Comparative Analysis of Convolutional Neural Network Architectures

Comp 4730: Machine Learning

Oct 16, 2022

University of Windsor

Ali Naqvi, Noah Adams, Sergio Oliveira

Abstract

With advancements in machine learning techniques growing for image classification, the MNIST dataset serves as an excellent basis for evaluating the effectiveness of architectures. Particularly, convolutional neural networks (CNN) have proven to be extremely effective in classifying the images in the MNIST dataset, with many boasting 98% accuracy or greater. We explore one such CNN architecture for classifying these images and determine its effectiveness compared with that of the most accurate architectures. In this paper, we developed a CNN architecture consisting of a few convolutional layers with ReLU activation and maximum pooling. The neural network follows the set of convolutional layers with a fully connected layer. With a series of preprocessing steps and the neural network, we were able to achieve an accuracy of 99.45% for our model.

Introduction

Image classification in the field of machine learning has been for many researchers an area of study in the ways of improving the accuracy and percentage error of architectural models. The basis for many image classification datasets is the MNIST handwritten digit recognition dataset. This dataset uses 60,000 images for handwritten numbers 0 through 9, so learning models can predict with some accuracy what number is in an image [12]. Many models have been able to achieve high results, but CNNs in particular have performed the best with accuracy scores boasting 98% and with careful adjustments of hyperparameters, CNNs have been shown to improve to 99% or higher [10][11][12].

Convolutional networks themselves rely on a set of convolutional layers that essentially help define features within a dataset [1][6]. The convolutional layers cross-correlate and later convolute the dataset and network using filters or kernels that act to create outputs that provide features of the dataset. In many cases, an activation function follows the convolution to increase non-linearity, such as a rectified linear unit used in this research [3]. Commonly, a max pooling layer is then used after a set of convolutional layers to reduce the feature map size for later layers [2]. After the series of convolutional layers are done, a fully connected layer takes the high-level features from the dataset and determines the non-linear combinations of these features. A technique used in this research for the fully connected layer known as Dropout, helps to add extra noise into the training process by randomly dropping units and the resulting connections from the neural network which helps in reducing overfitting.

Accuracy and percentage error have been a benchmark for many of the architectures presented for the MNIST dataset. With many CNN's having 98% or more accuracy, we look at what complexities other

architectures have implemented that allow them to achieve high 99% accuracies. By developing a CNN, we evaluate its performance with varying hyperparameters, and compare the features of other architectures that aided in its effectiveness.

Relevant Literature Review

To see how this model faired to others we compared our results and techniques to the top architecture by Sanghyeon An et al [10]. In their paper *An Ensemble Of Simple Convolutional Neural Network Models For Mnist Digit Recognition*, they managed to achieve a test accuracy score of 99.91% through testing separate models of convolutional layers with different kernel sizes [10]. The model that faired the best consisted of a kernel size of 3x3 and used a set of convolutional layers followed by a fully connected layer like how we had done and many others using CNNs. Their key to breaking the 99.9% was the use of two-level ensemble networks to combine results from three networks of any kernel size.

In comparison to our model, many of the hyperparameters remained similar such as batch size (120 – 128), epochs (50-150), and learning rate (0.001-0.01) [10]. We each used a set of convolutional layers with ReLU activation and ending with a fully connected layer. They on the other hand did not do pooling since they had no padding, which resulted in subsequent feature maps still being reduced. Their models consisted of networks with different kernel sizes M3 (3x3), M5 (5x5), and M7(7x7) which allowed for different feature maps [10]. This was an important feature of theirs as when used with two-level ensemble methods, it allowed them to compare test results with other random networks. Unlike our training method of a single network, theirs trained 30 networks each of M3, M5, and M7 and selected a network from each type. The final results were taken from a majority vote of the 3 random networks. This process was repeated to create an ensemble of ensemble results from specific network types to achieve their best accuracy and average.

It is clear from their training compared to ours, that even though CNN models can achieve high accuracies of 99%, it alone can become saturated between 99-99.8%. By further collecting results through ensemble methods, the architecture can make up for the minute short comings of a singular result.

We also compared our model to the top model created by Adam Byerly et al () [11]. They were able to achieve an accuracy score of 99.83% in a single model and 99.87% in an ensemble in their paper *No Routing Needed Between Capsules* [11]. They did this by first using Homogeneous Vector Capsules to remove the need for dropout regularization, routing mechanisms or reconstruction sub-networks. They were also able to use 5.5 times fewer parameters and 4 times fewer training epochs than the top model at the time [11]. This greatly simplified the model and made it less complex and more efficient while maintaining the 99.83% accuracy of the top existing model [11]. They then used multiple classification branches with each one making its own prediction and took weighted votes from each to create an ensemble with 99.87% efficiency [11].

While the epochs were reduced from 1200 to 300 in their model compared to the top existing model [11], it still dwarfs the 10 epochs used in our model. Their model had a base learning rate of 0.001 plus 0.98 per epoch [11]. We each used a set of convolutional layers with ReLU activation and ending with a fully connected layer. They did not use pooling as the information in the set is limited and they did not want to waste any. They also removed padding in order to reduce dimensionality in deeper layers. The padding was not found to make a substantial difference, so it was fine to remove. Instead of using neurons like our model, their model used Homogeneous Vector Capsules to represent filters. The capsules were constructed from distinct X and Y coordinates from the combination of all the feature maps. This approach made their model more accurate than ours, which was furthered using significantly more

epochs. They branched off the output into 3 branches, each ascending branch increasing in number of filters and receptive field size [11]. They used backpropagation to learn ideal branch weights which were used as part of a weighted vote system for each image. This maximized the accuracy of the ensemble of branches as it allowed the branches to cover for each other's weaknesses.

Their methods proved that Homogeneous Vector Capsules can be used to achieve a greater accuracy at the same number of epochs. It also showed that using multiple branches and taking a weighted vote from each of them can cause a slight boost in accuracy in the model.

Experimental Setup and Methodology

To create a simple convolutional neural network (CNN), we used python as our programming language and TensorFlow as our machine learning library. Using TensorFlow, we can import the layers and models that need to be utilized. After importing the necessary prerequisites for implementing the CNN, the data will need to be preprocessed. Firstly, we load the dataset into training and testing sets for both input and output. We then reshape our training and testing data to the color channel of 1 as the dataset has grayscale images. To make our CNN more efficient, the values of pixels of the images are then normalized by dividing them by 255. By doing this, our values will be between 0 and 1 as the original values are between 0 and 255. Before defining the model, we first must check the shape of the input images and assign it to a variable that will be used later on. When this is done, we can proceed with defining the model. The model we will be using is a sequential model. The sequential model is a plain stack of layers where each layer will have one input and output tensor. The sequential model was chosen over a functional model as it is simpler and more convenient for implementation although functional models are more flexible. The first layer we add to the model is a convolutional layer. The hyperparameters chosen for the first convolutional layer are 32 filters [2], a kernel size of 3x3 which is a standard size of the matrix [2], ReLU as the activation hyperparameter since the neural network is then easier to train and achieves better performance, and the input shape being the image shape we derived earlier. Following the Convolutional layer is the max pooling layer which will give the maximum element from what the filter covers in the feature map [2]. Following this is the repetition of a convolutional layer and a max pooling layer. However, we decided to double the filters on the second layer taking inspiration from VGG16 which doubles the number of filters between each convolutional layer [1]. Afterwards, we implement the dropout layer to reduce overfitting. The rate we chose is 0.5, which is the fraction of the input units we will drop. Next, the flatten layer will be added to the model which will flatten the input making the array one-dimensional. Finally, for the two dense layers, we chose 400 units for the first and 10 for the output dense layer. Generally, the SoftMax function is used in the last output layer hence us using it in the last dense layer [6]. With this, the sequential model is complete and just needs to be compiled. Using the compile method, we will set the necessary hyperparameters and run the model. For the optimizer, we chose the Adam algorithm and for the loss function, we chose the standard for convolutional neural networks which is a sparse categorical cross-entropy [4][6][7]. Furthermore, the fit method runs 10 epochs with an average speed of 23 seconds per epoch. To see more of the data, we gave verbose the value of 2. After a manual search, the most effective values for the hyperparameters were a batch size of 1024 and a validation split of 0.2. Finally, we calculated the accuracy with the evaluation method.

Discussion

Much of the testing conducted in this paper came from the manipulation of values considered standard in a convolutional neural network using a manual search. The starting point of the network were values that are generally used. By changing the values, we saw an increase from 98.19% to 99.45% accuracy. The first values we decided to change were the filters for the convolutional layers. Doubling the values every time for a new convolutional layer worked fairly well compared to keeping them the same [1]. The change of filters depends on the dataset so finding the right values would be based on trial and error for each individual dataset. Furthermore, to expand upon why the activation function ReLU was chosen over a sigmoid function is because the gradient of the sigmoid multiplies when there are many layers. This will eventually add up. Even if the gradient is small, it will lead to a slow convergence. This is also called the vanishing gradient problem [3]. Here, the ReLU activation proves effective as the gradient isn't a fraction between 0 and 1 like a sigmoid activation but is instead either 0 or 1 [3]. With multiple layers, the gradient will not be too small or large if there is a product of many 1s. Secondly, the sigmoid activation gradient goes to zero if the input is too large or too small. When this happens, the convergence is very slow. Unlike its counterpart, the ReLU activation gradient only goes to zero if the input is too small but not too large. So here, ReLU activation will have only "half" of the issues of a sigmoid activation [3]. The change in values in the dropout layer also decreased the accuracy by around 1 percent when changing it to anything other than 50 percent (0.5). With a batch size of 1024, the dropout layer is very useful in preventing overfitting in training CNNs. If one is not used, the learning is influenced by the first batch of training in a high manner. The nodes for the first dense layer also decreased accuracy when it wasn't in the range of 400-500 units with an average of 98.2%. To compile the model we built so far, the Adam algorithm was used because of its efficiency over other algorithms [8].

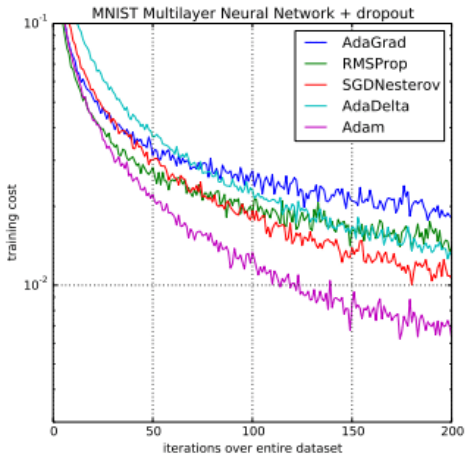

Figure 1: different optimizing algorithms and their efficiency

Figure 1 [3] displays the efficiency of the Adam optimizer algorithm against others on large datasets such the MNIST dataset. Finally, the batch size value was changed based on a manual search.

Batch size	**Accuracy**	**Speed per epoch**
2	98.19%	124s/epoch
16	99.12%	37s/epoch
32	99.30%	30s/epoch
64	99.23%	27s/epoch

512	99.43%	24s/epoch
1024	99.45%	23s/epoch

Values of different batch sizes on MNIST dataset

When performing the testing, we observed that the results improved when the batch size was larger. The model hence performs better on the Adam optimizer when the batch size is usually larger. Batch size is tested with numbers to the power of 2 to take advantage of parallel computing in the GPUs. Making the batch size too large will lead to a generalization and overfitting problem so we kept it to 1024.

Conclusion

In this paper we demonstrated how using conventional methods for designing a Convolutional Neural Network and fine tuning its hyperparameters, can create a very accurate model. We created a CNN model to use on the MNIST dataset. The MNIST dataset contains 60,000 images of handwritten numbers, numbered 0 to 9. Our CNN model learns to differentiate between these numbers. We took a standard approach in designing our neural network, choosing design methods which we thought best suited the purpose of our model. Beginning with standard parameters we were able to achieve an accuracy of 98.19%, after fine tuning these parameters to optimize our accuracy we were able to achieve as high as 99.45%.

We used python and TensorFlow as they are standard for CNNs and very easy to use. Ease of use is also the reason why we decided to create a sequential model in place of a functional model. This was a good tradeoff for the extra flexibility of the functional model as our CNN is quite simple and standard. We used ReLU activation on our CNN, which contains a pair of alternating convolutional and pooling layers, a dropout and flattening layer and 2 dense layers at the end. Using insight from VGG16, we found out that doubling the convolution and pooling layer filter values in the second pair of those layers increased our model's accuracy. We used Adam's algorithm for optimization due to it being highly efficient. Sparse categorical cross-entropy was used for calculating loss as it is standard in CNNs [7]. We manually adjusted the dropout layer values and batch size until we found the values which gave us the greatest accuracy. ReLU was favored over sigmoid for the activation as it does not have problems with slow convergence on small input sizes.

The top models in terms of accuracy for the MNIST dataset used ensembles to boost their accuracy by having some models make up for the shortcomings of others through voting systems. One of the models found that using various kernel sizes in their CNN boosted accuracy and another found the same with using Homogeneous Vector Capsules in place of neurons for representing filters. Their models were much more complex and unconventional when compared to ours. Thus, though our model had marginally lower accuracy, it was designed to be simple and still achieved a very high accuracy while meeting that goal.

References

[1] Great Learning Team, "What is VGG16 - Convolutional Network for Classification and Detection," *Great Learning Blog: Free Resources what Matters to shape your Career!*, Oct. 01, 2021. https://www.mygreatlearning.com/blog/introduction-to-vgg16/ (accessed Oct. 16, 2022).

[2] "CNN | Introduction to Pooling Layer - GeeksforGeeks," *GeeksforGeeks*, Aug. 05, 2019. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (accessed Oct. 16, 2022).

[3] J. Brownieee, "A Gentle Introduction to the Rectified Linear Unit (ReLU)," *Machine Learning Mastery*, Aug. 20, 2020. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (accessed Oct. 16, 2022).

[4] M. Walia, "Keras.Conv2D Class - GeeksforGeeks," *GeeksforGeeks*, Jun. 26, 2019. https://www.geeksforgeeks.org/keras-conv2d-class/ (accessed Oct. 16, 2022).

[5] Keras Team, "The Sequential model," *Keras: the Python deep learning API*. https://keras.io/guides/sequential_model/#:~:text=A%20Sequential%20model%20is%20appropriate,%22layer1%22)%2C%20layers. (accessed Oct. 16, 2022).

[6] R. Pramoditha, "Coding a Convolutional Neural Network (CNN) Using Keras Sequential API," *Towards Data Science*, Jun. 27, 2022. https://towardsdatascience.com/coding-a-convolutional-neural-network-cnn-using-keras-sequential-api-ec5211126875 (accessed Oct. 16, 2022).

[7] V. Yathish, "Loss Functions and Their Use In Neural Networks," *Towards Data Science*, Aug. 04, 2022. https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9#:~:text=The%20most%20commonly%20used%20loss,of%20the%20pre%2Dset%20categories. (accessed Oct. 16, 2022).

[8] GeeksforGeeks, "Intuition of Adam Optimizer - GeeksforGeeks," *GeeksforGeeks*, Oct. 22, 2020. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/#:~:text=The%20method%20is%20really%20efficient,How%20Adam%20works%3F (accessed Oct. 16, 2022).

[9] S. Panchal, "No, Kernels & Filters Are Not The Same," *Towards Data Science*, Oct. 09, 2021. https://towardsdatascience.com/no-kernels-filters-are-not-the-same-b230ec192ac9 (accessed Oct. 16, 2022).

[10] S. An, M. Lee, S. Park, H. Yang, and J. So, "AN ENSEMBLE OF SIMPLE CONVOLUTIONAL NEURAL NETWORK MODELS FOR MNIST DIGIT RECOGNITION." [Online]. Available: https://arxiv.org/pdf/2008.10400v2.pdf (accessed Oct. 16, 2022).

[11] A. Byerly, T. Kalganova, and I. Dear, "No routing needed between capsules," *Neurocomputing*, vol. 463, pp. 545–553, Nov. 2021, doi: 10.1016/j.neucom.2021.08.064 (accessed Oct. 16, 2022).

[12] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.