

# Project Report

## **Application of Clustering and CNN Classification in Image Dataset**

Richard Valent Tanuwijaya

[rvtanuwijaya@connect.ust.hk](mailto:rvtanuwijaya@connect.ust.hk)

20731411

## Dataset Description:

The dataset for this project is a set of biscuit cookie images with 4 categories:













Defect\_No: Cookie has no defect

Defect\_Shape: Cookie has irregular shape or incomplete shape

Defect\_Color: Cookie has strange colors

Defect\_Object: Anonymous object is detected in the cookie image

Here are some samples of each categories:

Defect_No	Defect_Shape	Defect_Color	Defect_Object
			
			
			

The source can be found here: [Industry Biscuit \(Cookie\) dataset | Kaggle](#)

The dataset includes a folder “Images” consisting 4900 images resized to size 256x256 with distribution each class as follows:

Defect\_No: 1896 images

Defect\_Shape: 1860 images

Defect\_Color: 512 images

Defect\_Object: 632 images

The class assigned to each image is recorded in “Annotations.csv” with the format “file, classDescription, classCode”. “file” represents the file name, “classDescription” describes the class the image is categorized to, and “classCode” being the enumeration of the “classDescription”.

## Data Splitting and Preprocessing:

For this Project, I used a balanced dataset of 450 images each class, and split them to 60% train set, 20% validation set, and 20% test set. After splitting, the files are separated as the following:

Train	defect_no	270 files
	defect_shape	270 files
	defect_color	270 files
	defect_object	270 files
Valid	defect_no	90 files
	defect_shape	90 files
	defect_color	90 files
	defect_object	90 files
Test	defect_no	90 files
	defect_shape	90 files
	defect_color	90 files

	defect_object	90 files
--	---------------	----------

The “ImageDataGenerator” class from the “tensorflow.keras.preprocessing.image” is used to preprocess the images. Various image augmentation techniques including rescaling the pixel values by 255, applying shear = 0.2 and zoom transformations = 0.2, and horizontal flip. These help in increasing the variation in the training data and prevent overfitting. Although the dataset has been resized to 256x256, I resized the image to 256x256 to be sure. The number of images in each batch is set to 32, and the ‘class\_mode’ is categorical, as the images are going to be one-hot encoded.

## Software and Hardware Environment:

This project is run in google colab environment. Here are more specific details:

- Python Version: 3.10.11
- Tensorflow Keras Version: 2.12.0
- Cv2 Version: 4.7.0
- Seaborn Version: 0.12.2
- Numpy Version: 1.22.4
- Matplotlib Version: 3.7.1
- NVIDIA-SMI Version: 525.85.12
- CUDA (Compute Unified Device Architecture) Version: 12.0

This project is run on hardware with these specifications:

- CPU Model: Intel Xeon CPU @ 2.20GHz
- GPU Model: NVIDIA GPU (Tesla T4)

## Machine Learning Tasks:

The objective of my project is to analyze an image dataset using both classification and clustering techniques on the test set. By performing classification, I aim to develop models that can accurately recognize and label different conditions of objects in the images. By performing clustering, I aim to group similar images together based on their visual features, such as color, texture, or shape. Since the images in the dataset are quite similar, this analysis is done to gain insights into the

distribution and diversity of the image dataset, and discover how well the features differ between 4 cookie conditions.

## Machine Learning Methods:

The machine learning techniques that are used include both supervised and unsupervised learning. In this project, a convolutional neural network is used to do classifications and used to predict an unknown image to a one hot encoded category. And to learn more from the dataset, I clustered the images using k-means clustering and hierarchical (agglomerative) clustering methods to see which clustering algorithm performs better for this dataset. Clustering allows us to learn if the features are identifiable without having the true classes of the dataset.

## Experiments:

### 1. Classification (Convolutional Neural Network)

After preprocessing the training and validation dataset, there are 1080 images in the training generator and 360 images in the validation generator, which are evenly distributed to 4 classes. The details of the data after preprocessing:

```
Train generator:  
Number of batches: 34  
Batch size: 32  
Image shape: (256, 256, 3)  
  
Validation generator:  
Number of batches: 12  
Batch size: 32  
Image shape: (256, 256, 3)
```

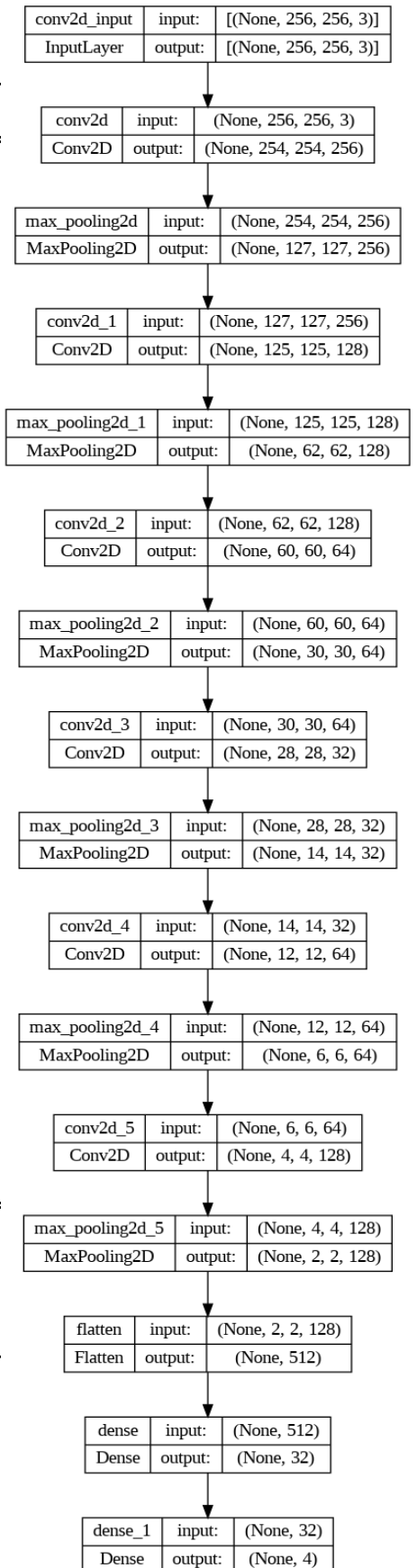
The CNN Model is built of 15 layers in the network. These include 6 convolutional layers, 6 pooling layers, 1 flattening layer and 2 dense layers. The details are as follows:

The first convolutional layer applies 256 filters with a kernel size of 3x3, followed by a max pooling layer. The subsequent convolutional layers apply 128, 64, 32, 64, and 128 filters, respectively, each with a kernel size of 3x3, followed by a max pooling layer. After the final max pooling layer, the output is flattened into a 1D

vector, and then passed through two dense layers. The dense layers have 32 and 4 neurons, producing a probability distribution over the 4 classes.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d	(None, 254, 254, 256)	7168
Max_pooling2d	(None, 127, 127, 256)	0
conv2d_1	(None, 125, 125, 128)	295040
max_pooling2d_1	(None, 62, 62, 128)	0
conv2d_2	(None, 60, 60, 64)	73792
max_pooling2d_2	(None, 30, 30, 64)	0
conv2d_3	(None, 28, 28, 32)	18464
max_pooling2d_3	(None, 14, 14, 32)	0
conv2d_4	(None, 12, 12, 64)	18496
max_pooling2d_4	(None, 6, 6, 64)	0
conv2d_5	(None, 4, 4, 128)	73856
max_pooling2d_5	(None, 2, 2, 128)	0
flatten	(None, 512)	0
dense	(None, 32)	16416
dense_1	(None, 4)	132
=====		
Total params: 503,364		
Trainable params: 503,364		
Non-trainable params: 0		



Before training, EarlyStopping callback is created with 'keras.callbacks.EarlyStopping()' to avoid overfitting. The callback is set with monitoring 'val\_loss' and patience = 5, which will stop the training if validation loss does not decrease for 5 epochs.

This model is then trained with some parameters including:

- train\_generator => the preprocessed train data
- steps\_per\_epoch = (1080/32) => total training images / batch size
- epochs = 15
- validation\_data = validation\_generator => the preprocessed validation data
- validation\_steps = (360/32) => total validation images / batch size
- callbacks = eraly\_stopping => the callback that is created previously

After training, the test set is predicted using the trained CNN model. The results will be shown and discussed in the “Result” section of this report.

## **2. Clustering (K-means and Hierarchical)**

Before applying the clustering algorithm, features are extracted from the images. The procedure of the feature extracting (including some preprocessing) from an image is :

- Read the image using 'cv2.imread' from the image path
- Resize the image to 256x256
- Split the red, green, and blue channel using array slicing ([ : , : , 0 ] selects the red channel, [ : , : , 1 ] selects the green channel, and [ : , : , 2 ] selects the blue channel)
- Use '.flatten()' to flatten each channel to 1 dimensional array
- Concatenates the flattened color channels into a single feature vector using numpy, 'np.concatenate'.

From the test dataset path, we extract the features for all images, and store them in an array called 'features'. Other than that, the classes / labels are stored using enumeration of the folder name / class name of the images extracted to an array 'labels'. For example, I did enumeration of the class names as this:

```
class_to_label = {  
    "defect_no": 0,  
    "defect_shape": 1,  
    "defect_color": 2,  
    "defect_object": 3  
}
```

The features can then be trained. In this project, K-means and hierarchical clustering algorithm is used:

- `k_means = KMeans(n_clusters=4, random_state=4211).fit(features)`
- `hierarchical = AgglomerativeClustering(n_clusters=4).fit(features)`

‘`n_clusters`’ represents the number of clusters to divide the image features to, and ‘`random_state`’ is to set the random seed, ensuring the same results after running multiple times. Both ‘`KMeans`’ and ‘`AgglomerativeClustering`’ can be imported from ‘`sklearn.cluster`’. The results of the training are numpy arrays of labels indicating the corresponding clusters assigned to every image, with the same order of image extracted features in ‘`features`’. They can be accessed using ‘`k_means.labels_`’ and ‘`hierarchical.labels_`’, based on the name of the trained model.

### 3. PCA Component

To better visualize the results of the clustering algorithms, I wanted to do a scatter plot in 2 dimensional space. In this case, Principal Component Analysis (PCA) is used to transform the features to 2 dimensional. The PCA model can be imported from ‘`sklearn.decomposition`’, and the input to train the PCA is the ‘`features`’ that is extracted from test data images. The input can be trained with PCA:

- `pca_features = PCA(n_components=2,  
 random_state=4211).fit_transform(features)`

‘`n_components`’ represents the number of principal components to retain after the transformation, and ‘`random_state`’ is just to ensure that the random number generator used is initialized equally. The output, ‘`pca_features`’ is a 2 dimensional



array, each row will represent a transformed version of a row in the original features array. So the shape of 'pca\_features' will be number of data points \* rows and 2 \* columns.

## Results:

### 1. Classification (Convolutional Neural Network)

After training the model, the model is used to predict the images in the test set. A heatmap is used to visualize the performance using 'seaborn' and 'matplotlib.pyplot'. The performance of the trained CNN model is:

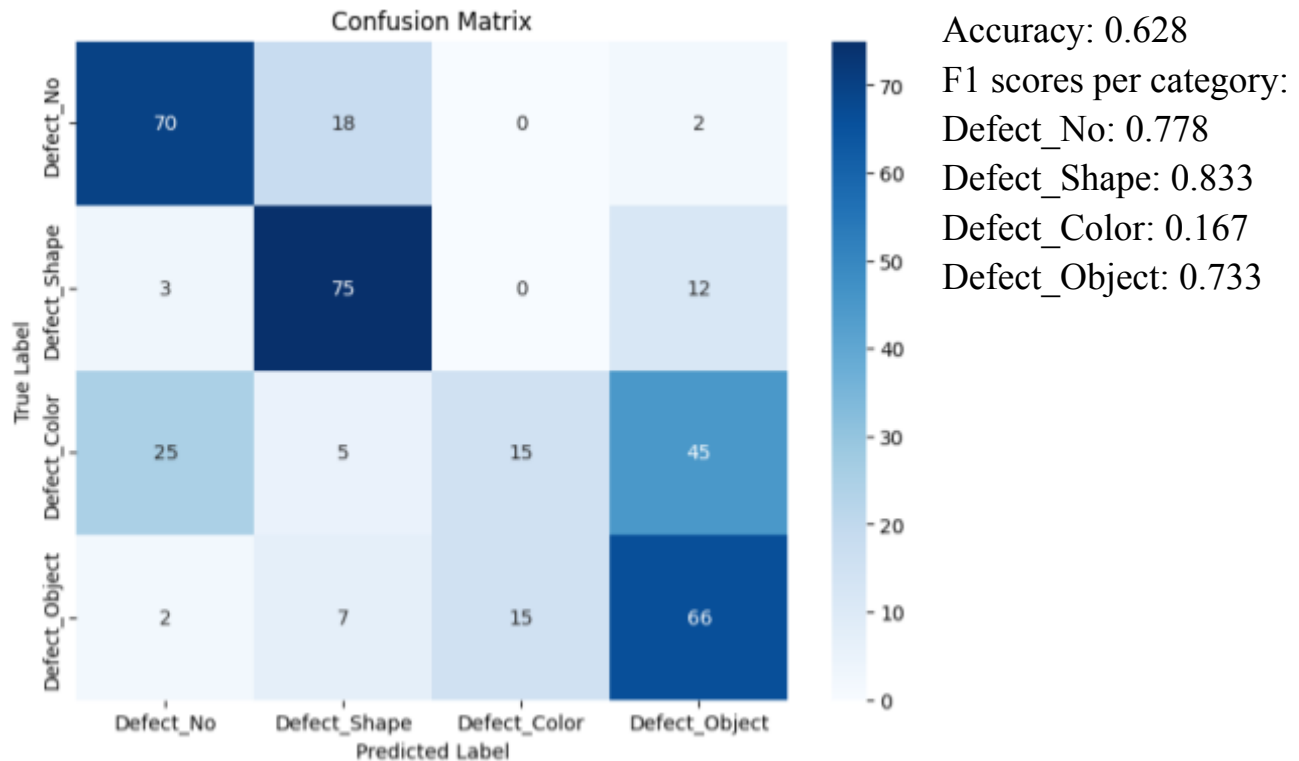


Figure 1. CNN Model Confusion Matrix

The model has an accuracy of 0.628, indicating the percentage of correctly predicted data. However, the F1 scores for each class reveal that the model performs well in some categories, such as 'Defect\_No' and 'Defect\_Shape' with F1 scores of 0.788 and 0.833 respectively. Conversely, the 'Defect\_Color' category

shows poor performance with an F1 score of 0.167. The 'Defect\_Object' category has an F1 score of 0.733, indicating moderate performance. More performance details will be discussed later after introducing the clustering and PCA performances.

## 2. Clustering (K-means and Hierarchical) + PCA

These are the performances of k-means algorithm clustering and hierarchical (agglomerative) clustering done to the test set (360 images total, 90 images per category):

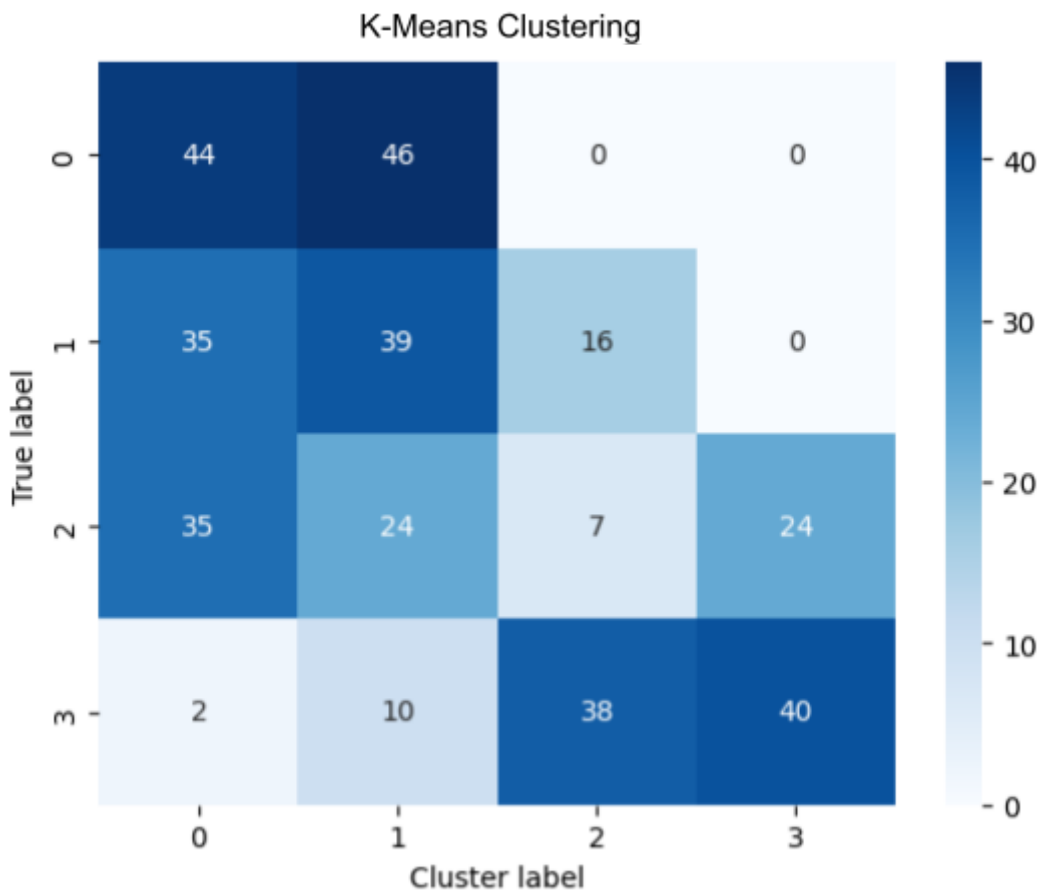


Figure 2. K-Means Clustering Confusion Matrix

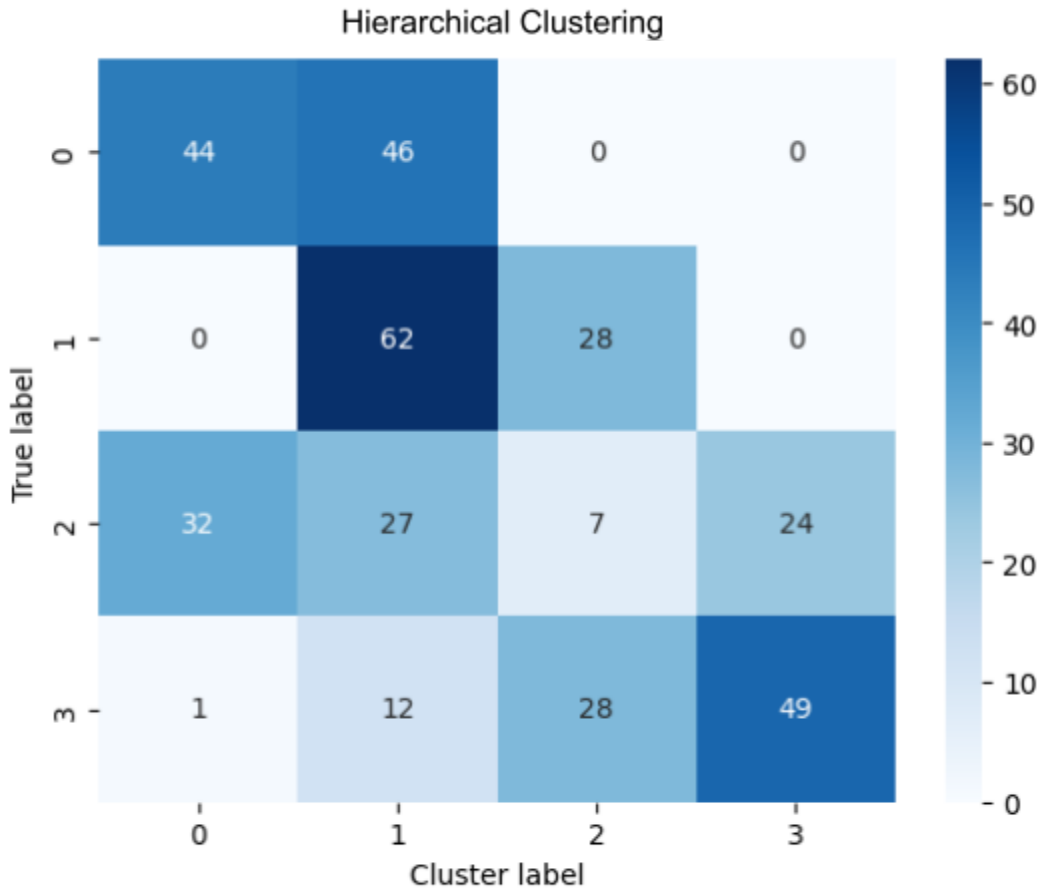


Figure 3. Hierarchical Clustering Confusion Matrix

The labels '0', '1', '2', '3' refer to the enumeration of the class labels as introduced in the experiment section:

```
class_to_label = {
    "defect_no": 0,
    "defect_shape": 1,
    "defect_color": 2,
    "defect_object": 3
}
```

The accuracy of K-means and hierarchical clustering algorithms are 0.361 and 0.45, respectively. Based on figures 2 and 3, it can be concluded that the hierarchical clustering algorithm performs better on this dataset. However, both algorithms struggle to cluster images belonging to the 'Defect\_Color' category,

possibly due to their scattered features causing them to be grouped together with other categories.

To get a clearer visual representation of how the classes are related to each other, the image features are reduced to 2 dimensions through PCA. This simplifies the process of observing the relationship between the true classes/labels and enables the clustering algorithm results to be visualized more effectively.

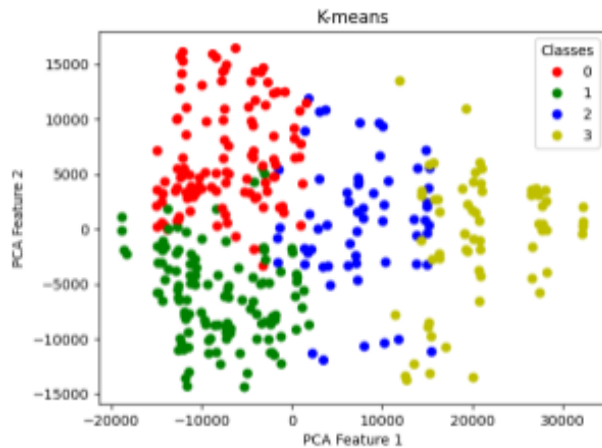


Figure 4. K-means Clustering Result

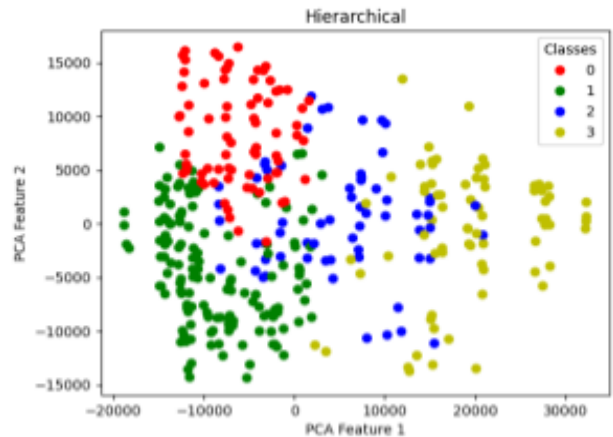


Figure 5. Hierarchical Clustering Result

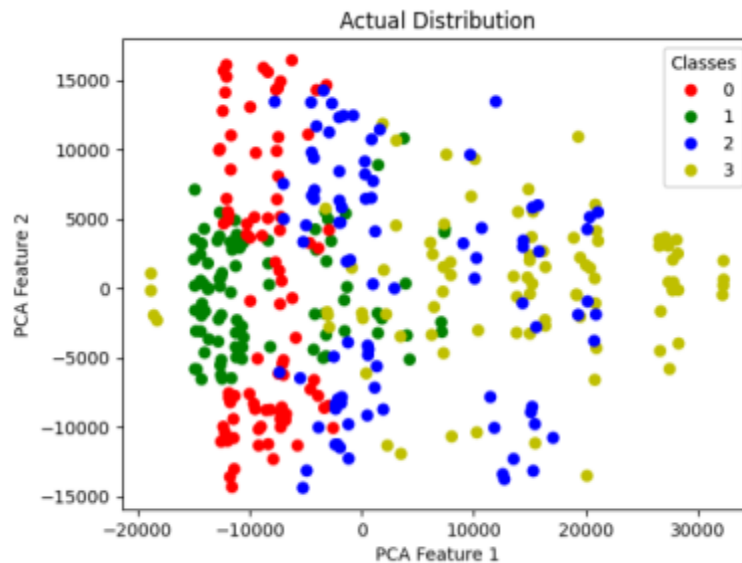


Figure 6. Actual Distribution of Test Dataset Features

From figures 4 and 5, we can see the slight difference in clusters. K-means algorithm works by partitioning the data into k clusters, where each data point belongs to the cluster whose mean is closest to it. While agglomerative (hierarchical) clustering starts with each data point in its own cluster and then

merges the closest pairs of clusters until all the data points belong to a single cluster.

In Figure 6, the true distribution of images is displayed, and it appears that the 'Defect\_Color' class is not clustered in a single central location as predicted by both the k-means and agglomerative algorithms. Instead, the blue-colored 'Defect\_Color' class appears to be spread out near other clusters. This might suggest that the features of the 'Defect\_Color' images are not distinct enough to be grouped together as a single cluster.

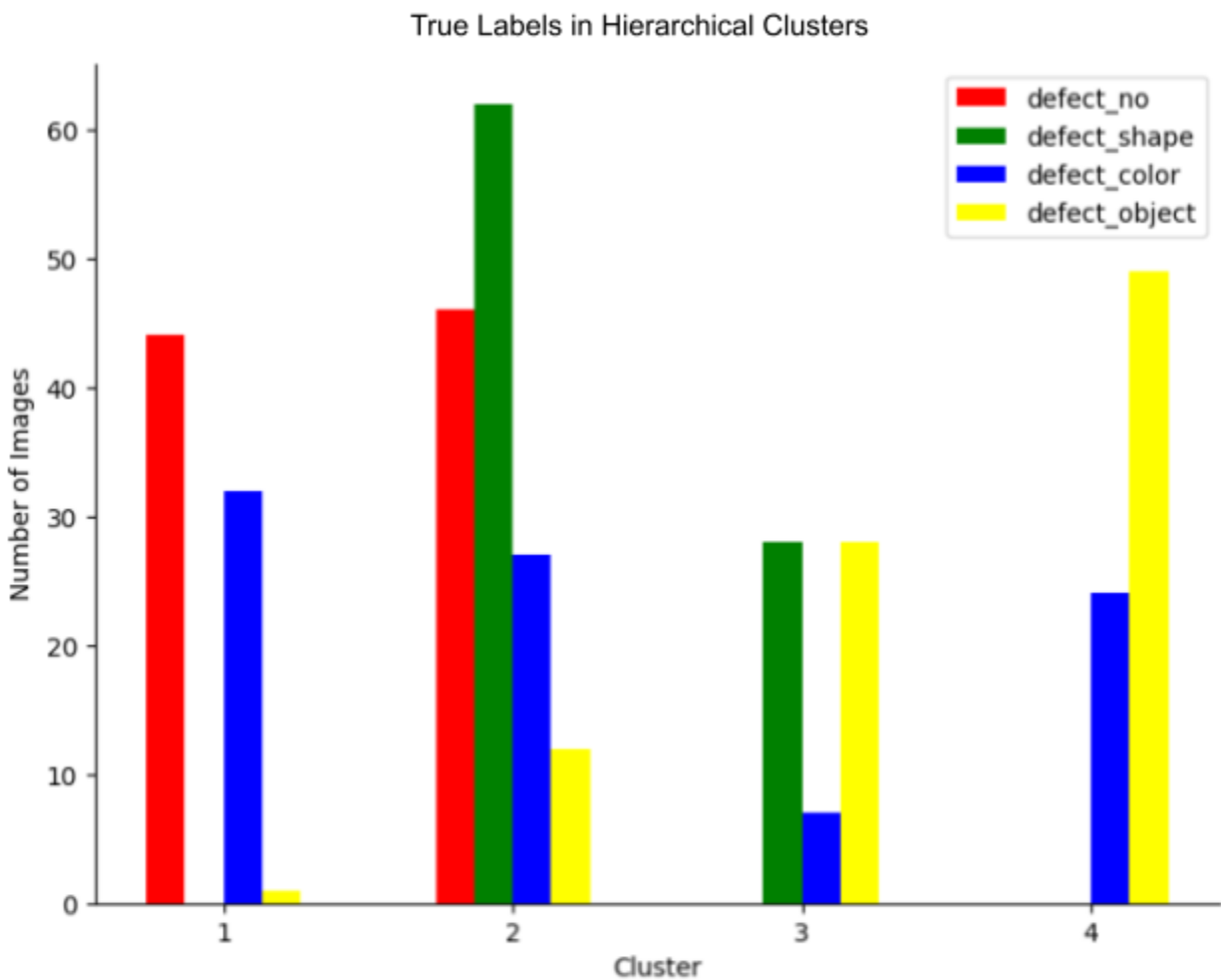


Figure 7. True Labels in Hierarchical Clusters

Overall, the hierarchical clustering algorithm was able to do a good job in clustering the image features of the test dataset without the need of true labels. Figure 7 shows that the clustering results are generally good, except for the 'Defect\_Color' class, which may have low similarity in their image features.

### 3. CNN + Hierarchical Clustering + PCA

We can apply PCA to the test dataset and visualize how the CNN performs based on the reduced PCA features. By doing this, we can gain a clearer understanding of CNN's performance. The resulting PCA features can be used to plot a scatter plot that displays the distribution of the test dataset's images. This allows us to see how well CNN is able to distinguish between different image categories.

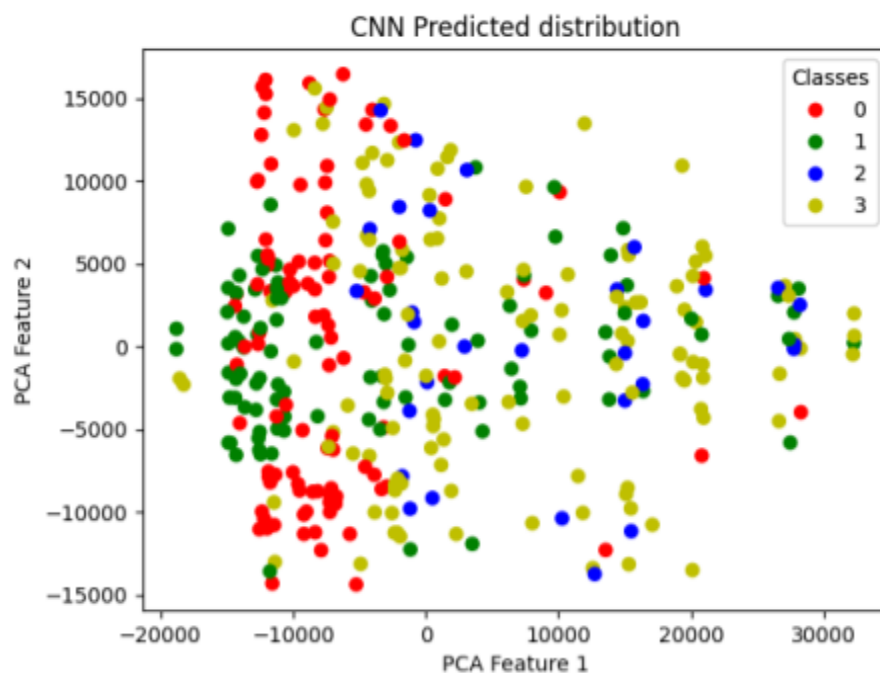


Figure 8. CNN Predicted Distribution Scatterplot

Classes:

0 => Defect\_No

1 => Defect\_Shape

2 => Defect\_Color

3 => Defect\_Object

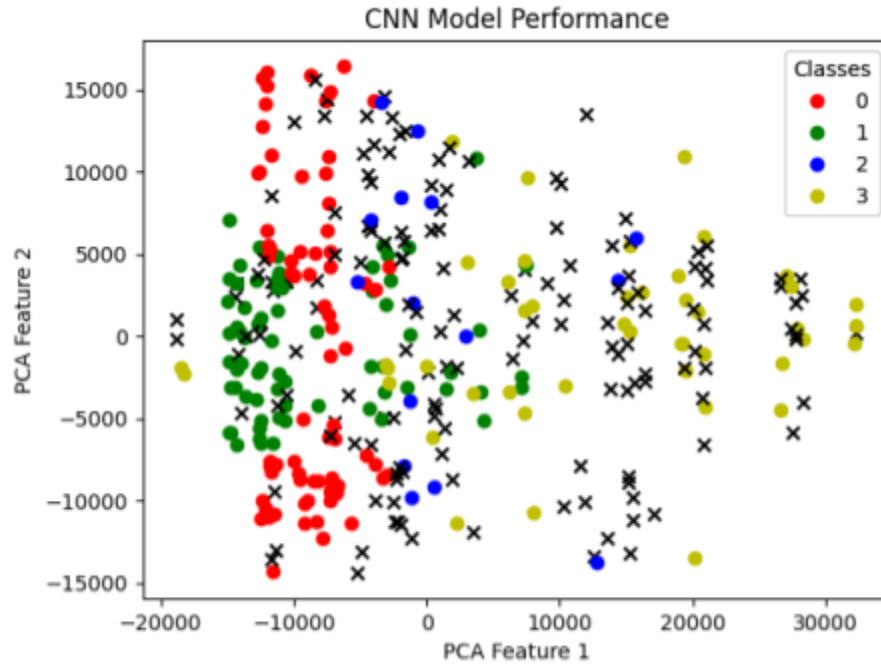


Figure 9. CNN Model Performance on PCA Features

Figure 8 shows the scatterplot of the CNN predictions using the reduced PCA features. Each point represents an image in the test set, and the color of the point corresponds to the predicted label of the image. Compared with Figure 7, the plot shows how well the CNN model is able to predict the labels of the images.

Figure 9 is a replacement for a previous plot that was incorrect, marked with 'x' symbols. This figure shows the correct scatterplot of the CNN predictions using the reduced PCA features. Like Figure 8, each point represents an image in the test set, and the color of the point corresponds to the true label of the image. This plot allows us to more accurately assess the performance of the CNN model, possibly suggesting ideas for further improvements.

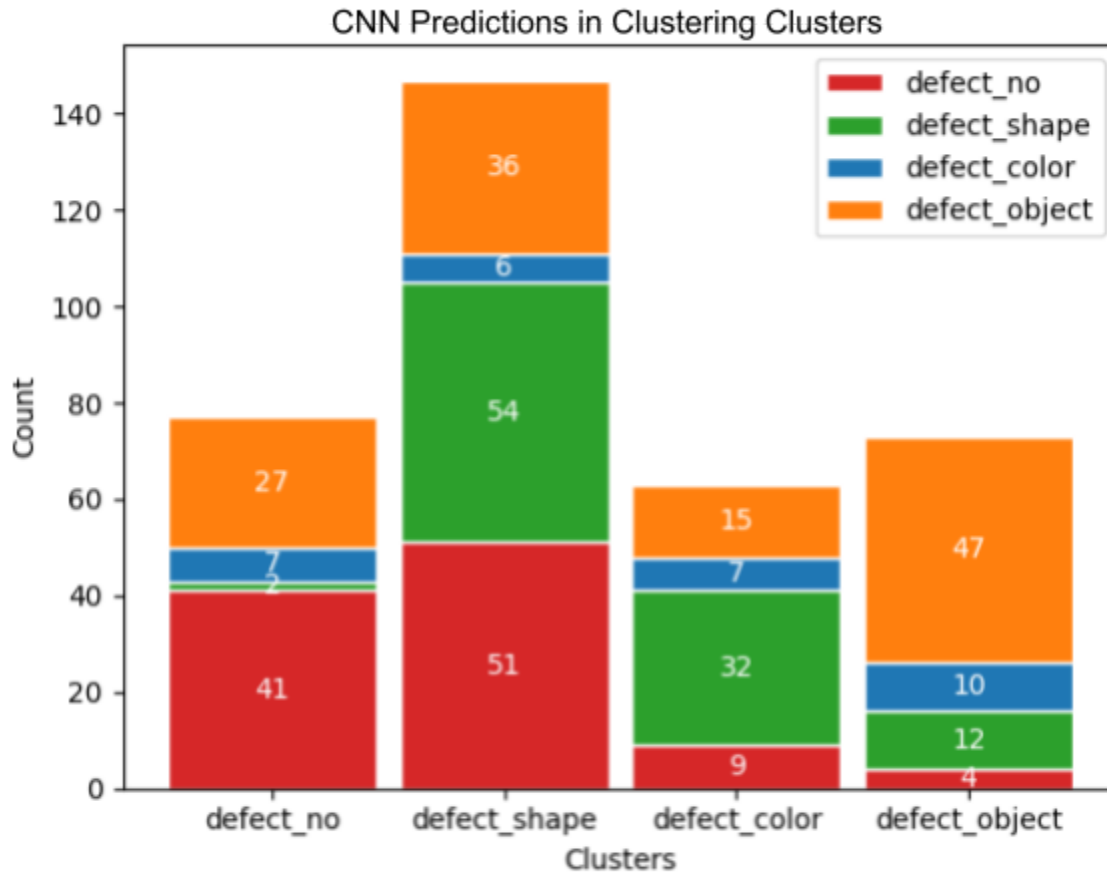


Figure 10. CNN Predictions in Hierarchical Clusters Stacked Bar Graph

Finally, by looking at Figure 10, we can see how the both algorithms (clustering and classification) correlate in their predictions. We can observe that the hierarchical clustering algorithm has effectively clustered the dataset, with each cluster consisting of a mixture of different classes. Additionally, we can see that CNN's predictions align well for clusters 'defect\_no' / cluster '0' and 'defect\_object' / cluster '3' with the clusters generated by the hierarchical algorithm, with most of the images in each cluster being predicted as the same class by CNN. The findings is that the features of the 'defect\_color' may be harder to identify, since both clustering and classification do not perform well in identifying that class. However, the classification algorithm can correctly identify more 'defect\_no' images, which are predicted in the same cluster as 'defect\_shape' by the clustering algorithm.