# Detection of Recyclable Plastics

**Anamika Singh**
CSE 700
UBIT: 50485492
singh27@buffalo.edu

## Abstract

This report delves into the utilization of machine learning and deep learning techniques to detect and classify recyclable plastics based on image and spectral data. Through the implementation of various algorithms, plastic categorization is achieved. This study not only serves as a valuable learning experience but also aims to provide results with potential utility for further applications.

## 1 Introduction

The Environmental Protection Agency (EPA) reports that merely a quarter of waste undergoes recycling, with only $60\%$ of US municipalities offering curbside recycling for their residents. Alarmingly, plastic recycling stands at a mere $8\%$, while an additional $16\%$ is subjected to combustion. The majority, a staggering $76\%$, finds its way into landfills. Several factors contribute to this low plastic recycling rate, notably the necessity for preliminary sorting based on plastic type.

Plastic re-claimers rely on the Resin Identification Code (RIC), a categorization system developed by the Society of the Plastics Industry. This code assigns resin numbers to various polymer types: Polyethylene terephthalate (PET) is labeled as resin code 1, High Density Poly Ethylene (HDPE) as resin code 2, Polyvinyl Chloride (PVC) as resin code 3, Low Density Polyethylene (LDPE) as resin code 4, Polypropylene (PP) as resin code 5, Polystyrene (PS) as resin code 6, and other plastic types as resin code 7. Addressing these challenges and complexities in plastic recycling is vital for improving the current scenario.

## 2 Dataset

### 2.1 Image Dataset

Through catalogs of plastic manufacturers and real world plastic objects, 835 images are collected, forming an image database. The dataset contains images of the five types of plastics (PET, HDPE, LDPE, PP, PS). Given an input image, we want to predict the plastic type associated with the image.

### 2.2 Spectral Dataset

Plastic spectral database size: 2505 spectra files. A database of 835 plastic items (167 objects per type) consisting of PET, HDPE, LDPE, PP, and PS were collected. A spectrometer was used to acquire mid-infrared spectra of the prepared samples. Three spectra per unique sample were recorded to build the database and introduce variability in intensity between each measurement. Spectra were acquired from 4000 to 650 cm -1 and processed. Each spectrum contains 3474 data points, where each point represents the intensity in absorbance units at a given wavenumber.

I Can't Believe It's Not Better Workshop III at NeurIPS 2022.

# 3 Implementation

This section delves into the heart of the study, where a diverse range of machine learning and deep learning algorithms were applied to both image and spectral datasets for the classification of recyclable plastics

## 3.1 Algorithms for Image dataset

### 3.1.1 CNN with Mutual information

**Convolutional Neural Network** (CNN) is used to perform plastic classification and features extracted from **Mutual information** were fed to it. Mutual information quantifies the relationship between variables, helping us identify significant features in the context of plastic classification.

- **Preprocessing**: Images from the training subset are resized to a consistent dimension of 200x200 pixels and normalized to the range [0, 1]. The images are then converted to tensors, and mutual information scores are computed.
- **Feature Selection**: Using mutual information scores, top features were selected that have the most informative power in differentiating between plastic types. The selected features are converted to a PyTorch tensor, which serves as the input to the CNN model.
- **CNN Architecture**: The CNN architecture consists of convolutional layers, ReLU activation functions, and max-pooling layers. The model learns to capture hierarchical features from the input images. A fully connected (linear) layer with appropriate output dimensions maps the learned features to the target classes.
- **Training**: The CNN is trained using a custom dataset, where each instance contains the selected features and their corresponding labels. The model is optimized using the stochastic gradient descent (SGD) optimizer, and the cross-entropy loss function measures the discrepancy between predicted and actual labels. The training process spans multiple epochs, with batch-wise optimization
- **Evaluation**: To assess the algorithm's generalization ability, it is evaluated on a test dataset. The test accuracy is calculated by comparing predictions against true labels.

### 3.1.2 Transfer Learning with ResNet

Power of transfer learning is harnessed by utilizing a pre-trained ResNet model for plastic classification using image data. The ResNet model has shown exceptional performance in image classification tasks, and it is adapted for our specific plastic classification problem.

- **Preprocessing**: Images from the training subset are resized to a consistent dimension of 200x200 pixels and normalized to the range [0, 1]. The images are then converted to tensors and then fed to ResNet.
- **Transfer Learning with ResNet**: We load a pre-trained ResNet-18 model and replace its fully connected (classification) layer with a new linear layer suitable for our plastic classification task. The model's architecture captures complex image features using convolutional layers and residual connections, and the final classification layer is modified to have output dimensions matching the number of plastic categories.
- **Training**: The modified ResNet model is trained using the training dataset. We employ the Adam optimizer and the cross-entropy loss function for optimization. The model undergoes multiple epochs of training, with batch-wise optimization using a training DataLoader.
- **Model Performance Visualization**: The algorithm's performance is visualized using plots of training loss and training accuracy across epochs.
- **Test Accuracy**: The model's generalization ability is assessed on a separate test dataset, and the test accuracy is computed by comparing model predictions against true labels.

#### 3.1.3 Vision Transformer (ViT)

This algorithm implements a Vision Transformer (ViT) with enhanced capabilities using Shifted Patch Tokenization and Locality Self Attention (LSA) for plastic classification. The algorithm employs a modern computer vision approach that utilizes the power of transformers for handling image data.

- **Preprocessing**: The dataset is preprocessed by resizing images to 200x200 pixels and normalizing pixel values to [0, 1]. Data augmentation techniques like horizontal flipping, rotation, and zoom are applied to improve model generalization
- **Model Architecture**: The algorithm creates a ViT-based model with multiple Transformer layers. Each Transformer layer includes layer normalization, multi-head self-attention (with optional LSA), and feedforward neural networks. The final representation is obtained through layer normalization, flattening, and dropout.
- **Training**: The model is compiled using the AdamW optimizer with warm-up cosine learning rate scheduling. The model is trained on the training dataset with a batch size of 150 for a specified number of epochs.
- **Evaluation**: After training, the model is evaluated on the validation dataset to assess its performance.
- **Analysis and Visualization**: The algorithm analyzes the performance through additional evaluations such as confusion matrices, classification reports, and visualization of incorrectly classified images.
- **Test Accuracy**: The algorithm outputs test accuracy and top-5 accuracy metrics, showing the model's capability to classify plastic types. It visually presents confusion matrices to provide insights into the model's performance across different classes.

#### 3.1.4 ConvNeXt

The ConvNeXt model is a convolutional neural network (CNN) that is inspired by the transformer architecture. The ConvNeXt model has a hierarchical self-attention mechanism that allows it to learn long-range dependencies in the input data.

- **Preprocessing**: Images from the training subset are resized to a consistent dimension of 200x200 pixels and normalized to the range [0, 1].
- **Architecture**: The ConvNeXt model consists of a stack of ConvNeXtStage blocks. Each ConvNeXtStage block consists of a depthwise convolutional layer, a pointwise convolutional layer, and a residual connection. The depthwise convolutional layer allows the model to learn local dependencies between channels. The pointwise convolutional layer allows the model to map the output of the depthwise convolutional layer to the desired number of output channels. The residual connection helps to improve the training stability of the model.
- **Training**: The ConvNeXt model is trained using the Adam optimizer with a learning rate of 0.001. The model is trained for 100 epochs
- **Evaluation**: The ConvNeXt model is evaluated using the cross-entropy loss function. The model's performance is measured by the accuracy on the validation dataset.

#### 3.1.5 GAN for Image Generation

This algorithm employs a Generative Adversarial Network (GAN) to generate synthetic images and classify them into predefined classes. GANs consist of two main components: the generator and the discriminator. The generator generates synthetic images, while the discriminator tries to distinguish between real and generated images.

- **Preprocessing**: Images from the training subset are resized to a consistent dimension of 200x200 pixels and normalized to the range [0, 1]. The image data is permuted to follow the (batch size, channels, height, width) format required by PyTorch.
- **Generator**: A generator neural network is defined, comprising fully connected layers with ReLU activation functions. The generator transforms random noise vectors (latent space) into images. The output of the generator is reshaped to match the image dimensions (3 channels, 200x200 pixels).

- **Discriminator**: A discriminator neural network is defined to classify images as real or fake. It consists of fully connected layers with LeakyReLU activation functions. The final layer uses a sigmoid activation function to output a probability score (0 to 1) indicating the likelihood of an image being real.
- **GAN Training**: During training, both the generator and discriminator networks are updated iteratively. For the discriminator, it learns to differentiate between real and generated images. For the generator, it learns to generate images that fool the discriminator. The binary cross-entropy loss is used as the loss function
- **Results and Insights**: The generated images and their progression over epochs are visualized. The algorithm tracks the losses of the generator and discriminator to assess the convergence and training progress.

### 3.2 Algorithms for Spectral dataset

#### 3.2.1 Recurrent Convolutional Neural Network

This algorithm employs a Recurrent Convolutional Neural Network (RCNN) for the classification of spectral data. The RCNN model combines convolutional layers for feature extraction, max pooling for down-sampling, and an LSTM layer for temporal modeling. The resulting features are fed into fully connected layers for classification.

- **Data Preprocessing**: Spectral data is loaded from a CSV file containing both features and labels. The dataset is split into training and testing sets using a 70-30 split ratio. The spectral data is converted into PyTorch tensors. The training and testing datasets are loaded using TensorDataset and DataLoader.
- **RCNN Model**: The RCNN model is defined using PyTorch's nn.Module class. The model comprises convolutional layers, ReLU activation functions, max pooling layers, an LSTM layer, and fully connected layers. The output of the model is a tensor containing predicted class scores.
- **Training**: The algorithm trains the RCNN model using a training loop. For each epoch, the model is trained on batches of data. The loss is calculated using the cross-entropy loss function. The gradients are calculated and the optimizer updates the model's weights.
- **Evaluation**: After training, the model is set to evaluation mode. The algorithm evaluates the model's performance on the test set. The predicted labels are compared with the ground truth labels, and the accuracy is calculated.

#### 3.2.2 Denoising Autoencoder and K-means

This algorithm performs feature extraction and clustering on a given dataset using a Denoising Autoencoder and K-means clustering. The Denoising Autoencoder is used to learn compressed representations of the input data, and the K-means algorithm is then applied to cluster the extracted features.

- **Data Preprocessing**: The dataset is loaded from a CSV file containing both features and labels. The data is split into training and test sets using an 80-20 split ratio. The data is scaled using MinMaxScaler.
- **Model Definition**: The model consists of encoder and decoder layers. The encoder layers consist of linear transformations and ReLU activation functions. The bottleneck layer captures the compressed representation of the data. The decoder layers mirror the encoder architecture.
- **Training**: The model is trained to minimize the mean squared error (MSE) loss between the reconstructed data and the original data. The training is performed using an optimizer (Adam) for a specified number of epochs.
- **Feature Extraction**: The encoder layers of the trained autoencoder are used to extract features from both the training and test data. Principal Component Analysis (PCA) is applied to reduce the dimensionality of the extracted features to 2D for visualization purposes.

- **K-Means Clustering**: K-means clustering is applied to the reduced-dimensional features. The number of clusters is set to 5 in this example. The cluster assignments are predicted for the test data.The silhouette score is calculated to measure the quality of the clustering. The silhouette score provides insights into the separation between clusters.
- **Visualization**: The 2D reduced features are plotted, colored according to the predicted cluster assignments. Cluster centers are marked with red "x" markers.

#### 3.2.3 Denoising Autoencoder and Random Forest

This algorithm demonstrates feature extraction using a Denoising Autoencoder and then performing classification using a Random Forest Classifier. The autoencoder is used to learn a compressed representation of the data, which is then used as features for the classification task.

- **Model Definition**: The model consists of encoder and decoder layers. The encoder layers consist of linear transformations and ReLU activation functions. The bottleneck layer captures the compressed representation of the data. The decoder layers mirror the encoder architecture.
- **Training**: The model is trained to minimize the mean squared error (MSE) loss between the reconstructed data and the original data. The training is performed using an optimizer (Adam) for a specified number of epochs.
- **Feature Extraction**: The encoder layers of the trained autoencoder are used to extract features from both the training and test data.
- **Classification using Random Forest**: A Random Forest Classifier model is defined. The model is fit on the extracted features from the training set. Predictions are made on the test set using the trained classifier.
- **Evaluation Metrics**: The accuracy of the classifier is calculated using the ground truth labels and predicted labels. A confusion matrix is generated to assess the performance of the classifier. The confusion matrix is normalized for better visualization. The time taken for the execution of the entire algorithm is recorded.

#### 3.2.4 Gradient Boosting Models - XGBoost and LightGBM

This algorithm demonstrates the training and evaluation of classification models using both XGBoost and LightGBM, which are popular gradient boosting libraries for tabular data classification tasks.

- **XGBoost Model** : Model parameters are defined for XGBoost:
  - objective: Specifies the objective function, which is set to 'multi:softmax' for multi-class classification.
  - num class: Specifies the number of classes in the classification task.
  - max depth: Specifies the maximum depth of each tree in the ensemble.
  - learning rate: Specifies the learning rate for the gradient boosting algorithm.
  - eval metric: Specifies the evaluation metric, set to 'mlogloss' for multi-class classification
- **LightGBM Model** : Model parameters are defined for LightGBM::
  - objective: Specifies the objective function, which is set to 'multi:softmax' for multi-class classification.
  - num class: Specifies the number of classes in the classification task.
- **Results**: Test accuracies for both XGBoost and LightGBM models are displayed.

#### 3.2.5 Ensemble methods - AdaBoost and Bagging

This algorithm demonstrates the utilization of an AdaBoost Classifier and Bagging Classifier with Decision Tree base estimators for the classification task.

- **Model Initialization** : initialize a base estimator using DecisionTreeClassifier with a specified maximum depth (e.g., max depth=2)

- **AdaBoost Classifier**: Create an AdaBoostClassifier instance:
  - Set the base estimator parameter to the initialized DecisionTreeClassifier.
  - Specify the boosting algorithm as "SAMME" using the algorithm parameter.
  - Specify the number of estimators using the n estimators parameter (e.g., n estimators=200).

  Fit the AdaBoost classifier on the training data.
- **Bagging Classifier**: Create a BaggingClassifier instance:
  - Set the base estimator parameter to the previously initialized DecisionTreeClassifier.
  - Specify the number of estimators using the n estimators parameter (e.g., n estimators=200).
  - Set the random seed using the random state parameter.

  Fit the Bagging classifier on the training data.
- **Results**: Predict the labels on the test data for both models using the trained bagging classifier and the trained adaboost classifier . Calculate the accuracy of the predictions using accuracy score.

#### 3.2.6 Gaussian Process Classifier with PCA

This algorithm combines Principal Component Analysis (PCA) for dimensionality reduction with a Gaussian Process Classifier for a classification task.

- **PCA Dimensionality Reduction**: Set the desired number of principal components using n components. Initialize a PCA instance with the specified number of components. Apply PCA on the training set using fit transform to obtain the reduced feature representation x train pca. Apply the same PCA transformation on the test set using transform to obtain x test pca.
- **Gaussian Process Classifier**: Initialize the kernel for the Gaussian Process Classifier, for example, using the radial basis function (RBF) kernel. Create a GaussianProcessClassifier instance: Set the kernel to the previously initialized kernel. Set the random seed using the random state parameter. Fit the Gaussian Process Classifier on the reduced training dataset (x train pca) and corresponding labels (y train).
- **Accuracy Calculation**: Calculate the accuracy of the trained classifier on both the training and test datasets: Use the score method of the Gaussian Process Classifier to obtain the accuracy. Print the train and test accuracies

## 4 Results and Conclusions

The following section presents the outcomes achieved through the application of various algorithms to the dataset. The main objective is to evaluate the performance of each algorithm, focusing on classification accuracy and pertinent metrics. Through this assessment, valuable insights into algorithm effectiveness are anticipated, aiding in the identification of the optimal approach for the dataset at hand.

| Deep Learning algorithms | Epochs | Train Accuracy | Test Accuracy |
|---|---|---|---|
| Transfer Learning (Resnet18) | 10 | 95.18 | 85.88 |
| Vision Transformer | 50 | 91.26 | 85.88 |
| ConvNeXt | 100 | 85.5 | 30.66 |
| Convolutional Neural Networks | 1000 | 76.54 | 44.71 |

Figure 1: DL Algorithms results on Image Data.

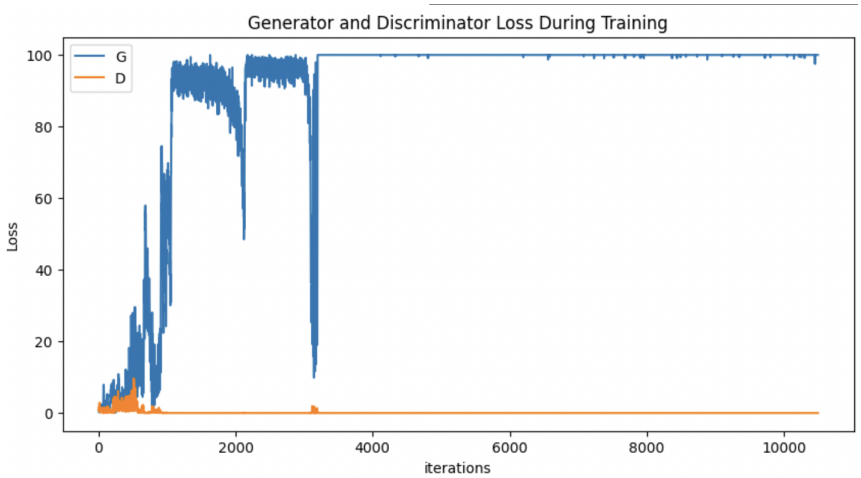


Figure 2: GAN Loss

### 4.1 Image classification

Four distinct classification algorithms were employed to categorize images, along with an image generation algorithm. The subsequent figure presents a comprehensive analysis encompassing both training and testing accuracies achieved by these classification algorithms.

The four considered algorithms are :

ResNet18 (2015), Vision Transformer (2021), ConvNeXt (2022), Convolutional Neural Network (1980)

The implementation of the Generative Adversarial Network (GAN) for image generation did not yield satisfactory outcomes. This could potentially be attributed to inherent challenges and limitations within the approach. Figure 3 shows generator and discriminator loss. A stable GAN has low variance between both the losses which is near 0.5. However, this pattern is not evident in the presented results.

For further comparison between ResNet and ViT, Figure 3 and Figure 4 provide a comparison between there losses as well as accuracies.

Based on the obtained results, it can be inferred that the ResNet architecture demonstrates the most promising performance among the existing architectures for the given dataset. While ConvNeXt shows potential for improvement, its relatively recent introduction makes it a more intricate model to fine-tune.

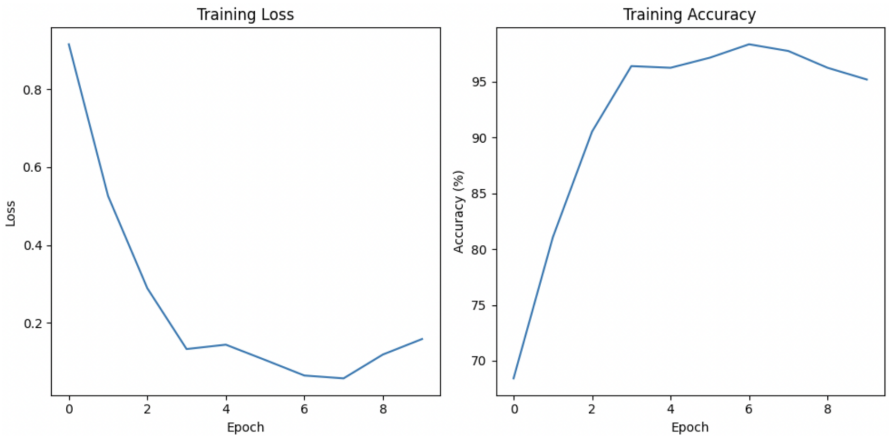


Figure 3: ResNet18 Epochs vs Training Loss and Training Accuracy.

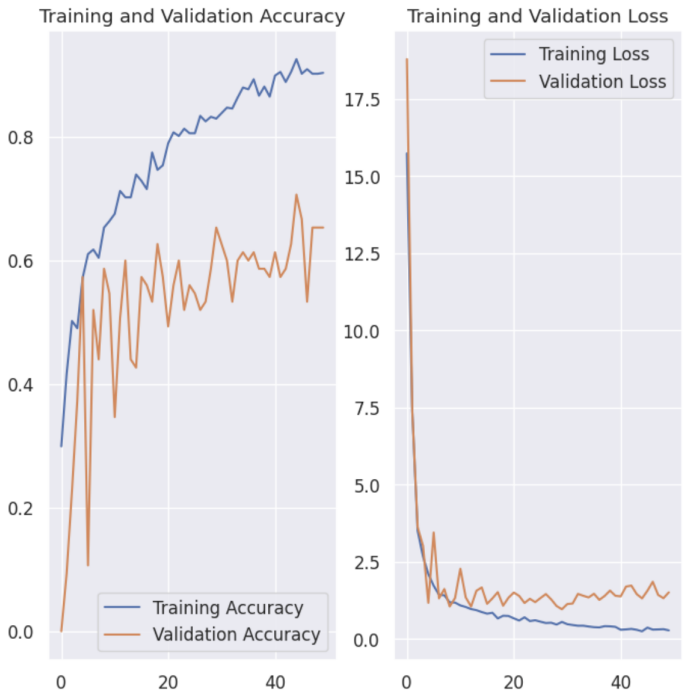


Figure 4: ViT Training Accuracy vs Validation Accuracy and Training Loss vs Validation Loss.

### 4.2 Spectral classification

Figure 5 displays the attained accuracies of various models applied to the spectral dataset. Although several models exhibit promising accuracy outcomes, the execution times vary, particularly noticeable when working with larger datasets. Notably, the algorithms are arranged in ascending order based on their execution times.

| **Algorithms** | **Test Accuracy** |
|---|---|
| Gradient Boosting Models: LightGBM | 99.40 |
| Gradient Boosting Models: XGBoost | 99.00 |
| Gaussian Process Classifier | 98.60 |
| Recurrent Convolutional Neural Network | 90.69 |
| Ensemble methods: AdaBoost | 90.40 |
| Ensemble methods: Bagging | 78.19 |

Figure 5: Algorithms results on Spectral Dataset.

The implementation of the Denoising Autoencoder (DAE) aimed to encode the data in a manner that would diminish its dimensions while retaining its essential features. By employing the DAE, the length of the sample sequence was effectively reduced from 3474 to 250, thereby resulting in a reduction in computational demands.

The classification task involved utilizing the Random Forest algorithm both with and without the integration of the Denoising Autoencoder. Although there exist minor disparities in the achieved accuracies, as shown in Figure 8, a more pronounced distinction emerges in terms of execution time. Notably, when dealing with larger datasets, this discrepancy in execution time could potentially magnify, thereby warranting careful consideration.

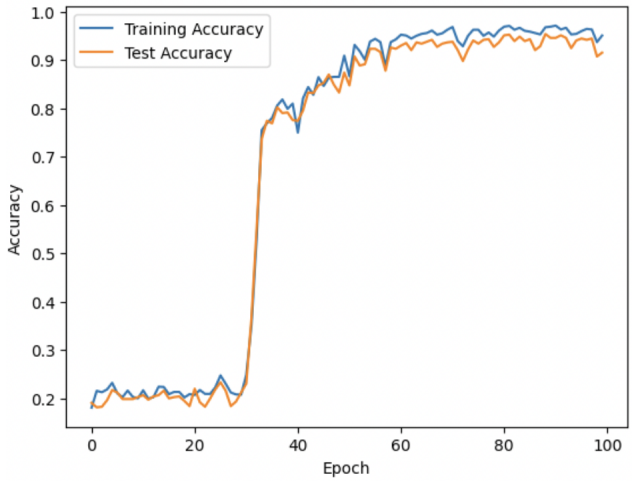


Figure 6: RCNN Training vs Test accuracy.

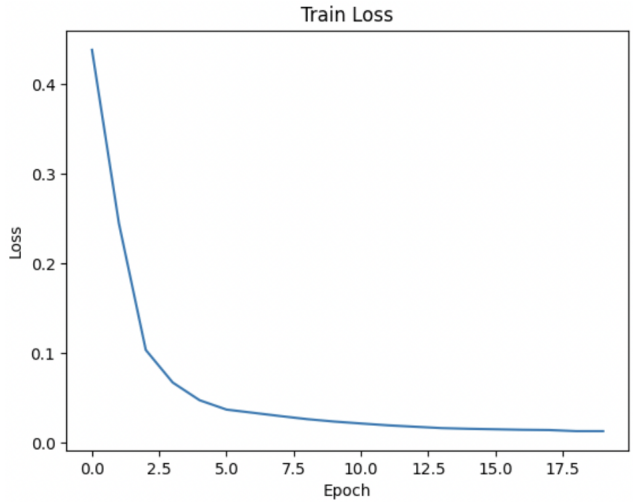


Figure 7: DAE loss over 20 epochs.

| Random Forest | Test Accuracy (%) | Execution Time (ms) |
|---|---|---|
| **Without DAE** | 99.60 | 15.61 |
| **With DAE** | 98.20 | 12.69 |

Figure 8: Random Forest performance without and with DAE.

Employing the Denoising Autoencoder (DAE) in conjunction with K-Means reveals notable distinctions in the nature of the resulting clusters. In particular, the silhouette score is notably higher when employing PCA (with a number of components equal to 500) in contrast to the utilization of DAE, yielding scores of 0.65 and 0.53, respectively. It's evident that when PCA is employed, the sequence length is twice that of DAE, with PCA achieving a sequence length of 500. Figure 9 show the clusters formed by K-Means.

The spectral dataset results highlight the potential of gradient boosting models for accuracy improvement. However, execution time considerations are influenced by the dataset's compact size. Notably, Denoising autoencoders offer a promising strategy for reducing computation costs. Combining them with classifiers like random forest yields both high accuracy and efficiency, even when scaling up to larger datasets.

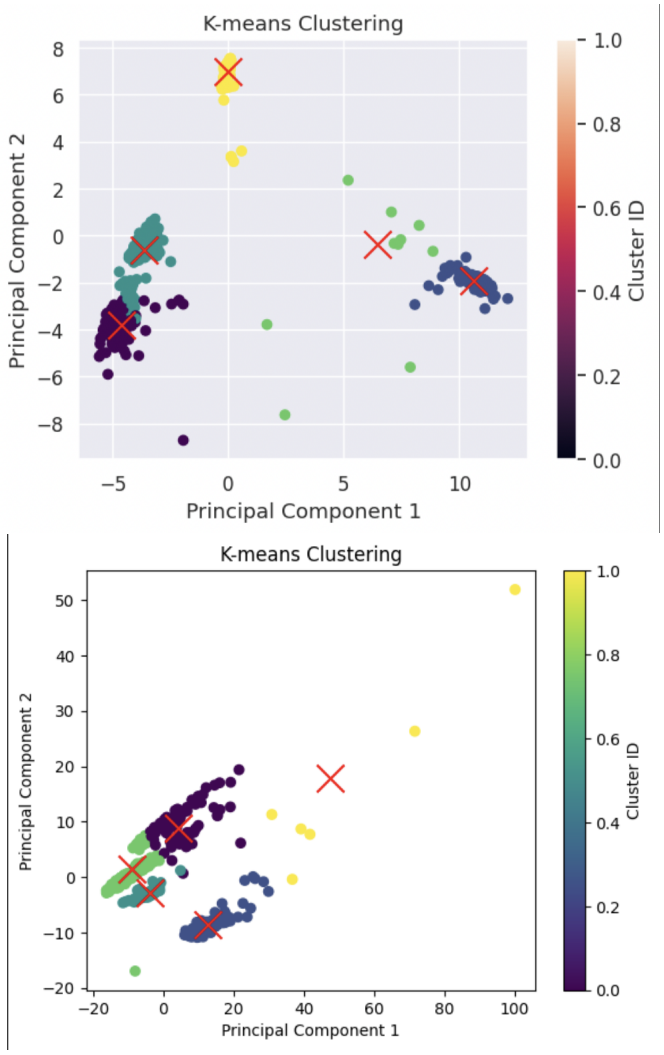


Figure 9: Top - K-Means in conjuction with PCA. Bottom - K-Means in conjunction with DAE

## Acknowledgments

I am sincerely grateful to Dr Karthik Dantu and Vaishali Maheshkar, whose invaluable guidance and insights greatly contributed to the completion of this independent study. Their mentorship and expertise were instrumental in shaping my approach and refining the methodologies

## References


- Vincent, Pascal, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of machine learning research 11, no. 12 (2010).
- Liang, Ming, and Xiaolin Hu. "Recurrent convolutional neural network for object recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3367-3375. 2015.
- Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

- Liu, Zhuang, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. "A convnet for the 2020s." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11976-11986. 2022.
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://xgboost.readthedocs.io/en/stable/get _started.html
- https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html