Airplane or Ship?

An Investigation of Classification Methods & Strategies for CIFAR-10

| By Colin Yee, Reece Buyan, Ky-Vinh Mai

Summary

The goal of this project is to train, modify, and compare different classifiers on the **CIFAR-10** dataset and utilize strategies and model configurations that would maximize the accuracy for each classifier. After extensive testing with the dataset and models, we found that the Convolution Neural Network (CNN) did the best among the different classifiers (kNN, logistics, FNN) in identifying and labeling CIFAR-10 images by a significant margin.

Data Description

CIFAR-10 consists of 60,000 RGB images. Each image is 32x32 pixels and is labeled as an airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. There is an equal amount of images for each class (10,000 images for each class). Images in the same class share common characteristics like the object's shape or color palette, but there are variations in color, background, or angle. For additional information in CIFAR-10, refer to the article here: https://paperswithcode.com/dataset/cifar-10.

Classifiers

1. K-Nearest Neighbor (kNN)

The kNN classifier would calculate the distance between any new datapoint (e.g. image) and all other data points in the training set. The model finds the K nearest data points to the new data point and assigns the majority class (e.g. label) among these neighbors to the new data point. We will be using `KNearestNeighbor` from `scikit-learn` and used hyperparameters:

- `n_neighbors` {x > 0, x <= size of dataset}: The number of neighbors closest to a new data point that will determine its class.
- `weights` ['uniform', 'distance']: Weight/influence of neighbors. If 'distance', neighbors closer to the new data point will have more influence on its class.

2. Logistics Classifier

A logistic classifier is a binary classification model that predicts whether or not a data point is in a particular class or not. It computes a weighted sum of the datapoint's features and applies a sigmoid function to calculate its probability of the point being in that class. If the number of classes exceeds two, it will create K logistic functions and apply a softmax function to find the probabilities for a point to be in a certain class. We assign the class with the highest probability to the new data point.

We will use `LogisticRegression` from `scikit-learn` and use hyperparameters:

- `penalty` ['None', 'l2', 'l1', 'elasticnet']: Regularization penalty applied to the model
- `solver` ['liblinear', 'saga']: Algorithm used to optimize the training time for the model
- `fit_intercept` [True, False]: Whether or not we have a bias in the model

- `multi_class` ['auto', 'ovr', 'multinomial']: By default ('auto'), the model will change how it calculates probabilities based on whether it's handling 2 or more classes.

3. Feedforward Neural Network (FNN)

A Feedforward Neural Network is a network of interconnected nodes ("neurons") arranged in different layers. These nodes would perform mathematical calculations on the data to produce outputs that are further processed in the subsequent layer or be utilized to create class probabilities for a given data point.

For our FNN, we used `pytorch` to create our own model using `nn.Model`. We control for these hyperparameters:

- `Hidden sizes` [(512, 256), (1024, 512), (2048, 1024)]: the number of nodes in the 1st and 2nd layer respectively
- `Learning rates` [0.001, 0.005, 0.01]: the amount the weights are updated during training
- `Batch sizes` [64, 128, 256]: size of batches for stochastic optimizer
- `Normalizing Functions` [None, Layer Norm+DropOut]: If not 'None', we will normalize the activations of each neuron across the features within a layer (Layer Normalization) and randomly drop a fraction of neurons in a layer during each training iteration (Dropout) to reduce overfitting.

4. Convolution Neural Network (CNN)

A Convolution Neural Network is similar to a FNN but utilizes filters to create layers of 1 or more response channels that can be used to identify patterns (i.e. bird's beak, wing of a plane, etc.). The data passed through these convolution layers are then flattened and passed through a FNN to predict the class probabilities for a given data point.

Just like FNN, we used `pytorch` to create our own model using `nn.Model`. We limited the FNN portion to 2 hidden layers, the ReLU activation function, 'adam' (stochastic gradient-based optimizer) solver, a 0.001 learning rate, and a 256 batch size. By default, we have max pooling for dimensionality reduction and control these CNN hyperparameters:

- `Num filters` [16, 32, 64]: the number of filters/channels in a convolution layer
- `Kernel sizes` [3, 5]: The n x n size of the filter

Experimental Setup

For general experimentation, we split the 60,000 images into 60% training, 20% validation, and 20% testing. In general, 60% of the data would go to testing, 20% goes to validation, and 20% goes to testing. We also standardized the color data, so the models would be more resistant against noise and treat all features equally to the analysis.

(kNN)

For the kNN classifier, we utilized these strategies and model configurations to try to improve the classification accuracy:

1. Dimensionality Reduction (RGB vs Grayscale CIFAR-10 Images)
2. Bagging Classification with Bootstrapping

I would select the best values for `n_neighbors` and `weights` for the RBG & Grayscale images using `GridSearchCV` to conduct 3 fold cross validations. The training and validation sets were used to determine the best hyper-tuned model, and the testing set was used to determine the final accuracy of the model. Once I determine the most accurate model, I would bag the classifier to see if a bag of smaller kNNs with bootstrapping would perform better than one large kNN.

(Logistics)

For the logistic classifier, we utilized these strategies and model configurations:

1. Changed `LogisticRegression` parameters (regularization & solver)
2. Ada-Boosting

I would change various parameters of the `LogisticRegression` class in order to decrease time spent on fitting the data and possibly increase the model accuracy. I could have applied grayscale to the CIFAR-10 images before fitting them. But since the accuracy of the kNN model dropped when we grayscale the CIFAR-10 images, I decided to focus instead on Ada-boosting, using the best parameters from my previous iterations to fit a new model.

(Feedforward Neural Network)

We first needed to preprocess the data and turn it into tensors so the neural network could take it in as input. We limited our FNN to 2 hidden layers, the ReLU activation function, and 'adam' (stochastic gradient-based optimizer) solver. We trained the model on 10 epochs (which represent the number of passes the training set takes around the model. Each epoch updates the weights of the model to improve the model's accuracy).

I created my own grid search function to find the best hyperparameters for the hidden sizes, learning rate, etc.

(CNN)

There were issues with the model taking the appropriate batch sizes, so the dataset had to be specifically divisible by 256, and the remainder would be thrown out. We used a basic CNN model with 2 convolutional layers and 1 max pooling layer while other hyperparameters like hidden sizes, learning rates, etc. were taken from the best performing FNN. This was to see how much those additional layers could improve the model performance. We trained the model on 10 epochs.

Experimental Results

Model	Note	(Best) Parameters	Train Accuracy	Test Accuracy
kNN	Regular RGB	n_neighbors = 7, weight = 'distance'	100%	35%
	Grayscale	n_neighbors =9, weight = 'distance'	100%	30%
	Bag+Bootstrap	base_estimator: (RGB, n_neighbors = 7, weight = 'distance) max_samples=1.0, max_features=1.0, bootstrap=True, n_estimators=17	100%	34%
Logistic Classifier	Changing Hyperparameters	penalty = 'l2', solver = 'saga', fit_intercept = True	100%	38%
	Ada-Boosting	base_estimator: (penalty = 'l2', solver = 'saga', fit_intercept = True), n_estimators=50, algorithm='SAMME'	100%	32%
FNN	Simple 2 layer FNN	hidden state (2048, 1024), learning rate = 0.001, batch size = 256	79%	51%
	Dropout + Layer Normalization	dropout rate = 0.4	59%	53%
CNN	Basic CNN Architecture	number of filters = 64, kernel size = 3	78%	69%

Insights

(kNN)

Strategies such as grayscaling and bagging kNN classifiers with bootstrapped features or samples reduced the average prediction time for the classifier. However:

- Grayscaling the images led to a non-beneficial loss of information.

- A bag of smaller kNNs with bootstrapped features and training data didn't outperform the large RGB model. It's possible that we could outperform the RGB model if we added enough classifiers. However, more testing is needed since the time it takes to classify images increases as we increase the number of estimators and max number of samples/features a model utilizes.

The only visible increase in accuracy came from setting `weight`='distance'. This makes sense because neighbors closer to a new data point would be more likely to be the data point's true class (which essentially regularizes kNN as we increase the number of neighbors).

But ultimately, kNN struggles to correctly classify CIFAR-10 images.

(Logistics)

When implementing the changes into the logistic regression model, There were a few things to note while experimenting with the hyperparameters:

- Changing the regularization from 'l1' to 'l2' increased the time to fit the data from 190 to 265 minutes. Changing the solver from 'liblinear' to 'saga' immensely decreased the fitting time down to 15 minutes.
- Even with the best hyperparameters for our base logistic regression model, Ada-boosting yielded a lower accuracy rate.
- Across all base hyperparameters for the logistic regression , ship pictures were the most accurately classified object in the dataset. However, for Ada-boosting, horses were the most accurately classified image.

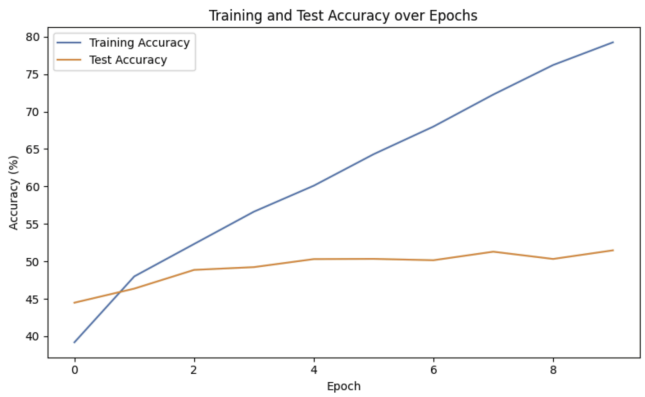
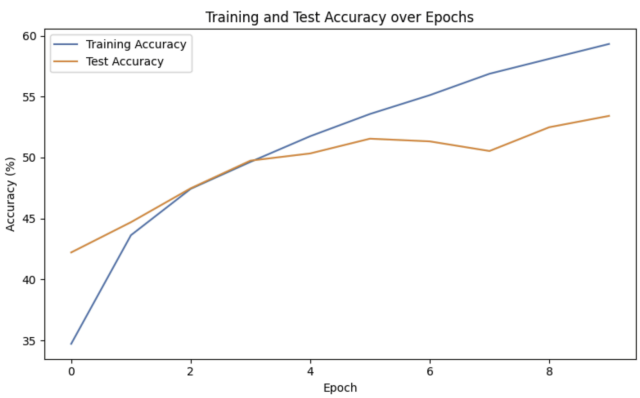
One of the explanations for the 'saga' solver running faster is because it utilized Stochastic Gradient Descent compared to liblinear, which struggles with convergence on large datasets due to its optimization approach.

However, the logistic classifier comes to similar accuracy results as the kNN Classifier and struggles to correctly classify CIFAR-10 images.

(Feedforward Neural Network)

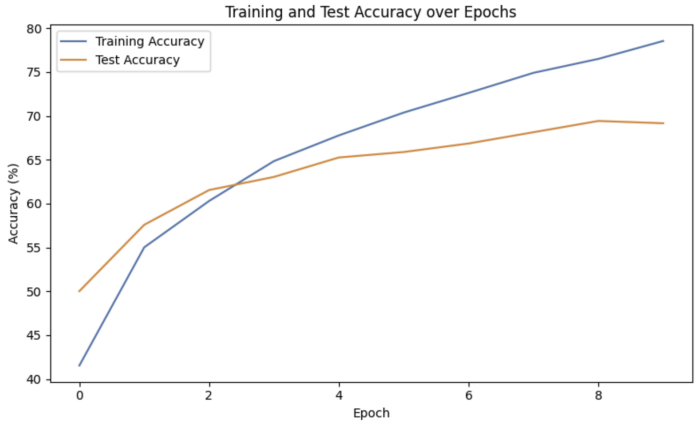
The Feedforward Network immediately outperformed the other models within the first couple epochs, despite having just 2 hidden layers.

Using strategies such as layer normalization and dropout gave a small performance increase for the model, by epoch 7 it had reached the basic FNN's accuracy. Its accuracy was a bit unstable and the training accuracy was lower than the basic FNN. A possible explanation for this is that dropout can sometimes lead to instability during training. The randomness introduced by dropout can cause fluctuations in the training loss, which might make it difficult to converge to an optimal solution. A solution would be to have it train on more epochs.

Plot for basic FNN:	Plot for FNN with dropout + layer normalization:
	

(CNN)

The CNN quickly got to the FNN's accuracy within the first epoch, getting a test accuracy of 50%. Yet interestingly, during the rest of the epochs it was only slowly improving performance. In the plot below, the test accuracy almost plateaus and is a little unstable between epoch 2 and 6. This likely is due to the learning rate needing to be a step size smaller as the epochs increase as in the later stages of the model training, so that the model can be more sensitive to the gradients pushing it towards a certain change in the weights. The CNN also took the longest to find the best hyperparameters for and the longest to train, because it's the largest and most complex model in this project..



Contributions

Colin Yee	I did all the work on the K-Nearest Neighbor models. I created the code for importing and preparing the CIFAR-10 dataset (grayscaling, flattening, standard scaling, etc.). I did half of the CNN and FNN writing in the report.
Reece Buyan	I was responsible for all of the work on the Logistic Classifier segment.
Ky-Vinh Mai	I was responsibles for the FNN and CNN coding and writing.