university of groningen / faculty of mathematics and natural sciences / artificial intelligence

The Effect of a Stacked MLP Trained on Output Probabilities in a Patch-Based Image Classification System

Bachelor's Project Thesis

Lennart Adriaan van de Guchte, s2408287, l.a.van.de.guchte@student.rug.nl,

Supervisor: Dr M.A. Wiering

Abstract: This thesis presents a patch-based image classification system in combination with an ensemble technique named stacking. In the standard model an image is reshaped into a collection of patches, from these patches feature vectors are extracted using the Histogram of Oriented Gradients (HOG). Then a simple 3-layered multilayer perceptron (MLP) is used for classification. Classification is done by applying the sum voting technique, here the output probabilities of each patch in an image are summed up. The image is then assigned to the class with the highest sum of probabilities. During the training of this MLP dropout is applied to prevent the network from overfitting. In the ensemble model an extra MLP is stacked after the usual MLP. Here the maximum, average or minimum output probabilities from all patches in an image are computed and serve as input values for the stacked MLP. Additionally, we varied with a pooling method that splits an image into quadrants. The patches of each quadrant are then used to train different MLPs, the output probabilities of these MLPs serve as input values for the stacked MLP. To measure the accuracy of these models the two widely known image recognition benchmarks MNIST (handwritten digits) and CIFAR-10 (objects) are used for experimentation. The standard model shows good performances on the MNIST dataset (99.45% correctly classified images). The addition of stacking and/or pooling results in a decrease in accuracy of 0.20-1.63%. On CIFAR-10 the standard model also outperformed all other models (68.30%) with a decrease of 1.17-12.97% when stacking and/or pooling is used.

1 Introduction

In the last few years several machine learning techniques have resulted in outstanding accuracy using artificial neural networks. Especially deep neural networks (Schmidhuber, 2015) have proven to be very useful in image recognition, speech recognition and many other domains. One disadvantage of this technique is the high amount of computational power that is needed to train these networks. Although a big decrease in training time can be achieved by using a graphics processing unit or GPU (Krizhevsky, Sutskever, and Hinton, 2012), it still will be useful in terms of efficiency and costs to get similar results with less computational power.

Previous work by Simard, Steinkraus, and Platt (2003) has shown that a simple fully connected 3-layer perceptron with 800 hidden units achieved remarkably good results (0.70 % error) on the widely known MNIST dataset for handwritten digits (LeCun and Cortes, 1998). However, the best result on this dataset (0.21 % error) is achieved by using deep neural networks in combination with DropConnect (Wan, Zeiler, Zhang, LeCun, and Fergus, 2013).

In this research we try to optimize the accuracy of a simple multilayer perceptron (MLP) when integrating it into a patch-based image classification system. Here an image is reshaped into a collection of patches. From all patches feature vectors will be extracted using a feature descriptor called the Histogram of Oriented Gradients (HOG; Dalal and Triggs (2005)). These feature vectors serve as input values for the MLP, which is trained by using back-propagation. Dropout (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014) is applied to prevent the network from overfitting. Finally each image is classified to the class with the maximum sum of output probabilities over

all patches.

In an attempt to optimize this standard model we studied the effect of stacking an MLP on top of another MLP by using the maximum output probabilities of an image. These maximum output probabilities are computed by taking the highest output probability per class from all patches in an image. The use of minimum and average output probabilities will also be evaluated. These output probabilities then serve as input values for the stacked MLP. A variation on these models is studied by using a pooling algorithm to split an image into four quadrants. The patches of each quadrant are then used to train an MLP, this means that four different MLPs are trained. When combining pooling and stacking the output probabilities of all MLPs serve as input values for the stacked MLP. This results in a comparison of four different models: standard model, pooled model, stacked model and the combined model (pooled and stacked). Experiments are done on two widely known image classification datsets: the MNIST dataset for handwritten digits and the CIFAR-10 dataset for object classification (Krizhevsky, 2009).

Thesis outline: Section 2 describes the whole model that is used for experimentation. Section 3 gives a description about the datasets and parameters that are used. The experimental results and subsequent analysis are shown in Section 4. In the last section a summarizing conclusion is given.

2 Model Description

The following descriptions are placed in the same order as how an image is processed through the system. There is varied with the use of pooling and stacking, therefore these additional methods can be disregarded in the standard model with one MLP. An illustration of the system is given in Figure 2.1.

2.1 Patches

All images are divided into partly overlapping subimages, called patches. This is done by applying a sliding window that can vary with height, width and stride. The stride parameter describes the amount of pixels that are skipped before extracting the next patch. During training a fixed number of patches are randomly selected to train the network while for classification all patches are used.

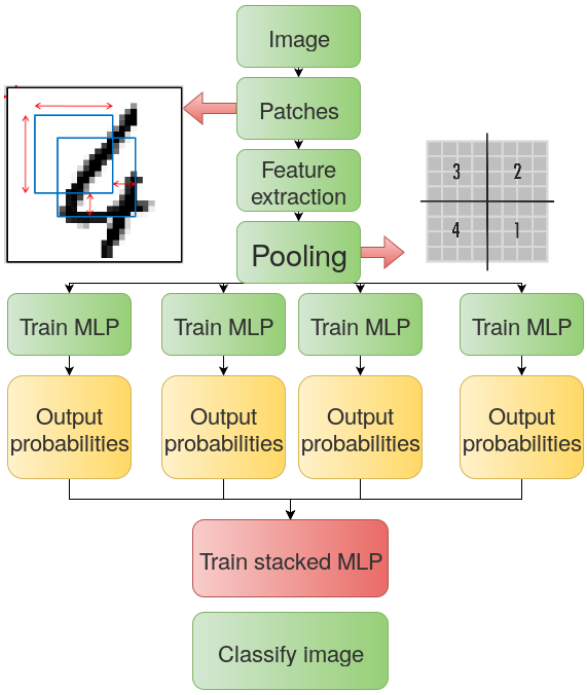

Figure 2.1: System overview with stacking and pooling.

2.2 Feature Extraction

To construct a feature vector from each patch two feature extraction techniques are studied, the image pixel-based method (IMG; Surinta, Schomaker, and Wiering (2013)) and the histogram of oriented gradients (HOG; Dalal and Triggs (2005)). The first method uses the raw pixel intensities of a patch to construct a feature vector. However, during experimentation the HOG descriptor outperformed IMG so only the HOG feature extractor is used.

2.2.1 Histogram of Oriented Gradients

First, the HOG descriptor divides a patch into smaller parts, so called 'blocks'. These blocks are represented by the gradient orientation and gradient magnitude (intensity of change). Then the gradient components Gx and Gy are calculated by applying a simple kernel $[-1, 0, 1]$ for horizontal re-

spectively $[-1, 0, 1]^T$ for vertical components:

$$Gx = f(x+1, y) - f(x-1, y) \tag{2.1}$$

$$Gy = f(x, y+1) - f(x, y-1) \tag{2.2}$$

Where $f(x, y)$ is the pixel intensity at coordinate x, y. With these components the gradient orientation $\theta(x, y)$ and gradient magnitude $M(x, y)$ can be calculated using equations 2.3 and 2.4 (Surinta, Karaaba, Mishra, Schomaker, and Wiering, 2015).

$$\theta(x, y) = tan^{-1}\frac{G_y}{G_x} \tag{2.3}$$

$$M(x, y) = \sqrt{G_x^2 + G_y^2} \tag{2.4}$$

After this a histogram is constructed by assigning the gradient orientation to a specific bin. The histogram consist of 9 bins in a range between $[0; \pi]$. If the gradient orientation is between $[\pi; 2\pi]$ it will be subtracted from 2π to mirror its orientation. From the pixels in a block that are in the domain of the bin the gradient magnitude is added to the histogram: this is done for each block. These histograms are combined into a one dimensional feature vector which is then normalized using the L2-norm. Finally, this vector is standardized by subtracting the mean of the feature vector from each value and dividing it by the standard deviation:

$$x_i' = \frac{x_i - \bar{x}}{\sigma} \tag{2.5}$$

2.3 Pooling

By pooling, the image is divided into four equal quadrants. The center point of each patch is calculated and from there each patch is assigned to a specific quadrant. Finally, the patches of each quadrant are used to train an MLP, resulting in four different MLPs.

2.4 Multi-layer Perceptron

A 3-layered fully connected MLP with back-propagation, as originally devised by Rumelhart, Hinton, and Williams (1988), is used to classify the images. To train this MLP a random sample of all feature vectors serve as input vector for the MLP. Before training the weights have been randomly initialized between $[-0.5; 0.5]$. A bias node is added to each layer, except the output layer, to increase the flexibility of the model to fit the data. After this, feed-forward propagation is used to generate output activations and then back-propagation will train the neural network by adjusting the weights.

2.4.1 Feed-forward Propagation

Computing the activation values is done by multiplying the given feature vector with the weights in the neural network. As each node in a layer is connected with all nodes of the previous layer the multiplication of each connection is summed up. Then an activation function is applied to all layers except the output layer. For this the ReLU (equation 2.6) and sigmoid (equation 2.7) function are implemented:

$$f(x) = max(0, x) \tag{2.6}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

The ReLU function outperformed the sigmoid function, therefore only this one is applied during experimentation. The output activations are transposed to output probabilities in a range between [0;1]:

$$p_O(k) = \frac{\exp(a_O(k))}{\sum_{k=1}^{n} \exp(a_O(k))} \tag{2.8}$$

Where $a_O(k)$ is the activation of an output node, n is the number of output nodes and $p_O(k)$ is the output probability of that node.

2.4.2 Back-propagation

Back-propagation is applied to adjust the weights in the network by computing a gradient for each weight. This is done by applying the following equation to all weights:

$$W_{ji}(new) = W_{ji}(old) + \eta * \Delta_H(j) * a_H(j) \tag{2.9}$$

Where W_{ji} is the weight between the neuron of the actual layer and the neuron of the previous layer, η is the learning rate, $a_H(j)$ the activation from the previous neuron and $\Delta_H(j)$ represents the gradient for each weight. In order to let the network converge properly we anneal the learning rate η by multiplying it with a factor 0.99 after each epoch. An epoch has occurred when all randomly chosen

training patches have been used to train the neural network. After this the same patches are used again to train the network. For the output layer the delta value for a neuron is calculated by subtracting the actual output probability from the desired output. For the hidden layer this is done by applying the derivative of the activation function and multiplying this by a summation of the output gradients ($\Delta_O(k)$) times the weights between the hidden node and the output nodes (W_{ik}):

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

$$\Delta_H(i) = f'(a(i)) * \sum_{k=1}^{n} (\Delta_O(k) * W_{ik}) \tag{2.11}$$

Here $a(i)$ is the activation value of the actual hidden node and n is the number of nodes in the output layer.

2.4.3 Regularization

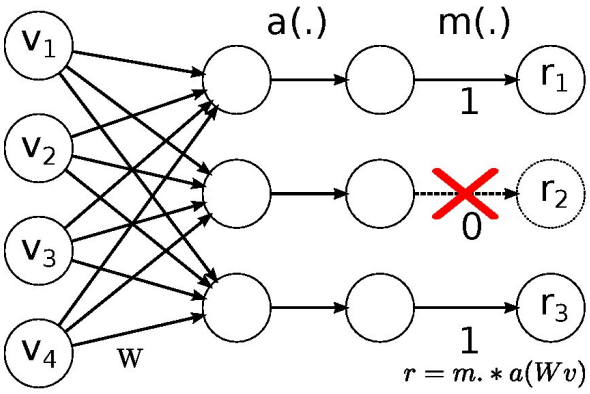

Figure 2.2: Dropout.

To prevent the network from overfitting a regularization method is used that was introduced by Srivastava et al. (2014), called dropout. This is a simple technique to prevent nodes from co-adapting too much by randomly dropping nodes, along with their connections, during training time (see Figure 2.2). Nodes are dropped with a probability of 0.5, this is only done for the hidden layer. Because the total activity that flows to the output nodes is less when applying dropout we adjust all activations by multiplying it with a factor that is based on the amount of activation that is dropped out. This factor (λ) is calculated by dividing the total activation of all nodes with the total activation of the dropped nodes (see equation 2.11).

$$\lambda = \frac{a_{totaldropped} + a_{totalnotdropped}}{a_{totaldropped}} \tag{2.12}$$

2.4.4 Classification

The classification of images is done by taking all feature vectors, which are extracted from the patches of an image, and giving them to the MLP. Output probabilities are calculated by applying feed-forward to the input vectors. Each output probability represents one class. In case of a pooled model the same is done for all patches but then using different MLPs for different quadrants. Finally an image is classified to the class with the maximum sum of probabilities over all patches.

2.5 Stacked MLP

In addition to the standard model we introduce a novel ensemble method named 'stacking MLPs' in which a second MLP is placed after the usual MLP. Here the maximum output probabilities serve as input values for the stacked MLP. The idea behind this technique is to let the stacked MLP learn to better classify images that have multiple potential classes (high probabilities). By giving these probabilities to a MLP the connections with the right class will be strengthened and those with the wrong class attenuated. Classification of images with multiple potential classes should then become better.

This is done by computing for each class (in our case 10) the highest output probability over all patches in one image, which are then given to the stacked MLP. In case of the combined model this will be done for all four MLPs separately and thus the stacked MLP will receive four times all output classes. The configuration for this stacked MLP is the same, except for the amount of hidden units. Parameter optimization revealed that less hidden units result in better performances.

In the experiment also the use of minimum and average output probabilities are evaluated. In Algorithm 2.1 some pseudo code is given to describe

Algorithm 2.1 Compute maximum, average or minimum output probabilities

Input: feature vectors from patches of an image
Output: maximum, average or minimum output probabilities
  for each patch p in an image **do**
    $a_O \leftarrow$ FEEDFORWARD(p)
    $OP \leftarrow$ COMPUTEPROBABILTIES(a_O)
    if type_output_probabilities = maximum **then**
      $P_{max} \leftarrow$ MAXPROBABILITIES(OP)
    else if type_output_probabilities = average **then**
      $P_{average} \leftarrow$ AVERAGEPROBABILITIES(OP)
    else if type_output_probabilities = minimum **then**
      $P_{min} \leftarrow$ MINIMUMPROBABILITIES(OP)
    end if
  end for

how to compute these maximum, average and minimum output probabilities.. First the output activations of the first MLP are computed by giving the feature vectors to the feed-forward network. Then these activations are transposed to probabilities. After this there is chosen between maximum, average or minimum output probabilities depending on the settings. For maximum output probabilities each patch of an image is evaluated and only the highest output probability for each class is saved into a vector. For minimum output probabilities the same is done but then with the lowest output probability for each class. The average output probabilities are computed by taking the sum of probabilities over all patches for each class and dividing it by the number of patches. These created vectors with output probabilities serve as input values for the stacked MLP.

3 Experiments

To measure the accuracy of the system the two widely known image classification datasets MNIST (LeCun and Cortes, 1998) and CIFAR10 (Krizhevsky, 2009) are used for testing. The accuracy is represented by the percentage of correctly classified images from the test set. For each model this accuracy is computed three times. From this the average accuracy and its standard deviation are calculated. A Student's T-test is used to see if two models perform significantly different from each other.

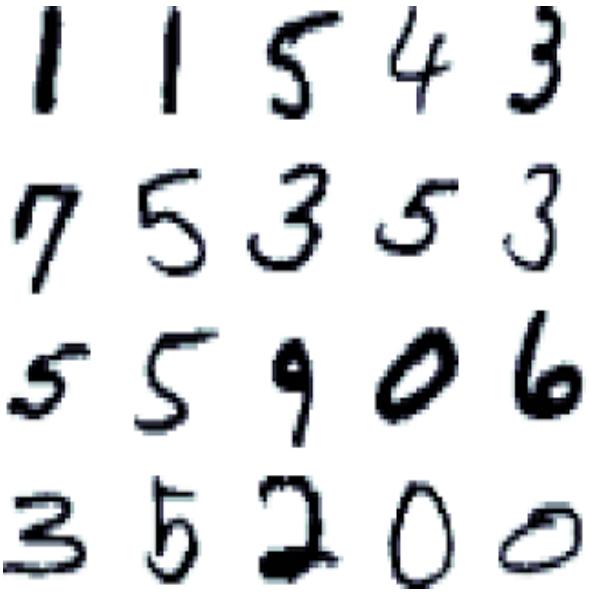

Figure 3.1: Samples from the MNIST dataset of handwritten digits.

3.1 MNIST

The MNIST database of handwritten digits (0-9) consist of 70,000 images (see Figure 3.1), which we divided randomly into 60,000 training images and 10,000 test images. By computing the center of mass of the pixels the images were centered to fit into 28×28 pixels. The anti-aliasing technique is used during normalization of the images to obtain gray levels. All images are not exactly equally distributed over all ten classes (digits 0-9).

3.2 CIFAR-10

The CIFAR-10 dataset (Figure 3.2) is a subset of the 80 million tiny images dataset (Torralba, Fergus, and Freeman, 2008). It contains 60,000 color images from which 50,000 are randomly chosen for training and 10,000 for testing. The images consist of 32×32 color pixels and are divided into 10 classes

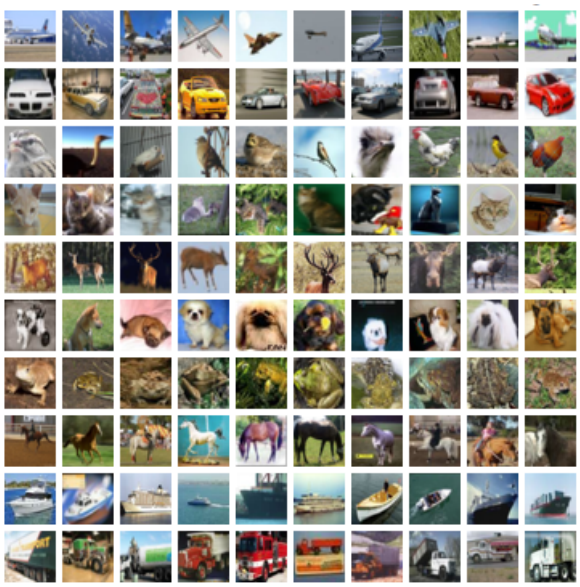

Figure 3.2: Samples from the CIFAR-10 dataset of objects.

of animals and objects. The dataset is equally divided into 6000 images per class.

3.3 Data Pre-processing

All images in the MNIST dataset are rescaled to 38×38 pixels. With a patch size of 30 and a scan stride of 1 each image consists of $9 \times 9 = 81$ patches. From each patch a feature vector is extracted by applying the HOG-feature descriptor, using 6×6 blocks of 5×5 cells. Each block has 9 orientation bins which ensures a one dimensional feature vector of $9 \times 6 \times 6 = 324$ values. For the CIFAR-10 dataset the color images are converted to grayscale and rescaled to 36×36 pixels. The patches have a size of 26 and a scan stride of 1. This results in $11 \times 11 = 121$ patches. Here HOG reduces the patches to 24×24 pixels to create 8×8 non-overlapping blocks of 3×3 cells. Having again 9 orientation bins results in a $9 \times 8 \times 8 = 576$ feature vector. In both datasets ten percent of the train images is used for cross-validation.

3.4 Overall Settings

Parameter optimization was first done with the standard model and based on those parameters some adjustments are made to better fit the other models. To give a good comparison between different architectures exactly the same parameters are used for each dataset (see Table 3.1). The number of splits for pooling is set on 2 when pooling is applied, and the amount of hidden units of the stacked MLP is only used in a stacked architecture.

Table 3.1: Program settings.

Parameter	MNIST	CIFAR-10
Number of splits for pooling	1 or 2	1 or 2
Input units	324	596
Hidden units	800	900
Output units	10	10
Number of layers	3	3
Learning rate	0.03	0.01
Epochs	50	50
Number of random patches	5×10^5	3×10^5
Hidden units of stacked MLP	500	400
Dropout probability for normal and stacked MLP	0.5	0.5

4 Results and Analysis

The results of the standard model together with models that include pooling and/or stacking are shown in Table 4.1. Here the mean percentages of correctly classified images over three runs are given together with their standard deviations. In addition to the stacked model where maximum output probabilities are used we also experimented with average and minimum output probabilities. These results are presented in Table 4.2.

4.1 Comparing non-stacked and stacked models

With a mean accuracy of 99.45% the standard model performs good, however it can not compete with the current state-of-the-art performance of 99.79% (Wan et al., 2013) on the MNIST dataset.

Table 4.1: Mean classification accuracies (%) with their standard deviations on both MNIST and CIFAR-10 datasets.

Method	MNIST	CIFAR-10
Standard model	**99.45 ± 0.01**	**68.30 ± 0.37**
Stacked model	97.81 ±0.24	55.33 ±1.43
Pooled model	98.82 ±0.23	67.13 ±0.30
Pooled+stacked	99.25 ±0.03	63.83 ±1.96

Table 4.2: Mean classification accuracies (%) with their standard deviations on both MNIST and CIFAR-10 datasets for all stacked models.

Method	MNIST	CIFAR-10
Maximum	97.81 ±0.24	55.33 ±1.43
Average	98.00 ±0.26	62.03 ±2.19
Minimum	69.08 ±2.88	27.68 ±12.92
Pooled+maximum	99.25 ±0.03	63.83 ±1.96
Pooled+average	99.28 ±0.13	60.79 ±4.37
Pooled+minimum	98.79 ±0.41	30.95 ±10.62

When comparing this to the stacked model a significant decrease in accuracy of 1.64% is measured with the Student's T-test ($t = 12.01, p < 0.05$). The addition of pooling also results in a loss of accuracy ($t = 4.74, p < 0.05$). Though, when combining pooling and stacking the accuracy increases with 0.43% in comparison to only pooling ($t = 3.19, p < 0.05$) and achieves a mean accuracy of 99.25%. However, this is still significantly worse than the standard model ($t = 10.16, p < 0.05$).

Although the current state-of-the-art performance on CIFAR-10 of 96.53 % (Graham, 2014) is not reached at all, the standard model again outperforms all other models (see Table 4.1). Student's T-tests show significant differences between the standard model and the stacked model ($t = 15.22, p < 0.05$). As well as between the standard and the pooled model ($t = 4.26, p < 0.05$). In contrast to the MNIST dataset, here a decrease in accuracy of 3.30% is found for the combined model (pooled and stacked) compared to the pooled model ($t = 2.89, p < 0.05$).

Therefore, from these results it is evident that stacking substantially leads to a drop in performance when using maximum output probabilities.

4.2 Evaluating the use of average or minimum output probabilities instead of maximum

In Table 4.2 it is clearly visible that the use of minimum output probabilities causes a reduction in accuracy on both datasets in comparison to models that use maximum or average output probabilities ($p < 0.05$). Except for the pooled model on the MNIST dataset, here no significant difference is found between maximum and average output probabilities ($t = 1.94, p > 0.05$ and $t = 2.00, p > 0.05$). This also is the result of too closely related output probabilities, even more than with maximum output probabilities.

When comparing average to maximum output probabilities no significant difference is found on the MNIST dataset for both non-pooled ($t = 0.95, p > 0.05$) and pooled models ($t = 0.47, p > 0.05$). On CIFAR-10 only a significant difference is found between average and maximum output probabilities for the non-pooled model ($t = 4.44, p < 0.05$). Here the use of average output probabilities increases the accuracy.

Though the non-pooled architecture with the use of average output probabilities outperformed both other models on CIFAR-10 it still can not compete with the non-stacked models. The next section gives a further insight into why these stacked models provide a decrease in accuracy compared to the standard model.

4.3 Deeper insight into the drop in accuracy of stacked models

To investigate why stacking leads to a decrease in performance we looked at the maximum/average output probabilities of the first MLP during classification. For this a test run was executed in which only 600 training and 100 test images of the MNIST dataset were used, the number of random patches was set to 10,000. Here the difference in output probability between the target class and the chosen class was computed for each test image. If an image was correctly classified the class with the second highest probability was subtracted from the target class. These results are shown in Figures 4.1 and 4.2 for correctly and wrongly classified images re-

spectively. The histograms do not add up to 100% because in Figure 4.1 the wrongly classified images and in Figure 4.2 the correctly classified images are not displayed.

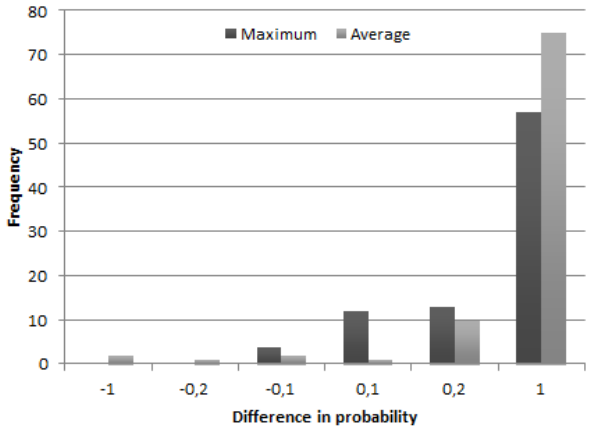

Figure 4.1: Histogram of the differences between output probabilities in case of correctly classified images for both maximum and average output probabilities.

For correctly classified images the target probability is usually higher than the second highest probability (target probability − second highest probability > 0). This means that the stacked model classified the image the same as the standard model would have done when taking the class with the highest output probability. The bigger the difference in probability the easier the stacked model assigns the image to the same class. This occurs more often with average output probabilities than with maximum. In an extraordinary case the difference is below zero which will say that the stacked model has learned to classify an image correctly which would be wrongly classified by the standard model.

More interesting are the cases where wrongly classified images by the stacked model would not be wrongly classified by the standard model as shown in the histogram of Figure 4.2 (difference in probability > 0). Because of these occasions the stacked models drop in accuracy. This occurs often when the difference between output probabilities is small. This makes sense because when output probabilities are closely related the stacked MLP earlier misclassifies an image. Therefore the decrease in accuracy of the stacked models is caused by too closely

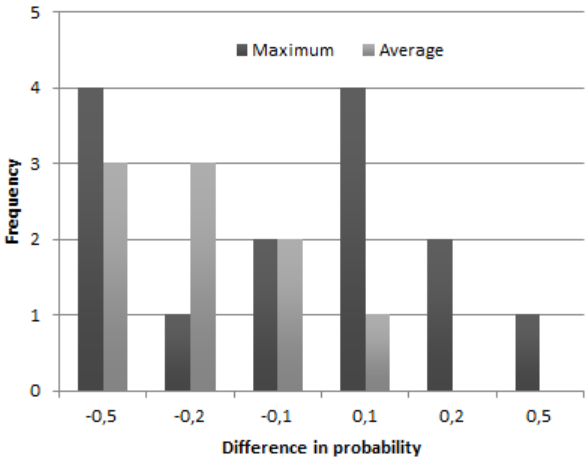

Figure 4.2: Histogram of the differences between output probabilities in case of wrongly classified images for both maximum and average output probabilities.

related output probabilities.

With average output probabilities this takes place less frequently because the difference between average output probabilities is often less small and therefore easier to classify. This occasion is also reflected in the high standard deviations shown in Table 4.2 (especially on CIFAR-10) and represents a lack of robustness of the stacked models.

5 Conclusion

In this research we studied the effect of a stacked MLP, trained on output probabilities, in a patch based image classification system. Our standard patch based model performed remarkably well on the MNIST dataset (99.45%). Nonetheless, the results showed that stacking consistently leads to a drop in accuracy on both MNIST and CIFAR-10 datasets compared to the standard model, also when minimum or average output probabilities are used instead of maximum. Deeper insight into the stacked model explained why this decrease in accuracy occurs, this is due to closely related output probabilities which causes misclassifications of images that would be correctly classified when no stacking was used.

Although the stacked models in this research could not compete with the standard model this

does not imply that stacking MLPs can not be effective. On the MNIST dataset the stacked MLP already caused some improvement in accuracy when pooling was added to the model (0.43%). In further research we intend to optimize the stacking technique by investigate what output probabilities lead to improvement of the stacked MLP. In this research we showed that closely related output probabilities causes a decrease in accuracy and therefore it can be beneficial to let the stacked MLP only train on images that do not have closely related output probabilities.

Another point of discussion is the fact that parameters for patch extraction and configuration of the MLP were settled for all models (on a specific dataset). This was preferred because it allowed to give a good comparison between non-stacked and stacked models. However, the consequence of this method is that maybe not every model performed optimal and so the comparison between pooled and non-pooled models is weak.

References

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893, 2005.

Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Y. LeCun and C. Cortes. *The MNIST database of handwritten digits*, 1998.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5 (3):1, 1988.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Olarik Surinta, Lambert Schomaker, and Marco Wiering. A comparison of feature and pixel-based methods for recognizing handwritten Bangla digits. In *2013 12th International Conference on Document Analysis and Recognition*, pages 165–169. IEEE, 2013.

Olarik Surinta, Mahir F Karaaba, Tusar K Mishra, Lambert RB Schomaker, and Marco A Wiering. Recognizing handwritten characters with local descriptors and bags of visual words. In *Engineering Applications of Neural Networks*, pages 255–264. Springer, 2015.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.