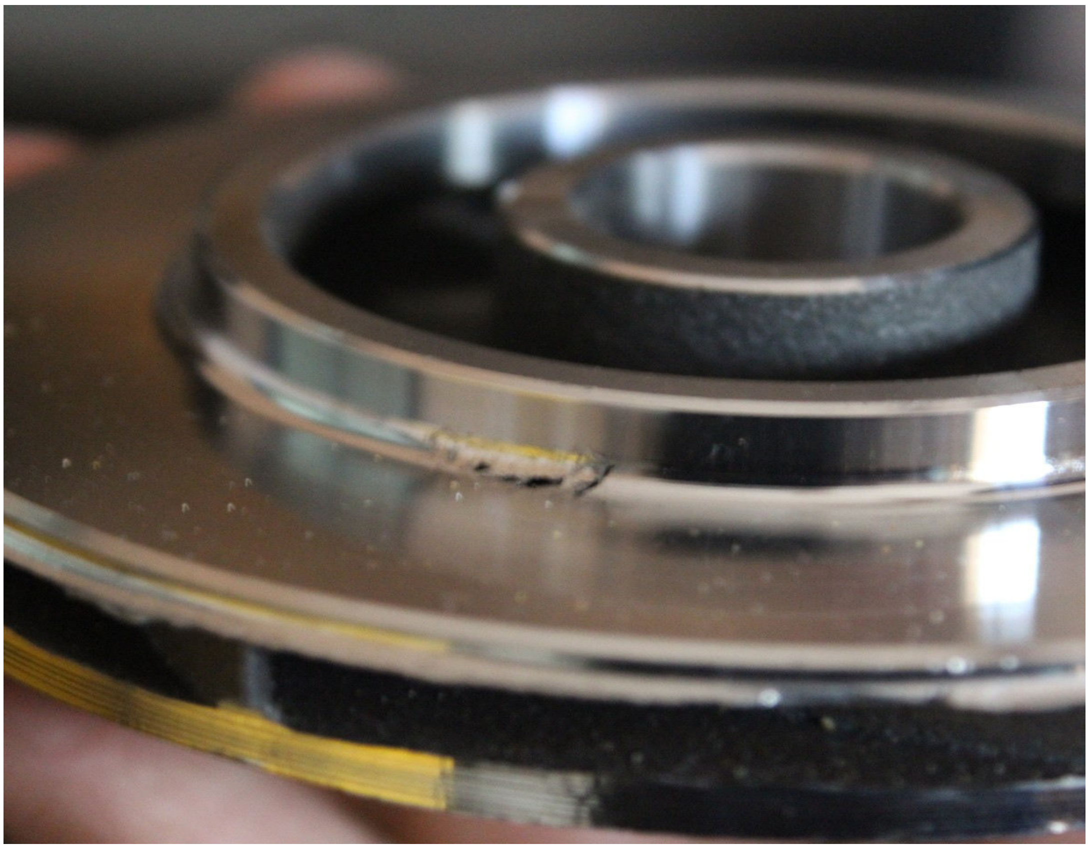
AI: Future of defect detection



Final Report for Assignment of AI61005
by

Kandagatla Kushall
B.Tech (20ME10108)
Department of Mechanical Engineering

Krishna Priya V R
Ph.D Student (21CD91R05)
Center for computational and Data Sciences

Lambu Kushi Reddy
B.Tech (20ME30032)
Department of Mechanical Engineering

1. Defect detection

1.1 Problem Description

To win in today's highly competitive global market, one needs to supply a quality product. Thus every organization must optimize its manufacturing process by making the nearly perfect product. Thus it is important to accurately detect the defect in the product before it hits the market. Thus in the stage of manufacturing itself, the items used in the manufacturing must be defect-free. Therefore defect detection is a core part of any manufacturing quality control assurance process. Manual identification of defects is not an efficient way to tackle this problem in the case of large-scale manufacturing. Artificial Intelligence has come to the rescue here.

Relying on the human expert's rules, these models learn on their own which feature is important and create implicit rules that determine which combination of features impact overall product quality. Some significant benefits for manufactures by using AI-based defect detection systems are reduced labor, increased production volume, early error detection, improved manufacturing efficiency, and tracking historical data to pinpoint issues and improve future production processes.

In this project, we identify defects in casting manufacturing products. Casting is a manufacturing process in which a liquid material is usually poured into a mould, which contains a hollow cavity of the desired shape and then allows them to solidify. But various defects can occur during the process of casting such as blowholes, pinholes, burr, shrinkage defects, mould material defects, pouring metal defects, metallurgical defects, etc. For removing this defective product all industry have their quality inspection department. But since the inspection is carried out manually, it is very time-consuming and due to human accuracy, this is not 100% accurate. This can cause rejection of the order and hence big loss for the company. The inspection process can be made automatic using AI.

1.2 Formal Problem Statement

The problem of defect detection using AI can be formulated as a clustering problem, that is grouping similar data together. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. Clustering is used for the categorization of objects into one or more classes based on the features. In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. There are different types of clustering algorithms such as K-means clustering, Fuzzy C- means clustering algorithm, Gaussian clustering algorithm, etc. The simplest clustering algorithm is the K-means clustering.

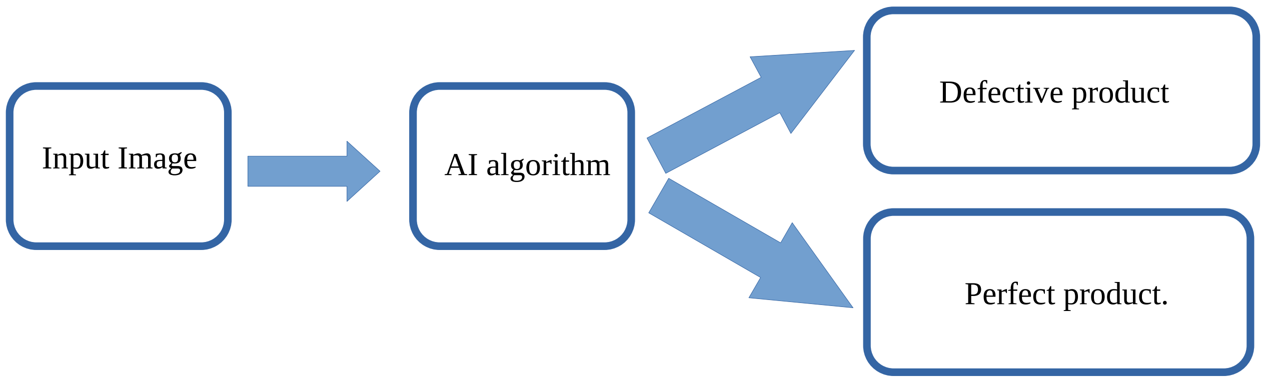


Figure 1: Flowchart showing input and output

The casting process defect detection can be formulated using a clustering algorithm.
The logic behind the clustering algorithm is that
1. Assume two random points anywhere near the data and consider them as the center of two clusters, one defective product image data cluster, and one perfect product image data cluster.
2. Find the distance of every data point from both centers.
3. Assign every data point to the centroid to which it is the nearest. This gives rise to two clusters.
4. Calculate the center of both the formed clusters and shift the centroids there.
5. Go to step 1 and repeat the process until there is no change in clusters formed.

It can also be formulated as a classification problem. But the complexity of clustering is less compared to any classification algorithm. Thus in this project, we are using a clustering algorithm to group the products based on similarity.
AI solutions used for quality control utilize Machine learning and defect prediction models to autonomously learn and make inferences from the Manufacture data. A machine learning model will use this training data and calculate how to best map examples of the input data to specific class labels.

2. AI Modelling

This problem of defect detection is modeled using an unsupervised machine learning algorithm called K-means clustering. Machine learning is a type of artificial intelligence that allows software applications to become more accurate in predicting the outcomes without being explicitly programmed to do so. Machine learning focuses on the development of computer programs that can access data and then use it to learn for themselves. Machine learning algorithms are often categorized into supervised and unsupervised. The main difference is one uses labeled data to help predict outcomes, while the other does not.

K- means clustering is one of the simplest unsupervised clustering algorithms which is used to cluster our data into K number of clusters. The algorithm iteratively assigns the data points to one of the K clusters based on how near the point is to the cluster centroid. Cluster centroid is a middle of a cluster or the arithmetic mean of all the data points that belong to that cluster.

2.1 K-means clustering algorithm.

The K means algorithm works as follows.

1. Specify the number of clusters K.(In this case 2)
2. Pick K points as the initial centroids from the dataset, either randomly or the first K.
3. Find the Euclidean distance of each point in the dataset with the identified K points (cluster centroids).
4. Assign each data point to the closest centroid using the distance found in the previous step.
5. Find the new centroid by taking the average of the points in each cluster group.
3. Keep iterating until there is no change to centroids.

If **p**=(p1,p2) and **q**=(q1,q2) are two points, the euclidean distance is given by

$$d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{(q_1 - p_1)^2 + (q2 - p2)^2}$$

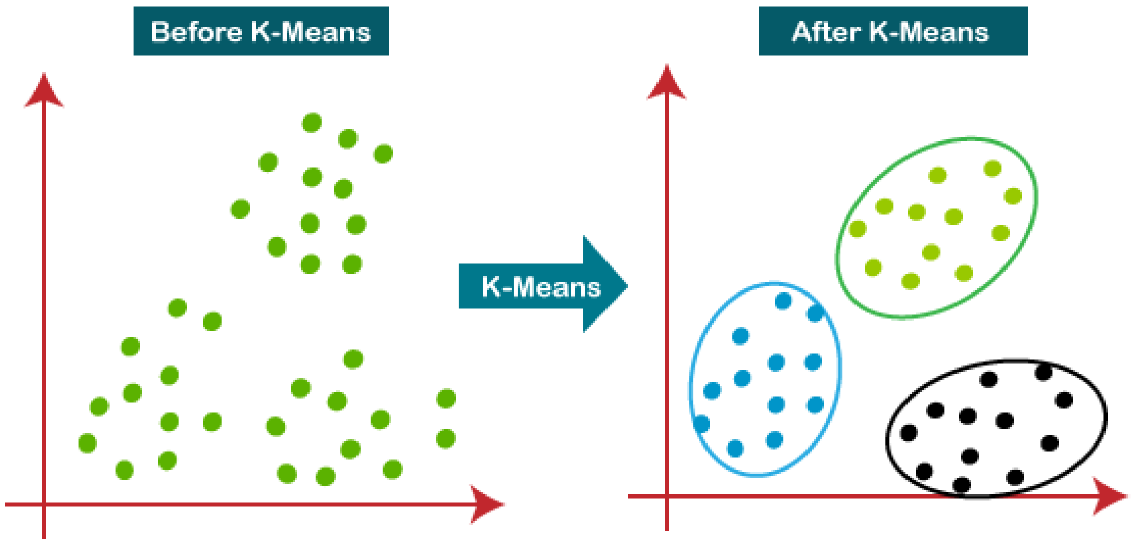


Figure 2: K-means clustering

The mini-batch K means clustering algorithm is a version of standard K-means algorithm. Mini K means algorithm can be used instead of K means algorithm while working on huge datasets because it doesnt iterate over the entire dataset. Sometimes it can even perform better than K means algorithm when working on huge datasets. It uses small random, fixed size batches of data to store in memory, and then with each irteration, a random sample of data is collected and used to udate the clusers. The advantage of using mini K-means algorithm is that it reduces the computational cost of finding a cluster.

3. Solution Approach

A dataset having a total of 7348 images of a submersible pump impeller is taken. They are all grey-scaled images. There are two categories of images,
1. Defective
2. Perfect

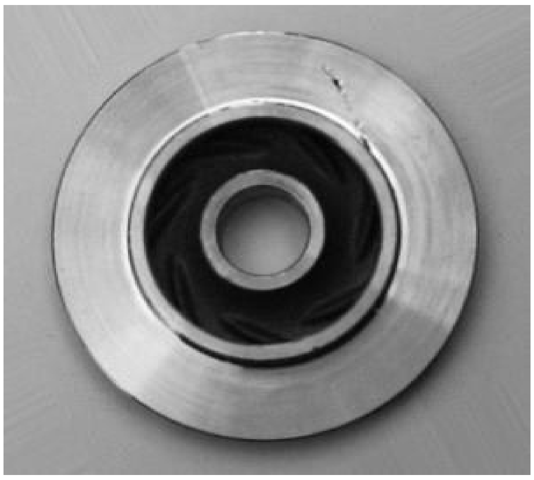


Figure 3: Defective

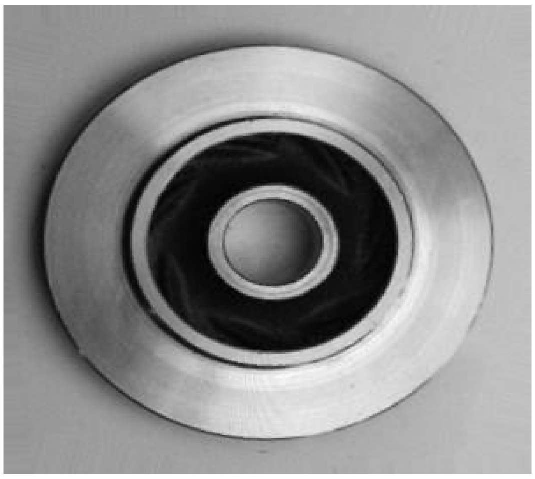


Figure 4: Perfect

The training set contains 3758 defective images and 2875 perfect images. The test data contains 453 defective images and 262 perfect images. This dataset is available on Kaggle for free. This data is used to train and test the validity of the algorithm. Since we are working on huge dataset we use Mini Batch K-means to model the problem.
One of the most important steps in image classification is preprocessing of the data. Since the dataset contains so many images, they need to be normalized. Data normalization is an important preprocessing step that ensures that each input parameter has a similar data distribution. This fastens the process of convergence while training the model. Normalization makes sure no one particular parameter influences the output significantly. The computations and programming to train and test the K mean clustering machine learning model is written in python.

The data of the image is three-dimensional. The shape of x_train is (6633,200,200) and x_test is (715,200,200). It has to be converted to 2D data to be used as the training data to be fed into the K-means Clustering.

```
# Reshaping input data:


X_train = x_train.reshape(len(x_train),-1)
X_test = x_test.reshape(len(x_test),-1)
```

The shape of the training and test data is checked using the x_train.shape and x_test.shape. The output is denoted by X_train whose dimension is (6633,40000) and X_test is (715, 40000). The y_train and y_test contain an array of ones and zeros. One's corresponds to the number of perfect images and zero's correspond to defective images data in X_train and X_test.

After preprocessing this data can be used to build the model with mini Batch K-means. In this, the most computationally costly step is conducted on only a random sample of observations as opposed to all observations. To do this, first from sklearn.cluster, import the MiniBatchKMeans. Sklearn or Scikit-learn is the most useful and robust library in python for machine learning. After importing the total number of clusters is calculated. The next step is initializing the K-means model. After initialization, the training set is fitted into this model using kmeans.fit.

```
from sklearn.cluster import MiniBatchKMeans
total_clusters = len(np.unique(y_test))

# Initialize the K-means model
kmeans = MiniBatchKMeans(n_clusters = total_clusters)
#Fitting model to the training set
kmeans.fit(X_train)
```

Now run the kmeans.label_.

```
kmeans.labels_
```

It assigns each image a cluster label value. It is an array of length 6633 as there are 6633 images in the training set. The images are classified into clusters based on the similarity of pixel values. This won't say if the product in the image is defective or not. Thus we need a retrieve function to get the necessary information from kmeans.labels_.

```
def retrieve_info(cluster_labels,y_train):
    reference_labels = {}
    for i in range(len(np.unique(kmeans.labels_))):
        index = np.where(cluster_labels == i,1,0)
        num = np.bincount(y_train[index==1]).argmax()
        reference_labels[i] = num
    return reference_labels
```

Use the print function to find the output of the referebce_labels.

```
print(reference_labels)
```

The output comes here as {0:0, 1:0}. A cluster labeled as 0 is the cluster of defective images and also the perfect images. Thus these clusters don't distinguish between the defective images and perfect images. We can also calculate the accuracy score.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(number_labels,y_train))
```

The accuracy score is about 56.65%. This is not a good result. Thus we need to optimize the algorithm.

3.1 Optimizing the algorithm

The algorithm is optimized for better results. The performance of the model is measured using 3 metrics. One is inertia, Homogeneity score, and Accuracy score. Inertia measures how well a dataset was clustered by K-means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A clustering result satisfies homogeneity if all of its clusters contain only data points that are members of a single class. Accuracy score is the percentage of correctly predicted values. The function below calculates all the metrics for the model.

```
def calculate_metrics(model,output):
    print('Number of clusters is {}'.format(model.n_clusters))
    print('Inertia : {}'.format(model.inertia_))
    print('Homogeneity: {}'.format(metrics.homogeneity_score(output,model.labels_)))
```

```
from sklearn import metrics
cluster_number = [10,16,36,64,144,256]
for i in cluster_number:
    total_clusters = len(np.unique(y_test))
    kmeans = MiniBatchKMeans(n_clusters = i)
    kmeans.fit(X_train)
    kmeans.labels_
    reference_labels = retrieve_info(kmeans.labels_,y_train)
    print(reference_labels)
    calculate_metrics(kmeans,y_train)
    reference_labels = retrieve_info(kmeans.labels_,y_train)
    number_labels = np.random.rand(len(kmeans.labels_))

    for i in range(len(kmeans.labels_)):
        number_labels[i] = reference_labels[kmeans.labels_[i]]
    print('Accuracy score : {}'.format(accuracy_score(number_labels,y_train)))
    print('\n')
```

We can run the model for different values of the number of clusters. Also calculated are the reference_labels for each value of clusters. The output is given below:

```
{0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 1, 9: 0}
Number of clusters is 10
Inertia : 4760183.0
Homogeneity: 0.05057120144888515
Accuracy score : 0.6120910598522539


{0: 1, 1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 0, 8: 1, 9: 0, 10: 0, 11: 0, 12: 0, 13: 1, 14: 1, 15: 0}
Number of clusters is 16
Inertia : 4429318.5
Homogeneity: 0.17958686548085034
Accuracy score : 0.7143072516206844


{0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 1, 11: 1, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17:
1, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 1, 24: 0, 25: 0, 26: 1, 27: 0, 28: 1, 29: 1, 30: 0, 31: 0, 32: 1, 33: 1,
34: 1, 35: 0}
Number of clusters is 36
Inertia : 3849421.0
Homogeneity: 0.3314590903788137
Accuracy score : 0.783355947535052


{0: 1, 1: 0, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 1, 12: 0, 13: 0, 14: 1, 15: 0, 16: 0, 17:
0, 18: 1, 19: 1, 20: 1, 21: 1, 22: 0, 23: 0, 24: 1, 25: 1, 26: 0, 27: 1, 28: 1, 29: 1, 30: 1, 31: 0, 32: 1, 33: 1,
34: 0, 35: 1, 36: 1, 37: 1, 38: 1, 39: 1, 40: 1, 41: 0, 42: 1, 43: 1, 44: 1, 45: 0, 46: 1, 47: 0, 48: 1, 49: 0, 5
0: 0, 51: 1, 52: 1, 53: 0, 54: 1, 55: 0, 56: 1, 57: 1, 58: 0, 59: 0, 60: 0, 61: 0, 62: 0, 63: 1}
Number of clusters is 64
Inertia : 3402359.5
Homogeneity: 0.4120129996201696
Accuracy score : 0.8068747173224785


{0: 1, 1: 1, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 1, 10: 0, 11: 0, 12: 1, 13: 0, 14: 0, 15: 1, 16: 0, 17:
0, 18: 1, 19: 0, 20: 0, 21: 1, 22: 0, 23: 0, 24: 0, 25: 1, 26: 1, 27: 0, 28: 1, 29: 1, 30: 0, 31: 0, 32: 1, 33: 1,
34: 1, 35: 1, 36: 0, 37: 1, 38: 1, 39: 0, 40: 0, 41: 0, 42: 1, 43: 0, 44: 1, 45: 1, 46: 0, 47: 1, 48: 0, 49: 1, 5
0: 0, 51: 1, 52: 1, 53: 0, 54: 0, 55: 1, 56: 1, 57: 1, 58: 0, 59: 1, 60: 1, 61: 0, 62: 1, 63: 0, 64: 0, 65: 0, 66:
0, 67: 1, 68: 0, 69: 1, 70: 1, 71: 1, 72: 0, 73: 1, 74: 0, 75: 0, 76: 1, 77: 0, 78: 1, 79: 0, 80: 1, 81: 0, 82: 1,
83: 0, 84: 0, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 0, 92: 1, 93: 0, 94: 0, 95: 1, 96: 0, 97: 1, 98: 1, 9
9: 0, 100: 0, 101: 0, 102: 0, 103: 0, 104: 1, 105: 0, 106: 0, 107: 0, 108: 1, 109: 0, 110: 1, 111: 0, 112: 1, 113:
1, 114: 0, 115: 0, 116: 1, 117: 0, 118: 1, 119: 0, 120: 0, 121: 1, 122: 1, 123: 0, 124: 0, 125: 0, 126: 0, 127: 1,
128: 1, 129: 1, 130: 1, 131: 1, 132: 1, 133: 1, 134: 1, 135: 1, 136: 1, 137: 0, 138: 1, 139: 0, 140: 0, 141: 1, 14
2: 1, 143: 1}
Number of clusters is 144
Inertia : 2780119.0
Homogeneity: 0.5664103480033359
Accuracy score : 0.8608472787577265


{0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 1, 16: 1, 17:
1, 18: 1, 19: 0, 20: 0, 21: 0, 22: 0, 23: 1, 24: 0, 25: 1, 26: 0, 27: 1, 28: 0, 29: 1, 30: 1, 31: 0, 32: 0, 33: 0,
34: 0, 35: 0, 36: 0, 37: 0, 38: 1, 39: 1, 40: 0, 41: 1, 42: 0, 43: 1, 44: 1, 45: 1, 46: 1, 47: 1, 48: 1, 49: 1, 5
0: 1, 51: 0, 52: 0, 53: 0, 54: 1, 55: 0, 56: 0, 57: 0, 58: 0, 59: 0, 60: 1, 61: 1, 62: 0, 63: 1, 64: 1, 65: 1, 66:
1, 67: 0, 68: 1, 69: 0, 70: 1, 71: 1, 72: 1, 73: 1, 74: 0, 75: 1, 76: 1, 77: 1, 78: 0, 79: 1, 80: 1, 81: 0, 82: 0,
83: 1, 84: 1, 85: 1, 86: 1, 87: 1, 88: 1, 89: 1, 90: 1, 91: 1, 92: 0, 93: 1, 94: 0, 95: 1, 96: 0, 97: 1, 98: 0, 9
9: 1, 100: 0, 101: 1, 102: 1, 103: 0, 104: 0, 105: 1, 106: 1, 107: 0, 108: 0, 109: 0, 110: 1, 111: 1, 112: 0, 113:
0, 114: 0, 115: 0, 116: 0, 117: 0, 118: 1, 119: 1, 120: 0, 121: 1, 122: 1, 123: 1, 124: 0, 125: 0, 126: 0, 127: 1,
128: 1, 129: 1, 130: 0, 131: 0, 132: 0, 133: 1, 134: 1, 135: 1, 136: 0, 137: 0, 138: 0, 139: 0, 140: 1, 141: 1, 14
2: 1, 143: 0, 144: 0, 145: 0, 146: 1, 147: 1, 148: 0, 149: 1, 150: 1, 151: 0, 152: 1, 153: 1, 154: 0, 155: 1, 156:
1, 157: 1, 158: 1, 159: 1, 160: 1, 161: 1, 162: 1, 163: 0, 164: 0, 165: 1, 166: 1, 167: 1, 168: 0, 169: 1, 170: 0,
171: 1, 172: 1, 173: 1, 174: 0, 175: 0, 176: 0, 177: 1, 178: 0, 179: 0, 180: 1, 181: 0, 182: 0, 183: 1, 184: 0, 18
5: 1, 186: 1, 187: 0, 188: 0, 189: 0, 190: 0, 191: 0, 192: 0, 193: 1, 194: 0, 195: 0, 196: 1, 197: 0, 198: 1, 199:
1, 200: 1, 201: 0, 202: 0, 203: 1, 204: 1, 205: 1, 206: 0, 207: 0, 208: 0, 209: 0, 210: 1, 211: 1, 212: 0, 213: 1,
214: 0, 215: 0, 216: 0, 217: 0, 218: 0, 219: 1, 220: 0, 221: 0, 222: 1, 223: 0, 224: 1, 225: 0, 226: 0, 227: 1, 22
8: 1, 229: 0, 230: 1, 231: 1, 232: 1, 233: 0, 234: 0, 235: 0, 236: 0, 237: 0, 238: 0, 239: 0, 240: 0, 241: 0, 242:
1, 243: 0, 244: 0, 245: 1, 246: 0, 247: 1, 248: 1, 249: 0, 250: 0, 251: 1, 252: 0, 253: 1, 254: 0, 255: 0}
Number of clusters is 256
Inertia : 2404986.0
Homogeneity: 0.6621658077341449
Accuracy score : 0.8902457409920097
```

3.2 Observatios and conclusion

Several observations can be made from this output. Firstly, the inertia score decreases because the sum of squares of the distance between data points and their respective clusters centroid decreases, and clusters become more inherently coherent. The second observation is that the homogeneity score increases the clusters become more differentiable and the number of data points having a single class label is high. The third observation is that the accuracy score increases with an increase in the number of clusters.

From this, we can conclude that the K-means clustering model runs successfully on this dataset with an accuracy of nearly 90%. Thus it was able to cluster the images into defective and perfect ones. But can we use it to predict other images of defective and perfect products is a question to be asked? To check this we imported an image and preprocessed it. Using the command kmeans.predit(), we successfully identified the product as defective from an image of defective material and perfect from a perfect image.

References

1. Evgeniy Krasnokutsky(2021)**AI Visual Inspection For Defect Detection**
2. Huseyn Gasimov(2021) **Automated Defect Detection (complete pipeline and demo)**
3. Yang J, Li S, Wang Z, Dong H, Wang J, Tang S. Using Deep Learning to Detect Defects in Manufacturing: A Comprehensive Survey and Current Challenges. *Materials*. 2020; 13(24):5755.