Implementing and Evaluating Classification Methods for CIFAR-10

1st Jordan Yee
University of California, Irvine
Irvine, USA
91427122

2nd Shifeng Hong
University of California, Irvine
Irvine, USA
53124338

3rd Travis Tran
University of California, Irvine
Irvine, USA
67987698

***Abstract*—This project involves implementing and evaluating various classification methods on the CIFAR-10 dataset. The dataset consists of 60,000 32x32 color images across 10 different classes. Our goal was to compare the performance of different classifiers and gain insights into their strengths and weaknesses.**

***Index Terms*—CIFAR-10, K-Nearest Neighbors, Logistic Regression, Neural Networks, Random Forest**

I. Summary

In this project, we implemented and analyzed various machine learning algorithms on the CIFAR-10 dataset, consisting of 60,000 32x32 color images in 10 classes. We investigated four classifiers: k-nearest Neighbors (kNN), logistic regression, neural networks, and random forest. After evaluating the results, we found that neural networks, more specifically convolution neural networks (CNNs), provided the highest accuracy. Random forests performed almost as well; however, they fell short of CNN's accuracy score, while the Multi Layer Perceptron (MLP) in the neural network trailed behind. All of the classifiers, including kNN and logistic regression, demonstrated robustness to hyper-parameter settings, helping our team learn and understand the strengths and weaknesses of each method.

II. Data Description

The dataset we decided to use in our project was CIFAR-10, a widely known dataset in the field of computer vision. It consists of 60,000 colored images that are 32x32, with the images being split evenly amongst the ten classes. These classes are plane, car, bird, cat, deer, dog, frog, horse, ship, and truck. To download the CIFAR-10 dataset, we utilized the library "torchvision" from PyTorch to load the data. We then applied transformations to the dataset that converted the images into tensors and normalized them. For dataset exploration, we randomly selected four training data points and displayed their images and labels. Additionally, we applied normalization and conducted a statistical summary of the dataset, including mean, standard deviation, minimum, and maximum pixel values.

```
Mean normalized pixel values: [-0.02206252 -0.0406169  -0.0463728  -0.05144313 -0.05472538 -0.05625335
 -0.05688907 -0.05714419 -0.05710053 -0.05685697 -0.05646389 -0.05660968
 -0.05717352 -0.05749096 -0.05814276 -0.05885128 -0.05890689 -0.0584798
 -0.05783283 -0.05715771 -0.05717792 -0.05719506 -0.05693044 -0.05661761
 -0.05669537 -0.0562898  -0.05605028 -0.05513093 -0.05241487 -0.04817131
 -0.0422875  -0.03172589]
Standard deviation of normalized pixel values: [0.5277244  0.5168944  0.51166594 0.50817966 0.50474596 0.5030011
 0.50025326 0.49946505 0.49738276 0.49654627 0.49538493 0.49517485
 0.49409014 0.49356443 0.49302414 0.49327558 0.49322405 0.49340874
 0.49391437 0.49465865 0.49617618 0.4961314  0.49740633 0.4979996
 0.49993607 0.50099    0.50395334 0.5055563  0.5085286  0.51147085
 0.51505154 0.5227425 ]
Minimum normalized pixel values: [-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]
Maximum normalized pixel values: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1.]
```

Fig. 1. Statistics for dataset

This process aimed to help us better understand the data distribution for easier processing. Subsequently, the research conducted in the "CIFAR-10 Image Classification Using Feature Ensembles" study by Felipe O. Giuste and Juan C. Vizcarra leveraged the CIFAR-10 dataset for image classification tasks. Their research aimed to improve accuracy by combining various feature extraction techniques, including Histogram of Oriented Gradients (HOG). Ultimately, they achieved an impressive accuracy of 94.6%. This outcome shows the potential of what can be done with this dataset.

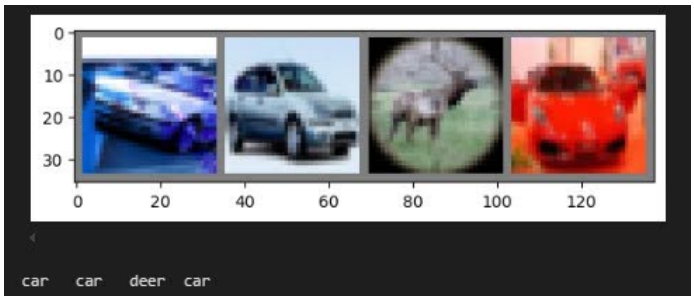

Fig. 2. Sample images drawn from dataset

III. Classifiers

A. kNN

kNN is an instance-based learning algorithm that essentially speaks for itself; it classifies data points based on the k nearest neighbors in the feature space of the majority class. We experimented with a different number of neighbors (k), using 5-fold cross-validation to determine the optimal k value. To

achieve this result, we used the scikit-learn package to evaluate performance on both training and test sets.

Hyperparameters investigated:

- k: The number of nearest neighbors.
  - Experimented with k = [1, 5, 10, 50]

B. Logistic

Logistic regression is a linear model primarily used for binary and multiclass classification. The model applies a logistic function to a linear combination of input features to analyze the probability of a given class. In our implementation, we scaled the features using normalization. Additionally, we experimented with different regularization strength (C) values, using 5-fold cross-validation to determine the optimal C value. To achieve this result, we used the scikit-learn package to evaluate performance on both training and test sets.

Hyperparameters investigated:

- Regularization Strength (C): Smaller values specify stronger regulation
  - Experimented with C = [0.01, 0.1, 1, 10, 100]
- Max iterations: Maximum number of iterations for the solver.
  - Experimented with max iterations = [500, 1000, 2000, 5000]

C. Random Forest

The classification made by random forest takes the majority vote of a group of decision trees; each tree trained on a random subset of the given training dataset focuses on random features (aspects of a data point). Inference via decision trees is achieved by choosing junctions depending on whether a threshold is satisfied. To achieve this result, we used the scikit-learn package to evaluate performance on both training and test sets.

Hyperparameter investigated:

- n_estimator: Number of decision trees trained
  - Experimented with n_estimator = [5, 25, 50, 100]
- min_sample _split: Minimum amount of data points allowed before splitting further
  - Experimented with min_sample_split = [2, 5, 10]
- min_sample _leaf: The smallest amount of data points allowed at a leaf node
  - Experimented with min_sample_leaf = [2, 5, 10]

D. Neural Network

For the project, we employed both a Convolutional Neural Networks (CNN) and a Multi layer Perceptrons (MLP). CNNs are deep learning models that consist of convolutional layers that apply filters to extract key features from an image. Additionally, they include pooling layers for cutting dimensionality along with other layers to ultimately classify images. MLPs are a type of feedforward neural network that consists of layers of nodes that are all fully connected to each other. To implement these neural networks, we used scikit-learn for the MLP and PyTorch for the CNN.

Hyperparameters investigated:

- Epochs (for CNN): Number of complete cycles on the entire training dataset
  - Experimented with epochs = [1, 5, 10, 20, 30]
- Batch Size (for MLP): Number of training examples used in one iteration of training
  - Experimented with batch sizes = [32, 64, 128, 256]
- Solver (for MLP): Algorithm used for optimizing weights
  - Experimented with solver = [‘sgd’, ‘adam’]
- Max iteration (for MLP): Maximum number of iterations training will run
  - Experimented with max iteration = [100, 500, 1000]

IV. Experimental Setup

For our project, we decided to use a subset of the data to speed up our models' runtime. As such, we used 10,000 images for the training dataset and 2,000 images for the testing dataset. To evaluate our model's performance, we looked at classification accuracy/error, analyzed the confusion matrix, and measured prediction speed. This would provide a solid benchmark to determine which model achieves high accuracy while maintaining speed.

To experiment with the hyper-parameters of K-Nearest Neighbors (kNN), Logistic Classification, Random Forest, Convolutional Neural Network (CNN), and Multi-Layer Perceptron (MLP), we made sure to implement a seed for reproducible results. For kNN, we tuned the hyper-parameter, k, to identify the best number of neighbors to predict. We did this by using 5-fold cross-validation, where we varied the value of k and returned the k value with the highest accuracy on the test data. We applied a similar approach to our Logistic classifier, using 5-fold cross-validation to get the best regularization strength, C, which helps reduce over-fitting. For Random Forest, we employed a brute-force technique, testing various values for the hyper-parameters, ”n_estimators,” ”min_samples_split,” and ”min_samples_leaf.” Optimizing these would give us the best values for the number of trees, the minimum number of data points to split on, and the minimum number of data points in a leaf node. In the case of CNN, we experimented with different epoch values to determine the best number of complete cycles through the training dataset. Lastly, for MLP, we varied batch sizes to find the most optimal number of training examples used in an iteration. We also experimented with different solvers and maximum iterations to help the model better converge.

For the kNN classifier, we found an optimal value of k=10, while for the logistic classifier, we set C=0.01 with a maximum iteration of 5000. In the case of Random Forest, we configured the number of estimators to 100, with a minimum sample split of 5 and a minimum sample leaf of 2. For CNN, we found the number of epochs of 20 to be the most optimal. Finally, for the MLP, we opted for a batch size of 128, the solver ‘adam’ and a maximum iteration of 1000.

V. Experimental Results

These results were obtained by running the fine-tuned models with a test set of 2,000 images.

TABLE I
ACCURACY

Classifier	Accuracy
kNN	28.15%
Logistic	35.10%
Random Forest	42.70%
MLP	40.55%
CNN	54.75%

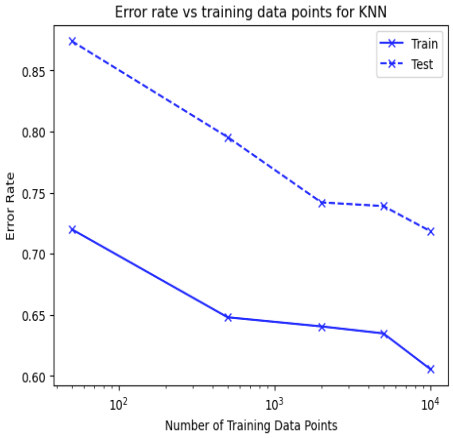

Fig. 3. Learning curve for kNN.

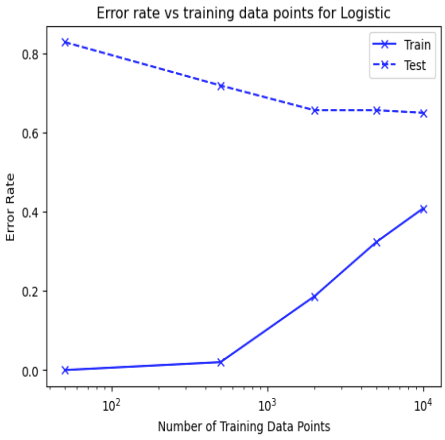

Fig. 4. Learning curve for Logistic.

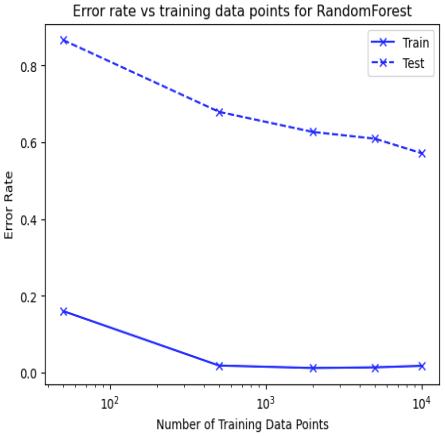

Fig. 5. Learning curve for RandomForest.

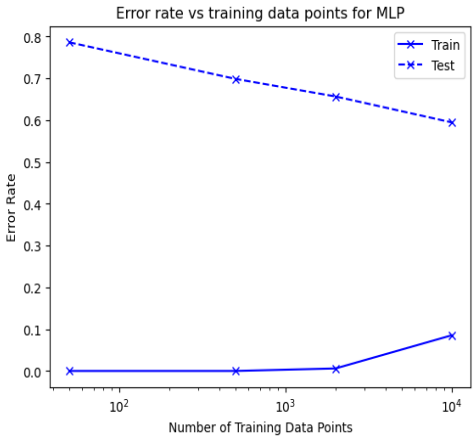

Fig. 6. Learning curve for MLP.

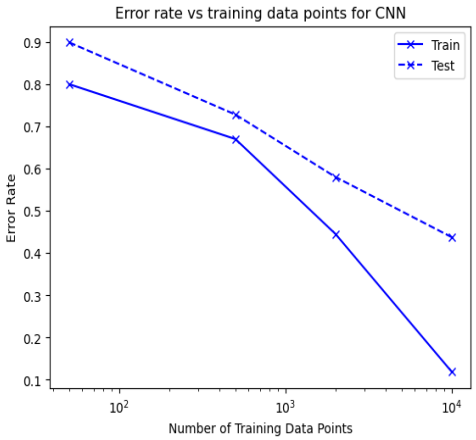

Fig. 7. Learning curve for CNN.

The testing Error rate decreases for all models as the training dataset grows. Among all, CNN showed the steepest decrease in testing error rate. CNN is the only model starting with a high error rate on training data, yet it decreases drastically as

the data size grows. The logistic classifier is the only model whose error rate increases as sample size increases, likely an effect of regularization (which other models do not implement)

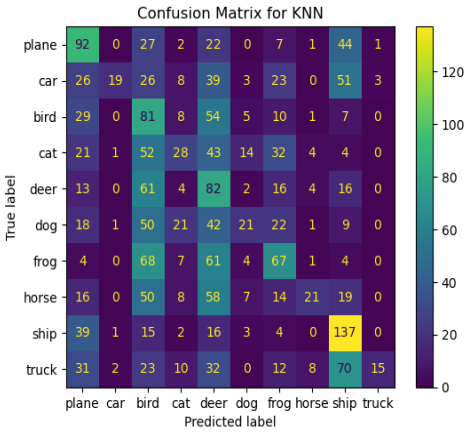

Fig. 8. Confusion matrix for KNN.

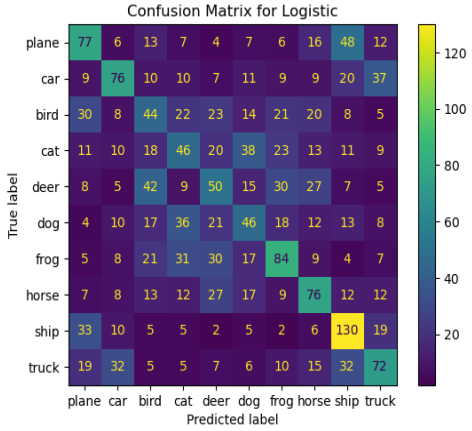

Fig. 9. Confusion matrix for Logistic.

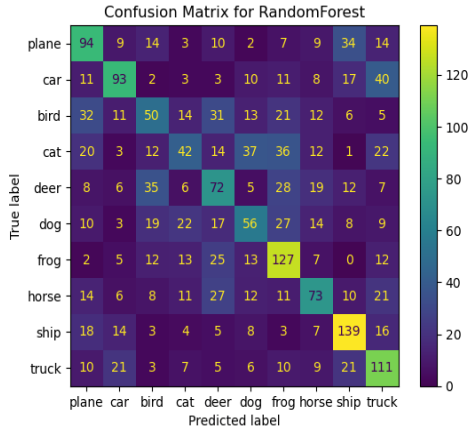

Fig. 10. Confusion matrix for Random Forest.

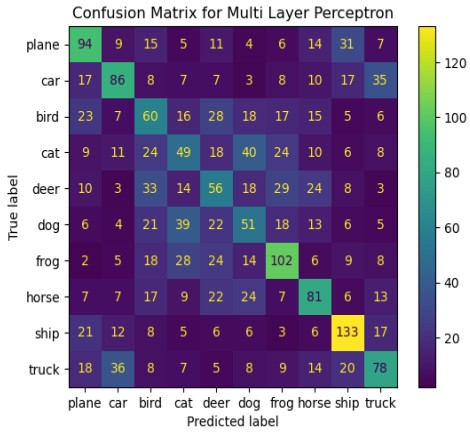

Fig. 11. Example of a figure caption.

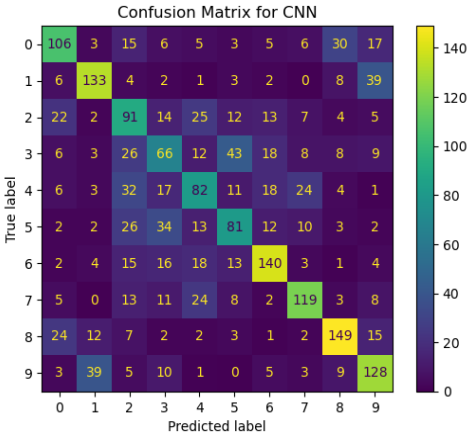

Fig. 12. Example of a figure caption.

KNN performs poorly, especially when misclassifying other animals as birds or deer, which doesn't happen for different classifiers. Logistic, RandomForest, and MLP have similar recognition patterns; they all seem to be confused among animals (the lighter blue square area in the middle covers animal classes: bird, cat, deer…horse). Most models seem to correctly classify non–animal classes at a higher rate.

```
# report Prediction speed
reportPredictionTime(knn, name = "KNN", X_test=X_test)
reportPredictionTime(lr, name = "Logistic", X_test=X_test)
reportPredictionTime(rf, name = "RandomForest", X_test=X_test)
reportPredictionTime(mlp, name = "MLPClassifier", X_test=X_test)
reportPredictionTimeCNN(cnn, te_loader, te_subset)
✓ 9.5s
Average prediction time per image by KNN: 0.000841 seconds
Average prediction time per image by Logistic: 0.000015 seconds
Average prediction time per image by RandomForest: 0.000083 seconds
Average prediction time per image by MLPClassifier: 0.000008 seconds
Average prediction time per image by CNN: 0.003815 seconds
```

Fig. 13. Predictions speed of classifiers

VI. Insight

1) Classifiers have a hard time telling animals apart !
    - Most classifiers have a hard time classifying between animal classes while having a much lower mis-identification rate between non-animal(Vehicle) and telling non-animal apart from animal. Hypothesis being that vehicle's picture tends to have a distinctive background and color while animal pictures are taken with a natural background(green for lawn or forest, gray, blue for sky), and their color is limited (you are unlikely to see a blue cat). However, vehicles, being artificial and their picture taken in many more settings, can have strong contrast colors (paint) and vastly different backgrounds (outdoor vs indoor).
2) Strengths and weaknesses of different models
    - CNN took more resources to train and infer, yet it provides the best accuracy. Random Forest and MLP offer a good balance as they provide good accuracy while taking up fewer resources to train and less time to predict. Logistic outperformed kNN in accuracy and prediction time; kNN is likely not an optimal model for this dataset. This could be because the features are too similar, causing more mis-classifications. In a real-world setting, it may be best to use MLP since it provides satisfactory accuracy and is the quickest predictor.
3) Discussion
    - Through confusion matrices, learning curves, and measurement of prediction speed, the models' strengths and weaknesses were exposed. This data helped us better understand and learn the intricacies of each model and raise awareness for choosing the adequate model and making trades for real-world scenarios like autonomous vehicles, security and surveillance, and medical imaging analysis, three critical and growing fields in artificial intelligence.

VII. Contributions

- **Jordan**: Implemented the k-nearest Neighbors and logistic regression classifiers. He was responsible for tuning the hyper-parameters, evaluating the performance of these models, and visualizing the results.
- **Travis**: Designed the neural network structure for the Multi-Layer Perceptron and Convolutional Neural Network. He also developed the architectures for tuning the hyper-parameters and evaluated performance.
- **Shifeng**: Translated the report into Jupyter Notebook to generate graphs and various plots used throughout the sections. He also implemented the random forest classifier and visualized the results in different hyper-parameter settings.