High-accuracy Traditional Machine Learning on CIFAR-10 with Integrated Feature Extraction Methods

Huang Junran*, Sun Xueli*, Zhen Qi*

April 2025

1 Introduction

Challenge Setting. In this challenge, we are given the CIFAR-10 image dataset. The goal is to reach 70% classification accuracy on the private testing set, *without using deep learning architecture such as RNN, CNN, transformer, etc.*

CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images of 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. In this challenge, test images are further divided into 2000 public test images and 8000 private test images.

- Training set: (50000, 32, 32, 3)
- Public testing set: (2000, 32, 32, 3)
- Private testing set: (8000, 32, 32, 3)

In this report, we will start with dataset analysis, followed by demonstrating the process of the data augmentation, features extraction, and classifier exploration, and finally conclude with the final solution description.

2 Dataset analysis

2.1 Analysis on Category Labels

The dataset has a total of 10 label values (0-9) that is, a total of 10 categories. The following graph indicates the label distribution of public training set.

Class ID	Class Name	Count
0	Plane	5,038
1	Car	5,016
2	Bird	5,032
3	Cat	4,991
4	Deer	4,982
5	Dog	4,967
6	Frog	4,985
7	Horse	4,998
8	Ship	5,002
9	Truck	4,989

Table 1: Class Distribution in Training Dataset

2.2 Visualization of each category

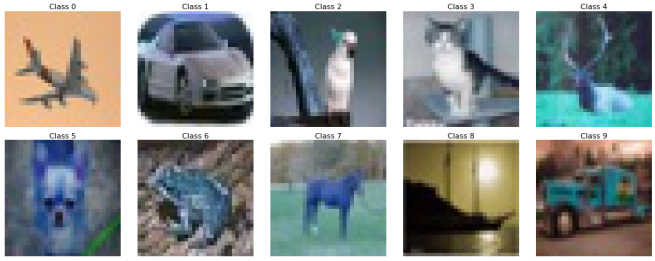

Figure 1: Visualization of one example for each category

3 Preprocessing and Data Augmentation

We have a training set with 50000 labeled images and a test set with 10000 unlabeled images. Instead of using the original training set, we make some changes to enlarge the training set.

Mini dataset Since the original training dataset is too large for model training, fine-tuning, and prediction, we further split the dataset, obtaining a mini training set of size (2000, 32, 32, 3) and a mini validation set of size (500, 32, 32, 3). In the remaining context, if there is no further declaration of the dataset utilized, we will be using the mini dataset for testing and illustration.

3.1 Flattened Image

Each image sample is a 32×32 pixels RGB image themed on a certain class, thus we naturally abstract the image into a matrix of size $32 \times 32 \times 3$. A vanilla preprocessing way is to flatten the matrix into a vector. After that, we simply stack all the image vectors to obtain the entire matrix for training (X_train) and testing data (X_test).

- X_train: (50000, 3072)
- X_test: (10000, 3072)

By experiments, unfortunately, training with flattened image vectors shows low accuracy for all the classifiers. A possible reason is that neighborhood information in

the 2D image is lost during flattening, which motivates us to find more effective data augmentation and feature extraction methods.

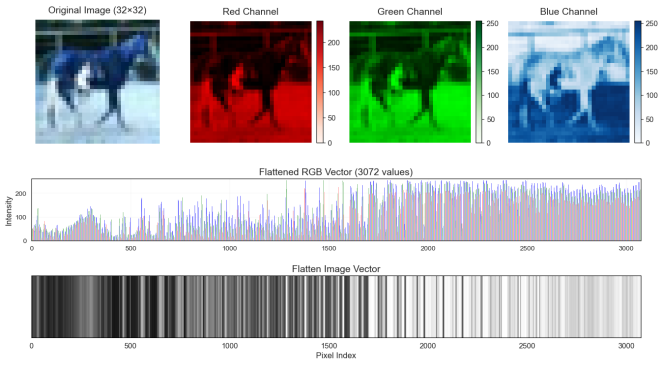

Figure 2: Flattened Image Vector

3.2 Rotation

We perform random rotation augmentation on the 32×32 pixel images. Each image in the training set undergoes a random rotation by an angle $\theta \in [-15°, 15°]$ with probability $p = 0.2$. Boundary handling is done by `BORDER_REFLECT` (reflects pixels at image boundaries). We also perform bilinear interpolation.

The augmentation strategy helps improve model robustness to orientation variations while preserving image content through appropriate boundary handling.

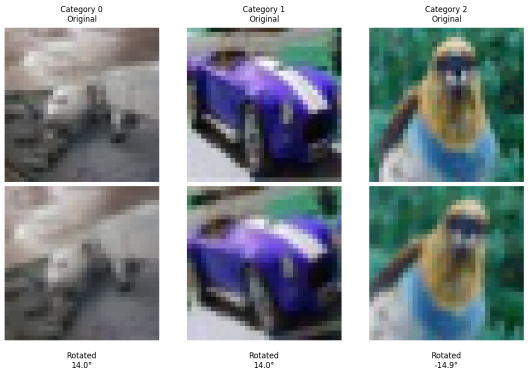

Figure 3: Comparison of Rotated and Original Image

3.3 Horizontal Flipping

We perform horizontal flipping augmentation on the 32×32 pixel images.

The strategy may improve accuracy by providing more training references for images with horizontal orientation.

Meanwhile, the sample size of the training dataset is doubled in the horizontal flipping process, which also reveals the uneconomical feature of this method.

Figure 4: Comparison of Horizontal Flipped and Original Image

3.4 Color Jitter

Color jitter augmentation randomly perturbs image color channels to improve model robustness to color variations. We first obtain three independent random factors from a uniform distribution: $\beta, \sigma, \gamma \sim \mathcal{U}(1 - 0.2, 1 + 0.2)$.

Brightness adjustment scales the Value channel by a random factor β, clipped to $[0, 255]$ range:
$V' = \text{clip}(V \times \beta, 0, 255)$

Saturation adjustment scales the Saturation channel by σ: $S' = \text{clip}(S \times \sigma, 0, 255)$

Contrast adjustment (applied in RGB space) uses mean-centered scaling with γ:
$\text{RGB}' = \text{clip}\left((\text{RGB} - \mu) \times \gamma + \mu, 0, 255\right)$

where μ is the per-channel spatial mean.

All transformations are applied sequentially with 100% probability, using OpenCV's color space conversions with 8-bit precision.

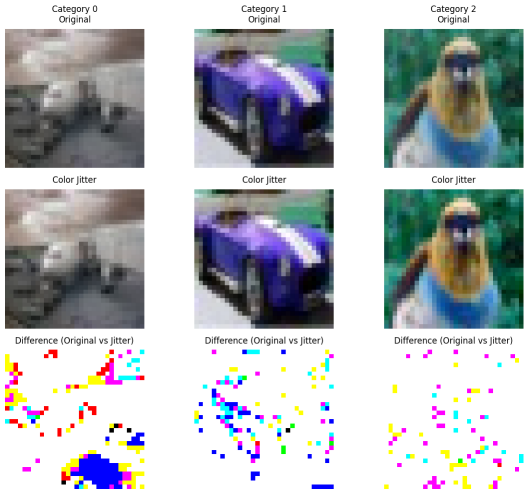

Figure 5: Comparison of Color Jitter Image and Original Image, difference = (Original Image) - (Color Jitter Image), pixel by pixel and channel by channel

3.5 Random Crop

We introduce a random crop to increase model robustness to object feature variations. Since we utilize some feature extraction method (e.g., HOG, EOH) in the upcoming process, if there exists particularly rigid features, the model may fail on some variations.

We introduce cropped images based on the original training dataset. Random cropping forces the model to learn from shifted and scaled features of some objects. It can also avoid overfitting for the model while preserving the discriminative features.

To implement that, the image will be cropped to $\frac{1}{4}$ of the original size as Figure 6 shows, then appended to the training dataset.

Figure 6: Comparison of Random Cropped and Original Image

3.6 Experiment on Data Augmentation Methods

Experimental Protocol Each augmentation method is tested independently on the mini set while maintaining the original flattened training set as baseline set, `SVC(kernel="rbf", C=3, gamma="scale")` as baseline model. The comparative performance metrics are presented in Figure 7.

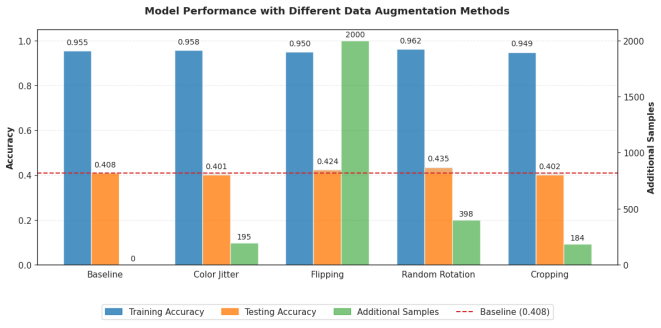

Figure 7: Comparison of different Augmentation Methods

Experiment results The impact of different data augmentation strategies on model performance is evaluated by comparing training accuracy, testing accuracy, and the number of additional samples generated for each method. The baseline model achieves a testing accuracy of 0.408. Among the augmentation methods tested—color jitter (training accuracy: 0.9576, testing accuracy: 0.4007), flipping (training accuracy: 0.95, testing accuracy: 0.424), random rotation (training accuracy: 0.9616, testing accuracy: 0.4347), and cropping (training accuracy: 0.9487, testing accuracy: 0.4018)—flipping and random rotation demonstrate the most significant improvements in generalization performance. Consequently, flipping and random rotation are selected for inclusion in the final training pipeline due to their positive influence on model generalization.

4 Feature Extraction

Models trained on raw pixel data gain low accuracy may because of the following reasons:

- **Local structure** is lost in flattened data, since it simply stretches the whole image to a vector
- **Color information** is nuanced in flattened data, since it records values of R, G, B channels without notating the specific channel the value belongs to
- **Information dimensionality** is huge, though lacking effective information

4.1 Color-based Features

From the image dataset, we can intuitively learn that those images labelled in the same class are more likely to be similar in color distribution. For example, class 4 (deer) tends to have green background, and class 6 (frog) tends to have a green target object. To effectively utilize color information, we introduce two color features in our training.

4.1.1 Mean RGB

Mean RGB captures the average R, G, and B values of each image to summarize the overall color tone style of a sample. For each image, we compute its mean value over R, G, B channels, then append three mean values as new features. The method brings in only three new features for each image, so it is computationally efficient for upcoming model training.

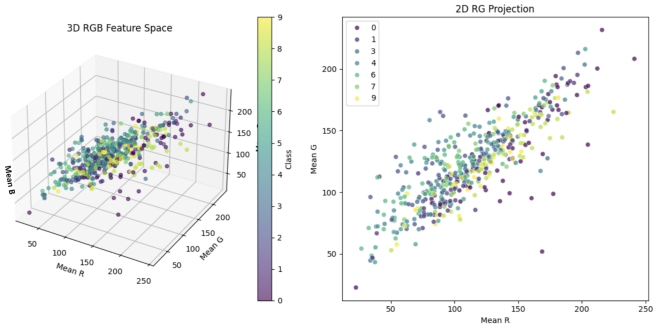

Figure 8: Visualization of Mean RGB as a classification feature

4.1.2 Color Histogram

We introduce the color histogram feature in model training, since the mean RGB cannot detect the color distribution well, though it does capture the overall color tone style. Color histograms provide an informative color distribution by showing how often each color appears.

4.2 Texture-based Features

Texture is a fundamental visual property that describes the spatial arrangement of pixel intensities. Texture arises from structured variations in gradients and repeated local features, making it essential for distinguishing images.

4.2.1 Histogram of Gradient (HOG)

Histogram of Oriented Gradients (HOG) extracts the target contour in the image based on the gradient of each pixel point and better expresses the information in the image with fewer features.

Since we are dealing with colored images, we extract HOG features from R, G, B three color channels, normalize the color for each channel, and compute gradient directions separately. For each pixel in R, G, B, gradients on the x and y direction are defined by $G_x = I(x+1, y) - I(x-1, y)$ and $G_y = I(x, y+1) - I(x, y-1)$. Computing gradient components is implemented by horizontal and vertical filters

- Horizontal gradient operator: $\mathbf{h}_x = [1, 0, -1]$
- Vertical gradient operator: $\mathbf{h}_y = \mathbf{h}_x^\top = [1, 0, -1]^\top$

The gradient direction is calculated through the inverse tangent.

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Then calculate Pixel's gradient magnitude, and distribute it to histogram bins.

$$G = \sqrt{G_x^2 + G_y^2}$$

Divide the 32 × 32-pixel image into cells, with 8×8 pixels in one cell, and 16 cells in total to describe each image. Visualize the images with respect to 3 channels and compare them with the original image.

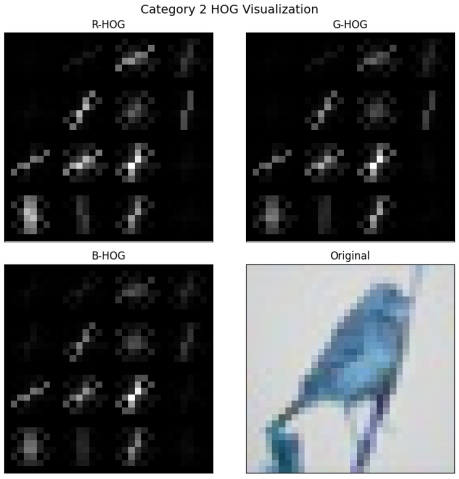

Figure 9: Comparison of HOG Images of R, G, B Channel and Original Image

4.2.2 Edge Orientation Histogram (EOH)

Edge Orientation Histogram (EOH) is similar to HOG, in the sense it utilises gradient information as well. Additionally, it includes an additional edge detection process. The algorithm first calculates the gradient magnitude of each cell. Then perform a threshold to obtain edge cells.

Similar to HOG, divide the target image into 16 sub-images. Based on detected edge cells and their gradients, create an edge orientation histogram for each sub-image.

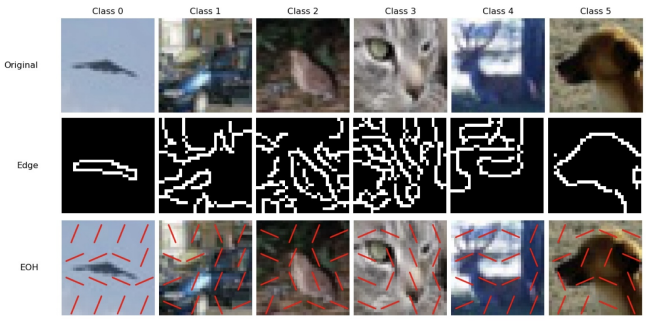

Figure 10: Canny Edge Detection and EOH. EOH visualization utilizes two tricks: (1) use the highest direction in each sub-image as block direction; (2) use edge direction, i.e., the normal direction to gradient as final visualization direction.

4.3 Shape Descriptor: Hu-Moments (HUM)

Image visualization reveals that objects like cars, ships, and planes may appear in varying orientations. While augmenting the dataset through rotations expands sample diversity, it remains insufficient for robust generalization. So we introduce Hu-Moments to reduce the variance caused by rotations.

Hu Moments are a set of seven statistical moments derived from the normalized central moments of an image. These invariants are rotation, scale, and translation invariant, making them suitable for robust shape-based classification.

We can see from the Figure 11 that, for the same object, even after rotating or flipping, they have similar Hu-Moments. For different objects, their Hu-Moments can be quite different.

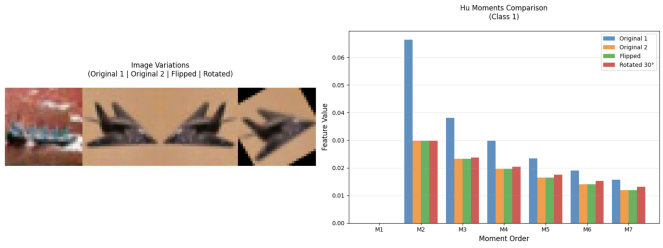

Figure 11: Comparison of different images' Hu-Moments

To implement the Hu-Moments, we first convert the image into grayscale. Then, we use the moment package in `scipy.stats` to help us compute the raw moments, central moments, and normalized central moments of the image. After that, we can derive the seven HUM invariants from the normalized central moments.

4.4 Keypoint-based Feature: Scale-invariant Feature Transform(SIFT)

Scale-Invariant Feature Transform (SIFT) identifies distinctive image features that remain detectable across scales and rotations. The algorithm first locates stable keypoints across different magnifications using
Gaussian differences, then refines their positions while filtering weak candidates. Each keypoint is assigned a dominant orientation based on local gradients, making features rotation-invariant. Finally, it generates compact 128-dimensional descriptors encoding the feature's neighborhood patterns.

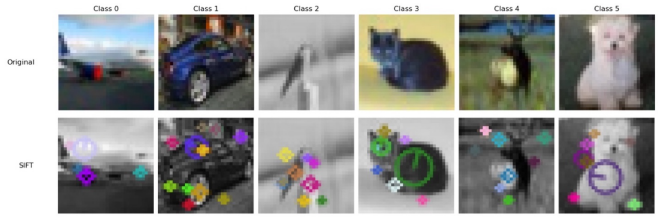

Figure 12: SIFT

4.5 Experiment Results on Different Features

Experimental Protocol We evaluate each feature in two ways. First, delete the features from six features and test the dropped accuracy. Second, each feature is tested independently in the mini set. We use `SVC(kernel="rbf", C=10, gamma="scale")` as baseline model. The comparative performance metrics are presented in Figure 13.

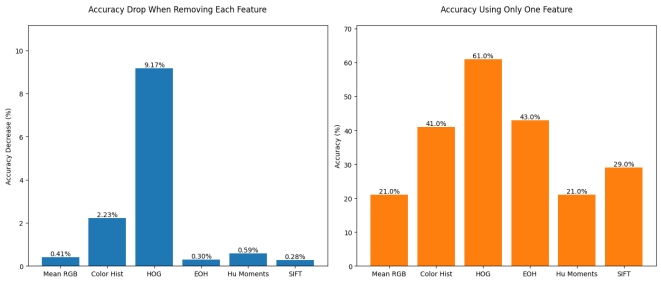

Figure 13: Comparison of different Features

Experiment results The impact of different feature strategies is evaluated by comparing the accuracy dropped when removing each feature and the accuracy when using only each feature.

For the first evaluation, we can see that for all features, accuracy drops when each feature is removed. It indicates that all features can help improve the performance.

For the second evaluation, we can see that some features can achieve high accuracy with only itself, like HOG (testing accuracy: 61%), Color Histogram (testing accuracy: 41%) and EOH (testing accuracy: 43%). For Mean RGB, Hu-Moments, and SIFT, with the individual feature, accuracies are below 30%, showing that these features are less powerful.

Consequently, all features are selected in the final training due to their positive influence on the testing accuracy.

5 Classifier exploration

The challenge requires us to implement at least 2 different classifiers. Our initial strategy is to select classifiers that we learned from the lecture, use them independently, and compare their accuracy, and perform a majority vote to obtain the final model.

However, ensemble learning does not perform as well as expected. The marginal benefit of devastating training time does not seem to worth it. As a result, we finally resort to fine-tuning Single Well-performed Model.

5.1 Classifier selection and comparison

The first step is classifier selection and comparison. We propose five classifiers, including Support Vector Machine (SVM), Random Forest, k-nearest Neighbor (KNN), and Logistic Regression. We test their accuracy

independently without ensembling.

Firstly, we test the performance of 4 different classifiers (Support Vector Machine, Random Forest, K-Nearest Neighbor, and Logistic Regression) with flattened image. Without fine-tuning, it shows that SVM performs much better than any other classifiers.

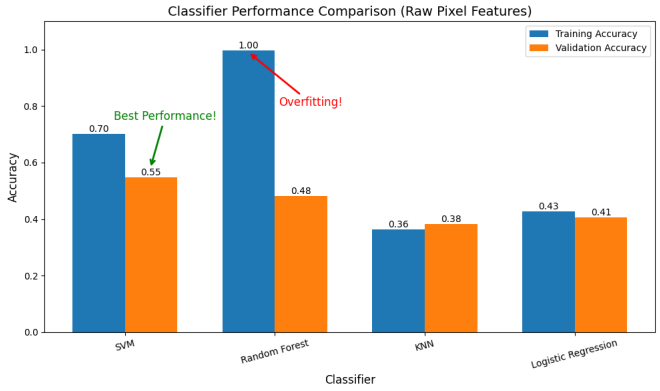

Figure 14: Comparison of different classifier performance on flattened image. From the results, Random Forest overfits on the training set. KNN and Logistic Regression perform poorly.

Then, we test each classifier with solely HOG features, on which SVM still works the best.

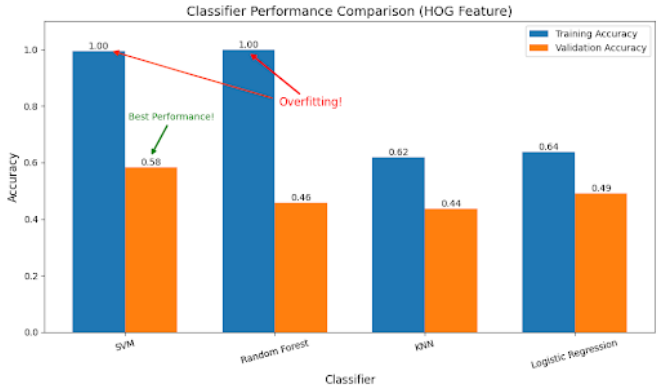

Figure 15: Comparison of different classifier performance in HOG feature. Though overfitting on the training set, SVM performs the best on the validation set.

We also test each classifier on the final selection of data augmentation and feature extraction methods. SVM still works the best.

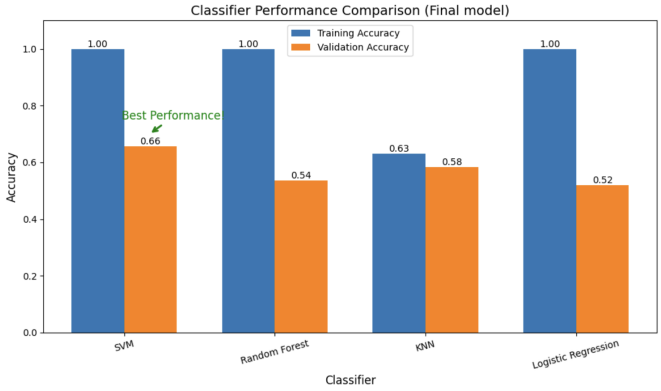

Figure 16: Comparison of different classifier performance in data augmentation and extracted features, though overfitting on the training set, SVM performs the best on the validation set

We try to utilize ensemble learning based on SVM, Random Forest, KNN, surprisingly, the majority voting model does not work better than solely using SVM, and even greatly increases the time cost.

As a result, SVM is chosen as the only classifier in our final model.

5.2 Fine-tuning SVM

We optimized the SVM classifier using grid search over the following parameter space:

```
param_grid = {
    'C': [0.1, 1, 3, 5, 10, 20,
          50, 100, 1000],
    'gamma': [1, 0.1, 0.01, 'scale'],
    'kernel': ['rbf', 'linear']
}
```

The optimal configuration achieving the highest accuracy was:

```
[C=10, gamma='scale', kernel='rbf']
```

This configuration was used for final model evaluation.

6 Final solution

Following the report, the methodology we determine our final model follows:

(a) Data augmentation selection
{Random Rotation, Horizontal Flipping} are chosen.
Training set: $(50000, 32, 32, 3) \longrightarrow (110000, 32, 32, 3)$

(b) Features extractors selection
{Mean RGB, Color Histogram, HOG, EOH, HUM, SIFT} are chosen.
Training set: $(110000, 32, 32, 3) \longrightarrow (110000, 1750)$
Testing set: $(10000, 32, 32, 3) \longrightarrow (10000, 1750)$

(c) Classifier selection
SVM is chosen.

(d) Fine-tuning parameters

`[C=10, gamma='scale', kernel='rbf']`

We are almost done. However, there is still one problem before the final training. That is, the features dimension is too large and represents different properties of the image. Thus, to improve the feature performance, we need to standardize the feature values and lower the dimension using PCA.

6.1 Standard Scaling

To ensure all features contribute equally to the downstream analysis, we standardize all features to zero mean and unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation of each feature.

6.2 Principal Component Analysis (PCA)

To lower the dimension of the dataset, we apply PCA to the scaled features, reducing dimensionality from 1750 to 500 principal components. We fine-tune `n_components` in the mini set:

n_components	accuracy (%)	Training time (s)
0.8	66.53	120.78
0.85	66.56	168.66
0.9	66.45	240.96
0.95	66.41	333.32
200	65.95	75.58
300	66.37	125.06
500	66.99	193.96
800	66.37	316.78

Table 2: PCA `n_components` fine-tuning

In conclusion, to balance the trade-off of efficiency and accuracy, we define n_components = 500.

6.3 Performance Analysis

6.3.1 Accuracy

Our final model gains 76.9% accuracy in the validation set. The confusion matrix is shown in Figure 17.

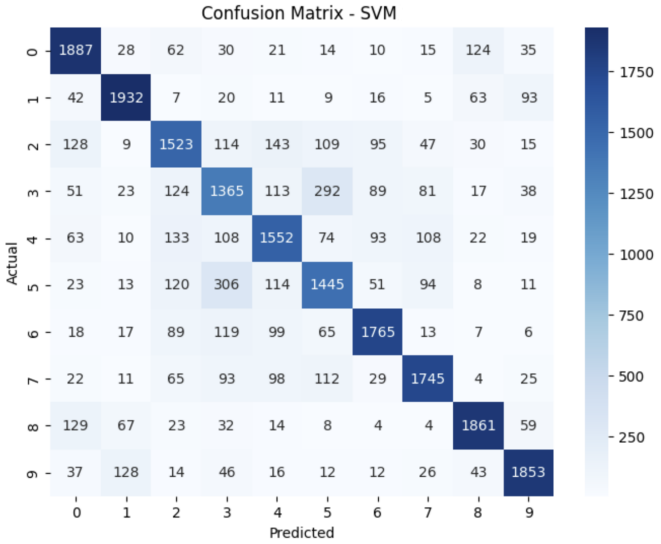

Figure 17: Confusion matrix

However, we achieve only 71.8% accuracy in the public testing set. This might be due to the overfitting issues led by the similarity between the original images and the flipped and rotated images.

7 Conclusion

This work demonstrates that traditional machine learning with integrated feature extraction can achieve competitive accuracy 71.8% on CIFAR-10, proving its viability for image classification. However, the approach shows inherent limitations in handling low-pixel image classification problems. While traditional machine learning remains valuable for interpretability and low-resource settings, achieving higher accuracy may require higher-level techniques.

8 Appendix

8.1 NPZ Data Compression

We use a methodology that systematically transforms raw image and label data into a standardized, compressed format suitable for efficient machine learning processing. It establishes a structured pipeline that enforces consistency in data dimensions while preserving the critical correspondence between images and their labels. By leveraging compression and array-based storage, it optimizes both storage efficiency and computational accessibility for downstream tasks (e.g. upload to Google Drive, data loading).