Cats and Dogs Breed Detection

Machine Learning: SE843
Institute of Information Technology
University of Dhaka

Submitted to

Dr. B M Mainul Hossain
Associate Professor
University of Dhaka

Submitted by

Imam Hossain Kawsar	BSSE0816
Atiq Ahammed	BSSE0817
Md. Jewel Rana	BSSE0842

Table of Content

Abstract

We investigate the issue of categorizing fine grained objects to determine the animal breed from an image. In this project we used a data set of cats and dogs, 12 cat breeds and 120 dog breeds. The visual problem is very challenging as these animals, particularly cats, are very deformable and there can be quite subtle dif-ferences between the breeds.

We have done a number of tasks here: first we compare two classification approaches:a hierarchical one, in which a pet is first assigned to the cat or dog family and then to a breed, and a flat one, in which the breed is obtained directly.

When applied to the task of discrimi-nating the 132 different breeds of pets, the models obtain an average accuracy of about 59%, a very encouraging result considering the difficulty of the problem.

Introduction

This report is on object category recognition has largely focused on the discrimination of well distinguished object categories. This work concentrates instead on the problem of discriminating different breeds of cats and dogs, a challenging example of fine object categorization. The difficulty is in the fact that breeds may differ only by a few subtle phenotypic details that, due to the highly deformable nature of the bodies of such animals, can be difficult to measure automatically. The method used in [1] followed from this observation: a cat’s face was detected as the first stage in detecting the entire animal.Here we go further in using the detected head shape as a part of the feature descriptor. Two natural ways of combining the shape and appearance features are then considered and compared: a flat approach, in which both features are used to regress the pet’s family and the breed simultaneously, and a hierarchical one, in which the family is determined first based on the shape features alone, and then appearance is used to predict the breed conditioned on the family.

Datasets and evaluation measures

Kaggle Cats and Dogs Dataset

This dataset for Microsoft Research with over three million images of cats and dogs, manually classified by people at thousands of animal shelters across the United States. Kaggle is fortunate to offer a subset of this data for fun and research[2]. This dataset contains around 12500 images for each category. We split those into 80% training and 20% means test Split each class images into 10,000 for train and 2,500 for test.

The Oxford-IIIT PET dataset

The Oxford-IIIT PET dataset is a collection of 2371 images of cats where there exists 12 cat breed. Images are divided into training, valida-tion, and test sets, in a similar manner to the PASCAL VOCdata. The dataset contains about 200 images for each breed (which have been split randomly into 50 for training, 50 for validation, and 100 for testing). The dataset is available for download at [3].

Stanford Dogs Dataset

The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization. Contents of this dataset is 20 dog breeds ~150 images per class, and total images: 20,580 [4].

Evaluation protocol

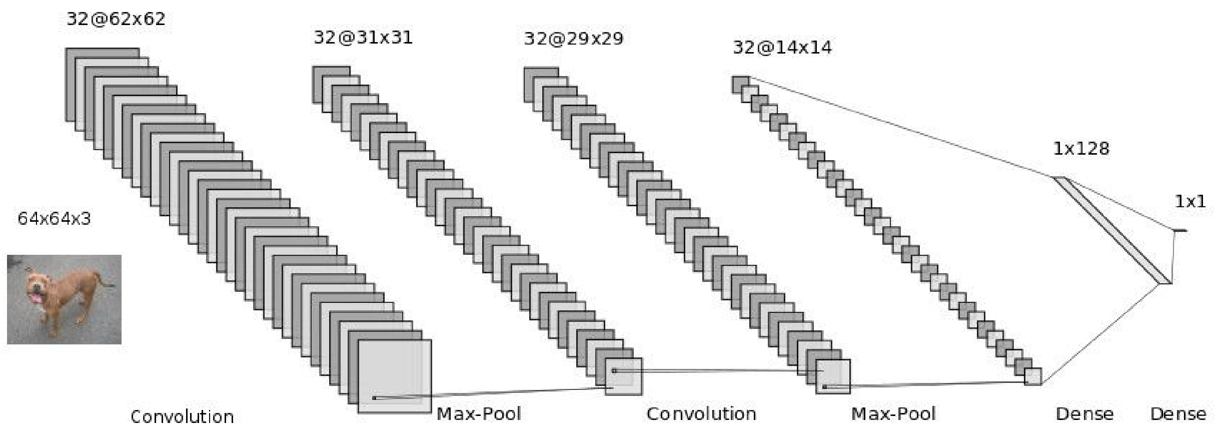
For making our Model we need to convert all the image in a specific format. We had to convert all the images into 64 * 64. To do so we used the following code snippet.

Architecture

In principle this architecture introduced the idea of applying several convolution and pooling layer before going to connect to a Neural Network and than to the outputs. Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

We use four main types of layers to build ConvNet architectures: Convolutional Layer Apply the ReLu (Rectified Linear Unit), Pooling Layer, Flattening, Fully-Connected Layer (exactly as seen in regular Neural Networks).

Images are made up of pixels. Each pixel is represented by a number between 0 and 255. Therefore each image has a digital representation which is how computers are able to work with images.



```
classifier = Sequential()
classifier.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2)) #if stride not given it
equal to pool filter size
classifier.add(Conv2D(32,(3,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2))
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
```

Convolution

A convolution is a combined integration of two functions that shows you how one function modifies the other.

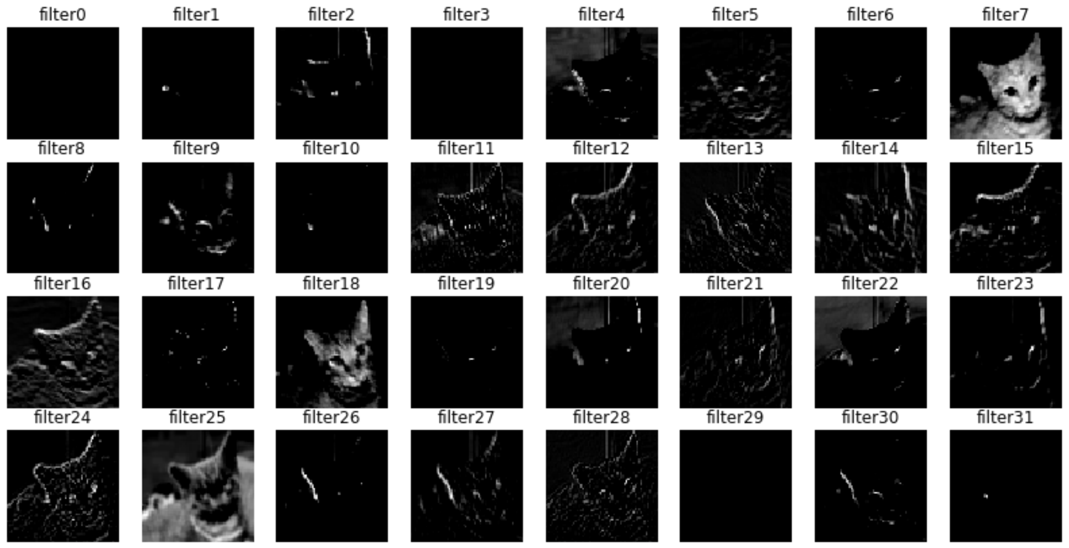
$$\begin{aligned}(f * g)(t) &\overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \\ &= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.\end{aligned}$$

Figure: The convolution function.

There are three important items to mention in this process: the input image, the feature detector, and the feature map.

The aim of this step is to reduce the size of the image and make processing faster and easier. Some of the features of the image are lost in this step.

However, the main features of the image that are important in image detection are retained. These features are the ones that are unique to identifying that specific object. For example each animal has unique features that enable us to identify it. The way we prevent loss of image information is by having many feature maps. Each feature map detects the location of certain features in the image.



Apply the ReLu (Rectified Linear Unit)

In this step we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other. Without applying this function the image classification will be treated as a linear problem while it is actually a non-linear one.

Pooling

Spatial invariance is a concept where the location of an object in an image doesn’t affect the ability of the neural network to detect its specific features. Pooling enables the CNN to detect features in various images irrespective of the difference in lighting in the pictures and different angles of the images.

There are different types of pooling, for example, max pooling and min pooling. Max pooling works by placing a matrix of 2x2 on the feature map and picking the largest value in that box. The 2x2 matrix is moved from left to right through the entire feature map picking the largest value in each pass.

These values then form a new matrix called a pooled feature map. Max pooling works to preserve the main features while also reducing the size of the image. This helps reduce overfitting, which would occur if the CNN is given too much information, especially if that information is not relevant in classifying the image.

Flattening

Once the pooled featured map is obtained, the next step is to flatten it. Flattening involves transforming the entire pooled feature map matrix into a single column which is then fed to the neural network for processing.

Full connection

After flattening, the flattened feature map is passed through a neural network. This step is made up of the input layer, the fully connected layer, and the output layer. The fully connected layer is similar to the hidden layer in ANNs but in this case it's fully connected. The output layer is where we get the predicted classes. The information is passed through the network and the error of prediction is calculated. The error is then back propagated through the system to improve the prediction.

The final figures produced by the neural network don't usually add up to one. However, it is important that these figures are brought down to numbers between zero and one, which represent the probability of each class. This is the role of the Softmax function.

Setup

In this step we need to import Keras and other packages that we're going to use in building the CNN.
Import the following packages:

- Sequential is used to initialize the neural network.
- Convolution2D is used to make the convolutional network that deals with the images.
- MaxPooling2D layer is used to add the pooling layers.
- Flatten is the function that converts the pooled feature map to a single column that is passed to the fully connected layer.
- Dense adds the fully connected layer to the neural network.

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

1) Initializing the neural network

To initialize the neural network we create an object of the Sequential class.

```
classifier = Sequential()
```

2) Convolution

To add the convolution layer, we call the add function with the classifier object and pass in Convolution2D with parameters. The first argument nb_filter. nbfilter is the number of feature detectors that we want to create. The second and third parameters are dimensions of the feature detector matrix.

It's common practice to start with 32 feature detectors for CNNs. The next parameter is input_shape which is the shape of the input image. The images will be converted into this shape during preprocessing. If the image is black and white it will be converted into a 2D array and if the image is colored it will be converted into a 3D array.

In this case, we'll assume that we are working with colored images. Input_shape is passed in a tuple with the number of channels, which is 3 for a colored image, and the dimensions of the 2D array in each channel. If you are not using a GPU it's advisable to use lower dimensions to reduce the computation time. When using a CPU, 64 by 64 dimensions performs well. The final parameter is the activation function. Classifying images is a nonlinear problem. So we use the rectifier function to ensure that we don't have negative pixel values during computation. That's how we achieve non-linearity.

```
classifier.add(Convolution2D(32, 3, 3, input_shape = (256, 256, 3),
activation='relu'))
```

3) Pooling

In this step we reduce the size of the feature map. Generally we create a pool size of 2x2 for max pooling. This enables us to reduce the size of the feature map while not losing important image information.

```
classifier.add(MaxPooling2D(pool_size=(2,2)))
```

4) Flattening

In this step, all the pooled feature maps are taken and put into a single vector. The Flatten function flattens all the feature maps into a single column.

```
classifier.add(Flatten())
```

5) Full connection

The next step is to use the vector we obtained above as the input for the neural network by using the Dense function in Keras. The first parameter is output_dim which is the number of nodes in the hidden layer. You can determine the most appropriate number through experimentation. The higher the number of dimensions the more computing resources you will need to fit the model. A common practice is to pick the number of nodes in powers of two. The second parameter is the activation function. We usually use the ReLu activation function in the hidden layer.

```
classifier.add(Dense(output_dim = 128, activation='relu'))
```

The next layer we have to add is the output layer. In this case, we'll use the sigmoid activation function since we expect a binary outcome. If we expected more than two outcomes we would use the softmax function.

The output_dim here is 1 since we just expect the predicted probabilities of the classes.

```
classifier.add(Dense(output_dim=1, activation='sigmoid'))
```

Tools and Technologies

For the completion of this project we take help from number of python libraries and tools. In this section, we will briefly discuss about dependencies.

Jupyter notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. We use the latest version of this tool.

Tensorflow 1.10

TensorFlow is an open source Deep Learning library developed by Google that is used to perform complex numerical operations and several other tasks to model Deep Learning models. It's architecture allows easy deployment of computations across multiple platforms like CPU's, GPU's, etc.

Python 3.6

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python 3.6.0 is the newest major release of the Python language, and it contains many new features and optimizations.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc.with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

Seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures. Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Scikit-Learn

Scikit-learn is probably the most useful library for machine learning in Python. It is on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Pandas

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas is the most widely used tool for data munging. It contains high-level data structures and manipulation tools designed to make data analysis fast and easy.

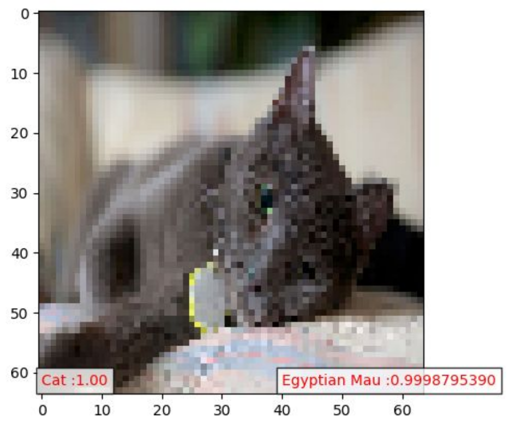
Numpy

NumPy is an open source Python package for scientific computing. NumPy supports large, multidimensional arrays and matrices. NumPy is written in Python and C. NumPy arrays are faster compared to Python lists. NumPy is a package in Python for Scientific Computing.

Family and Breed Discrimination

This section investigates classifying both the family and the breed. Two approaches are explored: hierarchical classification, in which the family is decided first and then the breed is decided. Out model provide 97% accuracy for classifying family for cat and dog. But it has a less accuracy for breed classification. For breed classification it provides average 59% accuracy.

As we have around 12,000 image for each of family classification. That provide a higher accuracy. But as we have ~150 image for each breed that's effect the breed classification accuracy.



Conclusion

CNN is a very fundamental deep learning technique. We tried to focus on the convolution and max pooling for our classification. We tried our best to implement this classification model with small knowledge on this topic within a short period of time. Hope that this would be the initial step to increase interest on this topic.

Reference

[1] O. Parkhi, A. Vedaldi, C. V. Jawahar, and A. Zisserman. Thetruth about cats and dogs. InProc. ICCV, 2011.
[2] https://www.kaggle.com/c/dogs-vs-cats/overview, last accessed: 1 November, 2019.
[3] O. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. TheOxford-IIIT PET Dataset.http://www.robots.ox.ac.uk/vgg/data/pets/index.html, 2019.
[4] http://vision.stanford.edu/aditya86/ImageNetDogs/, last accessed: 25 October, 2019.