Received: 6 July 2020 | Revised: 13 February 2021 | Accepted: 1 March 2021

DOI: 10.1002/cta.2998

ORIGINAL PAPER

Design of classifiers based on ANN approximations of traditional methods

Natan Kruglyak[1] | **Robert Forchheimer**[2]

[1]Department of Mathematics, Linköping University, Linköping, Sweden

[2]Department of Electrical Engineering, Linköping University, Linköping, Sweden

Correspondence
Robert Forchheimer, Division of Information Coding, Department of Electrical Engineering, Linköping University, Linköping SE-581 83, Sweden.
Email: robert.forchheimer@liu.se

Abstract

In this paper, we suggest basing the development of classification methods on traditional techniques but approximating them in whole or in part with artificial neural networks (ANNs). Compared to a direct ANN approach, the underlying traditional method is often easier to analyse. Classification failures can be better understood and corrected for, while at the same time faster execution can be obtained due to the parallel ANN structure. Furthermore, such a two-step design philosophy partly eliminates the 'guesswork' associated with the design of ANNs. The expected gain is thus that the benefits from the traditional field, and the ANN field can be obtained by combining the best features from both fields. We illustrate our approach by working through an explicit example, namely, a Nearest Neighbour classifier applied to a subset of the MNIST database of handwritten digits. Two different approaches are discussed for how to translate the traditional method into an ANN. The first approach is based on a constructive implementation which directly reflects the original algorithm. The second approach uses ANNs to approximate the whole, or part of, the original method. An important part of the approach is to show how improvements can be introduced. In line with the presented philosophy, this is done by extending the traditional method in several ways followed by ANN approximation of the modified algorithms. The extensions are based on a windowed version of the nearest neighbour algorithm. We show that the improvements carry over to the ANN implementations. We further investigate the stability of the solutions by modifying the training set. It is shown that the errors do not change significantly. This also holds true for the ANN approximations providing confidence that the two-step strategy is robust.

KEYWORDS

approximation, classification, machine learning, nearest neighbour, neural networks

1 | INTRODUCTION

The fast development of artificial neural networks (ANNs) in recent decades and in particular its successful applications in image analysis has completely changed the direction of the research in this field.[1] Prior to this, a fairly systematic approach had emerged for image analysis and object classification in which handcrafted image features were extracted followed by classification or regression. Feature extraction and classification were typically studied as separate disciplines. Research on feature extraction culminated in the late 1990s and early 2000s.[2,3] Apart from being useful for classification, these features were also used in image matching such as finding corresponding points in stereo images. Classification methods were either rule-based, based on non-parametric models or parametric models such as statistical techniques leading to linear and non-linear discriminants.[4] The main drawbacks with ANNs are well known. There is no general theory that tells how many layers and neurons a network should have to fit a particular problem. The network architecture (type of layers and choice of activation functions) also has to be chosen, a task that requires human experience in combination with trial-and-error runs. The back-propagation scheme is tedious, and its result depends on the initial parameter values. Since these are often randomised, repeated training sessions result in differences in performance. However, once trained, the ANN network often performs better and may even be more efficient, from a computational point of view, than a traditional method. Our work aims at finding bridges between the traditional methods and the ANN approach by basing the development on traditional techniques which are then turned into ANN solutions. The expectation is that ANNs which are strongly related to a traditional method will be better understood when it comes to performance (and failures). Also, this could possibly lead to improvements of the traditional method as well, so that it approaches the performance of a direct ANN solution.

From the Universal Approximation Theorem, it is known that an ANN with sufficient number of neurons can approximate any continuous function arbitrarily well.[5] Implementing functions in this way ('constructive ANN') has the benefit that the ANN architecture can usually be directly derived from the function to be implemented. However, requiring exact implementations typically leads to very large networks. As an alternative, we will show examples where parts of the function defining a classifier can be approximated by ANNs of lower complexity. Such implementations can be done constructively or through training of the ANN networks. Although, in this latter case, the 'guesswork' has partly returned, those parts typically have a well-defined task as we will see later on and can be easily checked for their performance and be replaced or retrained if necessary to reach the required performance.

In this paper, we illustrate the above principle by implementing local approximations with ANNs. When searching for a problem area for which such an approach make sense, we focus on the Nearest Neighbour (NN) algorithm used in image classification.

Our starting point will be the baseline NN method which we apply to a two-class classifier. We then extend the basic scheme to a windowed version (WNN). We describe the constructive ANN implementation followed by examples of how the parts of WNN can be implemented by ANNs of lower complexity. In principle, these approaches cannot be expected to give better performance than a well-designed ANN. However, this is not our goal as we are primarily interested in generating ANN classifiers which are strongly rooted in a corresponding traditional method to allow interpretability. For illustration, we apply our classifiers to a subset of the MNIST[6] database.

The paper is organised as follows. Following a summary of related works, the classification task and our use of the MNIST data are presented in Section 4. The NN method is presented in Section 5. Section 6 describes the extended version and how to approximate parts of it with ANNs. In Section 7, we introduce two improvements and show how these carry over to the ANN case. Section 8 illustrates that the stability of the traditional method to changes in training data is also seen in the ANN case. In Section 9, we discuss our approach and present some remarks. We summarise the results in Section 10. The appendices give more details about the simulation results and how we obtained the number of layers and neurons for the constructive ANNs. Additional details about the extraction of data and the programs which implement our presented algorithms can be found in the supporting information.

2 | RELATED RESEARCH

Early works to cast traditional methods into the ANN framework are presented in Chen et al.[7] and Murphy.[8] Similar to our work, they start with methods based on NN or k-nearest neighbour (kNN). However, the aim in these papers has been to obtain precise emulation leading to very large numbers of neurons ($O(N^2)$ where N is the size of the training set). The approach taken in Park and Bang[9] is closer to our philosophy. Here, the decision boundaries (polytopes) given by NN are approximated by spheres. This will not lead to an exact emulation of the original method, but the complexity of the resulting ANN is reduced to $O(N)$.

Where 'understanding' the network output is concerned, the work in Yang et al.[10] follows our philosophy by arranging a set of ANNs into a decision tree. Our approach thus borders the field of ANN *interpretability* which aims at understanding why a network has arrived at its prediction, particularly in erroneous cases. Papernot and McDaniel[11] first train an ANN in the conventional way (by back-propagation). The output from each layer is then compared to the corresponding output produced by the members of the training set, and a kNN decision is made to check which layer has faulted. The same idea is used in Lee et al.,[12] but the kNN method is only applied to the last (*softmax*) layer to 'explain' the prediction in terms of the nearest training vector. The authors further use the outcome from the kNN to improve the classification performance. Another approach to understanding the behaviour of a neural network is to translate its decision boundaries into a set of rules or decision trees.[13] Although these approaches produce hybrid ANN/NN solutions, they differ from our work in that they implement, or approximate, the trained ANN with a traditional method while our proposal is based on the opposite, namely, to implement or approximate a traditional method by ANNs.

3 | WEAK LEARNERS AND STRONG CLASSIFIERS

One of the major contributions in the field of traditional algorithms is the concept of combining many *weak learners* (also known as *weak classifiers*) into a *strong classifier* as well as *boosting* to compute the weights that are used to combine those weak learners (Bishop,[4] pp. 655–659). Examples are Ensemble of Decision Trees where each tree can be considered a weak classifier[14] and the Viola-Jones algorithm[15] where the weak classifiers correspond to features computed over rectangular windows. It is interesting to note that these latter methods share a property with ANNs in that they jointly optimise the feature extraction step and the classification step. However, although trained ANNs seem to produce outputs from their first layer(s) which resemble those of traditional feature detectors, there is no specific separation within an ANN between feature extraction and classification. Rather, intermediate layers, particularly in deep ANNs, can be seen both as increasingly more complex feature detectors as well as early steps towards classification. Only the last layer, which typically includes one or several fully connected neurons, fits into the category of traditional classification, such as a threshold neuron or softmax layer, or, if regression is the aim, a linear neuron.

Our chosen architecture for the extended version of the NN classifier will be based on the concept of combining weak learners into a strong classifier. Traditionally, the weak learners may correspond to extracted features which are combined in a linear or non-linear way to obtain the strong classifier. Here, we will consider linear combinations similar to what is used in Ensembles of Decision Trees and in the Viola-Jones classifier. This will make the connection to the last (regression/softmax) layer of an ANN more evident. With this view, all previous ANN layers can be viewed as implementing features or weak classifiers. In this context, it is important to note that the weak classifiers do not necessarily output a final class membership. The output from them can be a multi-level or continuous-level signal. The starting point of our research was work on decision tree classifiers where pixels rather than windows are used.[16]

4 | THE CLASSIFICATION TASK AND MNIST DATABASE

Our classification task uses images and is defined as follows. Assume two sets, A and B, of labelled images. Suppose that we have an image X which is not in either of the sets but shares some properties which relate it to either A or B. The classification task is to recognise, with small error, if X (having an unknown label) belongs to set A or to set B. We will illustrate our approach using the MNIST database. This database contains 70,000 labelled images of handwritten digits from 0 to 9.[6] The images have the size of $28 \cdot 28 = 784$ pixels, each with grey levels between 0 and 255. In our simulations, we have selected 6,000 images from each of the two subsets '5' (set A) and '3' (set B). Furthermore, we binarise the images by thresholding the pixels. If a pixel value is less or equal to 127, it will be set to 0 otherwise to 1. We further divide the sets A and B into three parts: A_{tr}, B_{tr} each consisting of 4,000 images which are used for training; A_{val}, B_{val} each consisting of 1,000 images used for finding weights and biases when constructing ANNs and A_{test}, B_{test} each consisting of 1,000 test images for estimating the quality of the classification method.

5 | NN CLASSIFICATION

In the baseline version of the NN classifier the unknown image X is compared with each image Y in A_{tr} and B_{tr} using a distance function or similarity measure. The set which has an image that is closest to X would then be chosen as the output. As we work with binarised images, there is a very natural metric for the distance between images, namely, the number of pixels where values are different (the *Hamming* or L_1 distance) which is also equal to ρ^2, where ρ is the Euclidean distance. Alternatively, a similarity measure, e.g., based on the correlation or scalar product between the images, can be used.

5.1 | Performance of the baseline NN algorithm

Evaluation of the baseline NN algorithm is straightforward. The classifier does not need any training. Instead, all the training (annotated) data are used every time an inference is done. For each of the images in the training sets A_{tr} and B_{tr} containing altogether 8,000 images, we compute the image which is closest in Hamming distance to the test image and assign the test image to the corresponding class. Performance is evaluated in terms of the number of misclassified images. The software implementation (see the supporting information) reports that 38 images from A_{test} and 26 images from B_{test} are misclassified by NN leading to an error rate of 3.2%.

5.2 | ANN implementation of the baseline NN algorithm

Exact mapping of the NN algorithm into an ANN can be done in two principal ways. Either the discriminating hyperplanes (Voronoi areas) of the NN classifier are mapped into polytopes by a suitably designed ANN or the algorithm itself is implemented with neurons as computing elements. The latter method ('constructive ANN') has the benefit that the architecture is directly derived from the function to be implemented. However, requiring exact implementation will lead to very large networks in both cases. As an alternative, parts of the function can be implemented or approximated by ANNs. Such implementations can be done constructively or through training of suitable networks.

Appendix B.1 specifies our notion of layers and neurons and gives the details regarding a constructive implementation of the baseline NN algorithm. The number of layers is three, and there are altogether around 8,000 neurons and around 3.2 million coefficients. All of the coefficients are deduced constructively, and no training is needed. Although complex, it should be kept in mind that this network solves the NN problem in parallel thus giving a very high throughput. Any other implementation of the algorithm that performs the distance computations in parallel would be similarly complex. A valid question is thus if these computations can be partitioned to allow for groups of smaller ANNs to solve the same problem, either exactly or approximately. Naturally, the distance function can be divided up in smaller parts each containing a subset of the pixels. Although the number of coefficients per neuron will be lower, this does not lead to a reduction in the total complexity. However, as we will see in the next section, this idea can be further developed to yield an extension of the NN algorithm that allows for both better performance and efficient mappings into ANNs.

6 | THE WNN CLASSIFIER

There are many extensions of the NN algorithm. In Section 1, we mentioned the kNN method which reduces the dependence on a single image in A_{tr} or B_{tr} (which might be an outlier) by voting among the k-nearest images before a decision is made. In line with the traditional philosophy to include locality and the concept of weak learners and strong classifiers, we extend the baseline algorithm in the following way.

Let W be a window with centre in some pixel with coordinates (i_W, j_W) and side length of L_W pixels. Note that L_W is an odd number and it can be equal to 1, 3, 5, 7, ... In the case when the window W extends outside of the image, we set the value of the outside pixels to zero. Thus, as the image size is 28·28 pixels, there are 784 different windows with fixed side length L_W.

The distance between two images X and Y with respect to window W is defined as

$$dist_W(X,Y) = \sum_{(i,j)\in W} (X(i,j) - Y(i,j))^2.$$

Note that $(i, j) \in W$ means that pixel (i, j) belongs to the window W and $dist_W(X, Y)$ is equal to the Hamming distance as images are binary. We further define the distance from image X to the sets of training images A_{tr}, B_{tr} with respect to the window W as

$$\begin{aligned} dist_W(X,A_{tr}) &= \min_{\tilde{A}\in A_{tr}} (dist_W(X,\tilde{A})), \\ dist_W(X,B_{tr}) &= \min_{\tilde{B}\in B_{tr}} (dist_W(X,\tilde{B})). \end{aligned}$$

6.1 | Main observation

If X corresponds to an image of an object belonging to set A_{test}, then usually, $dist_W(X, A_{tr}) < dist_W(X, B_{tr})$, and if X corresponds to an image of an object belonging to the set B_{test}, then usually, $dist_W(X, B_{tr}) < dist_W(X, A_{tr})$. This is illustrated in Figure 2a,b for two example objects from the test sets.

This observation suggests that each window W can act as a weak learner and that a strong classifier can be formed through a (possibly weighted) sum of their outputs.

Here, we add all the outputs from the respective set to form

$$\begin{aligned} dist_\Sigma(X,A_{tr}) &= \sum_W dist_W(X,A_{tr}), \\ dist_\Sigma(X,B_{tr}) &= \sum_W dist_W(X,B_{tr}). \end{aligned}$$

So our algorithm (WNN) decides that X is from family A if

$$dist_\Sigma(X,A_{tr}) < dist_\Sigma(X,B_{tr})$$

and, similarly, that X is from family B if $dist_\Sigma(X, B_{tr}) < dist_\Sigma(X, A_{tr})$. Equivalently, the WNN algorithm decides that X is from family A if the function

$$\begin{aligned} F(X) &= dist_\Sigma(X,B_{tr}) - dist_\Sigma(X,A_{tr}) \\ &= \sum_W (dist_W(X,B_{tr}) - dist_W(X,A_{tr})) \end{aligned}$$

is positive and that X is from family B if it is negative (no solution if $F(X) = 0$).

6.2 | Performance of the WNN classifier

Experiments were done with different window sizes. It was found that, for our specific objects, a window size of $L_W = 5$ gave the best result. In the following, our experiments will be based on this window size. The WNN algorithm misclassifies the images in A_{test} in 16 cases and the images in B_{test} in nine cases. This corresponds to an error rate of 1.25%. Thus, the WNN method gives a substantial improvement over the baseline method which has an error rate of 3.2%. This improvement is attributed to the fact that the windows are positioned at every pixel location and that each window can 'vote' for its own nearest image candidate. More details about the tests of the WNN method are given in Appendix A.1, and the software implementation is found in the supporting information.

The improved algorithm can now be carried over to an ANN implementation as described next.

6.3 | Approximating WNN with ANNs

A constructive ANN implementation that directly implements the WNN method with window size of $L_W = 5$ would consist of three layers with approximately 6.3 million neurons and 82 million non-zero coefficients (see Appendix B.2 for details). As is seen, such an implementation becomes very complex. A relevant question is how to design one or several simpler ANNs that approximate the WNN functionality and how well such approximations would work. To address this question, we will focus on the function

$$F(X) = dist_\Sigma(X, B_{tr}) - dist_\Sigma(X, A_{tr}).$$

The WNN algorithm decides that X belongs to A_{test} if this function is positive and to B_{test} if this function is negative. As

$$\begin{aligned} F(X) &= \sum_W dist_W(X, B_{tr}) - \sum_W dist_W(X, A_{tr}) \\ &= \sum_W (dist_W(X, B_{tr}) - dist_W(X, A_{tr})), \end{aligned}$$

one approach is to approximate each term

$$F_W(X) = dist_W(X, B_{tr}) - dist_W(X, A_{tr})$$

by some rather simple ANN as shown in Figure 1.

The coefficients (weights and biases) of the ANN will be found by using the hitherto unused sets A_{val}, B_{val}, for the training. To do this for each X from the union of A_{val}, B_{val}, we consider the function $X \to F_W(X)$ and consider a regression ANN of low complexity. Based on a series of experiments, still using the same window size as earlier, it was found that an ANN consisting of one layer of 40 fully connected ReLUs followed by a fully connected linear neuron that produces the output (see the supporting information for details) was sufficient to obtain a good result. More complex ANNs did not improve the performance while simpler ANNs were significantly worse. Note that, with our choice of $L_W = 5$, $F_W(X)$ depends only on the values of X on 25 pixels from W. The resulting ANN for window W will be denoted by $F_{W,\ ANN}$. So the function $F(X)$ is approximated by the function

$$F_{ANN}(X) = \sum_W F_{W,\ ANN}(X).$$

This construction leads to the following algorithm:

Algorithm WNN-ANN *(windowed neural network algorithm).* If $F_{ANN}(X) > 0$, then X belongs to the family A, if $F_{ANN}(X) < 0$, then X belongs to the family B and if $F_{ANN}(X) = 0$, there is no solution.

Figure 2c,d shows the function $F_{W,\ ANN}(X)$ for the same two objects used in Figure 2a,b.

The complete ANN implementation thus adds the output from the smaller ANNs and applies a threshold to produce the classification output. In total, the whole network will consist of two layers, around 32,000 neurons and about 850,000 coefficients. Although there are many coefficients, each small ANN which includes 1081 coefficients is trained independently. As is seen, the complexity is radically reduced compared to the case when the distance function is not

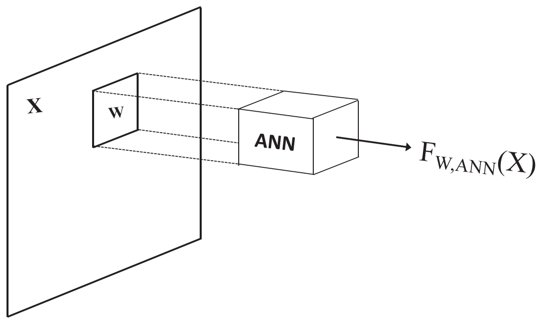

FIGURE 1 The function $F_W(X) = dist_W(X, B_{tr}) - dist_W(X, A_{tr})$ is approximated by an ANN

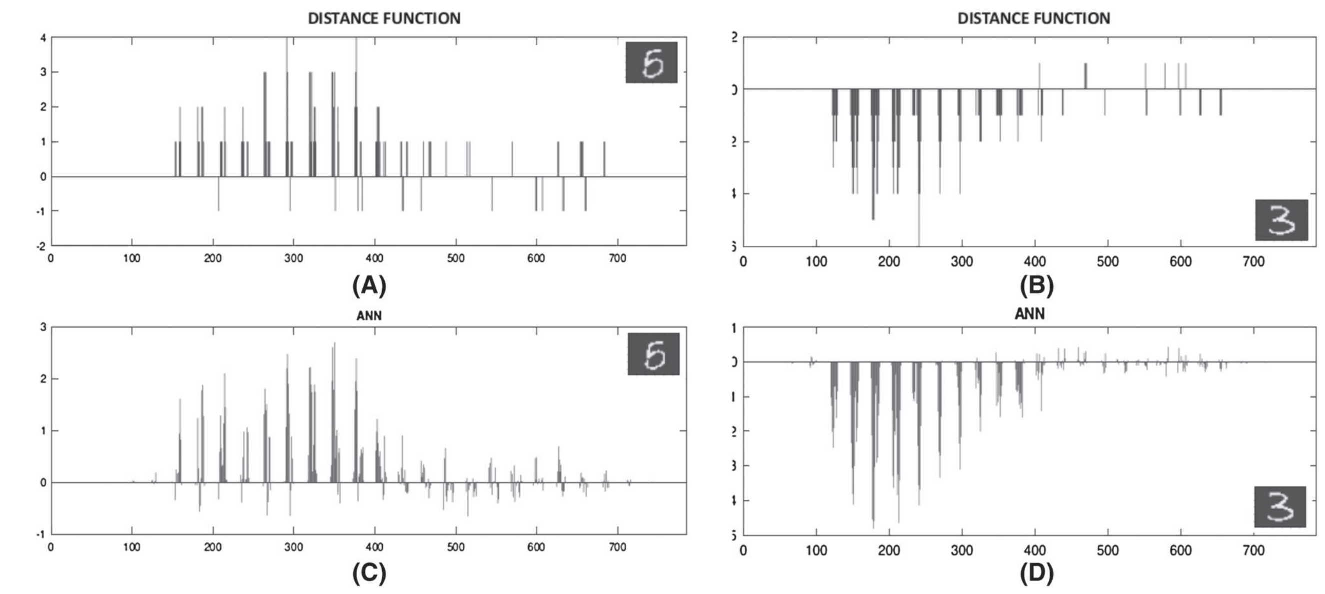

FIGURE 2 (a) Example evaluation of distance function $F_W(X)$ of object '5'. (b) The same for object '3'. (c) Example evaluation of ANN function $F_{W,\,ANN}(X)$ of object '5'. (d) The same for object '3'. Window index on horisontal axis. Insets: the objects '5' and '3'

approximated. An estimate of this gain can be obtained by comparing the number of operations a sequential computer would need in the two cases for each window. Computing the minimum distance between an input window and the training set requires $O(NL_W^2)$ operations, where N is the number of images in the training set and the operation amounts to the squared difference between pixel values followed by summation of these differences. With $L_W = 5$ and $N = 8000$, this gives approximately 200,000 operations. The corresponding ANN on the other hand requires $O(ML_W^2)$ operations (multiply and add) once it has been trained, where M is the number of neurons in the hidden layer. The exact number of such operations for $L_W = 5$ and $M = 40$ amounts to 1,081.

The performance of the ANN approximation when applied to the same test set as for the previous algorithms is given in Appendix A.2 (training set 1). From this, we see that 21 errors are generated for images in set A_{test}, and 8 errors are generated for images in set B_{test} giving a total error rate of 1.45%.

7 | IMPROVING THE PERFORMANCE

Assume that we want to investigate further improvements of the ANN classifier. As it is based on the WNN algorithm, we test the improvements first on the WNN system to understand the effects in performance and error behaviour. We then expect that the corresponding ANN version will behave similarly as it approximates the WNN version.

7.1 | Size of the training set

In the above algorithms, the training sets A_{tr}, B_{tr} consist of 4,000 images per set. We now investigate what happens with the errors when the size of the training set is substantially increased. This is done in the following way:

a) Each image from the original training set is shifted up to one pixel in each of eight directions, so from one image, we obtain nine images, and therefore, the training sets A_{tr}, B_{tr} will consist of 36,000 images. The WNN algorithm based on this new training set will be denoted 'Shift 1'.
b) The same as in (a) but shifts are done up to two pixels in each direction, and therefore, from each image, we will construct 25 images, and each training set A_{tr}, B_{tr} will now consists of 100,000 images. The WNN algorithm based on this training set will be denoted 'Shift 2'. It should be observed that the ANNs of the three generated algorithms only differ in the values of their coefficients due to the different training sets. The number of misclassified images from the initial algorithm and the two new WNN algorithms can be seen in Table 1.

WNN	# of errors A_{test}	# of errors B_{test}
No shift	16	9
Shift 1	17	6
Shift 2	13	6

TABLE 1 WNN errors

ANN	# of errors A_{test}	# of errors B_{test}
No shift	21	8
Shift 1	18	9
Shift 2	18	6

TABLE 2 WNN-ANN errors

In the case of no shift, we already reported 25 total errors. In the case of Shift 1 algorithm, we have 23 errors, and in the case of Shift 2 algorithm, we have 19 errors. The error rate thus drops from 1.25% to 1.15% and 0.95%, respectively, for the two latter cases.

If we approximate these three WNN algorithms by ANNs as described in Section 6.3, we will get misclassified images as described in Table 2.

As in the case of the WNN algorithms, we see some decrease in the number of total errors when we increase the size of the training set. The error rate drops from 1.45% to 1.35% and 1.2%, respectively, for the increased training sets.

It is expected that more data will reduce the error rate when training ANNs. The reason for this is not always easy to explain. In our case, we use ANNs to estimate the nearest distance to a set. For such regression estimates, the parameters will be better estimated when there is more data.

7.2 | Replacing the function $F_W(X)$

Is it possible to replace the distance function in some way when training the ANNs? Experiments show that the quantised function

$$F_W(X) = \begin{cases} 0 & \text{if} \max(dist_W(X,A_{tr}), dist_W(X,B_{tr})) = 0 \\ 1 & \text{if } X \in A_{val}, \max(dist_W(X,A_{tr}), dist_W(X,B_{tr})) > 0 \\ -1 & \text{if } X \in B_{val}, \max(dist_W(X,A_{tr}), dist_W(X,B_{tr})) > 0 \end{cases}$$

can be used. This approach can be seen to be very much in line with the concept of weak vs. strong classifiers. In fact, despite the coarse quantisation, the performance will be better (error rate of 1.25%) than the initial WNN-ANN algorithm. We think that the important property here is the following: $F_W(X) = 0$ if $dist_W(X,B_{tr}) = dist_W(X,A_{tr}) = 0$.

7.3 | Weighted windows

As indicated in Section 6, an extension to both WNN and WNN-ANN would be to add the outputs corresponding to the windows with different weights before forming the functions $F(X)$ and $F_{ANN}(X)$. The weights could be based on the importance of each window with respect to the final classification result. Despite numerous tests, it has turned out that the gain when applied to our problem is negligible in both algorithms compared to using equal weights.

A related issue is whether nonlinear combinations can improve the performance. This may be the case but has not been investigated. Despite the problem of finding suitable nonlinear combinations, the interpretability may get lost as such operations will make the method deviate further from the traditional NN principle.

8 | STABILITY OF WNN TO CHANGES OF TRAINING DATA

The advantage with basing ANN solutions on traditional methods is that the stability to changes in training data as well as other performance metrics can more easily be analysed. To check the stability of the WNN to different training data, we exchanged part of A_{tr} and B_{tr} with the validation data as described in Appendix A.1. From this, we can conclude that the WNN algorithm is quite stable to such changes. This also carries over to the approximative ANN solution as is shown in Appendix A.2.

8.1 | Error probability and prediction confidence

As is further seen from the tests described in Appendix A.1, the error probability of the classification can be estimated giving an indication of the prediction confidence. If the same images are always misclassified regardless of the training set, it indicates that these images are indeed difficult to handle. The performance of the classifier in such a case is robust against change of training set. The appendix describes such an experiment where 15 and 25 different versions of the training set are used. It is shown that 23 images are consistently misclassified, which is a fairly large part of the total misclassified images.

9 | DISCUSSION

It can be argued that the NN and WNN algorithms are unjustly treated as they are based on 8,000 annotated images while the various versions of WNN-ANN uses 10,000 annotated images (taking the validation set into account). For completeness, we therefore also state the following results. For the NN case, the error rate drops from 3.2% to 2.95% and for WNN the error rate drops from 1.25% to 1.15% when the training data for these algorithms is extended to also incorporate the validation set. This does not change the relative ratings between the methods as NN and WNN were already better than the ANN approximations even when using a smaller training set.

The reason to set aside the validation data was to use it in the approximation of the function

$$F_W(X) = dist_W(X, B_{tr}) - dist_W(X, A_{tr})$$

on the set $A_{val} \cup B_{val}$.

A relevant question is whether the validation set is really necessary. Can't we use the training set for this purpose? In fact, it turns out that such an approach works as well. A drawback though is that we would then not have enough data available to perform the reported stability check.

Another remark concerns the applicability to more complex problems. Our example was very simple and only intended to illustrate the main idea. We choose to use the MNIST dataset and the NN method as this dataset and the traditional method are well known in the ML community. Moreover, there is an interest to approximate the NN search.[17] The method that we have used is most likely only applicable when the two classes contain objects of the same type (like digits). If one of the classes were to contain any other type of object, it is not obvious that our approach would work. We believe however that the principle of combining several stable approximations is a useful approach in general. How to find the functions to approximate will depend strongly on the underlying traditional method. As mentioned in Section 1, inspiration should come from the most recent traditional techniques; e.g., the cascade idea described in Viola and Jones[15] seems to be useful for the purpose.

Mathematically, our technique can be viewed as adjusting the theory of local approximations to the problem of classification. It can be noted that local approximation looks similar to convolution layers in ANNs. In mathematics, local approximations are used to describe spaces of differential functions by approximation (in L^p norm) by polynomials of fixed degree on subcubes.[18]

In our approach, instead of subcubes, we use windows of fixed size 5·5 pixels. Instead of polynomials of fixed degree, we consider restrictions of training images of a given digit on the window (so in the case of classification for two different digits, we have two approximation subsets on each window). Instead of finding the best p for the L^p norm, we work with image binarisation, and therefore, p is not important, and we can set $p = 2$.

TABLE 3 Summary of error rates

	Baseline	Window	Shift-1	Shift-2	Quant
Distance	3.2%	1.25%	1.15%	0.95%	
ANN approx		1.45%	1.35%	1.2%	1.25%

An important part of the approach is the stability of the approximations. By stability we mean that the set of errors (in the test set) and the probability of error of a concrete image do not change significantly when we change training and validation sets.

Although our obtained results indicate that the approach seems to work, we have not been able to prove this in a statistical sense. The MNIST dataset is rather limited. We have utilised it as best as we can through shifts and permutations to achieve different training and test sets. Since the number of errors is small, and the data are not random, the number of different runs is not sufficient to allow for estimating confidence intervals.

As a final note, we repeat that the aim with our approach is not to arrive at a classifier with the lowest possible error rate but to investigate a bridge between a traditional method and ANN technology for image classification and to understand what makes the described combination successful. The reader is probably curious about what performance would be achieved by a general ANN applied to our simple classification problem. This depends on the chosen complexity of the ANN, but as a point of reference, we trained a convolutional network with three layers, around 3,900 neurons and around 4,000 coefficients, and achieved an error rate ranging from 1.1% to 1.4 % (see the supporting information for implementation details).

10 | SUMMARY

ANN is a very remarkable technology. However, it has important drawbacks: it is unclear how to find the architecture of an ANN for a given concrete problem. Furthermore, as the ANN can be considered as a black box, the reasons for the errors that occur are not clear, and it is unclear what to do to eliminate them. These problems might probably be solved if there were a convincing mathematical theory which could also explain the success of ANNs. Unfortunately, until now, all attempts to construct such a mathematical theory have failed. In this paper, we suggest going in the opposite direction: first, use an approach which has a clear mathematical background, and then use an ANN as an approximation tool to make the solution more efficient, taking advantage of the fast development of the ANN technology with regard to processing speed, its parallel architecture and implementation in hardware.

We have illustrated our philosophy through a specific example, namely, the NN classifier which we have used to distinguish between two digits in the MNIST image database. The baseline classifier was then extended to a windowed version and further improved by extension of the training set and quantisation of the distance function. For all these cases, we discussed how to map the algorithms into ANNs. By allowing approximations, we obtain a substantial decrease in complexity of the ANNs. Instead of computing the exact distance between the unknown input object and each member of the training sets, we have proposed to train small ANNs to estimate the minimum distance to the entire sets. Since individual and small ANNs are used for each window, this procedure becomes very efficient. The results are ANN-based solutions that behave performance-wise very similarly to the underlying traditional algorithms except that they are able to execute much faster. Our experiments show that the suggested method gives quite good results and is rather stable (the set of error images does not change much when we change training set for the same test set).

Table 3 summarises the error rates of the studied cases. Here, 'distance' represents the NN and WNN algorithms, while 'ANN approx' represents the WNN-ANN algoritms. 'Shift 1' and 'Shift 2' designate the extended training sets described in Section 7.1. 'Quant' designates the result of the quantised version of $F_W(X)$ described in Section 7.2.

A study of the misclassified objects shows that these are essentially the same both for the original algorithm and the ANN-based version. This shows that the approximation used in the ANN case does not substantially deviate in behaviour from the original scheme. This appears to be true also when the training set is augmented in various ways.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Eric Setterqvist and Dr. Jörgen Ahlberg for valuable comments. The comments from the anonymous reviewers have improved the paper and are gratefully acknowledged.

ORCID

Robert Forchheimer https://orcid.org/0000-0002-8382-2725

REFERENCES

1. LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521(7553):436-444.
2. Lowe DG. Object recognition from local scale-invariant features. In: International Conference on Computer Vision (ICCV); 1999. https://doi.org/10.1109/ICCV.1999.790410
3. Bay H, Ess A, Tuytelaars T, Van Gool L. SURF: Speeded Up Robust Features. *Comput Vision Image Underst (CVIU)*. 2008;110(3): 346-359. https://doi.org/10.1016/j.cviu.2007.09.014
4. Bishop CM. *Pattern Recognition and Machine Learning*. New York: Springer-Verlag; 2006.
5. Hornik K, Maxwell B., Stinchcombe M, Halbert W. Multilayer feedforward networks are universal approximators. *Neural Netw*. 1989;2: 359-366.
6. LeCun Y, Cortes C, Burges C. The MNIST database. https://yann.lecun.com/exdb/mnist/
7. Chen YQ, Damper RI, Nixon MS. On neural-network implementations of k-nearest neighbor pattern classifiers. *IEEE Trans Circ Syst I: Fundam Theory Appl*. 1997;44(7):622-629.
8. Murphy OJ. Nearest neighbor pattern classification perceptrons. *Proc IEEE*. 1990;78(10):1595-1598.
9. Park YH, Bang SY. A new neural network model based on nearest neighbor classifier. In: [Proceedings] 1991 IEEE International Joint Conference on Neural Networks, Vol. 3; 1991:386-2389.
10. Yang Y, Morillo IG, Hospedales TM. Deep Neural Decision Trees; 2018.
11. Papernot N, McDaniel PD. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning, abs/1803.04765. CoRR, https://arxiv.org/abs/1803.04765; 2018.
12. Lee R, Clarke J, Agogino A, Giannakopoulou D. Improving trust in deep neural networks with nearest neighbors. In: AIAA Scitech 2020 Forum; 2020; Orlando, Fl. https://doi.org/10.2514/6.2020-2098
13. Boz O. Extracting decision trees from trained neural networks. In: 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2002:456-461.
14. Breiman L. Random Forests. *Machine Learning*, Vol. 45: Kluwer Academic Publishers; 2001:5-32.
15. Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Vol. 1; 2001:I-I.
16. Ahlberg J, A˚ström A, Forchheimer R. Simultaneous sensing, readout, and classification on an intensity-ranking image sensor. *Int J Circ Theory Appl*. 2018;46(9):1606-1619.
17. Wang X, Chen J, Yu J. Optimised quantisation method for approximate nearest neighbour search. *Electron Lett*. 2017;53(3):156-158.
18. Brudnyi Y. Spaces defined by means of local approximations. *Trans Moscow Math Soc*. 1974;24:73-139.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Kruglyak N, Forchheimer R. Design of classifiers based on ANN approximations of traditional methods. *Int J Circ Theor Appl*. 2021;1–16. https://doi.org/10.1002/cta.2998

APPENDIX A: Simulation Results

A.1 | WNN performance

All simulations have been performed using MATLAB and its Deep Learning Toolbox for the ANN parts. Based on 6,000 labelled examples from family A and 6,000 labelled examples from family B, we set aside the labelled examples from 5,001 to 6,000 for the sets A_{test},B_{test} (so we have 1,000 images in each set and enumerate them as 1, 2, ..., 1,000). Moreover, from the first 5,000 images, we choose sets A_{tr}, B_{tr} (each of size 4,000), and the remaining part, we denote by A_{val}, B_{val} (each of size 1,000). Note that for different training sets A_{tr}, B_{tr}, we will get different WNN algorithms. For a given WNN algorithm, an image X from A_{test} is an error image if $F(X) \leq 0$, and similarly, an image X from B_{test} is an error image if $F(X) \geq 0$. The reasons for the errors could be errors in labelling and/or failures of the algorithm. To get a sense of how the errors change when we change sets A_{tr}, B_{tr}, we consider the following five WNN algorithms. In the first algorithm A_{tr}, B_{tr}, each consists of the first 4,000 labelled images, i.e., A_{val}, B_{val} consist of images with numbers from 4,001 to 5,000 from each class; in the second case, A_{val}, B_{val} consist of images numbered from 3,001 to 4,000 while the training sets each include the remaining 4,000 images. In the third algorithm, the validation sets consist of images with numbers from 2,001 to 3,000; in the fourth, from 1,001 to 2,000 and in the fifth, from 1 to 1,000. Tables A1 and A2 show errors images in the test sets for these five WNN algorithms (in the column ER we give the total number of errors).

From these tables, we see that the set of errors is rather stable; for example, the following 11 images from the set A_{test} with numbers 14, 31, 33, 63, 190, 400, 401, 402, 547, 659 and 961 and the five images 87, 315, 341, 482 and 638 from the set B_{test} are error images for all five algorithms. Other error images (there are 10 in the set A_{test} and 8 in the set B_{test}) appear in some but not all of the five algorithms.

To get a better understanding what happens with errors when we change the sets A_{tr}, B_{tr}, we introduce the notion of error probability. Suppose that we have n different sets A_{tr}, B_{tr} (sets A_{test}, B_{test} are fixed). Suppose further that image X from the test sets appears as an error image for m algorithms, then, by error probability of X, we use the number $\frac{m}{n}$. Note that when this value is small, we are more confident in the prediction of the WNN algorithm.

To see how the error probability changes when we increase the number of WNN algorithms used, we first add 10 randomly chosen sets A_{tr}, B_{tr} each of 4,000 images to now give 15 different algorithms. If we add an additional 10 randomly chosen sets A_{tr}, B_{tr} each of 4,000 images, we will have 25 different algorithms. Table A3 contains the error probability for those images that generate errors.

Analysing the above tables, we see that there are no large changes when we increase the number of WNN algorithms. Images from the set A_{test} with error probability more than 0.5 are shown in Figure A1, similarly for the set B_{test} in Figure A2.

TABLE A1 Set A errors when using different training sets (WNN)

Algorithm	ER	Error images in the set A_{test}
Training set 1	16	14, 31, 33, 63, 110, 190, 400, 401, 402, 465, 547, 631, 659, 683, 782 and 961
Training set 2	13	14, 31, 33, 63, 190, 400, 401, 402, 547, 659, 683, 961 and 965
Training set 3	14	14, 31, 33, 63, 110, 190, 399, 400, 401, 402, 547, 631, 659 and 961
Training set 4	17	14, 31, 33, 63, 86, 110, 190, 400, 401, 402, 465, 547, 631, 659, 683, 745 and 961
Training set 5	14	14, 31, 33, 63, 110, 190, 400, 401, 402, 454, 547, 659, 683 and 961

TABLE A2 Set B errors when using different training sets (WNN)

Algorithm	ER	Error images in the set B_{test}
Training set 1	9	26, 87, 177, 227, 233, 315, 341, 482 and 638
Training set 2	7	26, 87, 315, 341, 482, 606 and 638
Training set 3	11	87, 152, 177, 233, 315, 341, 464, 482, 606, 638 and 927
Training set 4	10	26, 87, 152, 177, 227, 233, 315, 341, 482 and 638
Training set 5	9	26, 87, 177, 227, 315, 341, 482, 606 and 638

TABLE A3 WNN error probability

Image number (family A)	Error prob. (15 algor.)	Error prob. (25 algor.)
14	1	1
31	1	1
63	1	1
400	1	1
401	1	1
402	1	1
33	1	0.96
547	0.93	0.96
659	1	0.96
961	0.93	0.96
190	0.8	0.84
110	0.67	0.76
683	0.73	0.76
631	0.6	0.52
465	0.4	0.32
86	0.27	0.28
454	0.33	0.28
782	0.27	0.2
832	0.27	0.2
399	0.2	0.16
745	0.2	0.16
576	0.13	0.12
204	0.67	0.08
688	0.07	0.04
Image number (family B)	**Error prob. (15 algor.)**	**Error prob. (25 algor.)**
315	1	1
341	1	1
638	1	1
482	0.93	0.96
26	0.87	0.88
87	0.87	0.84
177	0.87	0.84
233	0.8	0.8
227	0.73	0.68
606	0.53	0.48
152	0.33	0.24
927	0.13	0.12
39	0.07	0.04
327	0	0.04
405	0.07	0.04
464	0.07	0.04
625	0	0.04

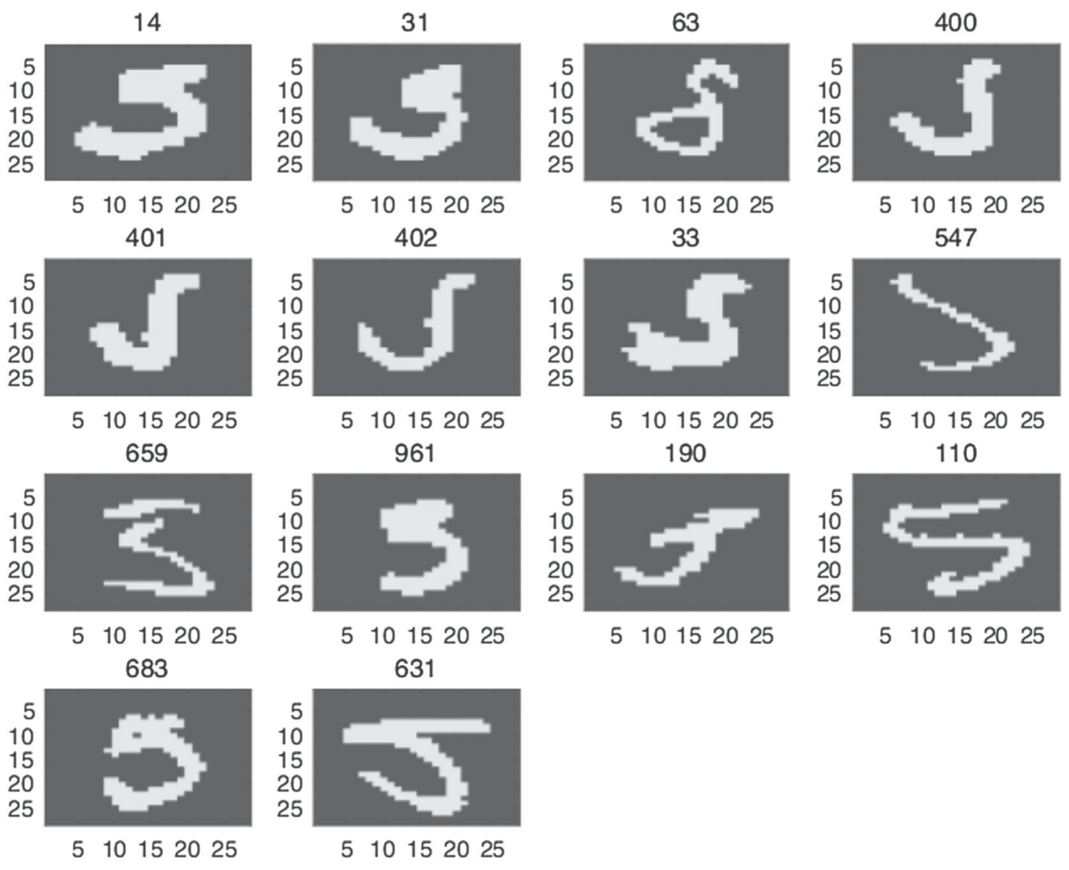

FIGURE A1 Images of ‘5’ with error probability larger than 0.5

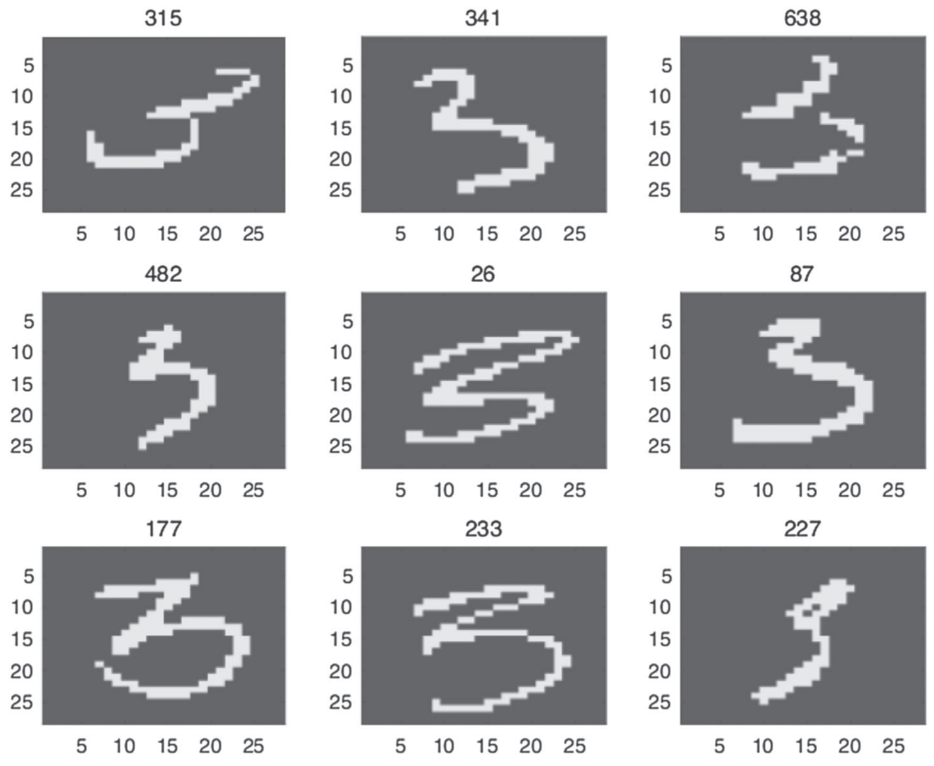

FIGURE A2 Images of ‘3’ with error probability larger than 0.5

TABLE A4 Errors from set A when using different training sets (WNN-ANN)

Algorithm	ER	Error images in the set A_{test}
Training set 1	21	14, 31, 33, 63, 86, 110, 190, 237, 310, 400, 401, 402, 454, 547, 631, 683, 688, 782, 818, 832 and 961
Training set 2	18	14, 31, 33, 63, 86, 119, 190, 310, 399, 400, 401, 402, 454, 547, 659, 683, 688, 832 and 961
Training set 3	17	14, 31, 33, 63, 86, 110, 190, 310, 400, 401, 402, 454, 547, 683, 688, 832 and 961
Training set 4	19	14, 31, 33, 63, 86, 190, 310, 399, 400, 401, 402, 454, 465, 547, 631, 683, 688, 832 and 961
Training set 5	18	14, 31, 63, 86, 110, 190, 310, 399, 400, 401, 402, 454, 547, 631, 683, 688, 832 and 961

TABLE A5 Errors from set B when using different training sets (WNN-ANN)

Algorithm	ER	Error images in the set B_{test}
Training set 1	8	26, 87, 152, 233, 315, 341, 482 and 638
Training set 2	9	26, 87, 177, 227, 233, 315, 341, 482 and 638
Training set 3	13	26, 87, 152, 177, 215, 227, 233, 315, 327, 341, 482, 638 and 713
Training set 4	11	26, 87, 177, 227, 233, 315, 327, 341, 482, 516 and 638
Training set 5	10	26, 87, 227, 233, 315, 317, 341, 482, 638 and 713

A.2 | WNN-ANN performance

Here, we give results when approximating the WNN by ANN as presented in Section 6.3.

We use training sets as in Tables A1 and A2. Again, as is seen from Tables A4 and A5, the result is rather stable.

Remark 1. From these tables, it is seen that about 50% of the errors appear in all cases while 80% of the errors are found in the Probability of Errors tables of the previous section.

APPENDIX B: Constructive ANN Implementations

B.1 | Constructive ANN implementation of baseline NN

Throughout this paper, we consider feed-forward ANNs and describe them in terms of number of layers, number of neurons and number of coefficients. As there are different notations when it comes to the concept of ‘layers’ in the ANN literature, we use the following definition influenced by a hardware view. A layer consists of one (artificial) neuron or several neurons acting in parallel. These neurons can be either linear or nonlinear (e.g., *ReLUs* and *max pooling*) or perform a joint layer operation such as *softmax*. A neuron performing a linear operation followed by a non-linear activation function is considered a single neuron. The input signals are not included in the layer count.

Translating the NN method into an ANN architecture leads to a fully parallel implementation. The network will take an image X of 784 pixels as input. As a neuron is able to compute a scalar product plus an offset, we will use this property instead of using the Euclidean distance ρ. Since $\rho^2 = (X-Y)^T(X-Y) = X^TX + Y^TY - 2X^TY$, finding the training image Y (viewed as a vector) that minimises the distance to X is equivalent to maximising the scalar product between X and Y minus half of the value of Y^TY.

For each training image, the scalar product can thus be computed by a single neuron with a maximum of 784 inputs with weights equal to the pixel values of the training image. The first layer computes the scalar products of X with all images in the two training sets A_{tr}, B_{tr} and also applies the biases Y^TY which are specific for each training image. This will give rise to $2{*}4{,}000$ values. The neurons in the following layers need to find out which of the 4,000 scalar product values for each class is the largest. Within each class, it is only necessary to find the largest value without having to keep track of which particular image is associated with this value. Assuming the availability of *max pooling* neurons, only two of them are needed in the next layer.

Finally, the maximum value from each class is inserted into the last layer consisting of a single neuron that checks which one is largest and outputs the classification result using a thresholding activation function.

In summary, the network will contain three layers with altogether 8,003 neurons. The number of coefficients will depend on the number of set pixels in the training images. Assuming that half of the pixels are set on average, there will be around 3.2 million coefficients (most all of them having the value 2).

B.2 | Constructive ANN implementation of WNN

The design principle is similar to the NN case. The main difference is now that a separate scalar product is computed for each of the 784 windows. This means that altogether, 784∗8,000 = 6,272,000 values need to be handled by the next layers. For each window, and for each training image, the scalar value can be computed with a neuron with a maximum of 25 inputs. Finding the maximum scalar values is done in the same way as for the NN implementation (a *max pooling* layer). The following layer consists of 784 neurons that will generate the weak classifier output for each window. Finally, these outputs are added (or weighted) into the last layer consisting of a single neuron to yield the final (strong) classification result.

In summary, the network will contain three layers with altogether approximately 6.3 million neurons and about 164 million coefficients out of which around half are non-zero.