A Comparison of Classification Methods for CIFAR-10

Marissa Dominijanni Richard Simpson
Santosh Phani Vishnu Aita
University at Buffalo, Department of Computer Science and Engineering

1. Problem Description

Our goal with this work was to compare classical machine learning and computer vision techniques against those of deep learning. Until the popularization of deep neural networks, SVMs were considered the "gold standard" of machine learning, but as neural net development progressed, they dethroned SVMs on classification performance of MNIST, and their popularity only grew from there. CIFAR-10 is a much more "complex" dataset than MNIST, and as such, we wished to compare the performance of an SVM using SIFT features with the performance of a multilayer perceptron (MLP) using CIFAR-10 made greyscale, and of a convolutional neural network (CNN) on the full RGB CIFAR-10. We expect the CNN to outperform the MLP, which in turn should outperform the SVM.

2. Literature Review

Image classification is one of the important and complex processes in image processing. There are several image classification methods. The two main image classification methods are supervised classification and unsupervised classification.

2.1. Support Vector Machines (SVMs)

In pattern recognition, Image processing and machine learning, a feature vector is f-dimensional vector of numerical features that represent some object. The vector space associated with these vectors is the called the feature space. An SVM is a supervised binary machine learning algorithm. If we have a dataset of feature vectors that belong to one of two classes, and also their corresponding binary class label, then we can train an SVM to classify unlabeled examples of these feature vectors.

An SVM is about cutting the space in two parts, between two families of points. Nonlinear will create a space transformation, it will be a linear SVM in the feature space, but a non-linear separation border in the input space.

2.2. Deep Learning

Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised. With deep neural networks, the more hidden layers there are, the "deeper" the model is said to be.

2.3. Previous Work

There are many models using which the image classification was done previously on the CIFAR-10 data set. Each of them required different environment settings and parameters such as no. of layers, technique involved, preprocessing methodologies, training methods etc.

Different methods used in previous research, such as Fractional-Max-pooling, Manual classification by human, Deep residual learning, Stacked auto encoders and Multi-Column Deep Neural Networks resulted in different accuracy on the testing data.

2.4. Parameters

As an input, we have image, *w* pixels wide and *h* pixels in height. Such image has a class. You can think about it as a label which is attached to an image–so whether it presents a car, a bird and so on. In real world there can be situations where an image depicts both car and a bird sitting on it–but this solution will ignore this fact and classify the image to one class only. The real class of the image will be named. In all the neural network models, the following parameters remain the same.

2.5. Image Representation

In the case of this solution image will be represented as a matrix:

$$D^{wh\times 3}$$

It is because the knowledge about the exact width and height is unnecessary, so the wand h dimensions are flattened to avoid operating three-dimensional entities.

2.6. The Classifier Function

The goal is to create a classifier–a function c which takes such matrix as an input and attaches a class to it (which is unknown):

$$c(D) = \hat{y}$$

2.7. Training Set Definition

In machine learning we need a set to learn from–it is a part of creating the classifier. It is called a *training set* and it'll be noted as:

$$T = \{X_{(i)}^{wh\times 3}, y_{(i)}\}$$

The training set contains n data samples and subscripted (i) is a i^{th} data sample from this set. It is a set, so the ordering of samples is irrelevant. Although when using SGD (either fully stochastic or minibatch), random ordering when presenting training examples over different epochs is essential.

2.8. Result Comparison on Various Models

The neural network model is the heart of the classifier. This varies from problem to problem. Previous work on image classification using CIFAR-10 Data set resulted in the following accuracies:

Method	Accuracy
Manual Classification	94.0%
Deep Residual Learning	93.57%
Convolutional Kernel Networks	82.13%
Fractional Max-Pooling	96.53%
Practical Bayesian Optimization	90.5%
ReNet	87.65%

3. Feature Extraction

3.1. Features

A Feature in Image Processing and Machine Learning is a piece of information in Images which is relevant while dealing with certain problems. Features may be specific structures in the images such as edges, blobs, corners or objects. The choice of features in computer vision varies from problem to problem.

The procedure involving the detection and representing the features in Image Processing is called Feature Extraction. The resulting features are often represented in terms sets of coordinates of image points where the features have been detected. There are many feature extraction methods/techniques such as Harris Corner Detection, Blob Detection, etc. For more complex problems extracting one feature may not be sufficient, instead two or more features are extracted resulting in different feature descriptors at a point in an image.

3.2. Scale Invariant Feature Transform

Scale Invariant Feature Transform generally known as SIFT is an algorithm in computer vision to detect and describe local features in Images. It is generally used in problems such as object recognition, image classification, image stitching, video tracking and gesture recognition. The detection and description of features assist in object recognition and image classification. The SIFT is chosen as the feature extraction methodology in our model because of the following advantages. The SIFT features are invariant to image rotation and scale. They are robust to changes in illumination, noise, and minor changes in viewpoint. These features are relatively easy to extract compared to other feature extraction methodologies.

We know that Harris detector is rotation invariant. However, it is not the case with if the image is scaled. A corner in the base image may not be same as the corner in the scaled image.

For example, consider the image in directly below, the object in the left as a part of the base image (un-scaled). The right object is a part of the same image but when it is scaled. We can clearly see that it is not the same scenario when image is scaled. Hence, we can clearly say that we can't use the same window to detect key points in scaled images. So SIFT comes into picture.

3.3. Feature Extraction with SIFT

The data set in our model is CIFAR-10. It consists of 60,000 (32×32 RGB) images. The set is split, with 50,000 images being designated for training in five separate batches and 10,000 images being designated for testing. CIFAR-10 is a ten-class dataset (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). There are 6,000 images of each class (5,000 training, 1,000 testing).

The feature extraction on CIFAR-10 training data set is done using SIFT. Initially the CIFAR-10 data set was obtained for python version. Another version for Matlab is also available. SIFT generates descriptors for each keypoint not with raw intensity values, but by computing the gradient of each pixel (For a pixel its gradient describes how intensity changes as X and Y change). Then it bins the gradients obtained into 8 directions namely N, NE, E, SE, S, SW, W, NW and creates an 8-bin histogram.

Divide the feature region into 16 square sub regions. Bin each region into an 8-direction histogram and then all the

computed 16 histograms are concatenated together to obtain the final 128-bit SIFT descriptor.

Initially, all the five batches of training images each having ten thousand images were loaded and clubbed together as a matrix having consisting of all the 50000 training images. The images obtained were in the form of a single 50000×3072 matrix. So each image was provided in one row of the matrix. Each row represents and image. The first 1024 values represented R layer values and the next 2048 values represented B and G layer values. So the matrix was reshaped to obtain each image and the features from each image in the training dataset were obtained using SIFT functionality available in OpenCV. SIFT's detectAndCompute() method detects the features in the image and returns the key points and the SIFT descriptors. The descriptors of all the images were added to stacked together.

The desired number of clusters were chosen and Now K-Means clustering was done on using the obtained SIFT Descriptors resulting in the centroids of the clusters. The next step is vector quantization, Vector Quantization basically assign codes from cookbook (centroids) to observations (descriptors) and visual words are obtained. Then finally the histogram of feature is computed using the visual words.

The features obtained in this system are passed to SVM training model.

4. Support Vector Machines

4.1. Theory

Support Vector Machines (SVMs) were a popular way of solving classification problems before neural networks gained popularity. SVMs perform classification by fitting a hyperplane into the large dimensional input space. When the SVM is asked to predict which class new data is a part of, it calculates on which "side" of this hyperplane the data is on. SVMs were traditionally a binary classification algorithm, but two approaches to multi-class classification were developed. These two approaches are called "one-versus-one" (ovo) and "one-versus-rest" (ovr). The mutliclass paradigm used here was the ovr paradigm. The ovr paradigm works by a winner-takes-all strategy, where the classifier with the highest output function assigns its class.

The SVM implementation scikit-learn uses solves the classification problem in the following manner. We start by defining the primal problem: Let $(x_1, y_1), \ldots, (x_n, y_n)$ be our training data set with n samples, where x_i is an m-dimensional real vector, and y_i indicates which class x_i belongs to. We want to find the "maximum-margin hyperplane" that divides the group of points for which $y_i = 1$ and $y_i = -1$. A "hinge-loss" function can be used to describe the fitting of the hyperplane, and is given by the following equation:

$$\max(0, 1 - y_i(w \cdot x_i + b))$$

Then the SVM problem becomes the problem of minimizing:

$$\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(w \cdot x_i + b)) + \lambda||w||^2 \qquad (1)$$

The above formulation of an SVM is for a linear SVM. The SVM used here used a nonlinear kernel, which works by replacing the dot products with the nonlinear kernel. We used a radial basis kernel, which can be described by:

$$k(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$$

for $\gamma > 0$. The scikit-learn implementation of SVMs then writes this minimizing problem as a constrained optimization problem, and using the Lagrangian dual of this problem it solves it using quadratic programming.

4.2. Results

The following parameters were used to calculate SIFT features:

Parameter Name	Value
nOctaveLayers	10
contrastThreshold	0.01
edgeThreshold	20
sigma	0.8
nfeatures	50
nclusters	50

Table 1. Summary of Parameters

These parameters gave an accuracy score of 27.3% with the following confusion matrix:

$$\begin{bmatrix} 405 & 65 & 115 & 38 & 47 & 34 & 47 & 57 & 149 & 43 \\ 72 & 306 & 38 & 78 & 38 & 79 & 88 & 63 & 119 & 119 \\ 138 & 62 & 213 & 105 & 99 & 90 & 134 & 65 & 38 & 56 \\ 56 & 64 & 82 & 222 & 52 & 203 & 127 & 81 & 34 & 79 \\ 69 & 79 & 127 & 80 & 184 & 94 & 147 & 96 & 56 & 68 \\ 29 & 71 & 93 & 171 & 40 & 245 & 145 & 80 & 43 & 83 \\ 47 & 76 & 103 & 117 & 73 & 115 & 329 & 53 & 26 & 61 \\ 53 & 101 & 71 & 138 & 65 & 126 & 101 & 212 & 41 & 92 \\ 124 & 131 & 47 & 63 & 33 & 32 & 50 & 42 & 371 & 77 \\ 68 & 138 & 38 & 103 & 51 & 91 & 81 & 78 & 109 & 243 \end{bmatrix}$$

Finally we have the classification report:

	precision	recall	f1-score	support
0.0	0.38	0.41	0.39	1000
1.0	0.28	0.31	0.29	1000
2.0	0.23	0.21	0.22	1000
3.0	0.20	0.22	0.21	1000
4.0	0.27	0.18	0.22	1000
5.0	0.22	0.24	0.23	1000
6.0	0.26	0.33	0.29	1000
7.0	0.26	0.21	0.23	1000
8.0	0.38	0.37	0.37	1000
9.0	0.26	0.24	0.25	1000
avg / total	0.27	0.27	0.27	10000

Table 2. Classification Report

5. Multi-Layer Perceptron

5.1. Principles

MLPs are widely considered the most "primitive" of the deep learning methods (in that they came first and are the simplist method), and extend the single-layer perceptron (SLP) by adding one or more "hidden layers". A perceptron is organized into an input layer, with a number of inputs equal to the dimensionality of the feature space (the number of features), an output layer with a number of outputs equal to the dimensionality of the hypothesis space (the number of classes), and optionally, one or more hidden layers of any size which are connected in a feedforward manner. That is, two adjacent layers form a complete bipartite digraph, where the the directionality of each edge goes from the set of neurons of the prior layer to the set of neurons of the next layer. The MLP architecture forms a universal function approximator.

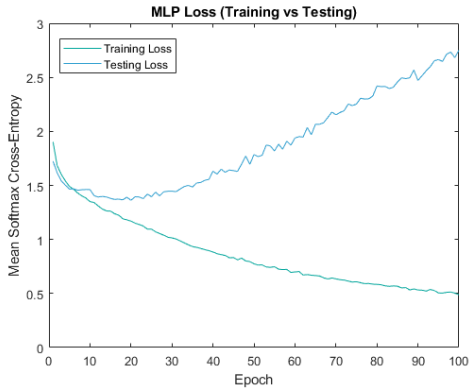
5.2. Architecture and Methods

For our test case, we decided to use a relatively conventional MLP setup in deep learning. We used an MLP with five total layers (three hidden) with sizes 1024 (the number of greyscale pixels), 512, 256, 128, and 10 (the number of classes). We used ReLU for our activation function. And for optimization, we decided to use Adam (Adaptive Moment Estimation) with $\alpha = 0.002, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1 \times 10^{-8}$. We implemented this architecture using TensorFlow 1.4 with the Layers API. We trained on a total of 100 epochs using a minibatch size of 200. We evaluated the test set every epoch, and ran a detailed analysis of the performance on the model with the best test accuracy. For our loss function, we used mean softmax cross entropy (so the average cross entropy on the softmax of the logits and the onehot representations of the true labels).

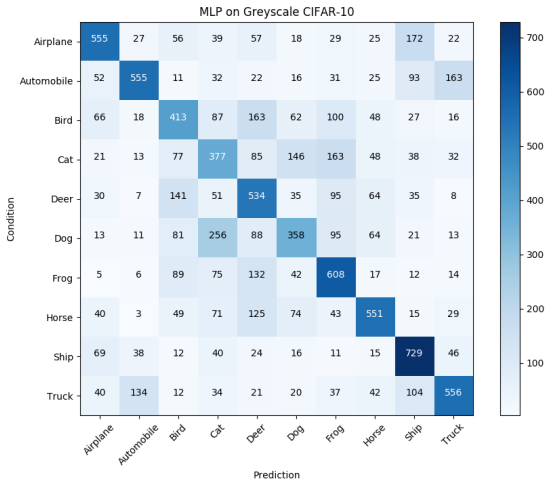
5.3. Results

Although the overall accuracy was somewhat disappointing with the MLP, it is still vastly better than the accuracy using the SVM. Overall, we were able to achieve a maximum test accuracy of 52.36% on the 27^{th} training epoch. Note that in the following examples, training loss and accuracy (or error) are taken as the average over each batch in that epoch.

From the above figure, we can see that while the training error continues to drop off after the 27^{th} training epoch, the testing error very slowly climbs. The problem of overfitting is very well managed by Adam, but nevertheless, does still occur. In a realistic classification scenario, this would be detected by using an early stopping algorithm as a form of regularization, but we wished to show the performance over a fixed number of epochs.



Here we can even more clearly see the point at which overfitting occurs. The maximum test accuracy occurs just after we begin to see overfitting by loss, and as loss continues to increase rather dramatically, accuracy continues to slowly deteriorate.



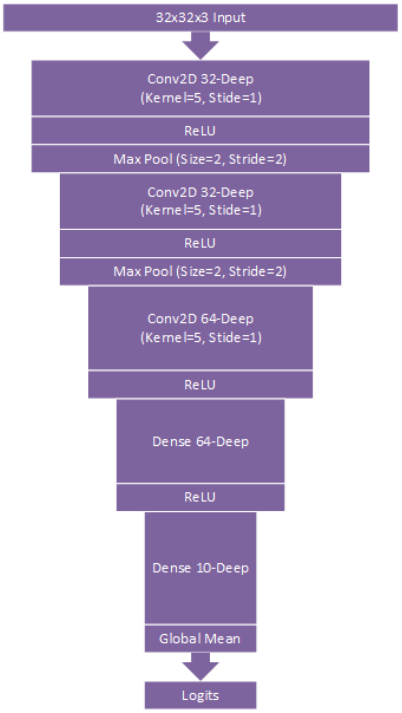
And here we can see the confusion matrix at the 27^{th} epoch on the testing set. The results from it generally make sense, but one point of interest to note is that although dogs were often misclassified as cats (256 out of 1000 examples) which does make a degree of sense intuitively, but cats were misclassified as dogs far less often (146 out of 1000 examples) and more likely to be misclassified as frogs (163 out of 1000 examples). It would be interesting to see if when color information is included in this (as in the CNN model), if this rate of misclassification decreases more sharply than average.

6. Convolutional Neural Network

6.1. Principles

CNNs were up until very recently the standard method used for image classification (although the very recent development of capsule networks (CapsNet) will almost certainly render CNNs obsolete). For our purposes, we stuck to a relatively straightforward CNN architecture made up of convolutional layers, max pooling layers, ReLU layers, dense layers, and global average pooling. The intuition behind the fundamental component of CNNs, the convolution operation, represents a sparse interaction between layers as opposed to one in which every point in one layer can affect every point in the next layer. This is because, unlike with an MLP where we assume all of the variables may have relevant mutual interactions, in a CNN, we assume that interactions are confined to some given neighborhood. Max pooling makes the network invariant to small translations on a local level, but on a global level, makes it so that the specific location of features doesn't matter. This is desirable in certain situations (such as identifying a dog regardless of its location in an image) but less desirable than others such as facial recognition, which can lead to rather comical results on Picasso-esque adversarial examples (this is the specific issue CapsNet seeks to resolve). We then also use dense layers (also called completely or fully connected layers), which are just matrix multiplication, like the layers in MLPs. We can use this to compute the logits with the same dimensionality as the hypothesis space. The final operation we need to consider is global average pooling. This simply involves reducing the dimensionalty of the tensor from the dense layer from $[B, H, W, C]$ to $[B, 1, 1, C]$ by averaging over the plane representing the surface of the layer (reduced from the surface of the image itself).

6.2. Architecture and Methods

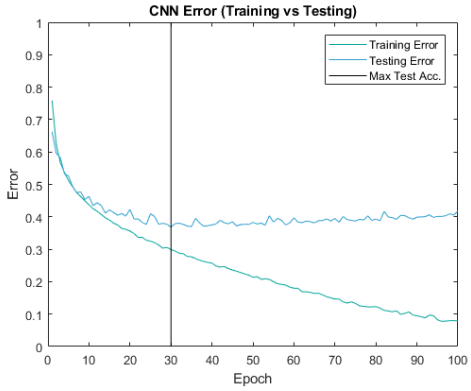


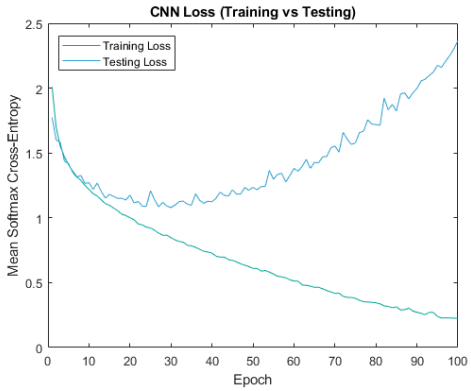
Unlike with the MLP, it's much easier to graphically describe the CNN architecture we used (as can be seen in the above diagram). We additionally used the Adam optimizer with the same hyperparameters as before, and we used the same loss function (mean softmax cross entropy) and used the same batch size of 200, additionally over 100 training epochs (a total of 25,000 batches).

6.3. Results

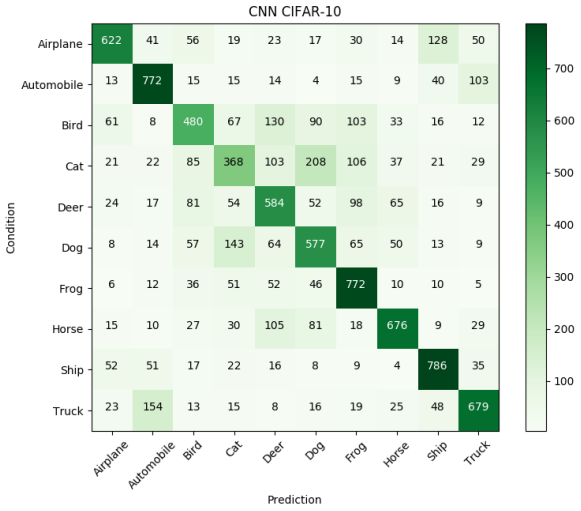
Overall, while we were able to achieve significantly better results with the CNN than with either the MLP or SVM, we did not achieve state of the art results. In order to do this, we would have most likely been required to use a much more complex pretrained model such as VGG-16. As we can see in the two plots below, the CNN achieves a significantly higher test accuracy before generalization error overtakes improvements in training error (63.16%), and that this occurs at around the same epoch as in the MLP (30^{th} for the

CNN versus 27^{th} for the MLP). It is also interesting to note that the actual shapes of the learning curves are very similar, but with the CNN hitting overall lower error and loss.





Additionally, as with the MLP, we want to examine the confusion matrix for at the best performing epoch (the 30^{th}, as previously stated). Generally speaking, this confusion matrix looks significantly better than the one produced by the MLP. And if we look at the example previously mentioned, cats are less often misclassified as frogs, but are instead more frequently misclassified as dogs (although the rate of misclassifying dogs as cats has dropped significantly).



7. Conclusions Future Work

From this analysis of these classification methods, we can very clearly see the deep learning methods strongly overtaking the classical methods. Below is a table of the maximum test accuray achieved with all three methods for comparison:

	SVM	MLP	CNN
Test Accuracy	0.2730	0.5236	0.6316

Additionally, we can see how the CNN, which is specialized to spatial grid information (such as images or board games like Go) overtake the MLP. In the future, this could be extended to using more advanced classification methods such as using a DCGAN (deep convolutional generative adversarial network) to augment the number of examples. Or, in general, using other forms of data augmentation to improve performance (such as applying different affine transformations to the training set) and by adding random Gaussian noise to the training examples as a form of adversarial training. And as suggested in the section on CNNs, attempting to use CapsNet from CIFAR-10 classification would be an interesting challenge, especially considering the relative newness of this architecture.

References

[1] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130.

[2] https://medium.com/@Killavus/naive-approaches-to-solving-cifar10-/dataset-image-classification-problem-/overview-and-results-636048ff1cc1.

[3] http://www.nvidia.com/content/events/geoInt2015/LBrown_DL_Image_ClassificationGEOINT.pdf.

[4] https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[5] http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html.

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, sep 1995.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.