Object Recognition using CALTECH 256 Dataset

Anubhab Majumdar
amajumd@ncsu.edu

Toshal Phene
tphene@ncsu.edu

Shubham Munot
samunot@ncsu.edu

1 Introduction

Object recognition is a quintessential machine learning problem of modern times. It is difficult for machines to recognize a large variety of objects in different conditions of lighting, occlusion and skew. In recent years, this field has seen huge leaps with machines becoming adept at object recognition. In this project, we explore different approaches of tackling this problem

We are using the CALTECH 256 dataset (link) for object recognition [1]. The dataset has 256 unique categories of images. The number of samples in each category varies from 80 to over 200. This dataset is quite a standard dataset in the field of object recognition and has been been used in important research papers like [2]. We experimented on a small subset (4-5 categories) of the dataset.

The experiments have been implemented using student version of MATLAB and python/tensorflow. Some of the less computationally intensive models are trained on personal computers, while the deep networks are trained on VCL servers.

2 Methodology

A typical classification system is shown in Fig.1. The paper focuses on the machine learning block shown in Fig.1. Different learning algorithms ranging from simple and lazy methods like KNN to deep convolution neural network is applied and results are compared for a comprehensive understanding of which method works better. The paper also tries to draw conclusion as to why certain methods work better than others.

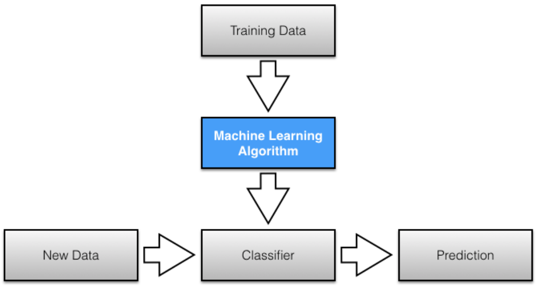

Figure 1: A typical prediction system

Before jumping into the algorithms applied, let's talk about the data itself. We have chosen the following 4 categories for our experimentation:

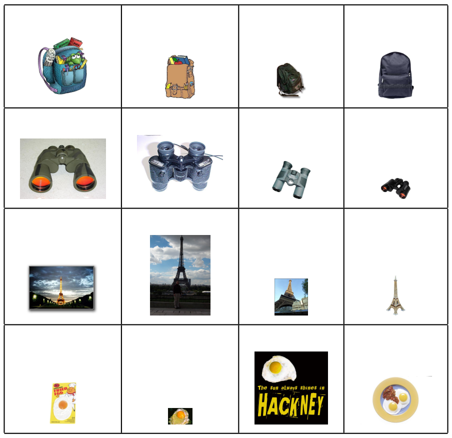

Figure 2: Few samples from our dataset

- Backpack (Sample size - 151)
- Binoculars (Sample size - 216)
- Eiffel Tower (Sample size - 83)
- Fried egg (Sample size - 90)

Sample of some images from the dataset are shown in Fig.2.

2.1 Baseline

For determining baseline, we have used the simple technique of majority class. As seen from above, among the 4 different classes we are experimenting with, binocular has the largest number of sample images. Therefore, given any image from the dataset, the probability of that image having a binocular is highest among the 4 classes. And this is what we use as baseline method.

2.2 KNN

Gabor filters are good models of how humans distinguish texture, and are therefore a useful model to use when designing algorithms to recognize texture. This example uses the basic approach described in (A. K. Jain and F. Farrokhnia, ”Unsupervised Texture Segmentation Using Gabor Filters”,1991) to perform texture segmentation.[4]

Design an array of Gabor Filters which are tuned to different frequencies and orientations. The set of frequencies and orientations is designed to localize different, roughly orthogonal, subsets of frequency and orientation information in the input image. Regularly sample orientations between [0,150] degrees in steps of 30 degrees. Sample wavelength in increasing powers of two starting from 4/sqrt(2) up to the hypotenuse length of the input image. These combinations of frequency and orientation are taken from [Jain,1991] cited in the introduction.[4]

Extract gabor magnitude features from source image. When working with Gabor filters, it is common to work with the magnitude response of each filter. Gabor magnitude response is also sometimes referred to as ”Gabor Energy”. Each MxN Gabor magnitude output image in gabormag(:,:,ind) is the output of the corresponding gabor filter g(ind).[4]

A nearest-neighbor classification object, where both distance metric (”nearest”) and number of neighbors can be altered. The object classifies new observations using the predict method. The object contains the data used for training, so can compute resubstitution predictions.

2.3 Convolutional Neural Network

Fully connected neural networks (NN) are not the most suitable architecture to classify images because such a network architecture does not take into account the spatial structure of the images.

The spatial location information of individual pixels are lost as soon as we transform it's 2D structure to one dimension to input into the neural network architecture. To solve this problem, LeCunn et. al. came up with convolutional neural network (CNN). CNN is a deep NN architecture that retains the location information of training data and is well adapted to classify images. Also, since all layers of CNN are not completely connected, we can train them faster than a fully connected architecture.

However, building and training a deep CNN architecture (ConvNet) is not easy because:

- There is no exact set of rules which determine how deep or how wide the network should be to produce good accuracy
- Small number of data samples per class in Caltech-256 dataset
- Huge computational cost of training a ConvNet

There are 4 main aspects of ConvNet that we need to determine:

- Training data
  - How large the training set should be? Should the training set size be increased by augmenting the available images?
  - What should be the resolution of each input image?
  - What should be the ideal batch size for training the network?
- Convolutional layers
  - What should be the depth of the network, i.e., how many convolutional layer should be present?
  - What should be the width of the network, i.e., how many features should be present per convolutional layer?
- Fully connected layer
  - What should be the size of the penultimate fully connected layer, i.e., how many neurons should it have?
- Prevent overfitting - CNN, like NN can easily overfit. How to prevent that?

Our focus in this paper is more on trying to understand how tweaking each of the above mentioned aspect affects the performance of ConvNet, rather than to try everything to somehow achieve good accuracy.

Since the number of samples per class is quite less, we decided to keep aside 20 samples from each class aside for test, and train our network with the remaining images. The batch size is kept 23 as there are 460 training images. We plan on staring with a simple 2 layer $lean$ model and tweak it's aspects in consecutive experiments and determine the affects. Till now we have trained 2 architectures. The first network we trained has the following architecture:

- Input image resolution - 28x28
- Convolutional layer 1 - 32 features extracted using 5x5 local receptive field with max pooling applied to 2x2 regions. Activation function used is rectified linear units (ReLU).
- Convolutional layer 2 - 64 features extracted using 5x5 local receptive field with max pooling applied to 2x2 regions. Activation function used is ReLU.
- A fully connected layer with 64 neurons with ReLU activation function.
- The output layer has 4 neurons denoting the 4 classes. The output is determined using softmax cross entropy function.

The second network we trained has the following architecture:

- Input image resolution - 28x28
- Convolutional layer 1 - 32 features extracted using 5x5 local receptive field with max pooling applied to 2x2 regions. Activation function used is rectified linear units (ReLU).

- Convolutional layer 2 - 64 features extracted using 5x5 local receptive field with max pooling applied to 2x2 regions. Activation function used is ReLU.
- A fully connected layer with 1024 neurons with ReLU activation function.
- The output layer has 4 neurons denoting the 4 classes. The output is determined using softmax cross entropy function.

The networks are created with python and tensorflow.

3 Experimental Results

3.1 Baseline

	Backpack	Binocular	Eiffel Tower	Fried Egg
Backpack	0	20	0	0
Binocular	0	20	0	0
Eiffel Tower	0	20	0	0
Fried Egg	0	20	0	0

3.2 KNN

	Backpack	Binocular	Eiffel Tower	Fried Egg
Backpack	12	5	0	3
Binocular	3	17	0	0
Eiffel Tower	3	5	9	3
Fried Egg	3	8	1	8

3.3 Convolutional Neural Network

	Architecture 1	Architecture 2
Training Set Accuracy	45.43%	79.78%
Test Set Accuracy	26.25%	27.5%

4 Conclusion

We are moving from simpler models to complex models and trying to compare results. We should expect a better accuracy percent as we move across them. Currently we have tried KNN and started on deep neural network. We plan to train SVM and a fully connected neural network also to have a better comparison during final presentation.

References

[1] Greg Griffin, Alex Holub and Pietro Perona.
Caltech-256 Object Category Dataset.

[2] Hao Zhang, Alexander C. Berg, Michael Maire and Jitendra Malik
SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition.

[3] Knuth: Computers and Typesetting,
`http://www-cs-faculty.stanford.edu/~uno/abcde.html`

[4] Texture Segmentation using Gabor Filters
`https://www.mathworks.com/help/images/texture-segmentation-using-gabor-filters.htm`

[5] Simona E.Grigorescu, Nicolai Petkov, and Peter Kruizinga
Comparison of Texture Features based on Gabor Filters.

[6] Anil K. Jain,Farshid Farrokhnia
Unsupervised Texture Segmentation Using Gabor Filters.