My solution to the task is considerably different from the suggested algorithm, since in fact that algorithm is not directly implementable. Below I explain why that is, and what workarounds I implemented to solve this problem.

Leading a hyperplane through a set of points

The algorithm description mentions leading a separation hyperplane through a set of patterns. Usually, this algebraic problem can be reduced to solving a system of linear equations (as each of the patterns has to satisfy the hyperplane equation), which in turn can be done by Gaussian elimination or inversion of a matrix created by arranging the pattern vectors (flattened images) row-wise.

In fact, I did spend a few days trying to invert a matrix in C++, which incidentally is unbelievably troublesome for such a basic algebraic task. In any case, I eventually realized that finding the hyperplane by directly solving a system of linear equations was impossible in this case. In the images, there are too many white pixels, represented as zeroes. This results in indeterminate systems. In fact, for every matrix representation created while trying to lead a hyperplane, I checked that there was at least one column containing only zeroes. Let us call such columns 'zeroed columns'.

Let us define the task as finding a hyperplane (a coefficients vector c and constant term t) passing through a set P of the N closest patterns (images) to a target pattern having different labels than that pattern, with M being a matrix corresponding to that set (containing images from the set, row-wise). With the assumption that M has at least one zeroed column, any hyperplane passing through the patterns has the following properties: $t = 0$, the i-th feature of the coefficients vector is equal to zero if the i-th column of M is zeroed, and otherwise contains an arbitrary number. I thought about two solutions for what to do next. It would have been possible to slightly perturb the image vectors so that leading a hyperplane through the perturbed vectors is a well-defined task (linear system is determinate), and the resulting hyperplane passes reasonably close to the original vectors. Alternatively, one could heuristically choose the arbitrary numbers in the indeterminate system, so that the resulting hyperplane can reasonably be expected to separate patterns of our target label from other patterns.

I decided to follow the latter approach. I first tried to assign random values from the range (-1, 1). While this, in theory, was already an implementation of the algorithm we were supposed to implement, it resulted in very poor performance. Granted that recall was equal to perfectly 1, but precision was about 0.1, giving an F1 score of 0.18 with the hyperplane selection method as described in the original algorithm (see the section below). I decided to improve this performance.

The Professor suggested normalizing gray pixels in the images into the range (-1, 1) and then choosing a hyperplane with the maximum distance to the target pattern. Algebraically, this means that the i-th feature of the coefficients vector (as long as it belongs to the features we are supposed to choose arbitrarily) should be equal to the i-th feature of the target pattern. This, however, also creates a problem, as most of these features are also equal to 0, which means that we end up with a hyperplane whose all coefficients are equal to 0. This is obviously useless, so something else has to be done in this case.

For this case, then, I decided to change the heuristic altogether and lead a hyperplane *above* the patterns from the set, but below the target pattern. In other words, we want the dot product of the coefficients vector and a vector v to be low for v representing a pattern from the set, and high for v representing the target. Then it will be easy to choose a constant term that separates the set patterns from the target well. A following heuristic can be applied: each feature is chosen separately. For the i-th feature c_i of the coefficients vector c, let us denote as m_i the mean of the values of the i-th features of the patterns from the set, and as t_i - the corresponding feature of the target. Let c_i be equal to $t_i - m_i$. Intuitively, this number is high if the i-th feature tends to be low for the set patterns, but is high for the target

pattern. Therefore, the i-th feature strongly suggests the label of the target pattern rather than that of the set patterns. Analogous properties hold for other cases, for example if the number is small, it represents a slight suggestion for or against the label. Having chosen the coefficients vector, we should choose the constant term so as to separate the set patterns and target pattern. We can calculate the mean of the dot products of c with the set patterns, and denote it as d_s, and the dot product of c with the target pattern, denoting it as d_t. Then the term is chosen to be $d_s + 0.1 \cdot (d_t - d_s)$. The number 0.1 is arbitrary - it could reasonably be anything between 0 and 0.5. I tested it versus 0.2 and it had better performance, so I concluded any higher value would have an even worse performance. Eventually, this increased precision to 0.67, recall to 0.63, and F1 to 0.65. This was still not satisfactory, so I tried to improve the algorithm of choosing the separating hyperplanes.

Choosing the separating hyperplanes

The original algorithm involves choosing separating hyperplanes with the highest number of true positives (after creating a separating hyperplane for each zero-pattern). However, as no penalty is included for false positives, I observed that this resulted in choosing hyperplanes which just recognize a high number of patterns, with false positive counts usually being higher than true positive counts. Therefore I changed the metric to be maximized from tp to

$$\frac{tp \cdot tp}{fp}$$

While this did result in good quality neurons with high accuracy, it also meant the selection was quite slow, often choosing neurons with tp around 10 and $fp = 1$ (not to mention that I forgot about the division by zero problem). This meant that each iteration for choosing a next separating hyperplane only removed a few zero-patterns (and the algorithm was supposed to be terminated after all 10000 patterns have been removed). I stopped the program after it had only eliminated 2200 zero-patterns in 24 hours, and replaced the metric with

$$\frac{tp \cdot tp}{fp + 10}$$

giving preference to hyperplanes with both high accuracy and tp, which meant that each iteration removed more zero-patterns. This reduced execution time to about 13 hours.

Here I would like to note that these execution times only include selecting the separating hyperplanes, not creating the hyperplanes for each zero-pattern. That step can be precomputed, and results saved to or loaded from an external file. It took about 15 hours.

Eventually, this selection algorithm improved the results to: precision 0.886, recall 0.764, F1 0.821. 114 neurons were selected.

Choosing the behavior of the final neuron

The original algorithm does not specify what kind of function the final neuron should perform on the binary outputs of the hidden layer. I decided on a simple sum function, followed by a comparison to a treshold - the pattern is recognized if the sum exceeds the treshold. This still gives some flexibility in choosing the treshold. While training, the final neurons simply tries out all possible tresholds (from 1 to the number of hidden neurons), and chooses the one which gives the highest $F1$ score. In the final run, for 114 neurons, this treshold was actually equal to just 3.