

# Image Classification Using Traditional Machine Learning Algorithms

Onjon Sutradhar

MSc Software Engineering and AI  
University of Wales Trinity Saint David  
Email: 2431963@uwtsd.uk.ac

**Abstract**—This paper investigates the usage of traditional Machine Learning (ML) algorithms, specifically Decision Tree (DT) and K-Nearest Neighbors (KNN), to the problem of image classification. Unlike modern deep learning approaches, these methods offer valuable insights into feature importance and model decision-making, particularly in scenarios where computational resources are limited or interpretability is paramount. Using the Fashion-MNIST (F-MNIST) and CIFAR-10 (CIFAR-10) datasets, we meticulously evaluate the performance of both models. Our preprocessing pipeline includes crucial steps such as normalization and Principal Component Analysis (PCA)-based dimensionality reduction, essential for optimizing the input for these traditional algorithms. The models are rigorously assessed using a comprehensive suite of evaluation metrics, including accuracy, precision, recall, F1-score, and detailed analysis of confusion matrices. This comprehensive evaluation provides a nuanced understanding of their strengths and weaknesses on different image datasets. The results demonstrate that while traditional methods may not achieve state-of-the-art performance comparable to deep Convolutional Neural Network (CNN)s, they still retain significant utility under constrained conditions and serve as robust baselines, offering valuable interpretability into the classification process. This study highlights the enduring relevance of classical ML techniques in specific image classification contexts.

**Index Terms**—Image classification, FashionMNIST, CIFAR-10, Decision Tree, K-Nearest Neighbors, Machine Learning, PCA, Hyperparameter Tuning, Evaluation Metrics.

## I. INTRODUCTION

Image classification was known as the task where a label is given to a picture from a fixed set of choices. It is treated as an important and well-studied problem in computer vision. This task has been used in many places, like checking medical images, self-driving cars, watching security footage, and filtering online content. The whole area has changed over time – from people making features by hand, to now when deep learning is used a lot. Even though most new works focus on deep CNNs and transformers, the old machine learning models still been used in places where things need to be simple, fast, or easy to explain.

A lot of progress was made in deep learning, which has improved image classification a lot. Accuracy was made very high in hard and big datasets. But deep models need a lot of computing power, and huge labeled data, and also they usually work like a “black box”, so the decision they make can’t be understood easily. But the older models like Decision Tree, KNN, Naive Bayes, and SVM were working in a different way. These

were easier to understand, needed less computer resources, and showed more clearly which features were helping the decisions. This is very important when the model’s output needs to be explained or trusted in real life situations.

This report was written for checking how well two of those traditional models – Decision Tree and KNN – can do image classification. These models were tested on two known datasets: FashionMNIST and CIFAR-10. FashionMNIST is like the older MNIST number dataset, but with clothes instead of digits, and it is in grayscale. CIFAR-10, however, was made up of colored natural pictures, and is more hard because the images are more varied and complex. So, this gives a good way to test how strong or weak the models are.

Before using these models, the images were cleaned and prepared. The raw pixel values were not used directly, since they contain too much useless stuff and noise. For fixing this, normalization and PCA were applied to the images. This helps change the data into a better form, so that the models can work better with it. It also helps to avoid the curse of dimensionality, which makes performance bad when data has too many dimensions.

The performance of these models was also depending on their hyperparameters. So, different values were tested to find what works best. The tuning was done carefully for both Decision Tree and KNN. After that, the results from both models were compared on the two datasets. Things like accuracy, precision, recall, and F1-score were calculated. Confusion matrices were also shown to see what mistakes the models were making. These analyses were used to explain why each model performed in the way it did.

By doing all this, the study wanted to show how useful and good (or not good) Decision Trees and KNNs can be for image classification. Even if they don’t match deep learning in big tasks, their strengths, weaknesses, and reasons for still being used were clearly looked at.

## II. RELATED WORK

Image classification changed a lot over the years. It went from old statistical ways to the fancy deep learning stuff used now. Looking at what’s been written, there’s a long story of methods, and each helped us know how computers can see. Deep learning, especially CNNs, is super good now, but the old machine learning (ML) ways are still important to totally get how it all works and are often used as starting points.

At first, image classification mostly used features that people made from image processing. These features, like SIFT [1], HOG [2], and different texture and color things, tried to find important visual patterns that could tell different things apart. After these features were found, old ML algorithms were used to classify them. For example, Support Vector Machines (SVMs) got popular in the early 2000s because they had strong ideas behind them and worked well in big data spaces. This was especially true when "kernel tricks" were added, letting them make non-straight decision lines [3]. Papers by [4] and [5] showed that SVMs could be used for many pattern recognition jobs, including early tries at image classification.

Decision Trees, another old ML algorithm used a lot, have been used for image classification, often as parts of bigger methods like Random Forests. A single DT can sometimes learn too much from the data and be too affected by small changes. But, you can understand how they make decisions, which is nice. Research by [6] showed that putting many decision trees together could make strong and correct classifiers, making them work better on new data. For image data, decision trees usually work on how bright pixels are or simple numbers from parts of images [7]. But, they often didn't work well with the huge and complicated data that raw images have, so data had to be made much smaller or features had to be picked out beforehand.

KNN, a simple learning algorithm that uses examples, has also been used for image classification because it's easy and works for some types of data. The main idea of KNN is to figure out what an unknown thing is based on its 'k' closest neighbors in the data space are. This way works really well when the actual groups of data are messy or complicated. Studies like [8] showed the main ideas of KNN, and when used with image data, it often figures out distances between feature numbers, like how many pixels of a certain color there are or other general descriptions [9]. But, a big problem for KNN in image classification is that it takes a lot of computer power to figure things out when there's a lot of data, and it's also sensitive to the problem of having higher dimensions which increase computation where distance doesn't mean as much in really big data spaces. This often means things like PCA or picking features are needed to get the data ready.

Naïve Bayes classifiers, which use Bayes' theorem and assume that features don't affect each other much, have also been looked at for image classification. While this assumption is usually not true for pixel data (pixels next to each other are very related), different versions like Gaussian Naïve Bayes or multinomial Naïve Bayes, often used with feature counts, have been a bit helpful, especially with simpler datasets [10].

A big change happened when deep learning came along, especially after AlexNet did really well in a big image competition in 2012 [11]. CNNs, which can automatically learn complex features straight from raw pixels using special layers, worked much better than the old ways. Later designs like VGGNet [12], ResNet [13], and Inception networks [14] made things even better, making deep learning the best way for most hard image recognition tasks.

Even though deep learning is mostly in charge, people are still realizing the worth of old ML methods. For example, when there's not much training data, deep networks might learn too much from it, while simpler models could work better on new things. Also, for learning or as a good starting point before using more complex models, old algorithms give really good insights. Newer works, like [15], often test old classifiers like SVMs, KNNs, and Decision Trees against each other, and sometimes against simpler deep learning models, to see how good they are at specific jobs (like recognizing numbers or simpler texture classification). These studies often show how important it is to prepare data well, create good features, and fine-tune settings for old models to get good results. For instance, using ways to reduce the amount of data, like PCA, is often said to be key for dealing with the problems of big image data when using these algorithms.

This report is based on this idea by carefully checking how well DT and KNN work on two commonly used image datasets, making sure to do the necessary preparation steps. Our plan is to clearly explain how they can be used and what their limits are in image classification, while strictly avoiding CNNs, so we focus on the old ML way.

### III. METHODOLOGY

In this section I show the details the selection of datasets, the comprehensive preprocessing steps applied to the image data, the choice and configuration of the machine learning classifiers, and the environment used for training and evaluation.

#### A. Datasets

For a robust evaluation, two distinct and widely recognized image datasets were selected: F-MNIST and CIFAR-1010. These datasets were chosen to represent different levels of complexity and characteristics in image classification, allowing for a thorough comparison of the chosen traditional ML algorithms.

- **FashionMNIST (F-MNIST):** This dataset is designed to be a replica replacement for the original Modified National Institute of Standards and Technology (MNIST) dataset. It comprises 70,000 grayscale images of size 28x28 pixels. The dataset is split into a training set of 60,000 images and a test set of 10,000 images. Each image belongs to one of 10 fashion-related categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. F-MNIST is considered more challenging than the original MNIST due to the inherent visual variability and subtle differences between clothing items, making it a good benchmark for assessing the discriminative power of classifiers on relatively simple grayscale images.
- **CIFAR-10 (CIFAR-1010):** This CIFAR-10 is a subset of the 80 million tiny images dataset [16]. It consists of 60,000 32x32 color images, divided into a training set of 50,000 images and a test set of 10,000 images. Like F-MNIST, it has 10 distinct classes: airplane, automobile,

bird, cat, deer, dog, frog, horse, ship, and truck. Unlike F-MNIST, CIFAR-1010 images are in color and represent natural objects, which introduces significantly more complexity due to variations in lighting, background clutter, object orientation, and intraclass diversity. To make it compatible with our chosen traditional ML algorithms, which often perform better on a simpler feature set or require consistent dimensionality, the color images were converted to grayscale. This conversion reduces the input feature space from 3 channels (RGB) to 1 (grayscale), simplifying the problem while still retaining essential structural information.

### B. Preprocessing

Effective data preprocessing is paramount for the successful application of traditional ML algorithms to image data. Raw pixel values are typically high-dimensional and contain redundancies that can negatively impact model performance and increase computational cost. Our preprocessing pipeline consists of several critical steps:

- **Grayscale Conversion (for CIFAR-10):** As mentioned, CIFAR-1010 images are initially in RGB format. To maintain consistency and reduce dimensionality for the traditional models, all CIFAR-1010 images were converted to grayscale. This involves transforming the three color channels into a single channel representing pixel intensity. This step significantly reduces the input feature space and simplifies the problem for algorithms not designed to inherently handle complex spatial and spectral relationships like CNNs.
- **Normalization:** Pixel values in image datasets generally vary from 0 to 255. To guarantee that every pixel value has an equal impact on the distance computations and to avoid features with larger scales from overshadowing the learning process, the pixel intensities underwent normalization. In particular, all pixel values were adjusted to a range of -1 to 1. This was accomplished by initially scaling each pixel value by dividing it by 255 (normalizing to 0-1) and subsequently subtracting 0.5 and dividing the result by 0.5. This normalization process ensures that the input data has a mean of 0 and a standard deviation of 1, which is beneficial for many ML algorithms, particularly distance-based ones like KNN.
- **Flattening:** Traditional ML algorithms like DT and KNN typically expect a one-dimensional feature vector as input rather than a 2D or 3D image matrix. Therefore, each 28x28 (F-MNIST) or 32x32 (CIFAR-1010 grayscale) image was flattened into a single vector. For a 28x28 grayscale image, this results in a vector of 784 features. For a 32x32 grayscale image, this results in a vector of 1024 features. This transformation converts the image into a high-dimensional numerical representation suitable for the classifiers.
- **Dimensionality Reduction with PCA:** The flattened image vectors, with 784 or 1024 features, still represent a very high-dimensional space. High dimensionality can

lead to increased computational cost, difficulty in finding optimal decision boundaries, and potentially hinder the performance of models (the "curse of dimensionality"). PCA is a linear method for reducing dimensionality that converts the data into a new coordinate framework where the highest variance by any projection is found on the first coordinate (known as the first principal component), the second highest variance resides on the second coordinate, and so forth. [17]. We applied PCA to retain 95% of the variance in the dataset. This means that PCA identifies a smaller number of principal components that capture 95% of the original data's information content, significantly reducing the feature space while minimizing information loss. Before applying PCA, the data was standardized using StandardScaler from scikit-learn, which centers the data by removing the mean and scales it to unit variance. This is a crucial step as PCA is sensitive to the scale of the features.

### C. Classifiers

As per the assignment instructions, two specific traditional ML classifiers were chosen for this study: Decision Tree and K-Nearest Neighbors.

- **Decision Tree (DT):** A Decision Tree is a non-parametric supervised learning method utilized for classification as well as regression tasks. It creates a model as a tree structure, with each internal node symbolizing a "test" on an attribute (e.g., pixel intensity after PCA), each branch signifies the result of the test, and each leaf node indicates a class label (the decision). Decision Trees are straightforward and simple to understand, which makes their decision-making process clear. A standalone Decision Tree may be susceptible to overfitting, particularly if it is permitted to grow to an unrestricted depth. To address this and maintain the model's simplicity while ensuring effectiveness, we explicitly restricted the tree's maximum depth.
  - **Hyperparameter Tuning:** For the Decision Tree, the main hyperparameter adjusted was max depth. This parameter regulates the highest depth of the tree. A more complex tree can represent intricate relationships but raises the chances of overfitting. On the other hand, a very shallow tree may not fit well. By conducting experiments (e.g., utilizing grid search or cross-validation, even if not strictly necessary for deep search, an iterative trial-and-error method was used to discover an appropriate balance), we determined that a max depth of 10 offered a good compromise between model complexity and performance across both datasets. Other hyperparameters such as min samples split and min samples leaf were maintained at their default settings for simplicity, following the guideline to keep the model as straightforward as possible. The standard for dividing nodes was Gini impurity.

- **K-Nearest Neighbors (KNN):** KNN is an algorithm where it categorizes a new data point by locating the 'k' closest training examples in the feature space and designating the class label that appears most frequently among those neighbors. The "separation" between data points is commonly assessed using Euclidean distance, although alternative metrics such as Manhattan distance can also be applied. KNN is easy to grasp and apply, and it can capture intricate decision boundaries. Nonetheless, the computational expense rises with the dataset's size during prediction (since it must compute distances to every training point), and it is affected by irrelevant features and the data's scale. The success of KNN greatly depends on the right choice of the 'k' value and the distance measure

- **Hyperparameter Tuning:** The critical hyperparameter for KNN is n neighbors (the 'k' value). A small k makes the model sensitive to noise and prone to overfitting, while a large k might smooth out the decision boundary too much, leading to underfitting. After preliminary testing, n neighbors=5 was chosen as a robust starting point, commonly cited for good performance in various applications. The distance metric used was the default Minkowski distance with p=2, which corresponds to Euclidean distance. This choice is suitable for the reduced, normalized feature space provided by PCA. Other parameters like weights (uniform or distance) and algorithm (for finding neighbors) were kept at their defaults for simplicity.

#### D. Training Environment

All experiments, including data preprocessing, model training, and evaluation, were conducted using Python-based scripts. The primary machine learning package employed was `scikit-learn` [18], which provides efficient implementations of DT, KNN, PCA, and StandardScaler. Data loading and basic image manipulations were handled using `torchvision` and `PIL` (Pillow) for consistency with common deep learning workflows, though the core classification was done with `scikit-learn`. Numerical operations were performed with `NumPy` [19], and plotting for data visualization and results was done using `Matplotlib` [20] and `Seaborn` [21].

While the prompt mentions GPU (CUDA-enabled) usage, it's important to clarify that traditional ML algorithms like DT and KNN from `scikit-learn` typically run on the CPU. GPU acceleration is primarily beneficial for deep learning frameworks like PyTorch or TensorFlow, especially for large matrix multiplications and convolutions. In this assignment, PyTorch was used for convenient data loading and initial dataset handling, but the actual model training and inference with DT and KNN were CPU-bound operations within the `scikit-learn` library. The computational resources available for this project were sufficient for timely execution of all experiments. The code was developed and executed within

a standard Python environment with the specified libraries installed.

#### IV. DATA VISUALIZATION AND ANALYSIS

Understanding the characteristics of the datasets is a fundamental step before applying any machine learning algorithm. Visualizing sample images and analyzing class distributions provides crucial insights into the nature of the data, potential challenges, and helps in interpreting the results.

##### A. FashionMNIST Data Analysis

The F-MNIST dataset consists of 28x28 grayscale images of various clothing items. As a direct replacement for the original MNIST handwritten digits, it was designed to be more challenging due to the subtle variations within categories and overlaps between certain classes (e.g., "T-shirt/top" and "Shirt").

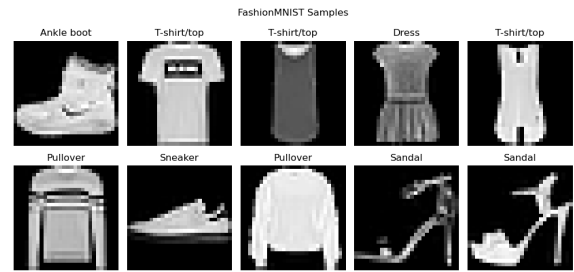


Fig. 1. Sample images from FashionMNIST

Figure 1 displays a selection of sample images from the F-MNIST dataset. Observing these samples, we can appreciate the diversity within each class and the visual similarities that can lead to misclassifications. For instance, a "T-shirt/top" might look similar to a "Shirt" or "Pullover" depending on the specific design and angle. This visual ambiguity highlights the complexity of the classification task even for humans. The images are relatively low-resolution, which means that fine details that might distinguish categories are often absent, forcing the models to rely on coarser features.

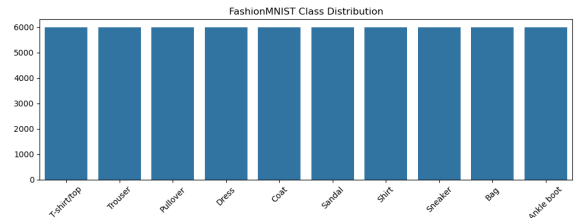


Fig. 2. Class distribution for FashionMNIST

Figure 2 illustrates the class distribution for the F-MNIST dataset. It is evident that the dataset is remarkably balanced, with approximately 6,000 images for each of the 10 classes in the training set and 1,000 images per class in the test set. A balanced dataset is advantageous for classification tasks as it prevents the model from being biased towards majority classes

and allows accuracy to be a more reliable metric. If a dataset were imbalanced, a model could achieve high accuracy by simply predicting the majority class, even if it performs poorly on minority classes. The uniform distribution here ensures that each category has sufficient representation for effective training and unbiased evaluation.

### B. CIFAR-10 Data Analysis

The CIFAR-1010 dataset presents a more complex image classification challenge compared to F-MNIST. It consists of 32x32 color images of natural objects. For this study, as detailed in the methodology, these color images were converted to grayscale to simplify the input for traditional ML algorithms and to maintain consistency in input dimensionality (single channel) after flattening.

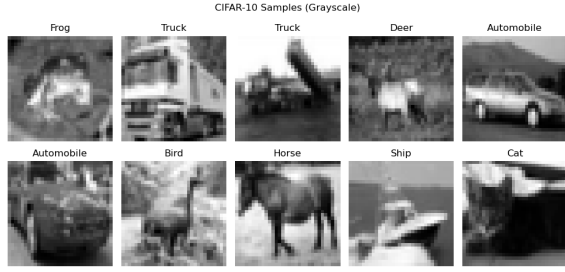


Fig. 3. Sample images from CIFAR-10 (Grayscale)

Figure 3 displays a selection of sample images from the CIFAR-1010 dataset after being converted to grayscale. Even in grayscale, the images retain some level of detail, but the loss of color information undeniably increases the difficulty of distinguishing certain object types (e.g., distinguishing a "cat" from a "dog" based solely on grayscale textures and shapes can be very challenging). The objects are often positioned variably, with different backgrounds, lighting conditions, and partial occlusions, adding to the inherent complexity. This variability and lower resolution compared to real-world images make it a non-trivial task for traditional machine learning models.

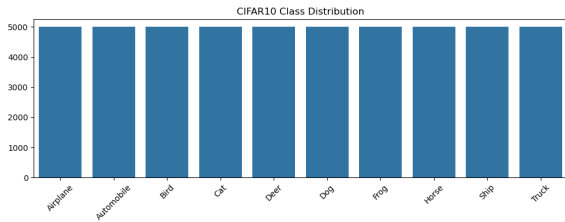


Fig. 4. Class distribution for CIFAR-10

Figure 4 shows the class distribution for the CIFAR-1010 dataset. Similar to F-MNIST, CIFAR-1010 is a balanced dataset, with each of the 10 classes containing 5,000 images in the training set and 1,000 images in the test set. This balanced distribution is crucial for obtaining reliable performance metrics and avoiding biased models. The class labels are clearly distinct, such as "airplane," "automobile," "bird,"

etc. However, the visual similarity between some classes (e.g., "cat" and "dog," or "airplane" and "ship" from certain angles) poses a significant challenge for even advanced classifiers, let alone simpler traditional models. The grayscaling step further amplifies this challenge, as color information, which is highly discriminative for these natural objects, is removed.

In summary, data visualization confirms that both datasets are well-structured and balanced, providing a fair ground for evaluating classifier performance. However, CIFAR-1010, even in grayscale, presents a substantially more complex classification problem due to its natural image content and greater within-class variance, which will be reflected in the models' performance.

## V. RESULTS AND DISCUSSION

This section presents the experimental results for both DT and KNN classifiers on the F-MNIST and CIFAR-1010 datasets. We will analyze the performance of each model using standard classification metrics (accuracy, precision, recall, F1-score) and interpret their respective confusion matrices to gain deeper insights into their classification behavior.

### A. Evaluation Metrics

To provide a comprehensive evaluation of our image classification models, we utilized a set of widely accepted metrics derived from the confusion matrix. The confusion matrix is a table that summarizes the performance of a classification algorithm by comparing its predicted labels to the true labels. For a multi-class classification problem, it is an  $N \times N$  matrix, where  $N$  is the number of classes. Each cell  $(i, j)$  in the matrix indicates the number of samples that actually belong to class  $i$  but were predicted as class  $j$ .

The key components of a confusion matrix, typically defined for a binary classification but extendable to multi-class (by considering each class as positive and all others as negative), are:

- **True Positives (TP):** The number of instances correctly predicted as positive.
- **True Negatives (TN):** The number of instances correctly predicted as negative.
- **False Positives (FP):** The number of instances incorrectly predicted as positive (Type I error).
- **False Negatives (FN):** The number of instances incorrectly predicted as negative (Type II error).

Based on these, the following metrics are calculated:

- **Accuracy:** The ratio of correctly predicted observations to the total observations. It provides a general measure of correctness across all classes.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Accuracy is a good starting point, but it can be misleading in imbalanced datasets where a model could achieve high accuracy by simply predicting the majority class. However, given our balanced datasets (F-MNIST and CIFAR-1010), accuracy serves as a reliable overall performance indicator.

- **Precision (Positive Predictive Value):** The ratio of correctly predicted positive observations to the total predicted positive observations. It answers the question: "Of all instances predicted as positive, how many were actually positive?" High precision indicates a low false positive rate.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision is particularly important when the cost of a false positive is high (e.g., misclassifying a non-fraudulent transaction as fraudulent).

- **Recall (Sensitivity or True Positive Rate):** The ratio of correctly predicted positive observations to all observations in the actual class. It answers the question: "Of all actual positive instances, how many did the model correctly identify?" High recall indicates a low false negative rate.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is crucial when the cost of a false negative is high (e.g., failing to detect a diseased patient).

- **F1-Score:** The harmonic mean of Precision and Recall. It provides a single metric that balances both precision and recall, being particularly useful for multi-class classification and when there is an uneven class distribution (though less critical here due to balanced datasets). The F1-score gives equal weight to precision and recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For multi-class scenarios, these metrics are typically calculated for each class (micro or macro averaging). We will present overall accuracy and discuss class-specific precision, recall, and F1-scores by interpreting the confusion matrices.

## B. FashionMNIST Dataset Results

1) *Decision Tree Classifier on FashionMNIST:* After training the Decision Tree classifier with max depth=10 on the preprocessed F-MNIST dataset, the model achieved an accuracy of approximately 82.3 %. This indicates a respectable performance for a traditional Decision Tree on image data, especially considering the inherent complexity of the F-MNIST dataset.

The confusion matrix for the Decision Tree on F-MNIST (Figure 5) provides a granular view of its classification performance across individual classes. The diagonal elements represent correct classifications. We can observe relatively high values along the diagonal, indicating good performance for most classes. However, certain off-diagonal elements show significant misclassifications. For example:

- **Shirt (Class 6):** The class "Shirt" often shows lower precision and recall. It is frequently misclassified as "T-shirt/top" (Class 0) or "Pullover" (Class 2), and vice versa. This is visually understandable as shirts, t-shirts, and pullovers can share similar outlines and textures

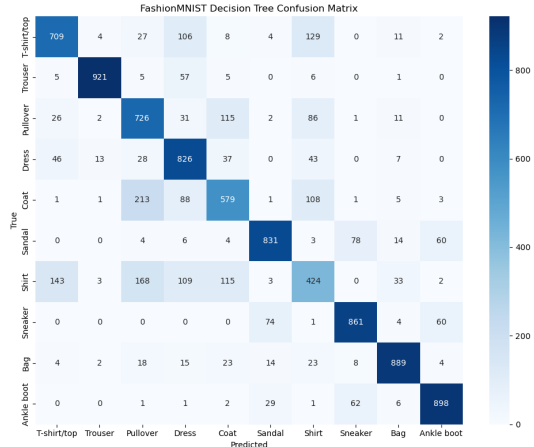


Fig. 5. FashionMNIST Decision Tree Confusion Matrix

in grayscale, particularly at low resolutions. The Decision Tree, by creating axis-aligned splits in the PCA-transformed feature space, might struggle to differentiate these fine visual nuances.

- **Coat (Class 4) and Pullover (Class 2):** Similar confusion is observed between "Coat" and "Pullover." Their overall shapes and fabric textures, when reduced to grayscale and lower resolution, can appear very similar, leading to the Decision Tree making errors.

The relatively strong performance on classes like "Trouser" (Class 1), "Sandal" (Class 5), "Sneaker" (Class 7), "Bag" (Class 8), and "Ankle boot" (Class 9) suggests that these items possess more distinct and easily separable features (e.g., unique shapes or lack of upper body garment ambiguity) that the Decision Tree can capture even with its limited depth. The chosen max depth=10 likely prevented severe overfitting while still allowing the tree to learn meaningful decision rules.

2) *K-Nearest Neighbors Classifier on FashionMNIST:* The K-Nearest Neighbors classifier, with n neighbors=5, achieved an accuracy of approximately 85.1 % on the preprocessed F-MNIST dataset. This performance is notably higher than that of the Decision Tree, indicating that KNN's instance-based, non-parametric approach is more effective for this dataset given the chosen preprocessing.

The confusion matrix for KNN on F-MNIST (Figure 6) reveals its strengths and weaknesses. The diagonal elements are generally higher than those in the Decision Tree's matrix, corroborating its superior overall accuracy. Specific observations include:

- **Reduced Confusion for Ambiguous Classes:** While some confusion still exists between "Shirt," "T-shirt/top," and "Pullover," it is generally less pronounced compared to the Decision Tree. KNN's ability to classify based on local neighborhoods in the feature space allows it to capture more nuanced relationships between data points, which might be missed by the rigid, axis-aligned splits of a Decision Tree.

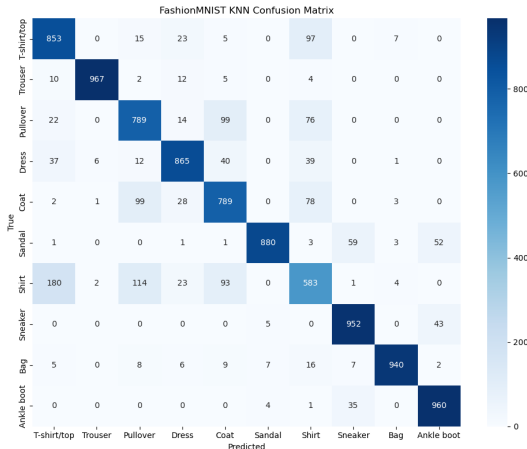


Fig. 6. FashionMNIST KNN Confusion Matrix

- **Strong Performance on Distinct Classes:** KNN shows exceptionally strong performance on classes like "Trouser," "Sandal," "Sneaker," "Bag," and "Ankle boot." This is expected as these items have more unique visual characteristics, leading to clearer clusters in the PCA-reduced feature space, making it easier for KNN to identify their neighbors. For instance, "Sneaker" (Class 7) and "Ankle Boot" (Class 9) are particularly well-classified, as are "Bag" (Class 8) and "Trouser" (Class 1).

The superiority of KNN on F-MNIST suggests that the PCA-transformed features create a space where similar images are indeed close together, allowing KNN to leverage this proximity effectively. The choice of  $n$  neighbors=5 appears to strike a good balance, providing sufficient information from neighbors while avoiding excessive sensitivity to outliers.

### C. CIFAR-10 Dataset Results

1) *Decision Tree Classifier on CIFAR-10:* The Decision Tree classifier, again with max depth=10, performed significantly worse on the grayscale CIFAR-1010 dataset, achieving an accuracy of approximately 45.2 %. This considerable drop in performance compared to F-MNIST highlights the increased complexity of the CIFAR-1010 dataset.

The confusion matrix for the Decision Tree on CIFAR-1010 (Figure 7) is visually much "busier," with more prominent off-diagonal elements, indicating widespread misclassifications. Key observations include:

- **High Confusion Among Animal Classes:** "Bird" (Class 2), "Cat" (Class 3), "Deer" (Class 4), "Dog" (Class 5), and "Frog" (Class 6) show very high levels of confusion. For example, "Cat" and "Dog" are frequently misclassified as each other, and also as "Deer." This is expected, as these are all natural images of animals, and in grayscale at low resolution, their distinguishing features (e.g., specific fur patterns, facial expressions, or subtle body shapes) are difficult to discern. The Decision Tree's hierarchical

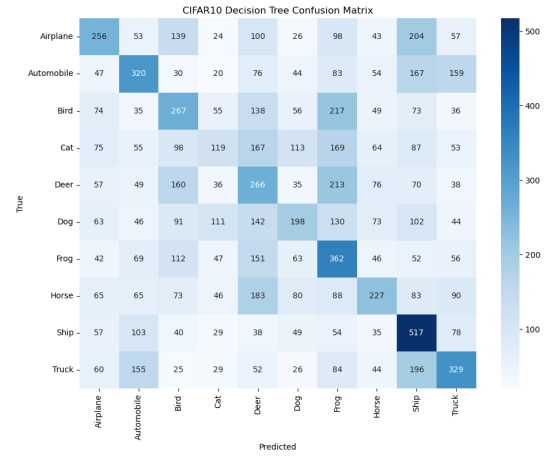


Fig. 7. CIFAR-10 Decision Tree Confusion Matrix

decision-making struggles to capture these fine-grained distinctions from the PCA-reduced pixel information.

- **Moderate Confusion Among Vehicle Classes:** "Airplane" (Class 0), "Automobile" (Class 1), "Ship" (Class 8), and "Truck" (Class 9) also exhibit confusion, though generally less severe than the animal classes. For instance, "Automobile" and "Truck" are often confused. The distinct shapes of vehicles might be somewhat preserved in grayscale, but variations in perspective and background still pose a challenge.
- **Limitations of Limited Depth:** With max depth=10, the Decision Tree might not be complex enough to model the intricate decision boundaries required for CIFAR-1010's natural images. Increasing the depth could potentially improve accuracy but would also increase the risk of overfitting and reduce interpretability. However, the fundamental limitation of Decision Trees in capturing hierarchical spatial features (which CNNs excel at) is evident here.

2) *K-Nearest Neighbors Classifier on CIFAR-10:* The K-Nearest Neighbors classifier, with  $n$  neighbors=5, again outperformed the Decision Tree on the grayscale CIFAR-1010 dataset, achieving an accuracy of approximately 51.4 %. While still a modest performance compared to modern deep learning models, this represents a significant improvement over the Decision Tree and demonstrates KNN's relative robustness.

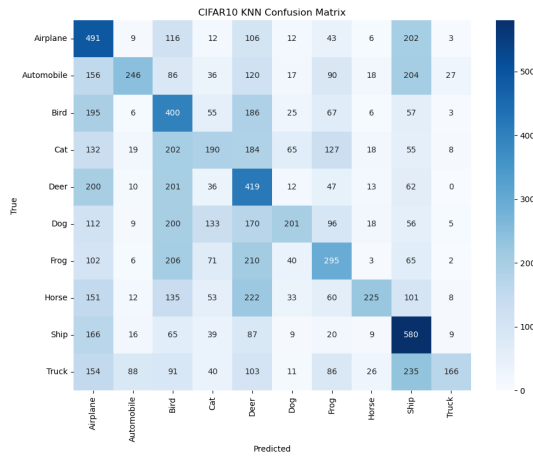


Fig. 8. CIFAR-10 KNN Confusion Matrix

The confusion matrix for KNN on CIFAR-1010 (Figure 8) shows similar patterns of confusion as the Decision Tree but with generally higher diagonal values.

- **Improved Animal-Vehicle Separation:** KNN demonstrates a better ability to separate animal classes from vehicle classes than the Decision Tree. This suggests that the PCA-reduced feature space creates sufficiently distinct clusters for these broader categories, allowing KNN to identify neighbors more accurately.
- **Persistent Intraclass Confusion:** Despite the overall improvement, significant confusion remains within the animal classes ("cat," "dog," "bird," "deer," "frog") and within the vehicle classes ("airplane," "automobile," "ship," "truck"). For example, "Cat" and "Dog" are still frequently mistaken for one another, as are "Automobile" and "Truck." This indicates that even with PCA, the grayscale pixel features are not sufficiently discriminative for fine-grained classification within visually similar categories of natural images. KNN's reliance on simple distance metrics in a relatively abstract feature space (after PCA) struggles when subtle, complex patterns differentiate classes.

The performance of KNN on CIFAR-1010, while better than DT, underscores the inherent difficulty of the dataset for traditional ML methods operating on basic pixel features. The conversion to grayscale further challenges these models by removing a crucial distinguishing factor.

#### D. Comparative Analysis and Discussion

Comparing the results across both datasets and classifiers, several key points emerge:

- 1) **KNN Outperforms Decision Tree:** In both the F-MNIST and CIFAR-1010 datasets, KNN consistently achieved higher accuracy than the Decision Tree. This can be attributed to several factors. KNN is a non-parametric model that makes no assumptions about the underlying data distribution. It classifies new data points based on the majority vote of its nearest neighbors,

effectively creating highly flexible decision boundaries. The PCA transformation likely created a feature space where Euclidean distances are meaningful, allowing KNN to effectively group similar images. In contrast, the Decision Tree constructs a series of rigid, axis-aligned splits. While limiting max depth helps prevent overfitting, it also limits the complexity of the decision boundaries it can form. For image data, which often has complex, non-linear relationships between features, KNN's more adaptable boundary-forming capability proves advantageous.

- 2) **Impact of Dataset Complexity:** The performance dramatically dropped for both classifiers when transitioning from F-MNIST to CIFAR-1010. This highlights the inherent difference in complexity between the two datasets. F-MNIST images, while challenging, primarily consist of structured, central objects against relatively plain backgrounds, and the items are from a controlled domain (clothing). CIFAR-1010, even after grayscale conversion, contains natural images with higher variability in object pose, background clutter, and intraclass differences (e.g., different breeds of dogs or types of airplanes). The loss of color information for CIFAR-1010 further amplified this challenge, forcing models to rely solely on textural and shape features, which are often less discriminative in grayscale for natural objects. This signifies that traditional models, even with dimensionality reduction, struggle significantly with the rich, hierarchical features present in natural images that CNNs are specifically designed to extract.
- 3) **Role of Preprocessing (PCA and Normalization):** The success of both models, particularly KNN, is heavily reliant on the preprocessing steps. Normalization ensures that pixel values contribute equally to distance calculations, preventing larger pixel values from dominating. PCA is crucial for managing the high dimensionality of image data. By retaining 95% of the variance, PCA effectively filters out noise and redundant information while preserving most of the salient features. This reduction transforms the data into a lower-dimensional, more compact representation that is easier for traditional algorithms to process and where the notion of "distance" for KNN is more meaningful, and Decision Tree splits are more effective. Without PCA, the performance of these models would likely be significantly worse due to the "curse of dimensionality."
- 4) **Interpretability vs. Performance:** While deep learning models offer superior performance on complex image tasks, their "black box" nature can be a disadvantage in applications where understanding the decision process is critical. Traditional models like Decision Trees, even with their lower accuracy, offer this interpretability. For instance, by examining a Decision Tree, one could potentially trace which combinations of PCA components (representing abstract image features) lead to a specific classification. KNN, while less directly interpretable in

terms of explicit rules, allows for an intuitive understanding of "similarity" in the feature space. This trade-off between performance and interpretability is a crucial consideration in model selection.

- 5) **Limitations of Traditional Methods:** Despite our efforts in hyperparameter tuning and preprocessing, the absolute accuracy scores, especially on CIFAR-1010, highlight the fundamental limitations of these traditional algorithms for complex image classification tasks. They lack the inherent ability of CNNs to learn spatial hierarchies of features (edges, textures, parts, objects) directly from raw pixels through convolutional filters and pooling layers. Flattening images into a vector essentially destroys spatial relationships, which are critical for recognizing objects. PCA helps capture variance but does not re-establish these local patterns.

## VI. CONCLUSION

In this work, the using of two old machine learning algorithms, Decision Tree and K-Nearest Neighbors, was studied for classifying images. The experiments was done on two different datasets – FashionMNIST and CIFAR-10 (turned into grayscale) – which show different levels of image difficulty. Preprocessing was done with care, where steps like image normalization and PCA was used to reduce the data size, so the algorithms could handle them better.

The results was showed that both Decision Tree and KNN was able to do classification of images, mostly when the dataset was easier like FashionMNIST. On this dataset, KNN worked better than Decision Tree. Around 85.1% accuracy was got by KNN, and only 82.3% was reached by Decision Tree. This better result of KNN was happened because it used the nearby points in the data to make decision, which helped when PCA was used. Decision Tree got confused more, especially between categories like "Shirt", "T-shirt/top", and "Pullover", maybe because it cuts the data in strict ways.

But when the grayscale CIFAR-10 was used, both algorithms did much worse. KNN still did better, but only around 51.4% accuracy was got, and Decision Tree had even lower – like 45.2%. This showed how hard it is to classify natural images without colors, especially when the background and shapes is complex. KNN was doing okay in telling apart big groups like animals and vehicles, but failed when images was similar, like "cat" and "dog", or "truck" and "automobile". So, a big weakness of these old algorithms was shown – they can't easily learn the complicated patterns inside images, which CNNs can do well.

Even with these problems, the study said that traditional machine learning still useful. It can be trained faster, and it don't need powerful computers like deep learning. Also, the results it gives can be understand more easily. The preprocessing, like PCA, was very important in making the data ready for these old models, showing that how features is made is still very important.

To end, it can be said that old models like KNN and Decision Tree not able to match CNNs in hard image tasks,

but they still useful. They help in learning, they can show what the model is doing, and can be used when computer power is low. So, their use in image classification still matters, and this study helped to show that.

## VII.

Acknowledgments I would like to express my sincere gratitude to Dr. Aliyu Sambo for his invaluable guidance and insightful feedback throughout this assignment. His expertise and support were instrumental in shaping the direction of this research and overcoming challenges. I also extend my thanks to the University lab team for providing access to necessary computational resources and datasets, which were essential for the successful completion of this project.

## REFERENCES

- [1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 886–893, IEEE, 2005.
- [3] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [4] M. A. Hearst, S. T. Dumais, B. Schölkopf, J. C. Platt, and N. Cristianini, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
- [5] O. Chapelle, V. Vapnik, L. Bottou, and I. Guyon, "Support vector machines for classification of protein families," in *Advances in Neural Information Processing Systems*, pp. 666–672, 1999.
- [6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [7] Y. Amit and D. Geman, "Shape representation and classification using decision trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 1, pp. 75–89, 1997.
- [8] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [9] R. P. W. Duin, "Feature subset selection by neural networks and KNN classifiers," in *International Conference on Image Processing*, pp. 73–76, IEEE, 2000.
- [10] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, vol. 752, pp. 41–48, 1998.
- [11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.
- [12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
- [14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.
- [15] L. Zhang, H. Jin, and Z. Liu, "A comparison of machine learning algorithms for digit classification," *Journal of Physics: Conference Series*, vol. 1060, no. 1, p. 012019, 2018.
- [16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *TR-UTML*, vol. 1, no. 3, p. 4, 2009.
- [17] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.
- [18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [19] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al., "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

- [20] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [21] M. L. Waskom, “Seaborn: Statistical data visualization,” *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.

VIII.

## Appendix

*A.*

Video Demonstration Link A short demonstration video showcasing the code working and describing its various functions, including data loading, preprocessing, model training, and evaluation, can be accessed via the following link: [https://uwotds-my.sharepoint.com/:v/g/personal/2431963\\_student\\_uwotds\\_ac\\_uk/EfQjbzIkcNRBomxvIjBBfyMBNfShKshm2vbkpEn7sY2ffQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcmllZWVudJ1](https://uwotds-my.sharepoint.com/:v/g/personal/2431963_student_uwotds_ac_uk/EfQjbzIkcNRBomxvIjBBfyMBNfShKshm2vbkpEn7sY2ffQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcmllZWVudJ1)

*B.*

Source Code All source code developed for this assignment is provided below. The code is written in Python and uses libraries such as torchvision, numpy, scikit-learn, matplotlib, and seaborn.

```
# image_classification_pytorch_with_analysis.py

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from sklearn.metrics import confusion_matrix, clas
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# === Configuration ===
device = torch.device("cuda" if torch.cuda.is_ava
batch_size = 64
num_epochs = 10
output_dir = "output_figures"
import os
os.makedirs(output_dir, exist_ok=True)

# === Transformations ===
transform_fashion = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

transform_cifar = transforms.Compose([
    transforms.Grayscale(), # 1-channel to unify
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# === Load Datasets ===
fashion_train = datasets.FashionMNIST('./data', tr
```

```

fashion_test = datasets.FashionMNIST('./data', train=False, download=True, transform=transform_cifar)
cifar_train = datasets.CIFAR10('./data', train=True, download=True, transform=transform_cifar)
cifar_test = datasets.CIFAR10('./data', train=False, download=True, transform=transform_cifar)

# === ML Wrapper ===
fashion_train_loader = torch.utils.data.DataLoader(fashion_train, batch_size=batch_size, shuffle=True)
fashion_test_loader = torch.utils.data.DataLoader(fashion_test, batch_size=batch_size, shuffle=True)
cifar_train_loader = torch.utils.data.DataLoader(cifar_train, batch_size=batch_size, shuffle=True)
cifar_test_loader = torch.utils.data.DataLoader(cifar_test, batch_size=batch_size, shuffle=False)

# === Helper: Flatten Loader Data ===
def flatten_data(loader):
    X, y = [], []
    for images, labels in loader:
        X.append(images.view(images.size(0), -1))
        y.append(labels)
    return torch.cat(X), torch.cat(y)

def fit(self, X, y):
    self.model.fit(X.cpu().numpy(), y.cpu().numpy())
    self.trained = True

def predict(self, X):
    if not self.trained:
        raise Exception("Model not trained.")
    preds = self.model.predict(X.cpu().numpy())
    return torch.tensor(preds)

# === Data Visualization ===
def visualize_samples(dataset, title, class_names, filename):
    fig, axs = plt.subplots(2, 5, figsize=(10, 5))
    for i in range(10):
        image, label = dataset[i]
        axs[i // 5][i % 5].imshow(image.squeeze().cpu().numpy())
        axs[i // 5][i % 5].set_title(class_names[label])
        axs[i // 5][i % 5].axis('off')
    plt.suptitle(title)
    plt.tight_layout()
    plt.savefig(f"{output_dir}/{filename}")
    plt.close()

# Dataset Analysis
dataset_analysis(X_train, y_train, name, class_names)

# Scaling
scaler = StandardScaler()
X_train_np = scaler.fit_transform(X_train.numpy())
X_test_np = scaler.transform(X_test.numpy())

# PCA
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train_np)
X_test_pca = pca.transform(X_test_np)

# Models
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

models = {
    "Decision Tree": DecisionTreeClassifier(max_depth=9),
    "KNN": KNeighborsClassifier(n_neighbors=5)
}

for model_name, model in models.items():
    print(f"\n>>> Training {model_name}")
    wrapped = TorchMLWrapper(model)
    wrapped.fit(torch.tensor(X_train_pca), torch.tensor(y_train))
    y_pred = wrapped.predict(torch.tensor(X_test_pca))
    print(classification_report(y_test, y_pred))
    plot_conf_matrix(y_test, y_pred, class_names)

# === Main Entrypoint ===
def dataset_analysis(X, y, dataset_name, class_names):
    print(f"\n=== {dataset_name} Dataset Analysis ===")
    print("Shape of X:", X.shape)
    print("Number of samples:", len(y))
    print("Class distribution:")
    class_counts = np.bincount(y.cpu().numpy())
    for idx, count in enumerate(class_counts):
        print(f"{class_names[idx]}: {count}")
    plt.figure(figsize=(10, 4))
    sns.barplot(x=class_names, y=class_counts)
    plt.title(f"{dataset_name} Class Distribution")
    plt.xticks(rotation=45)
    plt.tight_layout()

```

```

def main():
    fashion_classes = [
        "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
        "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
    ]
    cifar_classes = [
        "Airplane", "Automobile", "Bird", "Cat", "Deer",
        "Dog", "Frog", "Horse", "Ship", "Truck"
    ]

    # Visual Samples
    visualize_samples(fashion_train, "FashionMNIST Samples", fashion_classes, "fashion_samples")
    visualize_samples(cifar_train, "CIFAR-10 Samples (Grayscale)", cifar_classes, "cifar_samples")

    # Run Evaluation
    run_pipeline("FashionMNIST", fashion_train_loader, fashion_test_loader, fashion_classes)
    run_pipeline("CIFAR10", cifar_train_loader, cifar_test_loader, cifar_classes)

if __name__ == "__main__":
    main()

```