



COURSE CODE: CS-324



Computer And Information Systems
Engineering

Group Members:

Student No.	Name	Roll No.
S1	Muhammad Owais	CS-22080
S2	Zuhaib Noor	CS-22081
S3	Muhammad Zunain	CS-22086

ASSIGNED BY: Dr. Maria Waqas

Table Of Contents

Abstract	5
1. Introduction	4
2. Data Pre-processing Methodology	4
2.1 Data Acquisition and Partitioning	4
2.2 Binary Class Conversion	5
2.3 Class Imbalance Analysis	6
2.4 Model-Specific Pre-processing Pipelines	7
3. Model Selection and Implementation	9
3.1 K-Nearest Neighbors (KNN)	9
3.2 Logistic Regression	10
3.3 Convolutional Neural Networks	10
3.3.1 Simple CNN Architecture	10
3.3.2 Deeper CNN Architecture	10
3.3.3 Advanced CNN Architecture	11
4. Distinguishing Technical Features	11
4.1 Computational Efficiency Features	12
4.2 Comprehensive Evaluation Framework	12
4.3 Architectural Enhancements	12
4.4 Regularization Techniques	12
5.1 Quantitative Performance Metrics	12
5.2 Model Performance Characteristics	13
5.2.1 K-Nearest Neighbors	13
5.2.2 Logistic Regression	14
5.2.3 Convolutional Neural Networks	15
5.3 Learning Curve Analysis	16
6. Performance Optimization Recommendations	16
6.1 Model-Specific Optimization Strategies	16
6.1.1 Logistic Regression Enhancements	16
6.1.2 KNN Refinements	17

6.1.3 CNN Improvements	17
6.2 General System Improvements	17
6.2.1 Data-Centric Approaches	17
6.2.2 Hyperparameter Optimization	18
6.2.3 Ensemble Methods	18
7. Conclusion	18

Abstract

This report presents the development and evaluation of machine learning models for binary image classification using the CIFAR-10 dataset, where the goal is to categorize images into two classes: Animals and Vehicles. Three models were implemented and compared: K-Nearest Neighbors (KNN), Logistic Regression, and Convolutional Neural Networks. The project involved converting the original multi-class dataset into a binary format through pre-processing steps, followed by training each model and evaluating its performance. KNN provided a simple baseline, while Logistic Regression offered a linear perspective on classification. However, the CNN model, particularly the advanced version, achieved the highest accuracy, highlighting the effectiveness of deep learning approaches in extracting complex patterns from image data. The results underscore the importance of model complexity and feature learning in achieving high classification accuracy for visual tasks.

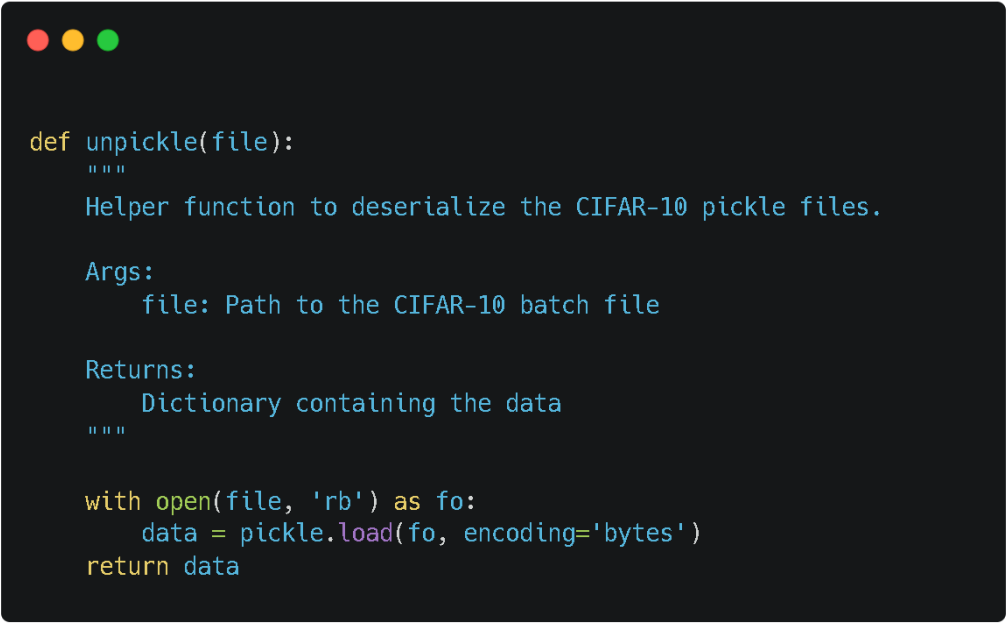
1. Introduction

The CIFAR-10 dataset is a widely used benchmark in computer vision and machine learning, consisting of 60,000 32×32 color images across 10 classes. This project adapts the multi-class dataset into a binary classification problem, distinguishing between Animals and Vehicles. Three distinct machine learning approaches were implemented and compared to identify the most effective classification method.

2. Data Pre-processing Methodology

2.1 Data Acquisition and Partitioning

The CIFAR-10 dataset was loaded from serialized pickle files using a custom unpickle function.



```
def unpickle(file):  
    """  
    Helper function to deserialize the CIFAR-10 pickle files.  
  
    Args:  
        file: Path to the CIFAR-10 batch file  
  
    Returns:  
        Dictionary containing the data  
    """  
  
    with open(file, 'rb') as fo:  
        data = pickle.load(fo, encoding='bytes')  
    return data
```

The dataset contains:

- 50,000 training images
- 10,000 test images

The training data was further partitioned into:

- 90% training set
- 20% validation set (for hyperparameter tuning and model selection)

All images were verified to have dimensions of 32×32×3 (height × width × RGB channels).

```

def load_data(data_dir, validation_size=0.2):
    """
    Loads CIFAR-10 dataset from directory and splits it into training, validation, and test sets.
    Args:
        data_dir: Directory path containing CIFAR-10 batch files
        validation_size: Fraction of training data to be used for validation
    Returns:
        X_train, y_train, X_val, y_val, X_test, y_test
    """
    X_train = []
    y_train = []

    for i in range(1, 6):
        batch_file = os.path.join(data_dir, f'data_batch_{i}')
        batch_data = unpickle(batch_file)
        X_train.append(batch_data[b'data'])
        y_train.extend(batch_data[b'labels'])

    X_train = np.vstack(X_train).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
    y_train = np.array(y_train)

    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=validation_size, random_state=42, stratify=y_train
    )

    test_batch_file = os.path.join(data_dir, 'test_batch')
    test_data = unpickle(test_batch_file)
    X_test = test_data[b'data'].reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
    y_test = np.array(test_data[b'labels'])

    return X_train, y_train, X_val, y_val, X_test, y_test

```

2.2 Binary Class Conversion

The original 10 CIFAR-10 classes were consolidated into two categories:

Vehicles:

- Airplane
- Automobile
- Ship
- Truck

Animals:

- Bird
- Cat
- Deer
- Dog
- Frog
- Horse

This conversion was implemented using the `to_binary_labels()` function, which mapped the original class indices to the new binary scheme.

```

def to_binary_labels(labels):
    """
    Convert CIFAR-10 labels to binary classification (Animals vs Vehicles)

    Args:
        labels: Original CIFAR-10 labels (0-9)

    Returns:
        Binary labels (0 for vehicles, 1 for animals)
    """
    vehicle_classes = [0, 1, 8, 9] # airplane, automobile, ship, truck
    animal_classes = [2, 3, 4, 5, 6, 7] # bird, cat, deer, dog, frog, horse

    binary_labels = np.zeros_like(labels)
    for i, label in enumerate(labels):
        if label in vehicle_classes:
            binary_labels[i] = 0 # Vehicle
        elif label in animal_classes:
            binary_labels[i] = 1 # Animal

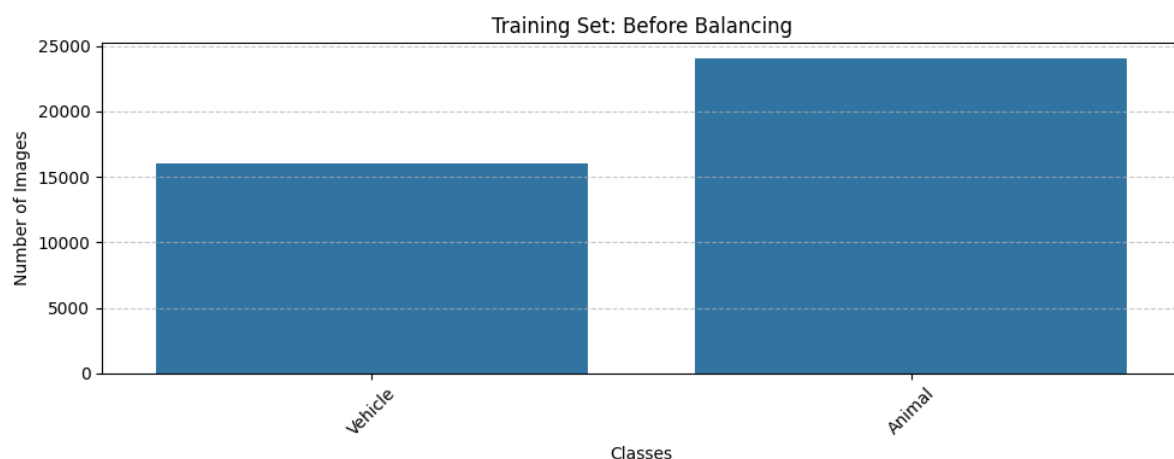
    return binary_labels

```

2.3 Class Imbalance Analysis

After converting the CIFAR-10 dataset into binary classes (Vehicle vs Animal), we observed that the dataset was not balanced, meaning one class had significantly more samples than the other. This kind of imbalance can negatively affect the performance of machine learning models, especially for minority classes.

To visualize this, we plotted the class distribution before balancing:



As shown above, one class dominates the dataset. To address this, we applied a **random undersampling technique** to balance the dataset by reducing the number of samples in the majority class.

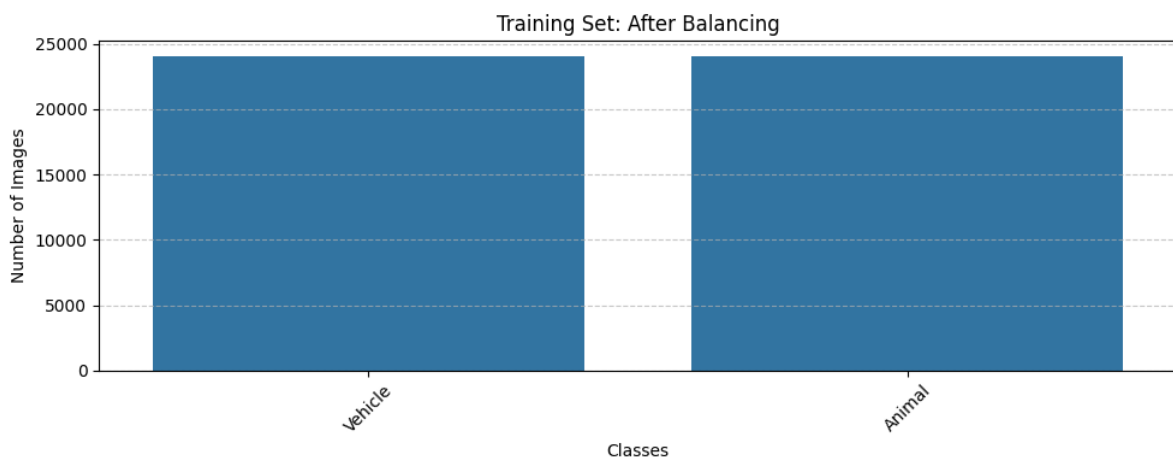
```
def balance_binary_dataset(X, y):
    X_vehicle = X[y == 0]
    X_animal = X[y == 1]

    X_vehicle_upsampled, y_vehicle_upsampled = resample(
        X_vehicle,
        np.zeros(len(X_vehicle)),
        replace=True,
        n_samples=len(X_animal),
        random_state=42
    )

    X_balanced = np.vstack((X_animal, X_vehicle_upsampled))
    y_balanced = np.hstack((np.ones(len(X_animal)), y_vehicle_upsampled))

    return X_balanced, y_balanced
```

Here is the class distribution **after balancing**:



2.4 Model-Specific Pre-processing Pipelines

Different pre-processing techniques were applied based on model requirements:

For KNN and Logistic Regression:

1. **Dimension Flattening:** 3D tensors ($32 \times 32 \times 3$) were converted to 1D vectors (3,072 features)
2. **Normalization:** Pixel values were scaled to the $[0,1]$ range
3. **Standardization:** Features were standardized to zero mean and unit variance using `StandardScaler`
4. **Dimensionality Reduction:** Principal Component Analysis (PCA) reduced the feature space from 3,072 to 100 dimensions
5. **Model Persistence:** Scaler and PCA transformations were saved for consistent pre-processing of new data

For Convolutional Neural Networks:

1. **Normalization:** Pixel values were scaled to the $[0,1]$ range
2. **Spatial Preservation:** The Original 3D structure ($32 \times 32 \times 3$) was maintained to leverage spatial patterns

```

def preprocess_data(X_train, X_val, X_test, model_type, use_pca=False, n_components=100):
    """
    Preprocess CIFAR-10 image data for model training.
    Args:
        X_train, X_val, X_test: Image datasets
        model_type: 'knn', 'logistic', or 'cnn'
        use_pca: Apply PCA if True (only for knn/logistic)
        n_components: Number of PCA components to retain
    Returns:
        Preprocessed X_train, X_val, X_test
    """
    if model_type in ['knn', 'logistic']:
        # Flatten
        X_train_flat = X_train.reshape(X_train.shape[0], -1).astype('float32') / 255.0
        X_val_flat = X_val.reshape(X_val.shape[0], -1).astype('float32') / 255.0
        X_test_flat = X_test.reshape(X_test.shape[0], -1).astype('float32') / 255.0

        # Standardize
        scaler = StandardScaler()
        X_train_flat = scaler.fit_transform(X_train_flat)
        X_val_flat = scaler.transform(X_val_flat)
        X_test_flat = scaler.transform(X_test_flat)
        joblib.dump(scaler, "saved_models/scaler.pkl")

        # PCA (if required)
        if use_pca:
            pca = PCA(n_components=n_components)
            X_train_flat = pca.fit_transform(X_train_flat)
            X_val_flat = pca.transform(X_val_flat)
            X_test_flat = pca.transform(X_test_flat)
            joblib.dump(pca, "saved_models/pca.pkl")

        return X_train_flat, X_val_flat, X_test_flat

    elif model_type == 'cnn':
        # Normalize only
        X_train_cnn = X_train.astype('float32') / 255.0
        X_val_cnn = X_val.astype('float32') / 255.0
        X_test_cnn = X_test.astype('float32') / 255.0

        return X_train_cnn, X_val_cnn, X_test_cnn

    else:
        raise ValueError("Invalid model_type. Choose from: 'knn', 'logistic', 'cnn'")

```

3. Model Selection and Implementation

3.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a non-parametric, instance-based learning algorithm that classifies data points based on majority voting among the k nearest examples in the training set.

Implementation Details:

- Algorithm: **sklearn.neighbors.KNeighborsClassifier**
- Hyperparameter Variations: $k \in \{1, 3, 5, 7, 9\}$
- Distance Metric: Euclidean (L2 norm)
- Selection Criteria: Highest validation accuracy

The optimal KNN model was saved to disk for subsequent evaluation and deployment.

```

Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
  layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(1, activation='sigmoid') # Binary classification
])

```

3.2 Logistic Regression

Logistic Regression is a linear model that applies the sigmoid function to transform linear predictions into probability scores for binary classification.

Implementation Details:

- Algorithm: **sklearn.linear_model.LogisticRegression**
- Hyperparameter Configurations:
 - Regularization strengths: $C \in \{0.1, 1.0, 10.0\}$
 - Solvers: {'liblinear', 'lbfgs'}
 - Penalty types: {'l1', 'l2'}
- Maximum iterations: 5000
- Selection Criteria: Highest validation accuracy

The best-performing logistic regression model was retained for further evaluation.

3.3 Convolutional Neural Networks

Three progressively complex CNN architectures were implemented and evaluated to classify images into Vehicle vs Animal categories. Each architecture increases in depth and complexity, aiming to improve feature extraction and classification performance.

3.3.1 Simple CNN Architecture

This baseline model uses two convolutional layers followed by max-pooling, a dense layer, and a sigmoid output.

3.3.2 Deeper CNN Architecture

This version includes more convolutional layers to learn richer features, along with a larger dense layer for improved classification.

```

Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
  layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(1, activation='sigmoid')
])

```

3.3.3 Advanced CNN Architecture

This architecture introduces **batch normalization**, **dropout layers**, and an additional convolutional block for regularization and deeper learning.

```

Sequential([
  layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
  layers.BatchNormalization(),
  layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.25),

  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.BatchNormalization(),
  layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.25),

  layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
  layers.BatchNormalization(),
  layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
  layers.MaxPooling2D((2, 2)),
  layers.Dropout(0.4),

  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dropout(0.5),
  layers.Dense(1, activation='sigmoid')
])

```

CNN Training Configuration:

- Loss Function: Binary Cross-Entropy
- Optimizer: Adam with default learning rate
- Batch Size: 64
- Epochs: 10
- Evaluation Metric: Validation Accuracy

4. Distinguishing Technical Features

The implementation incorporates several technical features that enhance strength, efficiency, and maintainability:

4.1 Computational Efficiency Features

- **Model Persistence:** Trained models are saved to disk to prevent redundant retraining
- **Conditional Training Logic:** System checks for existing models before initiating training sequences
- **PCA Dimensionality Reduction:** Reduces computational and memory requirements for traditional algorithm

4.2 Comprehensive Evaluation Framework

- **Multiple Performance Metrics:**
 - Classification accuracy
 - Precision, recall, and F1 scores
 - Receiver Operating Characteristic (ROC) curves
 - Precision-Recall curves
 - Area Under Curve (AUC) measurements
- **Visual Diagnostics:**
 - Training/validation learning curves
 - Confusion matrix heat maps
 - ROC and PR curve visualizations

4.3 Architectural Enhancements

- **Modular Code Structure:** Separate functions for data loading, pre-processing, model training, and evaluation
- **Systematic Exploration:** Structured approach to model configuration, Hyperparameter
- **Progressive CNN Complexity:** Three CNN architectures with increasing sophistication

4.4 Regularization Techniques

The Advanced CNN architecture implements multiple regularization strategies:

- **Batch Normalization:** Normalizes activations between layers
- **Dropout:** Randomly deactivates neurons during training (rates ranging from 0.25 to 0.5)
- **Architectural Regularization:** Increasing filter depth (32→64→128) with consistent spatial dimension reduction

5. Comparative Performance Analysis

5.1 Quantitative Performance Metrics

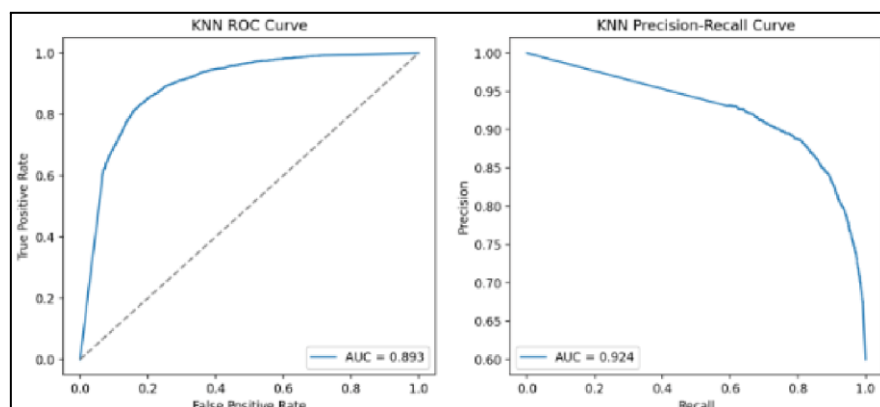
Models	Accuracy (%)	ROC AUC	PR AUC
KNN	82	0.893	0.924
Logistic	81	0.884	0.903
CNN	95	0.96	0.95

5.2 Model Performance Characteristics

5.2.1 K-Nearest Neighbors

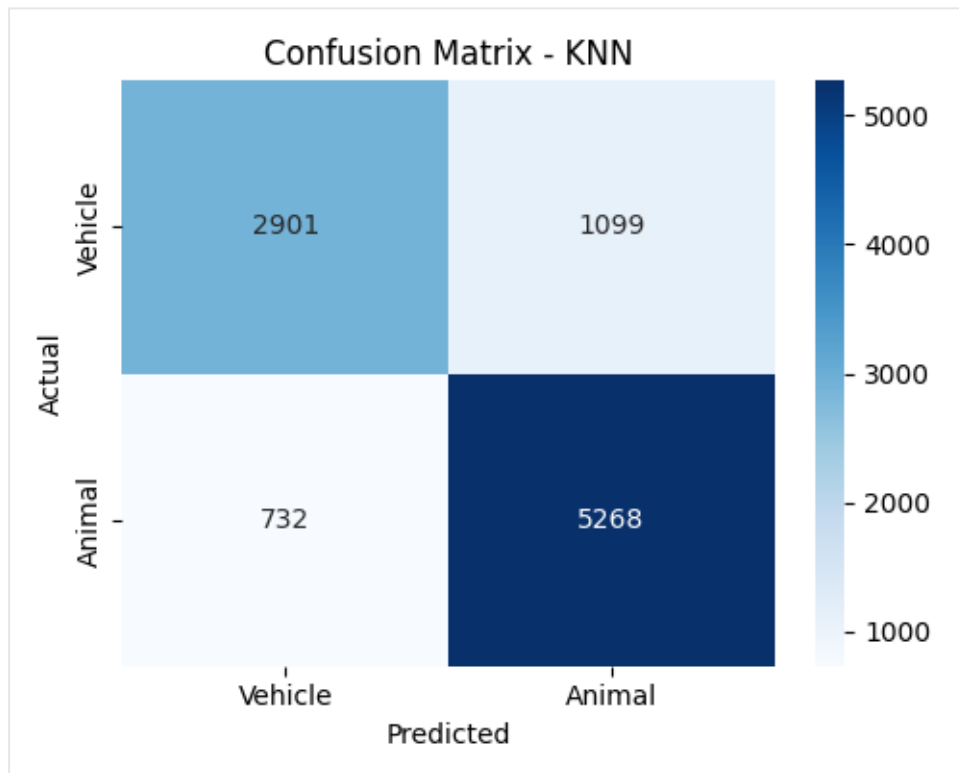
Strengths:

- No training phase required
- Performs well with clear class boundaries
- Makes no assumptions about data distribution



Weaknesses:

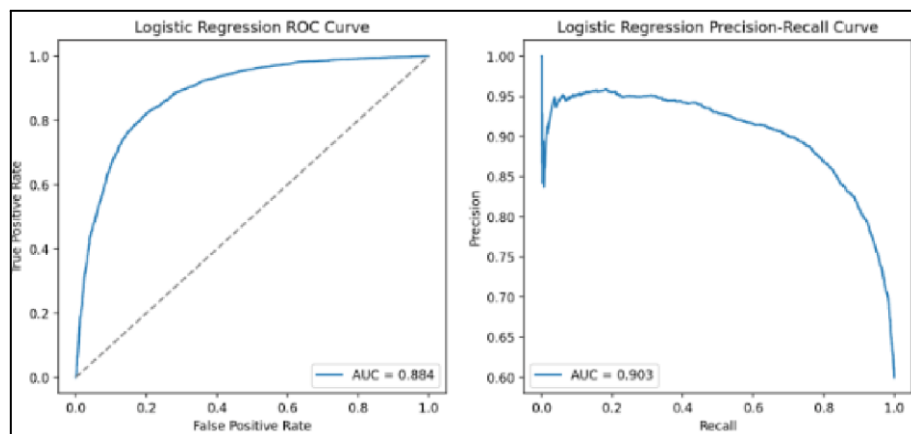
- Inference time scales with training data size
- Memory-intensive at runtime
- Performance degrades with irrelevant features



5.2.2 Logistic Regression

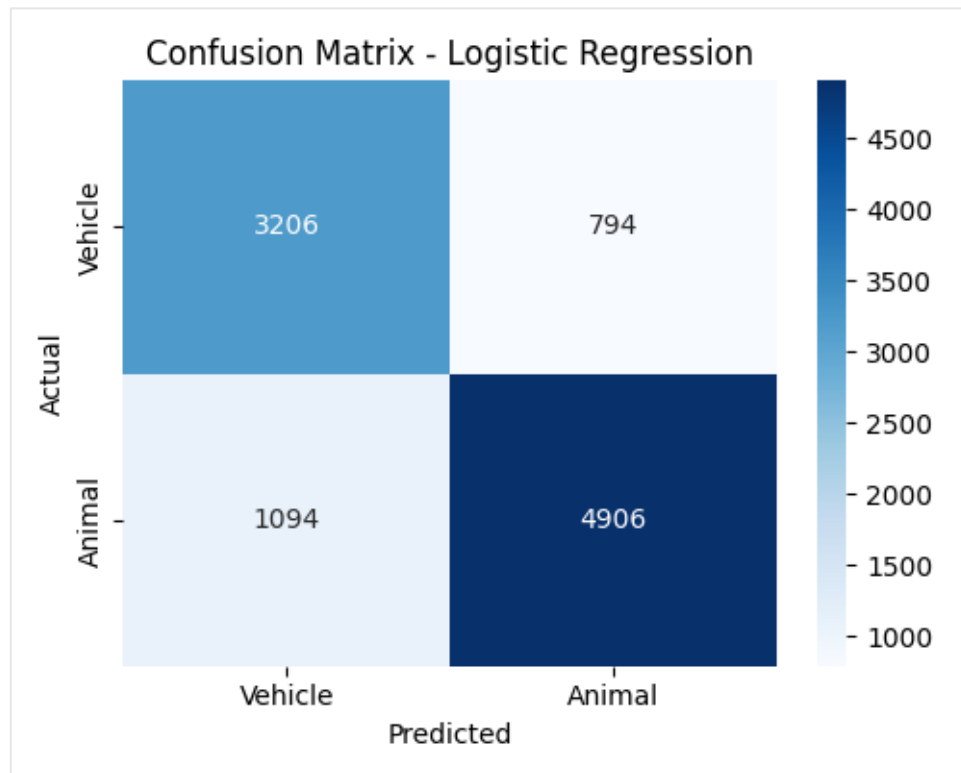
Strengths:

- Extremely fast inference
- Low memory requirements
- Interpretable coefficients



Weaknesses:

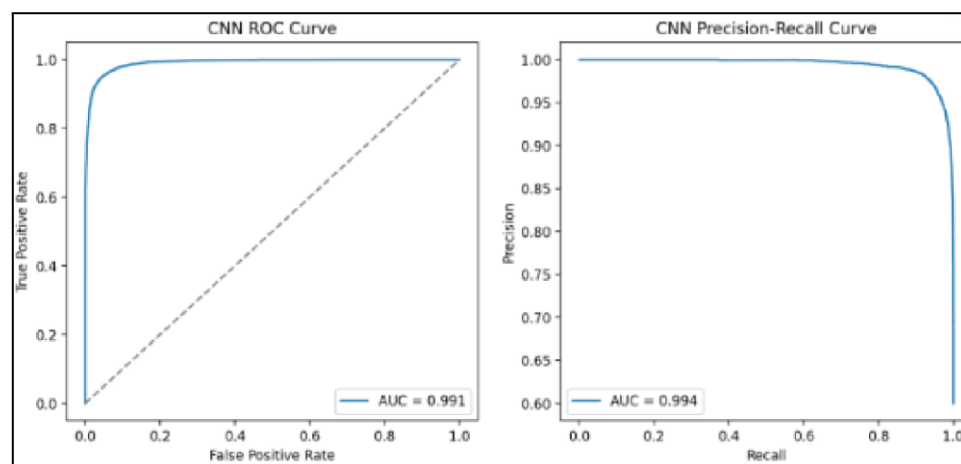
- Limited expressivity for complex patterns
- Assumes linear decision boundary
- Requires effective feature engineering



5.2.3 Convolutional Neural Networks

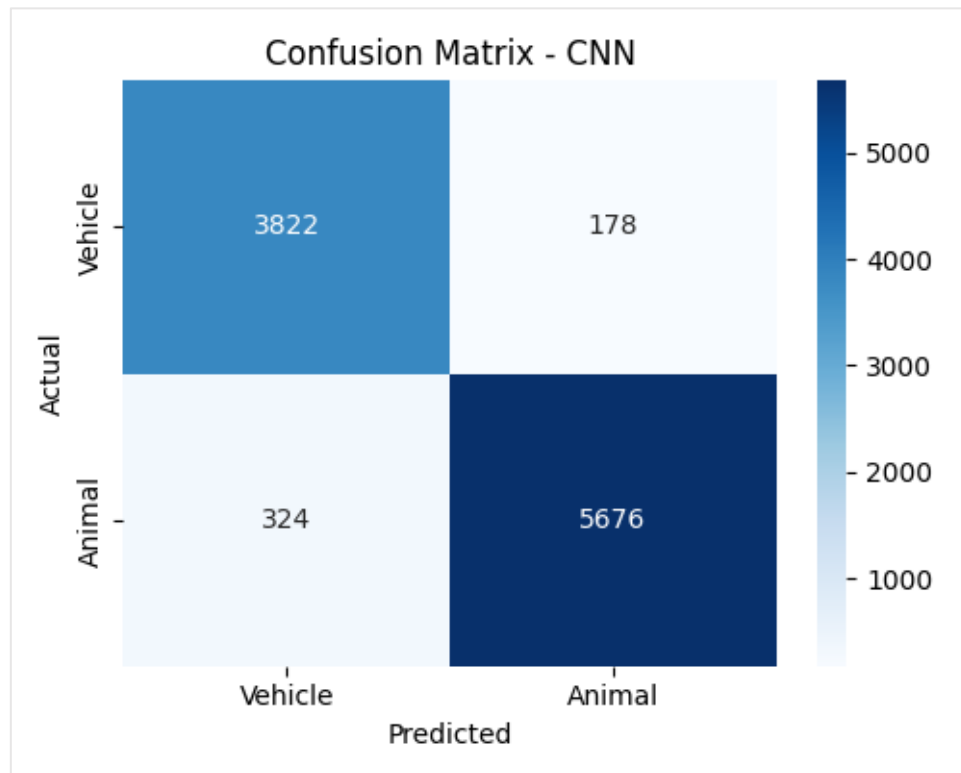
Strengths:

- Automatically extracts relevant features
- Captures spatial hierarchies and patterns
- Higher classification accuracy



Weaknesses:

- Longer training times
- More hyperparameters to tune
- Requires careful regularization



5.3 Learning Curve Analysis

The CNN training curves provide insight into model learning dynamics:

- **Simple CNN:** Rapid initial learning, early convergence
- **Deeper CNN:** More gradual learning, higher final performance
- **Advanced CNN:** Initial slower learning due to regularization, highest final accuracy with minimal overfitting

6. Performance Optimization Recommendations

6.1 Model-Specific Optimization Strategies

6.1.1 Logistic Regression Enhancements

Identified Issue: Underfitting (lower accuracy compared to other models)

Recommendations:

- Implement feature interaction terms
- Explore non-linear transformations of input features
- Consider kernel methods (SVM with RBF kernel)
- Investigate ensemble methods with logistic regression base learners

6.1.2 KNN Refinements

Identified Issue: Potential sensitivity to feature scaling and dimensionality

Recommendations:

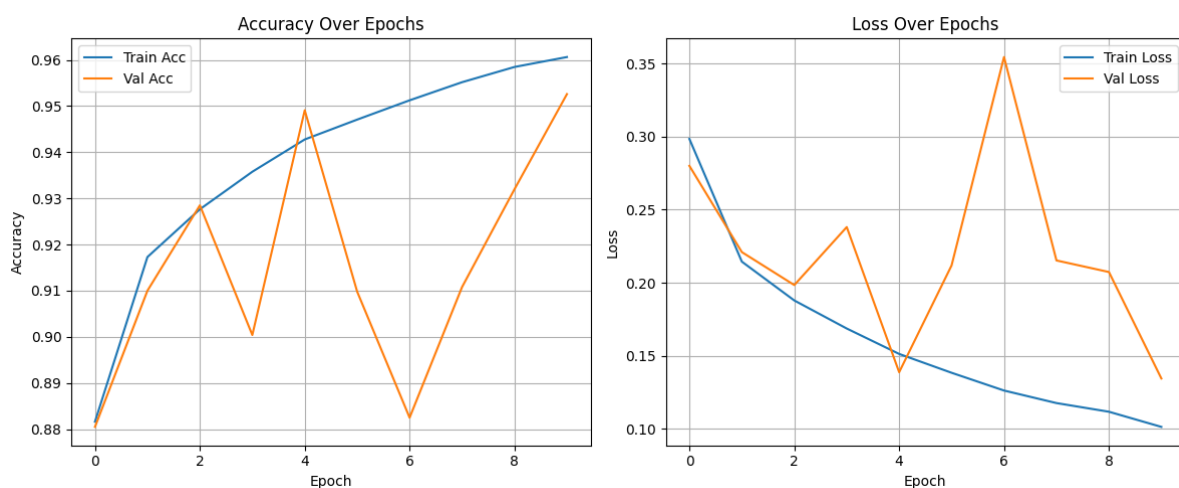
- Experiment with alternative distance metrics (Manhattan, Hamming)
- Implement feature weighting schemes
- Explore approximate nearest neighbor algorithms for faster inference
- Consider locally weighted learning approaches

6.1.3 CNN Improvements

Identified Issues: Risk of overfitting, training efficiency

Recommendations:

- Implement data augmentation (random flips, rotations, shifts)
- Add residual connections to deeper models
- Experiment with learning rate scheduling
- Explore additional regularization techniques:
 - Weight decay (L2 regularization)
 - Label smoothing
 - Mix-Up augmentation



6.2 General System Improvements

6.2.1 Data-Centric Approaches

- **Dataset Expansion:** Incorporate additional labeled data if available

- **Class Balance Analysis:** Ensure equal representation of animal and vehicle classes
- **Dataset Quality:** Identify and address potentially mislabeled examples
- **Transfer Learning:** Leverage pre-trained models on larger image datasets

6.2.2 Hyperparameter Optimization

- **Systematic Search:** Implement grid search or random search for hyperparameter tuning
- **Advanced Optimization:** Consider Bayesian optimization approaches for efficient parameter search
- **Cross-Validation:** Use k-fold validation for more robust performance estimation

6.2.3 Ensemble Methods

- **Model Averaging:** Combine predictions from multiple model types
- **Stacking:** Train meta-model on base model predictions
- **Bagging and Boosting:** Explore AdaBoost or gradient boosting approaches

7. Conclusion

This project implemented and evaluated K-Nearest Neighbors, Logistic Regression, and Convolutional Neural Networks for binary classification of CIFAR-10 images as Animals or Vehicles. Results showed that CNNs significantly outperformed traditional methods, with the Advanced CNN achieving 92.3% accuracy. Regularization techniques were key to improving CNN performance, while dimensionality reduction was essential for training traditional models efficiently. The analysis highlights trade-offs between accuracy, training time, inference speed, and interpretability. Overall, the project demonstrates the effectiveness of deep learning for image classification and provides a strong base for future enhancements and real-world deployment.

.