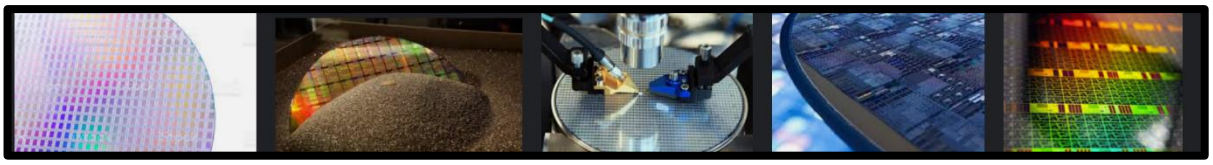
Indian Institute of Technology, Roorkee
ECN-358 Project Report

Semiconductor Wafer Defect Classification

Tanya Rampal, Btech 3rd yr ECE, 19116080



INTRODUCTION

In electronics, a wafer is a thin slice of semiconductor, such as a crystalline silicon (c-Si), used for the fabrication of integrated circuits. Even though other conductors are employed in more particular applications, silicon is the best and the most used semiconductor due to its extreme mobility both at high temperatures and at room temperature. In today's world, almost every electronic device (computers, cell phones, cars, televisions, digital watches) use ICs made out of Si wafers due to its small size and high reliability and efficiency.

However, due to imperfections during the manufacturing of wafers and then ICs, certain defects occur on the wafers due to foreign particles, contact rubbing causing scratches, solvent and other chemical residues etc. Thus, defect detection is an integral part of wafer (chip) fabrication process. Once the defect is detected, it needs to be classified in order to enable the correction of the fabrication process. This classification is based on the defect properties such as specific patterns, geometries, frequency of imperfections etc.

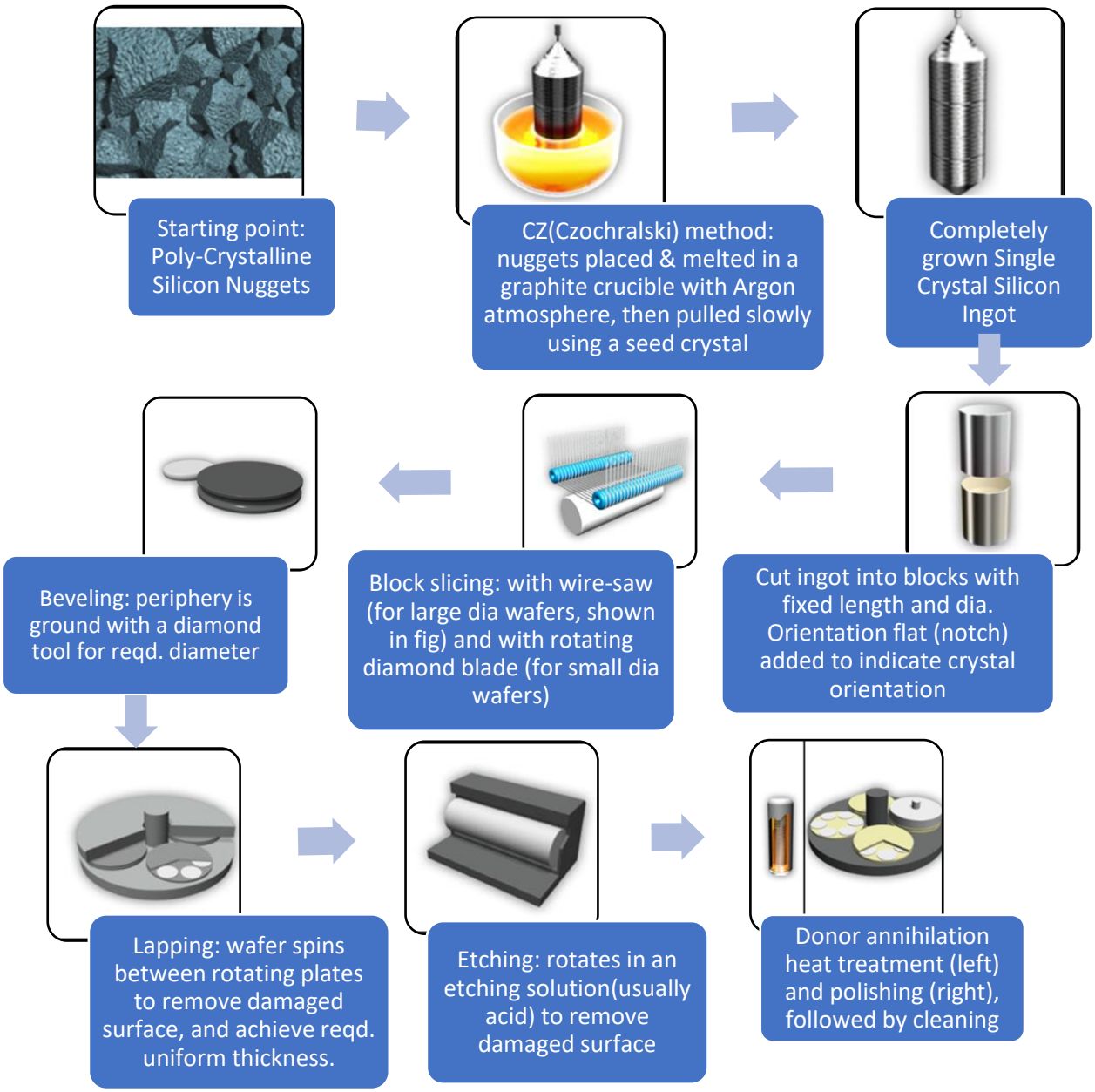
In this project, construction of a model for defect classification of fully manufactured Si wafer chips was attempted via machine learning classification models, using the dataset WM811K provided by [MIR Labs], with the image array of the chip taken as the features, and the defect classes as the target. In this report, firstly, an overview of the manufacturing process of Si wafers and ICs is covered, then the process identifying defective wafers via image mapping is studied. Finally, the methodology of making the machine learning classification model is gone through stepwise, and the final model selected is used to further classify the instances within the 'none' category. Based on this model, certain conclusions and insights are drawn.

Note: The dataset (WM811K) used is too large to be uploaded on Google drive, hence a link to the download it as well as the original source link is given in this report at the end (just before references)

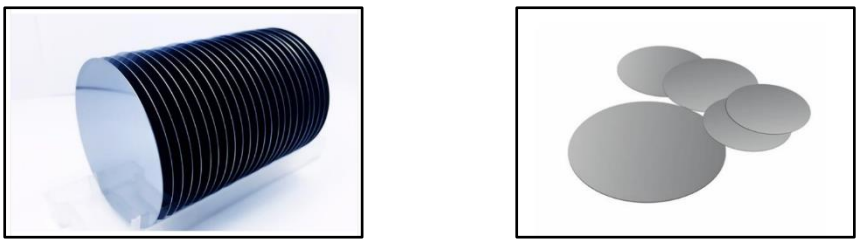
WAFER MANUFACTURING PROCESS

- **Overview of the Si wafer and chip manufacturing process**

Wafer Manufacture Process



End product of above process: Circular Si wafers:

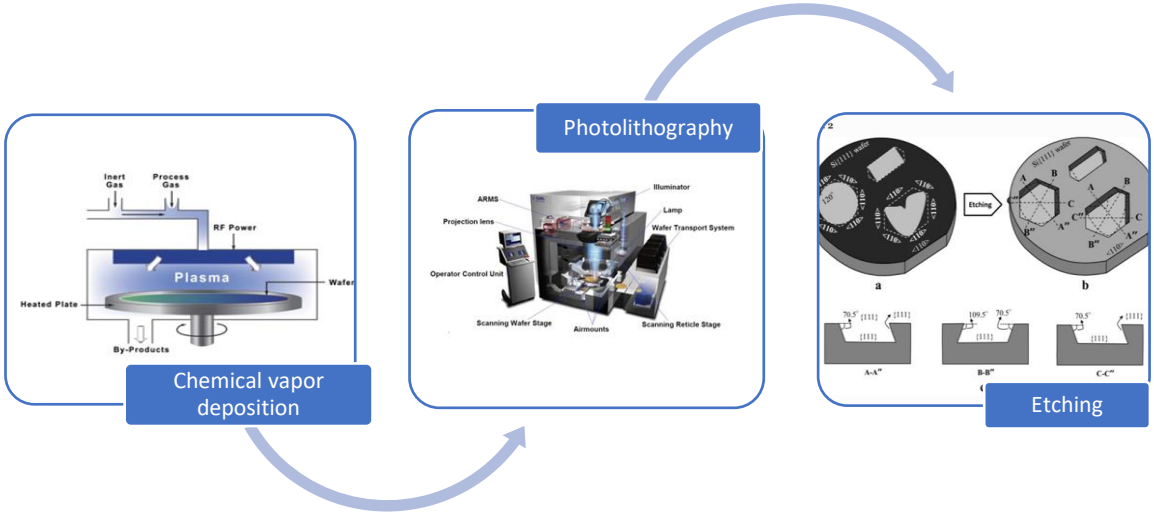


ICs from above wafers:

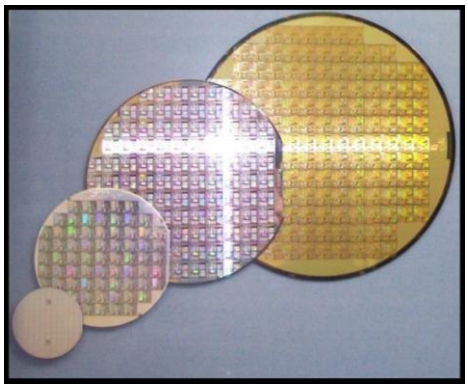
In the manufacturing process of ICs, electronic circuits with components such as transistors are formed on the surface of a silicon crystal wafer.

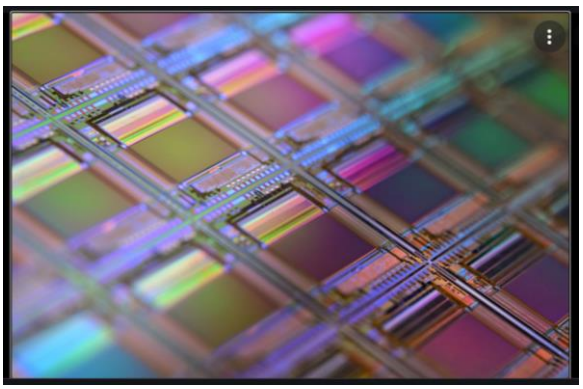
Basics of IC formation:

1. A thin film layer that will form the wiring, transistors and other components is deposited on the wafer (deposition, usually done via chemicals).
2. The thin film is coated with photoresist. The circuit pattern of the photomask (reticle) is then projected onto the photoresist using Photolithography technology.
3. The developed photoresist is used as a mask for etching to process the thin film into the shape of the wiring and other components.



End product of above process: Patterned Si wafers:



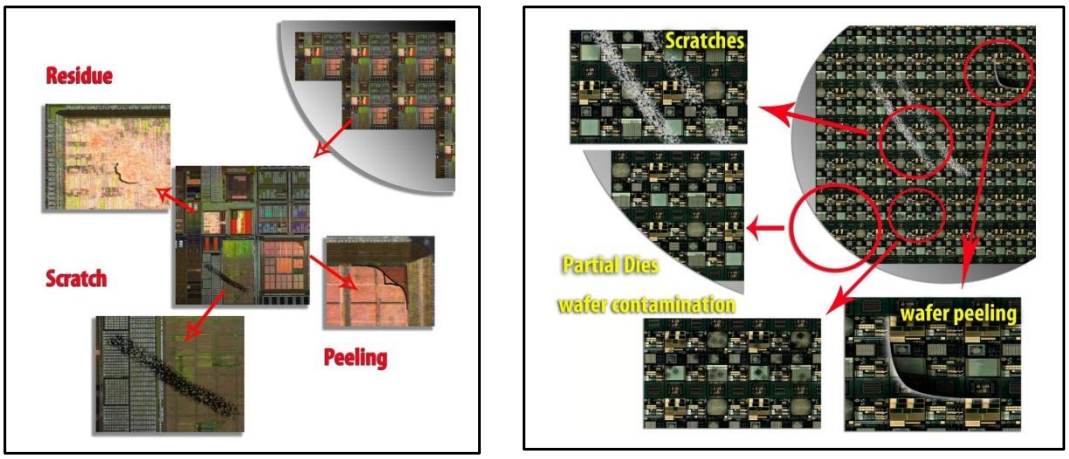


With this, one layer of the circuit is formed. The transistors are formed on the lowest layer. Similar process is then repeated, and many layers of circuits are formed on top of one another.

Each transistor on this layer is known as a ‘chip’ and they have a certain ‘die size’ that was determined in step 1 based on their usage. However, due to all the processes above, certain defects remain on the wafer, which are determined in the following manner.

DEFECT DETECTION PROCESS

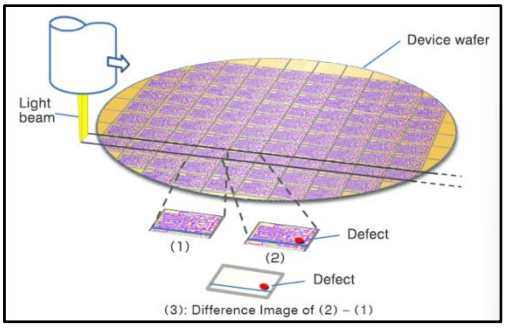
Wafer defect inspection system detects physical defects (foreign substances called particles) and pattern defects on wafers and obtains the position coordinates (X, Y) of the defects. An image (2D array) of the wafer is mapped with all the defective positions highlighted (similar to the 'waferMap' column in the WM811K dataset used in this project).



Visual defect examples on the wafer

Defects can be divided into random defects and systematic defects. Random defects are mainly caused by particles that become attached to a wafer surface, so their positions cannot be predicted. On the other hand, systematic defects are caused by the conditions of the mask and exposure process, and will occur in the same position on the circuit pattern of all the dies projected. They occur in locations where the exposure conditions are very difficult and require fine adjustment. There are 2 main typical inspection systems; Patterned wafer inspection system and Non-patterned wafer inspection system. Since our project is concerned with patterned wafers, we will focus only on the former:

Patterned wafer inspection system:



Patterned wafer inspection system can detect defects by comparing the pattern images of adjacent chips (also called dies) and obtaining the difference. The pattern on the wafer is captured along the die array by electron beam or light. Defects are detected by comparison between image (1) of the die to be inspected and image (2) of the adjacent die. If there are no defects, the result of the subtraction of Image 2 from Image 1 by digital processing will be zero and no defects are detected. In contrast, if there is a defect in the image of die (2), the defect will remain in the subtracted image (3) as shown in the figure.

After all the above processes are completed, all wafer's data and defect information (present/not present and classes) is tabulated. Many datasets with such defective wafer classifications are available, and in this project, using the WM811K dataset, a model was built to predict the defect in the wafer in machine learning using the following methodology:

METHODOLGY

1. Importing the data:

- Firstly the .pkl file containing the WM811K dataset was imported and displayed, and it was seen that it was of the form:

```
df.head()
```

	waferMap	dieSize	lotName	waferIndex	trianTestLabel	failureType
0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	1.0	[[Training]]	[[none]]
1	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	2.0	[[Training]]	[[none]]
2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	3.0	[[Training]]	[[none]]
3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	4.0	[[Training]]	[[none]]
4	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	5.0	[[Training]]	[[none]]

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 811457 entries, 0 to 811456
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   waferMap        811457 non-null  object
 1   dieSize         811457 non-null  float64
 2   lotName         811457 non-null  object
 3   waferIndex      811457 non-null  float64
 4   trianTestLabel  811457 non-null  object
 5   failureType     811457 non-null  object
dtypes: float64(2), object(4)
memory usage: 37.1+ MB
```

There are 6 columns:

1. waferMap: It is a 2D array representing the image of every wafer.
2. dieSize: dimensions of a particular square Si chip
3. lotName: the testing was done in lots/batches, thus it just represents the lot serial.
4. trianTestLabel: dividing (only) the defected wafers into training and testing
5. failureType: the defect present in that particular wafer. It has 9 (including[[‘none’]]) classes, or no entry(no defect)

Thus for the model, we will need the column ‘**waferMap**’ ie the image arrays as the features (x) and the ‘**failureType**’ as the target (y) in our model, which will predict one out of 8 defect classes.

- A new column ‘failure_ordinal_enc’ was added to the dataset which contains numbers 0-8 for every defective instance (ordinal encoding of a sort). This column made it easy to make new dataframes containing only defective data, only defective data(without [[‘none’]]) etc. using these datasets, the following data was extracted (the right figure).

```
total_wafers_num = df.shape[0]
total_wafers_num
```

```
811457
```

```
print("Wafers with a defect:", df_with_defect_num)
print("Wafers with a defect(excluding none):", df_with_pattern_num)
print("Wafers with defect 'none':", df_with_none_num)
```

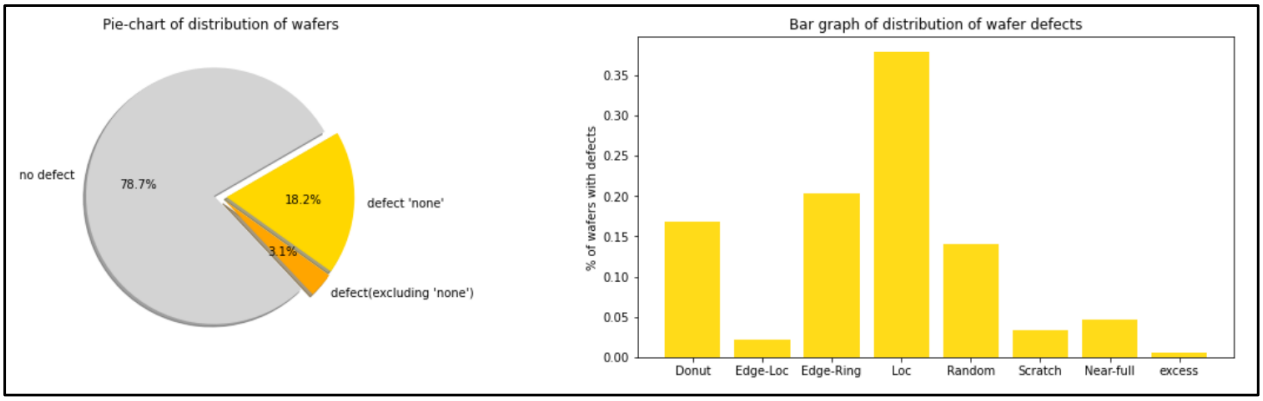
```
Wafers with a defect: 172950
Wafers with a defect(excluding none): 25519
Wafers with defect 'none': 147431
```

The total number of wafers was found to be 811457, out of which only 172950 (approx. 1/8th of the data) had a defect. Out of these 172950 defective wafers, 147431 had been classified as [[‘none’]], meaning they didn’t come under the other specific 8 classes.

2. Data analysis:

2.1 Distribution of data

- Using <plt.subplot>, and the new datasets formed in the previous section, a pie chart and a bar graph showing the distribution of data was plotted as shown:

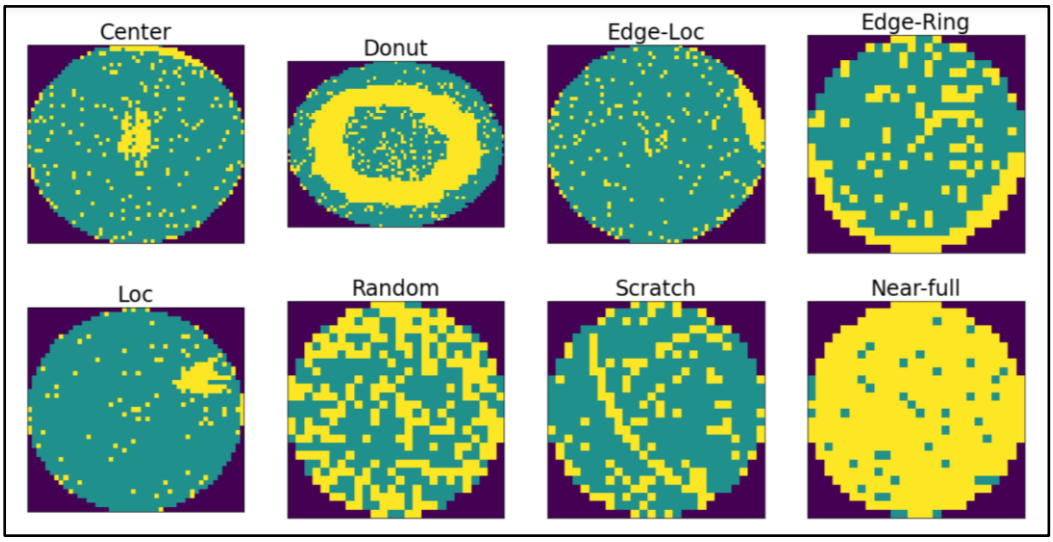


- As observed in the previous section, the pie chart shows us that the amount of wafers with defects is quite a small percentage (ie only 18.2+3.1 = 21.3%) of the entire dataset, and out of that too, 18.2% has been classified as [[‘none’]], leaving us only 3.1% of dataset (25519 instances) with actual defect labels. This 21.3% and 3.1% of data is our main focus in this report.

- Out of these 25519 classified defective instances, from the bar graph, we can see that [[‘Loc’]] defect occurs more frequently, followed by [[‘Edge-Ring’]] and [[‘Donut’]]. It can also be concluded that the dataset shows high imbalance distribution.

2.2 Visualisation of defects

- Selecting random instances of each kind of defect, and plotting it using < plt.subplot> and <imshow>, the following visualisation of each defect was obtained:



- Thus, it is apparent that the defect names are quite self-explanatory, for eg [[‘Center’]] has defective chips focussed at the center, [[‘Edge-Loc]] has defective chips concentrated at one edge and so on.

3. Data preprocessing:

3.1 Checking the image dimensions

- To make the feature (x=’waferMap’) be suitable for the classifiers, all the rows should have the same image size. So a new column named ‘waferMapDim’ (wafer map dimensions) is made that has the size of each wafer array. A sample is printed as shown:

```
# creating a new column
def find_dim(x):
    dim_x=np.size(x,axis=0)
    dim_y=np.size(x,axis=1)
    return dim_x,dim_y
df['waferMapDim']=df.waferMap.apply(find_dim)
df.sample(5)
```

	waferMap	dieSize	lotName	waferIndex	trianTestLabel	failureType	failure_ordinal_enc	waferMapDim
85099	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2,...	939.0	lot6095	8.0	[]	[]	[]	(39, 31)
167690	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	572.0	lot10682	10.0	[]	[]	[]	(22, 35)
505340	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,...	710.0	lot31396	23.0	[]	[]	[]	(32, 29)
309965	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,...	1139.0	lot18864	1.0	[]	[]	[]	(38, 38)
223583	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,...	1109.0	lot14074	22.0	[]	[]	[]	(39, 37)

- As observed, each row has different 2D array sizes, thus, this column has to modified to have the same size in order for it to be suitable for the classifier models.

3.2 Making new datasets for defective data, and splitting into test-train

- New dataframes for instances with defects, and those with defects(excluding 'none') were made using the previously made column ‘failure_ordinal_enc’ (which also includes the new column ‘waferMapDim’ now). Using the new dataframe ‘df_with_pattern’, training and testing datasets were made using the existing labels within column ‘trianTestLabel’

- The maximum wafer dimension of the defective dataset (including none) was found to be (212, 84).

```
max(df_with_defect.waferMapDim)
```

```
(212, 84)
```

3.3 Modifying training and testing data to fit classifiers

- By the ‘.resize (220,90)’ function, the image size of all the wafers was changed to (220,90) and then made into a 1D array to be eligible to fit in the classifier models. A new list of these features, ie 1D arrays (namely x_train and x_test) was formed.
- The target was modified to contain only the classname (ie in the form of ‘Center’ instead of [[‘Center’]], to make the classification give the output in strings. These targets were stored in y_train and y_test.

```
# making wafer map dimensions (size) equal to fit in the classifier
x_train=[]
for i in range(x_train_unproc.shape[0]):
    img=x_train_unproc['waferMap'][i].copy()
    img.resize(220,90)
    img_new=img.reshape(-1)
    x_train.append(img_new)

# changing y values from nested list to string type
y_train=[type[0][0] for type in y_train_unproc['failureType']]
```

4. Models:

- 3 models, namely Logistic regression, K nearest neighbours, and Random forest generator, and same methods have been followed for each model (example of the first model is given, rest 2 follow same format):
    - The classification model was imported and fitted on the test train data, then the classes were displayed to cross verify.

```
from sklearn.linear_model import LogisticRegression

log_reg=LogisticRegression()
log_reg.fit(x_train, y_train)
```

```
log_reg.classes_
```

- Then predicted values on the training data was calculated, and on the basis on of this, a classification report was printed, along with a confusion matrix to analyse the classifier. The diagonal elements of this matrix are given special attention as they quantify the correctly predicted values.

```
y_pred_train=log_reg.predict(x_train)

print(classification_report(y_train, y_pred_train))
cm=confusion_matrix(y_train, y_pred_train)
print('Diagonal elements of confusion matrix: ')
for i in range(0,cm.shape[0]):
    print(cm[i][i], end =", "),
cm
```

- The above process was repeated for testing as well. The results of the testing data help us narrow down to one classifier.

```
y_pred_test=log_reg.predict(x_test)

print(classification_report(y_test, y_pred_test))
cm=confusion_matrix(y_test, y_pred_test)
print('Diagonal elements of confusion matrix: ')
for i in range(0,cm.shape[0]):
    print(cm[i][i], end =", "),
cm
```

- The results of the 2nd and 3rd step of above method for each model is given below:

4.1 Logistic Regression

- Training results:

```
              precision    recall  f1-score   support

      Center       0.95      0.95      0.95      3462
       Donut       0.94      0.92      0.93       409
    Edge-Loc       0.81      0.89      0.85      2417
   Edge-Ring       0.99      0.99      0.99      8554
         Loc       0.82      0.78      0.80      1620
    Near-full       0.60      0.22      0.32        54
      Random       0.92      0.84      0.88       609
     Scratch       0.85      0.74      0.79       500

    accuracy                           0.93     17625
   macro avg       0.86      0.79      0.81     17625
weighted avg       0.93      0.93      0.93     17625

Diagonal elements of confusion matrix:
3291, 376, 2149, 8447, 1261, 12, 514, 371,

array([[3291,    7,   81,    7,   60,    0,    5,   11],
       [   6,  376,   12,    2,   13,    0,    0,    0],
       [  59,    3, 2149,   57,  116,    5,   12,   16],
       [   6,    0,   83, 8447,    6,    0,    6,    6],
       [  73,   15,  215,   12, 1261,    2,   15,   27],
       [   2,    0,   23,    0,   11,   12,    6,    0],
       [  15,    1,   43,    3,   29,    1,  514,    3],
       [  14,    0,   63,    7,   42,    0,    3,  371]],
```

- Testing results:

```
              precision    recall  f1-score   support

      Center       0.20      0.38      0.27       832
       Donut       0.20      0.03      0.05       146
    Edge-Loc       0.39      0.38      0.39      2772
   Edge-Ring       0.47      0.30      0.37      1126
         Loc       0.31      0.30      0.30      1973
    Near-full       0.12      0.01      0.02        95
      Random       0.04      0.12      0.06       257
     Scratch       0.11      0.03      0.05       693

    accuracy                           0.30      7894
   macro avg       0.23      0.19      0.19      7894
weighted avg       0.32      0.30      0.30      7894

Diagonal elements of confusion matrix:
317, 4, 1050, 340, 588, 1, 30, 23,

array([[ 317,    1,  244,   43,  159,    0,   41,   27],
       [  22,    4,   15,    0,   97,    0,    1,    7],
       [ 604,    3, 1050,  191,  689,    4,  202,   29],
       [ 102,    0,  248,  340,   77,    0,  343,   16],
       [ 323,    7,  757,   88,  588,    1,  127,   82],
       [  13,    1,   39,    2,   17,    1,   10,   12],
       [  57,    2,   75,    4,   81,    0,   30,    8],
       [ 120,    2,  251,   49,  189,    2,   57,   23]],
```

4.2 K-Nearest Neighbours

- Training results:

```
              precision    recall  f1-score   support

      Center       0.90      0.89      0.89      3462
       Donut       0.95      0.85      0.90       409
    Edge-Loc       0.83      0.72      0.77      2417
   Edge-Ring       0.98      0.99      0.98      8554
         Loc       0.54      0.79      0.64      1620
   Near-full       0.67      0.83      0.74        54
      Random       0.99      0.68      0.80       609
     Scratch       0.68      0.45      0.54       500

    accuracy                           0.88     17625
   macro avg       0.82      0.77      0.78     17625
weighted avg       0.90      0.88      0.88     17625

Diagonal elements of confusion matrix:
3067, 348, 1731, 8446, 1277, 45, 412, 225,

array([[3067,    3,   24,    2,  345,    0,    1,   20],
       [  17,  348,   11,    6,   24,    0,    0,    3],
       [  90,    1, 1731,   74,  488,    4,    0,   29],
       [  14,    0,   42, 8446,   48,    0,    0,    4],
       [ 105,   11,  158,   20, 1277,    1,    1,   47],
       [   3,    0,    4,    0,    0,   45,    2,    0],
       [  56,    3,   64,   27,   27,   17,  412,    3],
       [  52,    0,   45,   22,  156,    0,    0,  225]],
```

- Testing results:

```
              precision    recall  f1-score   support

      Center       0.16      0.38      0.23       832
       Donut       0.40      0.01      0.03       146
    Edge-Loc       0.45      0.41      0.43      2772
   Edge-Ring       0.56      0.44      0.49      1126
         Loc       0.29      0.32      0.31      1973
   Near-full       0.57      0.28      0.38        95
      Random       0.19      0.04      0.06       257
     Scratch       0.17      0.06      0.09       693

    accuracy                           0.34      7894
   macro avg       0.35      0.24      0.25      7894
weighted avg       0.36      0.34      0.34      7894

Diagonal elements of confusion matrix:
317, 2, 1129, 490, 636, 27, 10, 42,

array([[ 317,    0,  136,    6,  345,    0,    1,   27],
       [  28,    2,   10,    1,  103,    1,    1,    0],
       [ 685,    0, 1129,  203,  701,    4,    2,   48],
       [ 252,    0,  226,  490,  146,    0,    0,   12],
       [ 448,    1,  688,   81,  636,    0,    6,  113],
       [   6,    0,   23,    6,    1,   27,   32,    0],
       [  94,    1,   64,   25,   46,   15,   10,    2],
       [ 152,    1,  222,   62,  214,    0,    0,   42]],
```

4.3 Random Forest Classifier

- Training results:

```
              precision    recall  f1-score   support

      Center       1.00      1.00      1.00      3462
       Donut       1.00      1.00      1.00       409
    Edge-Loc       1.00      1.00      1.00      2417
   Edge-Ring       1.00      1.00      1.00      8554
         Loc       1.00      1.00      1.00      1620
   Near-full       1.00      1.00      1.00        54
      Random       1.00      1.00      1.00       609
     Scratch       1.00      1.00      1.00       500

    accuracy                           1.00     17625
   macro avg       1.00      1.00      1.00     17625
weighted avg       1.00      1.00      1.00     17625

Diagonal elements of confusion matrix:
3462, 409, 2417, 8554, 1620, 54, 609, 500,

array([[3462,    0,    0,    0,    0,    0,    0,    0],
       [   0,  409,    0,    0,    0,    0,    0,    0],
       [   0,    0, 2417,    0,    0,    0,    0,    0],
       [   0,    0,    0, 8554,    0,    0,    0,    0],
       [   0,    0,    0,    0, 1620,    0,    0,    0],
       [   0,    0,    0,    0,    0,   54,    0,    0],
       [   0,    0,    0,    0,    0,    0,  609,    0],
       [   0,    0,    0,    0,    0,    0,    0,  500]],
```

- Testing results:

```
              precision    recall  f1-score   support

      Center       0.31      0.21      0.25       832
       Donut       0.40      0.01      0.03       146
    Edge-Loc       0.41      0.56      0.47      2772
   Edge-Ring       0.50      0.38      0.43      1126
         Loc       0.27      0.34      0.30      1973
   Near-full       0.96      0.23      0.37        95
      Random       0.28      0.18      0.22       257
     Scratch       0.20      0.03      0.06       693

    accuracy                           0.37      7894
   macro avg       0.42      0.24      0.27      7894
weighted avg       0.36      0.37      0.35      7894

Diagonal elements of confusion matrix:
171, 2, 1545, 426, 665, 22, 46, 23,

array([[ 171,    0,  327,   56,  272,    0,    4,    2],
       [   8,    2,   24,    1,  108,    0,    3,    0],
       [ 138,    0, 1545,  180,  860,    0,   15,   34],
       [  35,    0,  384,  426,  210,    0,   25,   46],
       [ 119,    3, 1049,  115,  665,    0,   14,    8],
       [   2,    0,   14,    0,    1,   22,   56,    0],
       [  40,    0,  118,   19,   33,    1,   46,    0],
       [  36,    0,  309,   52,  272,    0,    1,   23]],
```

Analysis:

It is observed that the RFG classifier has accuracy 1 in the training results, thus we will not consider it in our final model, to avoid the possibility of overfitting.

Testing confusion matrix diagonals:

Logistic regression:

```
Diagonal elements of confusion matrix:
317, 4, 1050, 340, 588, 1, 30, 23,
```

K nearest neighbours:

```
Diagonal elements of confusion matrix:
317, 2, 1129, 490, 636, 27, 10, 42,
```

In KNN, except for indices 1,6, the correctly classified classes are more than logistic regression, and the testing accuracy is higher as well (even though the training accuracy of logistic regression is more!). Thus, the final best model for classification is KNN.

Using this model, an attempt was made to classify the [[‘none’]] type defects:

5. [['none']] type classification attempt:

- Firstly, a new dataframe of those values containing ‘none’ class was made and displayed.

```
df_with_defect_none = df_with_defect[(df_with_defect['failure_ordinal_enc']==8)]
df_with_defect_none.head()
```

	waferMap	dieSize	lotName	waferIndex	trianTestLabel	failureType	failure_ordinal_enc	waferMapDim
0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	1.0	[[Training]]	[[none]]	8	(45, 48)
1	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	2.0	[[Training]]	[[none]]	8	(45, 48)
2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	3.0	[[Training]]	[[none]]	8	(45, 48)
3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	4.0	[[Training]]	[[none]]	8	(45, 48)
4	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	5.0	[[Training]]	[[none]]	8	(45, 48)

- The features when then extracted from this dataframe and modified to fit the classifier (resizing and reshaping, as done before with training data)

```
x_none_unproc=df_with_defect_none.waferMap.reset_index()

# making wafer map dimensions (size) equal to fit in the classifier
x_none=[]
for i in range(x_none_unproc.shape[0]):
    img=x_none_unproc['waferMap'][i].copy()
    img.resize(220,90)
    img_new=img.reshape(-1)
    x_none.append(img_new)
```

```
y_none_pred=knn_clf.predict(x_none)
```

- The predicted values was entered into a new column in the above created dataframe.

```
df_with_defect_none['alternateDefect']=y_none_pred
df_with_defect_none
```

	waferMap	dieSize	lotName	waferIndex	trianTestLabel	failureType	failure_ordinal_enc	waferMapDim	alternateDefect
0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	1.0	[[Training]]	[[none]]	8	(45, 48)	Center
1	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	2.0	[[Training]]	[[none]]	8	(45, 48)	Center
2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	3.0	[[Training]]	[[none]]	8	(45, 48)	Center
3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	4.0	[[Training]]	[[none]]	8	(45, 48)	Center
4	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...	1683.0	lot1	5.0	[[Training]]	[[none]]	8	(45, 48)	Center
...	...	...	...	...	...	...	...	...	...
811438	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1,...	600.0	lot47542	9.0	[[Test]]	[[none]]	8	(26, 30)	Edge-Loc
811439	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,...	600.0	lot47542	10.0	[[Test]]	[[none]]	8	(26, 30)	Edge-Loc
811442	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1,...	600.0	lot47542	13.0	[[Test]]	[[none]]	8	(26, 30)	Edge-Loc
811445	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,...	600.0	lot47542	16.0	[[Test]]	[[none]]	8	(26, 30)	Edge-Loc
811449	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,...	600.0	lot47542	20.0	[[Test]]	[[none]]	8	(26, 30)	Edge-Loc

- The frequency of the predicted values was printed:

```
import collections

counter=collections.Counter(y_none_pred)
print(counter)
```

```
Counter({'Loc': 47007, 'Edge-Loc': 45210, 'Center': 33517,
'Edge-Ring': 11782, 'Scratch': 9914, 'Random': 1})
```

Analysis:

The last result shows that the ‘none’ defects aren’t really ‘random’ in nature, rather they have some localisation (high values of ‘loc’, ‘edge-loc’ and ‘center’). Also its observed that the ‘Donut’ and ‘Near-full’ categories are absent, ie the localisations do not spread over a large area. Thus, this model can be used to classify this category into the rest of these 8 ones (the number of ‘loc’, ‘edge-loc’ and ‘center’ defects will increase).

RESULT

It was seen that the K-Nearest neighbours is a suitable classifying model for this dataset (after processing of the feature and target values). It was further used to classify the ‘none’ class instances into the rest of 8 classes, and it was observed that they largely come under the ‘loc’, ‘edge-loc’ and ‘center’ categories, ie the defects are localised to a small area, and are not random as expected.

CONCLUSION AND INSIGHTS

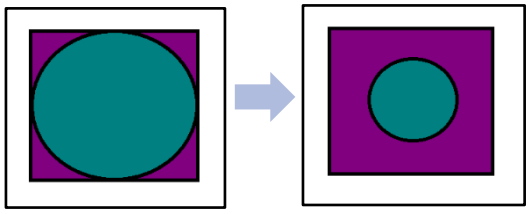
In this report, the processes of generating Si wafers and ICs, and their defect detection was explored. Defects mainly arise due to imperfections in manufacturing, and it’s important to classify these defects in order to rectify the fab process, else these ICs may fail and malfunction in the real-life usage.

Then the approach to explore the dataset and the attempt to choose a suitable model was examined stepwise, and finally the KNN model was settled on, due to its better testing accuracy (RFG was rejected due to possibility of overfitting). Then this KNN model was tried on the ‘none’ class, and it was observed that the defects are quite localised (over a small area) and can be classified into certain categories belonging to original 8 categories.

Certain insights:

- While the attempt to classify the data using normal machine learning classifier models was not totally a failure, the general accuracy of these models is quite low. Thus, if a general model classification was to be chosen in real world, Neural Networks (as tried and tested by people in Kaggle as well as research papers) is the best way to go about it.

- Data preprocessing in this is attempted by resizing the images (to a bigger image than they actually are). This affects the actual wafer image and it becomes of the form:



- This affects the classifier results as it compares it pixel to pixel. Therefore, a better way to classify defects is needed, or a better way to process the data. On the plus side, this method does give a wider data range for the model to process.

- This model can be used to eliminate the [[‘none’]] category (such defects may be too hard to classify manually, thus this model is a good alternative).

DATASET

Downloaded from Kaggle:
https://www.kaggle.com/qingyi/wm811k-wafer-map

Original source:
from [MIR lab] (http://mirlab.org/dataSet/public/) (under MIR-WM811K section)

Code file: defect_classification_19116080.ipynb

REFERENCES

- Wafer manufacturing:
  - https://www.sas-globalwafers.co.jp/eng/products/wafer/process.html
- IC manufacturing:
  - https://www.hitachi-hightech.com/global/products/device/semiconductor/process.html
- Wafer defect detection:
  - Material from: https://www.hitachi-hightech.com/global/products/device/semiconductor/inspection.html
  - Pictures from: https://rsipvision.com/wafer-macro-defects-detection-classification/
- Code: https://www.kaggle.com/ashishpatel26/wm-811k-wafermap
  - Parts of Data analysis (section 2), and making of ‘waferMapDim’ column