Machine Learning Engineer Nanodegree

Capstone Report

Aadam

June 14, 2017

Project Overview

Domain Background

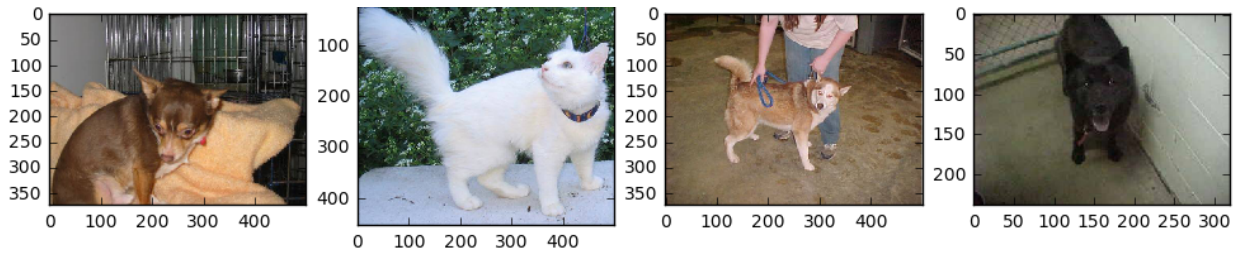
The project is taken from the kaggle competition, dogs-vs-cats-redux-kernels-edition, in which we need to predict whether a given picture if of a dog or a cat. For humans, this task seems quite easy and intuitive but for computers, the task of correctly recognizing objects in a picture seems quite difficult and daunting. For many years, researchers in computer vision have struggled with this problem, making computer understand a picture and what's in it. But with the advent of new techniques and algorithms, now the computers can be trained to predict what's in a picture quite effectively.

Datasets and Inputs

The dataset, provided here, contains labeled pictures of dogs and cats. 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, you should predict a probability that the image is a dog (1 = dog, 0 = cat).

We will use these pictures to train our CNN model and based on which we will generate predictions. Then we'll use the test dataset to validate our prediction model and see how well it performs.

Here are some of the sample images from the dataset:



Problem Statement

In this problem, our task is to identify whether a picture contains a dog or a cat. If it's a dog then we need to correctly identify him and vice versa. We need to analyze the pictures and then calculate the probabilities of every category, dog or cat, showing how confident we are in our predictions that it is a dog or a cat. So, we can summize that this is a classification problem. One possible solution for this problem is to train CNN models on the dataset and use those to classify whether a picture is of a dog or a cat.

Solution Statement

We can use the Convolutional Neural Network (CNN) which have been know to perform quite well on the Imagenet database. We will train our CNN model on the pictures that are provided of the dogs and cats and then we will validate it using the test dataset.

We will use **Keras** library to train our CNN model on this dataset and it will use **Theano** as the backend.

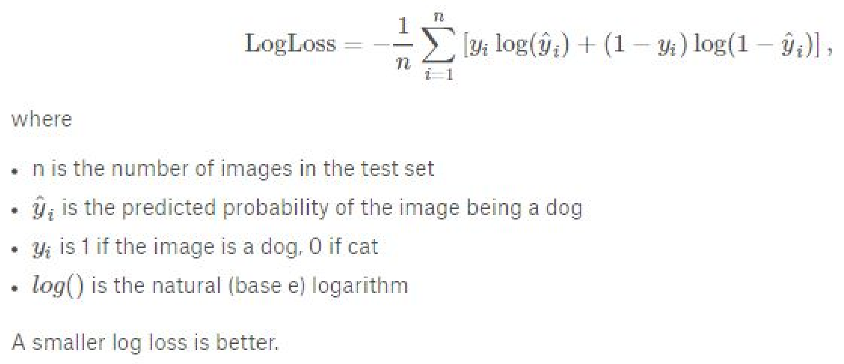
We will first extract some sample and validation datasets from the train dataset. We will use validation dataset to validate how our model is doing and we'll use sample dataset so that we can test our model quickly as it will contain smaller amount of data as compared to the train dataset. We will first divide the pictures in the train dataset into separate categories, namely dogs and cats and then we'll load those as batches and then train our CNN model on those batches. After we have fine tuned our model and trained it on the train dataset, we will then validate our model using the test dataset and see how well it performs.

So, our action plan will be something like this:

1. Create Validation and Sample sets
2. Rearrange image files into their respective directories
3. Finetune and Train model
4. Generate predictions
5. Validate predictions

Metrics

Instead of using the accuracy of predictions as our evaluation metric, the model will be evaluated based on its **LogLoss** value, which is defined to be:



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

where

- n is the number of images in the test set
- $\hat{y}_i$ is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- $log()$ is the natural (base e) logarithm

A smaller log loss is better.

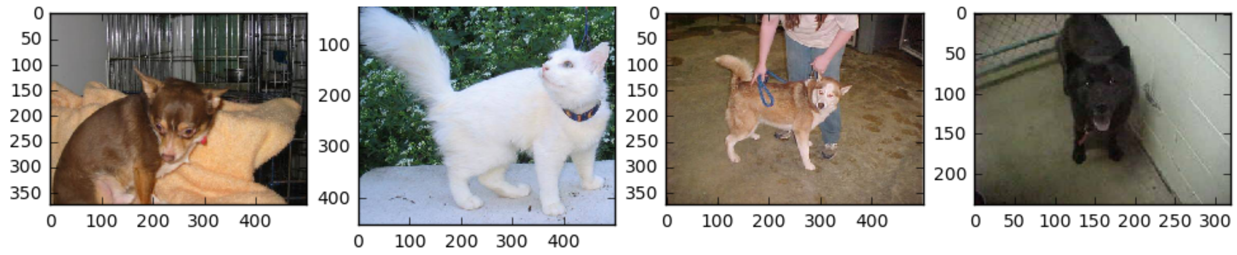
We are using LogLoss here because this is the metric that is being used in the Kaggle competition and our model will be evaluated on that. It will give a very low score if our prediction is way off from the correct answer.

Analysis

Data Exploration

The given dataset for this competition is the Asirra dataset provided by Microsoft Research. Our training set contains 25,000 images, including 12,500 images of dogs and 12,500 images of cats, while the test dataset contains 12,500 images. The average size for these images is around 350×500.

Here are some of the sample pictures from the dataset



Exploratory Visualization

As the data provided is simply the images of the cats and dogs, which are equal in number i-e 12,500 each, there’s no need for a visualization graph to see any hidden features and such because all the information that we need is already provided in the form of images.

Algorithms and Techniques

First of all, the data that we get from the Kaggle competition site is organized in two main directories, test and train, which contains pictures of cats and dogs that are named according to the animal they contain e.g. cats.124.jpg and dogs.786.jpg. So, at first, we’ll organize these into their separated directories i.e. all the cat pictures should be in a directory named cats and the dogs picture in dogs. We will also exclude a validation set from the train directory.

As this is an image classification problem, we will use a CNN model given their popularity and success with problems like these. Convolutional Neural Networks or ConvNets work quite similarly to the ordinary Neural Networks as they are also made up of neurons with learnable weights and biases. The main difference is that the ConvNet architectures assume that the input will be images which results in certain techniques and properties that are used in the architecture. There are three main types of layers that are used to build a ConvNet architecture, namely **Convolutional Layer**, **Pooling Layer** and **Fully-Connected Layer**. These layers are stacked up on top of each other to create a full ConvNet architecture.

ConvNets work best for tasks related to image recognition. They are especially good with finding patterns. As the images have a certain pattern that we want our model to learn, that's why CNN model is a good fit for this. If our input data doesn't follow some pattern, then CNN isn't a good model for that problem.

Training an entire ConvNet from scratch is quite rare as it requires a sufficiently large amount of data. Because of this limitation, it is common to pretrain the ConvNet on a large dataset like ImageNet and then use that pretrained model and fine tune it for their specific needs. This is known as Transfer Learning. It is quite time and resource intensive task to train a CNN model from scratch, so this strategy works very well because most of the features that CNN model learns in the first few layers are common to every problem, like finding edges, corners etc.

We'll be using a VGGNet model that has been pretrained on the ImageNet dataset. VGGNet is the runner-up model in ILSVRC 2014 from Karen Simonyan and Andrew Zisserman. It showed that the depth of the network is a critical component in achieving good performance. The VGGNet model contains 16 CONV/FC layers and is composed of CONV layers that perform 3x3 convolutions and POOL layers that perform 2x2 max pooling. The downside of the VGGNet model is that it is more expensive to evaluate and uses a lot more memory for weights.

Here is the breakdown of a VGGNet model and the layers it contains:

INPUT: [224x224x3] memory: 224*224*3=150K weights: 0

CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K weights: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K weights: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K weights: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K weights: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K weights: 0

FC: [1x1x4096] memory: 4096 weights: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 weights: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Benchmark Model

According to this kaggle competition, the state of the art, in this kind of task 3 years ago, was considered to have an accuracy score above 80%. Now, by looking at the leaderboard on the kaggle competition, this accuracy level has increased quite a bit, in some cases, reaching to 97%, meaning that 97% of the time, the prediction that the model produces is the correct one. But for this project, we are using the LogLoss as our evaluation metric, so by looking at the leaderboard of the kaggle competition, a score of or above 0.3 would be acceptable.

Methodology

As I'm taking the deep learning course provided by **Fast.ai**, that's why some of my methodologies and techniques are inspired from there and I'm also using the **Vgg16.py** class that they provided. You can find the link to that class here. It is basically the same class/functionality that Keras provides but here, we are loading the weights that have been precalculated for us based on the Imagenet dataset.

Data Preprocessing

First of all, we separate the images into their respective directories according to their type, whether it’s a cat or a dog. All cats go into the cats directory and vice versa. Then we normalize our images using

```
vgg_mean = np.array([123.68, 116.779, 103.939], dtype=np.float32).reshape((3,1,1))
 def vgg_preprocess(x):
          x = x - vgg_mean
          return x[:, ::-1] # reverse axis rgb->bgr
```

We subtract the mean from the images because the original pretrained model used this for normalization and as we are using the pretrained model, so we need to normalize our images the same way they were normalized in the pretrained model. And then we convert from RGB to BGR because of the same reason, the pretrained model was trained using this.

Implementation

The architecture that I used to train the model is VGG16. At first, I created a VGG16 network architecture containing 5 Conv blocks where 2 of them contains 2 CONV layers and 3 of them have 3 CONV layers, along with a zero padding layer and 2x2 POOL layers. After that, two fully-connected layers of 4096 neurons were added with the activation function “relu” and Dropout was used with a probability of 0.5 to reduce the overfitting. In the end, a Dense layer was added with 1000 neurons and then “softmax” activation function was used. After that, we load the pre trained weights from the ImageNet model into our VGG16 model.

During fine-tuning the model, we remove the last fully connected layer from the pretrained model and add a new Dense layer at the end. We set all the previous layers to be non-trainable except the last layer so that it only learns the weights for the last layer. The optimizer that we used for our model is “Adam” and the learning rate was 0.01

After separating the cats and dogs to their respective directories, the data was loaded into the notebook as batches and then it was feed into the Vgg16 model, which was pretrained on the ImageNet database using weights from here. First I loaded the batches from the train dataset and the validation dataset and then I fine tuned the Vgg16 model on this dataset and compared it with the validation dataset. It gave me an accuracy score of **97%** which was quite good. On the sample dataset, I ran **10** epochs, the more epochs I ran the better it’s accuracy. Then I trained the model on the whole train dataset which gave me an accuracy score of about **98.20%**. Here, I ran only **3** epochs, because it takes a lot of time to run multiple epochs on a large dataset.

Refinement

At first, the batch size that I selected was 32 which didn’t perform that well as compared to the batch size of **64**. Because it takes a lot of time to test different hyperparameters on the whole dataset, that’s why I extracted a sample dataset from the training dataset which contains 200 images for training and 50 images for testing. I performed all the below hyperparameter tests on that **sample dataset** so as to get a general overview of what works best and what doesn’t.

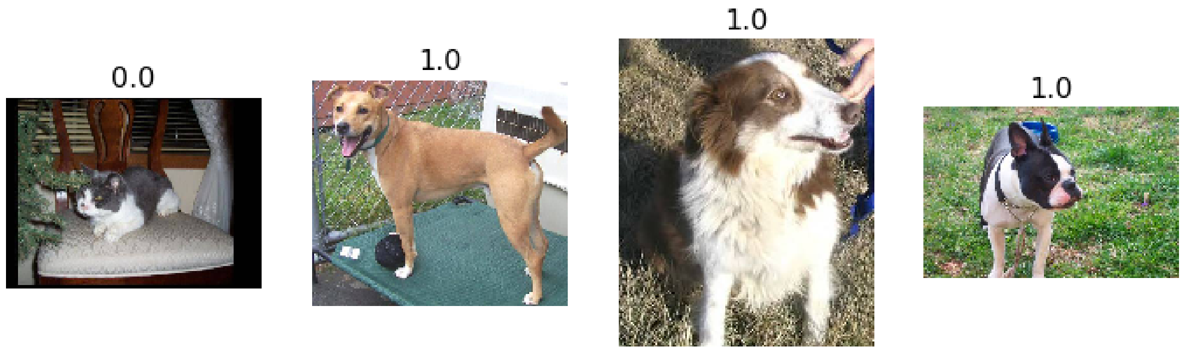
The accuracy of the learning rates 0.1 and 0.001 wasn't that good as compared to **0.01** that's why I chose 0.01 as the learning rate for my model. I tried different optimizers on the sample dataset, **SGD** performed better than **RMSProp** but **Adam** performed much better than the other two. The best accuracy score that I got from RMSProp was 86% while SGD surpassed it with 92% and Adam topped it with 98% on the sample dataset.

I also tried **InceptionV3** Model that the keras library provides with pre-trained weights from the Imagenet dataset. It didn't work as well as compared to the Vgg16 model. The best accuracy that I could get from the InceptionV3 model was **86%** on the sample dataset, while Vgg16 gave **98%** accuracy on the same dataset.

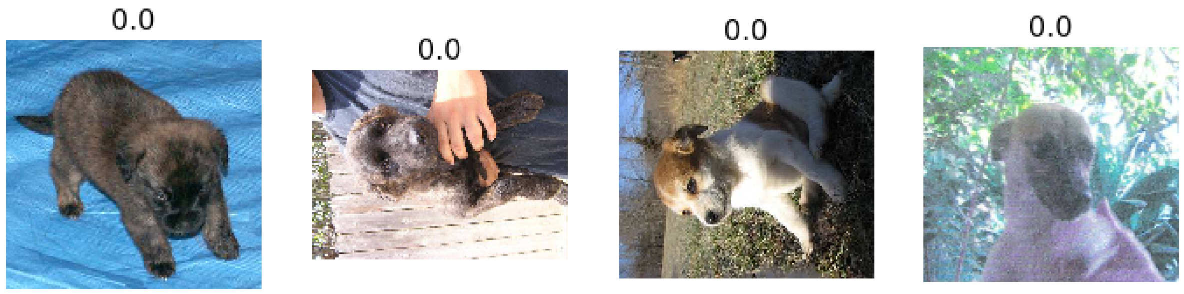
Results

After the model has been trained, we use the model to predict how well it performs on the test data. It returns a prediction array containing the value of prediction of each image that is how sure the model is whether a picture is of a dog or a cat. But we aren't evaluating the model based on the accuracy of predictions. We are evaluating our model based on the LogLoss value calculated by the Kaggle, so we create a submission according to the format specified by Kaggle. When we submitted the file on Kaggle, the score we got is **0.11278**.

To validate how correct our model is, we make predictions on the validation dataset so we can compare our predicted value with the actual value. Out of 2000 images, we got 1957 images correct. Here are some of those images, where 0 represents a cat and 1 represents a dog.

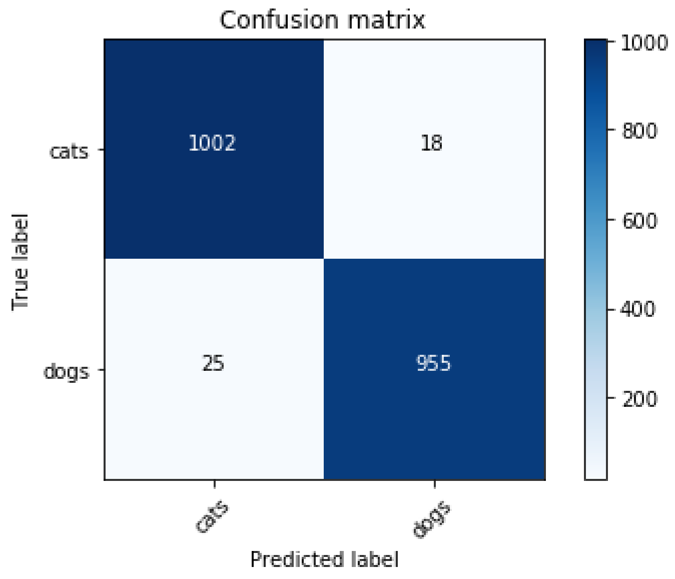


And the number of images that we predicted wrong is 43.



Conclusion

Here is the confusion matrix that was calculated for the validation data. We can see that our model works quite well with this dataset.



Reflection

This project was quite challenging as well as fun, considering my level at the time. After seeing the dataset provided and the problem that I had to solve, classify different categories of images, I knew a CNN was a good fit for this problem, given their popularity with this kind of problem domain, image classification.

First, find the dataset I need, download and process it. In this project, I chose the dataset from kaggle which collected a lot of daily images about cats or dogs. The preprocessing steps included dividing the images both in training set and validation set separately into two folders cat and dog according to their labels, resizing and rescaling the images.

Second, find the network which is suitable to solve this problem and build the classifier. For this problem, I tested Inception-V3 and VGG-16 models and selected VGG-16 and replaced the top layer to solve the problem in this project. Some little techniques like transfer learning are used to making improvements.

Then in the end, use the best model to make predictions.

The most interesting part of the project was hyperparameter tuning. Just changing a value slightly could make a huge impact on the model, like learning rates, optimizers etc. And it proved to be quite challenging too, because of the large dataset, it was really hard to test different hyperparameter values as it took a lot of time. To overcome this problem, I used a sample dataset which was extracted from the training dataset randomly and then I tried different hyperparameter values on that sample dataset which sped-up this whole process.

Improvement

I noticed that the more epochs one ran, the better the model became. But because of the size of the dataset, I couldn't do so as it takes a lot of time. And another thing I noticed is that some of the incorrectly labeled images are either too small or rotated. So, if we take this into account and create our model accordingly, like use data augmentation etc, we can further improve our model. We can also fine-tune one or more convolutional blocks to improve our model.