Comparison of Supervised Learning Algorithms on CIFAR-10 Dataset

Zibo Gong, Jiayi Yu, Hanqi Tang, Hao Liang

Group Member Contributions:

- **Zibo Gong:** Code Implementation, Experiment (Report Section)
- **Jiayi Yu(jiayiyu3):** Slide Preparation, Difficulties and Challenges (Report Section)
- **Hanqi Tang:** Methodology (Report Section)
- **Hao Liang:** GitHub repo, Problem Statement (Report Section)

1 Problem & Goal

1.1 Motivation

The demand for efficient and reliable computer vision systems is growing due to the widespread application of devices such as smartphones and drones. We did this project to mimic real-world constraints and try to find some possible solutions by classifying 60,000 low-resolution 32×32 RGB images from the CIFAR-10 dataset into one of ten pre-labeled classes. We aim to benchmark four model families under the fixed computational constraints: one single Google Colab T4 GPU). Thus, this is a task complicated by both technical limitation and visual ambiguities. Model families include logistic regression, random forest, KNN and MLP. We also explored some extensions of the traditional neural network model which includes BNN, simple CNN, Vgg-16 and Resnet-18.

1.2 Dataset Overview

The CIFAR-10 dataset (Canadian Institute for Advanced Research) is a widely used collection of images for training machine learning and computer vision algorithms ("AI Progress Measurement". Electronic Frontier Foundation. 2017-06-12. Retrieved 2017-12-11). It contains 60,000 32x32 color images equally distributed across 10 different classes, which are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

1.3 Evaluation Criteria

The primary metric we used is overall accuracy greater than 0.65 on the CIFAR-10 unseen test set. This threshold is concluded from our empirical evaluation after training several modeling families. For secondary metrics, we used log-loss to access probabilistic calibration performance and per-class precision/recall/F1 scores to access performance within individual class. We stratified 5-fold cross-validation in the 50k training set for hyper-parameter tuning, ensuring balanced representation in each fold. Due to the computational limits, all experiments run on a single Colab T4 GPU (8GB VRAM). Since we need to balance training time and model complexity, deep and Bayesian networks are limited to a maximum of 5 epochs to remain feasible within the constraints of free resources. This framework prioritizes models that deliver good accuracy while operating under edge-device resource boundaries.

2 Methodology

In this project, we set out to compare a range of supervised learning algorithms on the CIFAR-10 image classification task, which consists of small 32x32 color images spread over ten different categories. Given the relatively low resolution of the dataset and the limited compute power we had (a single free Google Colab T4 GPU), our goal was not just to see which model performed best in terms of accuracy, but also to understand the trade-offs in training speed, resource usage, and overall feasibility.

We started with several classical machine learning models: logistic regression, k-nearest neighbors (KNN), random forest, and a multilayer perceptron (MLP). Right away, we noticed a common challenge: traditional models don't handle raw image data very well. The pixel vectors in CIFAR-10 are 3072-dimensional (32x32x3), which made training slow and unstable, especially for logistic regression and MLP. To address this, we applied Principal Component Analysis (PCA) to reduce the input dimensionality to 200 components while still preserving about 0.95 of the variance. This significantly sped up training and helped with model convergence.

Logistic regression was extremely fast to train and evaluate, but it only reached 0.41 accuracy. This was expected—it's a linear model and doesn't capture complex patterns like edges or shapes. KNN performed slightly worse, at 0.36, and was also very slow at inference time, since it needs to compare each test image to every training image. We tuned the number of neighbors (k) and found that k=5 worked best, but overall, KNN wasn't practical for this task. Random forest also plateaued at around 0.41 accuracy. It was more robust than KNN and less sensitive to noise, but training was slow, especially as the number of trees increased. We tested different values and found that using 50 trees gave us a good balance between speed and performance.

The MLP did better than the other classical models, reaching 0.56 accuracy. Because it uses nonlinear activation functions, it can model more complex relationships in the data. But since it's a fully connected network, it still treats all pixels as independent features and doesn't take spatial structure into account, which limits its ability to work with image data.

To go further, we tried more advanced deep learning models. We implemented a Bayesian Neural Network (BNN), which is like a regular neural network but models uncertainty by learning distributions over weights rather than single values. The idea is to be able to estimate confidence in predictions, which is useful in applications where reliability matters. But BNNs are expensive to train. They use variational inference and Monte Carlo sampling, which means slower training and more memory usage. We limited training to just five epochs and reduced the number of forward passes during inference. Even so, BNN took the longest to train but managed 0.63 accuracy. In practice, it didn't outperform standard deep models, especially under our resource constraints.

Next, we implemented a simple CNN, which immediately improved performance to 0.7. CNNs are a natural fit for image tasks because they use filters that slide over the image and learn local features like edges and textures. This local structure is something traditional models completely miss. Building on that, we tested two well-known deep architectures: VGG-16 and ResNet-18. Both are significantly deeper and more powerful. VGG-16 is built from a stack of small 3x3 convolutions and pooling layers, while ResNet-18 introduces residual connections to help with gradient flow and allow training of deeper networks without vanishing gradients. We adapted both models for CIFAR-10 by modifying their final layers to output 10 classes. VGG-16 performed the best overall with 0.78 accuracy, while ResNet-18 followed closely with 0.76.

Training deep networks in Colab wasn't easy. GPU memory was a bottleneck, especially with larger models. To manage this, we first prototyped everything on the simpler CNN, which allowed us to debug and test ideas quickly. For final evaluation, we loaded the larger models and used batch sizes small enough to avoid out-of-memory errors. We also made sure to use tools like torch.no_grad() during inference to save memory. For BNN, we optimized training by limiting the number of samples and simplifying the loss calculation.

There were a few other practical tweaks that helped. For example, using grid search, we tuned key hyperparameters like the number of trees in the random forest and the k value in KNN. For deep models, we didn't have the resources to do large-scale tuning, so we rely on common defaults such as a learning rate of 0.001 and a batch size of 64. We also manually set the number of epochs and monitored the loss of training to prevent overfitting or undertraining.

Looking ahead, we see several ways to make the models even more efficient or accurate. An option is to apply INT8 quantization to the simple CNN so that it can run on mobile or embedded devices with much faster inference times. We also plan to experiment with data augmentation techniques, such as random cropping, flipping, and color jittering, which could improve performance on VGG-16 and ResNet-18. For BNN, we would like to explore temperature scaling to improve calibration and possibly reduce log-loss. Finally, we are interested in using Grad-CAM to visualize which parts of the image our CNNs are focusing on, which could help us better understand how the models make decisions.

Overall, our experiments confirmed what we expected: deep convolutional models outperform traditional machine learning methods on image tasks like CIFAR-10, especially when it comes to accuracy and generalization. However, they also come with real trade-offs in terms of training time, memory usage, and hardware requirements. Balancing these factors was a key part of our work, and we think the results offer a useful comparison for anyone facing similar constraints.

3 Experiments

3.1 Experimental Setup

We used Google Colab to implement and conduct all the experiments. Google Colab uses Intel Xeon CPU with 2 vCPUs, 13GB of RAM, and an Nvidia Tesla T4 GPU with 8GB of Memory. For classic algorithms, we use scikit-learn 1.5, and we use PyTorch 2.x for all other deep-learning-based algorithms.

We do not perform extensive preprocessing on the CIFAR-10 dataset, as it is already clean and consists of images with uniform dimensions. However, for models that require a specific input size, we resize the original images accordingly. Besides, we use PCA to reduce the dimensions for classic models before training.

3.2 Experiment Results

Table 1 shows the result summary. Logistic regression can only reach an accuracy of 0.41, but its advantage is that it is very simple so it has the fastest training and inference speed. Random Forest can also reach an accuracy of 0.41. But the training speed is very slow, and the time increases with the increase in the number of trees. The KNN has the worst performance, and it has a slow inference speed since it needs to calculate the distance to all points. The MLP can reach a fair accuracy of 0.56, but it is limited in capturing the image's texture and pixel correlation. BNN can achieve an accuracy of 0.63, but the training speed is the slowest since it has to use variational inference to approximate the posterior distribution over weights, which is very time-consuming. CNN can achieve the best results, and CNN-based modern deeper architecture can further improve the performance. This is because the CNN uses convolutional filters that slide over the image and learn local patterns like edges and textures, which are suitable for images.

Model	Accuracy	Log-Loss	Training Speed	Inference Speed
Logistic Reg	0.41	1.71	Very Fast	Very Fast
Random Forest	0.41	1.90	Slow	Medium
KNN	0.36	12.28	No Training	Slow
MLP	0.56	1.75	Medium	Fast
BNN	0.63	1.04	Very Very Slow	Fast
Simple CNN	0.70	0.87	Very Slow	Medium
Vgg-16	0.78	0.78	Very Slow	Fast
Resnet-18	0.76	0.75	Very Slow	Fast

Table 1: Performance Summary

An interesting comparison is between BNN and CNNs. While BNN has the advantage of modeling uncertainty by learning distributions over network weights, it often requires significantly more computational resources due to the added complexity of sampling, approximating posterior distributions, and maintaining multiple parameters per weight (e.g., means and variances). Therefore, under our limited GPU time, the 5-epoch training may not make BNN converge, which makes it underperform CNN.

Moreover, BNN is based on the assumption that network weights follow a probabilistic distribution. This may not be well suited to image data, which has many complex and random patterns. In contrast, CNN does not have such probabilistic constraints. This direct learning of weights leads to more precise feature extraction, which is suitable for image classification.

4 Overcoming the Difficulties

4.1 High Dimensionality and Visual Class Confusion

Training on the original 3072-dimensional pixel vectors (32x32x3) from CIFAR-10 was computationally intensive and tended to overfit. To solve this problem, we applied Principal Component Analysis (PCA) to reduce the dimensionality to 200 components while preserving 95% of the variance. This optimization significantly reduced memory usage and training time, facilitating faster and more efficient training of traditional models such as logistic regression and KNN.

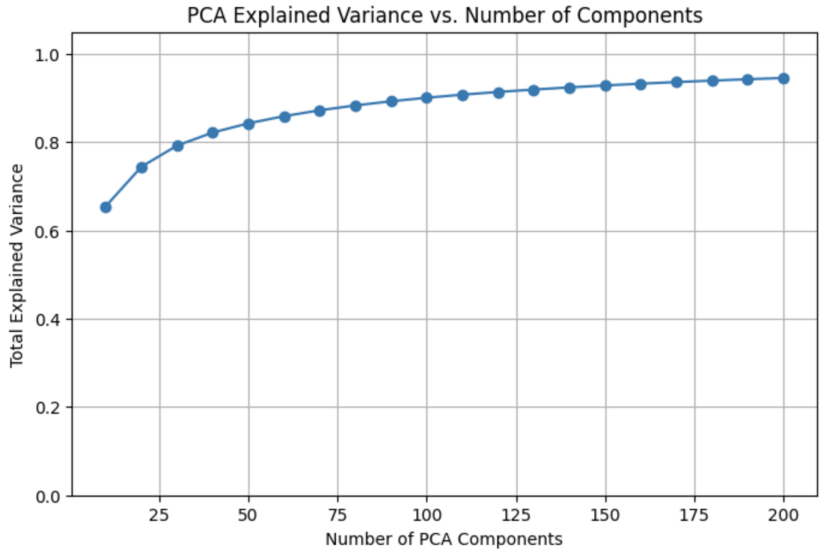


Figure 1: PCA Explained Variance vs. Number of Components

Despite the reduced dimensionality, the traditional classification model (logistic regression, KNN, random forest) still struggled to distinguish visually similar categories such as cat vs. dog and deer vs. horse. These models do not capture spatial information, making it difficult to distinguish fine details between similar classes.

To address this limitation, we introduced Convolutional Neural Networks (CNN), which leverage spatial convolutions to capture textures, edges, and hierarchical patterns. Simple CNN outperformed traditional models by learning spatial dependencies, while deeper architectures such as VGG-16 and ResNet-18 further improved the accuracy of the classification.

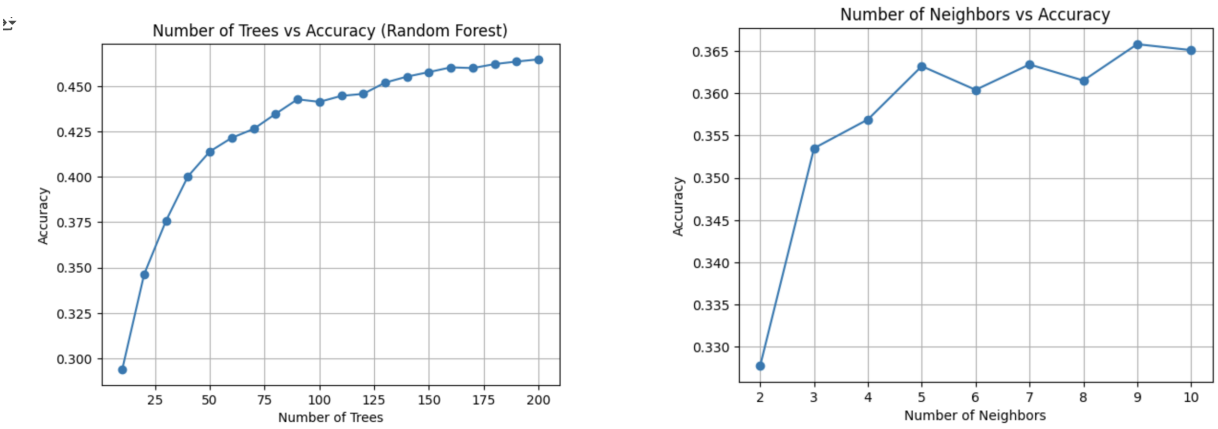
4.2 Hyperparameter tuning

To improve model performance, we adjusted key parameters that affect learning and accuracy. We used grid search to find the best settings.

- **n_estimators for Random Forest:** We tested values from 10 to 200. The best balance

between accuracy and speed was achieved at 50. Although 150 and 200 slightly improved accuracy, the extra memory and training time were not worth it.

- **k values for KNN:** We tried values from 1 to 10. The best result was found at $k = 5$, providing a good balance between bias and variance.



(a) Number of Trees vs Accuracy (Random Forest)

(b) Number of Neighbors vs Accuracy (KNN)

Figure 2: Hyperparameter tuning results for Random Forest and KNN

For deep learning models like CNN and MLP, we used a standard learning rate of 0.001 and set a fixed number of training epochs. Due to GPU limits in Google Colab, we could not perform extensive grid searches. Instead of early stopping, we manually set the number of epochs and monitored the loss to ensure the models trained properly without overusing memory.

4.3 Model Convergence and Stability

During the training of Logistic Regression and MLP without PCA, we faced convergence problems. Even after increasing the maximum iterations to 500 for Logistic Regression and 20 for MLP, both models still failed to converge. This was mainly due to the high dimensionality of the input (3072 features) and the lack of dimensionality reduction.

The warnings showed that the solvers (lbfgs for Logistic Regression and adam for MLP) struggled to find a solution in such a large feature space. After applying PCA to reduce the features to 200, both models converged smoothly and became more stable. This shows that PCA not only speeds up training but also helps models converge better in high-dimensional settings.

For CNN, we did not have this problem since it learns features through convolutional layers. However, memory usage was still a concern, which we managed with batch processing and memory-efficient testing.

4.4 Deep Learning Model Training

Training deep networks like VGG-16 and ResNet-18 was challenging due to memory limits in Google Colab. To solve this, we used a two-step strategy:

First, we started with a Simple CNN for quick testing. It allowed us to try different settings without running out of memory. We trained it for 5 epochs with a learning rate of 0.001 and a batch size of 64. For evaluation, we used `torch.no_grad()` to save memory. Second, we used VGG-16 and ResNet-18 for better accuracy during final evaluations. We fine-tuned both models for CIFAR-10 by adjusting the last layers to 10 classes and training with the Adam optimizer at a learning rate of 0.0001.

- **VGG-16:** Trained for 5 epochs, reached 78% accuracy with a log loss of 0.78.
- **ResNet-18:** Trained for 10 epochs, reached 76% accuracy with a log loss of 0.76.

We only used these deeper models during the final tests to avoid memory crashes. This two-step process helped us train efficiently and avoid out-of-memory issues in Colab.

4.5 Resource Optimization for Bayesian CNN

Training the Bayesian CNN required more memory and computation due to multiple forward passes for uncertainty estimation. Unlike standard CNNs, Bayesian CNNs use Monte Carlo Sampling during inference to estimate the distribution of weights, increasing memory usage and runtime.

To reduce resource consumption, we applied two optimizations: First, we limited the number of forward passes to 5 instead of the usual 20–30, reducing memory usage while preserving useful uncertainty estimates. Second, we adjusted the complexity cost weight during ELBO optimization to save memory while maintaining expressiveness.

4.6 Future Improvements

To further optimize the performance and deployment of the model, we propose four main improvements. First, we plan to apply INT8 quantization to the Simple CNN to achieve faster inference (approximately 0.5 ms) on mobile and Jetson devices. Second, we will introduce random crops, flips, and color jitter as data augmentation techniques to boost the accuracy of VGG-16 and ResNet-18 beyond 80%. Third, we intend to use temperature scaling on the Bayesian CNN to reduce log-loss and improve predictive reliability. Finally, we aim to apply Grad-CAM on all CNN models to visualize attention, enhancing interpretability and understanding of model decisions.

Appendix

The full source code used in this project is available on GitHub at the following URL:

`https://github.com/lh935617470/STAT542-Final-Project`

The repository includes all Python implementations, data preprocessing steps, modeling pipelines, and result visualizations discussed in this report.