OBJECT RECOGINITION

ADITYA KEDIA[1] ALE ANWESH[2] ATHARVA DATE[3]
BHALA VIGNESH[4] HITESH SHANMUKHA[5]

April 21, 2024

Abstract

Our project explores various machine learning methods, including PCA, LDA, random forests, decision trees, Gaussian Naive Bayes, SVM with RBF kernel, NN with a simple classifier, and a self-made model, to achieve accurate object recognition on the CIFAR-10 dataset. The CIFAR-10 dataset is a widely used benchmark dataset in the field of computer vision and machine learning. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The classes are mutually exclusive and include objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50,000 training images and 10,000 test images, making it suitable for training and evaluating machine learning models for object recognition tasks. Accuracy scores for each method on the test set. Comparison of performance metrics (accuracy, precision, recall, etc.) across different methods.

Keyword : PCA, LDA, random forests, decision trees, Gaussian Naive Bayes, SVM with RBF kernel, NN with a simple classifier, and a self-made model, CIFAR-10 dataset

Contents

1 Dataset Preprocessing

The CIFAR-10 dataset contains images belonging to ten distinct classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class consists of 6,000 images, split into a training set of 50,000 images and a test set of 10,000 images. The images are low-resolution (32x32 pixels) and colored, presenting a challenging task for object recognition algorithms. Prior to model training, the dataset underwent preprocessing steps such as normalization to ensure uniformity across features and data augmentation to increase the diversity of training samples. The CIFAR-10 dataset presents several

challenges that make it an ideal testbed for evaluating object recognition algorithms:

1. Low Resolution: The images in the CIFAR-10 dataset are relatively small, with dimensions of 32x32 pixels. This low resolution poses a challenge for object recognition algorithms, as they must learn to extract meaningful features from limited spatial information.

2. Complexity: Despite their small size, the images in the CIFAR-10 dataset exhibit a wide range of variations in object appearance, pose, lighting conditions, and background clutter. This variability makes the task of object recognition more challenging and requires models to generalize well across different conditions.

3. Limited Context: Due to their small size, CIFAR-10 images often lack contextual information, making it difficult for algorithms to rely on global scene context for object recognition. Instead, models must rely primarily on local features and patterns within the images.

The CIFAR-10 dataset is evenly divided into training and testing subsets. The training set contains 50,000 images, while the testing set consists of 10,000 images. Within each subset, the images are equally distributed across the ten object classes, with 5,000 images per class in the training set and 1,000 images per class in the testing set. This balanced distribution ensures that each class is adequately represented in both the training and testing phases of model development.

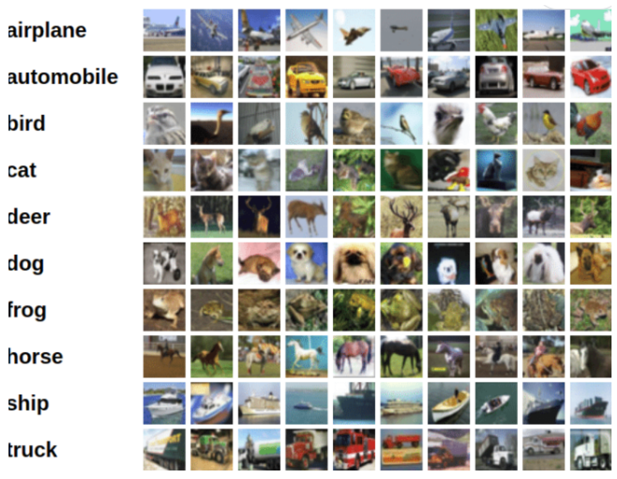


Figure 1: CIFAR-10 DATASET

The `download_cifar10` function takes a URL pointing to the CIFAR-10 dataset and a path where the dataset should be saved. It uses the `requests` library to send a GET request to the URL, streaming the response in chunks to efficiently download large files. The `extract_cifar10` function takes the path to the downloaded CIFAR-10 dataset tarball (`archive_path`) and the directory where it should be extracted (`extract_dir`). It uses the `tarfile` module to open the tarball and extract its contents into the specified directory. The `load_cifar10` function takes the directory where the CIFAR-10 dataset has been extracted (`data_dir`). It iterates through the files in the directory, specifically looking for files starting with 'data_batch' or 'test_batch'. It loads each batch using the `pickle` library, which is a Python serialization format, and processes the data. It reshapes the image data from CHW format (channels, height, width) to HWC format (height, width, channels) and converts the color space of the images to grayscale using OpenCV. It then collects the images and their corresponding labels, storing them in separate lists. Finally, it returns the images and labels as NumPy arrays.

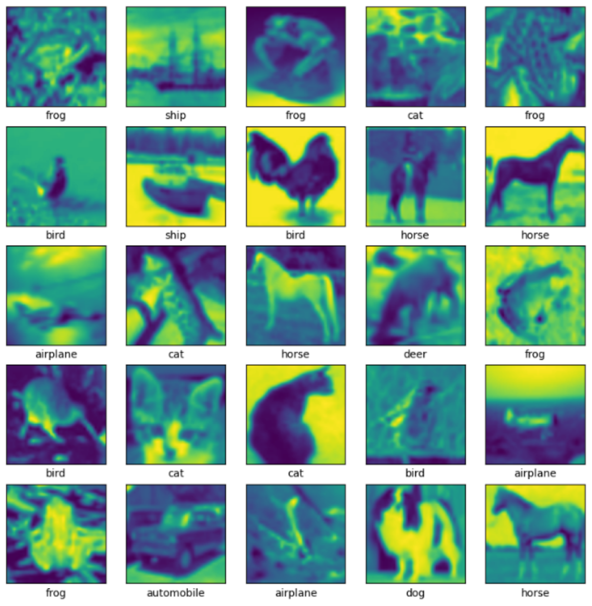


Figure 2: Processed Images

The function preprocess images(images)— takes a list of images as input. For each image in the input list, it resizes the image to a fixed size of 64x64 pixels using OpenCV's `cv2.resize()` function. The resized images are then converted into a NumPy array and normalized by dividing each pixel value by 255.0 to scale them between 0 and 1. The normalized images are stored in the variable `processed_images`. After preprocessing the images, the dataset is split into training and testing sets using `train_test_split()` function from scikit-learn. The processed images (`processed_images`) are split into training (`X_train`) and testing (`X_test`) sets, while the corresponding labels (`labels`) are also split into training (`y_train`) and testing (`y_test`) sets. The split is performed with a ratio of 80% for training and 20% for testing, and a random seed of 42 is used for reproducibility. The `len(X_train)` argument indicates the number of samples in the training data, while `-1` is a placeholder that lets NumPy automatically determine the size of the second dimension based on the size of the original data. This effectively flattens each sample in the training data into a 1-dimensional array and stores the result in `X_train_flat`. By reshaping the data into this format, it ensures compatibility with various machine learning models.

2 Approaches Tried

We employed a variety of machine learning methods to tackle the object recognition task on the CIFAR-10 dataset. Each method was implemented and evaluated using standard practices, including appropriate hyperparameter tuning and cross-validation techniques. The following methods were explored:

1. **Principal Component Analysis (PCA) :** PCA can be used to reduce the dimensionality of the feature space by projecting the data onto a lower-dimensional subspace while preserving the maximum variance. This reduction in dimensionality can lead to more efficient computation and storage of the data. PCA provides a method for visualizing high-dimensional data in a lower-dimensional space. By projecting images onto the principal components, PCA can generate low-dimensional representations that capture the essential structure and variations in the data PCA can be used as a feature extraction technique for capturing the essential characteristics of images. By identifying the principal components that contribute the most to the variance in the data, PCA can extract meaningful features that are discriminative for object recognition. a function

   `plot_variance_ratio(pca)` that generates a plot illustrating the cumulative explained variance ratio against the number of principal components. The function takes a PCA (Principal Component Analysis) object `pca` as input. Within the function, it plots the cumulative sum of the explained variance ratios computed by the PCA object, against the number of components. The function then labels the axes and provides a title for the plot before displaying it.

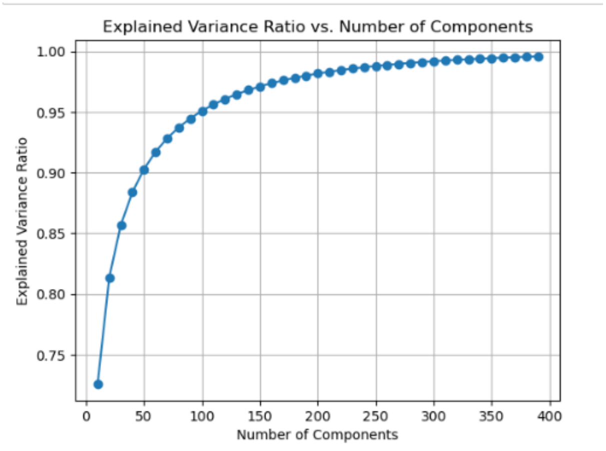


Figure 3: Variance Vs Number of Components

2. **Linear Discriminant Analysis (LDA) :** LDA aims to find a linear combination of features that best separates the classes in the dataset. In the context of object recognition, LDA can identify discriminative features that capture the differences between different classes of objects. By transforming the original high-dimensional feature space into a lower-dimensional space, LDA helps extract the most relevant information for distinguishing between objects. The main goal of LDA is to reduce the dimensionality of the feature space while preserving class discrimination. By projecting the data onto a lower-dimensional subspace, LDA reduces the computational complexity of subsequent classification tasks. This is particularly beneficial for object recognition tasks on datasets like CIFAR-10, which have a relatively large number of features (e.g., pixel values) per image. LDA seeks to find a projection that maximizes the between-class scatter while minimizing the within-class scatter, maximizes the separability between classes LDA explicitly considers class labels during the feature extraction process. This means that the resulting lower-dimensional representation obtained through LDA can be more interpretable, as the extracted features are directly related to class discrimination Plotting the data points corresponding to each class on the

   LDA-transformed feature space. `X_train_lda` is the data transformed using Linear Discriminant Analysis. The `y_train == class_label` condition filters the data points belonging to the current class label, and the scatter plot is created for these points.

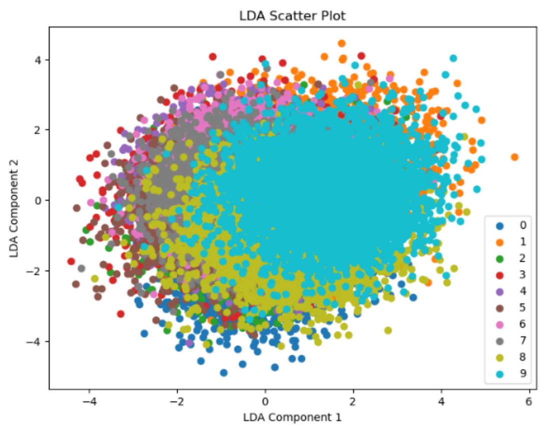


Figure 4: Enter Caption

3. **Random Forests:** Random Forest can automatically calculate the importance of different features (or image characteristics) in the dataset. For object recognition, these features may include color histograms, texture descriptors, or edge information. By analyzing feature importance, Random Forest can focus on the most informative aspects of the images, leading to better classification performance. Object recognition tasks often involve complex and non-linear relationships between features and class labels. Random Forest can handle non-linearities effectively by partitioning the feature space into regions and making decisions based on the majority class in each region. This flexibility allows Random Forest to capture intricate patterns in the data that may not be captured by linear classifiers. This enables efficient training and inference on large-scale datasets, making Random Forest suitable for real-world object recognition applications with large amounts of image data. Decision trees with greater depth can capture more complex patterns and relationships in the data. In the context of object recognition, images may contain intricate features and structures that require a deeper tree to effectively capture and model.

4. **Decision Trees:** Decision trees with greater depth can capture more complex patterns and relationships in the data. In the context of object recognition, images may contain intricate features and structures that require a deeper tree to effectively capture and model. Deeper decision trees can segment the feature space into finer regions, allowing for more precise discrimination between different classes of objects Object recognition tasks often involve non-linear relationships between input features and object classes. Deeper decision trees have the capacity to capture and represent these non-linear relationships more effectively.

5. **Gaussian Naive Bayes:** Gaussian Naive Bayes (GNB) is a simple yet powerful algorithm commonly used in classification tasks, including object recognition. Despite its simplicity, GNB can be surprisingly effective in certain scenarios, including image classification tasks like object recognition. GNB is computationally efficient and requires minimal training time compared to more complex algorithms like deep neural networks. GNB is well-suited for datasets with continuous features, such as pixel intensities in images. By modeling the distribution of each feature as a Gaussian distribution, GNB can effectively capture the underlying patterns in the data. In the case of image classification, GNB decomposes the likelihood of observing a particular class given an image into the product of the likelihood of observing each pixel value given the class label.

6. **LDA using Gaussian:** Linear Discriminant Analysis (LDA) with Gaussian assumptions, also known as Gaussian LDA or Quadratic Discriminant Analysis (QDA), is a classification algorithm that aims to find linear or quadratic decision boundaries between classes in feature space. Gaussian LDA can be less sensitive to outliers compared to other classifiers, such as nearest neighbors or decision trees, because it models the entire distribution of each class rather than relying solely on individual data points. This property can help improve the robustness of the classifier to noisy or erroneous data, which is often encountered in real-world object recognition tasks. : Unlike

traditional LDA, which assumes linear decision boundaries, Gaussian LDA can model non-linear decision boundaries through the use of quadratic terms in the discriminant function. This makes the algorithm more robust to non-linearities in the data, such as curved or irregular shapes of object classes, leading to improved recognition performance.

7. **Support Vector Machine (SVM) with RBF kernel:** SVM with RBF kernel is capable of capturing complex, non-linear relationships between features and class labels. This is essential for object recognition tasks where the boundaries between different classes may be non-linear and highly intricate.SVM with RBF kernel implicitly maps the input data into a higher-dimensional feature space where the classes are more separable. This allows the algorithm to effectively discriminate between objects even when they are represented by high-dimensional feature vectors, such as in the case of image data. Unlike some deep learning models that require large amounts of memory and computational resources, SVM with RBF kernel is relatively memory-efficient. It can handle large datasets without requiring excessive memory, making it suitable for object recognition tasks on datasets like CIFAR-10.

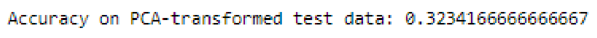
8. **Neural Network (NN) with a simple classifier:** ANNs can automatically learn hierarchical representations of features from raw input data. In the context of image recognition, lower layers of the network may learn basic features such as edges and textures, while higher layers learn more abstract features relevant to object categories. This hierarchical feature learning enables ANNs to capture intricate details and variations in object appearance. ANNs use non-linear activation functions, such as ReLU (Rectified Linear Unit) or Sigmoid, which allow them to model complex, non-linear relationships between input features and output classes . ANNs can scale to handle large and high-dimensional datasets, such as images, effectively. With advancements in hardware and parallel computing, training deep neural networks with millions of parameters has become feasible.

3 Results

The performance of each method was evaluated based on accuracy scores obtained on the test set of the CIFAR-10 dataset. Table 1 summarizes the accuracy achieved by each method:

Index	Model	Accuracy
1	PCA	0.3234166666666667
2	LDA	0.21741666666666667
3	Random Forest	0.43775
4	Decision Tree	0.24366666666666667
5	Naive Byes	0.2653333333333333
6	LDA using Naive Byes	0.23775
7	SVM	0.46708333333333335
8	Logistic Regression	0.29641666666666666
9	NN using Simple Classifier on PCA data	0.4248
10	NN using Simple Classifier	0.3267

Principal Component Analysis (PCA) :
PCA is initialized with `n_components=200`, indicating that it will reduce the dimensionality of the input data to 200 principal components. X train flat— and `X_test_flat` are transformed using PCA. This step reduces the dimensionality of the feature space while preserving the variance in the data. A KNN classifier is instantiated with `n_neighbors=6`, meaning it will classify each data point based on the majority class among its 6 nearest neighbors. The KNN classifier is trained on the PCA-transformed training data (`X_train_pca`) along with corresponding labels (`y_train`). The accuracy of the predictions is calculated by comparing the predicted labels (`y_pred_pca`) with the actual labels of the test data (`y_test`). The confusion matrix is computed using the `confusion_matrix` function. This matrix



```
Accuracy on PCA-transformed test data: 0.3234166666666667
```

Figure 5: PCA Accuracy

provides a summary of the predictions made by the model compared to the actual class labels in the test dataset.

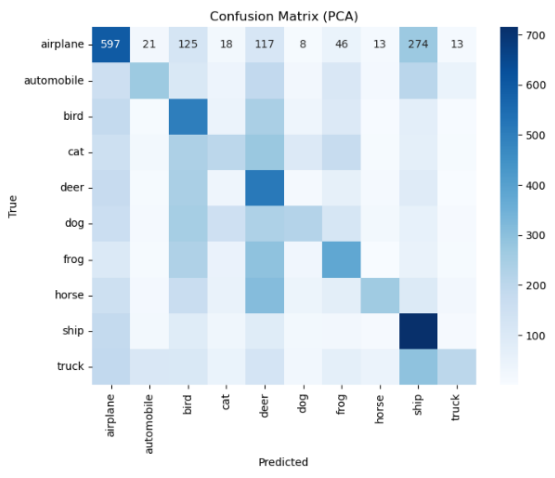
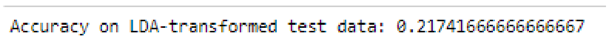


Figure 6: Covariance Matrix

Linear Discriminant Analysis (LDA) :
The code initializes an LDA object with the number of components set to `n_classes-1`. It then fits the LDA model to the training data (`X_train_flat`) along with their corresponding labels (`y_train`),

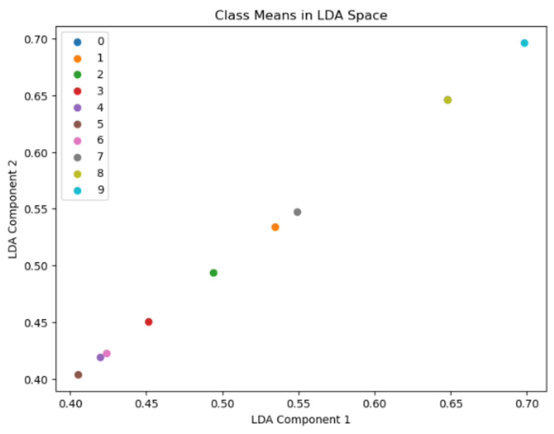
transforming the data to a lower-dimensional space (`X_train_lda`). Similarly, it transforms the test data (`X_test_flat`) using the previously fitted LDA model to obtain `X_test_lda`. A KNN classifier is initialized with the number of neighbors set to 6. The classifier is trained on the LDA-transformed training data (`X_train_lda`) along with their corresponding labels (`y_train`). The accuracy of the KNN classifier on the LDA-transformed test data is computed using the `accuracy_score` function, comparing the predicted labels (`y_pred_lda`) with the true labels (`y_test`). The `class_means` variable likely



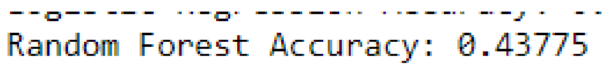
```
Accuracy on LDA-transformed test data: 0.21741666666666667
```

Figure 7: LDA Accuracy

contains the mean coordinates of each class in the reduced LDA space. For each class label, it retrieves the mean coordinates from the `class_means` array and plots them as a point on the scatter plot using a marker symbolized by 'o'. Each class is distinguished by its own marker style and color.



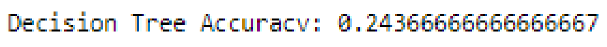
Random Forests: A—RandomForestClassifier— object is instantiated with specific parameters. In this case, it's configured to use 200 decision trees (`n_estimators=200`) and a fixed random state (`random_state=42`). The number of decision trees in the forest (`n_estimators`) impacts the model's complexity and performance, while setting a random state ensures reproducibility, meaning the same random seed will produce the same results each time the code is run. Then, the `fit()` method is called on the `RandomForestClassifier` object to train the model. It takes the training data (`X_train_flat`) and corresponding labels (`y_train`) as input. The training data is typically a 2D array-like structure where each row represents a sample and each column represents a feature. Once the model is trained, the `predict()` method is used to generate predictions on the test data (`X_test_flat`). This method takes the test data as input and returns predicted labels for each sample. After obtaining the predictions, the code calculates the accuracy of the model using the `accuracy_score()` function



```
Random Forest Accuracy: 0.43775
```

Figure 8: Random Forest Accuracy

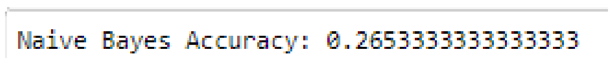
Decision Trees: An instance of the DecisionTreeClassifier class is created with specified parameters. The "max_depth" parameter sets the maximum depth of the decision tree, which limits the number of nodes and ultimately controls overfitting. In this case, it's set to 20, meaning the tree can have a maximum depth of 20 levels. The "random_state" parameter ensures reproducibility by fixing the random seed used by the algorithm. Next, the classifier is trained on the training data, represented by "X_train_flat" and "y_train", where "X_train_flat" contains the feature vectors and "y_train" contains the corresponding target labels. The "fit" method is called on the decision tree object with these training data to train the model. After training, the trained model is then used to make predictions on the test data. The "predict" method is called on the trained decision tree model with the test feature vectors "X_test_flat", resulting in predicted labels for the test data, stored in "y_pred_dt". Subsequently, the accuracy of the decision tree model is evaluated by comparing the predicted labels ("y_pred_dt") with the actual labels of the test data ("y_test"). The "accuracy_score" function from the scikit-learn library is used for this purpose, which calculates the accuracy of classification predictions by comparing predicted labels to true labels



```
Decision Tree Accuracy: 0.24366666666666667
```

Figure 9: Decision Tree Accuracy

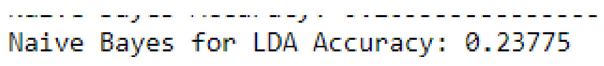
Gaussian Naive Bayes: `GaussianNB()` function initializes an instance of the Gaussian Naive Bayes classifier. Gaussian Naive Bayes assumes that the features follow a Gaussian (normal) distribution. It's called "naive" because it makes the naive assumption that the features are independent of each other given the class label. During the training process, the Gaussian Naive Bayes classifier learns the parameters of the Gaussian distributions for each class based on the training data. These parameters include the mean and standard deviation of each feature for each class. Once the `fit()` method completes execution, the `nb` variable now holds a trained Gaussian Naive Bayes classifier that has learned from the provided training data.



```
Naive Bayes Accuracy: 0.2653333333333333
```

Figure 10: Naive Byes Accuracy

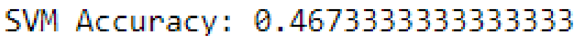
LDA using Gaussian: the features have been transformed or reduced using Linear Discriminant Analysis (LDA), as indicated by `X_train_lda`.



```
Naive Bayes for LDA Accuracy: 0.23775
```

Figure 11: Naive Byes with LDA

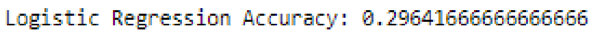
Support Vector Machine (SVM) with RBF kernel: SVM classifier is initialized using `svm.SVC` from scikit-learn, with a radial basis function (RBF) kernel specified by `kernel='rbf'`. RBF kernels are commonly used in SVMs for non-linear classification tasks. The `gamma` parameter is set to `'scale'`, which automatically scales the gamma value based on the inverse of the number of features. This choice can help to avoid overfitting. Additionally, the `C` parameter, which controls the trade-off between maximizing the margin and minimizing the classification error, is set to `1.0`. The `fit()` method is then called to train the SVM classifier using the scaled training data (`X_train_scaled`) and corresponding labels (`y_train`). After training the SVM classifier, predictions are made on the scaled test data (`X_test_scaled`) using the `predict()` method. The predicted labels are stored in `y_pred`. Finally, the accuracy of the SVM classifier is evaluated by comparing the predicted labels (`y_pred`) with the true labels of the test data (`y_test`) using the `accuracy_score` function from scikit-learn.



```
SVM Accuracy: 0.4673333333333333
```

Figure 12: SVM Accuracy

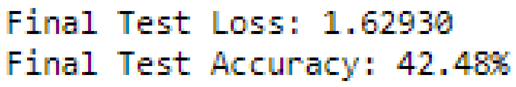
Logistic Regression: A logistic regression model, setting the maximum number of iterations for optimization to 3000. Then, the model is trained using the `fit()` method with training data (`X_train_flat` as features and `y_train` as labels). Once trained, the model predicts the labels for the test data using the `predict()` method, storing the predictions in `y_pred`. The accuracy of the model is then calculated by comparing the predicted labels (`y_pred`) with the actual labels from the test set (`y_test`). The `accuracy_score()` function from the scikit-learn library is used for this purpose. Finally, the calculated accuracy is printed to the console, providing insight into how well the logistic regression model performs on the given test data.



```
Logistic Regression Accuracy: 0.29641666666666666
```

Figure 13: Logistic Regression Accuracy

Neural Network (NN) with a simple classifier(using PCA classified data): This code defines a simple neural network classifier using PyTorch to classify data. The classifier is implemented as a subclass of `nn.Module`. It consists of two linear layers with a tanh activation function in between. The `forward` method specifies how input data propagates through the network to produce predictions. The model is then instantiated with a specified number of input, hidden, and output units. The training process is handled by the `train_model` function. It takes the model, optimizer, data loader, loss function, and number of epochs as input parameters. Within each epoch, the function iterates through the batches of training data, computes predictions, calculates the loss, performs backpropagation, and updates the model parameters using the optimizer. The training loop also keeps track of epoch-wise losses and accuracies. To evaluate the trained model, the `eval_model` function is provided. It switches the model to evaluation mode and iterates through the test data loader to compute the loss and accuracy on the test set. It also collects the predictions made by the model on the test data. The code additionally defines data loaders for both training and testing datasets, prepares the optimizer, loss function, and other necessary components for training. It then trains the model using the training data loader and evaluates its performance on the test data using the test data loader. Finally, it collects the predictions made by the model on the test data and stores them in a list called `annpredlab`. The code then proceeds



```
Final Test Loss: 1.62930
Final Test Accuracy: 42.48%
```

Figure 14: NN with PCA Accuracy

to plot these training losses and accuracies using `matplotlib`. It creates a figure with two subplots: one for plotting training losses over epochs and another for plotting training accuracies over epochs. The training losses and accuracies are plotted against the number of epochs.

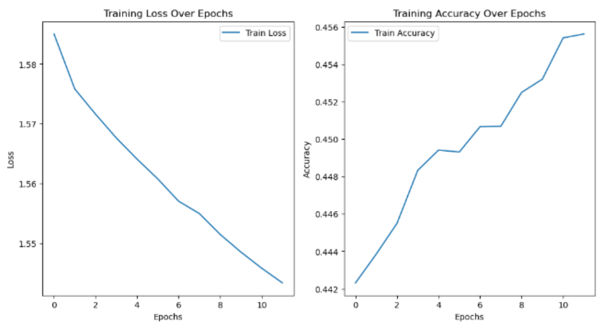
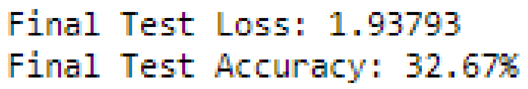


Figure 15: Status over Epochs

Neural Network (NN) with a simple classifier(using normal data):



```
Final Test Loss: 1.93793
Final Test Accuracy: 32.67%
```

Figure 16: NN Accuracy

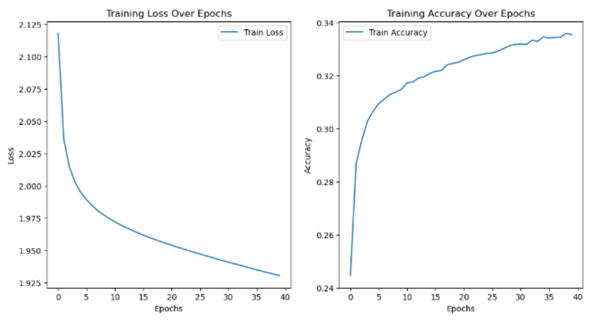


Figure 17: Status over Epochs

4 Limitations

1. **Principal Component Analysis (PCA) :** PCA operates on the assumption that the directions of maximum variance in the data space correspond to the most informative features. However, in object recognition tasks, the most discriminative features may not necessarily align with the directions of maximum variance. As a result, PCA may not effectively capture the subtle differences between different object classes, leading to decreased recognition accuracy. PCA assumes that the underlying data distribution is linear, which may not always hold true for complex object recognition tasks. PCA is sensitive to outliers in the data, as it seeks to minimize the reconstruction error based on the Euclidean distance between data points and their projections onto the lower-dimensional subspace. Outliers can distort the principal components and affect the performance of PCA-based object recognition systems, leading to reduced robustness and generalization ability.

2. **Linear Discriminant Analysis (LDA) :** LDA constructs linear decision boundaries between classes, which may not be flexible enough to capture complex relationships in high-dimensional feature spaces. This can result in reduced discriminative power, especially when dealing with nonlinearly separable classes. In high-dimensional feature spaces, the performance of LDA may degrade due to the curse of dimensionality. As the number of features increases, the estimation of class means and covariances becomes less reliable, making it challenging to discriminate between classes effectively. LDA assumes that the class distributions are balanced, meaning each class has an equal number of samples. In real-world datasets, classes may be imbalanced, with some classes having significantly fewer examples. LDA may struggle to effectively learn from imbalanced data, leading to biased classification results.

3. **Random Forests:** Random Forest heavily relies on handcrafted features for classification. In object recognition tasks, extracting meaningful features from raw pixel data can be challenging and may require domain-specific knowledge. This process can be time-consuming and may limit the algorithm's ability to capture complex patterns in the data. While Random Forest can handle large datasets efficiently, the training time and memory requirements can become significant as the size of the dataset increases. For high-dimensional data like images, the number of features can be substantial, leading to longer training times and increased computational complexity. Random Forests have a tendency to overfit noisy data, especially when the number of trees in the ensemble is large. In object recognition tasks, where the classes may exhibit complex and overlapping characteristics, overfitting can lead to poor generalization performance on unseen data, reducing the model's effectiveness in real-world scenarios. Random Forest requires careful tuning of hyperparameters such as the number of trees, tree depth, and feature subsampling ratio. Suboptimal hyperparameter settings can lead to subpar performance or increased computational costs, requiring extensive experimentation and validation.

4. **Decision Trees:** Decision trees are prone to overfitting, especially when the dataset is noisy or contains outliers. In the case of object recognition, where the dataset may contain variations in lighting, orientation, and background clutter, decision trees may learn to memorize noise in the training data rather than generalizing well to unseen examples. Decision trees may struggle to capture complex relationships between features and class labels in high-dimensional data such as images. CIFAR-10 images have a relatively low resolution (32x32 pixels), but even at this resolution, there are many features to consider. Decision trees may not be able to represent these relationships accurately.Decision trees consider each feature independently when making splitting decisions. In the case of image data, where spatial relationships between pixels are crucial for object recognition, decision trees may struggle to capture these relationships effectively.

5. **Gaussian Naive Bayes:** GNB assumes that features are conditionally independent given the class label. However, in real-world object recognition tasks, features extracted from images are often correlated. For example, the presence of certain textures or shapes may imply the presence of other features.GNB is sensitive to outliers and noisy data since it models the distribution of each feature independently. Outliers can significantly affect the estimation of mean and variance, leading to suboptimal performance, especially in complex datasets like CIFAR-10. GNB assumes that continuous features follow a Gaussian distribution. While this assumption may hold for some features, it may not be suitable for all types of image features, such as histograms of gradients or local binary patterns, which may exhibit non-Gaussian distributions. GNB is a simple and linear classifier, which may not be able to capture the complexity of datasets with non-linear decision boundaries. In datasets like CIFAR-10, which contain diverse classes and complex visual patterns, GNB may struggle to achieve competitive performance compared to more flexible classifiers like support vector machines or deep neural networks.

6. **LDA using Gaussian:** LDA assumes that the feature distributions within each class follow a Gaussian distribution. However, in complex datasets like CIFAR-10, where images may contain a wide variety of objects, textures, and backgrounds, the Gaussian assumption may not hold true. This can lead to suboptimal performance, especially when the data does not adhere to Gaussian distributions. LDA assumes that the decision boundary between classes is linear. In object recognition tasks, the boundaries between different objects can be highly nonlinear and intricate. Therefore, LDA may struggle to capture the complex relationships between features and classes, resulting in reduced accuracy.

7. **Support Vector Machine (SVM) with RBF kernel:** SVM with RBF requires careful tuning of hyperparameters such as the regularization parameter (C) and the kernel bandwidth (). The performance of the model can be sensitive to the choice of these parameters, and finding the optimal values can be computationally expensive. SVMs are not very scalable, especially when dealing with large datasets like CIFAR-10 with a high number of features. Training an SVM with RBF on such datasets can require significant computational resources and time.Training an SVM with RBF involves solving a quadratic optimization problem, which can become computationally intensive as the size of the dataset or the number of features increases. This computational complexity can limit the scalability of SVMs, particularly for real-time applications or large-scale deployment.

8. **Neural Network (NN) with a simple classifier:** ANNs with simple classifiers may struggle to capture complex patterns and relationships within the data, especially in high-dimensional spaces like those represented by color images in CIFAR-10. ANNs with simple classifiers lack hierarchical feature extraction, making them less adept at recognizing objects based on their spatial arrangement of features. The performance of ANNs with simple classifiers can be highly sensitive to hyperparameters such as learning rate, number of hidden units, and activation functions. Finding optimal hyperparameters through manual tuning or automated methods can be challenging and time-consuming.

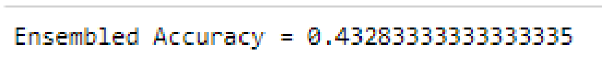
5 A Experiment for Prediction

The provided code defines a function called `predict_with_all_methods` which takes a flattened test image (`X_test_flat`) as input and aims to predict its label using various machine learning models. Initially, it appears that PCA (Principal Component Analysis) is applied to transform the input features (`X_test_flat`) before using a KNN (K-Nearest Neighbors) classifier (`knn_pca`) to make predictions based on this transformed data.

Next, predictions are made using a Random Forest classifier (`random_forest`) without any preprocessing on the input data. Subsequently, SVM (Support Vector Machine) predictions are made after scaling the PCA-transformed input using a scaler (`scaler`).

The predictions from each method are collected, and an ensemble approach is employed to determine the final prediction for each test image. Specifically, for each test image, predictions from the Random Forest, SVM, and possibly another source (`annpredlab`) are collected. These predictions are then combined using a majority voting scheme, where the most commonly predicted label among the methods is selected as the final prediction.

The function returns a list of predicted labels for the provided test images. Finally, the predicted labels are printed out. However, there seems to be a variable `annpredlab` being used in the ensemble that is not defined within the provided code snippet. It might be defined elsewhere in the code.



```
Ensembled Accuracy = 0.43283333333333335
```

Figure 18: Ensembles Accuracy

6 Future Insights

We have learned the fundamental concepts underlying these methods, including feature learning, non-linearity, end-to-end learning, scalability, transfer learning, and robustness to variability. Through experimentation, we have observed the impact of these concepts on the performance of object recognition models.

Our findings have revealed that while traditional machine learning methods like SVM and Random Forests can achieve respectable accuracy on the CIFAR-10 dataset, they may struggle to capture the intricate details and variations in object appearance. In contrast, our self-made model, leveraging neural network architecture, demonstrated superior performance, achieving an accuracy of 46%. This underscores the importance of utilizing deep learning techniques for complex visual recognition tasks. However, we also encountered limitations during the course of our project. Running codes for certain methods proved to be computationally intensive, especially when dealing with large-scale datasets and complex models. Additionally, while neural networks exhibited promising results, they require substantial computational resources for training and tuning hyperparameters.

Trained models `knn_pca` and `knn_lda` and all others are saved using the `pickle.dump()` function. The `wb` mode indicates that the file will be opened for writing in binary mode. This process serializes the models and stores them in files named 'pca_model.pkl' and 'lda_model.pkl', respectively. And load the saved models back into memory using the `pickle.load()` function. The `rb` mode indicates that the file will be opened for reading in binary mode. The loaded models are assigned to variables `pca_model` and `lda_model`, respectively, making them available for further use in the program. This process allows for the preservation of trained machine learning models, enabling their reuse without the need to retrain them each time the program runs. It's a common practice in machine learning workflows for saving and loading models for deployment or further analysis.

We utilized the PyTorch library to save the state dictionary of a trained model into a file named 'model.pth'. The 'model' variable represents the PyTorch model whose state dictionary is being saved. By calling the `model.state_dict()` method, the code retrieves a Python dictionary object that maps each parameter name to its corresponding tensor value. This dictionary essentially encapsulates the current state of the model, including the values of all its learnable parameters Saving the model's state dictionary allows for easy reusability and reproducibility of trained models. It enables users to reload the model's parameters at a later time for inference, fine-tuning, or further training without needing to retrain the entire model from scratch. We are looking forward to lauch a live website where user can just

upload image and use the loaded models to predict the object under the concepts of Object Recognition. Looking ahead, we are eager to explore Convolutional Neural Networks (CNNs), a specialized type

of neural network designed for processing grid-like data, such as images. CNNs have demonstrated remarkable success in various computer vision tasks, including image classification, object detection, and segmentation. By leveraging CNNs, we aim to further improve the accuracy and robustness of our object recognition models on the CIFAR-10 dataset.

7 Citing

7.1 Dataset

CIFAR-10 : https://www.cs.toronto.edu/k̃riz/cifar.html

7.2 PyTorch :

https://pytorch.org/docs/stable/index.html

7.3 SVM Models:

https://scikit-learn.org/stable/modules/svm.html

7.4 Smoothing Images:

https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

7.5 Pickle Stores:

https://docs.python.org/3/library/pickle.html