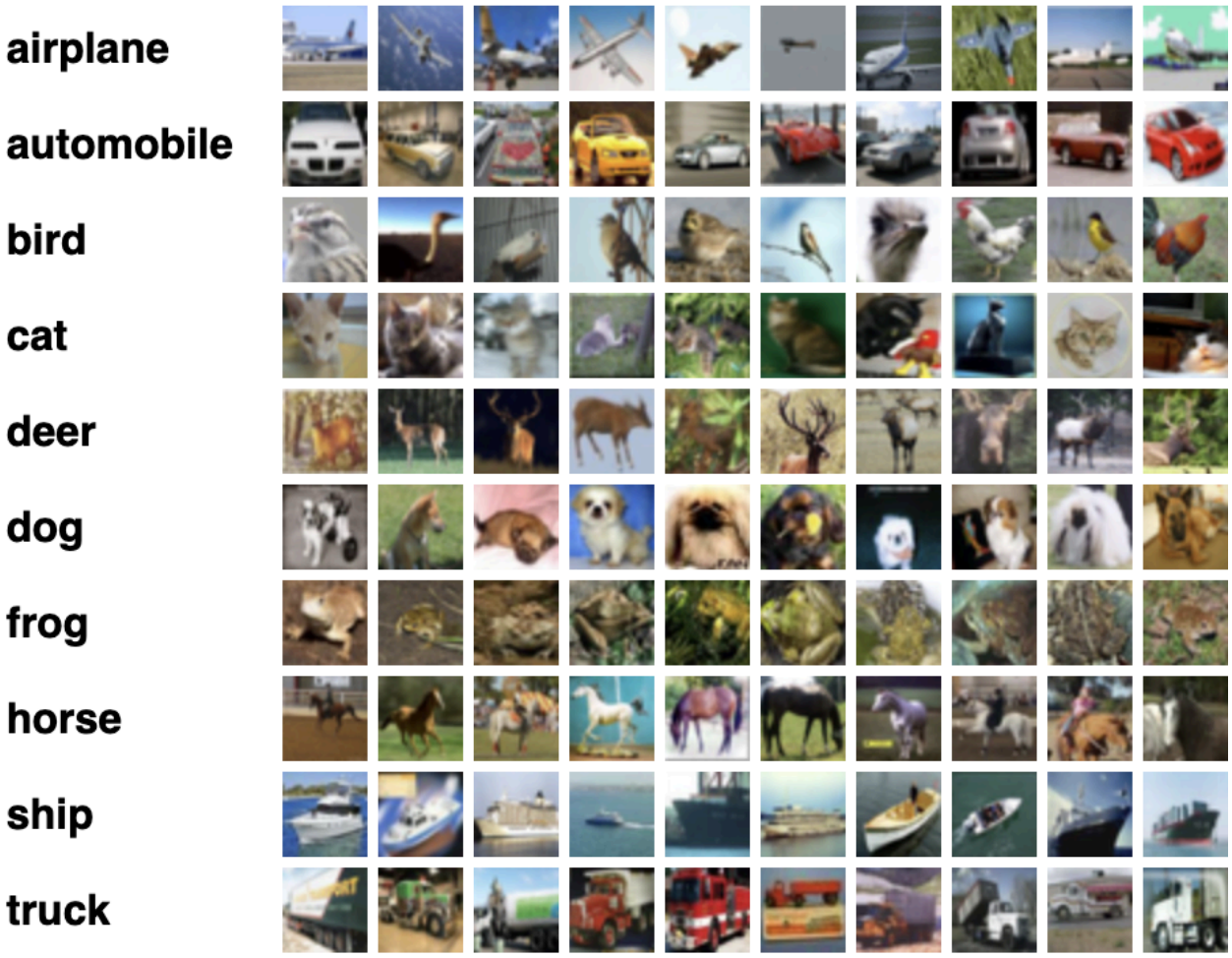
CIFAR-10 Data Set Learning Curve Report



Authors: Vishwanath Durgam, Dickson Yh Hong, and Ben T. Boben

Summary

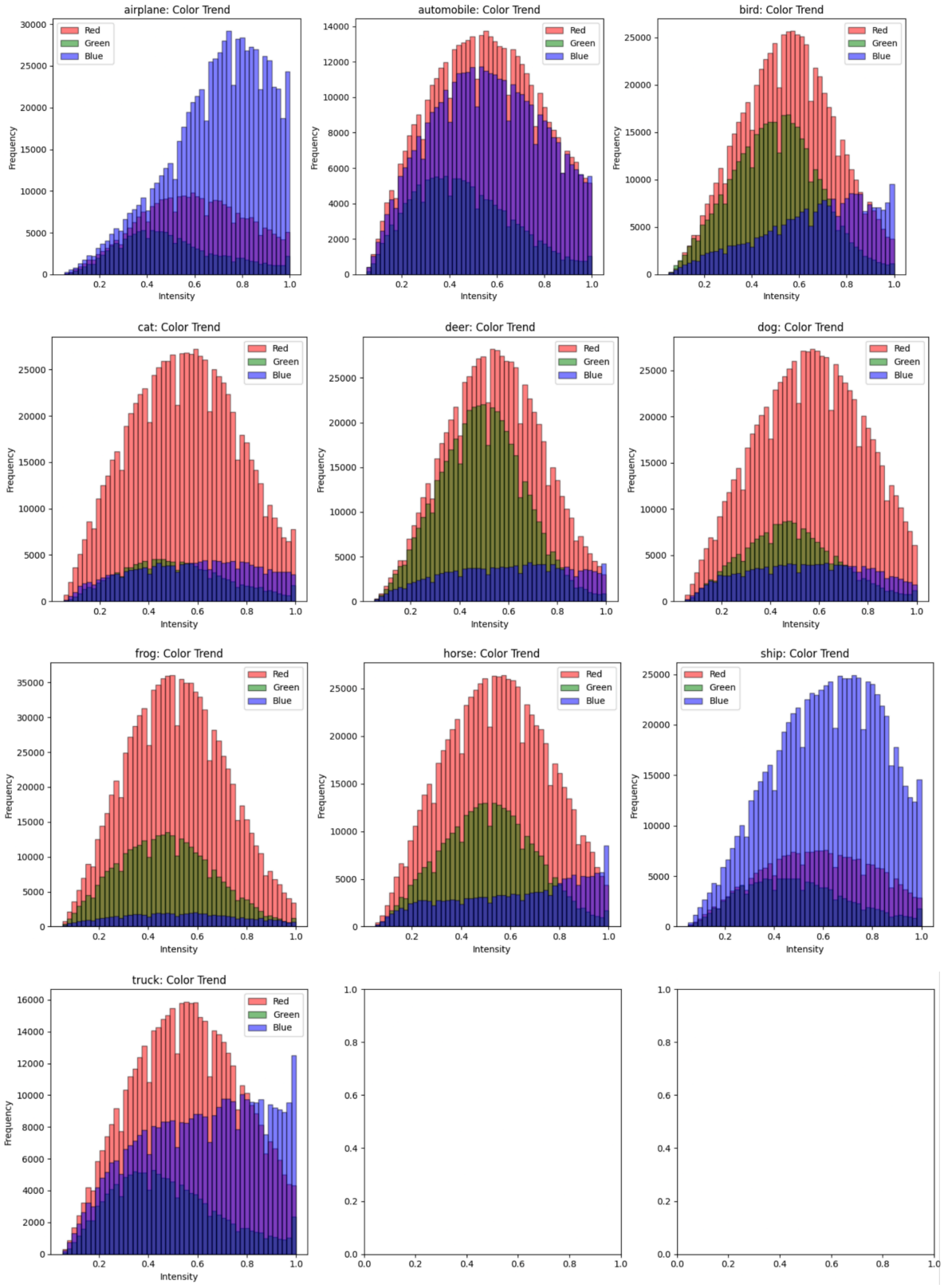
In our project, we explored the machine learning classification methods, k-Nearest Neighbors (kNN), Logistic Regression, Multilayer Perceptrons (MLP), and Random Forest, on the image-centric CIFAR-10 dataset. Our investigation aimed to understand and compare the performance of these classifiers in handling image recognition tasks. A key conclusion from our analysis was the variance in effectiveness and computational efficiency across the methods, with specific techniques showing superior performance in accuracy and training time.

Data Description

The CIFAR-10 dataset (Canadian Institute for Advanced Research, 10 classes) is a widely used dataset for machine learning and computer vision research. It consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 testing images. The ten classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Prior to training, we chose to visualize the color trend for each class in order to get a better idea of what the classifiers will generally be learning. Specifically, we graphed the frequency of each color channel (R, G, B) for each image in the training dataset, and grouped the results by class. Iterating through each pixel was too time consuming, so as a heuristic, we accumulated the values of every *third* pixel instead. Furthermore, we omitted pixels that were near-greyscale, and for each pixel, we only took the channel with the highest intensity. We found that a vast majority of the images contained a large amount of medium-intensity red as the dominant color, while specific objects such as Airplane and Ship have a strong blue as the dominant color, which makes sense (sky and ocean). Although we are unable to visualize color combinations due to running time constraints, we can make a guess that the large amount of red-dominance likely appeared due to the combination of red and green being brown, a natural color, while blue and its combinations (cyan/purple) are rare in nature. As for the remaining classes where there's proportionately more blue than green, such as automobile and truck, they're in the same class: automobiles, and we can see that the blue tends to lean on the higher-intensity side, which suggests "artificial" coloring like paint.

The data visualization is provided on the next page. The intensity represents the normalized color values (originally 0-255, now ranging between 0-1), while the frequency represents the number of pixels that have the respective color and intensity as the largest value of the three channels (r,g,b).


airplane: Color Trend
automobile: Color Trend
bird: Color Trend
cat: Color Trend
deer: Color Trend
dog: Color Trend
frog: Color Trend
horse: Color Trend
ship: Color Trend
truck: Color Trend
Red
Green
Blue
Frequency
Intensity

Classifiers

kNN Classifier

The k-Nearest Neighbors (kNN) classifier predicts the label of a given data point by looking for the k closest label data point(s) in the feature space. That unassigned data point becomes assigned, and is given the most common label among its nearest neighbor as its classification. It is a very simple method that helps us to classify based on the majority vote of neighbors. This was used to train the CIFAR-10 dataset to depict the learning curves and error rates using the kNN model.

- **Hyperparameters Tested:** The primary hyperparameter that was tested is the number of neighbors (k). The value of k is very important in kNN algorithms as it helps to determine the number of nearest neighbors to consult.
- **Value Range Tested:** k values of [1, 2, 5, 10, 50, 100, 110] are used for analyzing the error rates. The optimal k value is found by identifying the value k that yields the lowest validation error.

Logistic Classifier

Logistic regression uses linear regression as an underlying model, then applies a non-linearity "activation" function, such as sigmoid, in order to threshold the linear model to produce class predictions through probability values. In this case, we used the Stochastic Gradient Descent version for a faster running time so that more models could be created.

- **Hyperparameters:** Aside from the activation function type, the main hyperparameters of a logistic classifier are the regularization strength and type of regularization.
- **Value Range Tested:** Using random search, a few models were trained with varying regularization settings: Regularization strength was in the range [0,4], and regularization type was either L1 or L2.

MLP Classifier

A Multilayer Perceptron (MLP) is a class of feedforward artificial neural network that consists of at least three layers: an input layer, one or more hidden layers, and an output layer. Each layer is made up of neurons, where each neuron in one layer connects to every neuron in the next layer, with associated weights and biases. MLPs use a backpropagation algorithm for training the network, effectively adjusting the weights and biases to minimize the error between the predicted and actual outputs. Key hyperparameters in MLPs include the number of hidden layers, the number of neurons in each hidden layer, the activation function and the learning rate.

- Hyperparameters: There are many hyperparameters for the MLP classifier, including number of hidden units and layers, number of iterations, and activation function type.
- Value range Tested: In our MLP model training on the CIFAR-10 dataset, we focused on a single hyperparameter, setting hidden_layer_sizes to 512 neurons in one layer. Additionally, the model was configured to perform up to 100 training iterations and employed an early stopping mechanism with a validation fraction of 10% to prevent overfitting.

Random forest Classifier

A Random Forest Classifier is an ensemble of varying Binary Tree Classifiers, through the use of intentional complexity-reduction methods such as bagging and feature omitting. This classifier obtains the results from each of its classifiers in a “voting” style to obtain the majority prediction as the final output.

- Hyperparameters: The main parameters of the Random Forest Classifier are number of estimators in the ensemble, maximum depth of the trees, and the number of features that can be selected from during each split decision.
- Value range Tested: We focused on the n_estimators hyperparameter, by changing it to 100 trees. Then, observed the efficiency across different varying data set levels.

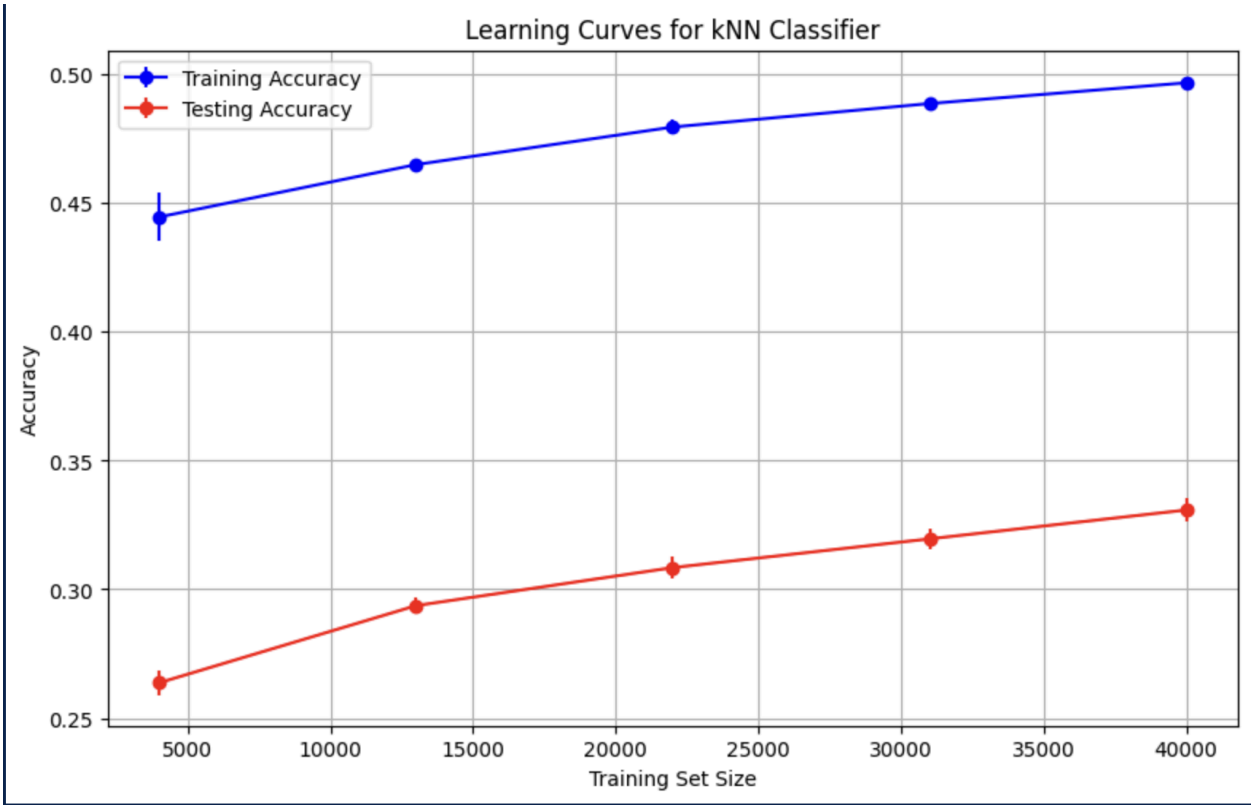
Software Used: Throughout the project, we found the Matplotlib, Numpy, and Scikit-Learn libraries to be of great help.

Experimental Setup

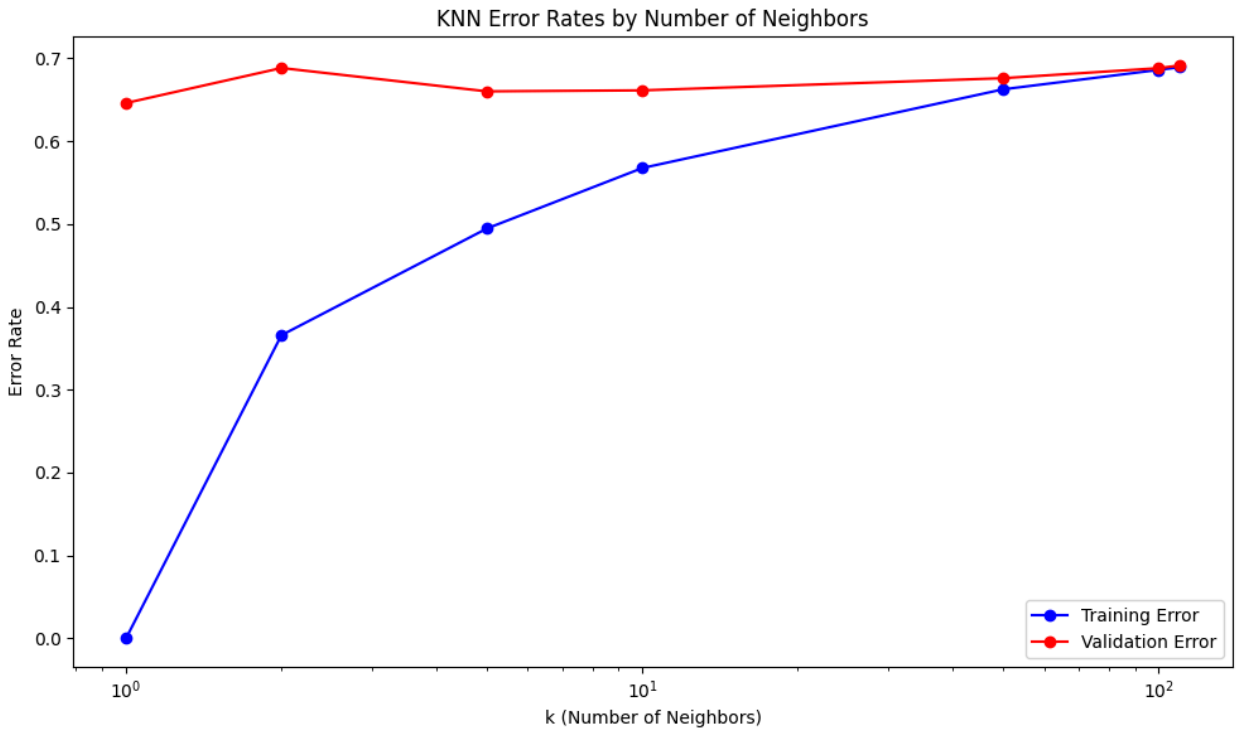
What we experimented with: We investigated the learning curves based on varying training set sizes of 4000, 13000, 22000, 31000, 40000 to access the model performance and generalizability of the different classifiers. We also tested different hyperparameters including the number of neighbors in kNN, regularization strength in logistic regression, layer sizes in MLP, and the number of trees in random forests.
How did we partition our data?: The CIFAR-10 dataset was divided into different distinct sets for training, validation, and testing. In our testing set we used 80% of the 50,000 images (40,000 images), in the validation test we used 20% of the 50,000 images (10,000 images), and in the testing set we used 100% of the 10,000 images.
How did we select hyperparameters?: Random search was used on relevant hyperparameters for each classifier. Multiple models were trained, with their results being graphed and analyzed. The best-performing ones were kept for benchmarking on varying dataset sizes.

Experimental Results

kNN Classifier

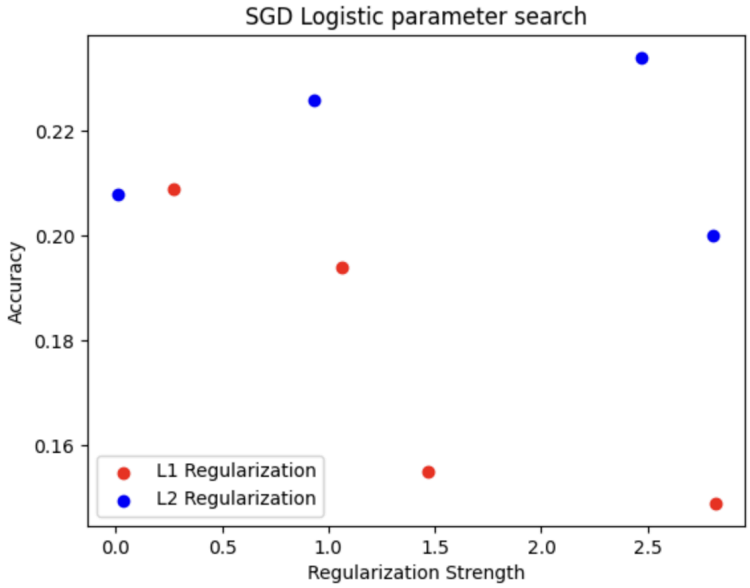


In this chart it shows the learning curves for the kNN classifier. We plotted the training set size against the accuracy for both training and testing data sets. We used training set sizes of [4000, 13000, 22000, 31000, 40000]. The error bars are used to signify the variability among the 3 runs we made, for each data set size, to check for any major variation within the accuracy of each run. The graph is used to understand how well the classifier is learning as the amount of training data increases. From this chart, we can observe that as the training set size increases, the accuracy on the training set also increases. However, there is a significant decrease in learning rate after a training set size of 13000, indicating that adding more data does not significantly improve the model's performance on the test set. The training accuracy is relatively stable across all training sizes, which is expected for a classifier model such as kNN, which tends to always have a high accuracy on the training data set.

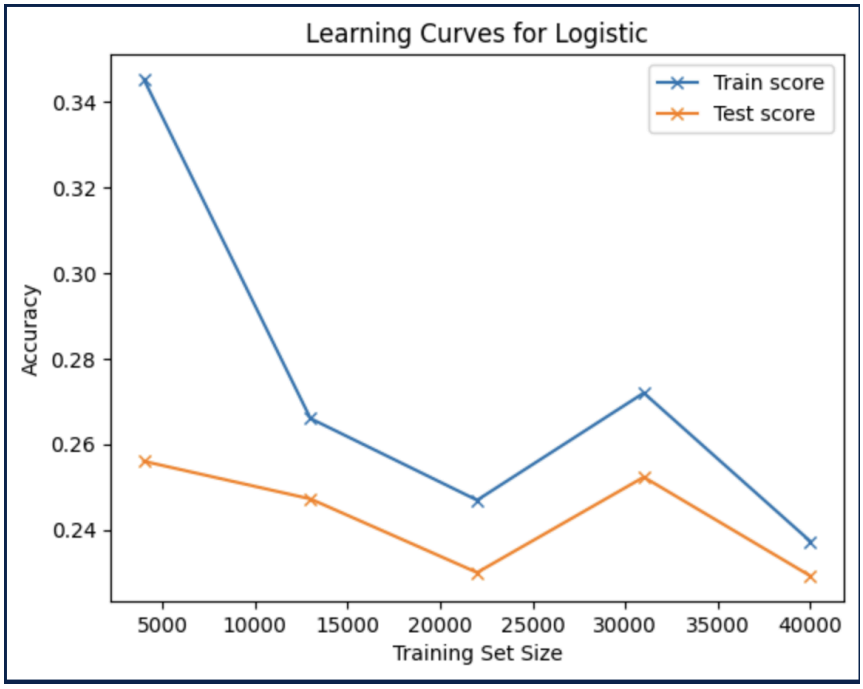


This chart is used to semi-log plot the relationship between the number of neighbors (k) in the kNN classifier and the error rates in both the training and validation (testing) datasets. The x axis is used as a logarithmic scale for the wide range of k values that we tested. This chart is testing k values of [1, 2, 5, 10, 50, 100, 110]. This made using a logarithmic scale very sensible when it comes to plotting these points to calculate the best value of k. In the chart, the testing error rate can be observed to decrease rapidly as k increases from 1 to 10 and then stabilizes, helping suggest the fact that a small number of neighbors are not sufficient for the model to generalize well, but too many neighbors also does not provide better generalization.

Logistic Classifier



This chart helps us visualize the random search that was done on the Logistic classifier. The x-axis represents the regularization strength of the model, and the color is used as the second dimension to represent the type of regularization used. The y-axis shows the testing accuracy of the model. We trained 8 different models (i.e. 8 different combinations of parameters) by randomly selecting a regularization type (50/50 for L1 and L2), and also randomly sampling from a gaussian distribution of regularization strengths within the range [0,4]. None of them did too well compared to the other classifier types, but the best-performing one out of these 8 models seems to have a regularization strength of approximately 2.5, using the ridge (L2) regularization. It is worth noting that the random sample here happened to not pick any regularization strengths beyond 3, so that was a downside of this random search.

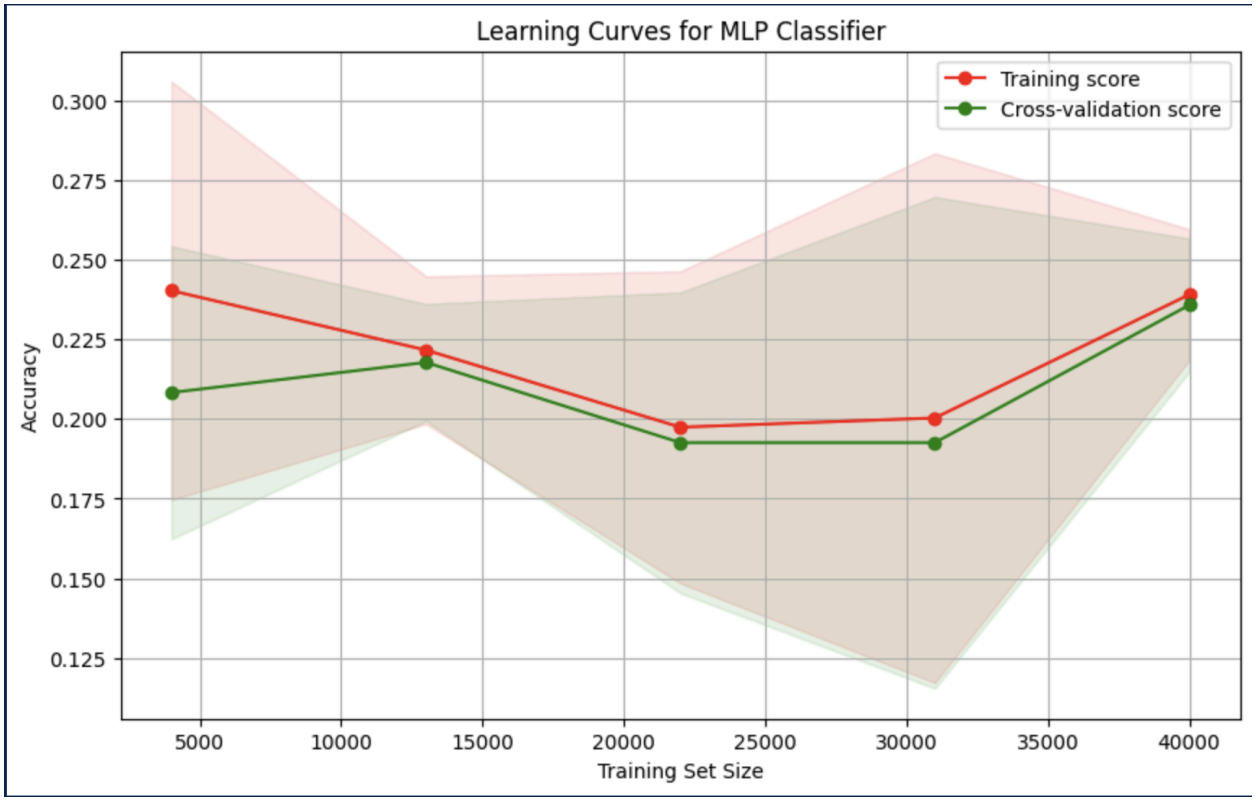


This is certainly an unusual graph, but we can try to make sense of it. The logistic classifier can only create a linear boundary using the given features. As our image dataset is so multi-dimensional and complex, the boundary is never able to converge, regardless of the training set size. In fact, during training, none of the models converged within the iteration limit.

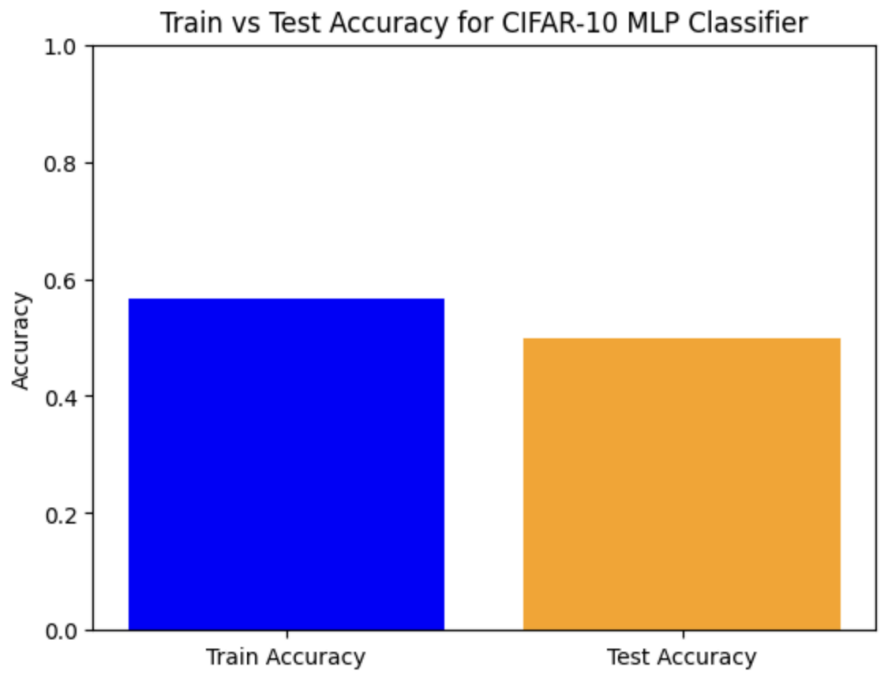
For this reason, the model is only able to get a fraction correct with a small training set size. There is too much data and variation for the model to fit correctly, and more data points will only result in the classifier "falling behind" (accuracy lowering). I believe this is the reason why the accuracy tends to decrease as the dataset size increases. Finally, the accuracy looks like it's not monotonic (see the jump in accuracy for a training size of 31,000 points), simply because this classifier is not made for learning this type of data. The only possible explanation for a graph like this is that the classifier is so underfit that it's almost making random guesses at that point. As a result, the results are somewhat erratic, even though there seems to be a trend in the first three trials (4,000-22,000).

In conclusion, logistic classification is not the best model when it comes to complex problems (i.e. lots of features), as the boundary it creates is too simple to completely capture all the nuances in the data.

MLP Classifier

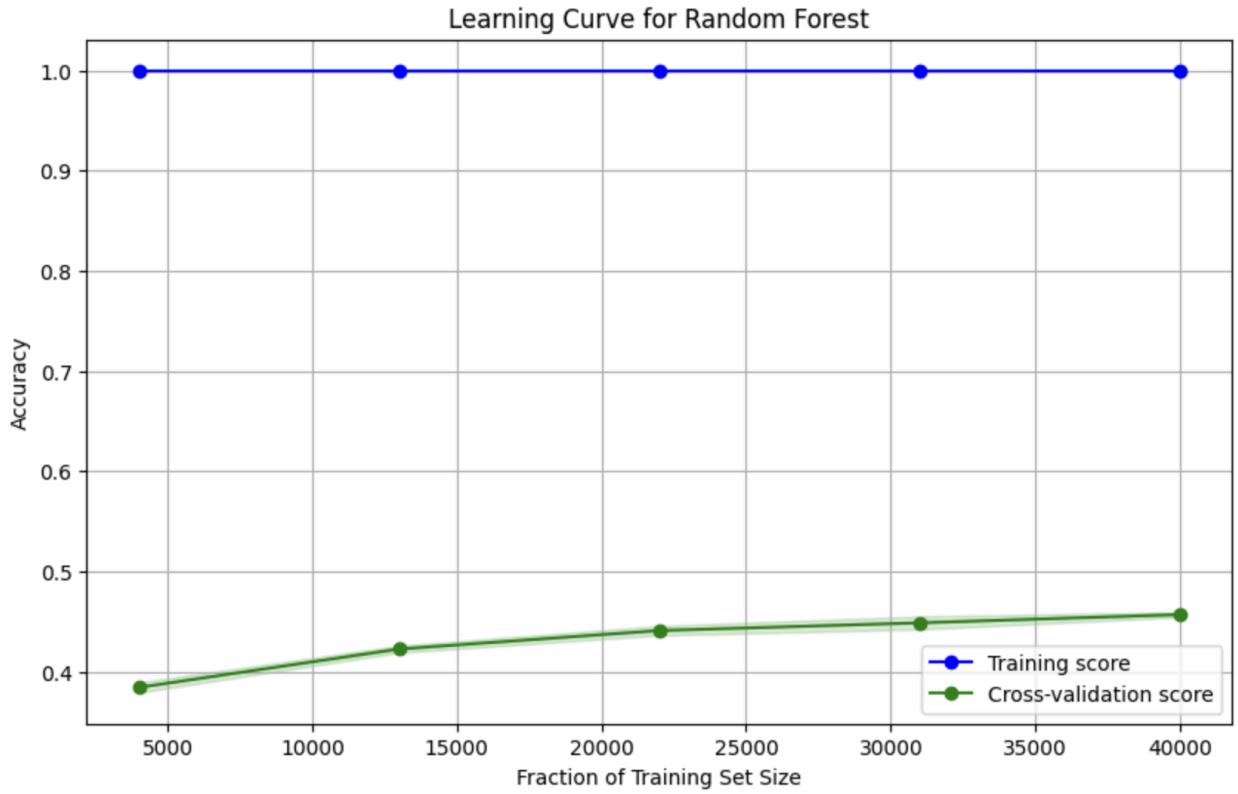


In this chart, we observe the learning curves for an MLP classifier with the accuracy plotted against training set sizes of [4000, 13000, 22000, 31000, 40000]. The inclusion of error bars indicates the variance across different trials, providing insight into the consistency of the model's performance with increasing data. Notably, the graph shows an increase in accuracy on the training set with more data, yet this trend tapers off, especially after the 13000 data point mark, suggesting a diminishing return on accuracy from adding more training data. The training score demonstrates a slight decline as the dataset size increases, which is typical due to the model's increasing difficulty in fitting a larger dataset perfectly. However, the relatively close performance of training and cross-validation scores, especially in larger training sets, implies that the model generalizes well and is not overfitting. This is an encouraging sign of a well-tuned model.



The stark contrast between the high training accuracy and the lower test accuracy is indicative of overfitting; the MLP model appears to have learned the training data well, possibly too well, as this does not generalize effectively to the unseen test data. This discrepancy suggests that the model's complexity may need to be adjusted, or regularization techniques could be introduced to mitigate the overfitting and enhance the model's ability to generalize from the training data to new, unseen data.

Random Forest Classifier



This learning curve graph for a Random Forest classifier indicates the training and cross-validation scores as a function of the training set size. Notably, the training score remains consistently high across all sizes, showcasing the model's capability to fit the data well. However, the cross-validation score starts significantly lower but improves as more data is provided, suggesting that the model is learning and generalizing better with more data. The persistent gap between training and validation accuracy could imply a certain level of overfitting, where the model's complexity may need to be addressed, or additional data might be necessary to improve the model's generalization on unseen data.

Insights

Because this is our first time training on colored images, we were surprised at the “scale” of processing the data, such as the amount of data points involved, the dimensionality of the data, and the time it took to train even a single model. It seems to be the case that high-variance models (such as kNN with a low k, and random forests) and complex models (such as Neural Networks) end up being relatively competent when it comes to such complicated data, while simpler decision models such as the logistic classifier yielded poor results, as expected. This testing confirmed the generalizations that we learned in class about how the model we use should be on the same “level” as the complexity of the dataset. An additional note: While we did

not expect kNN to perform very well, it somehow managed to achieve almost 50% accuracy on the dataset, which is impressive.

Contributions

Ben Boben: I focused on the implementation of the k-Nearest Neighbor (kNN) classifier, I was responsible for experimenting with the different values of k to determine the most optimal hyperparameter, also experimenting with different dataset sizes to obtain the learning curve. Additionally, I also helped visualize the CIFAR-10 dataset by developing clear colored histograms that displayed the color distribution trends of the various image classes.

Dickson Hong: I focused on the implementation of the Logistic classifier. I experimented with the regularization options to optimize the amount of fitting on the dataset. In addition, I trained the model on varying dataset sizes with fixed hyperparameters to analyze the impact of data size on learning. Finally, I assisted in the design of the data visualization, then interpreted the visualized data.

Vishwanath Durgam: I trained and analyzed both Multilayer Perceptron (MLP) and Random Forest classifiers, meticulously tuning their hyperparameters, such as the number of hidden layers and trees respectively, while also adjusting dataset sizes to construct learning curves and pinpoint optimal model configurations.

Github Repository:

https://github.com/benboben03/CIFAR-10