Comparative Study of an ANN and k-NN with the MNIST Dataset

Oliver Aarnikoivu
School of Computing Science and Digital Media
RGU
Aberdeen, UK

Abstract—**In this paper, we present a comparative study of an Artificial Neural Network (ANN) and K-Nearest Neighbour (k-NN) model in recognizing digits from the MNIST dataset. We evaluate the influence of different hyperparameters and their impact on model accuracy. Furthermore, we explore image rotation as an augmentation technique to assess the robustness of both models. Additionally, model performance is tested using our own digits generated using GIMP, a free and open source image editor as well as on handwritten digits using a whiteboard. Lastly, we propose a hybrid system where we combine the ANN and k-NN as a means to improve data representation. We make use of an autoencoder in which the input is encoded such that the hidden layer focuses only on the most critical features. The hidden activations are then used as the input into our k-NN. When using the optimal hyperparameters of the ANN, the hybrid system did not prove better results, however, when tuning the batch size, of the autoencoder, we observed a slight increase in accuracy in comparison to both the ANN and k-NN.**

Keywords—ANN; k-NN; MNIST; GIMP; Autoencoder

I. Comparative Study setup

We begin by evaluating the behavior and impact of the different hyperparameters on the performance of the ANN and k-NN. With regards to the ANN, we specifically investigate the role of epochs, learning rate, batch size and batch size with varying hidden nodes on model accuracy. For the k-NN, we evaluate the behavior of "k", weighted and unweighted voting functions as well as different similarity and distance metrics on accuracy. We show that using cosine similarity has significant advantages in terms of computational complexity, thus, decide to use it as our similarity metric for all k-NN related tasks. Furthermore, for both the ANN and k-NN we train using 1500 samples and test on 10000 examples. For section II, we train using the following values for hyperparameters:

- Epochs: 10, 25, 40, 50, 80, 100
- Learning rate: 0.01, 0.03, 0.1, 0.3, 0.6
- Batch size: 1, 20, 100, 200, 500, 1500
- Hidden nodes: 5, 200

For section III we evaluate impact on performance using the following hyperparameters:

- k: 1, 3, 21, 51, 81
- Weighted and Unweighted voting
- Euclidean distance, Manhattan distance, cosine similarity and dot product.

We demonstrate that 50 epochs, 0.3 learning rate, 1 batch size and 200 hidden nodes resulted in optimal performance for the ANN, whereas for the k-NN, we observed best results using weighted voting with k being equal to 3. Therefore, these optimal hyperparameters were used along with our hybrid implementation.

II. Neural Network

A. Neural Network Hyperparameters

In the case of Artificial Neural Networks, hyperparameters can be defined as variables which determine the structure of the network, as well as the variables which establish how the network is trained [1]. Some of the key hyperparameters include epochs, batch size and learning rate.

a) *Epochs* represent the number of times an entire dataset is passed forward and backward through a neural network, meaning that each time a neural network has "viewed" all features in a dataset, one epoch has been completed [2].

b) Batch size is defined as the number of examples considered within a single epoch before updating network parameters [3]. It is important to consider batch size when it comes to gradient descent. We can imagine gradient descent to be costly if our training set is large as it will be slower to update weights and take longer until converging to a global minimum. A solution to this is to make use of stochastic gradient descent (SGD) where in a given epoch, we generate batches from the training set and update weights based on an error computation for each batch. For example, if batch size was set to 1 then weights are updated after each training example [4], [5].

c) Learning rate is the amount of moderation used to manage by how much we adjust the weights of a network [6]. Choosing the correct learning rate is problematic as a value too small may cause the training process to be longer whereas a large learning rate may cause gradient descent to overshoot the minimum by either failing to converge or diverging [7].

B. *Visualisation of Results*

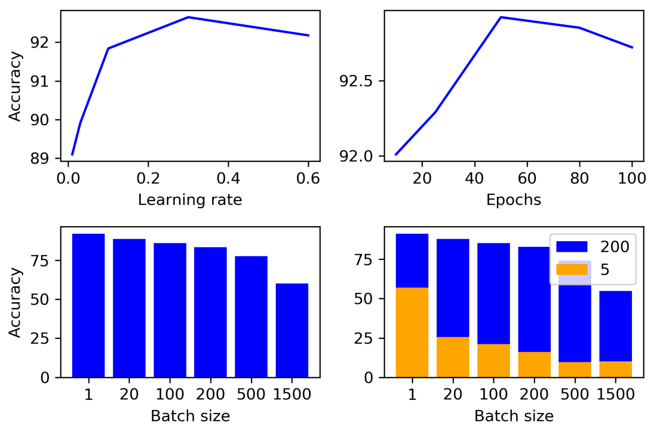

Fig 1: Graphs comparing the impact of varying hyperparameters on ANN accuracy—Learning rate, top left: Epochs, top right: Batch size: bottom-left: Batch size with hidden nodes: bottom-right.

C. *Discussion of Results*

When using accuracy as a sufficient metric of a good model, Figure 1 clearly indicates that the optimal learning rate is at 0.3. We can see that as the learning rate increases towards 0.3 so does accuracy, however, as the learning rate increases past this point, accuracy begins to fall. It's evident that none of the provided learning rates drastically affect the model's performance as accuracy overall ranges between 89% and 93%, nevertheless, the clear downward shift beginning from 0.3 could imply that as the learning rate increases, the model is beginning to converge too quickly to a suboptimal solution [6].

Interestingly, we can see that when batch size is at 1, the accuracy of the model is highest and as batch size is increased, accuracy decreases. Additionally, we can see that when batch size is set to the size of the training data, i.e. 1500, accuracy takes a substantial fall to approximately 60%. When using batches, we are computing the average error which results in fewer weight updates meaning that training time will likely be faster, however, a local minimum may not be found [4].

Furthermore, we can see that accuracy rapidly increases from 10 to 50 epochs and then steadily begins to decrease. In general, even though accuracy is above 90% for all epoch instances, it is possible that as we increase the number of epochs, we begin to overfit the data indicating that our model is "memorizing" data as opposed to learning [8].

III. K-NEAREST NEIGHBOUR

A. *k-NN Hyperparameters*

k-NN consists of multiple hyperparameters that can influence the overall performance of the algorithm. Some of the key hyperparameters to consider are "k", similarity and distance metrics, and voting functions. Below we evaluate the role of the hyperparameter "k" and the differences between the unweighted and weighted voting functions.

a) k represents the size of the "neighborhood", i.e. the number of nearest neighbors used to classify or predict outcomes from a dataset meaning that if k = 1, then testing samples are given the same label as the closest example in the training set, whereas if k = 3 then the labels of the 3 closest classes are used to identify the most common label [9].

b) Unweighted and *weighted* functions will differ from each other depending on if the domain is a regression or classification problem. In unweighted voting, all instances in the neighborhood have uniform weight whereas for weighted voting, each instance in the neighborhood contributes in proportion to some weight, typically using similarity [10].

B. *Choice of similarity metric*

We observed a significant advantage in using *cosine similarity* as a similarity metric. Figure 2 justifies our choice as we can clearly see that accuracy is only slightly lower than if we were to use the Euclidean distance. However, as also shown by the right-hand side image, the computational complexity of cosine similarity is multiple orders of magnitude less than that of the Euclidean distance. Hence, the time it takes to test is considerably reduced whilst still recording sufficient results.

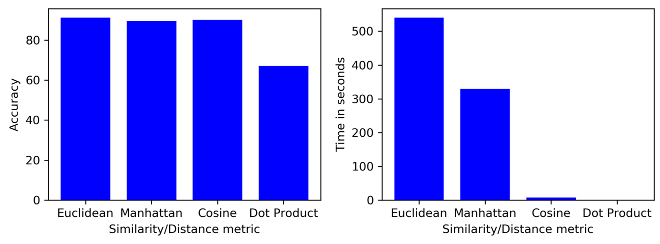

Fig 2: Bar charts comparing the impact of similarity/distance metric on accuracy and time—Similarity/Distance vs Accuracy: left, Similarity/Distance metric vs Time in seconds, right.

C. *Visualisation of Results*

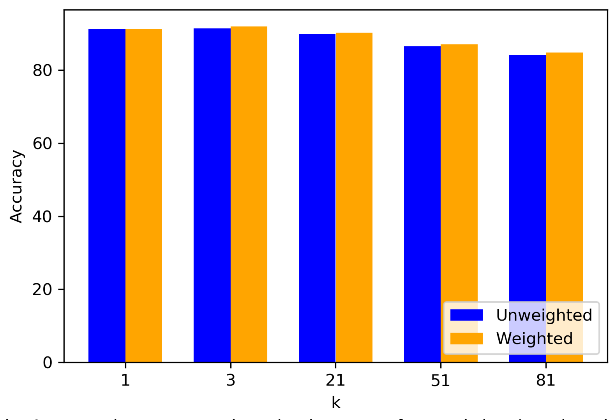

Fig 3: Bar chart comparing the impact of unweighted and weighted voting on k-NN performance with increasing "k".

D. *Discussion of Results*

With k-NN tested on the MNIST dataset using cosine similarity, we can clearly see that for both weighted and unweighted voting, accuracy is at its highest when k = 3. In addition, the results indicate that both weighted and unweighted voting perform best on lower values of k, while at higher values we observe a decrease in performance. This signifies that at higher values, both models struggle to consider information relevance based upon neighbour distance. We can also see that as k is increased, weighted voting performs slightly better in comparison to unweighted voting suggesting that for larger values of k, k-NN with weighted voting is better able to consider points from other classes into the neighborhood.

IV. Adding New Training Data

A. *Strategy for using rotated data*

Using more variations of data on a model can lead to an improvement in performance due to being able to learn additional discrepancies [11]. In order to make use of augmented training on the MNIST dataset, we define a function to rotate images both clockwise and anticlockwise by a random amount within a range of 0° and a specified maximum value. This ensures our results to be accurate in comparison to the ANN and k-NN as we don't need to append the rotated data to our training data. Instead, we loop through the training samples and apply the function to each instance. This assures that some instances remain unchanged whilst others will be rotated. We apply the image rotation function to the existing condensed training set consisting of 1500 samples. Additionally, as a means to ensure randomness we define another function to shuffle the data whilst keeping the order of values to targets intact.

B. *Results with added training data*

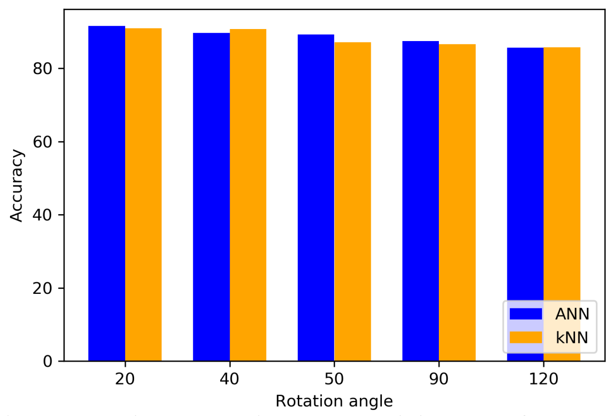

Fig 4: Bar chart comparing ANN and k-NN performance with increasing angles for image rotation.

It appears that performance for both the ANN and k-NN is at its highest when images are rotated with a maximum of 20°. Likewise, as the rotation range is increased accuracy begins to decrease for both models. Furthermore, both the ANN and k-NN prove accuracy to be lowest at 120° which is logical as such a large rotation significantly transforms the structure of the original image.

V. Adding Your Own Handwriting

A. *Creating my own handwriting characters*

In order to create our own handwritten digits, we made use of *GIMP*, a free and open source image editor where we create 28 by 28-pixel images for all possible input numbers and stored the images as PNG files. We also handwrite digits on a whiteboard and capture an image of each number using an iPhone 8. We transfer the images as HEIC files onto our computer and convert them to 28 by 28-pixel PNG images using Apple's image editing software.

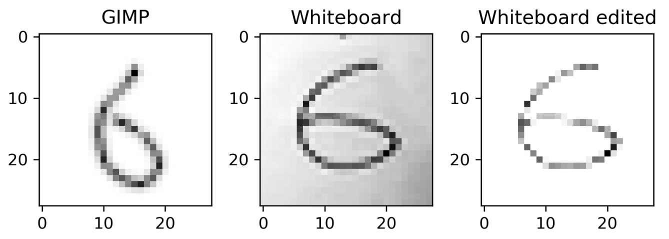

Fig 5: Comparison of noise between digits written using GIMP, on whiteboard and whiteboard edited using software—GIMP: left, Whiteboard: middle, Whiteboard & Edited: right.

B. *Testing on my own handwriting*

Model	GIMP Accuracy	Whiteboard Accuracy	Edited Accuracy
ANN	80.00%	11.11%	22.22%
Aug-ANN	**90.00%**	11.11%	22.22%
k-NN	70.00%	0%	55.56%
Aug-k-NN	**90.00%**	**33.33%**	**66.67%**

Table 1: Comparison of ANN & k-NN performance on digits created using GIMP, with a whiteboard, and with whiteboard images edited—Aug-ANN (Augmented ANN using image rotation), Aug-k-NN (Augmented k-NN using image rotation).

We show that our ANN model is able to reliably classify digits written using GIMP with 80% accuracy. In addition, our results indicate that the normal k-NN performs slightly worse in comparison to both neural network models. We noticed that some of the written digits looked slightly rotated and therefore decided to test digits on our ANN model and k-NN utilizing image rotation with a maximum rotation of 45°. Both the ANN and k-NN recorded an accuracy of 90%. This proves that when using image rotation as an augmentation technique, both models become more robustious.

We were interested to assess the performance of the neural network models on our own handwritten digits written on a whiteboard. Without using any editing software to make our images appear "cleaner", we were not surprised to see that our model showed inadequate predictions. This is likely due to the amount of noise that was captured using a phone as can be seen by figure 5. We attempt to denoise the original whiteboard images using Apple's image editing software by adjusting the exposure, contrast and sharpness. As shown by figure 5, we were able to significantly reduce the amount of noise. Furthermore, using this technique, we slightly increased accuracy from 11.11% to 22.22% for both basic neural network, and augmented neural network models. On the other hand,

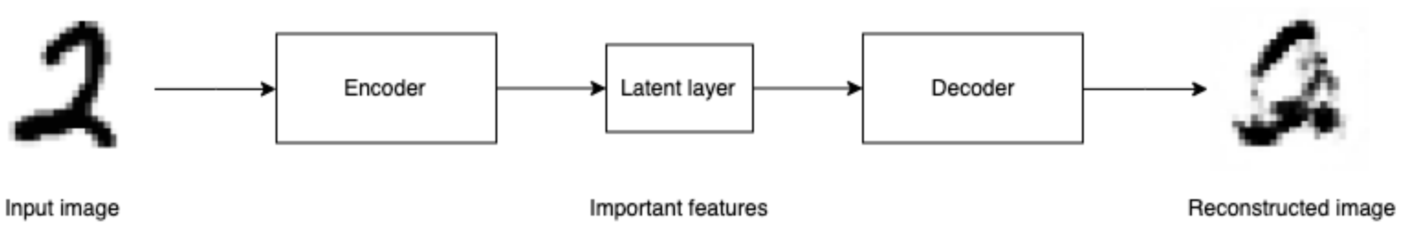

Fig 6: Illustration of a standard feedforward autoencoder**.** The autoencoder takes in as input an image and attempts to produce an estimate of the input by only focusing on the crucial features. The learning of the critical features occurs in the latent layer [12].

whilst k-NN was not able to correctly predict any of our handwritten whiteboard digits, it made a substantial jump from 0% to 55.56% on the same whiteboard digits which were edited using image editing software. Likewise, our augmented k-NN model increased from 33.33% to 66.67%. We found it surprising that the augmented k-NN would outperform the ANN on our own handwritten whiteboard digits considering a neural networks ability to learn from its mistakes. It's important to note that our neural network is trained on well-defined images provided by the MNIST dataset. The digits are also largely all positioned in the center of the image. Therefore, our model has not been trained to pick up on any inconsistencies. In order to build a more robust model, we would require training using other image augmentation methods along with our image rotation technique. For example, we could train our model on digits that consist of more background noise, on random shifts and random zooms [13].

VI. Hybrid

A. *Combining ANN and k-NN*

Since neural networks are able to learn high-level features from data in a gradual manner, the embeddings from a trained neural network can provide a suitable learned representation as input to a machine learning model for a supervised task, such as digit classification [10]. We propose using a standard feed forward autoencoder where we adjust the ANN such that we set the target values to be equal to the inputs. This allows our model to learn a lower dimensional feature representation [14]. Figure 6 and 7 provides a simplistic representation of our autoencoder. The inputs *x1, x2, x3, x4* represent the input vector of size 784. The hidden layer is kept at 200 nodes. It's crucial that the hidden layer is undercomplete as this forces our hidden layer to capture meaningful variation in the data [15]. The outputs *x1', x2', x3', x4'* represent the reconstructed input data. The hidden features (Latent layer) are provided as input to the k-NN.

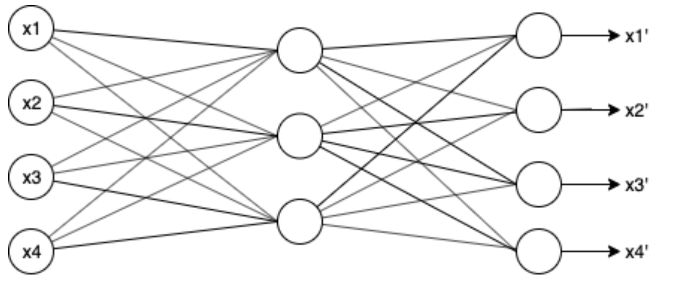

Fig 7: Illustration of a standard feedforward autoencoder.

B. *Visualisation of Results*

Model	Accuracy
MNIST	
ANN	**92.65%**
k-NN	91.43%
Hybrid	83.91%
GIMP Digits	
ANN	80%
k-NN	70%
Hybrid	80%
Handwritten whiteboard digits	
ANN	11.11%
k-NN	0%
Hybrid	0%

Table 2: Comparison of ANN, k-NN and hybrid model accuracies using the optimal hyperparameters obtained from section I and section II.

C. *Discussion of Results*

When using the optimal hyperparameters obtained from section II for our ANN model and section III for k-NN, we see that the hybrid variation is considerably worse in comparison to both the ANN and k-NN when assessing against the 10000 MNIST testing samples. The results do not strongly indicate that the activations our autoencoder encodes using the attained optimal hyperparameters provide a suitable learned representation as input into a k-NN.

We were curious to see if the hybrid system could beat both models in terms of accuracy by tweaking the number of epochs, hidden nodes, batch size and learning rate. As shown by figure 8, when using the optimal hyperparameters obtained from our ANN model, we can clearly see that the trajectory of using 1 batch size fails to minimize the loss. On the other hand, with batch size increased to 32 we see that our model is able to generalize much better. This suggests that an increase in batch size could allow our model to better capture meaningful variation in the data, which in return would improve accuracy. To test our hypothesis, we train our autoencoder on varying batch sizes and plot the effect on accuracy.

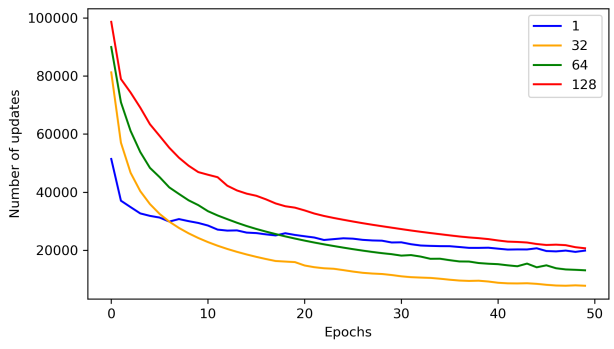

Fig 8: Impact of varying batch size on model error with increasing epochs—tested with batch sizes of: 1: blue, 32: orange, 64: green, and 128: red.

As shown by Figure 9, our supposition was correct as we observed an accuracy of **93.4%** by adjusting batch size to 24 and **92.2%** with 32 whist keeping all other hyperparameters unchanged. This is a significant improvement from the accuracy obtained with just 1 batch size. Although just a minimal improvement from the accuracy obtained by the ANN and k-NN, the shift in performance could suggest that when using the outputs of an autoencoder as the inputs to a supervised machine learning algorithm, data representation is improved.

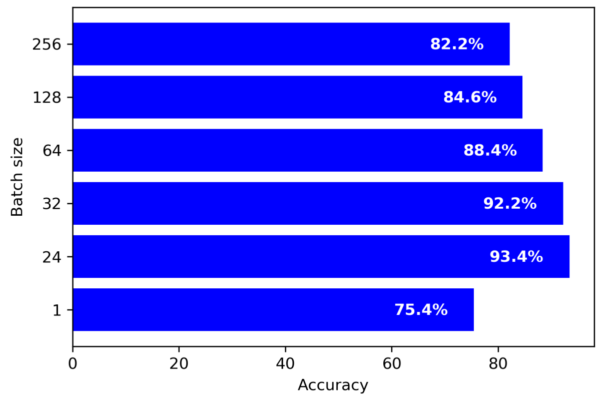

Fig 9: Bar chart showing the impact of different batch sizes on hybrid system accuracy—Tested on 500 samples.

To further verify our hypothesis, we compare the output images of our autoencoder trained with just 1 batch size to that of 24 and notice a significant difference in overall quality. As can be seen by Figure 10, the autoencoder using a batch size of 24 is able to better reproduce the input images as opposed to with a batch size of 1.

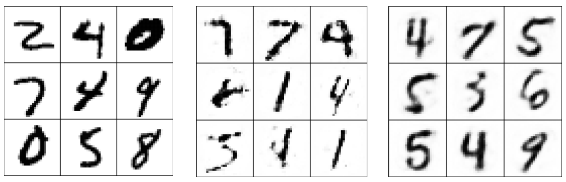

Fig 10: Comparison of random images captured by autoencoder. Input images: left, output images with batch size of 1: middle, output images with batch size of 24: right.

REFERENCES

[1] P. Radhakrishnan, "What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?," Towards Data Science, 2017.

[2] S. Sharma, "Epoch vs Batch Size vs Iterations," Towards Data Science, 2017.

[3] J. Browniee, "What is the Difference Between a Batch and an Epoch in a Neural Network?," Machine Learning Mastery, 2018.

[4] K. Shen, "Effect of batch size on training dynamics," Medium, 2018.

[5] Robert Gordon University, "Artificial Neural Networks (ANN) - Lecture," 2019.

[6] H. Zulkifli, "Understanding Learning Rates and How It Improves Performance in Deep Learning," Towards Data Science, 2018.

[7] J. Browniee, "Understand the Impact of Learning Rate on Neural Network Performance," Machine Learning Mastery, 2019.

[8] "Can the number of epochs influence overfitting?," StackExchange , 2018. [Online]. Available: https://datascience.stackexchange.com/questions/27561/can-the-number-of-epochs-influence-overfitting. [Accessed 17 October 2019].

[9] M. Krause, "StackExchange," 2015. [Online]. Available: https://stats.stackexchange.com/questions/134309/what-does-the-k-value-stand-for-in-a-knn-model. [Accessed 17 October 2019].

[10] Robert Gordon University, "Instance-Based Learner (kNN) - Lecture".

[11] A. H.-G. a. P. Konig, "Further advantages of data augmentation on convolutional neural networks," 2019.

[12] Guru99, "Guru99," [Online]. Available: https://www.guru99.com/autoencoder-deep-learning.html. [Accessed 23 October 2019].

[13] "Easy Image Augmentation Techniques for MNIST," Kaggle, 2017. [Online]. Available: https://www.kaggle.com/dhayalkarsahilr/easy-image-augmentation-techniques-for-mnist. [Accessed 21 October 2019].

[14] A. Ng, "CS294A Lecture Notes," [Online]. Available: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf. [Accessed 23 October 2019].

[15] N. Hubens, "Deep inside: Autoencoders," Towards Data Science, 2018.