A Study of Image Classification ML Algorithms, Including a Custom Implementation of CNNs

Rudraksh Kapil
Department of Computing Science
University of Alberta
rkapil@ualberta.ca

Code repository: https://github.com/rudrakshkapil09/CMPUT566-Project

1 Introduction

Image classification is a common task in machine learning research. Given an image, the goal is to predict its class label. Many different types of machine learning algorithms have been proposed to classify images. For this project, I have empirically analyze the performance of various classification algorithms using the standard CIFAR-10 image dataset.

The four algorithms I have compared are k-nearest neighbors (KNN), softmax regression, and convolutional neural network (CNN). Each algorithm has its own set of hyper-parameters that needs to be tuned to maximize the performance of the algorithm. I have also included a random guessing algorithm to act as a trivial baseline for drawing comparisons.

Through this project, I have empirically analyzed which machine learning algorithms work better than others for image classification on the CIFAR-10 dataset, and under what specific choices of hyper-parameters. Moreover, the benefit of implementing a custom CNN from scratch is twofold. First, it provides a deeper understanding of the inner workings of a convolutional neural network. Second, doing so allows for more flexible customization of neural network architectures as required in the future, beyond what is available through directly using generic implementations like those provided in the PyTorch library.

2 Related work

The machine learning algorithms that will be analyzed in this study have been around for a long time. KNN was first proposed in 1951 by Fix and Hodges [1]. The roots of softmax regression can be traced to the early 19th century [2]. The pioneering CNN publication is widely regarded to be Yann LeCun's 1998 paper [3]. Over the decades, each of these algorithms has been extended and improved countless times. In this project, I will analyze the performance of the basic versions of these algorithms on the image classification task.

Other similar studies on machine learning algorithms for image classification have been conducted in the past. In 2007, a well-cited survey on machine learning advances up to that point was published, which included neural networks and decision trees [4]. A comparison between the performance of SVMs and deep CNNs on the MNIST and COREL1000 datasets was also recently published [5]. However, one of the principal differences between this project and such previous work will be the inclusion of a custom CNN implementation in the analysis. Another difference is that this study is part of a course project, and hence is constrained to comparing three machine learning algorithms.

3 Methodology

3.1 Data

I have run my experiments on the standard CIFAR-10 dataset [6], which contains 60,000 32x32 RGB images belonging to 10 different classes. The distribution of images among the classes is balanced, with each having 6000. For such a large dataset, I have shuffled and split the training dataset of 50,000 images into two parts: one for training the model (49,500 images) and the other for validating the chosen hyper-parameters (500 images). The split is done such that each class label appears roughly the same amount of times in the validation set. Since the same random seed of 0 is set before shuffling, the exact same split of images is used for every ML algorithm considered. The creators of the dataset have also provided a separate test set of 10,000 images, which I will be using for obtaining an unbiased estimate of the final performance of each model. The data is normalized in two steps. First, min-max normalization is used to scale the pixel values from [0,255] to [0,1]. Second, channel-wise z-score normalization is applied so that each channel has mean 0 and standard deviation 1. This is done to improve the stability of training and hence to improve the performance of each model.

3.2 Evaluation

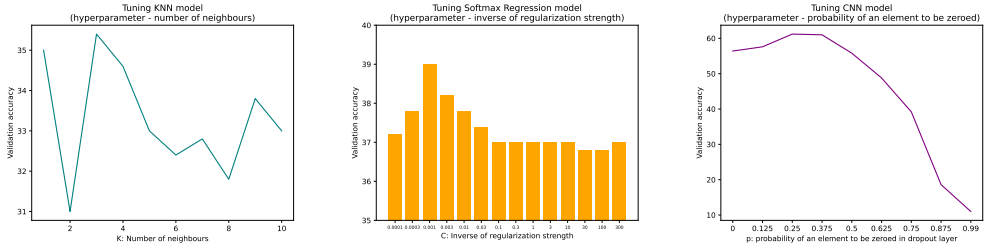
This task is multi-class classification, where each image has exactly one of ten possible labels. For this task, the trained classification model should be able to correctly predict the label of an unseen image. Therefore, I have opted to use classification accuracy to measure the performance of each model. This metric is closely aligned to the true goal of the task, which is to be able to predict the correct image of labels. Consequently, a model with a higher classification accuracy is preferred over a model with a lower accuracy. The formula used for calculating accuracy is as follows,

$$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions} \tag{1}$$

3.3 Algorithms studied

Here I provide a brief description of each algorithm considered and the hyperparameters that were tuned in each case. The sci-kit learn [7] implementations of KNN and Softmax regression (i.e. mulitnomial logistic regression) are used. CNN has two versions, one that is implemented in the PyTorch library and the other that I have implemented myself.

Figure 1 shows the validation accuracy obtained for different hyperparameter values for each algorithm during tuning. For each model, the hyperparameter judged to be most important was tuned. The other hyperparameters were set to their default values as provided in their implementations, and more details are provided in the following paragraphs. The figure does not include the custom CNN because the same hyperparameters as the PyTorch CNN were used to train this model.



(a) KNN (b) Softmax Regression (c) PyTorch CNN

Figure 1: Plots showing validation accuracy obtained for different hyperparameter values for each algorithm.

Trivial baseline The trivial baseline for comparisons is random guessing. Since the number of classes is 10, the expected accuracy of this baseline model is 10%. This model does not need to be trained, and the inference is simply implemented as a random choice between the 10 classes for each unseen image.

K-nearest neighbors (KNN) This algorithm assumes that data that is close together has the same label. During training, the model essentially learns every image in the dataset. During inference, each unseen image is assigned the majority label of its k nearest neighbors. The measure of closeness is the Euclidean distance between images when their 32x32 pixels are represented as 1-dimensional vectors. The hyperparameter to be tuned is k – the number of neighbors among which to take the majority vote. In my experiments, I have tried values from 1 (which is just the closest neighbor) to 10, at intervals of 1. The best value is empirically determined to be 3.

Softmax regression In softmax regression, also known as multinomial logistic regression, the idea is to estimate the probability $P[y = k|x]$ that an image x belongs to each class $k = 1, ..., 10$. The output of this model for each image is a vector of 10 probabilities that sum to 1. Cross-entropy loss is used during training. The solver used to train the model is the sci-kit learn implementation of SAGA, a variant of the stochastic average gradient algorithm. The training is limited to a maximum of 100 iterations. The hyperparameter to tune in this case is C, which is essentially the inverse of the regularization strength in the L2-penalized loss function. Higher values correspond to less regularization, i.e. a higher model capacity. This hyperparameter is tuned from $[0.0001, 0.0003, 0.001, ..., 300]$, and 0.001 is found to be the best choice.

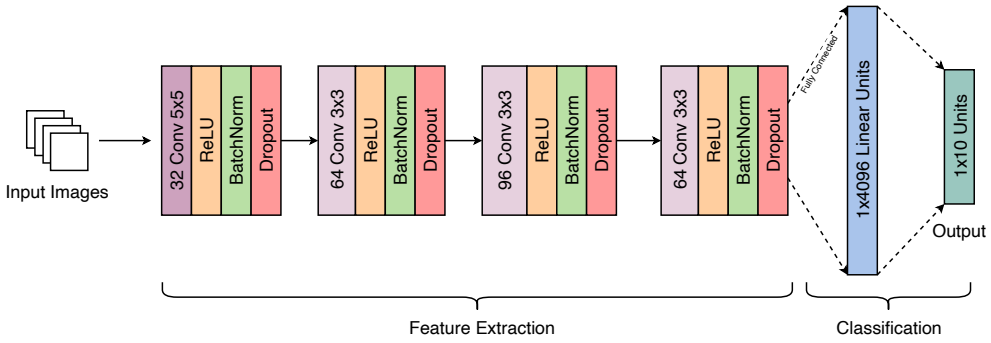


Figure 2: CNN network architecture

PyTorch convolutional neural network (CNN) Convolutional neural networks comprise neurons that have learnable weights and biases, and are well-suited for image data. The convolutions are usually done by image. The network architecture employed is shown in Figure 2. Convolutional layers are followed in order by ReLU activation, batch normalization, and dropout regularization. Each layer takes an input 3D volume and transforms it to an output 3D volume. The convolutional layers involve sliding multiple filters across the input. The step size of this sliding is referred to as the stride. The image may be padded to ensure the output of the convolution is of the required size. All convolutions use a stride of 1. The input to each is first padded with 0s, determined by the size of the filters. The first convolutional layer uses 32 5x5 convolution filters with padding two. The rest are 3x3, and have padding 1. The ReLU activation layers have no hyperparameters to tune – all values below 0 are simply clipped to 0. Batch normalization essentially normalizes the outputs of the activation layer, and scales and shifts them according to two learnable parameters. The dropout layer is used as a method of regularization to prevent overfitting. During training, each neuron is randomly blocked with probability p so as to allow the network to learn slowly and gradually. During inference, the dropout layer essentially becomes the identity function, with no neurons being dropped. These units comprise the feature extraction phase, which is followed by the classification phase. This phase

has two fully-connected layers, the first with 4096 units and the second with the 10 output units. The predicted class label is the one with the largest score.

To train the model, the Adam optimizer is used as provided in the PyTorch implementation, with a learning rate of 0.0001. The batch size is 64, and the model is trained for five epochs (i.e. five passes over the training set). These implementation choices are sufficient to reach a stable, low training loss value, as shown in Figure 3. The hyperparameter tuned here is p, the probability with which each neuron is dropped. This probability is tuned from different values between 0 and 1. Higher p values correspond to more regularization. The value that gives the highest validation accuracy is 0.25. With no regularization the performance is slightly worse. As the value approaches 1, the performance drastically deteriorates because the model capacity reduces drastically as well. At $p = 0.99$, the model performance is essentially equal to that of the trivial baseline.

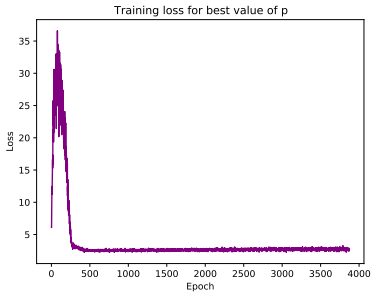


Figure 3: Training loss for the best PyTorch CNN model (p = 0.25).

Custom CNN I have further implemented both the forward and backward passes for commonly used layers in CNN architectures from scratch, using only basic NumPy and Torch functions. These layers are affine (fully-connected), convolutional, ReLU activation, batch normalization, and dropout regularization. To accomplish this, I have written code to define new PyTorch Autograd classes. These classes comprise a forward function and a backward function. Then, these classes are wrapped in PyTorch Module classes. The advantage of this is that the exact same training code can be used as the PyTorch-implemented CNN, except that the layers used inside the Sequential API are replaced by my custom layers.

The goal here is to achieve performance that is comparable to the PyTorch-implemented CNN on the same classification task. To ensure fair comparisons, the neural network architecture and the training hyperparameters are unchanged. Again, the architecture is as shown in Figure 2. The chosen dropout hyperparameter p is the best value as determined during validation of the PyTorch CNN, 0.25.

4 Results

Table 1: CIFAR-10 test set accuracy of each different trained ML model.

Model	Baseline	KNN	Softmax	PyTorch CNN	Custom CNN
Accuracy %	11.40	32.99	41.64	**61.79**	59.61

Table 1 shows the accuracy obtained for each algorithm's best model on the CIFAR-10 test set. The best models are the ones that are trained using the best hyperparameter value as determined using the validation set. The highest accuracy, 61.79%, is achieved by the PyTorch CNN. The performance achieved by the custom CNN, 59.61%, is almost equal to that achieved by the PyTorch CNN. The slight difference can be attributed to the random nature of training and initialization of network weights. One drawback of the custom CNN is that it is significantly slower than the PyTorch CNN. This is because my implementation uses nested for loops for the strided convolutions. The PyTorch

implementation, on the other hand, makes use of complicated tricks to avoid using for loops. However, since the primary goal of implementing the CNN layers from scratch for this project was to gain a clearer understanding of the inner workings and calculations of CNNs, I believe that the slower computation speed is a reasonable compromise.

Softmax regression has a linear decision surface, and thus it is not suited to non-linear tasks such as the image classification under consideration. This is a possible explanation for the lower classification accuracy obserevd – 41.64%. Apart from the lowest accuracy (32.99%) of all the three algorithms, another drawback of the KNN algorithm is that as the number of training images increases, the inference time does as well, since each unseen image has to be compared to more training images. This makes it impractical for use with the large CIFAR-10 dataset. Further, the storage size of the model is also very large for the CIFAR-10 dataset.

Nevertheless, all ML algorithms considered have outperformed the trivial baseline, which has an accuracy of 11.40% (slightly better than the expected value of 10% due to randomness). This shows that all three models are indeed able to learn some useful information (although to different degrees) about the CIFAR-10 data for the image classification task. The decisive winner here is the CNN algorithm, likely due to its suitability for image data, its higher model capacity, and its applicability to non-linear problems. Increasing the network complexity would probably improve performance further, provided that regularization is appropriately incorporated.

5 Conclusion

In this work, I have empirically analyzed the performance of three different ML algorithms, namely KNN, softmax regression, and CNN, for multi-class image classification on the CIFAR-10 dataset. Through my experiments, I have determined that the best accuracy on the test set is achieved by the CNN algorithm. These experiments involve tuning hyperparamaters for each algorithm to achieve the best performance possible using the validation set. I have also implemented CNN from scratch to satisfy the non-triviality component of the project. I have shown that my implementation achieves an accuracy similar to that of the PyTorch CNN implementation, albeit at a slower speed.

Future work There is plenty of scope to take this project further. For example, other classification algorithms can be considered, for example SVM or decision tree. Other possibilities include testing out more complex CNN architectures, for example those that use residual connections, or to incorporate more kinds of layers in the same architecture, such as pooling.

Acknowledgments

I would like to thank Dr. Lili Mou, Assistant Professor, Computing Science Department, University of Alberta, for facilitating this project and for all the knowledge disseminated during the CMPUT 466/566 Machine Learning course. I would also like to thank the course's teaching assistants for their hard work throughout the course.

References

[1] Evelyn Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989.

[2] J.S. Cramer. The Origins of Logistic Regression. Tinbergen Institute Discussion Papers 02-119/4, Tinbergen Institute, December 2002.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] D. Lu and Q. Weng. A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing*, 28(5):823–870, 2007.

[5] Pin Wang, En Fan, and Peng Wang. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recognition Letters*, 141:61–67, 2021.

[6] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[7] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.