Intro. to Computer Vision

Final Project

Cats vs Dog Classification

Team Name: dogs_ftw
Joel Lat - 260580913
Muhammad Ammar Raufi - 260504960

Table of Contents

Introduction

Classification is a classic computer vision problem where one tries to teach a computer to identify what is inside a given picture. However a large barrier that implementations face is differing objects that are visually similar even for a human to detect. This project is an attempt to create a classifier that can determine the difference between a cat and a dog. This proves more difficult as the similar features/shape of the two types of animals make it more difficult to distinguish.

For this project we were given a training set of 5907 images, of which 4004 were dogs and 1903 were cats. The dogs and cats were of multiple breeds and the images could have multiple objects e.g people, toys, furniture etc. which makes the classification even more difficult. The classifier would be tested on a test set of approximately 1500 images.

We implemented different techniques including segmentation of images, multiple features, various classifiers and a deep learning approach.

Image Segmentation

Inside a majority of testing photos there is more than simply the animal inside the photo which can cause issues when we attempt to pull features. To circumvent this issue segmentation of the image was to used to separate the foreground animal from its background. This was implemented using openCV's grabcut algorithm.

One challenge while using the grabcut algorithm is that it requires the user to manually select the area of probable foreground and background. This is done by giving coordinates of a rectangle inside which the foreground is. Without this bias the algorithm tends to struggle in some cases removing vital sections of the image.

In order to do this we started by defining a rectangle near the edges of the image and dividing the image into 2 sections. The section from the boundary of the rectangle to the edge of the image was considered to be the background while the rest of the image was considered to be the foreground. Features were extracted from the foreground and the background and each rectangle boundary was moved inwards till the features converged. The features that we used were edge density and the mean hue, saturation and lightness values. This led us to have pretty good results in defining the rectangle. Some examples are shown in figure 1.

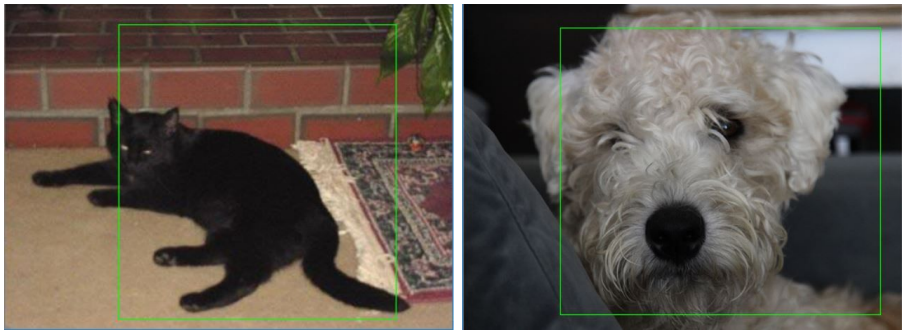

Figure 1. Found biassing rectangles

The grabcut algorithm was then applied and the segmentation results were quite promising for majority of the images. An example can be seen in figure 2. Grabcut was not successful for approximately 5% of the images where a completely black image was produced. These images were ignored when creating the classifier.

Figure 2. Original (left) and Segmented (right) Image

Classical Computer Vision approach

The Classical computer vision approach consists of manually defining and extracting features such as edges, corners, mean intensity values etc and then inputting them into some machine learning classifier.

Sift Features

In our implementation we utilized SIFT features to be used by our classifiers. This feature proved the most promising due to its ability to be consistent in varying scale, lighting and orientations. Other features such as intensity or edge detection could have trouble with classifying cats and dogs due to how similar the two families are. Both families had distinct ears, tend to be furry and come in similar colors causing a lot of features to overlap making this difficult.

As different images have different amount of features we utilized a bag of words implementation to create a feature “dictionary” for both cats and dogs. This was done by applying OpenCV’s BOW feature to add the Sift features of all the images and then calculating 500 cluster centers.
A histogram of each image was then created by calculating the euclidean distance of each sift feature from these cluster centers. This histogram was normalized so that the number of features would not affect the result. This histogram was then passed as features to the different classifiers.

Classifiers

We tested with 5 different types of classifiers available in the python Scikit-learn library to determine which would provide the best results. These included:

- Standard vector machine (SVM) with rbf kernel
- SVM with linear kernel
- Random forest classifier
- Adaboost
- Multi layer Perceptron (MLP)

In addition to each individual classifier we also looked at a voting systems that gave the output as the class with the maximum number of votes. On initial testing the SVM with rbf kernel and the Multilayer Perceptron proved to have the best accuracy.

We account this to the fact that the feature space we were using was quite large (at least 128 dimensions), alongside the the smaller sample size this is an area where a SVM excels giving us the best results.

Classifier Parameter Selection

Since the SVM and MLP were the most promising classifiers we wanted to optimize their hyperparameters to ensure the best results.

To do this for SVM, we implemented the GridSearchCV function in scikitLearn allowing us to test multiple parameter setups and see which returned the best results. The 2 parameters we focused on inside our grid search were C (The penalty parameter of the error term) and gamma (the kernel coefficient). These were identified as the 2 main factors that would aid in perfecting our classifier. From the results of the grid search we got that C=1000 and gamma = 0.0001 were the best parameters.

For MLP we looked at literature examples from people who had tried it before and in most cases it was said that one hidden layer with half the nodes of the number of features are the best parameters. Increasing the number of layers takes a longer time to compute and does not offer any significant advantages in accuracy. We tested around those numbers and did in fact get the most accuracy with them.

Results

As mentioned we attempted multiple types of classifiers and feature spaces with varying results. We also wanted to see whether our segmentation had helped or not, so we trained our classifier on both the segmented and unsegmented images. The ROC curve for SVM on segmented images and for unsegmented images in figure 3. The area under the curve for segmented images is 0.57 while for unsegmented images it is 0.65. This was a surprise to us as it showed that the segmentation actually made the results worse.

When we tested on the test images (for the classifiers trained on unsegmented images we did not segment the test images, while for classifiers trained on segmented images we first segmented the test images as well) we got very similar amount of accuracy for both the types of classifiers. These can be seen in figure 4

These accuracies were calculated by comparing predictions with a hand classified list which we made for the test images. This allowed us to repeatedly test different feature space and parameters without needing to submit to the Kaggle competition.

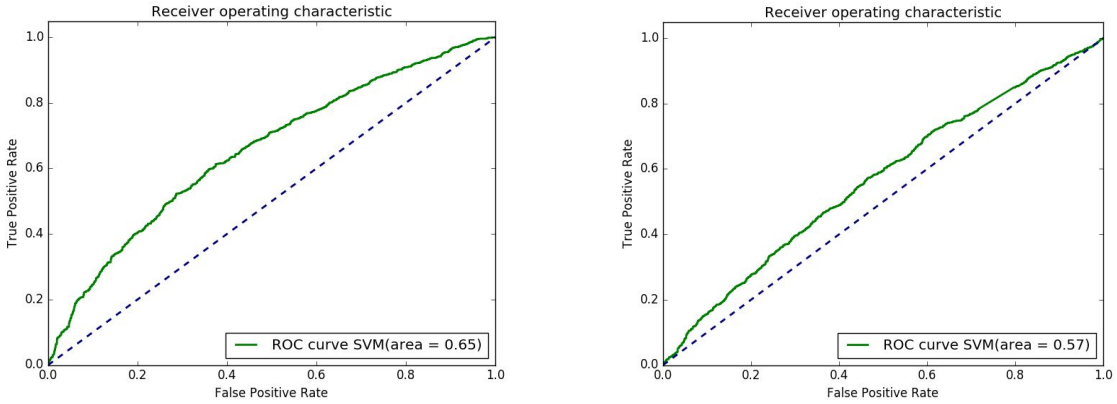

Figure 3. SVM ROC Curves. (left) unsegmented images, (right) segmented images

Classifier Type	**Accuracy Segmented (%)**	**Accuracy Unsegmented (%)**
Multilayer Perceptron Classifier	64	62
Standard Vector Machine	66	66
Random Forest	66	66
Adaboost	61	62
Voting	66	66

Figure 4. Testing Results based on X_Test Set

Deep Learning approach

We realized that utilizing the classical computer vision approaches discussed in class were proving to not provide enough flexibility to pass the 70% accuracy mark. Despite experiments of new feature spaces and classifier parameters we saw little to no improvements in terms of accuracy of the classifier.

In recent years the deep learning approach has gained popularity in image classification and has become the industry standard in most cases. This approach consists of defining a convolution neural network that itself defines features in the training set and uses them to make a classification.

Convolution Neural Network

A convolutional neural network (CNN) extends the MLP classifier in the scikitLearn library by adding convolutional layers inside the network. By adding these convolution layers the neural network is able to identify patterns found in the image giving us a more complex feature space. Over each convolutional layer we can identify more complicated features from a simple line in our first layer and hopefully feet, tails or eyes in a layer stage.

We implemented our CNN using google's open source TensorFlow library for machine intelligence. This library allows us to create the hidden layers inside our neural network and connect them accordingly. This gave us more control over the layers of the network and made implementation easier as there was a lot of documentation available online on how to do so.

Structure of Neural Network

For the structure of the CNN we made 4 convolution layer, 4 max pooling layers, 2 fully connected layers and a dropout layer to avoid overfitting, as shown in figure 5. We kept our batch size at 500 and and trained it for 100 epoch, which was where the accuracy had stabilised. The RelU activation function was used. The accuracy of the CNN over the 100 epochs can be seen in figures 6 and 7.

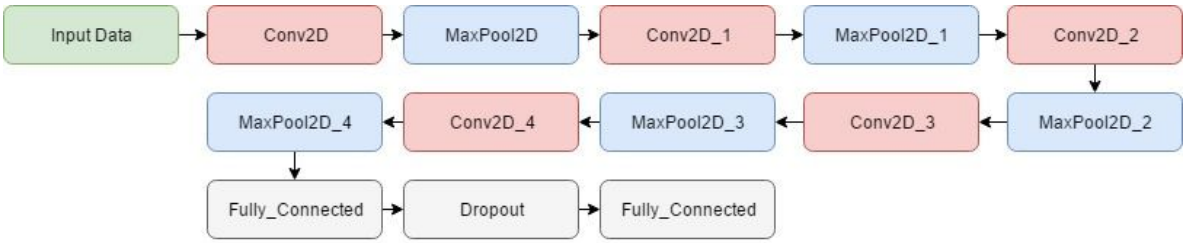

Figure 5. CNN structure

Results

We trained our neural network on both the segmented and unsegmented images. For the segmented images we got a training accuracy of 92% and a validation accuracy of 78%. When tested on the test set we got an accuracy of 76%.

Once again, when trained on unsegmented images, we saw better results. We got a training accuracy of 92% and a validation accuracy of approximately 83%. The graphs of the training over 1100 iterations can be seen in figure 6 and 7. When testing on the test set we got an accuracy of 81% which was much higher than what we got from any other classifier.

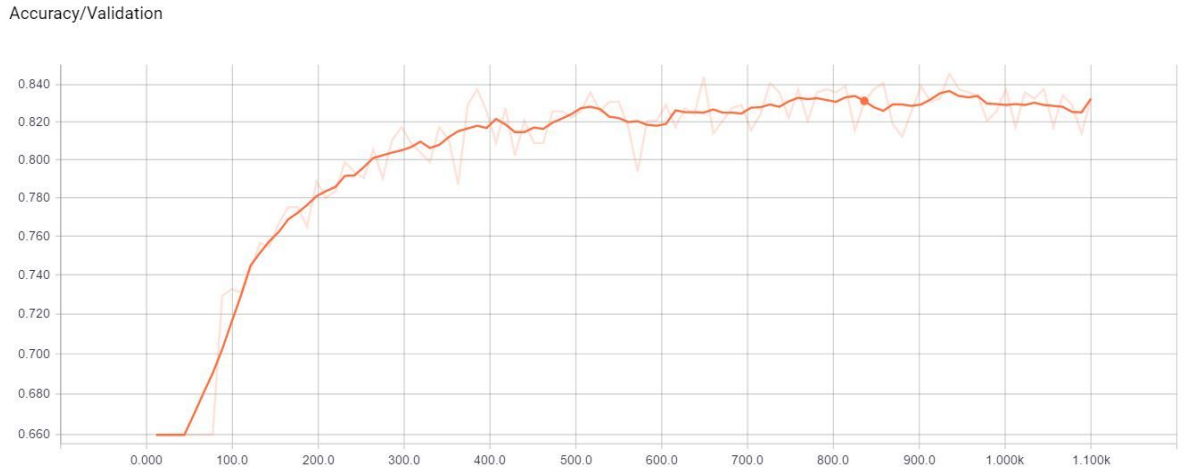

Figure 6. Validation Accuracy of CNN

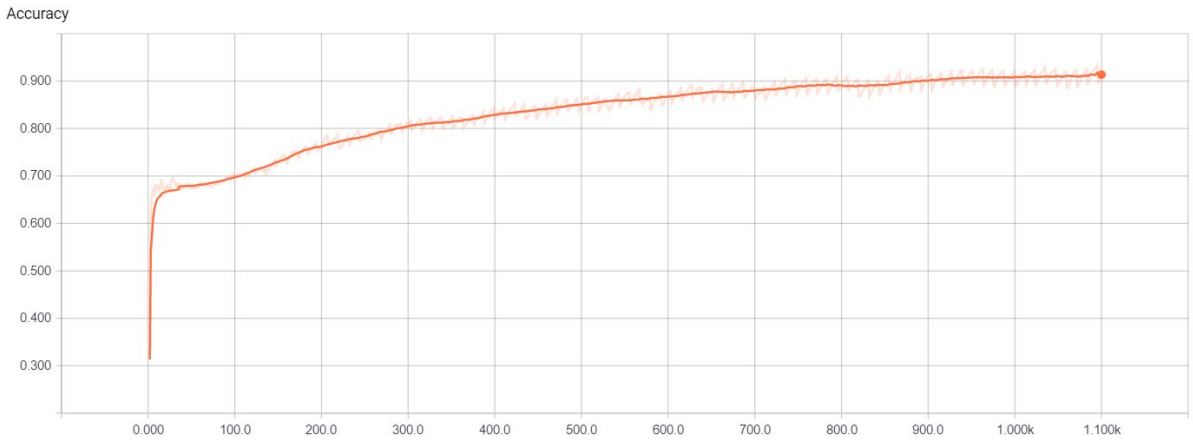

Figure 7. Training Accuracy of CNN

Discussion

Overall, our conclusion is that classical computer vision approaches are not good enough for such image classifications and will probably perform even more poorly in cases with more classes, though more data might help in improving classifier accuracy. A higher number of features and further optimization of classifier hyperparameters could have aso improved on the accuracy, but we doubt that it would meet the high levels of accuracy of CNNs.

The segmentation also did not help in improving the accuracy in either the classical or deep learning approach, which was a surprise for us. This could be partly because of the large black area in the image left after segmentation. Cropping the image after segmentation might be helpful in improving the accuracy.

We are satisfied with our resulting accuracy of above 80% of the CNN seeing as we only had 5000 images to train the classifier and CNNs usually require a large amount of data to have good accuracy and some of the images were also quite difficult to classify. That said, there is definitely still room for improvement. The main being that the parameters used to calibrate the classifier were not ideal and more testing could’ve yielded a better result.