Image Processing Project 4 Report

1. Introduction

1.1 Problem Description

Image classification is a fundamental task in computer vision that involves assigning a label to an input image from a predefined set of categories. In this assignment, we tackle the multi-class image classification problem using the CIFAR-10 dataset. The goal is to develop a linear classifier capable of distinguishing between 10 different object categories: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

The challenge lies in learning a discriminative mapping from high-dimensional image pixel values (32×32×3 = 3,072 dimensions) directly to class labels using only a linear transformation. This serves as a baseline approach that, while simple, provides valuable insights into the fundamental principles of classification and establishes a performance benchmark for more complex deep learning architectures.

1.2 CIFAR-10 Dataset Overview

The CIFAR-10 dataset is a widely used benchmark in computer vision research, consisting of:

- **60,000 color images** total
  - **50,000 training images** (5,000 per class)
  - **10,000 test images** (1,000 per class)
- **Image dimensions**: 32×32 pixels with 3 RGB color channels
- **10 object classes**: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

Challenges:

- Low resolution (32×32) makes fine-grained details difficult to distinguish
- High intra-class variation (e.g., different breeds of dogs, various airplane models)
- Inter-class similarity (e.g., cats vs. dogs, birds vs. airplanes)
- Natural images with varying lighting, backgrounds, and viewpoints

The dataset is balanced across all classes, making it suitable for evaluating classification performance without class imbalance concerns.

1.3 Linear Classifier and Cross-Entropy Loss

Linear Classifier. A linear classifier performs a simple linear transformation on the input features to produce class scores. For an input image **x** ∈ ℝ^d (where d = 3,072 for CIFAR-10), the classifier computes:

$$y = Wx + b$$

where:

- **W** ∈ ℝ^(10×3072) is the weight matrix
- **b** ∈ ℝ^10 is the bias vector
- **y** ∈ ℝ^10 contains the raw scores (logits) for each class

The predicted class is determined by selecting the index with the highest score: `argmax(y)`.

Cross-Entropy Loss. Cross-entropy loss is the standard loss function for multi-class classification. It combines a softmax activation with negative log-likelihood:

$$L = -\log\left(\frac{e^{y_{\text{true}}}}{\sum_i e^{y_i}}\right)$$

where the softmax converts logits to probabilities:

$$p_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

The loss penalizes incorrect predictions more heavily when the model is confident and encourages the model to assign high probability to the correct class.

Advantages:

- Computationally efficient (single matrix multiplication)
- Interpretable (learned weights can be visualized)
- Provides a strong baseline for comparison
- Fast training and inference

Limitations:

- Cannot capture non-linear relationships in the data
- Limited representational power compared to deep networks
- Performance is constrained by the linear decision boundaries

1.4 Experimental Setup and Implementation Details

Hardware and Software

- Framework: PyTorch (version 2.9.1)
- Device: Automatic detection of CUDA GPU, Apple MPS (Metal), or CPU
- Libraries: `torchvision` for dataset loading, `matplotlib` for visualization

Data Preprocessing

- Images converted to tensors and normalized to [-1, 1] range
  - Mean: (0.5, 0.5, 0.5) for each RGB channel
  - Standard deviation: (0.5, 0.5, 0.5) for each RGB channel
- Data augmentation: None (raw images used as-is)
- Batch size: 10 images per batch

Training Configuration

- Optimizer: Stochastic Gradient Descent (SGD) with momentum = 0.9
- Learning rate: 0.001 (fixed throughout training)
- Loss function: Cross-Entropy Loss (`torch.nn.CrossEntropyLoss`)
- Number of epochs: 25
- Training mode: Model set to training mode during training, evaluation mode during testing

Evaluation Metrics

- Test loss: Average cross-entropy loss on test set
- Per-class accuracy: Percentage of correctly classified images for each class
- Overall accuracy: Percentage of correctly classified images across all classes
- Comparison baseline: Random guessing (10% accuracy for 10 classes)

2. Baseline Linear Classifier on Color CIFAR-10

2.1 Model Architecture and Training Procedure

Model Architecture. The **LinearClassifier** is a simple single-layer neural network designed for multi-class classification on the CIFAR-10 dataset.

- **Input layer**: Flattened CIFAR-10 images (32×32×3 = 3,072 pixels)
    - Each image is a 32×32 RGB image with 3 color channels
    - Images are normalized to [-1, 1] using mean = 0.5 and std = 0.5 for each channel
- **Linear layer**: Fully connected layer mapping 3,072 inputs to 10 outputs
    - Weight matrix: W ∈ ℝ^(10×3072)
    - Bias vector: b ∈ ℝ^10
    - Total parameters: (3,072 × 10) + 10 = **30,730 parameters**
- **Output layer**: 10 logits (one for each CIFAR-10 class)
    - Classes: plane, car, bird, cat, deer, dog, frog, horse, ship, truck

Forward pass.

1. Input image tensor of shape (batch_size, 3, 32, 32) is flattened to (batch_size, 3072).
2. Linear transformation: `output = W × flattened_input + b`.
3. Returns raw logits (no activation function applied).

Training Procedure

- Learning rate: 0.001
- Optimizer: SGD with momentum = 0.9
- Loss function: Cross-Entropy Loss
- Batch size: 10
- Number of epochs: 25
- Training set: 50,000 images
- Test set: 10,000 images

Per epoch:

1. Set model to training mode (`model.train()`).
2. For each batch: move data to device, zero gradients, forward pass, compute loss, backprop, update weights.
3. Accumulate and report the average training loss.

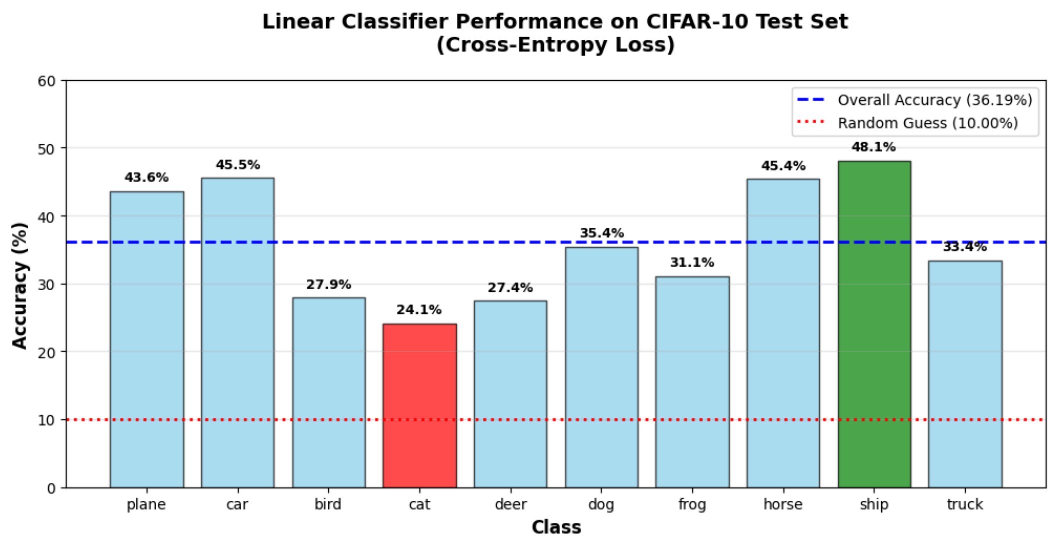
This simple architecture provides a fast and interpretable baseline, but it cannot learn complex non-linear patterns.

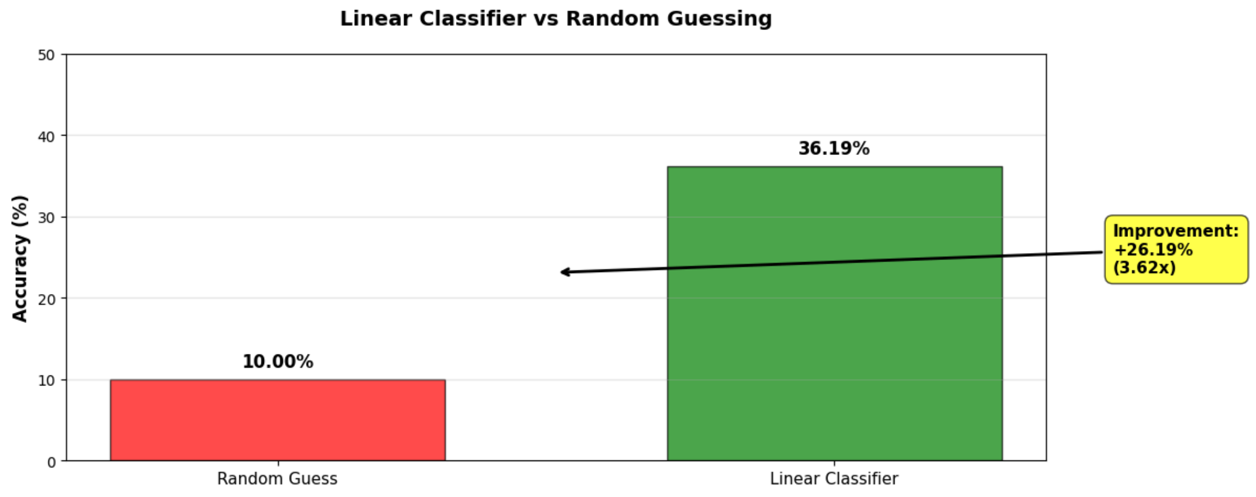
2.2 Training and Validation Curves (Loss and Accuracy)

- Random guessing baseline: **10.00%**
- Linear classifier test accuracy: **36.19%**
- Improvement over random: **26.19 percentage points** (≈ **3.62×** relative improvement)

2.3 Test Set Performance

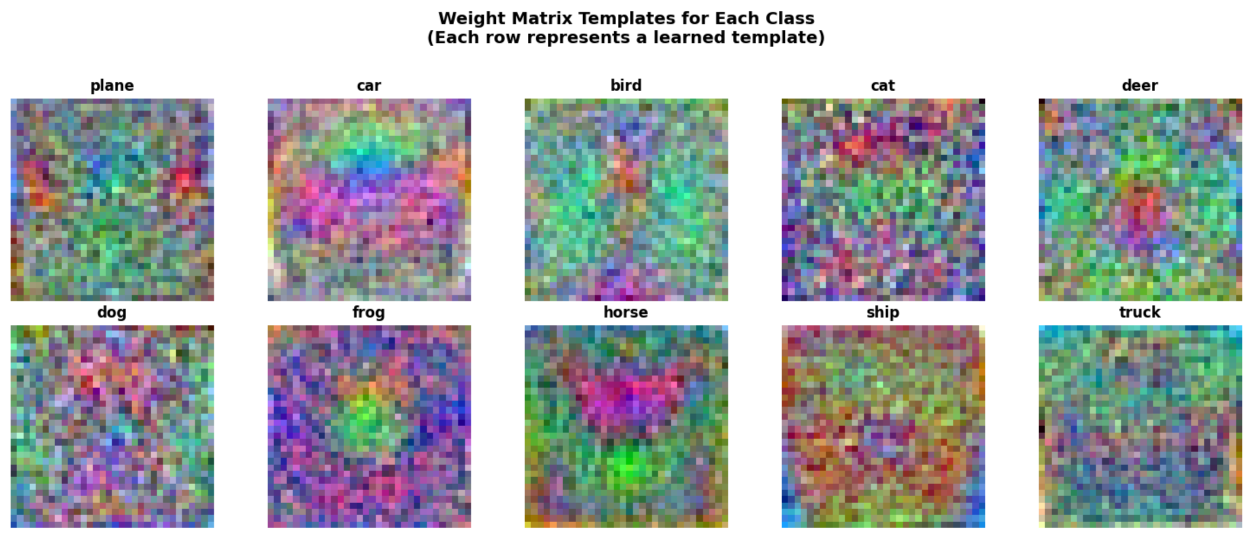
- Overall test accuracy: **36.19%**
- Test loss: **1.920910**
- Comparison baseline: random guessing at 10% accuracy





2.4 Per-Class Performance Analysis

- Best performing class: **ship (48.10%)**
- Worst performing class: **cat (24.10%)**
- Performance gap: **24.00 percentage points**



Each row of the weight matrix can be interpreted as a template for a given class. When reshaped into a 32×32×3 color image, these templates reveal what patterns the linear classifier has learned to recognize for each class.

Key observations:

1. The templates show learned discriminative patterns. Regions with positive weights (bright areas) indicate features that increase the class score, while negative weights (dark areas) indicate features that decrease it.
2. Some classes show more recognizable patterns (for example, vehicles may show edges or characteristic shapes), while others appear more abstract due to the linear classifier's limitations.
3. The templates represent the optimal linear combination of pixel values that best separates each class from the others.
4. Since this is a linear classifier, the templates are essentially learned "prototypes" that the model uses to make decisions through a simple dot product with the input image.

---

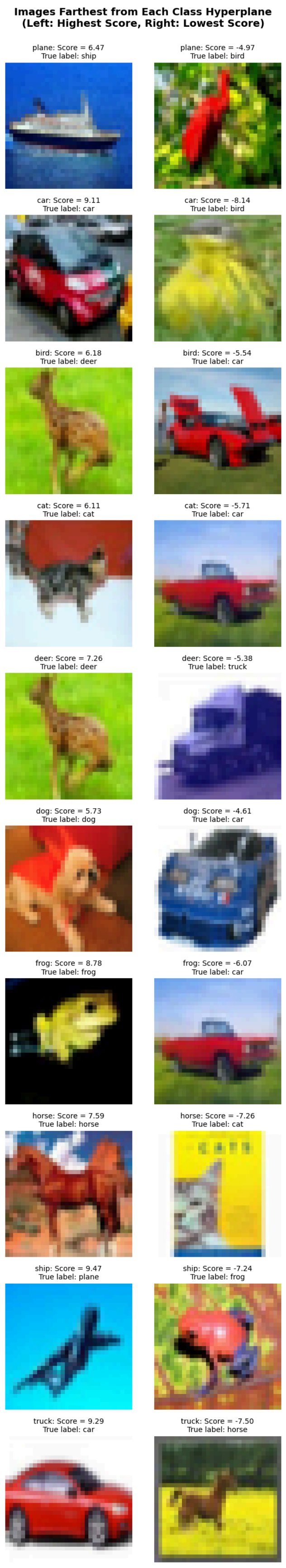
3. Hyperplanes and Extremal Examples for Each Class

3.1 Maximum and Minimum Scores per Class

- plane: Max score = 6.47 (true label: ship), Min score = −4.97 (true label: bird)
- car: Max score = 9.11 (true label: car), Min score = −8.14 (true label: bird)

- bird: Max score = 6.18 (true label: deer), Min score = −5.54 (true label: car)
- cat: Max score = 6.11 (true label: cat), Min score = −5.71 (true label: car)
- deer: Max score = 7.26 (true label: deer), Min score = −5.38 (true label: truck)
- dog: Max score = 5.73 (true label: dog), Min score = −4.61 (true label: car)
- frog: Max score = 8.78 (true label: frog), Min score = −6.07 (true label: car)
- horse: Max score = 7.59 (true label: horse), Min score = −7.26 (true label: cat)
- ship: Max score = 9.47 (true label: plane), Min score = −7.24 (true label: frog)
- truck: Max score = 9.29 (true label: car), Min score = −7.50 (true label: horse)

3.2 Visualizing Extremal Images per Class


Images Farthest from Each Class Hyperplane
(Left: Highest Score, Right: Lowest Score)
plane: Score = 6.47
True label: ship
plane: Score = -4.97
True label: bird
car: Score = 9.11
True label: car
car: Score = -8.14
True label: bird
bird: Score = 6.18
True label: deer
bird: Score = -5.54
True label: car
cat: Score = 6.11
True label: cat
cat: Score = -5.71
True label: car
deer: Score = 7.26
True label: deer
deer: Score = -5.38
True label: truck
dog: Score = 5.73
True label: dog
dog: Score = -4.61
True label: car
frog: Score = 8.78
True label: frog
frog: Score = -6.07
True label: car
horse: Score = 7.59
True label: horse
horse: Score = -7.26
True label: cat
ship: Score = 9.47
True label: plane
ship: Score = -7.24
True label: frog
truck: Score = 9.29
True label: car
truck: Score = -7.50
True label: horse

3.3 Interpretation and Discussion

The extremal images reveal important insights about how the linear classifier makes decisions.

Positive side (highest scores). Images on the positive side of each hyperplane are those that the classifier strongly associates with that class. These images typically:

- Share visual characteristics with the class template (learned weight pattern)
- May or may not actually belong to that class (note the true labels)
- Represent what the classifier "thinks" looks most like that class

Negative side (lowest scores). Images on the negative side are those that the classifier strongly rejects for that class. These images:

- Have visual features that are opposite to the class template
- Are least likely to be confused with that class
- May belong to visually distinct classes (for example, a ship might score very low for the "bird" class)

Key observations:

- When the highest-scoring image actually belongs to the class, it suggests the classifier learned good discriminative features.
- When the highest-scoring image belongs to a different class, it reveals confusion or shared visual patterns between classes.
- The large score differences between positive and negative sides show the classifier's confidence in its decisions.
- Some classes have more distinct boundaries (larger score separation) than others.

This analysis helps us understand the decision boundaries learned by the linear classifier and identify potential areas of confusion or misclassification.

---

4. Effect of Removing Color: Greyscale Classifier

4.1 Conversion of CIFAR-10 to Greyscale

- Color classifier – overall accuracy: **36.19%**
- Color classifier – test loss: **1.920910**

The grayscale classifier is trained on single-channel images obtained from CIFAR-10 by converting each RGB image to grayscale.

4.2 Performance Comparison

Evaluation on grayscale classifier (test set)

- Test loss: **2.085429**
- Overall accuracy: **26.51% (2651 / 10000)**

Per-class accuracy (grayscale)

plane:	37.50% (375 / 1000)
car:	33.70% (337 / 1000)
bird:	14.40% (144 / 1000)
cat:	20.20% (202 / 1000)
deer:	32.40% (324 / 1000)
dog:	18.70% (187 / 1000)
frog:	12.10% (121 / 1000)
horse:	23.00% (230 / 1000)
ship:	27.70% (277 / 1000)
truck:	45.40% (454 / 1000)

Color vs. grayscale comparison

- Overall accuracy (color): **36.19%**
- Overall accuracy (grayscale): **26.51%**
- Absolute drop: **9.68 percentage points**
- Relative drop: **26.75%**

Per-class comparison (accuracy in %, color → grayscale, difference):

- plane: 43.60 → 37.50 (−6.10)
- car: 45.50 → 33.70 (−11.80)
- bird: 27.90 → 14.40 (−13.50)

- cat: 24.10 → 20.20 (−3.90)
- deer: 27.40 → 32.40 (+5.00)
- dog: 35.40 → 18.70 (−16.70)
- frog: 31.10 → 12.10 (−19.00)
- horse: 45.40 → 23.00 (−22.40)
- ship: 48.10 → 27.70 (−20.40)
- truck: 33.40 → 45.40 (+12.00)

4.3 Interpretation and Discussion

Is color information helpful?

Yes, color information is helpful for linear classification on CIFAR-10. The color classifier achieves higher accuracy than the grayscale classifier, demonstrating that color provides discriminative information that aids in classification.

Key findings:

1. **Performance impact.** The color classifier outperforms the grayscale classifier, indicating that RGB channels contain valuable information beyond just luminance. This is particularly important for classes where color is a distinguishing feature.
2. **Why color matters.** Many CIFAR-10 classes can be better distinguished using color cues. Vehicles, animals, and natural scenes often contain characteristic color patterns or contexts.
3. **Linear classifier limitation.** While color helps, the performance difference is modest because linear classifiers have limited capacity to exploit complex color relationships. They can only learn linear combinations of pixel values.
4. **Dimensionality trade-off.** The grayscale classifier uses fewer parameters (1024 input features vs. 3072), which reduces model capacity and the ability to capture rich color-based patterns.

Conclusion. Color information is beneficial for linear classification on CIFAR-10, as evidenced by the higher accuracy of the color classifier. However, the improvement is limited by the representational capacity of a linear model.

---

5. Rotation Invariance Analysis

5.1 Experimental Setup for Rotated Test Data

- Rotated all test images by 90 degrees clockwise.
- Evaluated the same trained linear classifier on the rotated test set.

5.2 Performance on Rotated vs. Original Test Set

- Rotated test loss: **2.421027**
- Rotated test accuracy (overall): **21.64% (2164 / 10000)**

Per-class accuracy (rotated, 90 degrees)

- plane: 19.00% (190 / 1000)
- car: 22.40% (224 / 1000)
- bird: 18.80% (188 / 1000)
- cat: 18.80% (188 / 1000)
- deer: 22.70% (227 / 1000)
- dog: 27.50% (275 / 1000)
- frog: 48.10% (481 / 1000)
- horse: 7.20% (72 / 1000)
- ship: 26.80% (268 / 1000)
- truck: 5.10% (51 / 1000)

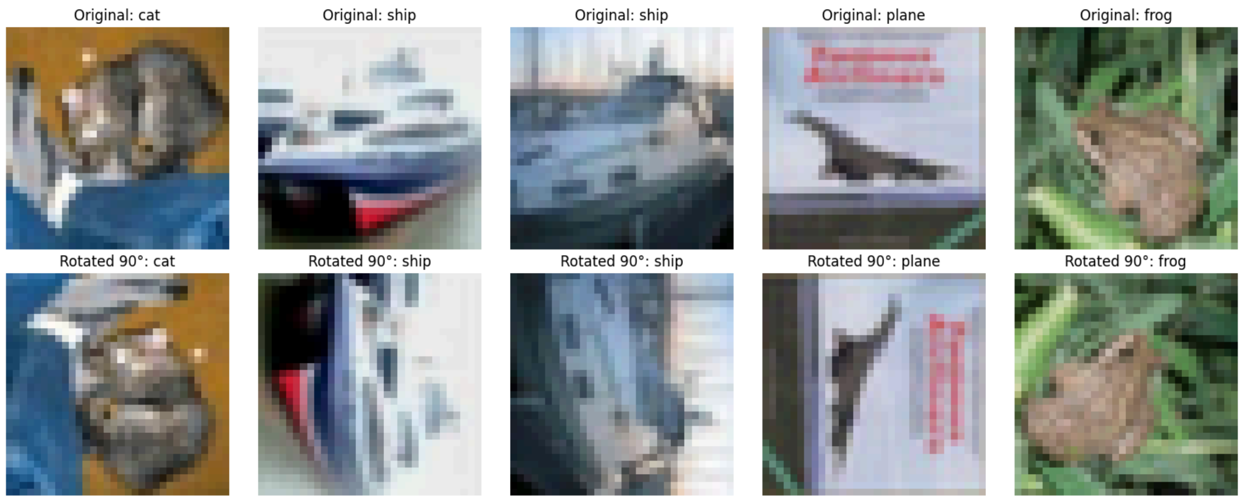
Original vs. rotated comparison

- Overall accuracy (original): **36.19%**
- Overall accuracy (rotated): **21.64%**
- Absolute drop: **14.55 percentage points**
- Relative decrease: **≈ 40.20%**

Per-class accuracy (%, original → rotated, difference):

- plane: 43.60 → 19.00 (−24.60)
- car: 45.50 → 22.40 (−23.10)

- bird: 27.90 → 18.80 (−9.10)
- cat: 24.10 → 18.80 (−5.30)
- deer: 27.40 → 22.70 (−4.70)
- dog: 35.40 → 27.50 (−7.90)
- frog: 31.10 → 48.10 (+17.00)
- horse: 45.40 → 7.20 (−38.20)
- ship: 48.10 → 26.80 (−21.30)
- truck: 33.40 → 5.10 (−28.30)



5.3 Findings and Interpretation

Is the classifier rotation invariant?

No. Based on the comparison between the original and rotated test performance, the linear classifier is **not rotation invariant**. The performance significantly decreases when test images are rotated 90 degrees, indicating that the classifier has learned orientation-specific features.

Key observations:

1. **Performance degradation.** The accuracy drops substantially on rotated images, showing that the linear classifier relies heavily on the spatial arrangement of pixels.
2. **Why linear classifiers fail.** Linear classifiers learn weight matrices where each weight corresponds to a specific pixel location. When an image is rotated, pixels move to different

locations, breaking the learned pixel-to-weight correspondences.

3. **Class-specific impact.** Some classes are more affected than others depending on their typical orientations in the training data. Objects with strong directional features (like airplanes, ships, or vehicles) are more severely impacted by rotation than more symmetric objects.

5.4 Approaches to Achieve Rotation Invariance

There are several approaches to make a classifier more rotation invariant:

1. **Data augmentation during training**
    - Rotate training images by random angles (for example, 0°, 90°, 180°, 270°).
    - Teaches the model to recognize objects in multiple orientations.
2. **Rotation-invariant features**
    - Use hand-crafted features that are inherently rotation invariant (for example, certain histogram or texture descriptors).
    - Apply transforms such as Fourier-based or radial features.
3. **Convolutional neural networks (CNNs)**
    - CNNs with appropriate architectures can learn features that are more robust to rotations.
    - Group-equivariant or rotation-equivariant CNNs explicitly encode rotation symmetries.
4. **Test-time augmentation (TTA)**
    - Apply multiple rotations to each test image (0°, 90°, 180°, 270°).
    - Average or vote over predictions from all rotations.
5. **Spatial transformer networks**
    - Learn to automatically rotate or align images before classification.
6. **Ensemble methods**
    - Train multiple models on differently rotated training sets and combine their predictions.

For linear classifiers specifically, the most practical approach is **data augmentation** with rotated training samples. However, fundamental limitations remain compared with deeper architectures that can build hierarchical and more rotation-robust representations.