# COMPARING CONVOLUTIONAL NEURAL NETWORKS AGAINST K-NEAREST NEIGHBOR AND BAYES CLASSIFIERS WITH CIFAR-10

**Connor Sweeney**
Department of Physics
University of Arizona
1118 E. Fourth Street, Tucson, AZ, 85721, USA
cosweeney@arizona.edu

**ECE 523 Term Project**

**ABSTRACT**

We compare how well a Convolutional Neural Network, K-Nearest Neighbor classifier and Bayes classifier classify images of the CIFAR10 dataset. Using the same training-testing split of the 60,000 32x32 color image dataset, we find accuracies of 78.87%, 33.58%, and 51.99% for each method, respectively. The underlying concepts for each method are introduced. Finally, we discuss our implementation for each of these classifiers and the design choices made for each, as well as how these contribute to the overall accuracy.

## 1 Introduction

Since their introduction in 1980, Convolutional Neural Networks (CNN) have proven to be a powerful architecture for classifying complex data, particularly image data. The structure of CNN's (see Figure 1.) allows them to learn particular spatial features in images with fewer parameters than a typical feed-forward neural network. Further, the design of CNN's are much better suited to image analysis than more simple supervised learning algorithms. The main goal of this paper is to demonstrate the effectiveness of this type of network against the performance of traditional algorithms in a multi-class classification task. With the primary application of CNN's in mind, we use the CIFAR-10 image dataset for such a test, and compare the classification accuracy with that of a K-nearest Neighbor (KNN) classifier and Bayes classifier (BC). We investigate how these classifiers differ in terms of where they succeed and fail classification, and discuss how these differences are related to the complexity of their implementation.

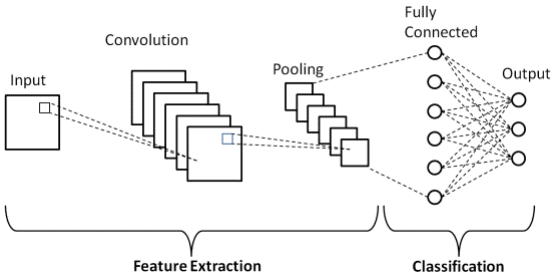


Figure 1: A diagram illustrating the general architecture of a Convolution Neural Network. Credit: *Phung Van Hiep and Rhee Eun Joo.*

## 2 Data

The CIFAR-10 dataset [1] consists of 60,000 32x32 color images of various animals and vehicles. Each image has an associated categorical label, which can be one of the following 10 labels: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. The large number of labelled, low-resolution images offers a sample of sufficient complexity for modern neural networks. As of May 2024, researchers have been able to use state-of-the-art architectures such as transformers to classify the dataset with upwards of 95% accuracy [2]. While such high performance is beyond the scope of this paper, we intend to demonstrate the performance one can still achieve on a typical workstation and with simple supervised learning methods.

In order to use the stated methods with this dataset, we applied some minor modifications to the raw data. This includes, for all methods, normalizing the image color scale so that pixel values in each color channel take values $\in [0, 1]$ instead of $\in [0, 255]$. This was done in order to simplify the image inputs as much as possible, and for numerical stability when computing e.g. loss gradients (see Section 3.1) or covariances (see Section 3.3). For the K-Nearest Neighbor and Bayes classifiers, 1-dimensional input vectors are required by the classifier designs, so we also augmented the data shape by flattening each 32x32x3 image to a 1-dimensional vector.

For training and testing each classification method, we make use of the same split of the dataset, dedicating 50,000 of the 60,000 images for training, and the remaining 10,000 for testing. The KNN required additional tests for hyperparameter determination (see Section 3.2), so for this method we make use of a cross-validation split as well. This involves splitting the 50,000 image training set into 5 subsets, and testing the classification accuracy when training using a portion of the subsets while varying hyperparameters. This can be used to choose hyperparameters in a robust manner without using the final test data.

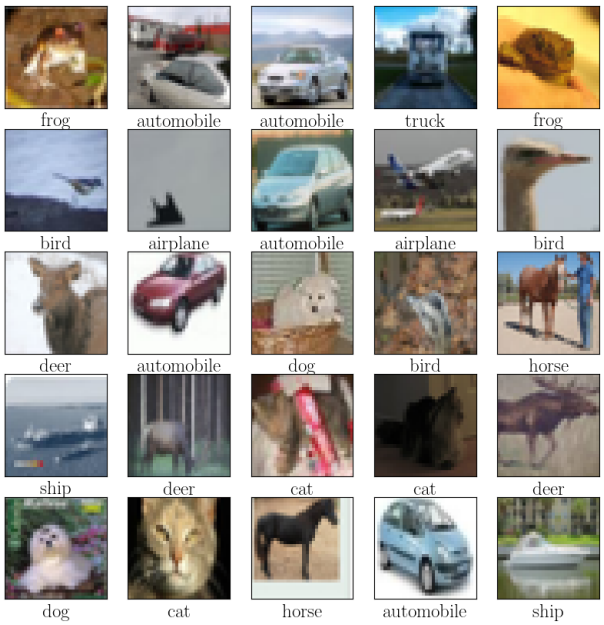


Figure 2: 25 examples of CIFAR-10 Images, and their associated labels.

## 3 Methods

### 3.1 Convolutional Neural Network

Figure 1 illustrates the basic structure of a general CNN. The network takes a 2d input (i.e. a color channel or channels of a specific image) and *convolves* it with a kernel filter. This results in a set of activation maps, which are then "pooled": the average or maximum values of a subset of the activation maps are collected to form a set of activation maps with reduced spatial dimensions. This sequence of convolution and pooling can be repeated as desired, until finally passing the activation maps to a fully connected network. Here, the features obtained by the previous layers are used to adjust the fully connected layers' weights using backpropagation and iterating, and finally the outputs are passed through a

softmax activation function to arrive at 1-dimensional output with the length equal to the number of classes. The index of the predicted class label will be given as a "1", while the rest will be "0".

We base the CNN for our classification task off of that found in [3], with some modifications. The full network diagram can be seen in Figure 3. The 32x32x3 image input is first sent through a batch normalization layer, which standardizes the input by transforming the feature distribution to have zero mean and unit variance. As with the image normalization, this process allows for more stable performance and learning of more dominant image features. This normalized input is fed to a series of two convolution layers with ReLu activation functions and 32 and 64 5x5 filters, respectively. The ReLu, or Rectified Linear unit, activations introduce non-linearity to the weights of the convolution layer and prevent vanishing gradients when performing backpropagation. The output of these layers is max-pooled using 2x2 pools with a stride of 2, and then we apply dropout with a rate of 25%: this disposes of that fraction of training weights in order to prevent overfitting. Then, the features are passed to another pair of convolution layers both with 64 3x3 filters. Dropout is applied once again, and a final convolution layer follows with 64 3x3 filters, followed by a final max-pool and dropout. Finally, the output is flattened and fed to a fully-connected layer with 10 units and a softmax activation layer in order to give the desired output vector with the class predictions. For training, we use a categorical cross-entropy loss function and the "Adam" optimization method. Using the aforementioned training/testing split split, we train the network for 50 epochs.

The updates to the network in [3] include the introduction of an initial batch normalization and the removal of an additional fully connected layer prior to our final layer. These modifications are made with better network performance and simplicity in mind. We find that this network yields comparable accuracy ($\sim$80% accuracy for [3]) for a dramatically lower amount of final training parameters (205,456 versus 2,267,274). While a thorough investigation into these choices is beyond the scope of this work, we find the performance of our simpler network sufficient to demonstrate the potential capability of CNN's over other methods.

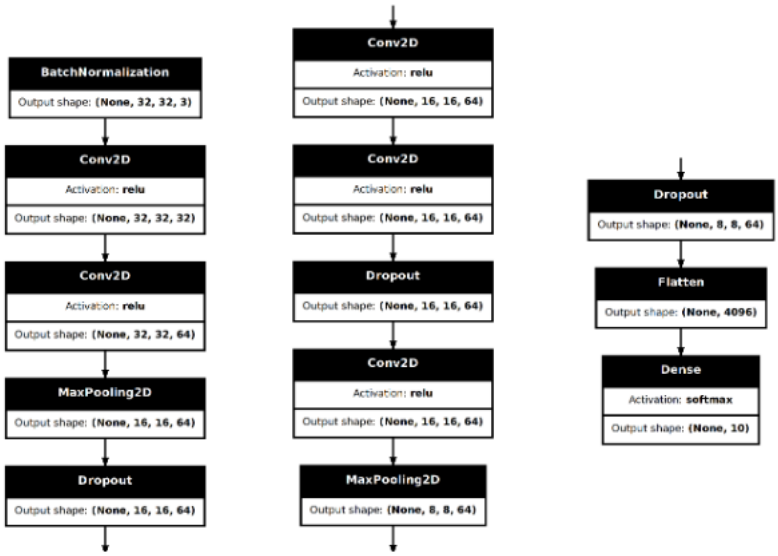


Figure 3: A diagram of the CNN used for our classification task. All layers are sequentially connected, with the first layer at the top left, and the final layer at the bottom right.

### 3.2 K-Nearest Neighbor

K-Nearest Neighbor is a non-parametric algorithm for classifying data based on the separation between data points in feature space. For a test data point of interest, a KNN classifier assigns the most represented class among $k$ of the closest training data points. It is typical to use the Euclidean or L2 distance to determine the separation in feature space (as we do for our classifier):

$$d_E^2(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} (p_i - q_i)^2 \tag{1}$$

The process for the KNN classifier can be summarized as follows: 1. Load in the training data. 2. Compute the Euclidean distances between testing points and training data points in feature space. 3. For each test data point, sort the

distances to the training data points from least to greatest and select the nearest $k$ neighboring data points. 4. Finally, assign the most common class among the $k$ neighbors to the test data point. It is not immediately obvious which value of $k$ one should use for a given classification problem; in order to choose, we follow the example of [4] and use the training data to cross-validate the accuracy of our KNN classifier for several proposed choices of the single hyperparameter. We choose that with the greatest validation accuracy while considering the impacts of overfitting. With this hyperparameter, we classify the final 10,000 image test dataset and report the accuracy to be compared with the competing methods.

### 3.3 Principal Component Analysis + Bayes Classifer

A Bayes Classifier is one which classifies a data point by choosing the most probable class for the given set of features. That is, the decision is rule is

$$\hat{y}(\mathbf{x}) = \text{argmax}_y P(y|\mathbf{x}) \tag{2}$$

Using Bayes theorem, the conditional probability $P(y|\mathbf{x})$ can be written

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \tag{3}$$

where $P(\mathbf{x}|y)$ is the likelihood of a data pointy given some class, $P(y)$ is the prior probability of the given class, and $P(x)$ is the probability of observing the data point (usually termed evidence). As the evidence does not depend on the class, the decision rule becomes

$$\hat{y}(\mathbf{x}) = \text{argmax}_y P(\mathbf{x}|y)P(y) \tag{4}$$

For the CIFAR-10 dataset, all classes are uniformly represented, so that all priors $P(y)$ are the same. A *Naive* Bayes Classifier would make the assumption that the features of the data are independent of one another; we take the position that this assumption is a poor one for an image dataset, where there are sure to be correlations among extended objects in the images and hence in the image features. Instead, motivated by the large number of images and the central limit theorem, we assume a multivariate Gaussian distribution for the likelihood.

With these assumptions, we use the training data to compute the sample mean and covariances of the likelihoods for each class, and then assign the class corresponding to equation 4 to the test data point of interest.

In order to increase the performance and memory usage of the BC, we perform Principal Component Analysis (PCA) on the image data before sending it to the classifier. The idea behind PCA is to decompose the data of interest into it's constituent eigenvectors, and retain only those that can account for most of the data covariance [5]. Given a data matrix $\mathbf{X}$, the data correlation matrix $\mathbf{X}\mathbf{X}^T$ can be expressed in terms of it's eigenvectors:

$$\mathbf{X}\mathbf{X}^T = \sum_{i=1}^{n} \lambda \phi_i \phi_i^T \tag{5}$$

by selecting the $m$ dominant eigenvectors such that $\mathbf{\Phi} = [\phi_i, ..., \phi_m]$, the data matrix can be approximated as

$$\hat{\mathbf{X}} = \mathbf{\Phi}\mathbf{\Phi}^\mathbf{T}\mathbf{X} \tag{6}$$

For this method, we select the $m$ eigenvectors which can account for 99% of the observed data covariance.

## 4 Results

Figure 4 shows the cross-validation accuracy of the KNN classifier. We show the accuracy for each $k$ for each of the 4 training-validation splits, as well as the final testing-validation split. While the overall highest cross-validation accuracy was observed for $k = 1$, we choose a value of $k = 7$ for our final classifier in order to avoid over-fitting.

Figure 5 shows the data eigenvalues and variance as a function of the number of retained PCA features. We find that for our training-testing split, 99% of the data variance can be captured with 231 ($\sim 7.5\%$) of the 3072 features.

Table 1 lists the classification accuracy of the final test split. Our CNN shows the highest accuracy, followed by the PCA+BC and then the KNN. We also show the confusion matrices for each classifier in Figure 6.

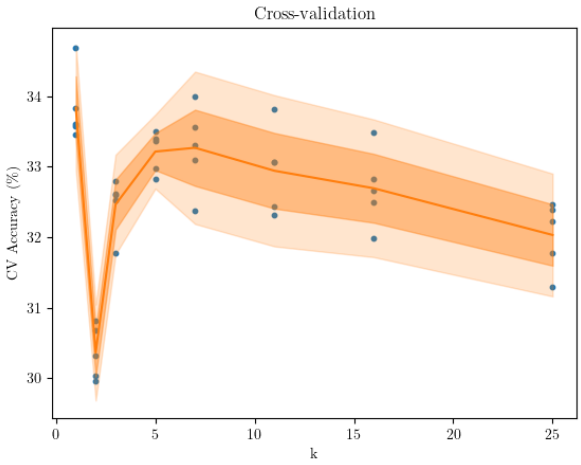


Figure 4: Cross-validation accuracy for the KNN classifier as a function of $k$. Points show the accuracy when evaluating on each sub-split of the training data, and the orange shows the final validation set along with 1 and 2-$\sigma$ error bands.

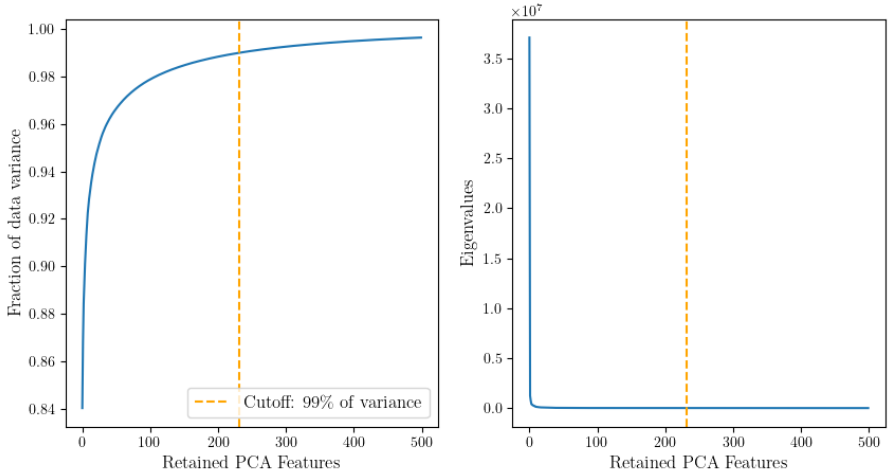


Figure 5: Cumulative data variance and eigenvalues as a function of retained PCA features.

## 5 Discussion and Conclusions

As we anticipated from the design of our classification task, the CNN classifier was able to classify the CIFAR-10 dataset with the highest accuracy compared to to the other methods we tested. The confusion matrix for the CNN does not exhibit any strong patterns for misclassified images, and from the plot in Figure 6 we can see that "automobile" images were classified most successfully while "dog" images were classified least successfully. Even with a relatively simple CNN design with only $\sim$200,000 parameters, this method is able to achieve a significantly higher (more than 25% greater) accuracy than the simpler methods. It would be interesting to repeat our classification task with a more

| Method | Accuracy (%) |
|---|---|
| CNN | 78.87 |
| PCA+BC | 51.99 |
| BC (no PCA) | 13.26 |
| KNN | 33.58 |

Table 1: Percentage of correctly classified test data for each method.

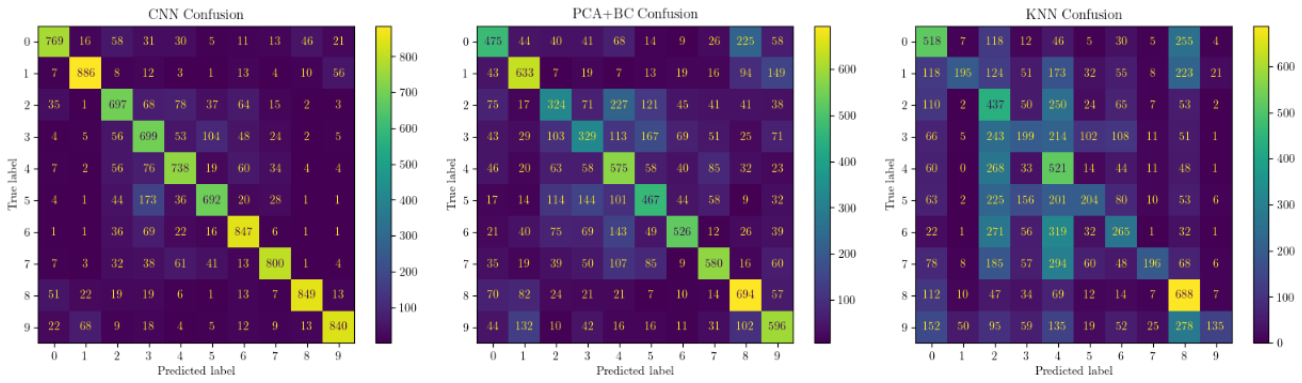


Figure 6: Confusion Matrices for each classifier, with true labels stated in rows and predicted labels in columns. The diagonals indicate the number of correctly classified images, and the rest of the cells indicate the number of mis-classifications. The label indices 0-9 correspond to the following labels: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', and 'truck'.

complex CNN, and to explore in more detail the relationship between the number of parameters and classification performance.

The BC using PCA showed the next highest accuracy. Suprisingly, the application of PCA to the image data before using the BC resulted in an improvement in accuracy of nearly 40%. We suspect that the distributions of each class amongst only the most dominant PCA features are more easily discriminated than in the full-feature space. It is likely that this method would yield an improved accuracy with a greater number of image samples, as in this case our application of the central limit theorem for the BC becomes more valid.

The KNN classifier achieved the lowest classification accuracy amongst the primary methods tested. The confusion matrix for this classifier reveals some interesting patterns: images are mis-classified as "birds" and "deer" more often than other classes, almost regardless of the true class. This may indicate that portions of the feature space are crowded by the distributions of these classes in such a way as to over represent them in predictions. In any case, this shows that a conceptually simple method which works well for data of a few classes may not work well with larger numbers of classes.

Considering the boost in performance for the BC when using PCA, and the possible difficulty of the KNN method to resolve different classes in the unaltered feature space, it would be interesting to repeat our comparison using PCA features for all methods. We anticipate that this would improve the overall accuracy, but we do not investigate this further here. We find that the observed boost in accuracy for the BC method when using PCA sufficient evidence for a brief demonstration of the utility of PCA and data pre-processing in general. We suggest [6] for more discussion of this topic.

Convolutional Neural Networks greatly outperform simple supervised learning methods for image classification in particular. The ability of CNN's to learn multidimensional features of images by convolving images with kernels unsurprisingly leads to lower classification error than methods which do not explicitly capture such information. It is still possible to achieve greater accuracy than random classification with methods such as K-Nearest Neighbor and Bayes classification, but not without greater modifications to image features or the sample size. For the relative ease of implementation, CNN's are a powerful method for tasks similar to that we perform here.

## Acknowledgments

Many thanks to Professor Abhijit Mahalanobis, Natnael Daba, and Daniel Brignac for an insightful and interesting class. We acknowledge the use python, matplotlib, numpy, and jupyter notebooks for implementing and testing the methods of interest. We used the Tensorflow/Keras library to implement our Convolutional Neural Network, and the sci-kit learn library to evaluate accuracy scores and display confusion matrices.

## References

[1] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[2] Kaggle. CIFAR-10 - object recognition in images. Available at `https://www.kaggle.com/c/cifar-10/leaderboard`. no date a.

[3] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[4] Fei-Fei Li. CS231n convolutional neural networks for visual recognition: Validation sets for hyperparameter tuning. CS231N convolutional neural networks for visual recognition. no date.

[5] David W. Hogg, Jo Bovy, and Dustin Lang. Data analysis recipes: Fitting a model to data, 2010.

[6] Enda Howley, Michael G Madden, Mary L O'Connell, and Alan G Ryder. The effect of principal component analysis on machine learning accuracy with high dimensional spectral data. In Ann Macintosh, Richard Ellis, and Tony Allen, editors, *Applications and Innovations in Intelligent Systems XIII. SGAI 2005*, London, 2006. Springer.