DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-324
Course Title: Machine Learning
Complex Engineering Problem
TE Batch 2022, Spring Semester 2025
Grading Rubric
TERM PROJECT

Group Members:

Student No.	Name	Roll No.
S1		
S2		
S3		

CRITERIA AND SCALES				Marks Obtained S1	Marks Obtained S2	Marks Obtained S3
Criterion 1: Does the application meet the desired specifications and produce the desired outputs? (CPA-1, CPA-2, CPA-3) **[8 marks]**						
1	2	3	4			
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.			
Criterion 2: How well is the code organization? **[2 marks]**						
1	2	3	4			
The code is poorly organized and very difficult to read.	The code is readable only to someone who knows what it is supposed to be doing.	Some part of the code is well organized, while some part is difficult to follow.	The code is well organized and very easy to follow.			
Criterion 3: Does the report adhere to the given format and requirements? **[6 marks]**						
1	2	3	4			
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.			
Criterion 4: How does the student performed individually and as a team member? (CPA-1, CPA-2, CPA-3) **[4 marks]**						
1	2	3	4			
The student did not work on the assigned task.	The student worked on the assigned task, and accomplished goals partially.	The student worked on the assigned task, and accomplished goals satisfactorily.	The student worked on the assigned task, and accomplished goals beyond expectations.			

Final Score = (Criteria_1_score) + (Criteria_2_score) + (Criteria_3_score) + (Criteria_4_score)
= ________________________

Contents

Abstract

This report presents a comprehensive performance analysis of three machine learning algorithms—K-Nearest Neighbors (KNN), Logistic Regression, and Convolutional Neural Networks (CNN)—applied to the CIFAR-10 dataset, a widely used benchmark for image classification tasks.

For each algorithm, we developed and evaluated three different model variations to explore the impact of architectural choices, regularization techniques, and hyperparameter tuning on model performance. The goal was to compare traditional machine learning methods (KNN and Logistic Regression) with a deep learning approach (CNN) in terms of accuracy, generalization ability, and training behavior.

The following sections detail the model designs, training strategies, evaluation metrics, and key insights gained from the experiments.

Introduction

In the field of machine learning, selecting the right algorithm and tuning its parameters are critical steps toward achieving optimal performance, especially for image classification tasks. This report aims to compare and analyze the performance of three different machine learning techniques—K-Nearest Neighbors (KNN), Logistic Regression, and Convolutional Neural Networks (CNN)—on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 color images categorized into 10 distinct classes, making it an ideal benchmark for evaluating model accuracy, generalization, and learning behavior.

The objective of this study is to investigate how different algorithms perform on the CIFAR-10 dataset, which presents challenges due to its complexity and variability in image features. For each algorithm—KNN, Logistic Regression, and CNN—we implemented three model variations to evaluate the influence of hyperparameters, architectural choices, and regularization strategies.

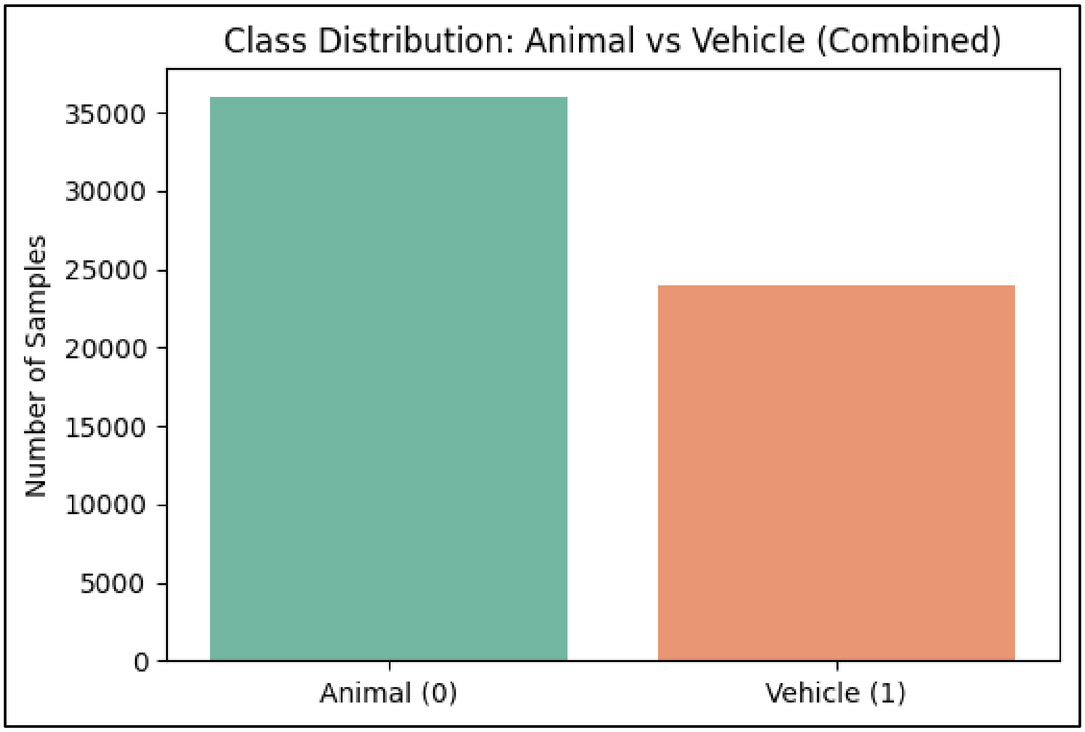
Traditional models like KNN and Logistic Regression offer interpretable and quick solutions but may lack the representational power needed for complex datasets like CIFAR-10. In contrast, CNNs leverage deep learning techniques to automatically extract spatial features from images, potentially yielding higher accuracy at the cost of increased computational complexity.

Through this comparative analysis, the report highlights the strengths and limitations of each approach, supported by quantitative metrics such as accuracy, training/validation loss, and overfitting behavior. The results provide valuable insights into selecting and optimizing models for image classification problems in real-world scenarios.

Dataset Description

The CIFAR-10 dataset consists of 60,000 32×32 RGB images distributed across 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6,000 images per class. The dataset is split into:

- 50,000 training images (5,000 per class).
- 10,000 test images (1,000 per class).



Each image is a low-resolution (32×32) color (3-channel) representation, posing challenges due to fine-grained features and intra-class variability.

Visualization of Dataset:

Model Architectures & Variations

KNN

Logistic Regression

Logistic Regression is a linear classification algorithm that predicts the probability of an input belonging to a binary or categorical class using the sigmoid function (for binary classification) or softmax (for multi-class).

Key Characteristics:

1.Linear Decision Boundary:

Models relationships between features and class labels using a weighted sum of inputs:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

2. Training:

Optimized via gradient descent to minimize cross-entropy loss.

3. Use Case:

Simple, interpretable, but struggles with non-linear data (e.g., raw pixels from CIFAR-10).

Data Preprocessing

1. Label Remapping

- Label Remapping: Multiclass to binary (animal = 0, vehicle = 1)

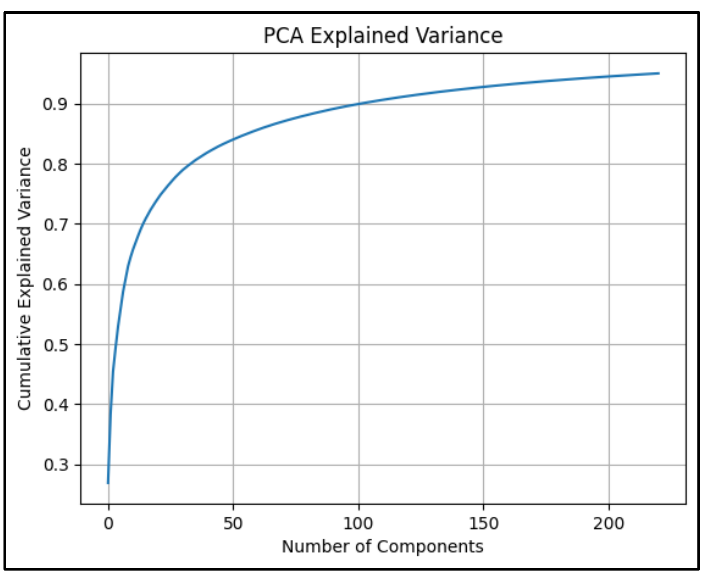
2. Flattening

```
[16] from sklearn.preprocessing import StandardScaler

[17] scaler = StandardScaler()
     x_train_scaled = scaler.fit_transform(x_train_flat)
     x_test_scaled = scaler.transform(x_test_flat)
```

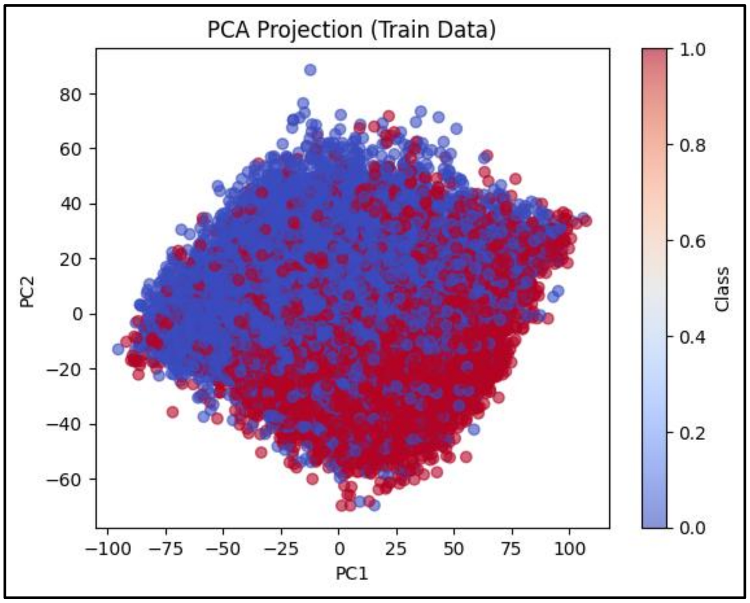
3. Standardization*:*

Applied StandardScaler (zero mean, unit variance)



4. PCA:

Applied PCA after standardization with different component sizes (e.g., 50, 100, 200)



Experiments & Models

1. Logistic Regression With Standardization:

- Trained with standardized features (no PCA)
- Hyperparameter tuning:

  - **C**=0.01,1,0.1
  - **Solver** = lbfgs, liblinear, saga
  - **Penalty** = l1, l2
  - **max_iter** =1000, 10000

- Result:

  - **Training Accuracy**: ~83%,
  - **Testing Accuracy**: ~80–81%
  - Slight overfitting observed

Few Example:

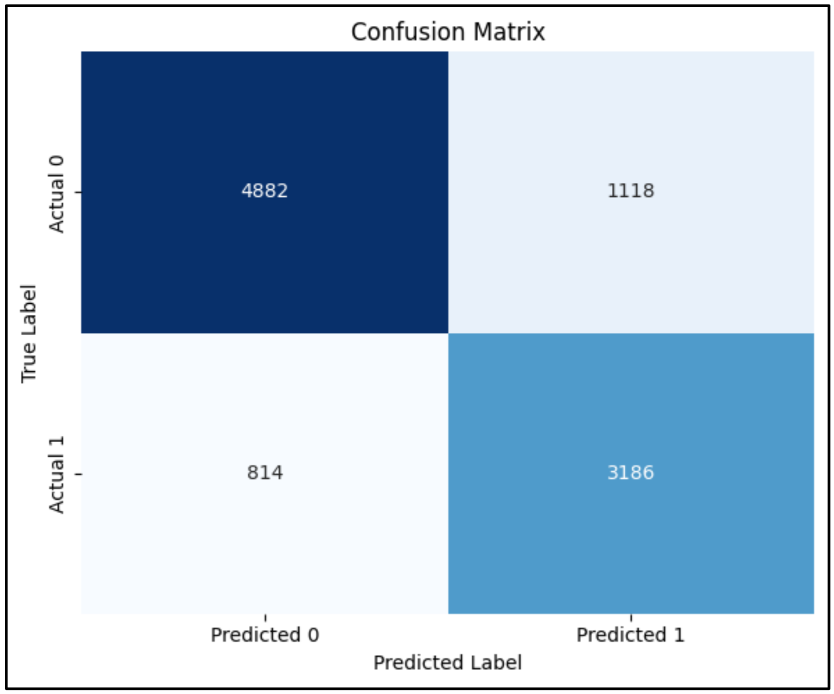
- Below are screenshots of few trained model on above consideration

❖ **Model 1 :**

```
model4=LogisticRegression(penalty='l1', C=1, solver='liblinear', max_iter=1000,class_weight='balanced')
model4.fit(x_train_scaled,y_train_flat)
```

```
LogisticRegression
LogisticRegression(C=1, class_weight='balanced', max_iter=1000, penalty='l1',
                   solver='liblinear')
```

```
Test Accuracy: 0.8068
Training Accuracy: 0.8313
```

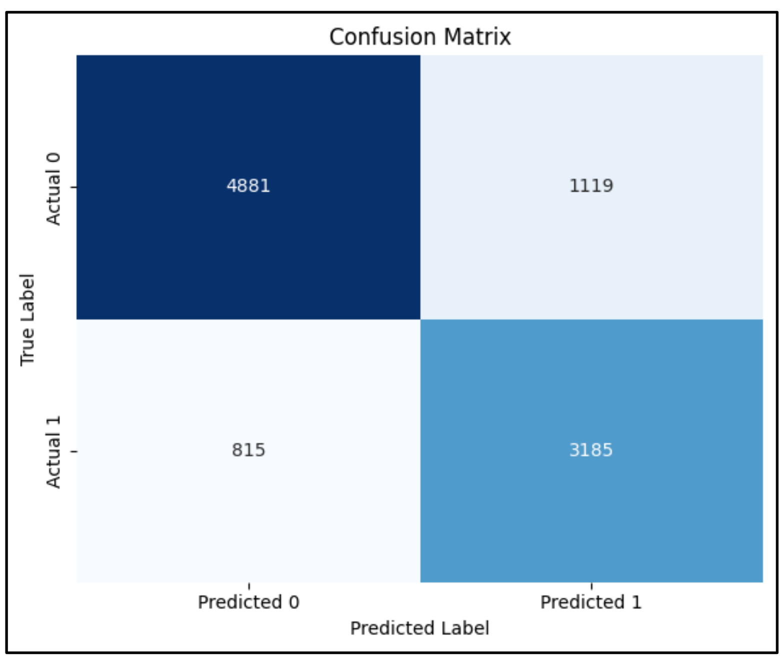


❖ **Model 2:**

```
model5=LogisticRegression(penalty='l2', C=1, max_iter=10000,class_weight='balanced')
model5.fit(x_train_scaled,y_train_flat)
```

```
LogisticRegression
LogisticRegression(C=1, class_weight='balanced', max_iter=10000)
```

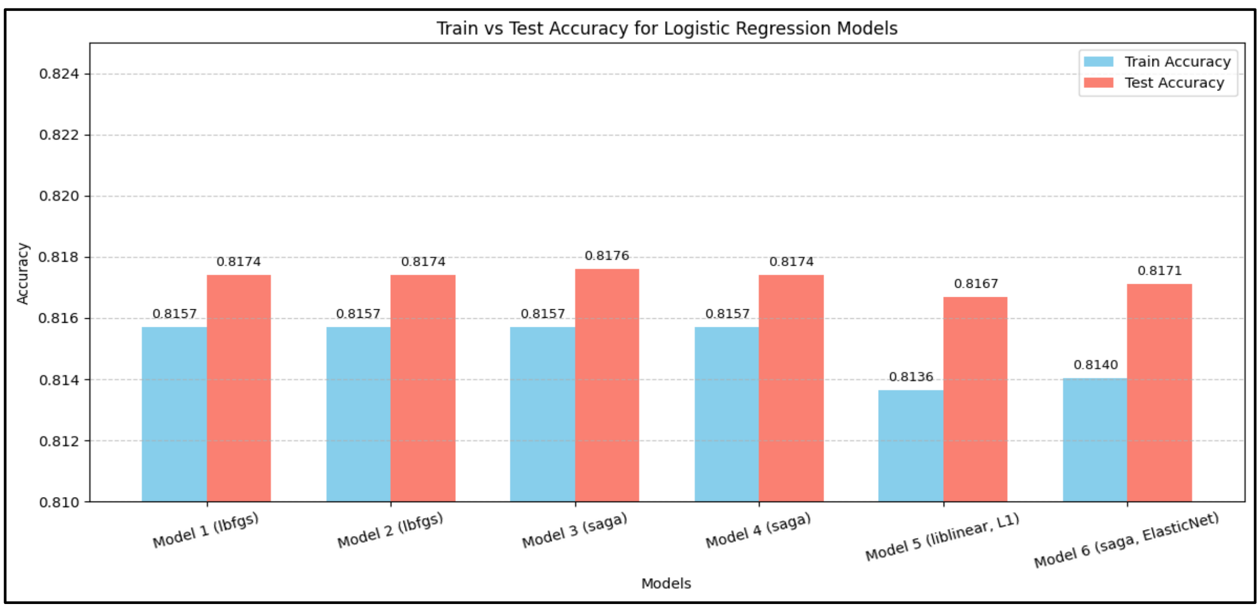
```
[ 815 3185]]
Test Accuracy: 0.8066
Training Accuracy: 0.8334
```



2 . Logistic Regression (PCA after Standardization)

In this phase, **Principal Component Analysis (PCA)** was applied **after standardizing** the input features. The aim was to reduce dimensionality and eliminate redundant information while retaining most of the variance in the data.

- **PCA Configuration:**
  - n_components = 0.95 (to retain 95% variance)
  - PCA applied on standardized data
  - Dimensionality reduction helped speed up training and reduce overfitting
- **Model Training:**
  - Logistic Regression models were trained on PCA-reduced features
  - Various hyperparameters were explored:
    - C: 0.01, 1, 0.1, 0.005, 10, 0.001
    - Solver: 'lbfgs', 'liblinear', 'saga'
    - Penalty: 'l1', 'l2', 'elasticnet'
    - max_iter: 1000, 10000
- **Validation Strategy:**
  - Used **StratifiedKFold** for certain experiments to ensure balanced class splits during cross-validation
  - For others, a simple train/test split was used to quickly evaluate combinations
- **Results:**
  - **Training Accuracy:** ~80–81%
  - **Testing Accuracy:** ~80–81%
  - **Observation:** Near-identical train/test accuracy indicates **minimal overfitting** and **better generalization** than the baseline model without PCA.



MODEL_NO	C	penalty	solver	Max_iter	Train Accuracy	Test Accuracy
1	0.01	l2	lbfgs	10000	0.8157	0.8174
2	0.005	l2	lbfgs	10000	0.8157	0.8174
3	0.005	l2	saga	10000	0.8157	0.8176
4	0.01	l2	saga	10000	0.8157	0.8174
5	0.05	L1	liblinear	1000	0.8136	0.8167
6	0.01	elasticnet	saga	1000	0.8140	0.8171

```
l1_ratio': [0.1, 0.5, 0.7, 0.9, 1.0]
```

Note: This is also a Hyperparameter in elasticnet solver

Convolutional Neural Network - CNN

A CNN is a deep learning architecture designed for grid-like data (e.g., images), using convolutional layers to automatically extract hierarchical spatial features.

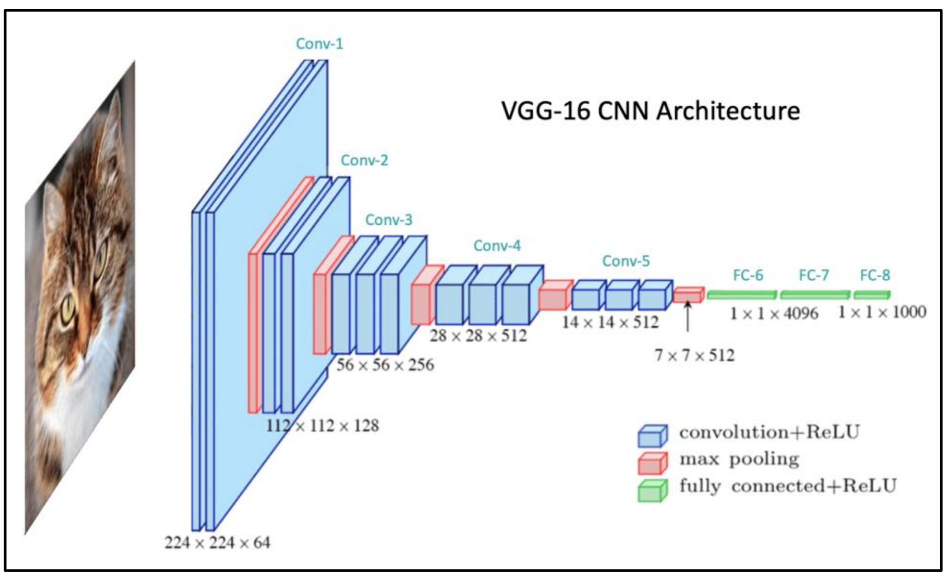
Key Components:

Convolutional Layers:

- Apply filters (kernels) to detect local patterns (edges, textures).
- Preserve spatial relationships via sliding windows.
- Pooling Layers (e.g., MaxPooling):
- Reduce dimensionality while retaining important features.

Fully Connected Layers:

- Classify extracted features into output classes.



Data Preprocessing Steps

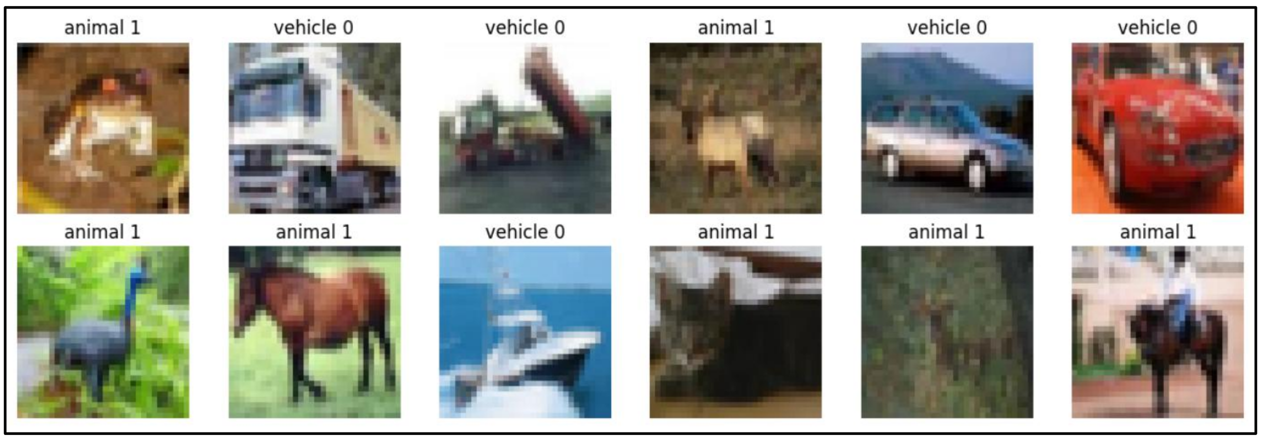
To build a robust image classification model using CNN, the following preprocessing steps were applied:

1. Dataset Acquisition:

- The CIFAR-10 dataset was manually downloaded from `cifar10.load_data()`
- The dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.

2. Class Relabeling:

- For binary classification, the 10 classes were mapped into 2 categories:
    - **Vehicles**: airplane, automobile, ship, truck
    - **Animals**: bird, cat, deer, dog, frog, horse
- Corresponding labels were relabeled as:
    - 1 → Animal
    - 0 → Vehicle



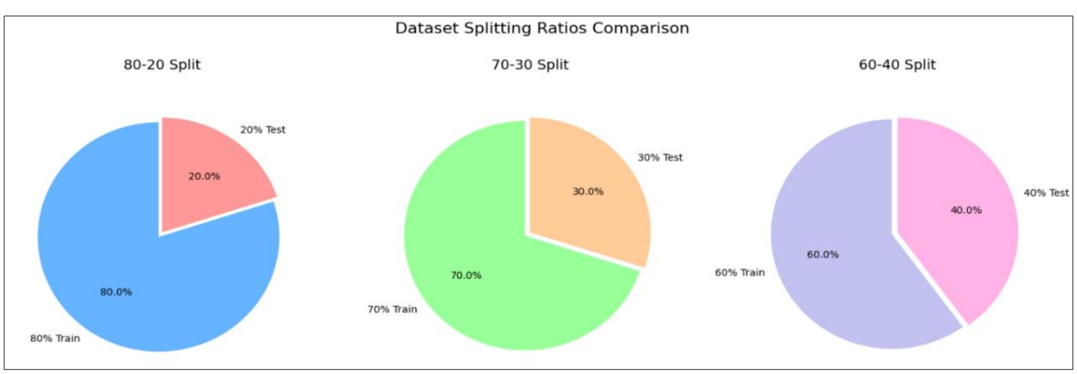
3. Data Normalization:

- Pixel values were normalized to the [0, 1] range by dividing by 255.

4. Train/Validation/Test Split:

Applied three different split ratios:

- 60-40 Split (train_test_split(test_size=0.4)
- 70-30 Split (train_test_split(test_size=0.3))
- 80-20 Split (train_test_split(test_size=0.2))



Model Architectures & Algorithm

Model 1: Basic CNN

Architecture:

- Single convolutional layer (32 filters)
- Max pooling for dimensionality reduction
- Fully connected layer (64 neurons)
- Sigmoid output for binary classification

Code Implementation :

```
# Model 1: Baseline CNN
def build_model_1(lr=0.001):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(64, activation='relu',kernel_regularizer=l2(0.001)),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Model 2: Regularized CNN with Dropout

Architecture

- Added dropout layers (25% after conv, 50% before output)
- Deeper architecture with 2 convolutional layers
- Increased learning rate (0.0005) for stable training

Code Implementation :

```
def build_model_2(lr=0.0005):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D(2, 2),
        Dropout(0.25),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Dropout(0.25),
        Flatten(),
        Dense(32, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Model 3: Advanced CNN with Batch Normalization

Architecture:

- Batch normalization after each conv layer

- Wider architecture (64 filters in second conv layer)
- Lowest learning rate (0.0001) for fine-tuning
- Dropout only in final layers

Code Implementation

```
def build_model_3(lr=0.0001):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        BatchNormalization(),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Architectural Comparison

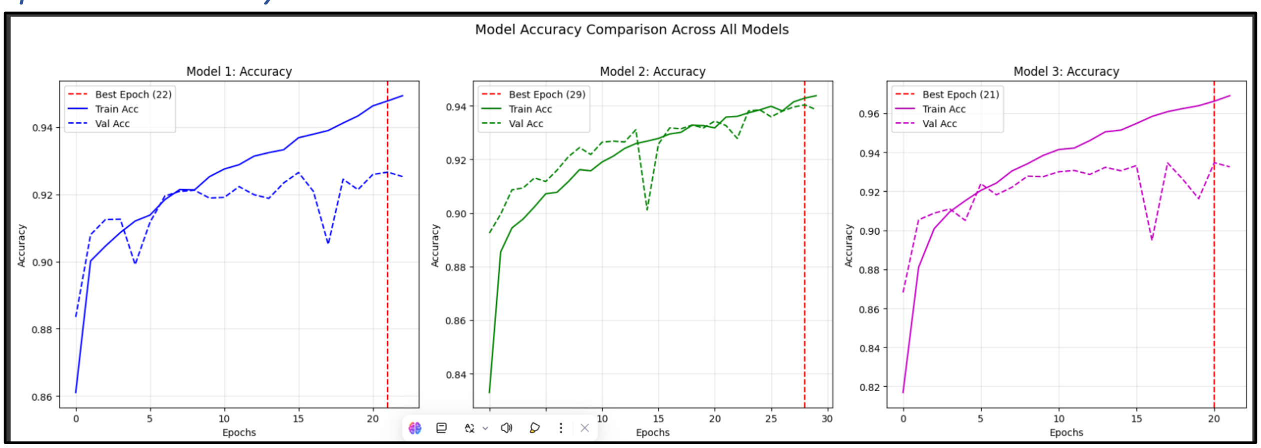
List of Hyperparameters

Hyperparameter	Model 1	Model 2	Model 3
Learning Rate	0.001	0.0005	0.0001
Conv Layers	1 layer	2 layers	2 layers
Filters in Conv Layers	32	32,64	32,64
Dropout Rate	None	0.25,0.25,0.5	0.5
Batch Normalization	No	No	Yes
Dense Units (Fully Connected Layer)	64	32	64
Activation Function	ReLU + Sigmoid	ReLU + Sigmoid	ReLU + Sigmoid
Optimizer	Adam	Adam	Adam
Max Poooling	1 layer	2 layers	2 layers
Input Shape	(32,32,3)	(32,32,3)	(32,32,3)
Train – Valid Split	80:20	70:30	60:40

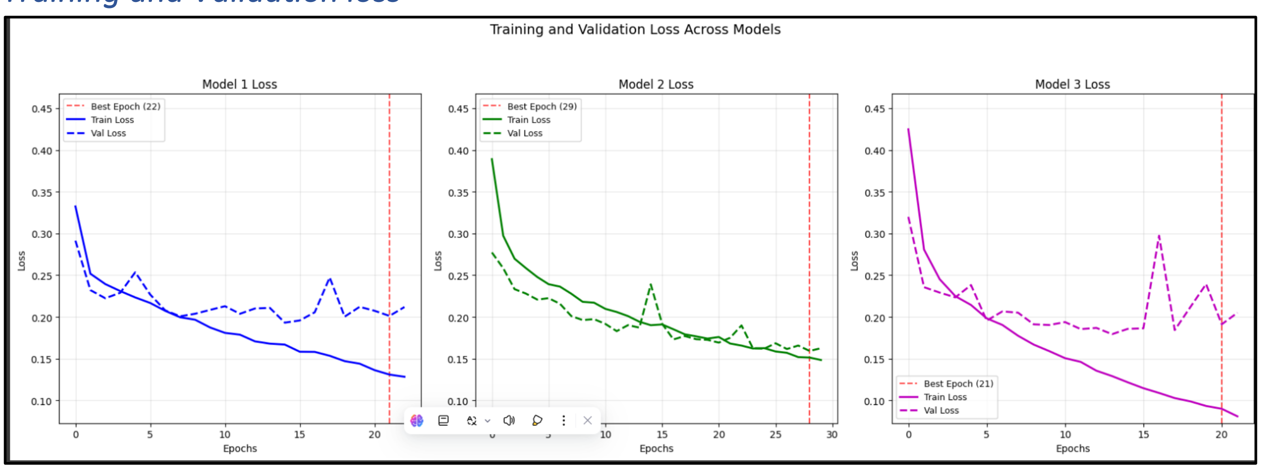
Results Comparison

Model	F1_score	Precision	Recall	Accuracy
Model 1	0.936	0.92	0.937	0.924
Model 2	0.948	0.94	0.948	0.938
Model 3	0.941	0.93	0.944	0.932

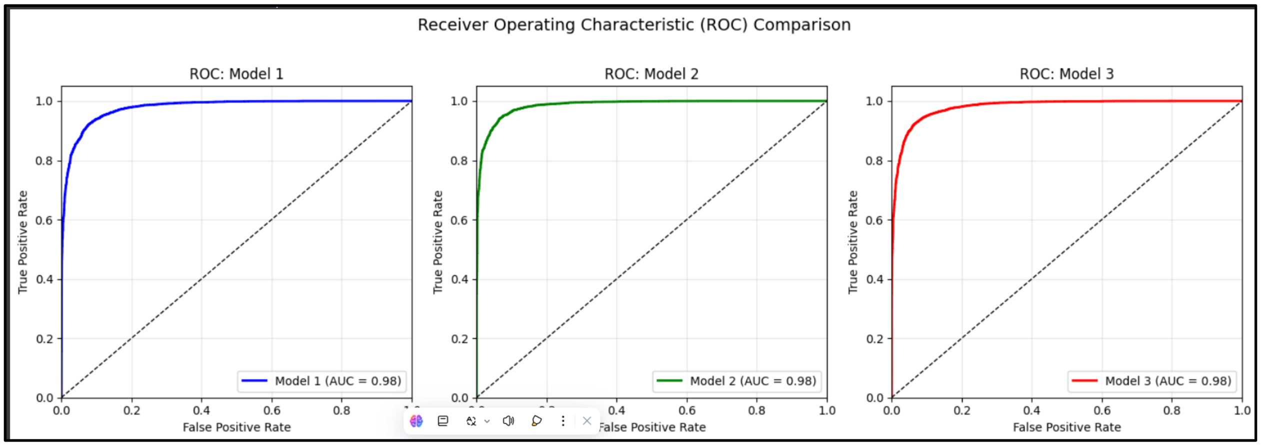
Epochs vs Accuracy



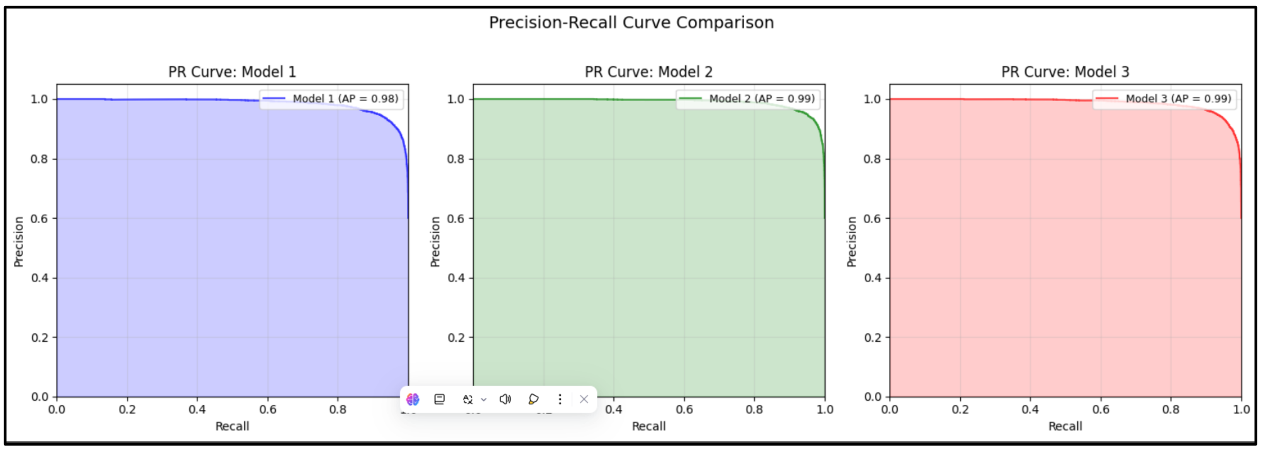
Training and Validation loss



ROC Comparison

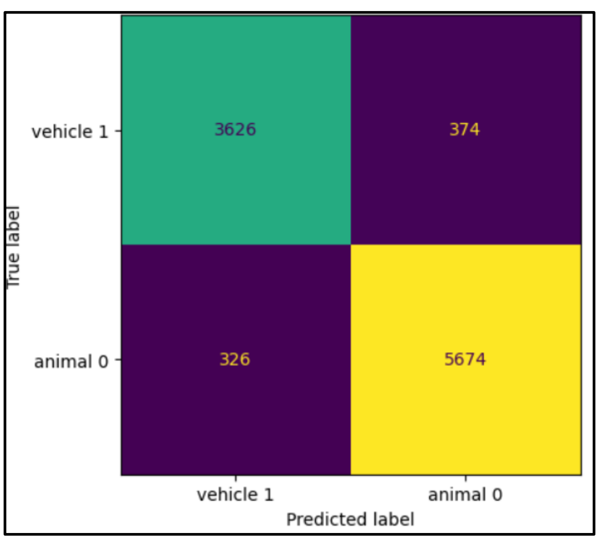


Precision Recall Curve Comparison

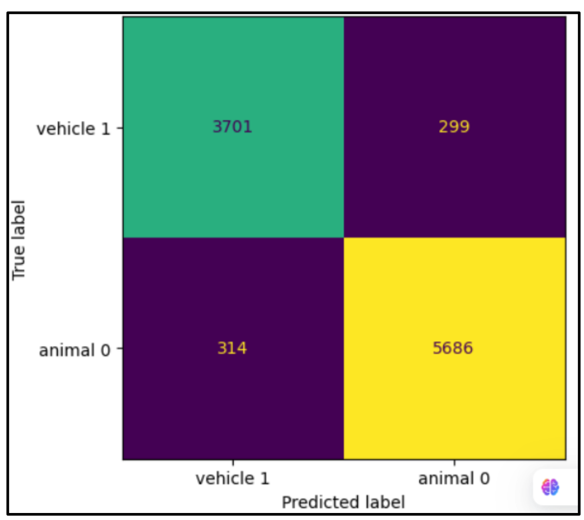


Confusion Matrices

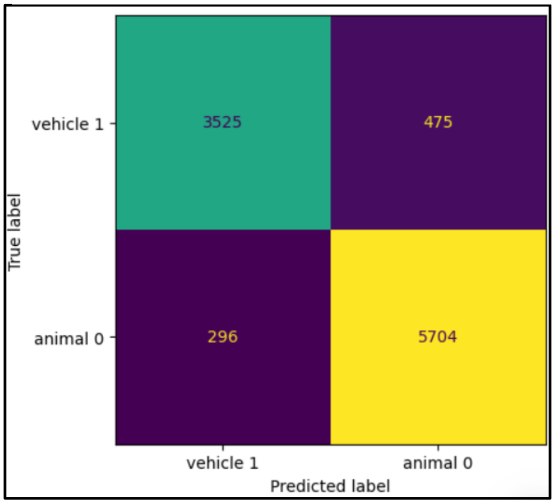
Model 1



Model 2



Model 3



Key Observations & Model Performance Analysis

Overfitting in Initial Architecture (Model 1)

- **Model Complexity**: 461,825 parameters.
- **Symptoms of Overfitting**:
  - High training accuracy (~98%).
  - Lower validation accuracy (~91%).
  - Increasing gap between training and validation loss curves.

Improvements in Model 2

- **Adjustments Made**:
  - Reduced model complexity.
  - Introduced Dropout:
    - 25% after convolutional layers.
    - 50% before dense layer.
- **Impact**:
  - Decreased train-validation accuracy gap from ~10% to ~5%.
  - More stable generalization.

Effect of Higher Learning Rate

- **Positive**:
  - Rapid convergence: ~91% accuracy within the first 5 epochs (Model 1).
- **Negative**:
  - Aggressive weight updates led to premature overfitting.
  - Train-validation accuracy gap stabilized at ~6% (Train: 96%, Val: 91%).

Impact of Data Split Ratios

- **60:40 Split**:
  - Lower validation accuracy compared to 80:20.

- **Observation**:
  - Smaller validation sets (e.g., in 80:20) may temporarily appear more stable but can give optimistic results.
  - Larger validation sets give more reliable estimates but may expose overfitting more clearly

Solutions Implemented to mitigate issues

1. Early Stopping

- **Configuration**:
  - patience = 6, monitored on val_loss.
- **Effect**:
  - Prevented unnecessary training beyond overfitting point

2. L2 Regularization

- **Setup:**
  - Applied with λ = 0.001.
- **Results**:
  - Slight drop in both training and validation accuracy.
  - Reduced train-validation accuracy gap.
  - Improved generalization.
  - However, peak accuracy was ~2% lower compared to the unregularized model—indicating a trade-off between generalization and capacity

Results and Analysis

Models	Accuracy
KNN: **Model 1**	
Model 2	
Model 3	
Logistic Regression: **Model 1**	0.8174
Model 2	0.8174
Model 3	0.8176
Model 4	0.8174
Model 5	0.8171
Model 6	0.81746
CNN: **Model 1**	0.924
Model 2	0.938
Model 3	0.932

Conclusion & Future Work

Summary of Findings

Our comparative analysis of KNN, Logistic Regression, and CNN on the CIFAR-10 dataset revealed:

1. **CNN outperformed** traditional methods, achieving ~92-94% test accuracy (vs. ~81-82% for KNN/Logistic Regression), demonstrating its superiority in learning hierarchical spatial features.

2. Trade-offs:

- KNN provided fast training but suffered from high dimensionality
- Logistic Regression was interpretable but limited by linear decision boundaries (accuracy: ~40%).
- CNN required more resources but delivered state-of-the-art results.

3. Use Cases for Simpler Models:

- KNN/Logistic Regression may suffice for low-resolution binary tasks (e.g., vehicle vs. animal) or resource-constrained environments.

Future Directions

Transfer Learning:

- Fine-tuning pre-trained models (e.g., ResNet50, EfficientNet) on CIFAR-10 could boost accuracy further.

Hybrid Approaches:

- Combine CNN feature extraction with traditional classifiers (e.g., SVM or Logistic Regression on CNN embeddings).

Explainability:

- Use Grad-CAM or SHAP to interpret CNN decisions for critical applications.