Machine Learning Engineer Nanodegree

Capstone Project

Kaggle Project - Dogs vs. Cats Redux: Kernels

Vijay Souri Maddila

August 20 2020

I. Definition

Project Overview

Convolutional neural networks have had a dramatic impact on the image recognition space. Even simple models are able to make highly accurate predictions on datasets like the Dogs vs Cats database .[1]

Recognition of images taken under real world conditions is a considerably more difficult problem than a data set taken in specific conditions. Differences in camera quality, lighting conditions, backgrounds, and angles all contribute to make the problem more difficult. And even more challenging is recognizing multiple objects and interpreting them as a single entity.

There are many uses for being able to classify similar shape bodies in the real world. This classification is regarded as the hello world of the image recognition world. The dataset was originally used as a CAPTCHA (or Completely Automated Public Turing test to tell Computers and Humans Apart), that is, a task that it is believed a human finds trivial, but cannot be solved by a machine, used on websites to distinguish between

human users and bots. Specifically, the task was referred to as "*Asirra*" or Animal Species Image Recognition for Restricting Access, a type of CAPTCHA. The task was described in the 2007 paper titled "Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization".

In this project we will use a simple CNN model to classify images of dogs and cats.

Problem Statement

The objective of this project is to develop and train a deep learning convolutional network that can distinguish between a cat and a dog in a given image. Before CNN was introduced this problem was solved with an accuracy of 80% with Support Vector Machine.

In this project we will train and test our model with the images from the Kaggle **Dogs vs. Cats Redux: Kernels Edition** data set.

We will design a CNN with tflearn. Using the tflearn library we will try to construct a Convolution Neural Network with a relu activation function and a maxpool layer after every convolution layer.

Metrics

The end evaluation metrics for this project is is done through Kaggle with :

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

Accuracy will be defined when an image is correctly labeled.

To gain confidence in our model will also be showing metrics when we try to fit the data to our model.

The evaluation can be seen happening as our model is training for each epoch. Furthermore the tensorboard can give a further detailed analysis.

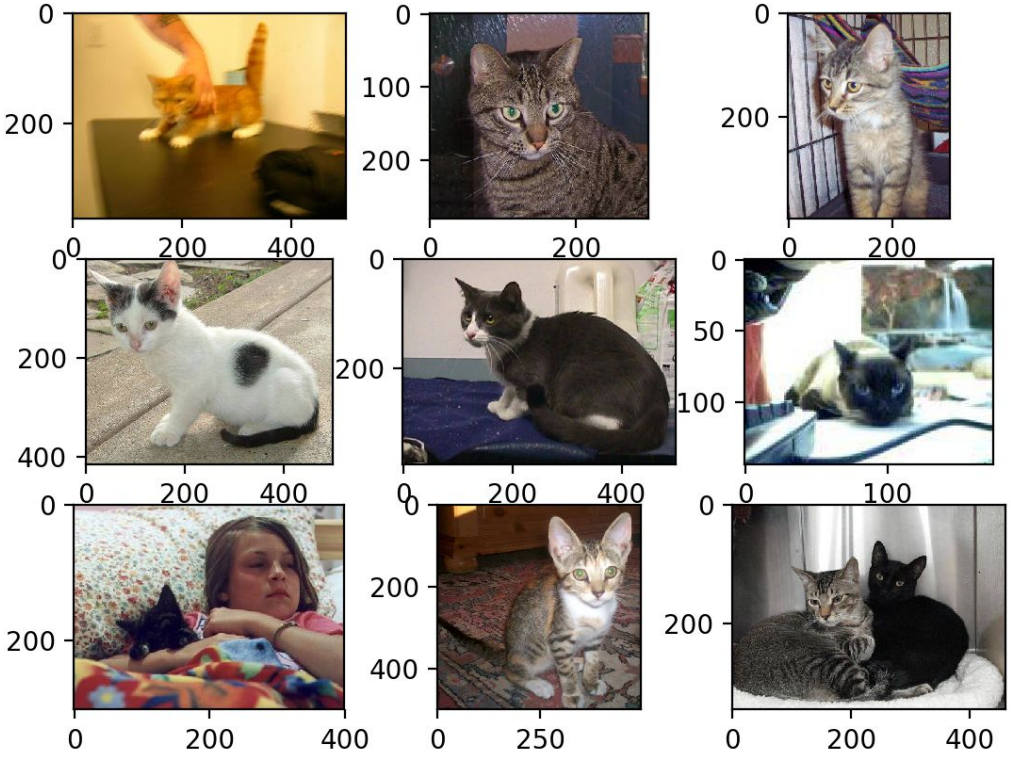
II. Analysis

Data Exploration

The file downloaded from kaggle gives us two folders namely ‘train’ and ‘test’. Opening these folders we will observe that the train folder contains 25,000 .jpg files of dogs and cats. The photos are labeled by their filename, with the word “*dog*” or “*cat*“. The file naming convention is as follows in the training set:

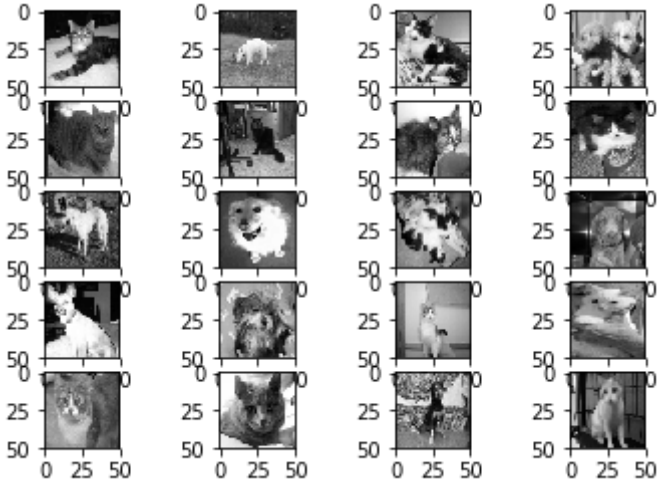
1. Cat0.jpeg
2. Cat1.jpeg and so on.

While outputting the raw images as a plot we see that the images are color and of different sizes. So to process the images accurately we need to set a standard image size. Let us take the standard of IMG_SIZE by IMG_SIZE for all images, where IMG_SIZE is a variable we are going to use for future calibration.



Exploratory Visualization

After downloading the data set and preprocessing it to use it in our development environment, we can begin to explore the data and modify it to better fit our end results.



In the above subplots we can see the end result of setting a standard image size to 50. This standard size will give us the ability to learn in a much better way because of the generalisation of image sizes.

Algorithm and Techniques

In this project we will be using CNN for classifying our dogs and cats images. So, in this section let us find out how a CNN works.

The CNN is similar to normal neural networks in the regards that they have learnable weights and biases. ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. In CNN we have local connectivity between the neurons which makes it possible for one neuron to connect to only a few neurons of the next layer. This makes

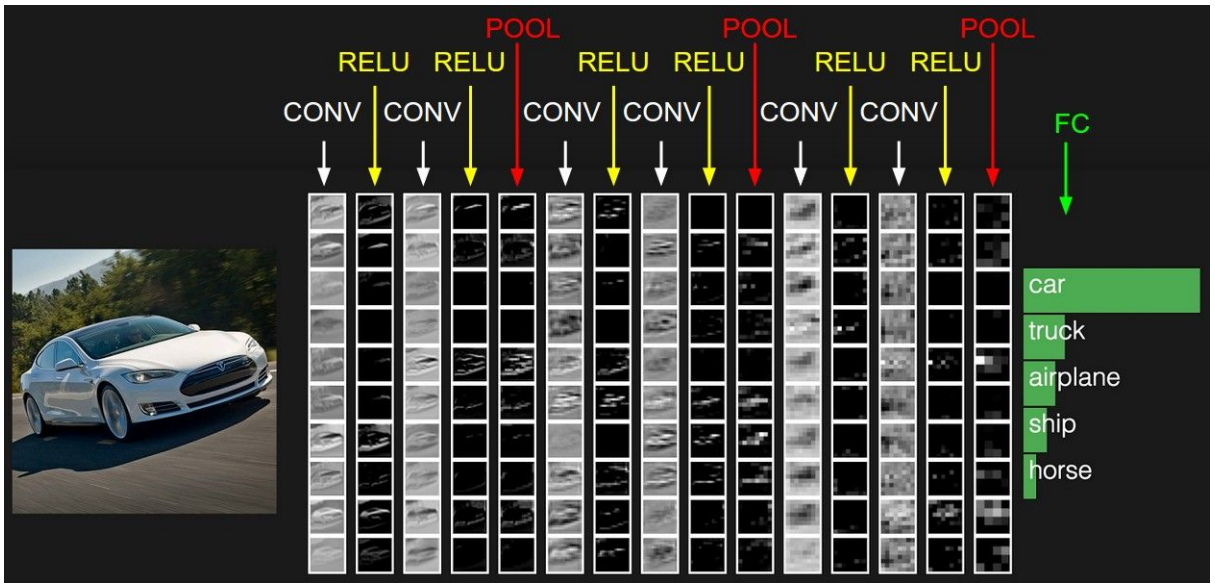
forward propagation a little less complex. In CNN parameters are also shared between layers. This reduces the total parameters in the net.

ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function.

There are three main layers in the CNN:

1. The convolution layer: This layer has a filter which separates the required feature for further computational use.
2. The pooling layer: This layer reduces the dimensionality for better computation.
3. The fully Connected layer: This layer is the final layer which gives us the desired output.

Now to realise a CNN in our project we are going to use tflearn library and build a custom convolution neural net.



We can use the above image as an example for CNN architecture.

Now to build a CNN we have to consider a number of factors such as :

* Number and types of layers. The more the number of layers in a model, the more is the complexity of the network. So, we will try to keep the number of layers as few as possible while not compromising the learning ability of the network.

* Activation functions. The ACtivation function for this network will be the RELU function because of its ability to increase non-linearity.

* Dropouts. Using dropout layers is a powerful way to prevent overfitting a model by randomly setting a fixed percentages of outputs to zero during training. This builds

robustness into the model by forcing it to build redundant representations because it cannot count on the presence of any given neuron to form its predictions.

* Optimizer:. We will use the Adam optimizer, a modern stochastic gradient descent optimizer that has shown itself to work well under a variety of different conditions. An Adam optimizer is able to adapt its learning rate based on moment to more quickly converge on a solution.

* Loss function. This is the objective that the optimizer will seek to minimize. Each of the output classifiers will seek to minimize the softmax cross-entropy between the predicted and expected values.

Benchmark

The Benchmark for this project will be the many number of submissions in the kaggle leaderboards for this challenge. The kaggle leaderboards can be accessed here.

In this leaderboard we observe the best score of 0.03302 log loss. And considering the top 20 submissions in the leaderboard we get an average score of 0.0379745. The score of the 50th rank of the leaderboard is 0.04842. Getting a score close to the top of the leaderboard can be considered to be a successful attempt.

III. Methodology

Data Processing

The Data from the train folder is processed through a function called create_train_data() where we will read the image and its corresponding label through cv2. The label is processed by splitting the name of the image with '.' .When reading the image through cv2 we will also be applying a size parameter of IMG_SIZE for standardising the image data. After reading the image and its label we will append the values into a numpy array. This process is done for all the images in the train folder. Now to keep the model from being biased we will shuffle the data with shuffle function from random library.

Next to visualize the extracted data we will plot the images with matplotlib.pyplot. We should make sure that the image data observed is uniform.

Now to process the test data we use process_test_data() function. Here we will read the image in a similar way with cv2 and also read the image number. We get the image

number by splitting the name of the file with '.' . Then both image and image number is appended to a numpy array.

To internally test and fit the data let us split the train data into two parts namely train and Validation sets. We give all the data except the last 500 images to train and only the last 500 images to validation set.

Implementation

Now we need to construct a CNN with tflearn. For this we need to import the following packages:

```
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
import tflearn.datasets.mnist as mnist
```

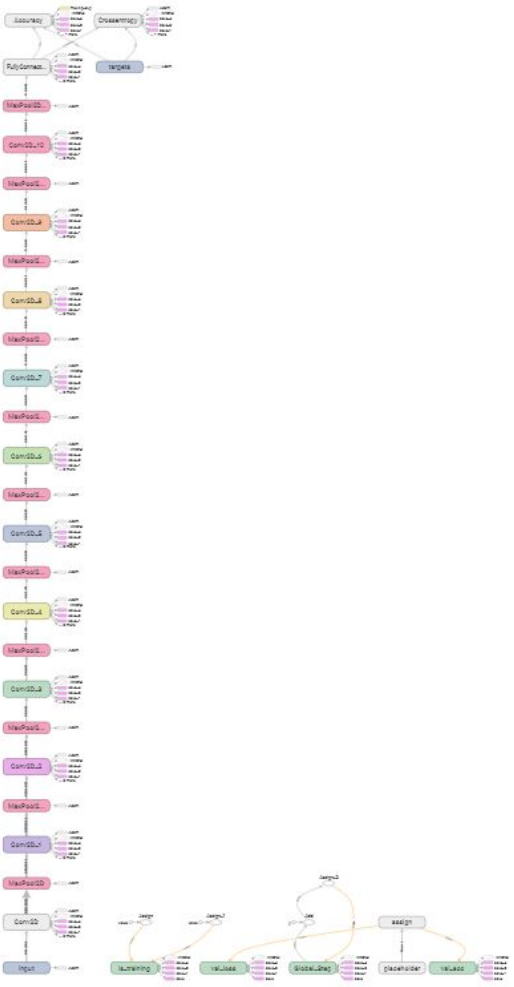
After importing all the necessary packages and libraries we need to define layers and all the aspects discussed in the Algorithm and Techniques section. We first define a convnet as "convnet = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')". Then creating layers can be described as:

```
convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)
```

In this we can observe that a convolution layer is immediately followed by a max pooling function with a filter size of 2. This keeps the dimensionality of the output volumes manageable.

The visualization of the model graph in TensorBoard, shown below, gives some indication of the complexity of the model we will be training.



A much clearer image of this above model visualisation is included in the submission.

Our model consists of twelve convolution layers. Each convolution layer consists of a 2x2 convolution filter with an RELU activation function. The depths of the filters are varying between 32 and 64. Each convolution is followed by a max pooling layer and a 2x2 filter. The conv layers are followed by a fc layer with a softmax activation function.

After defining the Network we have to fit the data to the model and begin the training.

While training the model we observe that the model starts out with less accuracy but as the training goes on we observe an improvement in the results.

```
model.fit({'input': X}, {'targets': Y}, n_epoch=7, validation_set=({'input': validation_X}, {'targets': validation_Y}), snapshot
```

```
Training Step: 5361  | total loss: 0.30514 | time: 25.542s
| Adam | epoch: 007 | loss: 0.30514 - acc: 0.8644 -- iter: 24448/24500
Training Step: 5362  | total loss: 0.30427 | time: 26.606s
| Adam | epoch: 007 | loss: 0.30427 - acc: 0.8608 | val_loss: 0.34472 - val_acc: 0.8400 -- iter: 24500/24500
```

Refinement

Tuning the learning rate turned out to have the most significant impact on the results of the project. We chose to use the Adam optimizer, a modern variant of stochastic gradient descent that is able to adapt the learning rate based on the momentum of the training.

The initial run with a learning rate of 1e-3 gave results bordering 80%. To increase the accuracy we changed the learning rate to 1e-5 which then gave an accuracy of 84%.

Upon further changing the learning rate to 1e-6 the performance dropped to lower 70%.

So the end learning rate is fixed at 1e-5 which gave better results and was faster.

IV. Results

Model Evaluation and Validation

The final evaluation at kaggle competition submission page gave a result of :

Submission and Description	Private Score	Public Score
submission-file_4.csv just now by VijaySouri Sub2	0.44135	0.44135
submission-file_2.csv 10 hours ago by VijaySouri upload7	0.44073	0.44073

This evaluation is based on log loss given by:

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

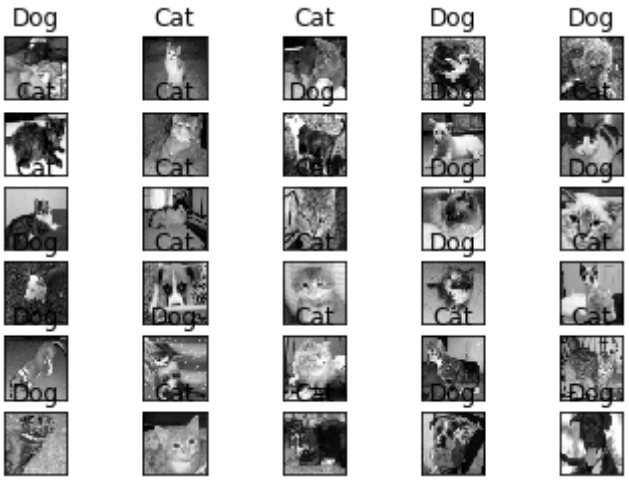
where

- 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

Justification

The accuracy of 84% and a score of 0.44 is well placed considering the simple CNN model used here.

V. Conclusion



This model can be effectively used to classify cats and dogs with an accuracy close to 85%. When Dogs vs Cats classification challenge was first introduced there was no concept of CNN and the best possible solution with SVM was able to yield an accuracy of 80%. This accuracy was enough to abolish using dogs vs cats from normal turing test for CAPTCHA. Considering an accuracy of 84% comparatively can also be used to validate whether similar CAPTCHA can be used or not.

While this may not be the ideal way to approach this problem, the idea and process through which this project took me through taught me a lot. This opens a new world where many other things are waiting to be classified to better serve our needs.

Real World Implementation

To test out the working of the model from outside the trained and tested data, we pick out a few pictures of cats and dogs from the web. These images are stored in a new directory with a number as their file name. For this test a total of 8 pictures are taken. Basing on these new pictures we again test the model.

Observing the outputs we can say that the model does a very good job of classifying the images even with difficult angles and shapes. Although the 2nd image is a cat it is classified as a Dog. This classification is hard even by human standards. So, we can now say with a definite confidence that this model can classify cats and dogs.

Furthermore these results showed us that the model is not biased to a particular data set and can be implemented in the real world to get dependable results.

Improvement

There are two major areas of improvement that we could explore moving forward. The first is improvements in the use of TensorFlow. The second is improvements to the architecture of the model.