ROBOTICS & CV

Final Project

Niral Shah

Professor K. Dana | 12/14/16

I. Introduction:

Over the past several weeks, an effort was made to understand the computer vision concept of appearance based recognition. Various appearance based recognition techniques were studied, notably Bag of Words, Principal Component Analysis (PCA) and Deep Learning. Furthermore, this project was an excellent foray into the world of machine learning, specifically supervised learning. Supervised learning is a form of machine learning where training data is used to give the algorithm data to associate a label with, this represents the supervised learning step. Subsequently, test data is passed in which will compare specific features of this test data to the training set to make a classification decision. The accuracy of this classification is measured by comparing the output label of the algorithm to the actual label associated with specific test datum.

In this project, three different types of appearance based recognition algorithms were written. In part one of the project, a Bag of Words object recognition algorithm and PCA Facial Recognition algorithm were written in MATLAB. For the Bag of Words object recognition algorithm, the *Caltech 101 dataset* was utilized. The goal was to identify ten classes of objects with as high accuracy as possible. The ten classes include *a camera, accordion, revolver, computer keyboard, pizza, saxophone, stop-sign, strawberry, and a tennis ball*. In the remaining portion of the report, I will discuss and compare the data and performance of the implemented Bag of Words algorithm to the two pre-trained CNNs, ImageNet-VGG-F and Imagenet-Resnet-101-Dag.

II. Methods:

a. Bag of Words

The Bag of Words image recognition algorithm is a specific form of supervised machine learning. Bag of Words relies on a training data-set of numerous images from each class that the algorithm needs to identify. The first step is to identify interest points in each image and their associated (128 x 1) descriptors. The interest points and descriptors are calculated with a scale invariant feature transform (SIFT). Interest points by definition are points that are extrema of its 26 neighbors in scale space. Around each key interest point, a 16x16 region is considered and the gradient orientation and magnitude is calculated at each pixel. In each 4x4 sub-window a histogram is calculated that places the orientation in 8 bins. As a result for each interest point there is a 128x1 descriptor.

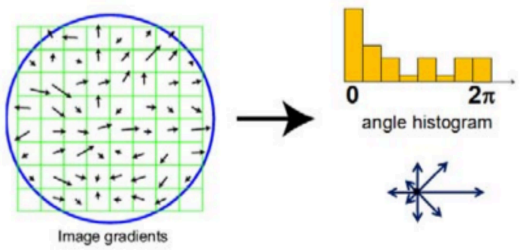


Figure 1:
In SIFT, the gradients of each pixel are computed the orientation of each gradient is put into a histogram with 8 bins from 0 to 2pi.

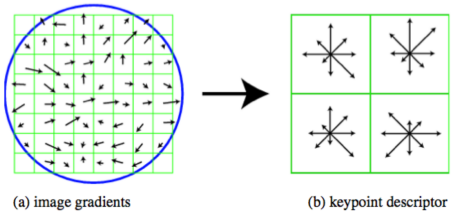


Figure 2:
The figure above shows how a histogram is computed for each 4x4 sub-region in a 16x16 window. To build a 128x1 descriptor for each interest point in the image.

In some way each of these interest points and their corresponding descriptors are like visual words that can be combined into a dictionary. Bag of Words follows this logic by taking a subset (e.g. 20%) of the training data set to build a Visual Word Dictionary with size K determined by the user. In the current implementation, K was set to be 100. The selection of K was purely empirical, based on which value maximized the accuracy of the results. It was observed that for low values of K, such as K = 20 and 50 there were a high number of misclassifications especially for a large set of data like the one used with 500 images. This makes sense since the number of visual words is limited. It was found that for K = 100 the accuracy was the best. For greater values of K it was found that the accuracy either didn't improve materially or got worse as the number of visual words increased significantly. This is also makes sense since an increased number of words increases the spread of the histogram and may lead to over fitting.

Moving on, the first step in the Bag of Words Algorithm is to iterate over the chosen subset of training data set to compile a global matrix of descriptors. In order to avoid over-fitting to the training set and to make the visual word dictionary more general, K-means is used

to reduce the numerous number of visual words to simply K centroid visual words. At this point the Visual Word Dictionary is built. To verify that this dictionary is correctly built, the k-distinguishable colors method in MATLAB was used to represent each visual word as a color. Each interest point in the training images was marked with a specific color. This was visually inspected to verify that similar features were being marked with the same visual word. See figures 3 and 4, notice for example that the pink markers correspond to dark features in the images. For the soccer ball these mean the dark patches, and the same is true for a black and white chessboard.

Figure 3:

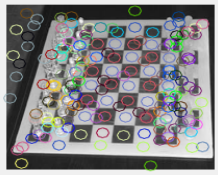
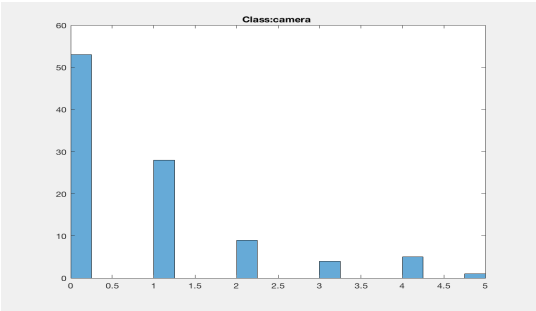


Figure 4:

Figures 3 and 4 show how visual words in the dictionary (marked by a color scale from 1:K) are identified on the training images. The accuracy of Bag of Words can verified if similar features in different images are marked with the same type of word.

At this point, a visual word dictionary is built. However, with a visual word dictionary alone, the algorithm does not have enough information to make classification decisions about the test data. In many ways this is analogous to simply knowing the letters of the alphabet without knowing the spellings of any words. Thus the next step is to identify visual words that belong to specific classes of images and build a histogram to capture the frequency of each of the K words that appears for each image in the training set and associating it with the correct class label. In order to build these histograms nearest neighbors and a hard-weighting scheme was used. For each training image the interest points and descriptors were calculated using SIFT. Subsequently the nearest neighbors algorithm was used for each descriptor to find the closest centroid or visual word (by Euclidean distance) for the descriptor. This determines what bin number it will belong to and subsequently increment the frequency of the occurrence of the visual word. This step was verified by comparing the histogram for images within the same class and dissimilar images. See figures 5 and 6.



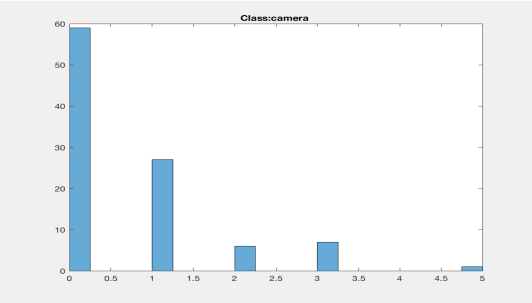


Figure 5: (above) note how the histograms or visual words are similar two different images of a camera. Word on the x and the frequency on the y axes.

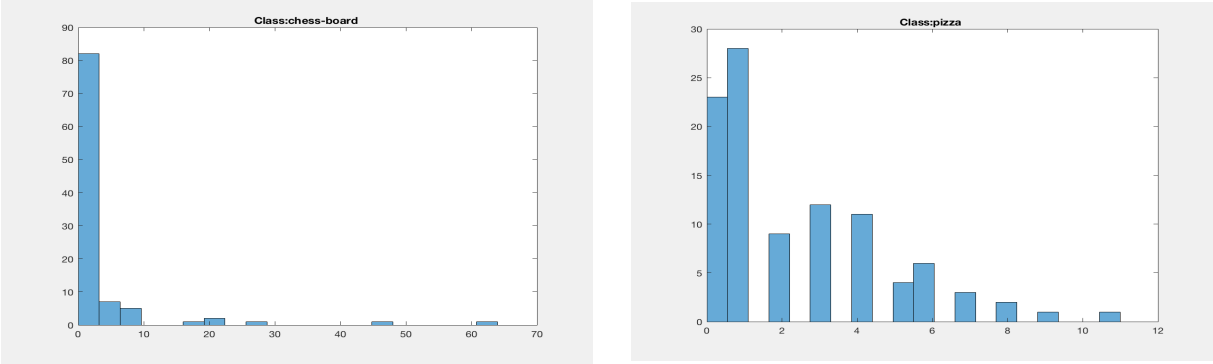


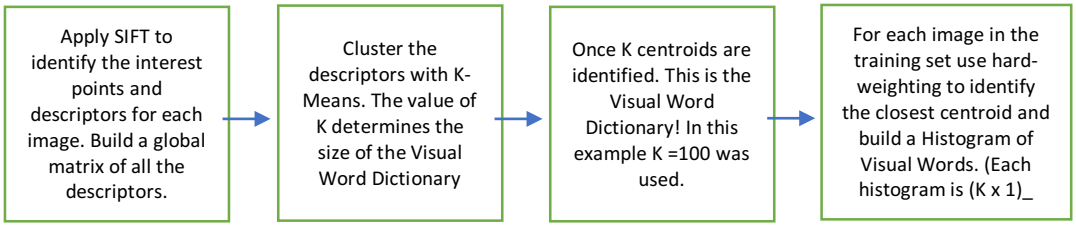
Figure 6: (above) compare the histogram and frequency of words in the camera class to that of the chess-board and pizza objects.

Finally, now that a histogram for each image in the training set is constructed, we can now move on to the classification step. The classification of test data follows a similar process to the construction of histograms for the training set. For each image in the test data nearest-neighbors and hard-weighting is used to find the closest centroid by euclidean distance for each descriptor corresponding the interest points in the image. Once a unique histogram for the image is constructed, nearest neighbors is used again to find whether a similar image exists in the training-database. Once it finds the most-similar histogram in the training data it applies that training image's label to the test image.

Bag of Words works fairly well because it takes advantage of the likelihood for similar features to exist in various different classes and the spatial locality of these features. As a result, the algorithm is very straightforward to implement.

Here's a flow-chart to summarize how the BOW Algorithm works:

BOW Training Step:



BOW Classification:

BOW's Discussion and Results:

Legend:

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Camera	Accordion	Revolver	Comp. Keyboard	Pizza	Sax	Stop Sign	Soccer Ball	Strawberry	Tennis Ball

Let's take a look at five splits of the data and analyze specific statistics, comment on the results, and discuss any anomalies.

Split 1: (normalized confusion matrix)

```
confusionMatrix =
    0.3333    0.0667    0.2000         0         0    0.0667    0.1333         0    0.1333    0.0667
    0.0667    0.6000    0.0667         0         0    0.0667         0    0.0667    0.1333         0
    0.0667         0    0.6667         0         0    0.1333    0.0667         0    0.0667         0
         0    0.1333         0    0.7333         0         0         0    0.0667         0    0.0667
    0.0667         0    0.0667         0    0.6667    0.0667    0.0667    0.0667         0         0
    0.2000         0    0.1333         0    0.0667    0.4000         0    0.0667         0    0.1333
    0.1333         0    0.0667         0         0         0    0.7333         0         0    0.0667
         0         0         0         0         0    0.0667         0    0.8000    0.1333         0
    0.0667         0    0.0667         0         0         0         0         0    0.8667         0
         0         0    0.2000         0         0    0.1333         0         0         0    0.6667
```

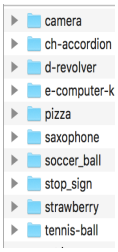
Split 2:

```
confusionMatrix =
    0.1333    0.1333    0.1333         0         0    0.1333    0.2000         0    0.0667    0.2000
    0.0667    0.8000         0         0         0         0         0    0.0667         0    0.0667
         0         0    0.8000         0    0.0667    0.0667         0         0         0    0.0667
    0.1333    0.1333    0.0667    0.6000         0         0         0    0.0667         0         0
         0         0    0.0667         0    0.7333    0.1333         0         0    0.0667         0
    0.0667         0    0.0667         0    0.0667    0.7333         0         0         0    0.0667
         0         0    0.0667         0         0         0    0.8000         0    0.0667    0.0667
         0    0.0667         0         0         0         0         0    0.8667         0    0.0667
    0.0667    0.0667         0         0         0         0         0         0    0.8000    0.0667
         0         0         0         0    0.0667    0.1333         0         0    0.0667    0.7333
```

Split 3:

```
confusionMatrix =
    0.2667    0.1333    0.4000         0         0         0         0    0.0667    0.1333         0
    0.1333    0.7333    0.0667         0         0    0.0667         0         0         0         0
    0.0667         0    0.6000         0         0    0.0667         0         0    0.1333    0.1333
         0    0.1333         0    0.7333         0         0         0         0         0    0.1333
    0.0667         0    0.0667         0    0.6000    0.2000         0         0         0    0.0667
         0         0    0.1333         0    0.0667    0.5333         0         0    0.0667    0.2000
         0         0    0.2000         0         0         0    0.5333    0.0667    0.0667    0.1333
    0.0667         0         0         0         0         0         0    0.7333    0.0667    0.1333
         0    0.0667    0.0667         0         0         0         0         0    0.8667         0
    0.1333         0         0         0         0    0.0667         0         0    0.1333    0.6667
```

Split 4:




```
confusionMatrix =
    0.1333         0    0.4667         0         0         0    0.2667         0    0.0667    0.0667
    0.0667    0.7333    0.1333         0         0         0         0         0         0    0.0667
         0         0    0.6667         0         0         0    0.1333         0         0    0.2000
         0    0.0667         0    0.8667         0         0         0    0.0667         0         0
         0         0         0         0    0.8667    0.1333         0         0         0         0
    0.2000    0.1333    0.1333         0         0    0.4667         0         0         0    0.0667
    0.0667         0    0.0667         0         0         0    0.7333         0    0.0667    0.0667
    0.2000         0         0         0    0.0667         0    0.0667    0.6667         0         0
    0.0667         0    0.2000         0    0.1333    0.0667         0    0.0667    0.3333    0.1333
    0.0667         0    0.1333         0         0         0    0.2000         0         0    0.6000
```

Split 5:

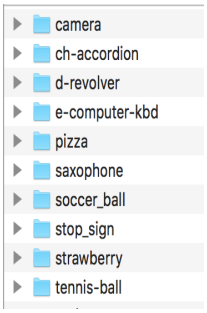

```
confusionMatrix =
    0.2667         0    0.2000         0         0    0.2667    0.0667         0    0.0667    0.1333
    0.1333    0.7333         0         0         0         0         0         0    0.0667    0.0667
         0         0    0.8667         0         0         0         0         0         0    0.1333
         0    0.2000         0    0.7333    0.0667         0         0         0         0         0
    0.1333         0    0.0667    0.0667    0.4667    0.0667    0.1333         0         0    0.0667
    0.0667         0    0.0667         0         0    0.5333    0.0667         0         0    0.2667
         0         0         0         0    0.0667         0    0.6667         0    0.1333    0.1333
    0.0667         0         0         0         0         0    0.0667    0.7333    0.0667    0.0667
    0.0667         0         0         0         0    0.1333    0.1333         0    0.6000    0.0667
    0.0667         0    0.0667    0.0667         0         0    0.0667         0         0    0.7333
```

Looking at just the confusion matrices it is evident the camera class did very poor in recognition with only 26.67% getting correctly recognized. While on the other hand, strawberries on average had the highest recognition rate. However it can be observed that the recognition rates varied significantly over every split. This suggests that the accuracy of the results are highly dependent on the training data. If a split contained training data that poorly represented the images in the test data, recognition rates can be very low. As a result for classes like strawberries the recognition rates varied from 33.33% to 86.67%! The next section takes a look at further statistics of the confusion matrix.

Discussion & Interpretation of Data

Statistics & Discussion:

Accuracy:



	1
1	0.8733
2	0.9400
3	0.8867
4	0.9733
5	0.9600
6	0.8867
7	0.9467
8	0.9533
9	0.9400
10	0.9333

Average Accuracy: 92.93%
Split 1

```
acc =
    0.8800
    0.9400
    0.9400
    0.9600
    0.9533
    0.9267
    0.9600
    0.9733
    0.9533
    0.9133
```

Average Accuracy: 93.399%
Split 2

```
acc =
    0.8467
    0.9533
    0.8533
    0.9867
    0.9667
    0.9267
    0.9067
    0.9533
    0.9200
    0.9000
```

Average Accuracy: 92.93%
Split 3

```
acc =
    0.8800
    0.9400
    0.8667
    0.9733
    0.9533
    0.9133
    0.9533
    0.9600
    0.9267
    0.8867
```

Average Accuracy: 92.134%
Split 4

```
acc =
    0.8733
    0.9533
    0.9467
    0.9600
    0.9333
    0.9067
    0.9133
    0.9733
    0.9267
    0.8800
```

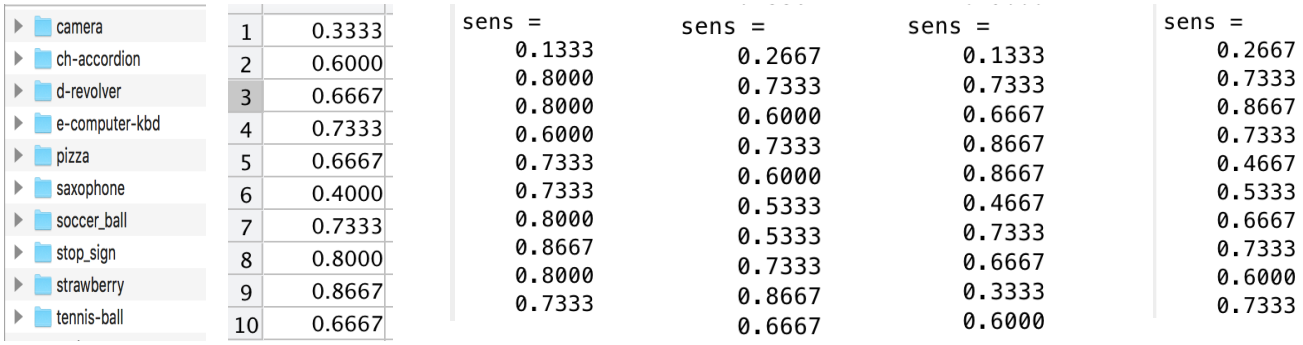
Average Accuracy: 92.063%
Split 5

Average Accuracy/class across 5 splits: 92.8094%

Each row in the accuracy matrix shows how many images were accurately classified in the class. The figure shows two interesting details. The camera class had the lowest accuracy. This maybe due to the wide variation of many different types of cameras and makes. Similar arguments can be made for class 3, the revolver and class 6, the saxophone. The one common denominator in this is that there are numerous variations and potentially sub-classes that belong to cameras, revolvers and saxophones.

Sensitivity:

$$\text{sensitivity} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false negatives}}$$



Sensitivity is a measurement of how good a test is at detecting the accurate results. The results in sensitivity mirror the results we see in accuracy. The camera class and saxophone class had a hard time recognizing images of their own class. Again, this is most likely due to the internal variation of the images within the class. Not all saxophones and cameras share the same features. Compare this to a stop-sign, soccer ball or strawberry which by and large all look the same. Generally it appears that bag of words does not have a high sensitivity or ability to detect accurate results.

Specificity:

$$\text{specificity} = \frac{\text{number of true negatives}}{\text{number of true negatives} + \text{number of false positives}}$$

	1
1	0.9333
2	0.9778
3	0.9111
4	1
5	0.9926
6	0.9407
7	0.9704
8	0.9704
9	0.9481
10	0.9630

```
specfs =
    0.9630
    0.9556
    0.9556
    1.0000
    0.9778
    0.9481
    0.9778
    0.9852
    0.9704
    0.9333
```

```
specfs =
    0.9481
    0.9630
    0.8963
    1.0000
    0.9926
    0.9556
    1.0000
    0.9852
    0.9333
    0.9111
```

```
specfs =
    0.9259
    0.9778
    0.8741
    1.0000
    0.9778
    0.9778
    0.9259
    0.9852
    0.9852
    0.9333
```

```
specfs =
    0.9407
    0.9778
    0.9556
    0.9852
    0.9852
    0.9481
    0.9407
    1.0000
    0.9630
    0.8963
```

Specificity is a measure of how likely the algorithm is to avoid making a false classification. An interesting anomaly is that for almost all of the splits, except for split five, for computer keyboards none of the other classes were misclassified as a keyboard. This is most likely due to the presence of text and letters on the keyboard which easily gets picked up by SIFT as interest points. Interestingly, while sensitivity for bag of words was relatively low, it appears that specificity is high. This suggests that the algorithm does a good job making false classifications for images that actually belong to the dataset.

Precision:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

1
0.3571
0.7500
0.4545
1
0.9091
0.4286
0.7333
0.7500
0.6500
0.6667

```
prec =
    0.2857
    0.6667
    0.6667
    1.0000
    0.7857
    0.6111
    0.8000
    0.8667
    0.7500
    0.5500
```

```
prec =
    0.3636
    0.6875
    0.3913
    1.0000
    0.9000
    0.5714
    1.0000
    0.8462
    0.5909
    0.4545
```

```
prec =
    0.1667
    0.7857
    0.3704
    1.0000
    0.8125
    0.7000
    0.5238
    0.8333
    0.7143
    0.5000
```

```
prec =
    0.3333
    0.7857
    0.6842
    0.8462
    0.7778
    0.5333
    0.5556
    1.0000
    0.6429
    0.4400
```

Precision is a measured of how many images of a particular class were correctly classified along with the total number that exist out of all the ones that were classified as an image. Evidently it can be seen that computer keyboard correctly classified all of its images. Thus it seems images of other classes were also incorrectly classified and brought down the accuracy rate. In the same turn it can be seen that camera had a very poor precision rate, thus many cameras were most likely mis classified as other objects. Overall bringing down accuracy.

Recall:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

	1
1	0.3333
2	0.6000
3	0.6667
4	0.7333
5	0.6667
6	0.4000
7	0.7333
8	0.8000
9	0.8667
10	0.6667

```
recall =
    0.1333
    0.8000
    0.8000
    0.6000
    0.7333
    0.7333
    0.8000
    0.8667
    0.8000
    0.7333
```

```
recall =
    0.2667
    0.7333
    0.6000
    0.7333
    0.6000
    0.5333
    0.5333
    0.7333
    0.8667
    0.6667
```

```
recall =
    0.1333
    0.7333
    0.6667
    0.8667
    0.8667
    0.4667
    0.7333
    0.6667
    0.3333
    0.6000
```

```
recall =
    0.2667
    0.7333
    0.8667
    0.7333
    0.4667
    0.5333
    0.6667
    0.7333
    0.6000
    0.7333
```

Recall is a measurement of the number of correctly classified images out of the total number of images that were classified. (This is the diagonal of the confusion matrix). As it can be seen again camera had the lowest recall, suggesting that it was significantly misclassified. While most strawberries were correctly classified and few other objects were classified as strawberries.

b. Deep Learning w/Pre-trained Convolution Neural Networks (CNNs)

A neural network typically describes something called the single hidden layer back propagation also known as a single layer perceptron. Neural networks are simply a composition of linear and nonlinear models that can be used for regression and classification. Neural networks must be a motley of both linear and nonlinear models otherwise if it just had linear models it would collapse into simple linear classifier, which does not serve the purpose. In recent years, Deep Learning has become a popular buzzword. Though deep learning just involves learning at a deep number of layers, which for image recognition allows for invariance that can help to recognize more complex images (e.g. those with background scenes, etc..)

Convolution Neural Networks (CNN's) are a particular type of neural network that integrates additional data structures, convolution, back-propagation, and something called a loss-layer. For deep learning, CNN's are unique in that the data is greater than two dimensions. Typically the data takes the form of MxNxK where M and N span space and K spans the number of channels. Furthermore, the use of convolution means that each layer input x_i has a convolutional type structure, where it local and translation invariant.

$$y_{i'j'k'} = \sum_{ijk} w_{ijkk'} x_{i+i',j+j',k}$$

Figure 1: A linear layer
(simply covolution of filter banks)

$$y_{ijk} = \max\{0, x_{ijk}\}.$$

Figure 2: A nonlinear layer
(Rectified Linear Unit)

As discussed, Convolutional neural networks are a composition of linear and nonlinear functions. An example of a linear layer of the neural networks is convolution of several banks of filters (See figure 1). Subsequently non-linear functions are added in between the layers. A simple non-linear function would be a Rectified Linear Unit, see figure 2. A Rectified Linear unit simply removes all the negative values for input x and sets them to zero. This is good for convolution neural networks since convolution on linear layers tends to produce negative pixel values and harder to visualize the image properly. Other types of non-linear layers maybe added as well, including max-pooling which replaces nearby values with the maximum pixel value, operating on specific feature channels.

$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2)\ldots), \mathbf{w}_L).$$

Figure 3:
A composition of several linear and nonlinear functions represent a CNN. The parameters *w* are learned from the training dataset.

At the core, the functions that represent a neural network are decided previously by the programmer or mathematician. However the learning step involves determining the weight values or parameters that are determined based on the training set, and using the theory of back-propagation and stochastic gradient descent. This learning occurs first by developing a

generalized objective function. The goal will be to minimize a loss function *l* which expresses the penalty of misclassifying an image. This loss function is treated is an additional layer in the CNN, called the *loss layer*. Since the loss function is convex, the parameter w which minimizes loss can be solved iteratively through a process called stochastic gradient descent. Stochastic gradient descent randomly chooses a step size n and solves the gradient with respect to w of the loss function. It keeps repeating this descent until the results no longer improve. This function is summarized in equation 4:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \frac{\partial f}{\partial \mathbf{w}}(\mathbf{w}^t)$$

Equation 4: The parameter w is iteratively changed with gradient descent to find the parameter that minimizes the objective function or loss function. For SGD, the step size *n* is randomly chosen.

However in order to calculate the gradient of the loss function, which is the last layer of the CNN, by definition the gradient of the previous layers must be calculated in order to get the final scalar result. Since the function at the loss layer is simply a recurrence of the previous relation all the way back to the first function layer, the chain rule can easily be applied.

$$\frac{\partial f}{\partial w_l}(x_0; w_1, \ldots, w_L) = \frac{\partial f_L}{\partial x_{L-1}}(x_{L-1}; w_L) \times \cdots \times \frac{\partial f_{l+1}}{\partial x_l}(x_l; w_{l+1}) \times \frac{\partial f_l}{\partial w_l}(x_{l-1}; w_l)$$

Figure 5: A equation representation of the chain rule

However, for inputs that involve multi-dimensional arrays (such as images) computing the chain rule becomes extremely memory inefficient as the size of these tensors grows exponentially. To handle this large computation a different approach is taken called back-propagation. First back-propagation deals with the multi-dimensional tensors by simply vectorizing them into a matrix with stacking. At this point computing the gradient is as simple as completing matrix multiplication (see equation 6).

$$\frac{\partial \operatorname{vec} f}{\partial \operatorname{vec}^\top \mathbf{w}_l} = \frac{\partial \operatorname{vec} f_L}{\partial \operatorname{vec}^\top \mathbf{x}_L} \times \cdots \times \frac{\partial \operatorname{vec} f_{l+1}}{\partial \operatorname{vec}^\top \mathbf{x}_l} \times \frac{\partial \operatorname{vec} f_l}{\partial \operatorname{vec}^\top \mathbf{w}_l}$$

Equation 6: Back-propagation with vectorizing

This back-propagation is made more memory efficient by computing the product left to right starting at the output of the equation to the input of the equation. Since the output is scalar, multiplying it starting at the end will project down each matrix into a 1x1 dimension, making the operation generally memory efficient.

Pre-trained CNN Discussion and Results:

Pretrained CNN 1: Imagenet-VGG-F

Split 1:

```
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0    0.3200
         0    0.9600         0         0         0         0         0         0         0    0.0400
         0         0    0.8400         0         0         0         0         0         0    0.1600
         0         0         0    0.8000         0         0         0         0         0    0.2000
         0         0         0         0    0.4800         0         0         0    0.0400    0.4800
         0         0         0         0         0    0.8000         0         0         0    0.2000
         0         0         0         0         0         0    0.8000         0         0    0.2000
         0         0         0         0         0         0         0    0.4000         0    0.6000
         0         0         0         0         0         0         0         0    0.9200    0.0800
         0         0         0         0         0         0         0         0         0    0.8400    0.1600
         0         0         0         0         0         0         0         0         0         0    1.0000
```

Split 2:

```
confusionMatrix =
    0.7200         0         0         0         0         0         0         0         0         0    0.2800
         0    1.0000         0         0         0         0         0         0         0         0         0
         0         0    0.8400         0         0         0         0         0         0         0    0.1600
         0         0         0    0.7200         0         0         0         0         0         0    0.2800
         0         0         0         0    0.4800         0         0         0         0         0    0.5200
         0         0         0         0         0    0.8000         0         0         0         0    0.2000
         0         0         0         0         0         0    0.8400         0         0         0    0.1600
         0         0         0         0         0         0         0    0.6000         0         0    0.4000
         0         0         0         0         0         0         0         0    0.8400         0    0.1600
         0         0         0         0         0         0         0         0         0    0.9200    0.0800
         0         0         0         0         0         0         0         0         0         0    1.0000
```

Split 3:

```
confusionMatrix =
    0.6400         0         0         0         0         0         0         0         0         0    0.3600
         0    0.9600         0         0         0         0         0         0         0         0    0.0400
         0         0    0.8800         0         0         0         0         0         0         0    0.1200
         0         0         0    0.8000         0         0         0         0         0         0    0.2000
         0         0         0         0    0.5200         0         0         0    0.0400         0    0.4400
         0         0         0         0         0    0.8400         0         0         0         0    0.1600
         0         0         0         0         0         0    0.8000         0         0         0    0.2000
         0         0         0         0         0         0         0    0.4800         0         0    0.5200
         0         0         0         0         0         0         0         0    0.9600         0    0.0400
         0         0         0         0         0         0         0         0         0    0.8400    0.1600
         0         0         0         0         0         0         0         0         0         0    1.0000
```

Split 4:

```
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0         0    0.3200
         0    0.9600         0         0         0         0         0         0         0         0    0.0400
         0         0    0.8800         0         0         0         0         0         0         0    0.1200
         0         0         0    0.7200         0         0         0         0         0         0    0.2800
         0         0         0         0    0.5200         0         0         0    0.0400         0    0.4400
         0         0         0         0         0    0.8400         0         0         0         0    0.1600
         0         0         0         0         0         0    0.8400         0         0         0    0.1600
         0         0         0         0         0         0         0    0.4400         0         0    0.5600
         0         0         0         0         0         0         0         0    0.8800         0    0.1200
         0         0         0         0         0         0         0         0         0    0.8800    0.1200
         0         0         0         0         0         0         0         0         0         0    1.0000
```

Split 5:

```
confusionMatrix =
    0.6000         0         0         0         0         0         0         0         0    0.4000
         0    1.0000         0         0         0         0         0         0         0         0
         0         0    0.9200         0         0         0         0         0         0    0.0800
         0         0         0    0.7600         0         0         0         0         0    0.2400
         0         0         0         0    0.6400         0         0         0         0    0.3600
         0         0         0         0         0    0.8400         0         0         0    0.1600
         0         0         0         0         0         0    0.7600         0         0    0.2400
         0         0         0         0         0         0         0    0.3600         0    0.6400
         0         0         0         0         0         0         0         0    0.8400    0.1600
         0         0         0         0         0         0         0         0         0    0.8800    0.1200
         0         0         0         0         0         0         0         0         0         0    1.0000
```

A trend that is observable is that on all five splits of test data very similar confusion matrices were generated. This indicates that the pre-trained CNN is highly consistent. In addition, this makes sense because the CNN's are pre-trained. The accuracy of an identification algorithm is highly dependent on the training data. In most cases, the CNN did not misclassify one of the ten classes with each other. This only happened for the strawberry, which got confused for a computer keyboard. Otherwise any error in classification resulted in the images being classified as one of the 990 classes in the CNN.

Discussion & Interpretation of Data for Pre-trained CNN 1

Accuracy:

Average: **95.9%** **96.29%** **96.23%** **95.87%** **96.03%**

```
acc =
    0.9709
    0.9964
    0.9855
    0.9818
    0.9527
    0.9818
    0.9818
    0.9455
    0.9891
    0.9855
    0.7782
```

```
acc =
    0.9745
    1.0000
    0.9855
    0.9745
    0.9527
    0.9818
    0.9855
    0.9636
    0.9855
    0.9927
    0.7964
```

```
acc =
    0.9673
    0.9964
    0.9891
    0.9818
    0.9564
    0.9855
    0.9818
    0.9527
    0.9927
    0.9855
    0.7964
```

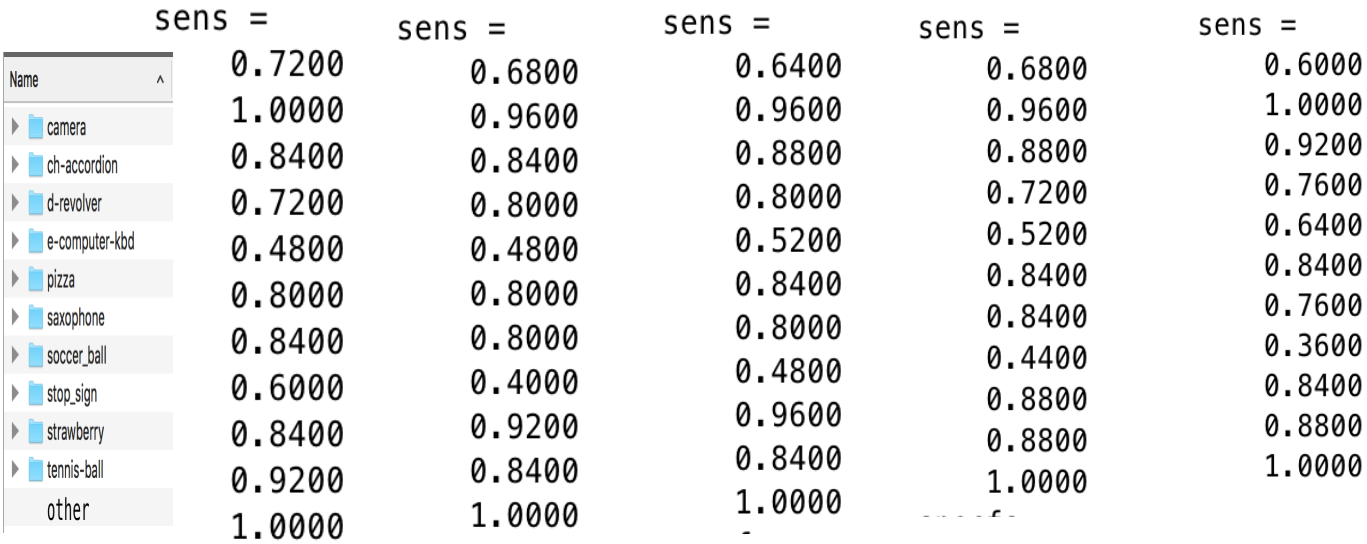
```
acc =
    0.9709
    0.9964
    0.9891
    0.9745
    0.9564
    0.9855
    0.9855
    0.9491
    0.9855
    0.9891
    0.7891
```

```
acc =
    0.9636
    1.0000
    0.9927
    0.9782
    0.9673
    0.9855
    0.9782
    0.9418
    0.9855
    0.9891
    0.7818
```

Average Accuracy across 5 splits: 96.05 %

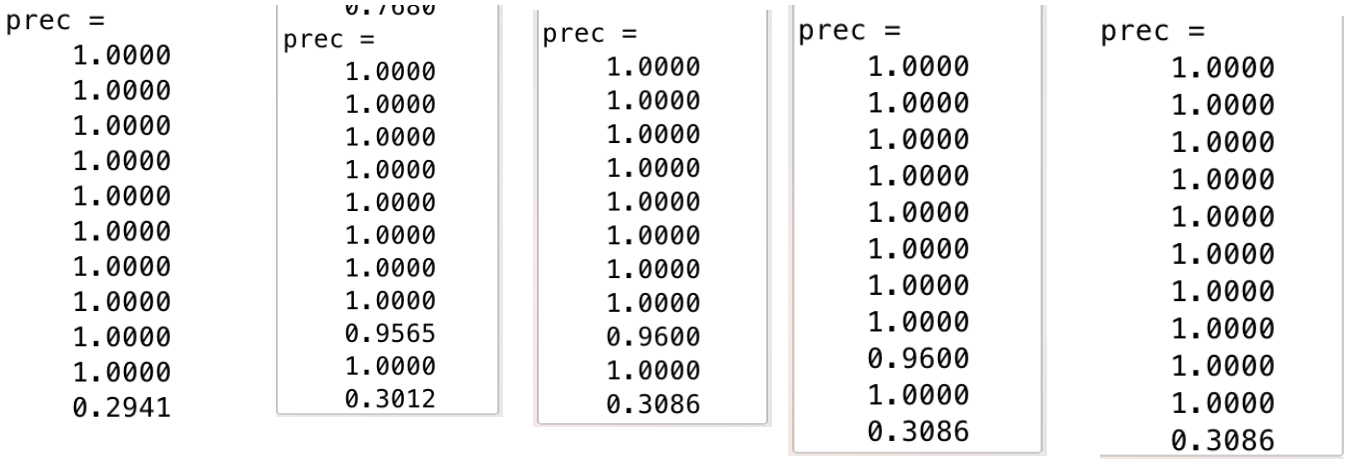
Each row in the accuracy matrix shows how many images were accurately classified in the class. Obviously compared to the Bag of Words Implementation, the CNN has a higher accuracy. In the Bag of Words implementation, the camera class was often misclassified and low accuracy with 87.8%. However in this pre-trained cnn, the accuracy is fairly good with approximately 97% accuracy. Furthermore, this cnn has the highest average accuracy compared to the other pre-trained CNN used.

Sensitivity:



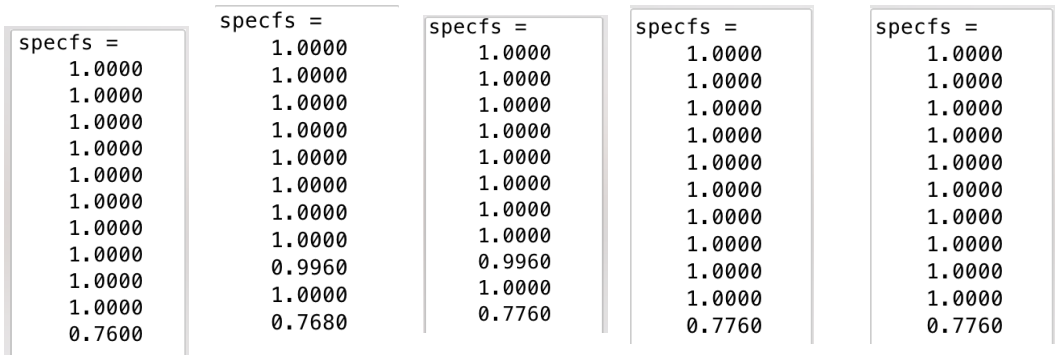
Sensitivity is a measurement of how good a test is at detecting the accurate results. Similar to the Bag of Words implementation, the camera class was difficult to accurately classify. A interesting note is that pizza and saxophone had low classification compared to the Bag of Words Implementation. In the bag of words implementation, the saxophone ranged from 73.33% to 86.6% sensitivity. Overall though sensitivity is significantly better for this pre-trained CNN compared to the Bag of Words implementation, suggesting that the CNN does a better job of correctly labeling classes.

Precision:



Precision is a measured of how many images of a particular class were correctly classified along with the total number that exist out of all the ones that were classified as an image. Interestingly precision was very strong for the camera class, which the BOW's algorithm had trouble identify. Opposite to the BoW's implementation, precision was lowest in the CNN for strawberries. This may be due to the face that most of the strawberry photos were just individual strawberries with only one or two showing groups of strawberries. Another interesting observation is for the last three precision calculations they were exactly the same.

Specificity:



Specificity is a measure of how likely the algorithm is to avoid making a false classification. An interesting anomaly is that for all the splits the soccer ball was classified correctly every single time. In addition, something else interesting is the computer keyboard has an equal measure of sensitivity in the bag of words implementation. Both avoid making false classifications for the keyboard a 100% of the time. This may have something to do with the presence of text and details on the keyboard.

Recall:

```
recall =
    0.6800
    0.9600
    0.8400
    0.8000
    0.4800
    0.8000
    0.8000
    0.4000
    0.9200
    0.8400
    1.0000
```

```
recall =
    0.7200
    1.0000
    0.8400
    0.7200
    0.4800
    0.8000
    0.8400
    0.6000
    0.8400
    0.9200
    1.0000
```

```
recall =
    0.6400
    0.9600
    0.8800
    0.8000
    0.5200
    0.8400
    0.8000
    0.4800
    0.9600
    0.8400
    1.0000
```

```
recall =
    0.6800
    0.9600
    0.8800
    0.7200
    0.5200
    0.8400
    0.8400
    0.4400
    0.8800
    0.8800
    1.0000
```

```
recall =
    0.6000
    1.0000
    0.9200
    0.7600
    0.6400
    0.8400
    0.7600
    0.3600
    0.8400
    0.8800
    1.0000
```

Recall is a measurement of the number of correctly classified images out of the total number of images that were classified. (This is the diagonal of the confusion matrix). As it can be seen again camera had a relatively low recall, suggesting that it was misclassified often. Compared to the Bag of Words, the accuracy of recognition of the strawberries was much lower. Suggesting that the BOW implementation may have been over fitted to my particular dataset or the types of images in the CNN's training set were not representative of the test images used.

Pre-Trained CNN 2: ImageNet-Resnet-101-Dag

```
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0         0    0.3200
         0    0.9200         0         0         0         0         0         0         0         0    0.0800
         0         0    1.0000         0         0         0         0         0         0         0         0
         0         0         0    0.7600         0         0         0         0         0         0    0.2400
         0         0         0         0    0.3600         0         0         0    0.0400         0    0.6000
         0         0         0         0         0    0.9200         0         0         0         0    0.0800
         0         0         0         0         0         0    0.9600         0         0         0    0.0400
         0         0         0         0         0         0         0    0.3600         0         0    0.6400
         0         0         0         0         0         0         0         0    0.8800         0    0.1200
         0         0         0         0         0         0         0         0         0    0.9600    0.0400
         0         0         0         0         0         0         0         0         0         0    1.0000
```

```
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0         0    0.3200
         0    1.0000         0         0         0         0         0         0         0         0         0
         0         0    0.9600         0         0         0         0         0         0         0    0.0400
         0         0         0    0.6800         0         0         0         0         0         0    0.3200
         0         0         0         0    0.3600         0         0         0    0.0800         0    0.5600
         0         0         0         0         0    1.0000         0         0         0         0         0
         0         0         0         0         0         0    0.9200         0         0         0    0.0800
         0         0         0         0         0         0         0    0.4400         0         0    0.5600
         0         0         0         0         0         0         0         0    0.9200         0    0.0800
         0         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0         0    1.0000
```

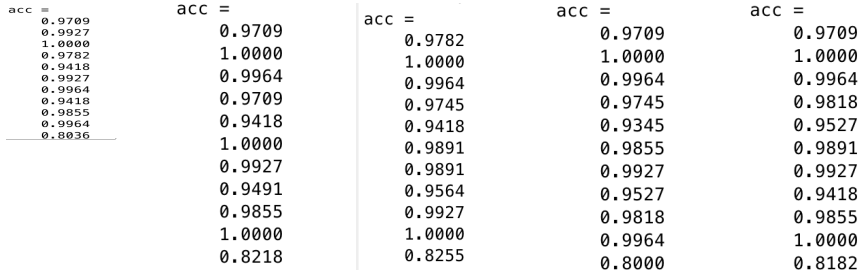
```
accuracy =
    0.8040
confusionMatrix =
    0.7600         0         0         0         0         0         0         0         0         0    0.2400
         0    1.0000         0         0         0         0         0         0         0         0         0
         0         0    0.9600         0         0         0         0         0         0         0    0.0400
         0         0         0    0.7200         0         0         0         0         0         0    0.2800
         0         0         0         0    0.3600         0         0         0    0.0400         0    0.6000
         0         0         0         0         0    0.8800         0         0         0         0    0.1200
         0         0         0         0         0         0    0.8800         0         0         0    0.1200
         0         0         0         0         0         0         0    0.5200         0         0    0.4800
         0         0         0         0         0         0         0         0    0.9600         0    0.0400
         0         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0         0    1.0000
```

```
accuracy =
    0.7720
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0         0    0.3200
         0    1.0000         0         0         0         0         0         0         0         0         0
         0         0    0.9600         0         0         0         0         0         0         0    0.0400
         0         0         0    0.7200         0         0         0         0         0         0    0.2800
         0         0         0         0    0.2800         0         0         0    0.0800         0    0.6400
         0         0         0         0         0    0.8400         0         0         0         0    0.1600
         0         0         0         0         0         0    0.9200         0         0         0    0.0800
         0         0         0         0         0         0         0    0.4800         0         0    0.5200
         0         0         0         0         0         0         0         0    0.8800         0    0.1200
         0         0         0         0         0         0         0         0         0    0.9600    0.0400
         0         0         0         0         0         0         0         0         0         0    1.0000
```

```
accuracy =
    0.7960
confusionMatrix =
    0.6800         0         0         0         0         0         0         0         0         0    0.3200
         0    1.0000         0         0         0         0         0         0         0         0         0
         0         0    0.9600         0         0         0         0         0         0         0    0.0400
         0         0         0    0.8000         0         0         0         0         0         0    0.2000
         0         0         0         0    0.4800         0         0         0    0.0400         0    0.4800
         0         0         0         0         0    0.8800         0         0         0         0    0.1200
         0         0         0         0         0         0    0.9200         0         0         0    0.0800
         0         0         0         0         0         0         0    0.3600         0         0    0.6400
         0         0         0         0         0         0         0         0    0.8800         0    0.1200
         0         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0         0         0         0    1.0000
```

A trend that is observable on both CNN's is that class 9, strawberries was misclassified as a computer keyboard. This is an interesting anomaly. It may have something to due with the fact some strawberries had packaging with letters on it. Since keyboards also have text and lettering on it, these like features might have contributed to the misclassification.

Accuracy:

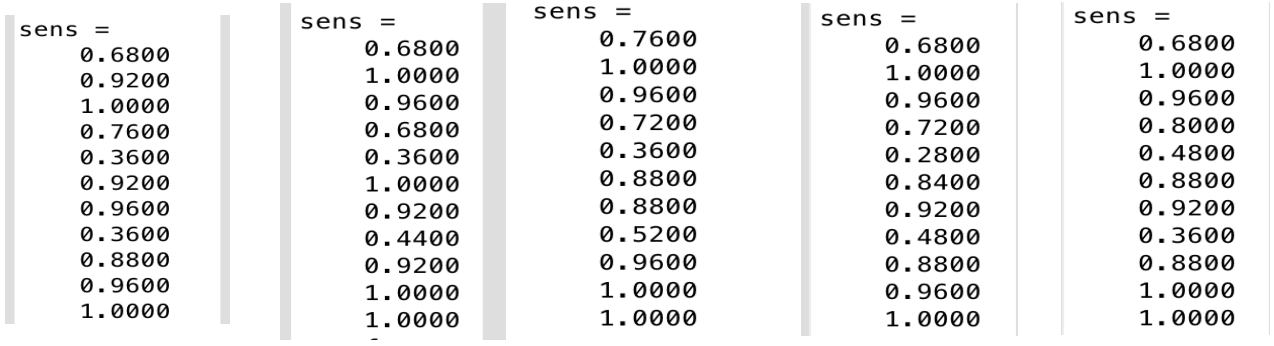


acc =	acc =	acc =	acc =	acc =
0.9709	0.9709	0.9782	0.9709	0.9709
0.9927	1.0000	1.0000	1.0000	1.0000
1.0000	0.9964	0.9964	0.9964	0.9964
0.9782	0.9709	0.9745	0.9745	0.9818
0.9418	0.9418	0.9418	0.9345	0.9527
0.9927	1.0000	0.9891	0.9855	0.9891
0.9964	0.9927	0.9891	0.9927	0.9927
0.9418	0.9491	0.9564	0.9527	0.9418
0.9855	0.9855	0.9927	0.9818	0.9855
0.9964	1.0000	1.0000	0.9964	1.0000
0.8036	0.8218	0.8255	0.8000	0.8182

Average Accuracy per class across 5 splits: 96.524 %

Each row in the accuracy matrix shows how many images were accurately classified in the class. Obviously compared to the Bag of Words Implementation, the CNN has a higher accuracy. The accuracy of the resnet is higher compared to the VGG CNN.

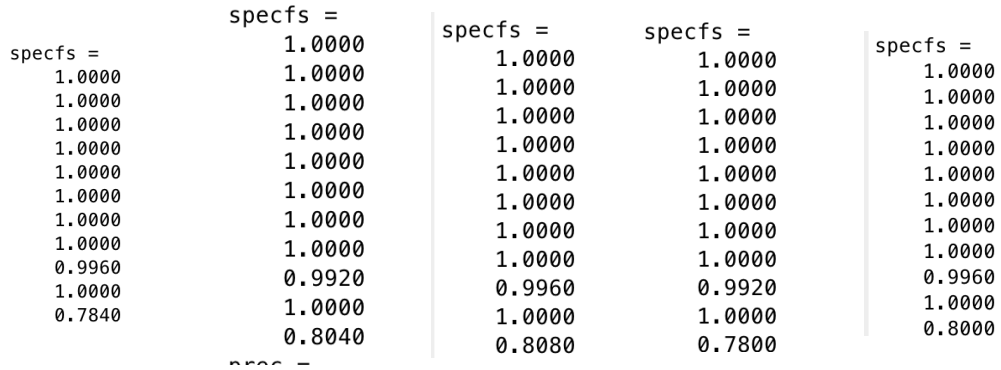
Sensitivity:



sens =	sens =	sens =	sens =	sens =
0.6800	0.6800	0.7600	0.6800	0.6800
0.9200	1.0000	1.0000	1.0000	1.0000
1.0000	0.9600	0.9600	0.9600	0.9600
0.7600	0.6800	0.7200	0.7200	0.8000
0.3600	0.3600	0.3600	0.2800	0.4800
0.9200	1.0000	0.8800	0.8400	0.8800
0.9600	0.9200	0.8800	0.9200	0.9200
0.3600	0.4400	0.5200	0.4800	0.3600
0.8800	0.9200	0.9600	0.8800	0.8800
0.9600	1.0000	1.0000	0.9600	1.0000
1.0000	1.0000	1.0000	1.0000	1.0000

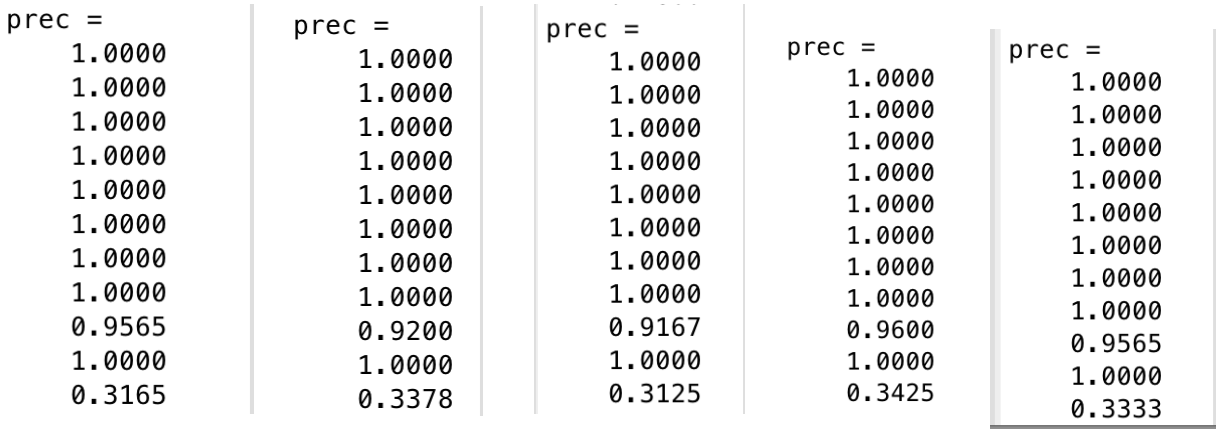
Sensitivity is a measurement of how good a test is at detecting the accurate results. Similar to the Bag of Words implementation, the camera class was difficult to accurately classify. As observed with CNN1, the sensitivity was lower for saxophone and pizza compared to the Bag of Words Implementation. This might suggest that since Bag of Words compares patches of similar features does a better job of recognizing the saxophones in the test images and the pizza pies.

Specificity:



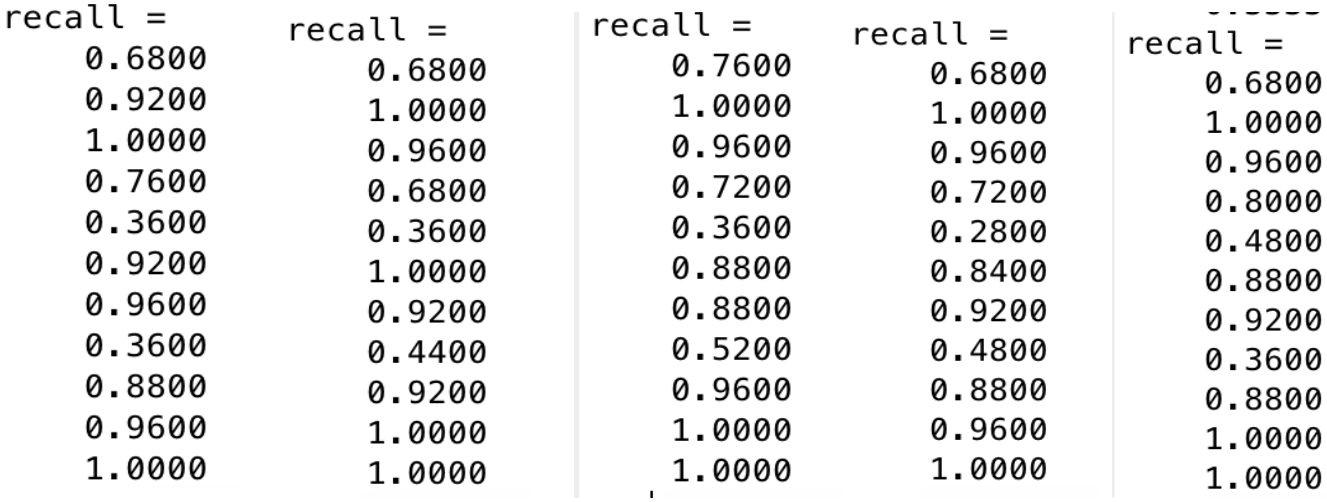
Specificity is a measure of how likely the algorithm is to avoid making a false classification. An interesting anomaly is that for both pre-trained CNNs all the splits the soccer ball had a specificity of 1. This is most likely due to the low-degree of variation among different soccer balls. They all generally look the same and have the same features. As a result, it is unlikely to make a false classification. In addition, similar to the VGG CNN, the resnet CNN had lower specificity for strawberries compared to the Bag of Words implementation.

Precision:



Precision is a measured of how many images of a particular class were correctly classified along with the total number that exist out of all the ones that were classified as an image. Following the trend observed in CNN 1 the strawberries had the lowest precision. Suggesting that strawberries were typically the images that were misclassified and bringing down accuracy.

Recall:



Recall is a measurement of the number of correctly classified images out of the total number of images that were classified. (This is the diagonal of the confusion matrix). As it can be seen again camera had the lowest recall. Evidently the camera class was very difficult to correctly classify. Though interestingly, the Bag of Words implementation did a better job of recognizing in terms of having a better recall, for strawberries.

Conclusion:

In summary, the performance of two pre-trained CNN's were compared to the performance of a custom implementation of Bag of Words. In each case a random thirty percent of 500 images with 50 belonging to one of ten classes was used as the test-data. In the Bag of Words implementation, looking at the accuracy statistics of the confusion matrix, it appears that over five splits the average accuracy per class was 92.8094%. This suggests that the algorithm works to a high degree of confidence. It appeared overall from the results of the CNN and Bag of Words implementation that one or two classes (camera, and saxophone) had wide differences in the test set and training set and as a result it was sensitive to mis-classification. However over all the Convolutional neural networks were more precise and very consistent. The two different neural nets had similar performances in terms of accuracy. However the Resnet Convolutional Neural Network had the best accuracy per class with 96.524% while the VGG Convolutional Neural Network had a slightly lower accuracy of 96.03%. This may be related to the fact the Resnet CNN had a greater number of layers, which helped it deal with complex invariances like background scenes. Looking deeper, other statistics from the confusion matrix were measured and compared. First, sensitivity was compared among the five splits between the CNNs and the Bag of Words. Sensitivity is a measurement of how good a test is at detecting the accurate results. A general trend observed was that the camera class low sensitivity, suggesting it was difficult to accurately classify cameras. Next, specificity was compared between the different algorithms. Specificity is a measure of how likely the algorithm is to avoid making a false classification. An interesting anomaly is that for both pre-trained CNNs all the splits the soccer ball had a specificity of 1. This suggests the soccer ball, was very good at being falsely classified and that all 50 images of the soccer ball were correctly recognized. It appears that other images such as a camera, and accordions were at time classified as a soccer ball. This makes sense in terms of the typically black and white features associated with each of these objects. Finally, recall was compared between each algorithm. Recall is a measurement of the number of correctly classified images out of the total number of images that were classified for that class. Overall it appears that for all three algorithms, strawberry had the best average recall, being very close to 1. This suggests that strawberries had enough distinction from the other objects in the class to avoid getting misclassified.

In summary, each algorithm did a good job of identifying the majority of images correctly using supervised machine learning techniques. It was evident that the pre-trained CNN's were much more efficient and consistent in its results. Overall the Resnet CNN (2) did the best job and was the best of the three algorithms used for appearance based recognition.