MEEG 54403 - Machine Learning for Mechanical Engineers

Assignment 3 Tutorial

Mohammad Kokash
University of Arkansas
Department of Mechanical Engineering

Tutorial III: Dimensionality Reduction and Clustering

Acknowledgment

1. This tutorial was prepared by Mohammad Kokash (`mkokash@uark.edu`) at the University of Arkansas.
2. The dataset and algorithms used in this tutorial are from C. Dunlap, H. Pandey, and H. Hu, “Supervised and Unsupervised Learning Models for Detection of Critical Heat Flux During Pool Boiling,” in Proceedings of the ASME 2022 Heat Transfer Summer Conference, HT2022-85582. [Link]
3. This tutorial was implemented in MEEG tech elective "Machine Learning for Mechanical Engineers." [Link]

1 Introduction and Motivation

In many engineering and scientific domains, boiling is a key heat transfer mechanism. It efficiently removes large amounts of heat with minimal temperature increase. For instance, high-power electronics, nuclear reactors, and aerospace systems often rely on boiling for thermal management. One critical parameter that engineers must carefully monitor is the *Critical Heat Flux* (CHF). When the boiling process surpasses the CHF, the heater surface temperature can suddenly increase, potentially leading to equipment damage or failure.

Traditionally, CHF detection relies on temperature and pressure sensors, as well as expert judgment. However, in recent years, **image-based analysis** of boiling surfaces has emerged as a promising complementary technique. By examining patterns of bubbles and vapor formation in the images, we may distinguish between stable, *pre-CHF* conditions and dangerous, *post-CHF* conditions. Such an approach could enhance early warning systems and improve the safety and efficiency of thermal management processes.

1.1 Why Clustering and Dimensionality Reduction?

The core idea of this assignment is to explore **unsupervised learning** methods, specifically **clustering** and **dimensionality reduction**, applied to boiling images. Here is why:

- **Clustering:** We want to group similar images together without using their labels. If the images naturally form clusters that align with pre-CHF and post-CHF labels, it suggests that the images contain distinct visual signatures of each regime. Clustering helps us discover these underlying patterns without prior knowledge.
- **Dimensionality Reduction:** Each image is essentially a high-dimensional data point (thousands of pixels). Working directly in such a high-dimensional space is challenging and can obscure meaningful patterns. Techniques like *PCA*, *t-SNE*, and *UMAP* allow us to represent images in a lower-dimensional space that is easier to visualize and cluster. - **PCA** (Principal Component Analysis) is a linear method that finds directions of maximum variance. - **t-SNE** (t-Distributed Stochastic Neighbor Embedding) is a nonlinear technique often used to create insightful 2D visualizations. - **UMAP** (Uniform Manifold Approximation and Projection) is another nonlinear approach that preserves local and some global data structure efficiently.

By combining these approaches, we can reduce the complexity of the data and then group similar images together, potentially revealing clear differences between pre-CHF and post-CHF conditions.

1.2 What Will be Learned?

In this tutorial, we will walk through the entire process, step-by-step:

1. **Data Preparation:** How to load and preprocess the boiling images.
2. **Feature Extraction:** Two methods will be introduced:
    - **Flatten:** Convert images directly into 1D vectors of pixel intensities.
    - **Conv_Flatten:** Use a pre-trained convolutional neural network (CNN) to extract features before flattening, leveraging learned representations from a large image dataset.
3. **Dimensionality Reduction:** Apply PCA, t-SNE, and UMAP to compress the feature vectors into fewer dimensions.
4. **Clustering:** Use K-Means to form two clusters. We choose two clusters because we have two regimes (pre-CHF and post-CHF), but we do not tell the algorithm the labels.
5. **Evaluation:** Assess the quality of the resulting clusters using:
    - Internal metrics (Silhouette, Davies-Bouldin) that measure cluster quality without labels.
    - External metrics (ARI, Purity) that compare clusters to the known labels.
6. **Visualization:** Generate scatter plots of t-SNE/UMAP embeddings, confusion matrices, and silhouette plots to gain intuitive insight into the clustering results.

1.3 Who Is This Tutorial For?

This tutorial is designed for students who are new to unsupervised learning techniques and want to understand the entire pipeline of handling image data, extracting features, reducing dimensionality, and clustering. A basic understanding of Python, NumPy, and scikit-learn will be helpful. While some familiarity with CNNs is beneficial, we will explain the Conv_Flatten step conceptually so you can follow along even if you are not a deep learning expert.

1.4 What You Will Need

Before starting, ensure that you have:

- A working Python environment (e.g., Anaconda).
- The necessary libraries: `numpy`, `matplotlib`, `pillow` (PIL), `tensorflow/keras` (for Conv_Flatten), `scikit-learn`, and `umap-learn`.
- A basic LaTeX environment if you wish to produce the final report in PDF form.

With these preparations, let's move on to setting up your environment and prerequisites, ensuring you have all the tools you need before diving into the data.

2 Prerequisites

Before diving into the technical steps, ensure you have the necessary background knowledge and computational tools. This will help you follow along smoothly and understand the rationale behind each method.

2.1 Technical Background

While this tutorial aims to be accessible, having some foundational knowledge will enhance your learning experience:

- **Python and NumPy:** You should be comfortable writing basic Python code, handling arrays, and performing simple computations with `numpy`.

- **Basic Machine Learning Concepts:** Understand the difference between supervised and unsupervised learning, and have a high-level idea of what clustering is. Familiarity with terms like “features” and “dimensionality” will help.
- **Scikit-learn:** Experience with `scikit-learn` is useful since we will use it for PCA, K-Means, and various metrics. Even if you are new, we will show code snippets and explain them step-by-step.
- **Neural Networks (Optional):** A basic idea of what a CNN is can be helpful for the Conv_Flatten step. However, we will treat it as a feature extractor and not dive deeply into its architecture.

2.2 Software and Libraries

Make sure you have the following tools installed:

- **Python 3.x**: A recent version of Python 3.
- **NumPy, Matplotlib, Pillow**: For numerical operations, plotting, and image handling.
- **TensorFlow/Keras**: For the Conv_Flatten approach using a pre-trained VGG16 model. You can install TensorFlow via `pip install tensorflow`.
- **scikit-learn**: For PCA, K-Means, and metrics. Install via `pip install scikit-learn`.
- **umap-learn**: For UMAP dimensionality reduction. Install via `pip install umap-learn`.

2.3 Data Access

Ensure you have access to the dataset of boiling images:

- Organized into two folders, for example: `pre-CHF/` and `post-CHF/`.
- Each folder contains multiple image files (e.g., JPG or PNG).
- Make note of the paths to these directories, as you will need them when loading the data.

With these prerequisites in place, you will be ready to proceed. In the next section, we will move on to the practical aspects of loading and preparing the dataset for analysis.

3 Data Preparation

In this section, we focus on loading the boiling images from the provided dataset and preparing them for analysis. We will assume you have two main directories: `pre-CHF/` and `post-CHF/`, each containing multiple image files. Our goal is to:

1. Load all images into memory.
2. Convert them to a consistent format (e.g., grayscale or RGB).
3. Resize them to a common dimension for ease of processing.
4. Assign labels (0 for pre-CHF, 1 for post-CHF) so we can later evaluate external metrics.

3.1 Directory Structure and Assumptions

For this tutorial, let’s assume you have the following directory structure:

```
dataset/
    pre-CHF/
        img_01.jpg
        img_02.jpg
        ...
    post-CHF/
        img_101.jpg
        img_102.jpg
        ...
```

Here:

- `pre-CHF/` contains images before the critical heat flux point.
- `post-CHF/` contains images after the CHF point.

We will label `pre-CHF` images as `0` and `post-CHF` images as `1`.

3.2 Choosing Image Dimensions and Color Format

We will resize all images to a uniform size, for example, 150×150 pixels. This is a common step since images may come in varying sizes and orientations.

For color format, we have two choices:

- **Grayscale**: Simplifies data by having only one intensity channel.
- **RGB**: Preserves full color information, which might be useful for certain feature extraction methods.

  For the Flatten approach, using grayscale can reduce data size and complexity. For Conv_Flatten, since the VGG16 model expects RGB inputs, we will use RGB images there. We will show both approaches in code.

3.3 Loading the Data

We will use the Python Pillow library (`PIL`) to load and resize images, and `os` to navigate directories. Below is a code snippet to load all images and record their labels:

```
import os
import numpy as np
from PIL import Image

pre_chf_dir = 'dataset/pre-CHF'
post_chf_dir = 'dataset/post-CHF'

pre_chf_images = [os.path.join(pre_chf_dir, f) for f in os.listdir(pre_chf_dir) if f.
    endswith('.jpg')]
post_chf_images = [os.path.join(post_chf_dir, f) for f in os.listdir(post_chf_dir) if f.
    endswith('.jpg')]

all_paths = pre_chf_images + post_chf_images
labels = [0]*len(pre_chf_images) + [1]*len(post_chf_images)
labels = np.array(labels)

img_height, img_width = 150, 150
```

At this point, `all_paths` contains all image file paths, and `labels` is a corresponding array of 0s and 1s.

3.4 Preprocessing and Resizing

Let's write a helper function to load and preprocess each image. Here, we will convert images to grayscale and resize them for the Flatten approach. For Conv_Flatten, we will later use RGB format.

```
def load_and_preprocess_images(image_paths, img_height=150, img_width=150, grayscale=True
    ):
    data = []
    for path in image_paths:
        img = Image.open(path)
        if grayscale:
            img = img.convert('L')  # Convert to grayscale
        else:
            img = img.convert('RGB')  # Keep RGB format if needed
        img = img.resize((img_width, img_height))
        arr = np.array(img)
        data.append(arr)
    return np.array(data)

# Example: Load all images in grayscale for Flatten approach
images_gray = load_and_preprocess_images(all_paths, img_height, img_width, grayscale=True
    )
print("Loaded images shape:", images_gray.shape)
```

The `images_gray` array now holds all images as 150×150 arrays. The shape should look like `(num_images, 150, 150)`. We have preserved spatial information at this stage, and we will decide how to extract features in the next steps.

3.5 Storing Data for Later Steps

Keep `images_gray` and `labels` handy, as we will use them in the Feature Extraction section to create Flatten features. For the Conv_Flatten approach, we will similarly load and store RGB images.

```
# If we need RGB for Conv_Flatten:
images_rgb = load_and_preprocess_images(all_paths, img_height, img_width, grayscale=False
    )
print("Loaded RGB images shape:", images_rgb.shape)
```

3.6 Verifying the Data

Before proceeding, it's a good practice to verify a few images:

```
import matplotlib.pyplot as plt

plt.imshow(images_gray[0], cmap='gray')
plt.title(f"Label: {labels[0]} (0=pre-CHF, 1=post-CHF)")
plt.show()
```

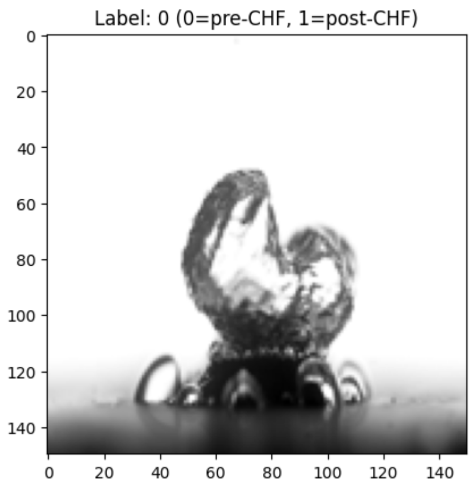

Figure 1: Verifying the Data

This quick check ensures that the resizing and loading worked correctly. If the image looks reasonable and matches the expected label, you are ready to move on.

With the data now loaded, resized, and preprocessed, we can proceed to the next step: extracting meaningful features from these images.

4 Feature Extraction

The next step is to convert raw images into a form that is more suitable for clustering and dimensionality reduction. We will explore two feature extraction approaches:

1. **Flatten**: Directly convert images into one-dimensional vectors of pixel intensities.
2. **Conv_Flatten**: Use a pre-trained convolutional neural network (CNN) to extract features before flattening.

4.1 Flatten Approach

The Flatten approach is the most straightforward. Each image, currently a 150×150 array (if grayscale) or a $150 \times 150 \times 3$ array (if RGB), is reshaped into a single long vector. For grayscale images, this results in a 22,500-dimensional vector per image ($150 \times 150 = 22{,}500$). While this preserves raw pixel information, it does not incorporate any learned features and may include a lot of redundant information.

4.1.1 Flattening Grayscale Images

Assume we have our grayscale images stored in `images_gray` as described in the Data Preparation section. We can flatten them easily:

```
# Flatten images from (num_images, 150, 150) to (num_images, 22500)
num_images = images_gray.shape[0]
flatten_data = images_gray.reshape(num_images, -1)
print("Flattened data shape:", flatten_data.shape)
```

This gives us a `flatten_data` array of shape (num_images, 22500), suitable for passing into dimensionality reduction and clustering algorithms.

4.2 Conv_Flatten Approach

While Flatten captures raw pixel values, it does not leverage the power of feature extraction learned by convolutional neural networks. A pre-trained CNN, such as VGG16 trained on ImageNet, can serve as a powerful feature extractor. By removing the top classification layers, we can feed in our images and extract lower-level features (e.g., edges, textures, and shapes) learned from a large-scale dataset.

4.2.1 Conceptual Steps

1. Load a pre-trained CNN model (e.g., VGG16) without its top (fully connected) layers. 2. Preprocess images to fit the model's input requirements (e.g., RGB, image size, mean subtraction). 3. Pass each image through the CNN to obtain a feature map. 4. Flatten the feature map into a one-dimensional vector. This vector now represents a rich set of learned features rather than raw pixels.

4.2.2 Code Implementation

First, ensure that you have TensorFlow and Keras installed. Then, we load VGG16 without top layers and extract features.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input

# Load VGG16 model without top layers
model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

def extract_cnn_features(images_rgb):
    features = []
    for img_arr in images_rgb:
        # Expand dimensions to create batch of size 1
        img_batch = np.expand_dims(img_arr, axis=0)
        # Preprocess for VGG16 (mean subtraction, etc.)
        img_batch = preprocess_input(img_batch)
        # Extract features
        feat = model.predict(img_batch)
        # Flatten the feature map
        feat_flat = feat.flatten()
        features.append(feat_flat)
    return np.array(features)

conv_data = extract_cnn_features(images_rgb)
print("Conv_Flatten data shape:", conv_data.shape)
```

This process may take longer depending on the number of images and the speed of your hardware. After running it, `conv_data` might have a shape like (num_images, $feature\_dim$), where $feature\_dim$ depends on the CNN architecture (often a few thousand features).

4.3 Choosing Between Flatten and Conv_Flatten

The Flatten approach is simpler and does not require a CNN. It might retain very fine-grained pixel-level details, which could be either helpful or noisy.

The Conv_Flatten approach leverages patterns learned from a large dataset and may extract more meaningful features for complex image patterns. However, these features might not always be perfectly aligned with the domain-specific details of your boiling images.

We will apply both Flatten and Conv_Flatten methods and compare their performance after dimensionality reduction and clustering. This comparison will highlight which representation best captures the underlying differences between pre-CHF and post-CHF images.

4.4 Next Steps

Now that we have two sets of features (`flatten_data` and `conv_data`), we can proceed to apply dimensionality reduction techniques (PCA, t-SNE, UMAP) to simplify and visualize these high-dimensional vectors.

5 Dimensionality Reduction

After extracting features, you may have very high-dimensional data:

– For Flatten: Each image is represented by thousands of pixel values.

– For Conv_Flatten: Each image is represented by a feature vector extracted from a CNN.

High-dimensional data can be difficult to cluster and visualize. Dimensionality reduction techniques help by projecting data into a lower-dimensional space (e.g., 2D or a small number of components) while preserving important structures or variations.

5.1 PCA (Principal Component Analysis)

PCA is a linear dimensionality reduction method. It finds directions (principal components) along which the data varies the most. By selecting a small number of principal components, you can reduce dimensionality while retaining as much variance as possible.

Key idea: PCA does not necessarily produce an easily interpretable space for visualization, but it is fast and often serves as a good baseline.

5.1.1 Applying PCA in Code

Suppose we have our chosen feature set, e.g., `flatten_data` or `conv_data`. We can apply PCA as follows:

```python
from sklearn.decomposition import PCA

def apply_pca(data, n_components=50):
    pca = PCA(n_components=n_components)
    reduced = pca.fit_transform(data)
    return reduced

# Example with flatten_data
reduced_pca_flatten = apply_pca(flatten_data, n_components=50)
print("PCA reduced shape:", reduced_pca_flatten.shape)
```

This will transform your data into 50 principal components. You can adjust `n_components` based on how much variance you want to preserve.

5.2 t-SNE (t-Distributed Stochastic Neighbor Embedding)

t-SNE is a non-linear dimensionality reduction technique often used for visualization. It tries to preserve local neighborhoods in the data, making points that are similar in high-dimensional space stay close in the 2D or 3D projection.

Key idea: t-SNE can create very informative 2D plots. However, it can be slow and has a few parameters (like perplexity) that influence the result. It is also primarily used for visualization, not for scaling to very large datasets.

5.2.1 Applying t-SNE in Code

```python
from sklearn.manifold import TSNE

def apply_tsne(data, n_components=2, random_state=42):
    tsne = TSNE(n_components=n_components, random_state=random_state)
    reduced = tsne.fit_transform(data)
    return reduced

# Example with conv_data
reduced_tsne_conv = apply_tsne(conv_data)
print("t-SNE reduced shape:", reduced_tsne_conv.shape)
```

After running t-SNE, you will get a 2D array that you can easily visualize with a scatter plot.

5.3 UMAP (Uniform Manifold Approximation and Projection)

UMAP is another non-linear technique similar in spirit to t-SNE, but often faster and better at preserving some global structure. UMAP can reveal meaningful clusters in a 2D space and is a popular choice for exploratory data analysis.

Key idea: UMAP creates a low-dimensional embedding that tries to preserve both local neighborhoods and some global relationships. It typically produces stable and interpretable embeddings.

5.3.1 Applying UMAP in Code

```python
import umap

def apply_umap(data, n_components=2, random_state=42):
    reducer = umap.UMAP(n_components=n_components, random_state=random_state)
    reduced = reducer.fit_transform(data)
    return reduced

# Example with flatten_data
reduced_umap_flatten = apply_umap(flatten_data)
print("UMAP reduced shape:", reduced_umap_flatten.shape)
```

5.4 Choosing a Dimensionality Reduction Method

The choice between PCA, t-SNE, and UMAP depends on your goals:

– **PCA**: Good for a quick, linear reduction and can retain variance systematically. Faster and deterministic, but might not reveal complex structures.

– **t-SNE**: Excellent for visualizing clusters in a small dataset, but can be slow and sensitive to parameters.

– **UMAP**: Often a good balance between speed, interpretability, and quality of embedding. Good for both exploration and visualization.

5.5 Next Steps

In the upcoming sections, we will:

1. Take the reduced data (e.g., from PCA, t-SNE, or UMAP).
2. Apply clustering (K-Means) to group the images.
3. Compare how well each combination (Flatten+PCA, Flatten+t-SNE, Flatten+UMAP, etc.) performs.

By systematically exploring each combination, we will identify which pipeline produces clusters that best align with the pre-CHF and post-CHF labels.

6 Clustering

Now that we have reduced the dimensionality of our feature vectors using PCA, t-SNE, or UMAP, the next step is to cluster the data. Clustering will help us group similar images together without using their labels. Since we know the dataset contains two conditions (pre-CHF and post-CHF), we will set $k = 2$ in K-Means.

6.1 K-Means Clustering

K-Means is a popular and straightforward clustering algorithm:

1. Initialize k cluster centroids.
2. Assign each data point to the nearest centroid.
3. Update centroids by taking the mean of all points assigned to each cluster.
4. Repeat until convergence.

We choose $k = 2$ because we have two regimes. Even though we are not using labels to guide clustering, we want to see if the algorithm can discover two natural groups that correspond to these conditions.

6.2 Applying K-Means to Reduced Data

You can cluster the output of any dimensionality reduction method. For example, if you used Flatten + UMAP, you might have a variable `reduced_umap_flatten` holding your 2D embeddings. Let's apply K-Means to that data:

```python
from sklearn.cluster import KMeans

def cluster_data(data, n_clusters=2, random_state=42):
    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
    cluster_labels = kmeans.fit_predict(data)
    return cluster_labels

# Example: clustering Flatten+UMAP data
cluster_labels_flatten_umap = cluster_data(reduced_umap_flatten)
print("Cluster labels shape:", cluster_labels_flatten_umap.shape)
```

After this, `cluster_labels_flatten_umap` is an array of the same length as the number of images, with values 0 or 1 indicating the cluster assignment of each image.

6.3 Multiple Combinations

We won't just cluster one combination. We will:

– Take Flatten features and reduce them with PCA, t-SNE, UMAP, and cluster each result.
– Take Conv_Flatten features and reduce them with PCA, t-SNE, UMAP, and cluster each result.

This gives us six pipelines in total:

$$\underbrace{\text{Flatten + PCA, Flatten + t-SNE, Flatten + UMAP}}_{\text{Flatten-based}}\ \underbrace{\text{Conv\_Flatten + PCA, Conv\_Flatten + t-SNE, Conv\_Flatten + UMAP}}_{\text{Conv\_Flatten-based}}$$

For each pipeline, we will store the resulting cluster labels. Then, we will use evaluation metrics to understand how good these clusters are.

6.4 Next Steps

Clustering alone tells us how points are grouped, but not how good the clusters are. To assess this, we need metrics. In the next section, we will introduce internal metrics (that do not require labels) and external metrics (that use true labels) to evaluate the quality of our clusters. We will then compare all pipelines and see which one performs best.

7 Evaluation Metrics

Clustering produces groups without using the true labels, so how do we know if the clusters are meaningful or correspond well to the known classes (pre-CHF vs. post-CHF)? To address this question, we use two types of metrics:

1. **Internal Metrics**: Evaluate cluster quality based solely on the data and cluster assignments, without using true labels.
2. **External Metrics**: Compare the cluster assignments to the known true labels, providing a direct measure of how well clustering recovers the actual classes.

7.1 Internal Metrics

Internal metrics assess how well-separated and cohesive the clusters are:

– **Silhouette Score**: For each sample, the silhouette value measures how similar it is to points in its own cluster compared to points in other clusters. Values range from -1 to 1:
  ∗ Near 1: The sample is well within its cluster.
  ∗ Near 0: The sample is close to the decision boundary.
  ∗ Negative values: The sample might be assigned to the wrong cluster.

  A higher average silhouette score indicates better cluster quality.
– **Davies-Bouldin Index (DB)**: This index measures the average similarity between clusters. Lower values are better, indicating that clusters are more distinct and have less overlap.

7.1.1 Code for Internal Metrics

```python
from sklearn.metrics import silhouette_score, davies_bouldin_score

def internal_metrics(data, cluster_labels):
    sil = silhouette_score(data, cluster_labels)
    db = davies_bouldin_score(data, cluster_labels)
    return sil, db

# Example:
sil, db = internal_metrics(reduced_umap_flatten, cluster_labels_flatten_umap)
print("Silhouette Score:", sil)
print("Davies-Bouldin Index:", db)
```

7.2 External Metrics

External metrics use the true labels to evaluate cluster quality:

– **Adjusted Rand Index (ARI)**: The ARI measures the similarity between the clusters and the true labels, correcting for chance. ARI = 1 means perfect agreement with the labels, while ARI near 0 means random alignment.

– **Purity**: Purity measures how often the majority of points in a cluster share the same true label. Purity = 1 means each cluster perfectly corresponds to a single class.

7.2.1 Code for External Metrics

```python
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics.cluster import contingency_matrix
import numpy as np

def purity_score(true_labels, cluster_labels):
    cont_matrix = contingency_matrix(true_labels, cluster_labels)
    return np.sum(np.amax(cont_matrix, axis=0)) / np.sum(cont_matrix)

def external_metrics(true_labels, cluster_labels):
    ari = adjusted_rand_score(true_labels, cluster_labels)
    pur = purity_score(true_labels, cluster_labels)
    return ari, pur

# Example:
ari, pur = external_metrics(labels, cluster_labels_flatten_umap)
print("Adjusted Rand Index:", ari)
print("Purity:", pur)
```

7.3 Interpreting the Metrics

– If internal metrics (Silhouette, DB) show well-separated clusters and external metrics (ARI, Purity) indicate strong alignment with true labels, then the clustering method is performing well.

– Discrepancies may occur. For example, clusters might look well-separated (high silhouette) but not align with true labels (low ARI). This suggests that the natural grouping in the data does not match the labeled categories.

7.4 Next Steps

With these metrics in hand, we can now systematically run all combinations (Flatten/Conv_Flatten with PCA/t-SNE/UMAP) and evaluate the results. In the next section, we will outline how to perform a systematic analysis and compare all pipelines to identify the best-performing approach.

8 Systematic Analysis

We have introduced two feature extraction methods (Flatten, Conv_Flatten), three dimensionality reduction techniques (PCA, t-SNE, UMAP), and one clustering algorithm (K-Means). Now, we will combine these steps to form a systematic analysis pipeline.

8.1 Combining All Steps

We have the following six pipelines:

$$\begin{gathered}\text{Flatten} + \text{PCA}, \quad \text{Flatten} + \text{t-SNE}, \quad \text{Flatten} + \text{UMAP}, \\ \text{Conv\_Flatten} + \text{PCA}, \quad \text{Conv\_Flatten} + \text{t-SNE}, \quad \text{Conv\_Flatten} + \text{UMAP}.\end{gathered}$$

For each pipeline, we will:

1. Select the feature extraction result (`flatten_data` or `conv_data`).
2. Apply one dimensionality reduction method.
3. Cluster the reduced data with K-Means.
4. Compute internal and external metrics.
5. Store the results for comparison.

8.2 Code Implementation

We assume we have variables `flatten_data`, `conv_data`, and `labels` available from previous steps. We will define a loop or set of calls to systematically run all methods.

```python
# Assume we have these functions already defined:
# apply_pca(data), apply_tsne(data), apply_umap(data)
# cluster_data(data), internal_metrics(data, cluster_labels), external_metrics(
    true_labels, cluster_labels)

feature_extractions = {
    'Flatten': flatten_data,
    'Conv_Flatten': conv_data
}

dim_reductions = {
    'PCA': apply_pca,
    't-SNE': apply_tsne,
    'UMAP': apply_umap
}

results = []

for feat_name, feat_data in feature_extractions.items():
    for dim_name, dim_func in dim_reductions.items():
        # Apply dimensionality reduction
        reduced_data = dim_func(feat_data)

        # Cluster the reduced data
        cluster_labels = cluster_data(reduced_data, n_clusters=2)

        # Compute metrics
        sil, db = internal_metrics(reduced_data, cluster_labels)
        ari, pur = external_metrics(labels, cluster_labels)

        # Store results in a dictionary
        results.append({
            'Feature_Extraction': feat_name,
            'Dim_Reduction': dim_name,
            'Silhouette': sil,
            'Davies-Bouldin': db,
            'ARI': ari,
            'Purity': pur
        })
```

```python
# Convert results to a table-like structure (optional)
import pandas as pd
df_results = pd.DataFrame(results)
print(df_results)
```

This code systematically tries each combination and records the metrics. The final `df_results` DataFrame (if you choose to use pandas) will help you see which method performed best.

8.3 Analyzing the Results

Once you have this table of results, you can:

- Identify which combination yields the highest ARI or Purity (for best alignment with true labels).
- Check which combination has a good balance of internal metrics (high Silhouette, low DB) and external metrics (high ARI, high Purity).

In the next section, we will discuss how to present and interpret these results, show visualizations (e.g., scatter plots of UMAP embeddings, confusion matrices, and silhouette plots), and identify the best-performing pipeline.

9 Results and Discussion

After running all six pipelines, we have obtained a set of internal and external metrics for each combination. Table 1 summarizes these metrics:

Feature Extraction	**Dim. Reduction**	**Silhouette**	**DB**	**ARI**	**Purity**
Flatten	PCA	0.1426	2.6824	0.0008	0.5163
Flatten	t-SNE	0.4328	0.9570	0.9318	0.9827
Flatten	UMAP	0.4298	0.9771	0.5785	0.8803
Conv_Flatten	PCA	0.1023	2.7474	0.4833	0.8477
Conv_Flatten	t-SNE	0.4632	0.8629	0.9565	0.9890
Conv_Flatten	UMAP	0.6255	0.5549	0.9383	0.9843

Table 1: Clustering results for all combinations of feature extraction and dimensionality reduction. Silhouette and Davies-Bouldin (DB) are internal metrics; ARI and Purity are external metrics.

9.1 Identifying the Best Combination

From Table 1, we can observe several patterns:

- **Flatten + PCA** yields poor alignment with the true labels (ARI $\approx$ 0.0008, Purity $\approx$ 0.5163), indicating that this pipeline does not separate pre-CHF and post-CHF images effectively.
- **Flatten + t-SNE** performs significantly better, with ARI $\approx$ 0.9318 and Purity $\approx$ 0.9827. This suggests that t-SNE, combined with raw pixel features, can unveil a structure that closely matches the underlying classes.
- **Flatten + UMAP** achieves moderate results (ARI $\approx$ 0.5785, Purity $\approx$ 0.8803), better than Flatten+PCA but not as good as Flatten+t-SNE.
- **Conv_Flatten + PCA** improves upon Flatten+PCA (ARI $\approx$ 0.4833), but still lags behind the best performers.
- **Conv_Flatten + t-SNE** shows the best overall results: ARI $\approx$ 0.9565 and Purity $\approx$ 0.9890. This combination nearly perfectly aligns the clusters with the true labels, indicating that CNN-based features combined with t-SNE produce a very clear separation between pre-CHF and post-CHF images.

  - **Conv_Flatten + UMAP** also performs excellently (ARI $\approx$ 0.9383, Purity $\approx$ 0.9843), slightly lower than Conv_Flatten + t-SNE but still outstanding.

Among all pipelines, **Conv_Flatten + t-SNE** yields the highest ARI and Purity, making it the best performer overall.

9.2 Visualizing the Best Performer

To understand why Conv_Flatten + t-SNE is so effective, we can visualize the 2D t-SNE embedding of the data.

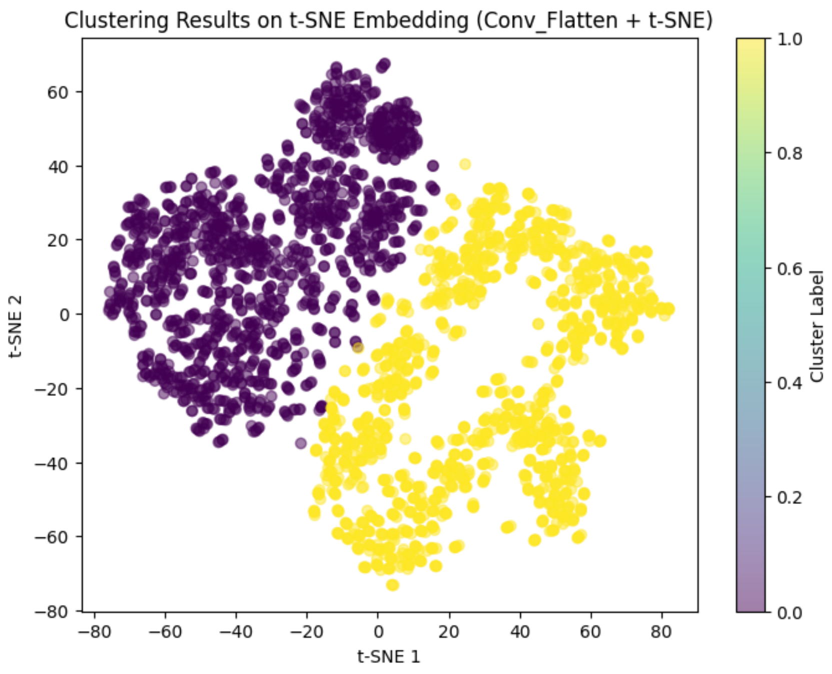

Figure 2: 2D t-SNE embedding (Conv_Flatten features) colored by cluster labels. Two well-separated clusters are visible.

Figure 2 shows the t-SNE embedding, with each point representing an image. Clusters assigned by K-Means appear as distinct groups.

Figure 3 colors the same embedding by the true labels. The near-perfect correspondence between clusters and true labels confirms the high ARI and Purity scores.

9.3 Confusion Matrix and Silhouette Plot

We can further support these findings with a confusion matrix and silhouette plot.

The confusion matrix (Figure 4) shows minimal misclassifications.

The silhouette plot (Figure 5) reveals that most points have positive and relatively high silhouette values, indicating cohesive and separated clusters.

9.4 Discussion

These results suggest that leveraging CNN-based feature extraction (Conv_Flatten) combined with a non-linear dimensionality reduction like t-SNE can effectively separate pre-CHF and post-CHF images. This likely occurs because the CNN extracts more meaningful, generalized features (edges, textures, shapes) that highlight subtle differences between boiling regimes. t-SNE then preserves local neighborhoods in a way that makes these differences stand out in the 2D embedding.

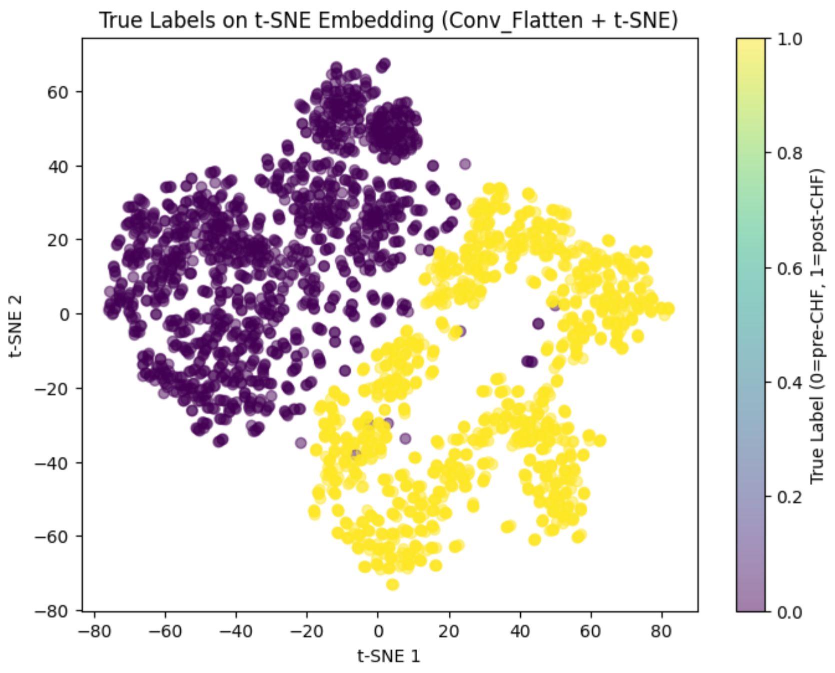

Figure 3: 2D t-SNE embedding (Conv_Flatten features) colored by true labels (0=pre-CHF, 1=post-CHF). The alignment with clusters is nearly perfect.

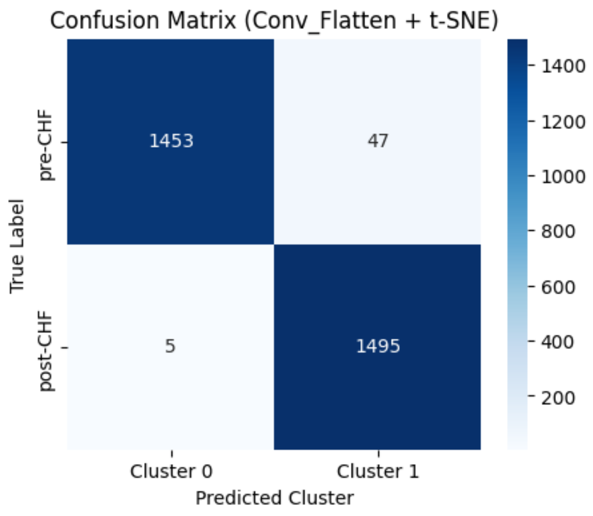

Figure 4: Confusion matrix for Conv_Flatten + t-SNE clustering. Most pre-CHF and post-CHF images fall into separate clusters.

In contrast, simpler methods (like Flatten+PCA) fail to uncover the structure corresponding to class labels. While PCA preserves variance, it may not highlight the discriminative features needed to differentiate pre-CHF from post-CHF images.

9.5 Next Steps

In the next section, we will conclude our findings, summarize key takeaways, and suggest potential future directions for students who wish to explore similar methods or improve upon these results.

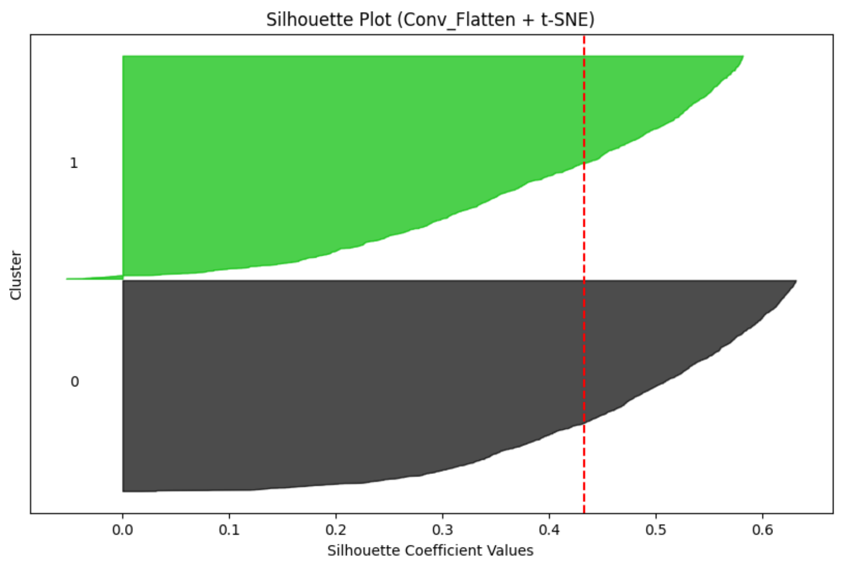

Figure 5: Silhouette plot for Conv_Flatten + t-SNE. High average silhouette values indicate well-defined clusters.

10 Conclusion

In this tutorial, we explored a complete pipeline for analyzing boiling images and determining whether unsupervised clustering techniques can separate pre-CHF and post-CHF regimes. Starting from raw image data, we covered each stage:

1. **Data Preparation:** We resized and preprocessed the boiling images into a uniform format.
2. **Feature Extraction:** We considered two approaches:
   - **Flatten:** Directly converting images into pixel-intensity vectors.
   - **Conv_Flatten:** Using a pre-trained CNN (VGG16) to extract richer, learned features.
3. **Dimensionality Reduction:** We applied PCA (linear) and t-SNE/UMAP (non-linear) to reduce dimensionality, making the data easier to cluster and visualize.
4. **Clustering:** We used K-Means with $k = 2$, aiming to separate the dataset into two groups without using labels.
5. **Evaluation Metrics:** Both internal (Silhouette, Davies-Bouldin) and external (ARI, Purity) metrics were employed to assess cluster quality.
6. **Systematic Analysis:** By evaluating all six combinations (Flatten/Conv_Flatten with PCA/t-SNE/UMAP), we identified which pipeline performed best.

Our results indicate that the choice of feature extraction and dimensionality reduction method significantly impacts clustering performance:

- Simple, linear methods (e.g., Flatten + PCA) did not yield clusters aligned with the true labels.
- Non-linear methods (t-SNE, UMAP) improved clustering quality for both Flatten and Conv_Flatten features.
- Among all combinations, **Conv_Flatten + t-SNE** produced the best results, achieving high ARI and Purity scores and clearly separating pre-CHF and post-CHF images. The CNN-based features, combined with t-SNE's ability to preserve local neighborhoods, led to near-perfect alignment with the true labels.

Visual inspections supported these conclusions:

- The t-SNE embedding colored by cluster labels showed two distinct groups with minimal overlap.

– The same embedding colored by true labels confirmed that the clusters correspond almost exactly to pre-CHF and post-CHF classes.
– A confusion matrix revealed only minor misclassifications.
– The silhouette plot, while indicating a moderate average score, still showed that points were generally well-assigned to their clusters, especially in one of the clusters.

Implications and Future Directions

These findings suggest that image-based cluster analysis can serve as a useful tool for understanding boiling regimes. By relying on learned features from a CNN and sophisticated dimensionality reduction techniques, we can uncover meaningful patterns that simple methods cannot.

For future exploration, students could:

– Experiment with different CNN architectures or fine-tune the CNN on boiling images for even more domain-specific features.
– Adjust parameters in t-SNE or try other non-linear methods like UMAP with different `n_neighbors` or `min_dist` parameters.
– Investigate other clustering algorithms (e.g., DBSCAN, Gaussian Mixture Models) and compare results.
– Explore other datasets or incorporate temporal information if available, to understand how boiling patterns evolve over time.

By following the steps in this tutorial, students now have a practical roadmap for applying unsupervised methods—feature extraction, dimensionality reduction, clustering, and metrics evaluation—to real-world image data, enabling them to tackle new challenges in machine learning and image analysis.