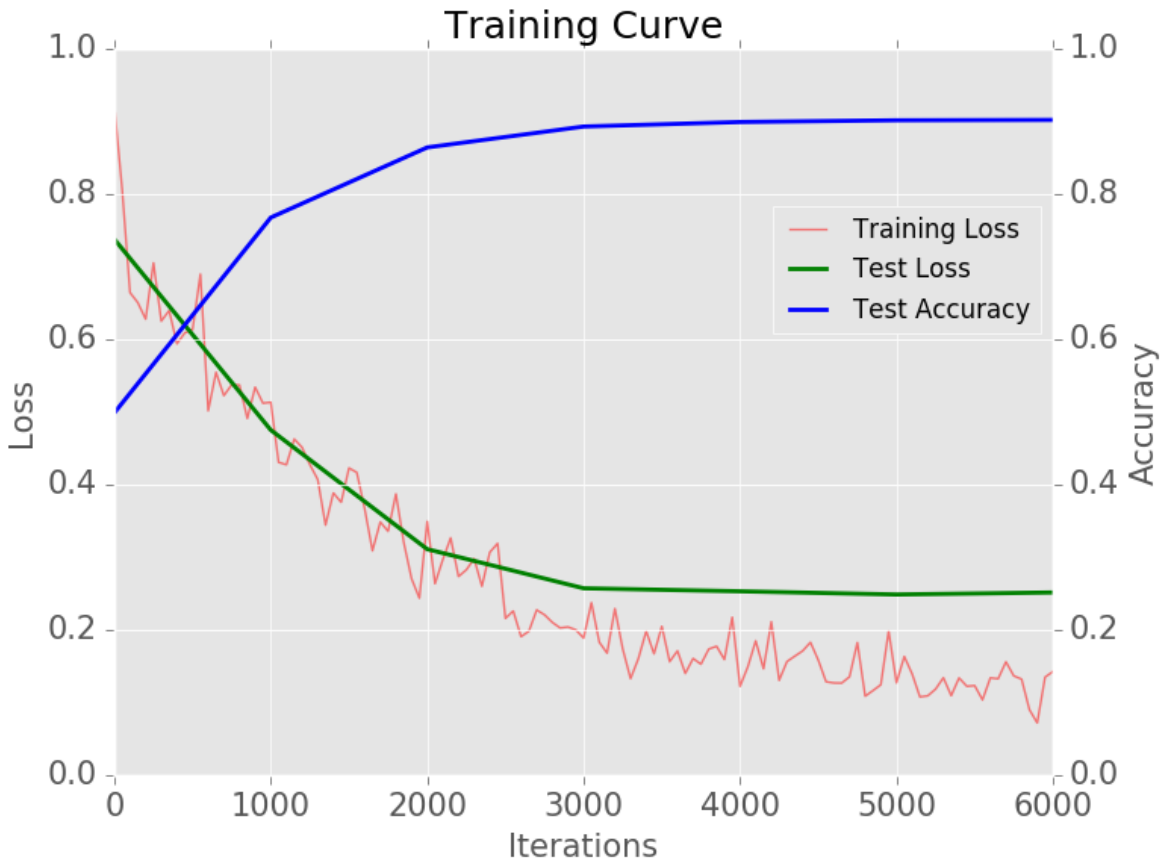


Fig. 6 A plot of the training/validation losses.

After submitting the prediction to Kaggle, it gives an accuracy of 0.89691.

Results

Model Evaluation and validation

During the training step, we divided the training data into training and validation. We use a validation set to evaluate the model. We refer to figure 2 describes to describe the model and the training process, as following:

We can see from the learning curve that the model achieved a validation accuracy of 90%, an d it stopped improving after 3000 iterations.

Justification

Using one AWS EC2 instance of type g2.2xlarge. This instance has a high-performance NVIDIA GPU with 1,536 CUDA cores and 4GB of video memory, 15GB of RAM and 8 vCPUs. The machine costs $0.65/hour with the deep learning framework caffe and anaconda2 installed. This help me to win in time.

Conclusion

Free-form Visualization

No form to visualize.

Reflection

My first challenge was to familiarize with the framework caffe and aws ec2 instance for deep learning, both of which were technologies that I was not familiar with before the project.

Improvement

To improve the model, we can make the following modification:

- Reduces the training time and the size of dataset by using techniques such as Transfer Learning instead of training the model from scratch.
- Add a web interface or mobile interface for our application for the public to use it.

Quality

Presentation

Functionality

https://www.kaggle.com/c/dogs-vs-cats

http://caffe.berkeleyvision.org/tutorial/solver.html

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://en.wikipedia.org/wiki/Lightning_Memory-Mapped_Database