CS696 Applied Computer Vision: Class Project

Manav Sanghavi
RedID: 822549761

Reeta Patil
RedID: 823351367

Chenwen You
RedID: 819791044

December 16, 2018

Abstract

Object Classification is one of the many challenging problems within Computer Vision. In this paper we present some research & experimentation using a variety of approaches to perform Object Classification using a database of fruits. We then go on to investigate how this can be expanded to perform Object Detection.

1 Introduction

Object Classification involves developing a system which would take an image as input and predict what kind of object it is, given a set of classes that the system already knows. For our experiments we used the Fruits 360 dataset from Kaggle [1]. A detailed description of the dataset is available in Section 4.1.

2 Task Description

The goal of this project is twofold: 1) to create a classifier that can classify an image of a fruit and 2) to try and use this classifier to perform object detection. A bigger focus is placed on creating a classifier than on object detection. The Object Detection experimentation information is summarized in Section 5.

3 Challenges and Solutions

There are a few challenges associated with trying to create a classifier for images. Images can be of varying sizes, scales, orientations, etc. The object itself can also be in a different position on the image. We will talk about how to address each of these items.

3.1 Scale

Scaling involves two images having the same object at different resolutions. This affects a classifier since an image of size 200x200 has 4 times more information than the image of size 100x100. Hence, if we try to run feature extraction without accounting for this scale change, we will get a lot more features from the first image than the second, thus making it very challenging to compare the 2 images. See Figure 1 for an example.

The simplest yet effective solution is to resize the 2 images to a standard size, like 50 x 50. This will ensure that the number of features that are extracted are almost, if not exactly, equal. For our experiments, all images were resized to be of size 45 x 45.

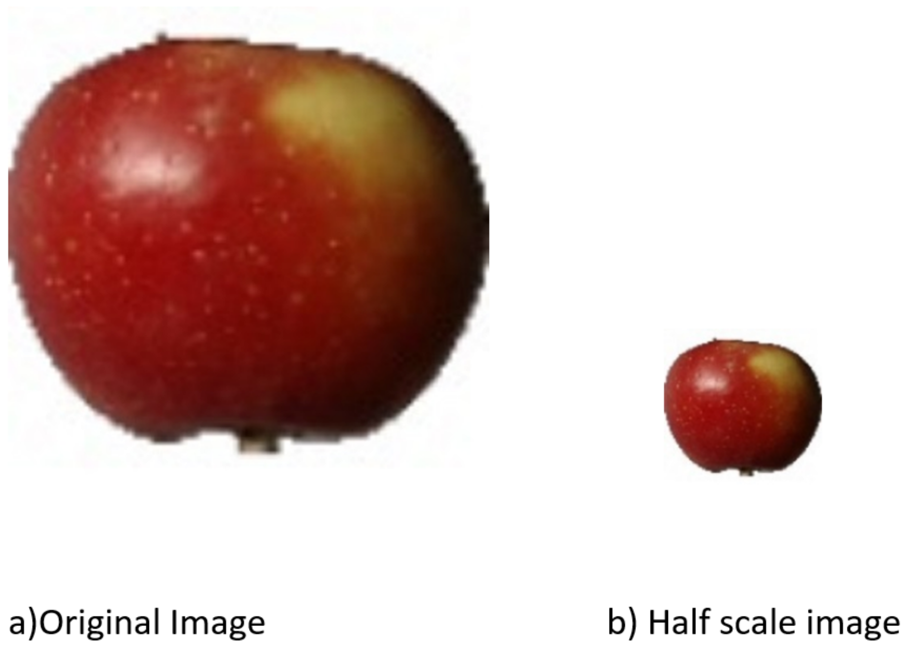


Figure 1: Scaling an object

3.2 Orientation

An object in an image can have multiple orientations. Consider a photo of an apple. The apple can be standing right side up, or it can be lying on its side, or it could be upside down. If we try to observe the location of the tip of the top of the apple, in the first case it will be very much near the top center of the image. In the second case it will likely be near the leftmost or rightmost edge whereas in the last case it will be found near the bottom center of the image. The fact that the same point of interest of the object lies in different parts of the image is an issue since it will give varying results during feature extraction. See Figure 2 for an example.

One way to solve this is to use rotation invariant features, like Histogram of Gradients. Another solution is to rotate the objects and train the system to recognize that all these orientation changes belong to the same class of object. In this experiment we have opted to train our system with multiple orientations of images.

3.3 Translation

When the same object lies in different positions on an image, that referred to as translation. See Figure 3 for an example.

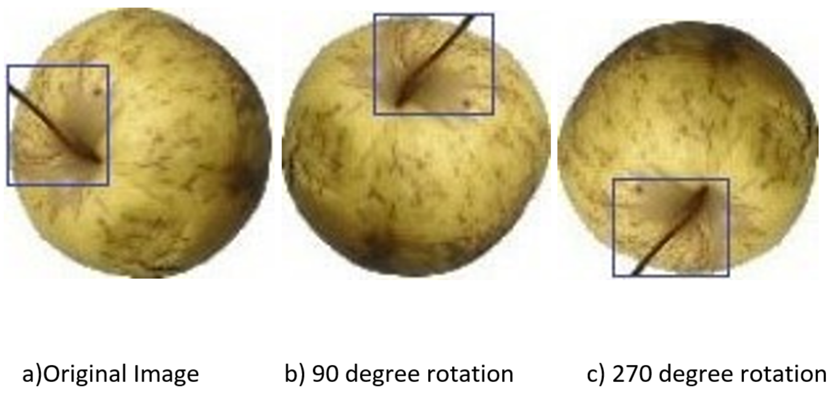


Figure 2: Rotating an object

Translation invariance can be achieved using a few different techniques. Firstly we can use a sliding window to find the object on all possible locations on the image. We can also use image segmentation techniques to find likely locations of an object, and then proceed to classify it. The image dataset that we have chosen to use is well made and hence all objects fit perfectly inside the image, with no possibility of translation. Hence we have no need to address this issue.

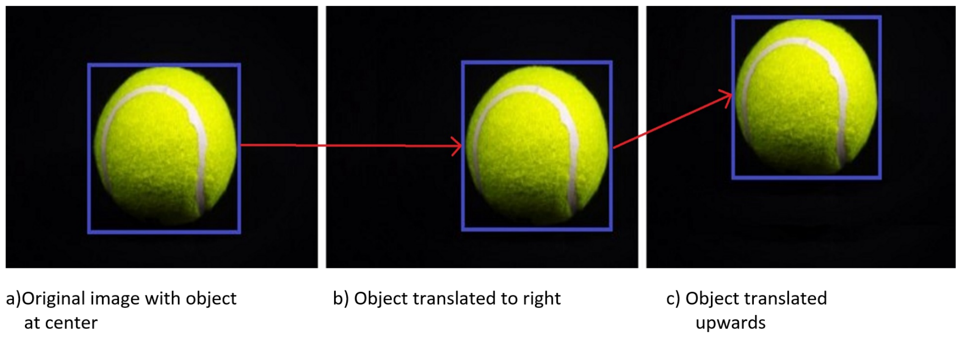

a)Original image with object at center
b) Object translated to right
c) Object translated upwards

Figure 3: Translation of an object

4 Experiments

There are a few popular ways of creating a classifier for images. For our experiment we focused on 2 broad subsets: Support Vector Machines (SVMs) and Neural Networks (NNs).

The most basic NN is a Deep Neural Network (DNN). It consists of fully connected layers. Images are structured spatially and there is information to be gained from how the pixels of the image are arranged with respect to each other. DNNs do not take this information into account. A better NN is a Convolutional Neural Network (CNN). CNNs can make use of these spatial structures, and hence perform better on image data. We created multiple

SVMs, DNNs, and CNNs with different hyperparameters. Details on how each type of model's hyperparameters were tuned is presented in Section 4.3. Then the best model of each kind was selected and tested on a testing dataset that was not used to train the models. The error rate on this was calculated and used to compare which type of model gave the best performance. As noted in Section 3.1, all models were trained by setting the image size to be 45 x 45 pixels.

4.1 Dataset Description

The Fruits 360 database from Kaggle [1] was used for this experiment. The dataset contains pictures of 81 different fruits. Each fruit was rotated slightly and a photo was taken from multiple such rotated angles. Two sets of such photos per fruit were taken: One by keeping the fruit standing upright, and another by keeping the fruit on its side. Some sample images are shown in Figure 4a. Note how the same fruit has had the same photo taken from multiple angles. This dataset was divided into training and test sets. Figure 4b shows some of the images used for testing.

4.2 Evaluation Metrics

All models were tested on a held out dataset to test the performance. The percentage of correct predictions was used as an accuracy metric. Since all models were tested on the same images, this metric allows us to compare all the models directly.

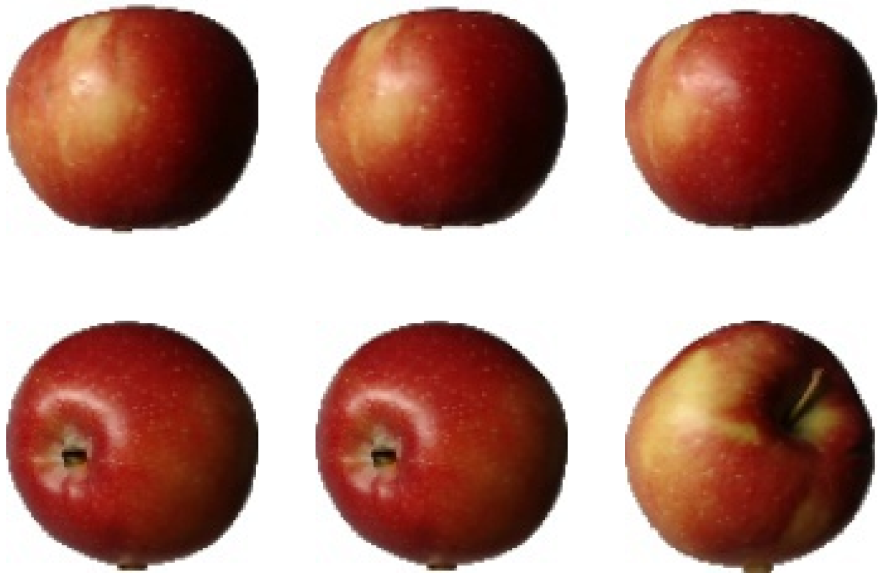
4.3 Tuning Models

SVMs [2], DNNs [3], and CNNs [4] are customizable and can be tuned to give better performance. Thus, to optimize the results, we have to find the best optimizing within each model before we can compare the different types of models. The next few sections describe what was tuned for each kind of model. All the results are detailed in Section 4.4.

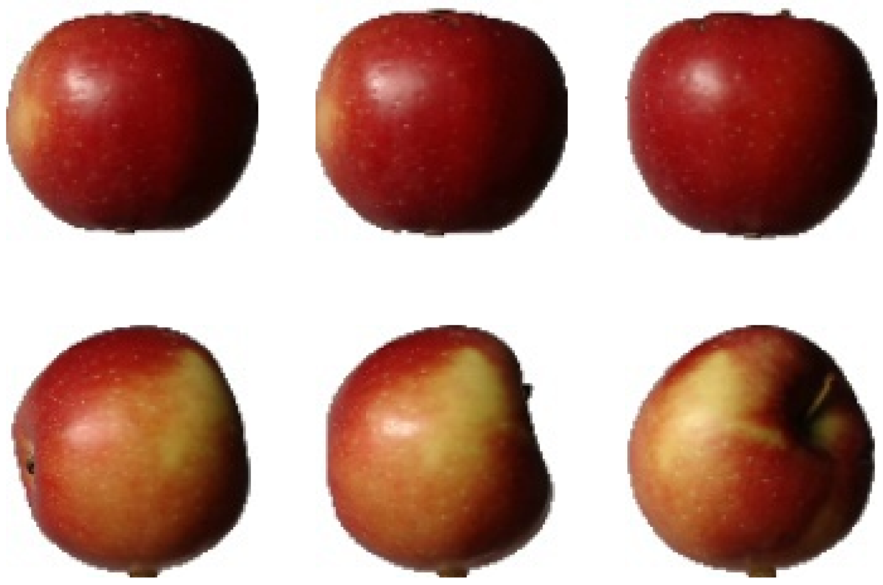
4.3.1 SVMs

For SVMs, we can modify which *kernel* is used. In the classification process, the data is dispersed which makes it difficult to separate the data into classes linearly. SVM uses the kernel function, which transforms the original data space into a higher dimension space where it would be more likely to find a good linear boundary. In this project, we have compared three kernels for classification:

1. RBF
2. Linear
3. 3rd Degree Polynomial



(a) Training Images



(b) Test Images

Figure 4: Images from the dataset. Top rows show images of the apples standing upright and the bottom rows show images of the apple on its side.

4.3.2 DNNs

DNNs can have multiple different hyperparameters. For this experiment, we chose to keep the activation function as ReLU. However, we varied the number of hidden layers in the DNN architecture. We fixed the width to be 90 nodes in each layer. L^2 Regularization was used in each layer, with α set to 0.01. A softmax output was used as the final layer. A batch size of 128 was used and all models were trained for 10 epochs. Adam [5] optimizer was used, and the cross entropy was chosen as the loss function.

4.3.3 CNNs

CNN architecture was fixed to have 2 convolutional layers, followed by Max Pooling, then a hidden Dense layer before a softmax output layer. However, the width of the hidden convolutional and dense layers was varied. Regardless of the width, all models had a stride of 3 in the convolutional layers, and max pooling was applied on a 2 x 2 window. L^2 Regularization [6] with $\alpha = 0.01$ was used in the hidden dense layer. Dropout [7] was applied with a dropout rate of 25% between the max pooling and dense layers, and with a rate of 50% between the hidden dense and output layers. Similar to DNNs, 128 was set as the batch size & models were trained for 10 epochs using the Adam optimizer on the cross entropy.

4.4 Results

Table 1 shows the accuracies of different SVM kernels. Clearly, the RBF kernel had the best results.

Table 1: Accuracy for different SVM kernels

Kernel	**Accuracy (%)**
Polynomial with degree 3	14.46%
Linear	41.24%
RBF	**51.74%**

Table 2 shows the performance of different DNN architectures. We see that the 3 layer architecture gave the best performance.

Table 2: Accuracy for different DNN architectures

Number of hidden layers	**Accuracy (%)**
3 layers	**43.37%**
4 layers	25.3%
5 layers	9.74%

Lastly, Table 3 shows the accuracy of different CNN architectures. The format of the architectures is as follows: the first number shows the number of nodes in the first convolutional layer, followed by the number of nodes in the next convolutional layer, and the last number is the number of nodes in the hidden dense layer.

Table 3: Accuracy for different CNN architectures

CNN Architecture	Accuracy (%)
16-32-64	79.14%
32-64-128	82.58%
64-128-256	**84.83%**

Figure 5 shows the error rates of the different models. We can see that while the SVM and the DNN had similar performance of near 50% accuracy, the CNN had great accuracy.

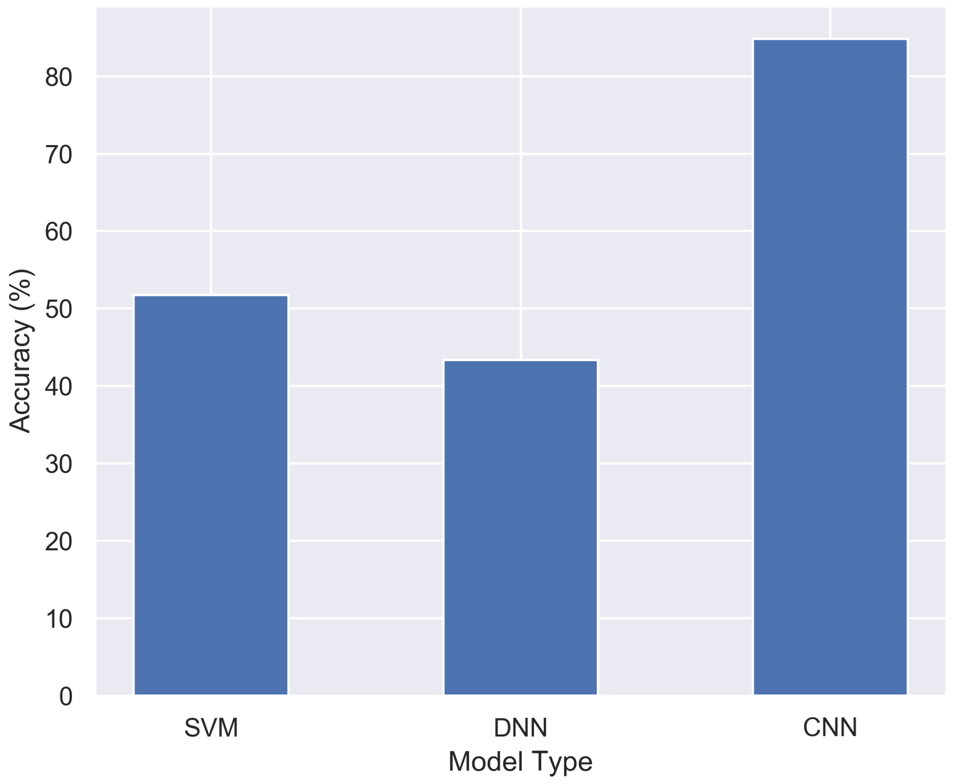


Figure 5: Comparison of models

4.5 Analysis

We see that the CNNs give great performance, with SVMs and DNNs giving respectable but ultimately poorer performance. However, CNNs and DNNs have a huge advantage over SVMs as they are a lot quicker to train. A single SVM takes hours to train, whereas a CNN or DNN can be trained in a matter of minutes. This is done by a GPU, with which we can parallelize the training of the model leading to great reductions in training time.

An important point of observation for DNNs is how adding layers *decreased* performance. This is because of model capacity: we need to choose appropriate capacity for the amount of available data. This shows that choosing appropriate model capacity is an important point of consideration for neural networks.

From the CNN results we can see that increasing the capacity of the model improved the results, though not by much.

5 Object Detection

The best CNN model was used to perform Object Detection, with a pipeline similar to R-CNN. Selective Search was performed on an image, then the resulting region proposals were fed to the trained CNN for classification. The result is shown in Figure 6. Unfortunately the results were poor. Time and computational constraints meant that we could not train a model large enough to accommodate all of the rotational variances of all the fruit images, leading to poor results.

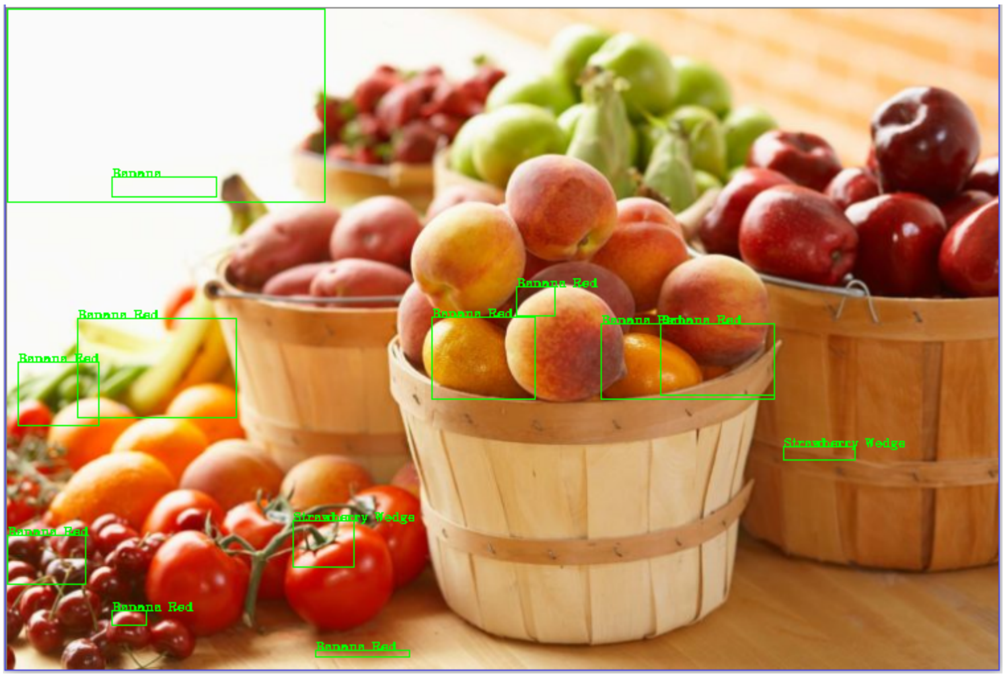


Figure 6: Object Detection Results

6 Conclusion and Future Work

In this project we have seen how to perform image classification on a dataset of fruits by using 3 different kinds of classifiers. We saw that CNNs gave the best overall performance. CNNs and DNNs were a lot quicker to train and the CNNs' accuracy means they would be more favourable to use for a classification task than any other model. We also saw the CNN model used for Object Detection and found that it would need more time and computational power to get good results.

References

[1] Fruits 360 Database. Available at `https://www.kaggle.com/moltean/fruits`

[2] K. Murphy, *Machine Learning: a Probabilistic Perspective, Chapter 14.5*, available at `https://www.cs.ubc.ca/ murphyk/MLbook/`.

[3] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning, Chapter 6*, available at `https://www.deeplearningbook.org/contents/regularization.html`.

[4] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning, Chapter 9*, available at `https://www.deeplearningbook.org/contents/regularization.html`.

[5] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980*, available at `https://arxiv.org/abs/1412.6980`.

[6] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning, Chapter 7.1*, available at `https://www.deeplearningbook.org/contents/regularization.html`.

[7] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.