Final Project

Siddharthkumar Patel-spatel53@masonlive.gmu.edu
Harshil Patel-hpatel21@masonlive.gmu.edu
Jeminkumar Patel-jpatel22@masonlive.gmu.edu

I. Introduction & Related Work

Using what we have learned in our Computer Vision class, my initial goal was to build an Image Classification algorithm with the help of professor Stein. To speed up the process, our first step was to down-sample the image. However, it is hard to detect down-sampled images due to image pixelation. As suggested, our approach was to incorporate Bag of Words (BOW) representation to these images. Bag of Words works by first extracting sparse features from all the training images and then using k-means over the space to compute a vocabulary. You can then turn images into histograms, where the number of items in each bin is the amount of time that mean appeared on this image. The steps mentioned above will train our neural net using our fruit data set. The importance of Bag of Words is that it converts images into histograms so that a machine learning algorithm can be applied to it.

The first research paper proposes several color-shape-texture representations based on a Bag of Words framework to represent blob attributes in the vocabularies.

Citation: Alvarez, Susana, and Maria Vanrell. "Texton Theory Revisited: A Bag-of-Words Approach to Combine Textons." Pattern recognition 45.12 (2012): 4312–4325. Web

Second article detailed a series of experiments conducted to compare the classification accuracy results of Bag of Topics (BoT) and Bag of Words (BoW) representation. Ultimately, results found that BoT can provide a semantically significant representation of data.

Citation: Bahmanyar, Reza, Shiyong Cui, and Mihai Datcu. "A Comparative Study of Bag-of-Words and Bag-of-Topics Models of EO Image Patches." IEEE geoscience and remote sensing letters 12.6 (2015): 1357–1361. Web.

Third article focused on evaluating the usefulness of machine translation for Bag of Words models across different languages and found that Google Translate is helpful when using Bag of Words text models.

Citation: de Vries, Erik, Martijn Schoonvelde, and Gijs Schumacher. "No Longer Lost in Translation: Evidence That Google Translate Works for Comparative Bag-of-Words Text Applications." Political analysis 26.4 (2018): 417–430. Web.

Fourth article studied k-means clustering algorithms on large datasets. Their findings provided an enhancement to k-means clustering by requiring k or fewer passes to a dataset.

Citation: Haraty, Ramzi A., et al. "An Enhanced K-Means Clustering Algorithm for Pattern Discovery in Healthcare Data." International Journal of Distributed Sensor Networks, June 2015, doi:10.1155/2015/615740.

The last article shows the use of a K-means clustering algorithm.

Citation: Ali, Huda H. and Lubna Emad Kadhum. "K-Means Clustering Algorithm Applications in Data Mining and Pattern Recognition." (2017).

Be sure to include citations for the algorithms for both the implementation and [for 3-person teams] the comparison components of your code.

II. Related Work

The most famous example of related work is using a convolutional neural network to classify images. CNN's are known to be one of the best methods for image classification, because they are very good at extracting relevant features and associating them with individual objects. CNN's take a filter and slide it over all parts of an image. This is the part that gives it the name "convolutional", because it creates convolutions by multiplying pixel values with the ones in the filter. After repeating this for a set number of times over different areas of the image, the frames are joined together and "pooling" is used to extract the most important features within the image. The entire idea behind CNN is to repeat this enough times to the point where the model extracts the most important parts of the image and uses that to train the neural network in recognizing a specific object.

Examples of what layers in the beginning extract include lines, edges, curves, corners, and as you get deeper, they start extracting higher level features.

III. Theory and Implementation Details for Algorithms

The different methods we used to classify images were K-nearest neighbors and K-Means Clustering. In K-NN, we use the training data to compute distances and "plot them" on some plane. Then we iterate through the test data and use Euclidean distance to find the distances from the current sample to all training samples. Then we use np.argmin to find the indices of the images which have closest distance to the test sample. With K-value of 3, we would find the 3 closest distances, then take the majority vote of labels to classify the test sample.

The methods that we are using to compare for accuracy are the folder names. All the testing data is given in sets of different folders. The folder name is the label for all the images inside.We can use this to evaluate the accuracy of my model by checking if an image was correctly classified using the folder name as comparison. The formula we use as an evaluation metric is: (number_correct_predictions / number_total_predictions) as the accuracy metric to derive how accurate the model is in predicting.

$$argmin \sum_{i=1}^{k} \sum_{x \epsilon S_i} ||x - \mu_i||^2 \quad (1)$$

The equation above depicts process of k-means clustering. Suppose you have X objects that are to be divided into K clusters. The input can be a set of features,

$$X = x_1, x_2, x_3, x_4, ... \quad (2)$$

Through this equation, we can minimize the distance between each point in our scatter cloud and the assigned central points.

The next method we used to classify images was K-Means clustering with KMeans++ as the centroid selection algorithm. For KMeans clustering, we needed 36 different clusters, where each cluster represents a specific fruit or vegetable. We used KMeans++ to initialize the 36 clusters by first selecting a random centroid, then computing distances to all other points from this centroid. We then set the next centroid as the point which has the maximum distance from the last centroid, and repeat these steps until all 36 centroids are selected. Afterwards, we iterate through all the images and for each one we find the distance between all clusters, and assign it to the cluster which it has the shortest distance to. Once the first iteration is over, we adjust the center of gravity for each cluster, by making the new centroid the average of all points in a specific cluster. We repeat these steps until no point changes clusters, meaning that the model has converged.

The accuracy metric we used for KMeans clustering was V-measure. The V-measure is the harmonic mean between homogeneity and completeness: v = (1 + beta) * homogeneity * completeness / (beta * homogeneity + completeness) This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.

IV. DATA

The data we used were 3600 images of 36 different fruits and vegetables. We collected the images from this online data-set: fruit-and-vegetable-image-recognition. The training data has 36 folders where each folder contains a different fruit and vegetable. Contained inside each folder are 100 different images of that specific fruit or vegetable. The sizes of each image differ, but most are more than 1000x1000 pixels in dimensions, as a result I had to resize all of them to 64x64. The test data has 36 folders where each folder has 10 different images.

The libraries we imported were Open CV, Pillow Image, Matplotlib, sklearn's KNN & KMeans, and tensorflow & keras to test our algorithms against Convolutional neural networks for results.

V. Data Handling and Transformation

I created a numpy array to store all the pixel data for each individual image, and another array to store the labels for each corresponding image. While looping through the directory of images, I use Open CV to read the images and then use the Image module from Pillow to convert each image into an array. After running this the first time, I noticed that some of the images were more than 2000x2000 pixels in dimension, and this was leading to a problem with efficiency. Additionally, this would also lead to memory problems when working with the entire data-set, because it would require 4,000,000 blocks of memory just to store one image. As a result, I resized all images to 64x64 pixels and this saved 3,995,904 blocks of memory per image, while also increasing computation speeds. Some other challenges I faced were realizing that Open CV reads images in BGR channel, which is different from the regular RGB, so I had to address this issue as well.

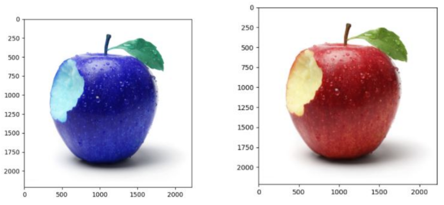


Fig. 1. BGR Apple vs RGB Apple

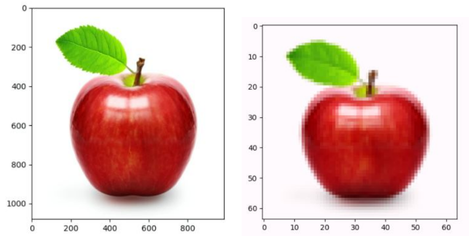


Fig. 2. Regular Image vs 64x64 Image

After reading in all the images into numpy arrays and storing their respective labels, I normalize all the pixel values in all 3 image channels between 0 and 1. I also saved all the image data and label reading to a pickle file, so I can just load it again later without having to wait 30 minutes for the computer to read the images data over and over.

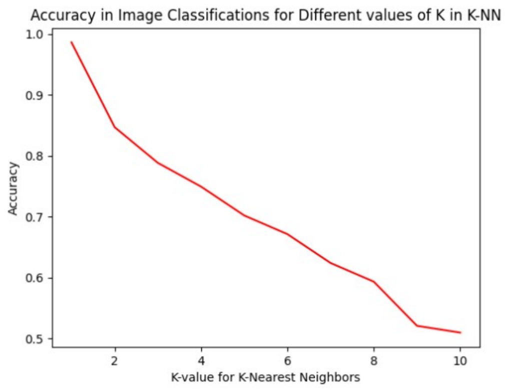
VI. Results

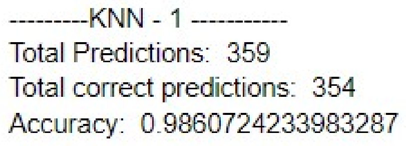
The implementation choices and details are described above in the Theory and Implementation section, so we won't cover the details again to keep the paper in the designated page range.

We experimented with K values 1 through 10, and the results in accuracy are plotted below:

The best K value we found after experimenting for KNN was 1. For K=1, our model correctly classifies the test data with a 98.6% accuracy.

The accuracy we got using KMeans on the testing set was 47% which is not the most optimal. As a result, we decided





```
---------KNN - 1 -----------
Total Predictions:  359
Total correct predictions:  354
Accuracy:  0.9860724233983287
```

that KNN with n_neighbors = 1 was the best method for classifying images of fruits and vegetables.

To further compare our model, we decided to test our results versus convolutional neural networks and artificial neural networks. We used tensor flow and keras to create the models.

The explanations of parameter choices and model selections are listed next:

To create the model, I use the Sequential() class, because we're using a stack of layers in the Neural Network, and this will accept every layer. Additionally, in the Sequential() class, we take a single input and return a single output. An example is, we input a single image, and it will output a single image label.

For my layers, I use Dense Layers. Dense means all the neurons in a specific layer are connected with every other neuron in the next layer. The input shape is the dimensions of the image which we are using (64,64,3), where 64x64 is the img_height x img_width, and 3 is the RGB channels. The activation function is the one which decides whether a neuron should be activated or not. We can manipulate this by choosing different activation functions. For example, the sigmoid activation function has a range of [0, 1], and can be useful when we want probabilities of binary predictions such as "credit card transaction legit or fraudulent". I experiment with soft max, sigmoid, and Re Lu activation functions.

Optimizers allow us to train the model efficiently during processes like backward propagation. Backward propagation is essentially how neural networks learn. Once the model makes a prediction, it compares its prediction with the correct label, and fine tunes the weights accordingly based on error rates obtained in the previous epoch. The information is transmitted backwards to all layers and neurons to effectively "learn from the mistakes" the model made. For my optimizer, I use Adam, because according to the authors of this algorithm, the benefits of using Adam are:

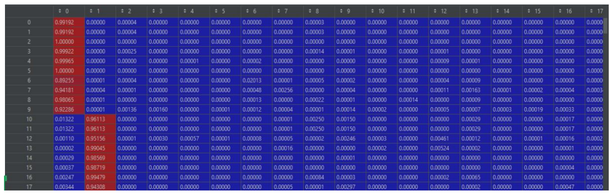
- Computationally efficient
- Little memory requirements
- Invariant to diagonal re-scale of the gradients
- Well suited for problems that are large in terms of data and/or parameters

The loss function is essentially what the model is aiming to minimize. For my model, I use categorical_crossentropy, because the problem I am solving is multi-class classification with images, and this is exactly what categorical_crossentropy is for. Categorical cross entropy outputs the probability of an input belonging to all "x" labels, where x is the number of different classes. The class which has the highest probability is identified as the label for the input.

Epochs is how many times we want to run through the entire data-set. If epochs = 10, then we will go through our entire data-set 10 times. Usually the higher the epochs is, the better the accuracy will be, because the model will have more time to adapt and learn. However, there are certain cases where the model might start to have a problem of "over fitting" if ran for too many epochs and this can lead to a decrease in accuracy. I use 50 epochs for most of the models in this project to have a fair way of measuring performance between different models.

In the models I create, I also run cross-validation where 20% of the training data is held for validation purposes. For my CNN model, I basically follow the same steps described above, but add a max pooling and convolutional layer. The convolutional layer is for extracting relevant features and associating them with individual objects. CNN's take a filter and slide it over all parts of an image. This is the part that gives it the name "convolutional", because it creates convolutions by multiplying pixel values with the ones in the filter. After repeating this for a set number of times over different areas of the image, the frames are joined together and "pooling" is used to extract the most important features within the image. The last Dense layer for my CNN has 36 neurons, because I want the final classifications to be divided into 36 different labels.

The predictions the model returns are probabilities for each image belonging to a specific class. An example is displayed below:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.99192	0.00000	0.00004	0.00000	0.00000	0.00000	0.00000	0.00000	0.00003	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
1	0.99192	0.00000	0.00004	0.00000	0.00000	0.00000	0.00000	0.00000	0.00003	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
2	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
3	0.99922	0.00000	0.00025	0.00000	0.00000	0.00000	0.00000	0.00000	0.00014	0.00001	0.00000	0.00000	0.00001	0.00000	0.00000	0.00000	0.00000	0.0000
4	0.99965	0.00000	0.00000	0.00000	0.00001	0.00000	0.00002	0.00000	0.00000	0.00000	0.00000	0.00000	0.00009	0.00001	0.00000	0.00000	0.00000	0.0000
5	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
6	0.89255	0.00001	0.00004	0.00000	0.00000	0.00000	0.02013	0.00001	0.00005	0.00002	0.00000	0.00000	0.00004	0.00009	0.00000	0.00000	0.00000	0.0000
7	0.94181	0.00004	0.00001	0.00000	0.00000	0.00000	0.00048	0.00256	0.00000	0.00004	0.00000	0.00000	0.00011	0.00163	0.00001	0.00002	0.00004	0.0003
8	0.98065	0.00001	0.00000	0.00000	0.00000	0.00000	0.00013	0.00000	0.00022	0.00001	0.00000	0.00014	0.00000	0.00009	0.00000	0.00000	0.00000	0.0000
9	0.92286	0.00001	0.00136	0.00160	0.00000	0.00001	0.00012	0.00004	0.00001	0.00014	0.00002	0.00000	0.00005	0.00007	0.00003	0.00019	0.00033	0.0000
10	0.01322	0.96113	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00250	0.00150	0.00000	0.00000	0.00000	0.00029	0.00000	0.00000	0.00017	0.0000
11	0.01322	0.96113	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00250	0.00150	0.00000	0.00000	0.00000	0.00029	0.00000	0.00000	0.00017	0.0000
12	0.00110	0.95156	0.00001	0.00030	0.00057	0.00001	0.00008	0.00005	0.00002	0.00246	0.00003	0.00000	0.00461	0.00012	0.00000	0.00001	0.00016	0.0002
13	0.00002	0.99045	0.00000	0.00000	0.00000	0.00000	0.00000	0.00016	0.00000	0.00000	0.00002	0.00000	0.00524	0.00002	0.00000	0.00000	0.00001	0.0000
14	0.00029	0.98569	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
15	0.00037	0.98719	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00004	0.0000
16	0.00247	0.99479	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00084	0.00003	0.00000	0.00000	0.00002	0.00065	0.00000	0.00000	0.00000	0.0000
17	0.00344	0.94308	0.00000	0.00000	0.00000	0.00000	0.00000	0.00005	0.00001	0.00297	0.00000	0.00000	0.00000	0.00002	0.00000	0.00000	0.00047	0.0000

0 is the class label for Apple, and the first 10 images in the test file are of apples, so the values in the cell at the first index are the highest.

To track which fruits get classified wrongly as which ones, we created a dictionary, where the keys are every fruit and vegetable class, and the value is a list of things that were incorrectly classified. Using the artificial neural network, these are the results we got.

ANN result: Fruits that were incorrectly classified: 'apple': ['capsicum', 'capsicum', 'raddish'], 'banana': ['ginger', 'ginger', 'cauliflower', 'ginger', 'cauliflower', 'pineapple'], 'beetroot': [], 'bell pepper': ['capsicum', 'capsicum', 'cucumber'], 'cabbage': ['peas', 'peas', 'cucumber', 'cucumber', 'cauliflower', 'pear', 'pineapple', 'cauliflower', 'cauliflower', 'raddish'], 'capsicum': [], 'carrot': [], 'cauliflower': [], 'chilli pepper': [], 'corn': ['capsicum', 'capsicum', 'capsicum', 'chilli pepper', 'capsicum'], 'cucumber': [], 'eggplant': [], 'garlic': [], 'ginger': [], 'grapes': [], 'jalepeno': ['cucumber'], 'kiwi': ['cucumber', 'capsicum'], 'lemon': [], 'lettuce': [], 'mango': ['capsicum', 'capsicum', 'capsicum', 'capsicum', 'capsicum'], 'onion': [], 'orange': [], 'paprika': [], 'pear': [], 'peas': [], 'pineapple': [], 'pomegranate': ['capsicum', 'onion', 'capsicum', 'paprika'], 'potato': ['raddish', 'cauliflower'], 'raddish': ['kiwi'], 'soy beans': ['orange', 'raddish', 'paprika', 'lemon', 'cauliflower', 'ginger', 'ginger', 'raddish', 'potato', 'capsicum'], 'spinach': ['cucumber', 'cucumber', 'pineapple', 'lettuce', 'capsicum', 'peas', 'lettuce', 'lettuce', 'jalepeno', 'lettuce'], 'sweetcorn': ['kiwi', 'capsicum', 'pineapple', 'kiwi', 'lemon', 'capsicum', 'capsicum', 'potato', 'lemon', 'corn'], 'sweetpotato': ['ginger', 'chilli pepper', 'chilli pepper', 'potato', 'capsicum', 'potato', 'potato', 'paprika', 'capsicum', 'ginger'], 'tomato': ['capsicum', 'capsicum', 'potato', 'paprika', 'paprika', 'apple', 'capsicum', 'paprika', 'pineapple', 'chilli pepper'], 'turnip': ['cauliflower', 'cauliflower', 'raddish', 'raddish', 'pineapple', 'raddish', 'raddish', 'raddish', 'raddish', 'pineapple'], 'watermelon': ['capsicum', 'raddish', 'paprika', 'chilli pepper', 'capsicum', 'onion', 'capsicum', 'ginger', 'raddish', 'apple']

Total predictions: 359
Total correct predictions: 247
Accuracy: 0.6880222841225627

CNN Results: Fruits that were incorrectly classified: 'apple': [], 'banana': [], 'beetroot': [], 'bell pepper': ['capsicum'], 'cabbage': [], 'capsicum': ['bell pepper'], 'carrot': [], 'cauliflower': [], 'chilli pepper': [], 'corn': [], 'cucumber': [], 'eggplant': [], 'garlic': [], 'ginger': [], 'grapes': [], 'jalepeno': [], 'kiwi': [], 'lemon': [], 'lettuce': [], 'mango': [], 'onion': [], 'orange': [], 'paprika': [], 'pear': [], 'peas': [], 'pineapple': [], 'pomegranate': [], 'potato': [], 'raddish': ['jalepeno'], 'soy beans': ['potato', 'raddish', 'mango', 'ginger', 'cauliflower', 'potato', 'ginger', 'ginger', 'potato', 'ginger'], 'spinach': ['peas', 'pineapple', 'peas', 'peas', 'jalepeno', 'peas', 'lettuce', 'cabbage', 'jalepeno', 'capsicum'], 'sweetcorn': ['corn', 'capsicum', 'pear', 'mango', 'mango', 'corn', 'corn', 'mango', 'corn', 'corn'], 'sweetpotato': ['potato', 'pomegranate', 'pomegranate', 'potato', 'onion', 'potato', 'garlic', 'orange', 'onion', 'potato'], 'tomato': ['capsicum', 'capsicum', 'pomegranate', 'apple', 'paprika', 'pomegranate', 'bell pepper', 'paprika', 'raddish', 'chilli pepper'], 'turnip': ['ginger', 'ginger', 'garlic', 'eggplant', 'garlic', 'raddish', 'pineapple', 'pineapple', 'garlic', 'corn'], 'watermelon': ['capsicum', 'chilli pepper', 'capsicum', 'kiwi', 'lettuce', 'pomegranate', 'bell pepper', 'chilli pepper', 'pineapple', 'apple']

Total predictions: 359
Total correct predictions: 286
Accuracy: 0.7966573816155988

We expected it to be a challenge for the neural networks to have a hard time classifying some images, because after looking at a few of the images, many of them look very similar. For example, the images of corn/ sweet corns, and pepper/chili/paprika look alike. It is very hard to distinguish even for the human eye, so it makes sense that even after feature extraction, it was hard for the models to predict certain images accurately.