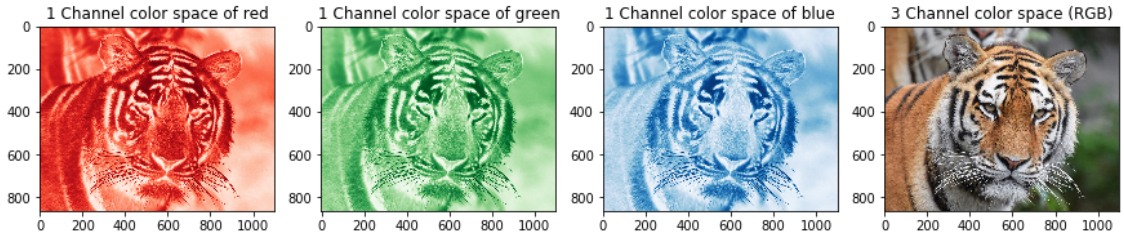
Nabeel Kahlil Maulana 2201773134
Abhishta Gatya Adyatma 2201754614
Tonto Claudinus 2201730740

Object Recognition

A. Introduction

Object Recognition is a computer task that can recognize an object on a given image. In this project assignment, we will discuss how the computer may recognize an image step by step from the theory, the explanation of each algorithm, to the application of the process. We also provide a couple of .ipynb which apply the algorithms and can give visual representation of the algorithm with the help of a great library.

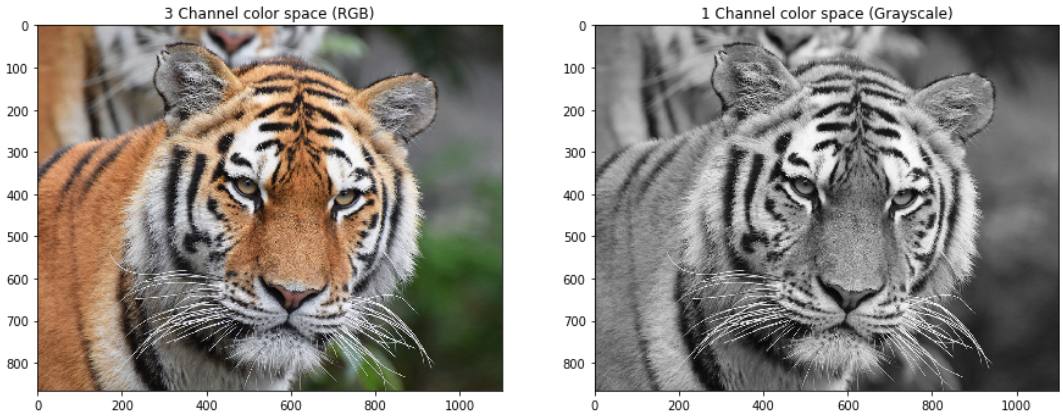
B. Images



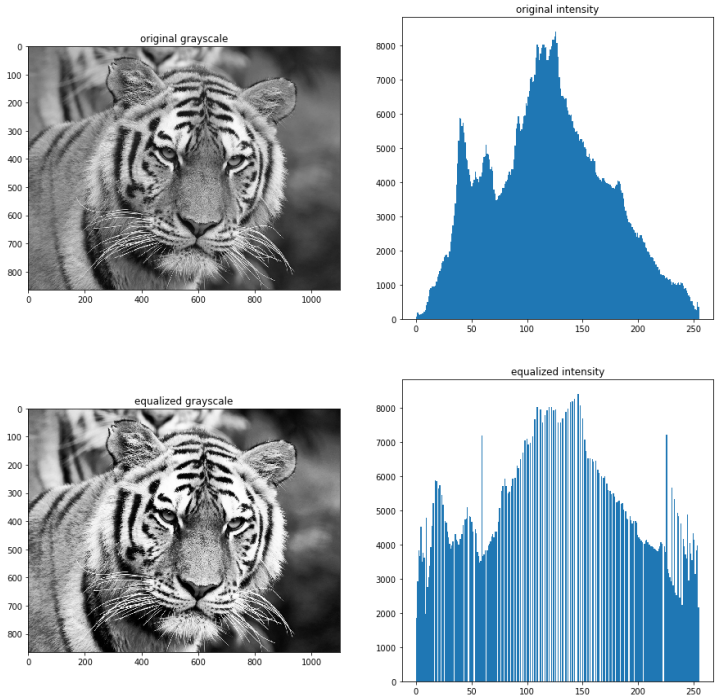
Digitally, an image on a computer can be defined as 3 Dimensional Array. Each cell in that array will hold the value of the colour map. Usually the value of the color will has the range of 256 values in 8 bit color format. The dimension size is determined by the height, width, and the number of colour map of the image. Thus the size of the image is HxWxC. Which H is the height of the image, W is the width of the image and the C is the number of color channels that the image is formed. Generally, the widely used color map format of an image is RGB colormap. Which consist of Red, Green, and Blue. Each pixel on an RGB image that is defined on coordinate HxW, will hold three color values. The value usually determines the contrast on that colour map. For example on the blue channel colormap will hold a value between 0 to 255. The value 0 represents black colour and the colour above will gradually become closer to blue. While the 255 value represents the most blue value. This applied to other color channels but with the high value is corresponding to the color of the color channel.

C. Preprocessing

Before the image goes through the main image process, the image needs to be prepared to get a better result and optimize the model. There are a number of preprocessing processes that can be used depending on the use case.



One of which is grayscaling. Grayscaling is converting multicolor channel images into one dimension color maps. The reason for this method of finding interest points on images such as edges, corners, or flat pattern, one colour space is enough. The other reason is reducing the computing complexity and can make use of a simpler algorithm. But other use cases can make the use of multicolor channels part of the process and may result in better results.



HIstogram equalization is also another form of preprocessing by transforming a grayscale image and converting it into a higher range of contrast of the image. What it does is making a histogram that plots the frequency color value of the image. The histogram will have 256 bins which represent the value range of in picture (0 to 255 in 8 bit format). The height of the bin is determined by the number of occurence of each value on the image. And then stretch the frequency bin by distributing the value across the range. The formula on equalizing each pixel of the image can be used in this formula:

$$g_{i,j} = \text{floor}((L-1)\sum_{n=0}^{f_{i,j}} p_n),$$

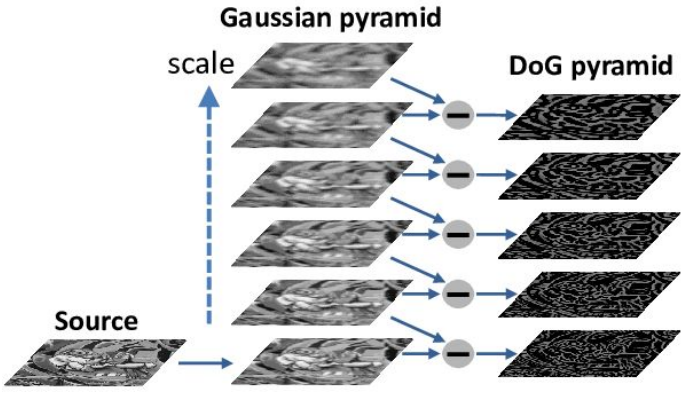
While the Pn is the normalized intensity value of that pixel taken from that intensity histogram value.

$$p_n = \frac{\text{number of pixels with intensity } n}{\text{total number of pixels}} \qquad n = 0, 1, ..., L-1.$$

D. Feature extraction

Every image has important features that uniquely define a pattern on the image. This pattern can be made up of many intrinsic information. But in order to use this valuable information, we need to convert it into a simpler form so we can apply it to our specific needs. The terms of these patterns are called interest points. One of which method on extracting useful patterns in our image is Scale-Invariant Features Transformer (SIFT).

In SIFT, the interest points are made up of key points and the descriptor. Where key points is the location of the image that we call interest point and descriptor is the value that describes the pattern of that interest point.

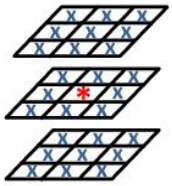


To determine the key points, SIFT uses a difference of gaussian or DOG. What it does first is to convolve the image using gaussian blur with a different scale of blur (σ). So the original image will produce many blurred images with different levels of blurriness. After having these blurred images, we find the difference of the gaussian by subtracting the level blurred image with a different σ scale. That subtracting image can be defined by this equation:

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma),$$

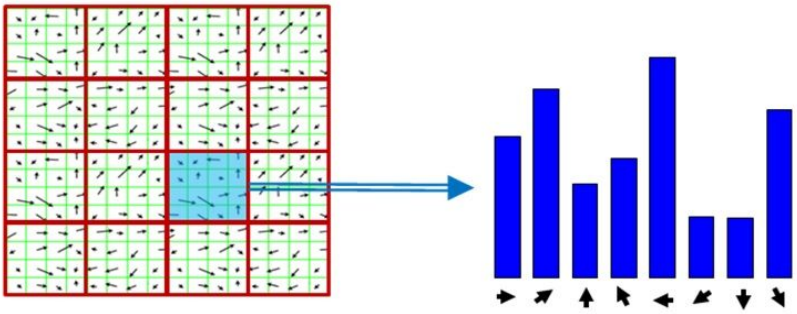
Where L is the gaussian blur function with the parameter σ, blur scale k. Notice that both the k parameters are different so the left and right L value will be slightly

different and by subtracting both of them will find the *difference* of the gaussian blurred image.



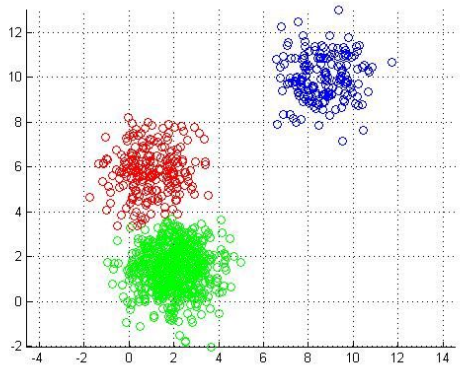
After having these differences, we can find the key points by comparing it to its neighbour pixel. We iterate all pixels in every layer of DoG and compare it to the 26 pixel neighbour. Where 8 pixels that surround it in the same level, 9 pixels in the upper DoG level, and 9 pixels in the bottom level. If that pixel is a local minima or maxima, we determine that location of the pixel is the key point.

Now we have a bunch of interesting points that location is defined by SIFT key points. Now we need to find a way to represent that interest point. The SIFT descriptor is formed by the image gradient of that key point. That area surrounding that key point is called patches and we use its gradient intensity value. The key point patches are formed by 16 sub-area which each of it has 4x4 pixels.



Each sub area then transforms into a histogram of gradient direction. The histogram consists of 8 bins that each of it counts the value in a similar direction (in range of 45 degrees). Now the patch has 16 histograms and each of it has 8 values in it. To represent the key descriptor we concatenate the histogram value into an inline vector of length 128. Now that vector is the description of that interest point.

Now we have those interest points from our image. Some images may have the same or similar patches to other images. To represent that image we can count every patch in that image and make a histogram based on that. So when images have the same object, they might have similar histogram value.



Doing this we can group up those similar patches into one cluster. And then generate patches that can represent patches in that cluster. One of algorithms that can

process clustering in K-means clustering. It can group up similar patches using centroids which each have the average value of that cluster member value.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{\mathbf{x},\mathbf{y} \in S_i} \|\mathbf{x} - \mathbf{y}\|^2$$

First thing on k-means clustering is to determine how many clusters we want to find, which is represented by (*k*). Then locate those centroids into different locations. Initiating those centroid can be done by random, specific location, or other algorithm. And then assign each patch to the nearest centroid.

If the average value of the distance between a centroid to the member cluster is large, we can displace the centroid location by using the above formula. Iterate each centroid location until the average distance of the centroid is the lowest. After “training” the k-mean centroids, we can compare each input patch to those centroids by calculating the shortest distance. If the shortest centroid is found, then that input patch is part of that cluster feature patch.

The result of counting the patches in the picture to a vector is called Bag of Features. The vector is really good to describe what object in the image by the pattern of patch occurrence. This clustering process really helps make image representation that is simple and fast to process in the next classification process.

E. Object Recognition

Now we have a way to represent an image by counting the patches in that image, using Histogram (Bag of Features), in a form of vector. To classify that image on a specific label we use a machine learning algorithm to classify that image. We will use Support Vector Machine ,or SVM, model to classify these image’s labels.

Just like other linear models, the SVM can predict or classify an input data. But in this case, we use SVM to classify our image vector and name its label. But the difference is on how it optimizes the location of the boundary line in the vector space. It creates a hyperplane that measures the distance between the line and data in the space. The SVM algorithm will maximize hyperplane’s margin which results in locating the line in the best location.

F. The Algorithm

We have provided our project code in a form of .ipynb Jupyter notebook file and this report we will summarize the main algorithm in order to make a model that can recognize an object, in this case we can recognize whether the image is a hotdog or not.

1. First we load our dataset

Our dataset contains three classes, hotdog. pizza, poutine. So our objective is to make a model that can recognize between those image classes. The train

dataset contains 155. The test set contains 40 images. We load our dataset into an image list and also create a label list which the index is corresponding to the image. The dataset images are acquired from Google images (Compressed files can be acquired using @binus.ac.id email).

2. **Find keypoint descriptor in training set**

   As mentioned in the previous section, we use Scale-Invariant Feature Transformer as our interest point detection. We use an openCV library that is ready to use to detect those train set interest points. The sift model will return the key points location and the descriptor. Because we only use the image descriptors, we won't use the location in this scenario. An image may return a different number of descriptors and each descriptor is a vector that is length is 128.

3. **Cluster all our training descriptors**

   The reason for clustering all our descriptors is because we see each of them as our image patches. But we have not been able to represent an image from all those patches. We use the K-Mean clustering algorithm that is provided by the Scikit-learn machine learning library. We set our kmean model to have 150 centroids, 300 times of iterations, and 2 times on retry the centroid location. The reason we re compute the centroid is because the kmean algorithm may fall into the local optima. So it is best to restart the model and use the best interia model to use on our next process.

4. **Create Bag of Feature of training images using cluster centroids**

   From all patches available now we have reduced the number of possible patches into 150 patches (number of centroid). Now we can efficiently create image representation using these much simpler 150 vector values.
   On each image, we create an array of zero of the length of 150. We walk through all image descriptors and categorize it using the kmean trained model. Each image patch will increase the histogram value that patch has the similar value of centroid in the cluster. We do this on all our test images which result in a n-th row of list which each value is a vector that represents an image.

5. **Scale our BoF train set vector values**

   We scale all our BoF vector values using scikit-learn's StandardScaler class which scales each value to 0 to 1 floating point of value. Using this scaled value will result in better classification results.

6. Train our classification model

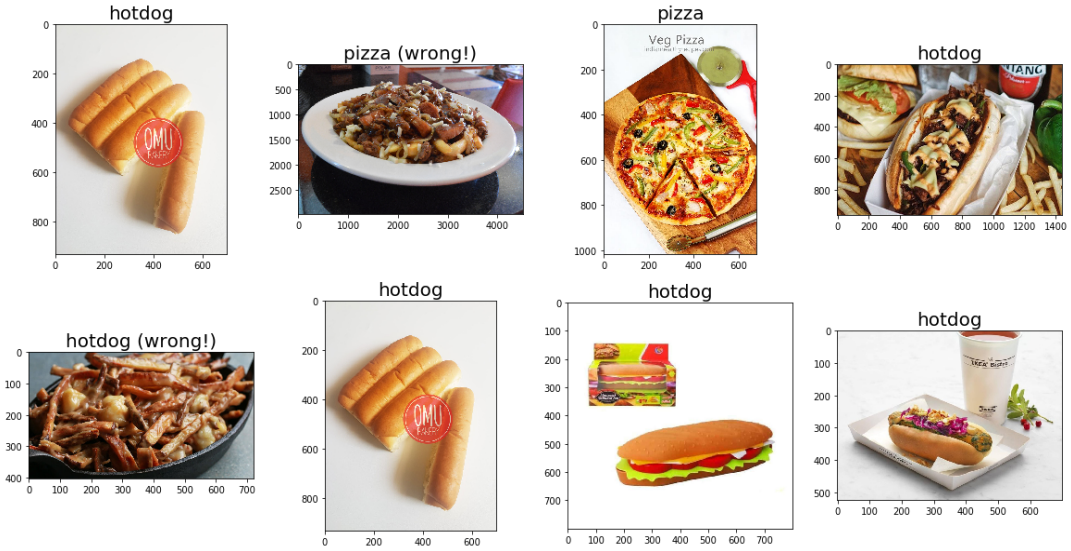
Scikit-learn also provides a library on different types of classifications. We use the SVM library that has C-Support Vector Classification which fits our needs. The library is an implementation of libsvm library which is a paper published by C. Chang et al. The model trains and predicts our data really fast in our system. We train the SVM model by feeding our Bag of Feature train data and its truth label.

7. Do the same preprocess on our test set

The test set goes through the same process as the train set from finding its descriptor, the making its Bag of Features vectors using the trained kmean model, into scaling its value. But the difference is that we do not train our K-mean model and SVC model, because our test set proposes to evaluate that model.

8. Try our Classification model using our test data

After making the test set BoF, we test our classification to distinguish whether an image is a hotdog or not. On our .ipynb file will randomly select 4 test sets and display the image and its predicted class. If the prediction is wrong, the label will also have “Wrong” indication on the image title. Here is the example result of our prediction model.



9. Evaluate our model

To evaluate our model accuracy, we use scikit-learn accuracy_score to evaluate the model prediction. The result is that the model has 0.575% which is still not an ideal model in the real world scenario. That means there is still room for us to tweak the model with different hyperparameters or even, with other better algorithms.

REFERENCES

https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf
https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40
https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e
https://scikit-learn.org/stable/index.html
https://opencv.org/
https://matplotlib.org/
https://www.google.com/search?q=pizza&tbm=isch&ved=2ahUKEwio9aiBlPztAhUXm0sFHamiBTkQ2-cCegQIABAA
https://www.google.com/search?q=poutine&safe=strict&sxsrf=ALeKk02c9-mDlVOGro_TdSReEqk7jBUCrw:1609553016140&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjPtJielPztAhVBbysKHZmRDGkQ_AUoAXoECBIQAw&biw=1536&bih=722
https://www.google.com/search?q=hotdog&safe=strict&rlz=1C1CHBF_enID904ID904&sxsrf=ALeKk03-MTlUzv9qtPBosyHLqO91n2nBiQ:1609553039426&source=lnms&tbm=isch&sa=X&ved=2ahUKEwipz6WplPztAhXaZSsKHXAcAGEQ_AUoAXoECAMQAw&biw=1536&bih=722
https://binusianorg-my.sharepoint.com/personal/nabeel_maulana_binus_ac_id/_layouts/15/guestaccess.aspx?guestaccesstoken=uaCDnUNnEGXhyEwg1awFnEuYiBEeg%2BR8QIkL%2FvjNCmM%3D&docid=2_028c530ebff8645e58d2e1959c51e9b15&rev=1&e=08iHtV