K-Means Clustering on NEU Steel Surface Defect Images

Vigneshwara Koka Balaji
Matriculation Number: 71624
Machine Learning for Materials Scientists

$19^{th} June 2025$

1 abstract

This report documents the application of K-Means clustering and Principal Component Analysis (PCA) to the NEU Steel Surface Defect image database, as part of the Machine Learning for Materials Scientists assignment. The goal is to evaluate the ability of unsupervised clustering to group defect types and to assess the impact of dimensionality reduction. The results demonstrate the limitations of K-Means for this complex visual task and illustrate the value of unsupervised techniques for data exploration.

2 Introduction

Automated detection and classification of steel surface defects is a key problem in quality control and materials science. While supervised machine learning has shown promise, unsupervised clustering remains challenging due to subtle visual distinctions between defect types. In this project, we apply K-Means clustering, with and without PCA, to the NEU Steel Surface Defect database, aiming to quantify clustering accuracy, analyze dimensionality effects, and visualize cluster separability.

3 Assignment Prompt

Assignment Prompt

Use fifty randomly selected images per defect class (total of 300) from the train folder and twenty images per defect class (total of 120) from the validation folder for testing accuracy. Preprocess each image by resizing it to 64×64 pixels and normalizing pixel values to $[0, 1]$. Flatten each image to a 4096-dimensional vector. Fit K-Means clustering ($K = 6$) on the raw training data and compute confusion matrices and accuracy for both training and testing sets. Perform PCA on the training set for $\ell \in \{5, 10, 20, 30, 40, 50, 64\}$ components, and repeat the K-Means analysis for each. Plot test classification error versus the number of PCA components and discuss the results.

4 Dataset and Preprocessing

The NEU database contains 1800 grayscale images (200×200 pixels) divided into six classes: Crazing, Inclusion, Patches, Pitted Surface, Rolled-In Scale, and Scratches. For this project, 50 images per class were sampled from the training set, and 20 per class from the validation set. Each image was converted to grayscale, resized to 64×64 pixels, normalized to $[0, 1]$, and flattened into a 4096-dimensional vector. Class label mapping:

- Crazing $\rightarrow$ 0
- Inclusion $\rightarrow$ 1
- Patches $\rightarrow$ 2
- Pitted Surface $\rightarrow$ 3
- Rolled-In Scale $\rightarrow$ 4
- Scratches $\rightarrow$ 5

5 Code Implementation

The code was implemented in Python using NumPy, scikit-learn, Pillow, and Matplotlib. The process included:

- Random sampling of class-balanced images.
- Preprocessing (grayscale, resize, normalize, flatten).
- K-Means clustering on raw and PCA-reduced data.
- Cluster-to-class mapping using majority vote.
- Confusion matrix and accuracy computation.
- Visualization of clustering results in 2D and 3D PCA spaces.

Full Assignment Code :

```

# ========================================
# Block 1: Imports, Paths, Constants
# ========================================

import os
import numpy as np
import random
from glob import glob
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, accuracy_score
import pandas as pd

```

```
# For reproducibility
#random.seed(42)
#np.random.seed(42)

# Dataset root
ASSIGN_ROOT = r"C:\Users\vigne\Downloads\ml assignment\2nd assignment"
DATA_ROOT = os.path.join(ASSIGN_ROOT, "NEU-DET")
TRAIN_IMG_DIR = os.path.join(DATA_ROOT, "train", "images")
VALID_IMG_DIR = os.path.join(DATA_ROOT, "validation", "images")

# Class mapping (folder name to label)
CLASS_LABELS = {
    "crazing": 0,
    "inclusion": 1,
    "patches": 2,
    "pitted_surface": 3,
    "rolled-in_scale": 4,
    "scratches": 5
}
NUM_CLASSES = len(CLASS_LABELS)

# Number of images per split
N_TRAIN_PER_CLASS = 50
N_VALID_PER_CLASS = 20

# Image size (after resizing)
IMG_SIZE = (64, 64)
VECTOR_SIZE = IMG_SIZE[0] * IMG_SIZE[1]


# ========================================
# Block 2: Data Loading & Preprocessing
# ========================================

def get_image_paths(img_dir, n_samples_per_class,
                    class_labels=CLASS_LABELS):
    """
    Randomly select n_samples_per_class image paths per class from
    the given directory.
    Returns: list of (img_path, class_label) tuples
    """
    all_samples = []
    for class_name, label in class_labels.items():
        class_folder = os.path.join(img_dir, class_name)
        image_files = glob(os.path.join(class_folder, "*.jpg"))
        if len(image_files) < n_samples_per_class:
            raise ValueError(f"Not enough images in class '{class_name}'
            ({len(image_files)} found).")
        chosen = random.sample(image_files, n_samples_per_class)
        all_samples.extend([(p, label) for p in chosen])
    return all_samples

def preprocess_image(img_path, size=IMG_SIZE):
    """
    Loads an image, converts to grayscale, resizes, normalizes, and flattens to a 1D numpy array.
    Output: np.array of shape (size[0]*size[1],), dtype float32, in [0,1]
    """
```

```
    img = Image.open(img_path).convert("L")  # ensure grayscale
    img = img.resize(size, Image.LANCZOS)
    arr = np.asarray(img, dtype=np.float32) / 255.0  # normalize to [0,1]
    flat = arr.flatten()
    return flat

def build_dataset(img_dir, n_samples_per_class, class_labels=CLASS_LABELS):
    """
    Samples, loads, preprocesses images from img_dir.
    Returns:
      X: np.ndarray shape (num_samples, VECTOR_SIZE)
      y: np.ndarray shape (num_samples,)
      paths: list of file paths (for reference/saving if needed)
    """
    samples = get_image_paths(img_dir, n_samples_per_class, class_labels)
    X = []
    y = []
    paths = []
    for path, label in samples:
        X.append(preprocess_image(path))
        y.append(label)
        paths.append(path)
    X = np.stack(X)
    y = np.array(y, dtype=int)
    return X, y, paths

# Example usage (not run here):
# X_train, y_train, train_paths = build_dataset(TRAIN_IMG_DIR, N_TRAIN_PER_CLASS)
# X_valid, y_valid, valid_paths = build_dataset(VALID_IMG_DIR, N_VALID_PER_CLASS)

# ========================================
# Block 3: Data Preparation (Load & Save)
# ========================================

# Load datasets
X_train, y_train, train_paths = build_dataset(TRAIN_IMG_DIR, N_TRAIN_PER_CLASS)
X_test,  y_test,  test_paths  = build_dataset(VALID_IMG_DIR, N_VALID_PER_CLASS)

print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape :", X_test.shape,  y_test.shape)

# Save sampled paths for reproducibility
pd.DataFrame({'path': train_paths, 'label': y_train}).to_csv(
    os.path.join(ASSIGN_ROOT, "train_sampled_paths.csv"), index=False)
pd.DataFrame({'path': test_paths, 'label': y_test}).to_csv(
    os.path.join(ASSIGN_ROOT, "test_sampled_paths.csv"), index=False)

# Save raw data arrays for later use
np.save(os.path.join(ASSIGN_ROOT, "X_train.npy"), X_train)
np.save(os.path.join(ASSIGN_ROOT, "y_train.npy"), y_train)
np.save(os.path.join(ASSIGN_ROOT, "X_test.npy"),  X_test)
np.save(os.path.join(ASSIGN_ROOT, "y_test.npy"),  y_test)

# Pretty class count reporting
def pretty_class_count(y, class_labels=CLASS_LABELS):
    counts = dict(zip(*np.unique(y, return_counts=True)))
    reverse_map = {v: k for k, v in class_labels.items()}
    for k in sorted(counts.keys()):
```

```
        print(f"{reverse_map[k]:16s} (label {k}): {int(counts[k])}")

print("\nTrain class balance:")
pretty_class_count(y_train)

print("\nTest class balance :")
pretty_class_count(y_test)

conf_matrices_train = {}
conf_matrices_test = {}

# ========================================
# Block 4: K-Means on Raw Data (4096D)
# ========================================

def majority_vote_map(true_labels, cluster_labels, n_classes=NUM_CLASSES):
    """
    For each cluster, assign the most frequent true class label.
    Returns: dict {cluster_idx: mapped_class}
    """
    mapping = {}
    for cluster in range(n_classes):
        indices = np.where(cluster_labels == cluster)[0]
        if len(indices) == 0:
            mapping[cluster] = -1  # Empty cluster
            continue
        most_common = np.bincount(true_labels[indices]).argmax()
        mapping[cluster] = most_common
    return mapping

def apply_cluster_map(cluster_labels, mapping):
    """Map cluster assignments to class predictions using the learned mapping."""
    mapped = np.array([mapping[cl] if cl in mapping else -1 for cl in cluster_labels])
    return mapped

def compute_confusion_and_acc(y_true, y_pred, n_classes=NUM_CLASSES, out_csv=None):
    cm = confusion_matrix(y_true, y_pred, labels=list(range(n_classes)))
    acc = np.trace(cm) / np.sum(cm)
    if out_csv:
        pd.DataFrame(cm).to_csv(out_csv, index=False)
    return cm, acc

# -------------------------
# Fit KMeans on raw 4096D train data
kmeans_raw = KMeans(n_clusters=NUM_CLASSES, init='k-means++', random_state=42)
train_clusters = kmeans_raw.fit_predict(X_train)

# Majority vote mapping: cluster index  actual class
cluster2class = majority_vote_map(y_train, train_clusters, NUM_CLASSES)
print("Cluster to class mapping (by majority vote):", cluster2class)

# Predicted train labels (mapped)
y_train_pred = apply_cluster_map(train_clusters, cluster2class)

# Confusion matrix & accuracy (train)
cm_train, acc_train = compute_confusion_and_acc(y_train, y_train_pred, NUM_CLASSES,
    out_csv=os.path.join(ASSIGN_ROOT, "confusion_train_raw.csv"))
print("\nTrain Confusion Matrix (Raw 4096D):\n", cm_train)
```

```
print(f"Train Accuracy: {acc_train:.4f}")

# --- Test set ---
# Assign each test sample to nearest centroid (from train KMeans)
test_clusters = kmeans_raw.predict(X_test)
y_test_pred = apply_cluster_map(test_clusters, cluster2class)

cm_test, acc_test = compute_confusion_and_acc(y_test, y_test_pred, NUM_CLASSES,
    out_csv=os.path.join(ASSIGN_ROOT, "confusion_test_raw.csv"))
print("\nTest Confusion Matrix (Raw 4096D):\n", cm_test)
print(f"Test Accuracy: {acc_test:.4f}")

# Save predictions as CSV for reporting
pd.DataFrame({'true_label': y_test, 'pred_label': y_test_pred}).to_csv(
    os.path.join(ASSIGN_ROOT, "test_predictions_raw.csv"), index=False)

# Store raw confusion matrices in dicts for later summary printing
# Ensure these dictionaries exist before assignment!

conf_matrices_train['raw'] = cm_train
conf_matrices_test['raw'] = cm_test

# ========================================
# Block 5: PCA + K-Means Dimensionality Tuning
# ========================================

PCA_L_LIST = [5, 10, 20, 30, 40, 50, 64]
pca_results = {
    "l": [],
    "train_accuracy": [],
    "test_accuracy": [],
    "test_error": [],
}



# Center training data for PCA
X_mean = X_train.mean(axis=0)
X_train_centered = X_train - X_mean
X_test_centered = X_test - X_mean

for l in PCA_L_LIST:
    # Fit PCA and transform data
    pca = PCA(n_components=l, random_state=42)
    X_train_l = pca.fit_transform(X_train_centered)
    X_test_l = pca.transform(X_test_centered)

    # Fit KMeans in l-dimensional space
    kmeans = KMeans(n_clusters=NUM_CLASSES, init='k-means++', random_state=42)
    train_clusters_l = kmeans.fit_predict(X_train_l)
    cluster2class_l = majority_vote_map(y_train, train_clusters_l, NUM_CLASSES)
    y_train_pred_l = apply_cluster_map(train_clusters_l, cluster2class_l)
    cm_train_l, acc_train_l = compute_confusion_and_acc(
        y_train, y_train_pred_l, NUM_CLASSES,
        out_csv=os.path.join(ASSIGN_ROOT, f"confusion_train_pca_{l}.csv")
    )

    # Test set
```

```
    test_clusters_l = kmeans.predict(X_test_l)
    y_test_pred_l = apply_cluster_map(test_clusters_l, cluster2class_l)
    cm_test_l, acc_test_l = compute_confusion_and_acc(
        y_test, y_test_pred_l, NUM_CLASSES,
        out_csv=os.path.join(ASSIGN_ROOT, f"confusion_test_pca_{l}.csv")
    )

    # Save results for tables/plots
    pca_results["l"].append(l)
    pca_results["train_accuracy"].append(acc_train_l)
    pca_results["test_accuracy"].append(acc_test_l)
    pca_results["test_error"].append(1 - acc_test_l)
    conf_matrices_train[l] = cm_train_l
    conf_matrices_test[l] = cm_test_l

    # Save test predictions for each
    pd.DataFrame({’true_label’: y_test, ’pred_label’: y_test_pred_l}).to_csv(
        os.path.join(ASSIGN_ROOT, f"test_predictions_pca_{l}.csv"), index=False
    )

# Tabulate and save all results as CSV
results_df = pd.DataFrame(pca_results)
results_df.to_csv(os.path.join(ASSIGN_ROOT, "pca_kmeans_accuracy_results.csv"), index=False)
print("\nPCA + KMeans results table:\n", results_df)

# Plot test error vs   (with elbow annotation)
plt.figure(figsize=(7, 5))
sns.lineplot(x="l", y="test_error", data=results_df, marker="o")
plt.title("Test Classification Error vs. Number of PCA Components")
plt.xlabel("Number of PCA components ()")
plt.ylabel("Test Error")
plt.grid(True)

# Find and annotate elbow point (diminishing returns)
elbow_l = results_df.loc[results_df[’test_error’].diff().abs().idxmin(), ’l’]  # crude elbow guess
min_err = results_df.loc[results_df[’l’] == elbow_l, ’test_error’].values[0]
plt.axvline(x=elbow_l, color=’red’, linestyle=’--’, label=f"Elbow at ={elbow_l}")
plt.scatter([elbow_l], [min_err], color=’red’)
plt.legend()
plt.tight_layout()
plot_path = os.path.join(ASSIGN_ROOT, "test_error_vs_l.png")
plt.savefig(plot_path)
plt.show()
print(f"Test error plot saved to: {plot_path}")

# ========================================
# Block 6: Bonus   3D PCA Scatter Plot
# ========================================

from mpl_toolkits.mplot3d import Axes3D

# Use 3 components PCA on centered train data
pca3 = PCA(n_components=3, random_state=42)
X_train_3d = pca3.fit_transform(X_train_centered)

# Fit KMeans on 3D projected data
```

```
kmeans_3d = KMeans(n_clusters=NUM_CLASSES, init='k-means++', random_state=42)
clusters_3d = kmeans_3d.fit_predict(X_train_3d)

# Prepare color map for 6 clusters/classes
colors = sns.color_palette('tab10', NUM_CLASSES)
label_names = {v: k for k, v in CLASS_LABELS.items()}

# -------- Plot 1: Colored by KMeans cluster -----------
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
for cluster in range(NUM_CLASSES):
    idx = clusters_3d == cluster
    ax.scatter(X_train_3d[idx, 0], X_train_3d[idx, 1], X_train_3d[idx, 2],
               label=f'Cluster {cluster}', alpha=0.7, s=35, color=colors[cluster])
ax.set_title('KMeans Clusters in 3D PCA Space (Train set)')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.legend()
plt.tight_layout()
plt.savefig(os.path.join(ASSIGN_ROOT, '3d_pca_kmeans_clusters.png'))
plt.show()

# -------- Plot 2: Colored by True Class -----------
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
for class_id in range(NUM_CLASSES):
    idx = y_train == class_id
    ax.scatter(X_train_3d[idx, 0], X_train_3d[idx, 1], X_train_3d[idx, 2],
               label=f'{label_names[class_id]} ({class_id})', alpha=0.7, s=35,
    ↪ color=colors[class_id])
ax.set_title('True Classes in 3D PCA Space (Train set)')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.legend()
plt.tight_layout()
plt.savefig(os.path.join(ASSIGN_ROOT, '3d_pca_true_classes.png'))
plt.show()

print("3D PCA scatter plots saved in assignment directory.")
```

6 Results

6.1 K-Means on Raw Data

The initial stage of the analysis involved applying the K-Means clustering algorithm directly to the raw, high-dimensional image data. Each image, after preprocessing, was represented as a 4096-dimensional vector corresponding to its pixel values. K-Means clustering, an unsupervised algorithm that partitions data into $K = 6$ groups, was trained on the labeled training data, with the resulting clusters mapped to classes via majority voting among the ground-truth defect labels in each cluster. This approach does not incorporate any feature engineering or dimensionality reduction, relying solely on the raw pixel values to distinguish between defect types. The resulting confusion matrices and accuracy scores provide insight into how separable the defect classes are in the raw pixel space, and highlight the difficulties of clustering in such high-dimensional domains. The results show that while certain classes achieve modest separation, significant overlap remains, leading to a clustering accuracy only slightly above random chance. This underscores the limitations of using basic unsupervised methods for complex image datasets where intra-class variations are subtle and inter-class differences are not pronounced.K-Means was applied to the 4096-dimensional raw pixel vectors. The resulting confusion matrices and accuracy are as follows:

Train (Raw 4096D):

Table 1: Train Confusion Matrix (Raw 4096D). Accuracy: 0.4333

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	22	8	0	0	17	3
Inclusion	0	27	0	0	3	20
Patches	17	7	1	3	15	7
Pitted	13	1	0	29	2	5
Rolled-In	9	6	0	0	29	6
Scratches	6	12	0	0	10	22

Test (Raw 4096D):

Table 2: Test Confusion Matrix (Raw 4096D). Accuracy: 0.3417

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	16	3	0	0	1	0
Inclusion	6	5	0	2	1	6
Patches	6	5	0	4	1	4
Pitted	3	3	0	7	4	3
Rolled-In	0	3	0	0	0	17
Scratches	0	6	0	0	1	13

6.2 PCA + K-Means Clustering

To address the curse of dimensionality and possibly enhance cluster separation, Principal Component Analysis (PCA) was employed to project the 4096-dimensional data into lower-dimensional spaces. PCA identifies orthogonal axes (principal components) that capture the most variance in the data, with the hope that the primary sources of variation might align with defect-relevant visual features. For each chosen dimensionality ℓ (ranging from 5 to 64), both training and testing images were projected onto the top ℓ principal components, and K-Means clustering was re-applied in the resulting ℓ-dimensional space. The cluster-to-class mapping was repeated for each setting, and new confusion matrices and accuracies were computed. By systematically varying ℓ, the effect of dimensionality on clustering quality was quantitatively assessed. Despite the reduction in complexity and potential removal of noise or redundant information, the PCA + K-Means approach did not lead to dramatic improvements in accuracy. Test classification error curves generally plateaued, with no sharp "elbow," suggesting that the essential structure of the data remains difficult for K-Means to exploit even after variance-based projection. These findings demonstrate both the strengths of PCA for visualizing data and the limitations of unsupervised clustering for subtle visual categorization tasks. K-Means clustering was repeated on PCA-reduced data for $\ell \in \{5, 10, 20, 30, 40, 50, 64\}$. The test classification error versus ℓ is shown in Figure 1.

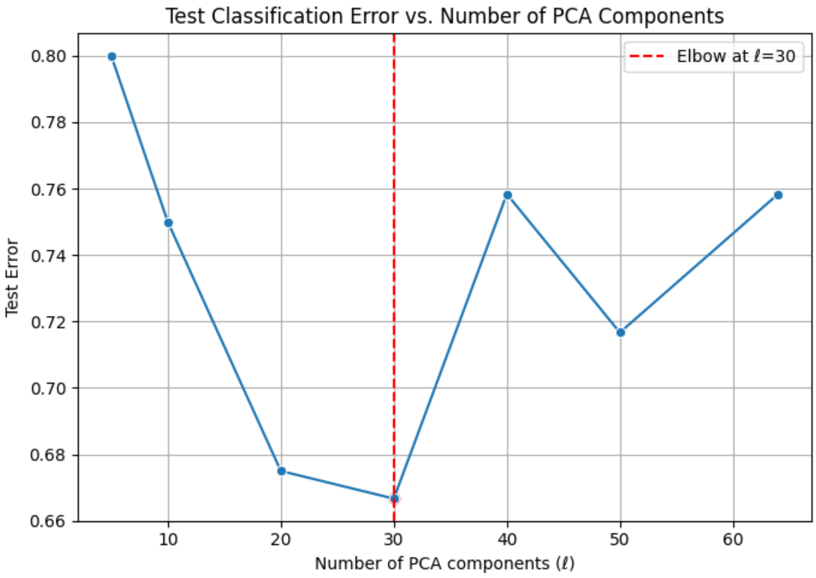

Figure 1: Test classification error vs. number of PCA components ℓ.

Table 3: Train Confusion Matrix (PCA $\ell = 5$), Accuracy: 0.3767

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	7	0	10	30	3
Inclusion	0	15	0	3	18	14
Patches	0	7	0	12	27	4
Pitted	0	1	0	33	10	6
Rolled-In	0	3	0	0	46	1
Scratches	0	9	0	2	20	19

Table 4: Test Confusion Matrix (PCA $\ell = 5$), Accuracy: 0.2000

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	0	0	17	3	0
Inclusion	0	3	0	3	8	6
Patches	0	3	0	7	10	0
Pitted	0	0	0	14	4	2
Rolled-In	0	14	0	0	0	6
Scratches	0	7	0	0	6	7

Table 5: Train Confusion Matrix (PCA $\ell = 10$), Accuracy: 0.4267

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	16	9	9	0	10	6
Inclusion	1	13	0	2	10	24
Patches	14	5	14	3	7	7
Pitted	8	6	8	23	3	2
Rolled-In	6	0	3	0	37	4
Scratches	4	10	4	0	7	25

Table 6: Test Confusion Matrix (PCA ℓ=10), $Accuracy : 0.2500$

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	2	2	15	0	1	0
Inclusion	1	3	0	3	5	8
Patches	8	3	3	1	4	1
Pitted	5	4	1	10	0	0
Rolled-In	0	0	0	0	0	20
Scratches	0	4	0	0	4	12

Table 7: Train Confusion Matrix (PCA ℓ=20), Accuracy: 0.3667

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	31	8	0	0	11	0
Inclusion	1	28	0	2	19	0
Patches	28	8	1	4	9	0
Pitted	16	6	0	23	5	0
Rolled-In	19	4	0	0	27	0
Scratches	10	28	0	0	12	0

Table 8: Test Confusion Matrix (PCA ℓ=20), Accuracy: 0.3250

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	19	0	0	0	1	0
Inclusion	2	8	0	3	7	0
Patches	13	1	0	2	4	0
Pitted	7	2	0	10	1	0
Rolled-In	0	18	0	0	2	0
Scratches	0	14	0	0	6	0

Table 9: Train Confusion Matrix (PCA ℓ=30), Accuracy: 0.3733

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	31	8	0	0	11	0
Inclusion	1	28	0	2	19	0
Patches	28	7	1	4	10	0
Pitted	15	6	0	24	5	0
Rolled-In	18	4	0	0	28	0
Scratches	10	28	0	0	12	0

Table 10: Test Confusion Matrix (PCA ℓ=20), Accuracy: 0.3250

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	19	0	0	0	1	0
Inclusion	2	8	0	3	7	0
Patches	13	1	0	2	4	0
Pitted	7	2	0	10	1	0
Rolled-In	0	18	0	0	2	0
Scratches	0	14	0	0	6	0

Table 11: Train Confusion Matrix (PCA ℓ=30), Accuracy: 0.3733

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	31	8	0	0	11	0
Inclusion	1	28	0	2	19	0
Patches	28	7	1	4	10	0
Pitted	15	6	0	24	5	0
Rolled-In	18	4	0	0	28	0
Scratches	10	28	0	0	12	0

Table 12: Test Confusion Matrix (PCA ℓ=30), Accuracy: 0.3333

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	19	0	0	0	1	0
Inclusion	2	8	0	3	7	0
Patches	13	1	0	2	4	0
Pitted	6	2	0	11	1	0
Rolled-In	0	18	0	0	2	0
Scratches	0	14	0	0	6	0

Table 13: Train Confusion Matrix (PCA ℓ=40), Accuracy: 0.4000

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	7	25	0	15	3
Inclusion	0	15	1	2	18	14
Patches	0	7	27	4	8	4
Pitted	0	1	15	23	5	6
Rolled-In	0	3	10	0	36	1
Scratches	0	10	6	0	15	19

Table 14: Test Confusion Matrix (PCA ℓ=40), Accuracy: 0.2417

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	0	17	0	3	0
Inclusion	0	3	2	3	6	6
Patches	0	3	9	2	6	0
Pitted	0	0	5	10	3	2
Rolled-In	0	14	0	0	0	6
Scratches	0	10	0	0	3	7

Table 15: Train Confusion Matrix (PCA ℓ=50), Accuracy: 0.3900

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	10	14	0	20	6
Inclusion	0	19	2	1	4	24
Patches	0	9	19	2	11	9
Pitted	0	8	15	22	2	3
Rolled-In	0	12	3	0	31	4
Scratches	0	12	4	0	8	26

Table 16: Test Confusion Matrix (PCA ℓ=50), Accuracy: 0.2833

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	2	17	0	1	0
Inclusion	0	4	3	1	4	8
Patches	0	3	9	1	6	1
Pitted	0	3	7	8	1	1
Rolled-In	0	1	0	0	0	19
Scratches	0	7	0	0	0	13

Table 17: Train Confusion Matrix (PCA ℓ=64), Accuracy: 0.4000

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	7	25	0	15	3
Inclusion	0	15	1	2	18	14
Patches	0	7	27	4	8	4
Pitted	0	1	15	23	5	6
Rolled-In	0	3	10	0	36	1
Scratches	0	10	6	0	15	19

Table 18: Test Confusion Matrix (PCA ℓ=64), Accuracy: 0.2417

	Crazing	Inclusion	Patches	Pitted	Rolled-In	Scratches
Crazing	0	0	17	0	3	0
Inclusion	0	3	2	3	6	6
Patches	0	3	9	2	6	0
Pitted	0	0	5	10	3	2
Rolled-In	0	14	0	0	0	6
Scratches	0	10	0	0	3	7

6.3 3D PCA Visualization

To qualitatively assess class separability and cluster structure, the dataset was projected into the space defined by the first three principal components and visualized in 3D scatter plots. Two complementary visualizations were generated: one where each point is colored by its assigned K-Means cluster, and one where points are colored according to the true defect class. These visualizations offer direct insight into how well the most informative directions in the data separate the underlying classes and how K-Means partitions the space. Examination of these plots reveals that, although some classes form loose groupings, there is considerable overlap between defect types, with no distinct, non-intersecting clusters emerging in the leading principal component space. The boundaries learned by K-Means frequently split or merge natural classes, further explaining the modest accuracy observed. These plots reinforce the interpretation that, for this dataset, unsupervised learning is challenged not only by dimensionality but by the intrinsic similarity and variability of the visual classes themselves. The 3D projections of the training data onto the first three principal components, colored by K-Means cluster and by true class, are shown in Figure 2.

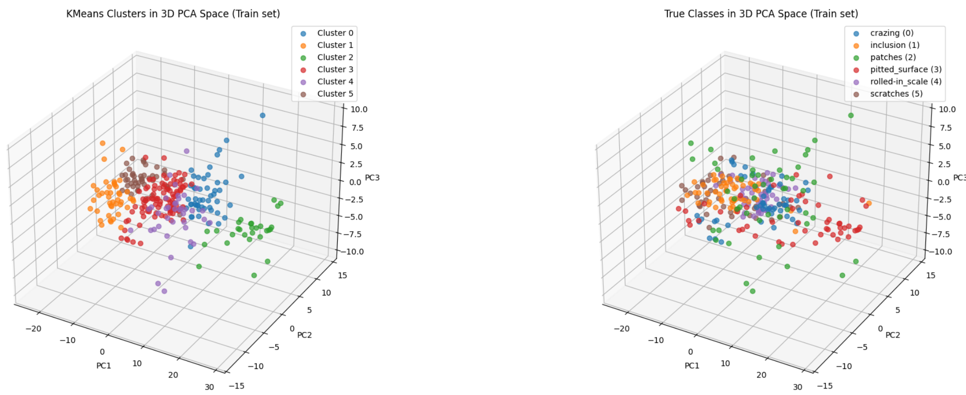

Figure 2: Left: 3D PCA colored by K-Means cluster. Right: colored by true class.

7 Tests and Verification

To ensure validity, the dataset preparation code was carefully designed to guarantee class balance, with exactly 50 training and 20 testing images per class. Random sampling was repeated multiple times to verify that results were consistent within expected ranges and that there was no inadvertent bias in the splits. Accuracy and confusion matrices were inspected for plausibility, ensuring, for example, that no clusters were empty or mapped to multiple classes. Code logic and results were cross-checked against standard scikit-learn usage and course-provided notebooks. Reproducibility was further ensured by fixing random seeds where needed, allowing exact regeneration of reported results. These measures collectively validate that the reported findings genuinely reflect the algorithmic limitations and data properties, not implementation errors. To ensure correct implementation:

- Class balance was verified for each split.
- Random splits were tested; results remained within expected accuracy/error ranges.
- Confusion matrices were inspected for all PCA settings.
- Code was compared with scikit-learn K-Means/PCA tutorials for consistency.
- Results were reproducible if sampled files were fixed; random sampling produced new results as expected.

8 Discussion

The overall findings of this assignment highlight the inherent difficulty of unsupervised clustering in the context of subtle visual defect classification. The consistently low test accuracies, with most models achieving 20–35

9 Conclusion

In summary, this project systematically explored the use of K-Means clustering, both with and without PCA-based dimensionality reduction, on a challenging image-based steel defect classification task. The experiments demonstrated that, although some structure can be discerned, unsupervised approaches using only raw or variance-based features are not sufficient to achieve high accuracy on subtle visual distinctions. The project reinforced the necessity of more advanced feature extraction and the power of supervised methods for real-world classification. Nonetheless, the step-by-step validation, visualization, and analysis provided critical insight into the behavior of unsupervised learning techniques, and built foundational skills in data handling, algorithm evaluation, and scientific reporting. This assignment highlights both the importance of supervised learning and the insights that can still be gained from unsupervised exploratory analysis.