December 22, 2019

Comparing the efficiency of ConvNet with Naïve Bayes and SVM classifiers

Joss Rakotobe
Project in IFT6390
Université de Montréal
Montréal, QC, Canada
rakotobe@dms.umontreal.ca

Abstract

The goal is to compare the performance and characteristic of three classifiers in different situations. I used two datasets, CIFAR-10 [1] and IMDB reviews [2], to make the comparison. The research results reveal that the convolutional network is, as expected, better in accuracy than the other two algorithms, but at what cost.

Introduction

Motivation

Neural network algorithms are very complex algorithms, that can be very powerful. The objective of this report is to test the convolutional neural network classifier (ConvNet), against the Naïve Bayes classifier (NB) and Support Vector Machine classifier (SVM), on different kind of data.

For these experiments, since I am very interested in image recognition, I first decided to use the CIFAR-10 dataset. I also read that ConvNet truly outperforms "standard" classifiers on images and so I was curious to witness that.

My second goal was to perform a text classification, which is why I picked IMDB reviews. After trying for so long to beat the Naïve Bayes classifier on the Kaggle competition, and failing to do so, I wanted to see if a ConvNet could have been a solution.

Datasets

CIFAR-10[1] is a dataset of 60 000 32 by 32 colour images equally distributed in 10 classes. The classes are mutually exclusive, meaning that each image belongs to exactly one class.

IMDB reviews[2] is a dataset of 50 000 reviews of movies on IMDB database. It only contains reviews that have a high rate (7/10 and greater) or a low rate (4/10 and lower) and were accordingly labeled as positive and negative review. Even though we have access to the actual rating of each review, I decided not to use it.

[1]https://www.cs.toronto.edu/ kriz/cifar.html
[2]http://ai.stanford.edu/ amaas/data/sentiment/

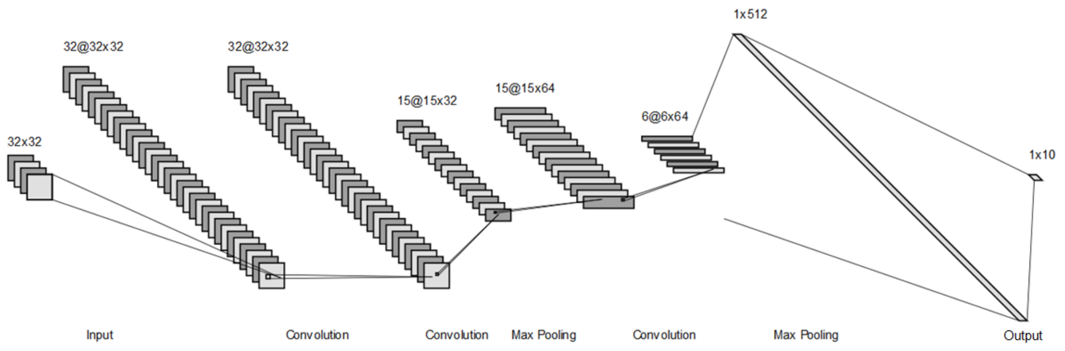
1 Experiments on images : CIFAR-10

1.1 ConvNet

Each example of the CIFAR-10 dataset comes in a one dimensional vector of size 3072 ($32 \times 32 \times 3$). In order to fully grasp the connectivity of the images, I use a 2D convolutional network. Therefore, the input need to be in a more conventional image shape: a three dimensional vector. I decided to use the RGB format since it is the default one used in the libraries I used.

I based my ConvNet architecture on the Keras example on CIFAR-10[3], which is using data augmentation (translations and horizontal reflections), and tuned some of the hyperparameters : I tried to add/remove some layers, the number and size of the filters in the convolutional layers. I could not try a lot of combinations of the hyperparameters due to the fact that each attempt needs a lot of time to compute.
Ultimately, I got this architecture[4]:

Architecture of the ConvNet



It turns out that I managed to reach a slightly better accuracy than the baseline given in Keras: 80% on the test set after 50 epochs.

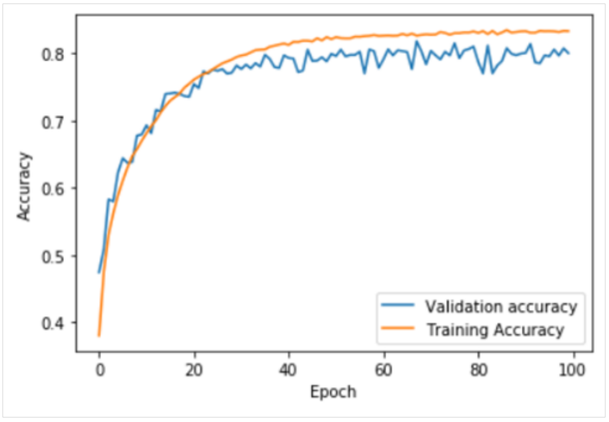


Figure 1: ConvNet learning curves on CIFAR

Despite its complexity, ConvNet happens to not overfit the model, even after a hundred epochs, and give a very good accuracy. Paper results are even showing that one can reach an accuracy of 99% [5].

[3]`https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py`
[4]This image was made using `http://alexlenail.me/NN-SVG/LeNet.html`
[5]`https://en.wikipedia.org/wiki/CIFAR-10`

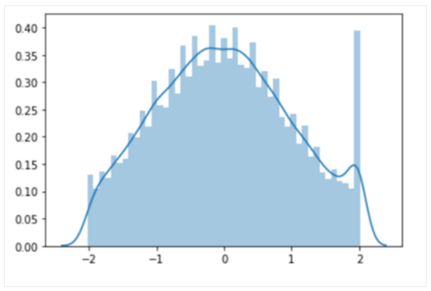
1.2 NB and SVM

For those algorithms, I already know that they only take one dimensional vector as input. Thus, I kept the vectors as such. This is already telling us that they won't capture the connectivity between the pixels, but will look at each one of them independently.

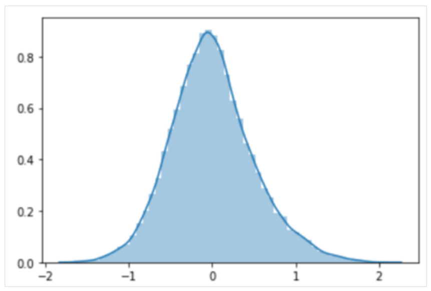
We can expect the classifier to only notice the colors, and not the frame of the objects/animals.

Starting with NB, my first task was to find in what distribution the data falls into. Choosing a pixel randomly and plotting its distribution (after centering it), we get the plot on Figure 2a. As we can see, they seem to follow a distribution close to a Gaussian. Thus, I decided to use a Gaussian kernel and got an accuracy of 29.7% on the test set.

Afterward, I tried using PCA to reduce the number of features. The distribution of the new features got a little better : less noises, and much closer to a Gaussian distribution (see Figure 2b).



(a) A pixel randomly chosen



(b) A randomly chosen component after PCA

Figure 2: Distribution of the features

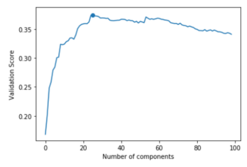


Figure 3: NB Validation curve

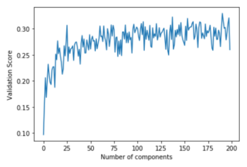


Figure 4: SVM Validation curve

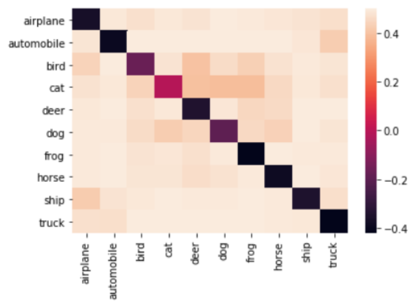
The new results confirmed that the new features were in fact better for this classifier. I selected the number of components using a validation curve (see Figure 6b) and got my best accuracy on the validation set with 25 components: 37% on the validation set and test set.

Using the same technique with a linear SVM classifier, I obtained an accuracy of 27% without PCA, and 33% on the validation set, 30% on the test set with PCA. Looking at its validation curve 4, we can see that the method is much less stable than with NB. My next attempt is to use a kernel on the SVM too. However, it needs a lot of time to compute, using the scikit-learn library.

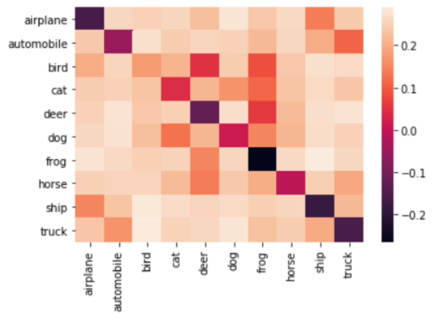
1.3 Comparison on CIFAR

Although the results on NB and SVM are no match compared to the ConvNet's, considering there are 10 classes, these simple models are still doing a lot better than a random classifier or a majority class classifier (which would have an accuracy around 10%).

In figures 5, we plotted the confusion matrices of the ConvNet and NB models in the form of heatmaps. As expected, both models are confusing classes that look alike : between animals, bird and airplane, automobile and truck, etc.
However, on the NB model, since it can only grasp the pixel's color and not the connectivity between them, we get more classes confused when the colors are similar. Since the model cannot distinguish the main object/animal from its environment, the colors of the environment will have more effect, which can explain the confusions between some classes that do not have the same color (ex: deer and frog). It can also explain why the bird's class has the worse accuracy and frog's class has the best one. Birds come in a wide variety of color, whereas frogs are mostly in shades of green, which is also not common in the other classes.



(a) Confusion matrix of ConvNet



(b) Confusion matrix of NB

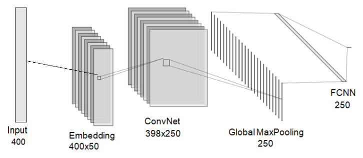
Figure 5: Confusion matrices

2 Experiments on text classification : IMDB reviews

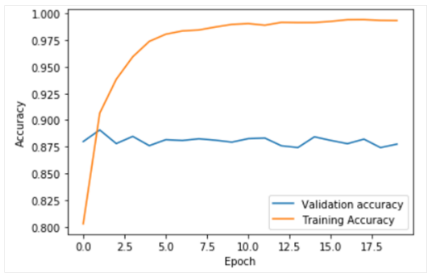
2.1 ConvNet

For the IMDB reviews, I entirely used the architecture on Keras[6]. After trying to tune some of its hyper-parameters (the number of filters, the kernel size), it did not improve the result.
In order to not lose the semantic, the vectorization of the text was divided in three steps. First, we need to build a vocabulary of all the words in the corpus. Second, we need to determine the words to keep assign each word to a number. Last, we need to determine the number of word to use in our vector representation and for each review, we create a vector by converting each word that are in our dictionary to its assigned number, keeping the order of those words.
The architecture of the ConvNet is based on a 1D convolutional network, which can learn features that grasp the semantic of the reviews (see figure 6a).

After only three epochs, I realised that the model was already starting to overfit, hence I used two epochs and got an accuracy of 89% on the test set.



(a) Architecture of the ConvNet on IMDB Reviews



(b) ConvNet learning curves on IMDB reviews

The article of *Learned in Translation: Contextualized Word Vectors*[7], by B. McCann, J. Bradbury, C. Xiong and R. Socher gave a summary on some of the best results on this classification problem, with more than 94% accuracy.

2.2 NB and SVM

For NB and SVM, I used a bag of word representation. I've selected all the words having frequency higher than 4, and made a word-frequency vector for each review.

Assuming the probability of a word occurring is independent of the other word, the distribution is multinomial.
I used a multinomial NB with smoothing and a linear SVM. I tuned the smoothing hyper-parameter alpha for NB (see figure 7), and for both of them, I got an accuracy of 83% on the test set.

[6]`https://keras.io/examples/imdb_cnn/`
[7]`https://arxiv.org/pdf/1708.00107.pdf`

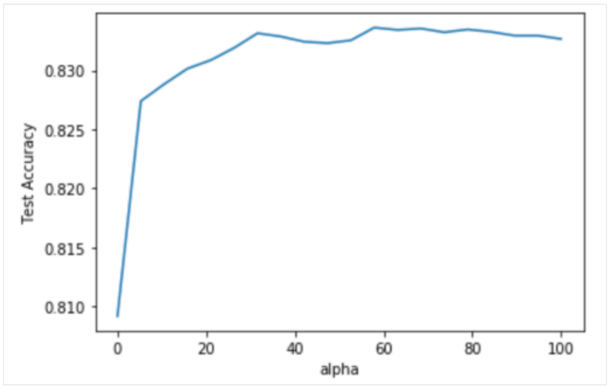


Figure 7: Training accuracy of NB against its smoothing parameter alpha

2.3 Comparison on IMDB

The results are quite satisfying. NB and SVM are good text classifiers, and easy to implement. Here, we did not bother to find a very good set of features (n-grams, Tf-Idf, chi square, etc.), and still got a good accuracy.
In the end, ConvNet performed best than the other two on this text classification problem. It was to be expected, since it can take into account the semantic of the word, hence it does not need to have a good set of features, it will learn it himself. However, the challenge for such classifier is to come up with the architecture. This can take a lot of time to come up with it.

3 Conclusion

To conclude, ConvNet has exceeded my expectations. It is a powerful learning algorithm that can grasp different types of connectivity without overfitting our data. I believe that those results can be generalized to other text or image classification problems. The only issue is to find the right architecture for the problem. This was not done by myself, but I am confident that it requires a lot of work. When time is in the essence, this method might not be the one to try first ! The other two algorithms are very good on text classification. I did not do a lot of preprocessing and yet the accuracy was good. However, on images they need more preprocessing to give more decent results. Since I did a gender classification using profile pictures on another project, I can confirm that given good features extracted from images, NB and SVM can give really good results on an image classification.

Table 1: Accuracy on the test set

	ConvNet	NB	SVM
CIFAR	80%	37%	30%
IMDB	89%	83%	83%

Acknowledgments

As the unique author of this project, I hereby state that all the work presented in this report is mine.

References

[1] A. Krizhevsky, “Learning multiple layers of features from tiny images,” tech. rep., 2009.

[2] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 142–150, Association for Computational Linguistics, June 2011.