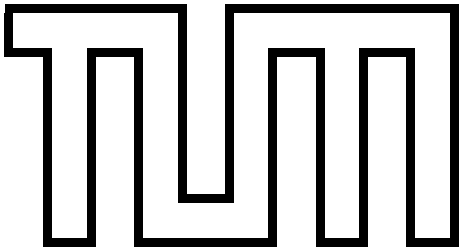


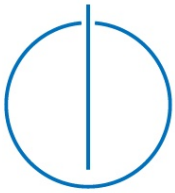
Technical University of Munich

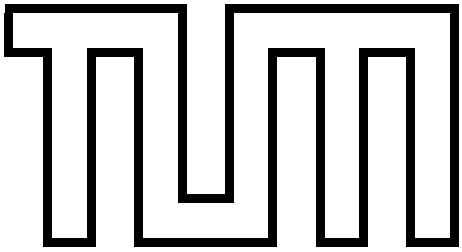
Department of Informatics

Bachelor's Thesis in Informatics

Image Classification with Geometrically Aware Sparse Grids

Tim Waegemans





Technical University of Munich

Department of Informatics

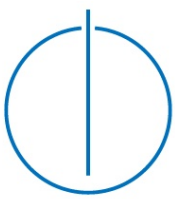
Bachelor's Thesis in Informatics

Image Classification with Geometrically Aware Sparse Grids

Bild Klassifizierung mit geometrisch bewussten Dünnen Gittern

Author: Tim Waegemans
Supervisor: Prof. Dr. Hans-Joachim Bungartz
Advisor: Kilian Röhner

Submission: 15.10.2017



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.10.2017

(Tim Waegemans)

Abstract

In this bachelor thesis a reduction of sparse grids is presented, to deal with the large number of dimensions of images. The reduction is based on the geometric structure of image data. The information of neighboring pixels is weighted more than pixels at different sides of the image. Additionally the use of Sparse Grid Density Estimation and Modified Linear Basis functions helps reaching the high dimensionality. An efficient evaluation of the modified grid is presented. The method is applied to different image datasets. Reaching promising results on selected digits of the MNIST dataset.

Contents

1 Introduction

With more sensors capturing more data the use of machine learning techniques have gained an increasing importance. Pictures make up a large part of that data. Every day many pictures are taken by security cameras, media, private smartphones, medical imaging and more. To deal with all this data image classification methods are needed.

So far sparse grids have been only used in machine learning application of up to 166 dimensions. This was in 2010 when Pflüger successfully applied a regression on the Musk-1 dataset [Pfl10]. Image data easily overshoots this number of dimensions.

In this thesis I will present a Sparse Grid approach for image classification. It is a further simplification from a regular Sparse Grid, by only including a detailed resolution only on neighboring dimensions. The goal is to use these simplifications to reach the high dimensionality of image data. Additionally Krenz and Khakhutskyy already achieved first results using this type of reduction [Kre16][Kha16].

In Section 2 I will give a brief introduction to sparse grids and classification methods using them. Then in Section 3 the geometrically aware grids are introduced with several possibilities of constructing them. Furthermore a brief discussion is held on the type of classification and basis functions. The implementation will follow in Section 4 where an efficient method for evaluating the geometrically aware grid is presented. Finally we will have a look at the performance of the method in section 5.

2 Foundation

This section gives a brief introduction to sparse grids and classification methods. For a more detailed explanation I refer to [BG04] for sparse grids, [Pfl10] and [PPB14] for classification methods.

2.1 Sparse Grids

Sparse Grids are a efficient method for interpolating high dimensional functions. Sparse Grids have an advantage over full grids that the number of grid points grows significantly slower with increasing dimensions. If the function that is being approximated is smooth enough, error to cost ratio is greatly increased, making sparse grids computationally feasible.

2.1.1 Hierarchical Function Space

Let us assume we have a one dimensional function space V_l with a certain level $l \in \mathbb{N}$ that defines the resolution of the function space. The hierarchical increment W_l is defined as the space that extends the lower lever space V_{l-1} to the current level V_l [BG04].

$$V_l = V_{l-1} \oplus W_l \tag{2.1}$$

With the trivial choice of $V_1 = W_1$, we can split V_l into several hierarchical subspaces [BG04].

$$V_l = \bigoplus_{i \leq k} W_i \tag{2.2}$$

When moving to higher dimensions we construct the d dimensional subspace $W_{\vec{l}}$ with the tensor product.

$$W_{\vec{l}} = \bigotimes_{i=1}^{d} W_{l_i}^{(i)} \tag{2.3}$$

Whereas the level vector $\vec{l} \in \mathbb{N}^d$ defines the level in each dimension and $W_l^{(d)}$ is the function subspace in dimension d. These subspaces can be combined to the d dimensional function space V_l by including all possible tensor products that can be constructed from one dimensional subspaces up to level l [BG04].

$$V_l = \bigoplus_{j \in \mathbb{N}^d \wedge |j|_\infty \leq l} W_{\vec{j}} \tag{2.4}$$

The construction of V_2 for the two dimensional case is shown in figure 2.1. The next section will focus on certain choices for V and W.

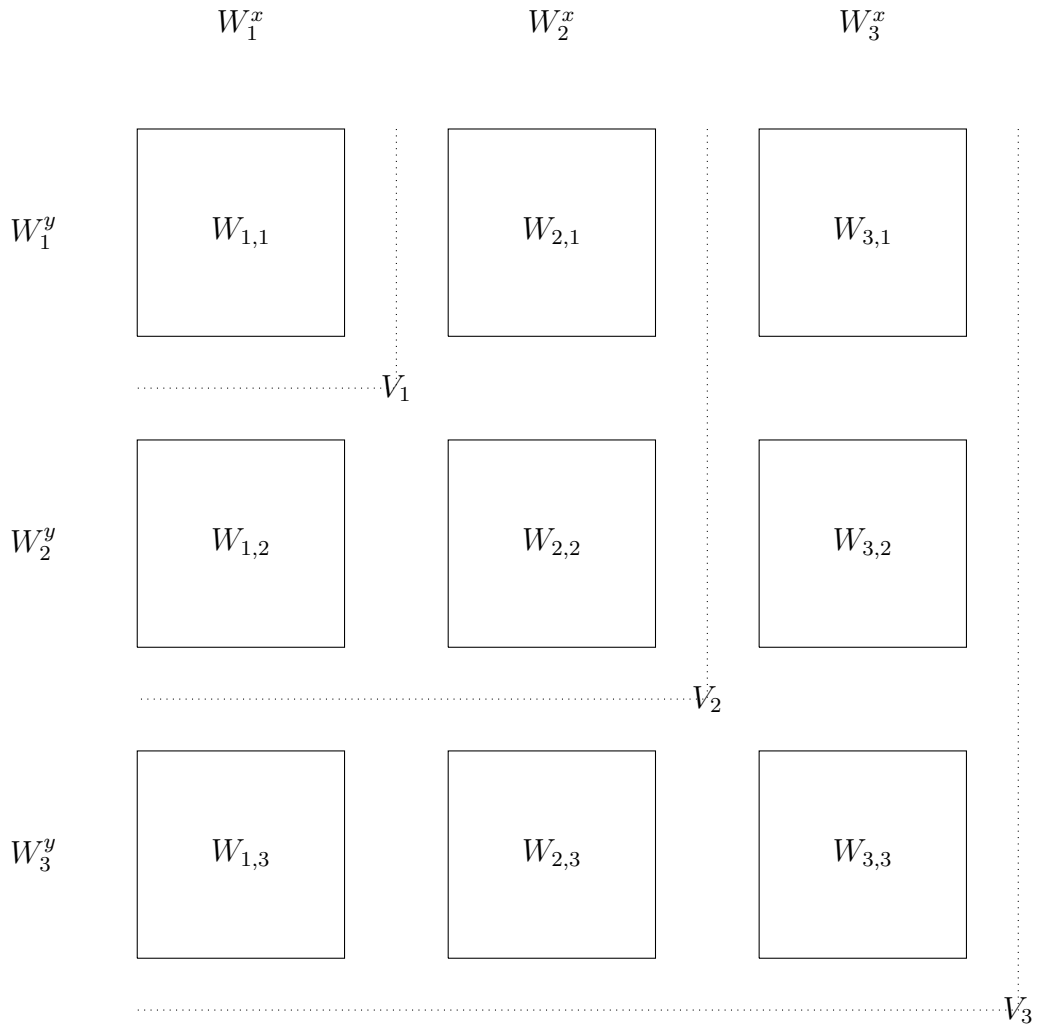


Figure 2.1: Hierarchical subspaces for dimension two. The two dimensional subspaces are created by the tensor product of the one dimensional hierarchical subspaces. By combining all subspaces in a square pattern we maintain the function space V_l.

2.1.2 Basis Functions

In this section I will introduce linear basis functions with and without boundary conditions and a modified version of the linear basis. Other basis functions, including non linear ones, are possible. However for computational reasons this thesis will only cover the most basic basis functions. Before the basis functions are introduced I will cover some assumptions that apply to the basis functions.

General assumptions For a more simple notation the domain is reduced to $\Omega = [0, 1]$. However every finite domain can be affinely transformed to the reduced domain.

The function space V_l will be constructed by several underlying basis functions $\phi_{l,i}$.

$$V_l = \text{span}\{\phi_{l,i}(x)\} \tag{2.5}$$

The basis functions of the multidimensional space constructed with the tensor product are obtained by multiplying the basis functions in each dimension [BG04].

$$\phi_{\vec{l},\vec{i}}(\vec{x}) = \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j) \tag{2.6}$$

The following paragraphs will explicitly describe the basis functions $\phi_{l,i}$.

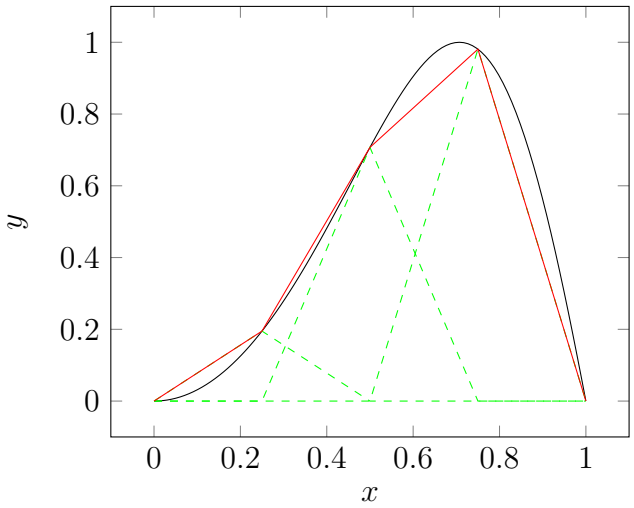
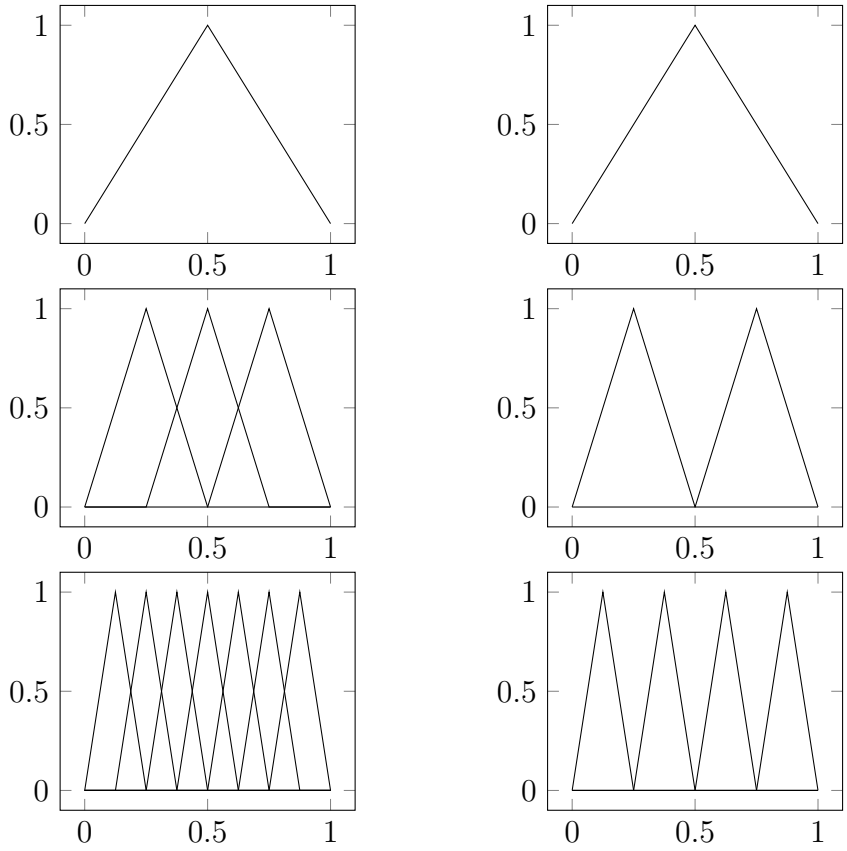


Figure 2.2: Piecewise linear interpolation of $\sin^2(\frac{x\pi}{2})$



(a) Nodal Basis functions (b) Hierarchical Basis

Figure 2.3: Nodal and Hierarchical Linear Basis Functions for levels 1 to 3.

Linear Basis I will start with a one dimensional piecewise linear interpolation problem. This can be easily accomplished with "hat" functions as seen in Figure 2.2. The basis functions can be derived from what Bungartz called the mother of all hat functions [BG04]:

$$\phi(x) = \max\{1 - |x|, 0\} \tag{2.7}$$

For the linear basis supporting points are chosen at multiples of 2^{-l} with level $l \in \mathbb{N}$. This results in the basis functions shown in figure 2.3a. The mother function (2.7) can be modified to fit V_l:

$$\phi_{l,i}(x) = \phi(2^l \cdot x + i) \tag{2.8}$$

Hence the function space can be denoted as:

$$V_l = \text{span}\{\phi_{l,i}(x) | 1 \leq i \leq 2^l - 1\} \tag{2.9}$$

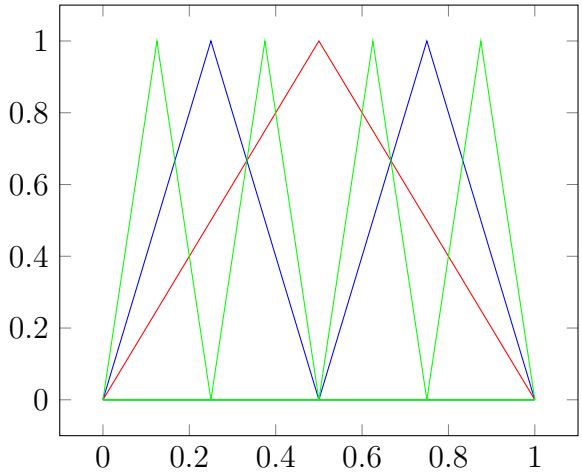


Figure 2.4: The hierarchical basis functions up to level 3.

To move from the nodal space V_l to a hierarchical space we can have a look at where information is gained by choosing a higher level. V_1 only contains the supporting point $x_1 = 0.5$ whereas V_2 contains the three points $\tilde{x}_1 = 0.25$, $\tilde{x}_2 = 0.5$ and $\tilde{x}_3 = 0.75$. Since the point $\tilde{x}_2$ is already in V_1 the use of $V_1 \oplus V_2$ contains redundant information. Instead only the ansatz functions at the new supporting points can be added. To obtain the new ansatz functions one has to only take the functions with odd indices. This gives us the hierarchical subspace W_l.

$$W_l = \text{span}\{\phi_{l,i}(x) | 1 \leq i \leq 2^l - 1 \wedge i \text{ is Odd}\} \tag{2.10}$$

To receive the full hierarchical function space all W_i up to the desired level have to be combined (see Figure 2.4). By construction and from Figure 2.3 it can be easily seen that the hierarchical and nodal space are identical since a piecewise linear interpolation is done at the same supporting points.

The two dimensional case can be seen in Figure 2.5. The red mesh $W_{(1,1)}$ is obtained by multiplying the the functions of W_1 for x and W_1 for y together. Equivalently the blue mesh is $W_{(2,1)}$ and $W_{(1,2)}$ while the green one is $W_{(2,2)}$.

By combining all subspaces in every dimension up level l a full grid of level l is obtained. The grid in Figure 2.5 is of level 2.

Boundary conditions With the linear basis we assume that the values at the boarder are 0. This however is not always the case. To extend the linear basis functions by boundary we can add two basis function to the subspace W_1. With

$$W_1 = \text{span}\{\phi_{0,0}(x), \phi_{0,1}(x), \phi_{1,1}(x)\} \tag{2.11}$$

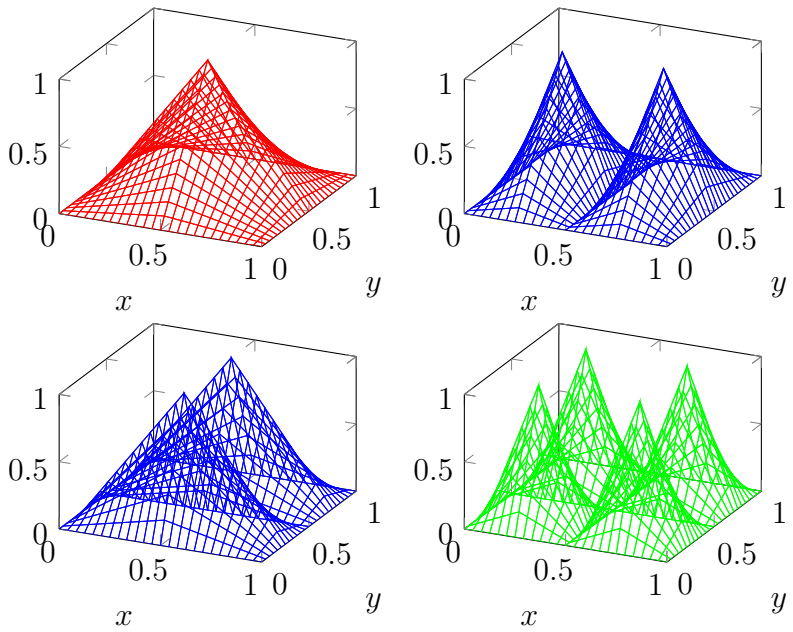


Figure 2.5: 2D Linear Basis

the basis functions in Figure 2.6 are created [Pfl10]. What is a small adjustment in the one dimensional case, greatly increases the number of basis functions for higher dimensional applications. $W_{\vec{1}}$ already contains 3^d basis functions. And all subspaces $W_{\vec{l}}$ with l containing k instances of level 1, have 3^k times the number of basis functions the regular linear basis has. Effectively this is an increase of $\mathcal{O}(3^d)$.

Modified linear basis The standard linear basis can be modified to contain boundary information without adding new basis functions. The following modification will give us the modified linear basis [Pfl10].

$$\phi_{l,i} = \begin{cases} 1 & \text{if } l = 1 \wedge i = 1 \\ \left\{ \begin{cases} 2 - 2^l \cdot x & \text{if } x \in [0, 2^{-l+1}] \\ 0 & \text{else} \end{cases} \right\} & \text{if } l > 1 \wedge i = 1 \\ \left\{ \begin{cases} 2^l \cdot x + 1 - i & \text{if } x \in [1 - 2^{-l+1}, 1] \\ 0 & \text{else} \end{cases} \right\} & \text{if } l > 1 \wedge i = 2^l - 1 \\ \phi(2^l \cdot x - i) & \text{else} \end{cases} \tag{2.12}$$

The basis functions are shown in 2.7. Aside from the added boundary information the modified linear basis has a further feature worth mentioning. The level 1 basis function always evaluates to 1. Thus we do not have to

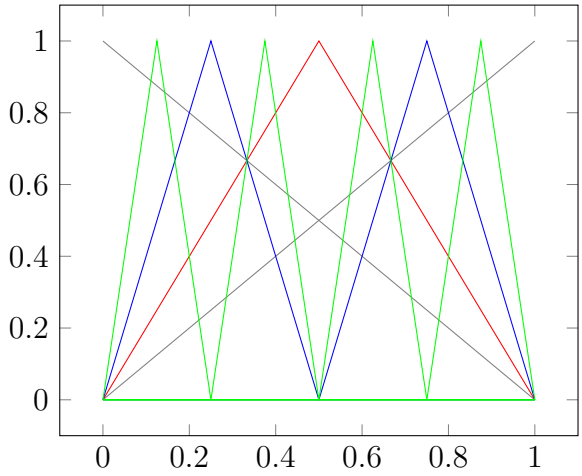


Figure 2.6: The hierarchical linear basis functions with boundaries up to level 3.

look at x for the evaluation. The 2 dimensional tensor product construction is shown in Figure 2.8.

2.1.3 From Full to Sparse Grid

Since the basis functions in V_l are organized in a grid fashion it is called a full grid. The number of grid points in the full grid grows exponentially with the number of dimensions. With the linear basis functions we have a gridsize of $(2^l - 1)^d$. With increasing dimensions this method is to complex to compute.

A reduced form of the full grid space is the sparse grid space $V_l^{(1)}$. The sparse grid however only includes the subspaces up to a diagonal instead of a square. We can formulate this as [BG04]:

$$V_l^{(1)} = \bigoplus_{|j|_1 \leq l-d+1} W_j \tag{2.13}$$

For a two dimensional sparse grid of level 2 the red and blue subspaces of Figures 2.5 and 2.8 have to be included. To reach level 3 the blue subspace and $W_{1,3}$ and $W_{3,1}$ would need to be included as well. The comparison to the full grid for two dimensions can be seen in Figure 2.9.

However we cannot simply remove grid points and expect the same accuracy. According to Pflüger the grid point to error ratio of the full grid is in $\mathcal{O}(N^{2+d})$ and for the sparse grid it is only $\mathcal{O}(N^3)$ for sufficiently smooth functions [Pfl10]. Whereas N is the number of grid points for the one dimensional grid. This makes sparse grid a more efficient method than full grids.

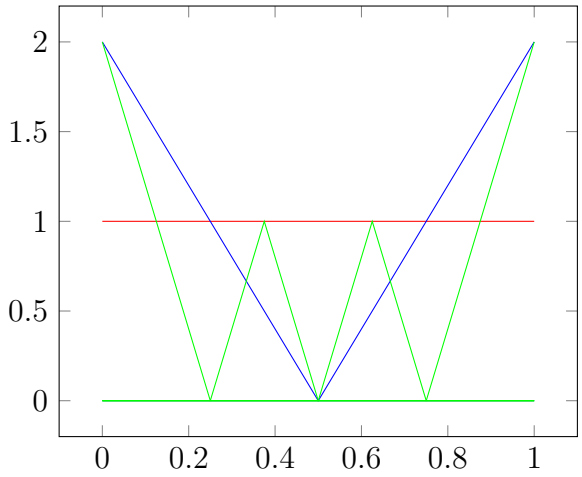


Figure 2.7: Modified linear basis function up to level 3.

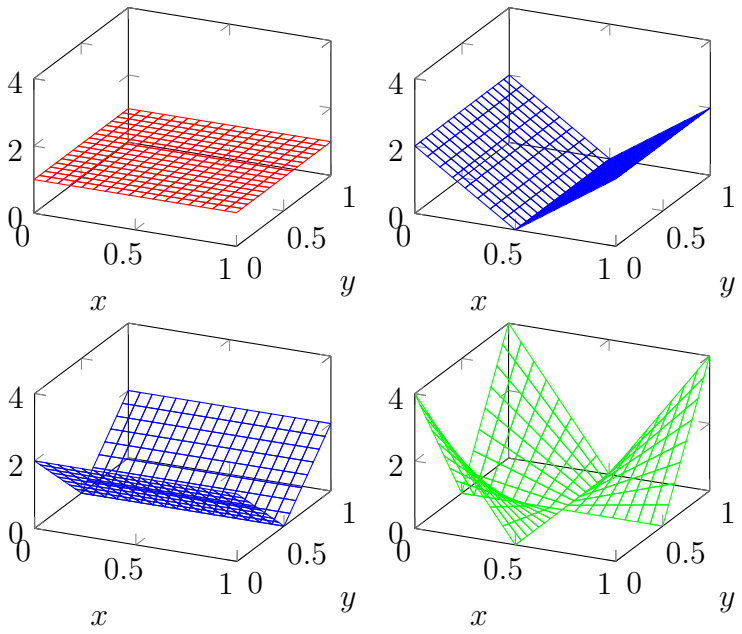


Figure 2.8: 2D Modified Linear Basis

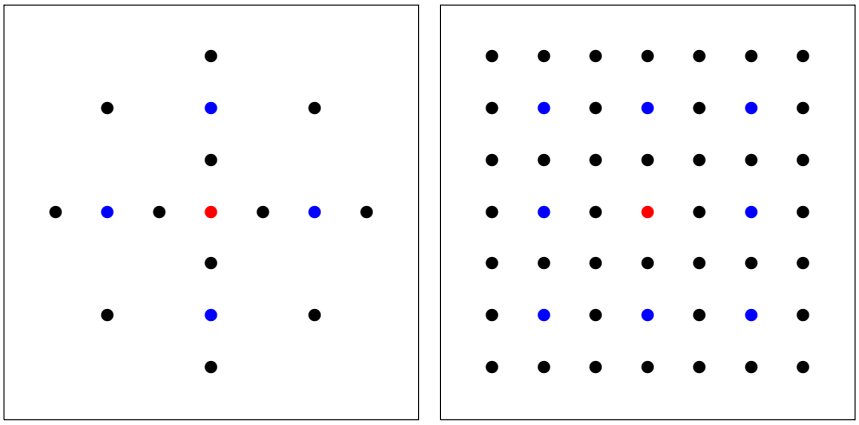


Figure 2.9: Gridpoints of the two dimensional sparse grid (left) and the full grid (right). Both are of level 3. The level 1 and 2 grids are shown in red and blue.

2.1.4 Adaptivity

Instead of increasing the level of the sparse grid to reach a lower error, it would be more efficient to only include new basis functions in critical areas. This can be done by refining a grid point. To refine a grid point usually all basis functions of the next level that cover the same area are added [Pfl10]. While basis functions that also cover area that is not covered by the refined point are excluded. An example for two dimensions is shown in Figure 2.10. When refining a point we add 2 new points per dimension (the two neighboring points).

One approach to identify the crucial parts of the domain is to use the values of our basis functions. The higher the value of a basis function the more it is correcting the higher level parent function. We assume that this region need a higher resolution. This refinement is called surplus refinement [Pfl10]. However other refinement criteria are also possible.

2.2 Classification

Sparse grids can be applied to a classification problem. I will introduce two classification methods. The Regression is a method very similar to interpolation. Whereas the density estimation interprets the data as random samples of a underlying density function and tries to approximate the density.

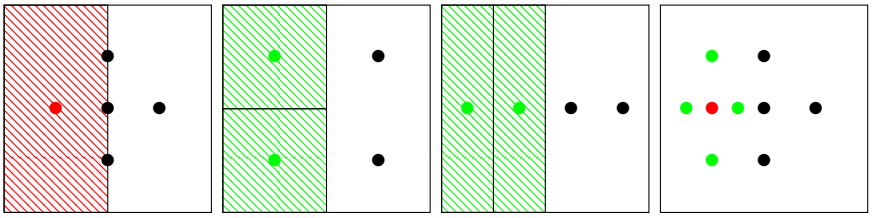


Figure 2.10: Refinement of the red point (on the left) is done by adding the next level basis functions that cover the same area. The added points (in green) and their effecting area are shown in the two middle boxes. On the right we have the refined grid.

2.2.1 Sparse grid regression

First the classification problem has to be transformed to a regression problem. I will first view a binary classification to introduce the regression and later present a method for more classes.

For two classes we can simply assign the value $+1$ to one class and -1 to the other and then apply a regression. Data is then classified by the sign of the output.

The basic idea of regression is to approximate a function $f(x)$ that is as close as possible to the targets for the given training data. To avoid over fitting a regularization term can be introduced. Pflüger used following least squares formula [Pfl10]:

$$f(x) = \arg\min_{f \in V^1} \frac{1}{M} \sum_{i=1}^{M} (f(x_i) - y_i)^2 + \lambda |\nabla f(x)|_2^2 \tag{2.14}$$

The sum $\sum_{i=1}^{M}(f(x_i) - y_i)^2$ assures that the function is close to the data. While $|\nabla f(x)|_2^2$ penalizes functions that are not smooth. The trade off between the closeness to the data and smoothness can be adjusted with the choice of λ. The value of λ is chosen problem specific and typically ranges between 10^-5 and 1. The problem can be formulated to the following system of linear equations

$$\left(\frac{1}{M} B B^T + \lambda C\right) \alpha = \frac{1}{M} B y \tag{2.15}$$

to calculate the coefficients α [Pfl10]. The regularization term is represented by $C_{i,j} = \langle \nabla \phi_i(x), \nabla \phi_j(x) \rangle_{L2}$. While B and B^T contain the evaluation of the basis functions $B_{i,j} = \phi_i(x_j)$ for each data point x_j. Since the matrix B is of size $N \times M$ and dependent on the data set, solving the linear system becomes more complex with a larger dataset. Instead of solving the system of linear equations explicitly gradient decent is commonly used.

For a problem with more than two classes we could assign each class with a different value and then apply a regression. This might not be the optimal approach for most problems, since classes that are close to each other in the input data may not have been assigned values that are close to each other. A different approach is a "one hot" method where a regression is applied for each class. Data points that lie in the class are assigned to value 1 and the other points to 0. We can then use $f_i(x)$ for class i as a measure of how certain the function is and assign the label of the highest evaluating function [Pfl10].

$$\text{label}(x) = \arg\max_i f_i(x) \tag{2.16}$$

2.2.2 Classification with sparse grid density estimation

In contrast to regression density estimation interprets the data as random samples generated from a probability distribution instead of noisy evaluations of a function. The goal of the density estimation is to approximate the underlying density function. Areas that have a lot of samples will have a higher density then areas with fewer samples.

Density based classification Classification based on a density functions follows the previously mentioned "one hot" approach. For each class the underlying density is approximated with the training data. Data is classified by choosing the label that has the highest density at a given point.

Sparse grid density estimation According to Peherstorfer in 2013 we can formulate the sparse grid density estimation with following system of linear equations [Peh13].

$$(R + \lambda C)\alpha = b \tag{2.17}$$

Where R is a matrix containing L-2 scalar products of the ansatz functions $R_{i,j} = \langle \phi_i, \phi_j \rangle_{L2}$. $C_{i,j} = \langle \Lambda\phi_i, \Lambda\phi_j \rangle_{L2}$ is a regularization matrix with $\lambda > 0$ as regularization factor. The vector α represents the weights α_i of the basis functions ϕ_i. on the right side, b is the average evaluation of the basis function ϕ_i, resulting in $b_i = \frac{1}{M}\sum_j^M \phi_i(x_j)$. For Λ there are many different options including very complex ones. However by choosing C equal to the identity I we obtain a regularization that benefits smooth functions and does not need any computation. An explanation can be found in [Peh13] and [PPB14].

This system of linear equations has some beneficial properties. The matrix $(R+\lambda C)$ has the dimensions $N \times N$. This allows the use of large datasets since the system does not scale with the size of the dataset. Furthermore for a fixed λ the matrix is completely independent of the data. Hence we can

precompute a matrix decomposition to reduce the complexity during training form $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$.

Cholesky decomposition A practical decomposition for the Density Estimation is the Cholesky decomposition. It allows us to efficently add new grid points even after the matrix is decomposed [Sie16]. With this property we can train the system, then apply adaptivity at crucial parts of the system and continue the training without needing to recompute the matrix decomposition. However a change of λ in the decomposed form is to time intensive to be done [Sie16]. So the value of λ has to be set before training.

3 Image Classification with Sparse Grids

The following section will deal with the challenges of using sparse grids for image classification. The most challenging aspect is the high number of dimensions since each pixel represents a dimension of the input data. However image data has the property of having organized dimensions. Each pixel has an exact position on a two dimensional grid with neighboring pixels in each dimension. This property will be used to construct a sparse grid classifier that can deal with large number of dimensions.

3.1 Geometrically aware sparse grids

The goal of geometrically aware sparse grids is to reduce a regular sparse grid intelligently to obtain a more efficient error to gridpoint ratio.

3.1.1 Interaction terms

Similar to the step from full to spare grid in 2.1.3 a further reduction can be performed by not including certain subspaces. $W_{2,1}$ only adds a new resolution in the x_1 direction as seen in the contour plots of 3.1. Equivalently $W_{1,2}$ only contains resolution in the x_2 direction. Only with $W_{2,2}$ information on the interaction between the two dimensions is gained. This becomes even clearer with the use of modified linear basis functions (see Figure 3.2). A d-dimensional subspace models the interactions between all dimensions that are of level two or higher.

This property can be used to reduce the grid by only including subspaces corresponding to set of interactions. To describe what subspaces should be included I will make use of interaction-terms mentioned by Krenz in 2016 [Kre16]. Each interaction-term is a subset of all input dimensions $D = \{d_1, d_2, ..., d_d\}$.

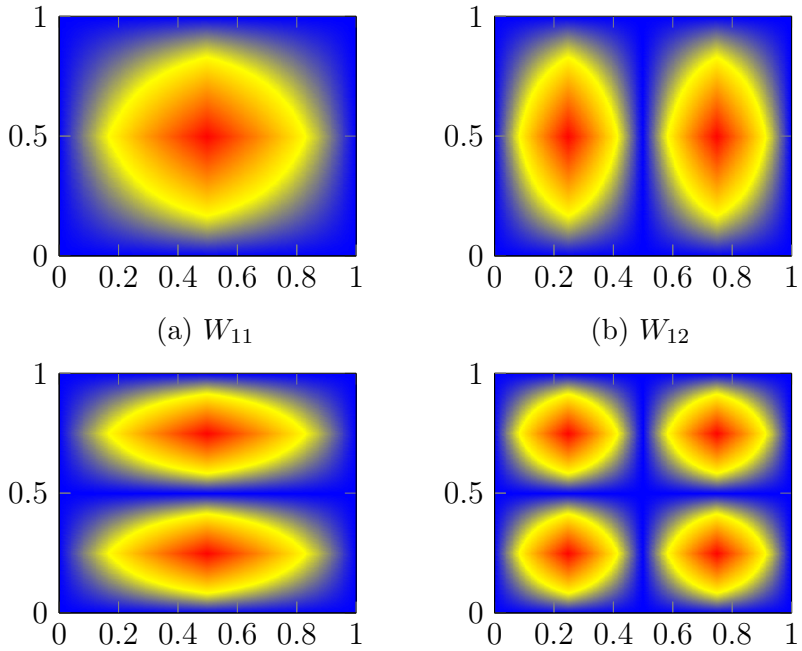
$$t := \{d_1, d_4, d_9\} \subseteq D \tag{3.1}$$

$$T := \{t_1, t_2, ..., t_m\} \subseteq 2^D \tag{3.2}$$

Lets define a function $\zeta(\vec{l})$ that returns the modeled interactions of the subspace $W_{\vec{l}}$:

$$\begin{aligned} &\zeta : \mathbb{N}^d \to 2^D \\ &\zeta(\vec{l}) = \{d_i | l_i \geq 2 \wedge i \in [d]\} \end{aligned} \tag{3.3}$$

With the use of this function the interaction term aware sparse grid V^T can be described. The grid contains all subspaces that model a interaction



(a) W_{11} (b) W_{12}

(c) W_{21} (d) W_{22}

Figure 3.1: 2 D contour plots of linear basis functions

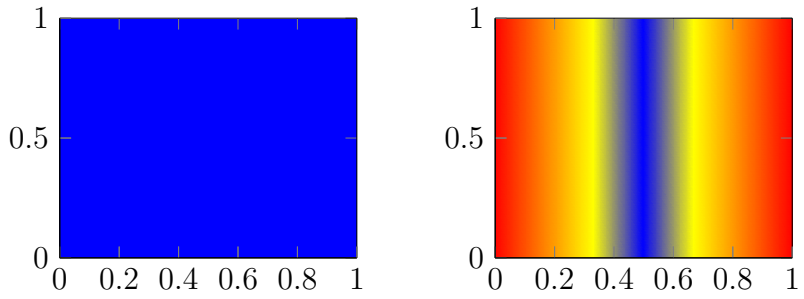
between dimensions given in the set of wanted interactions T.

$$V^T = \bigoplus_{W_l \in V^{(1)} \wedge \zeta(l) \in T} W_l \tag{3.4}$$

A three dimensional example grid is given in Figure 3.3. With a efficient choice of T the model can be drastically reduced in size, while still preserving the wanted information. Section 3.1.3 will focus on finding an appropriate T for image data.

3.1.2 Refinement

The refinement of a interaction based sparse grid is very similar to the regular sparse grid. The difference lies in the added points. Since our modified sparse grid is limited by the interactions we will continue to only add points if they model a wanted interaction. So instead of adding $2 \cdot d$ new grid points for each refined point only $2 \cdot (\max_{t_i \in T} |t_i|)$ new points are added. The number of new grid points is limited by the interaction containing the most dimensions. This can greatly decrease the number of grid points for each refinement step.



(a) W_{11} (b) W_{12}

(c) W_{21} (d) W_{22}

Figure 3.2: 2 D contour plots of modified linear basis functions

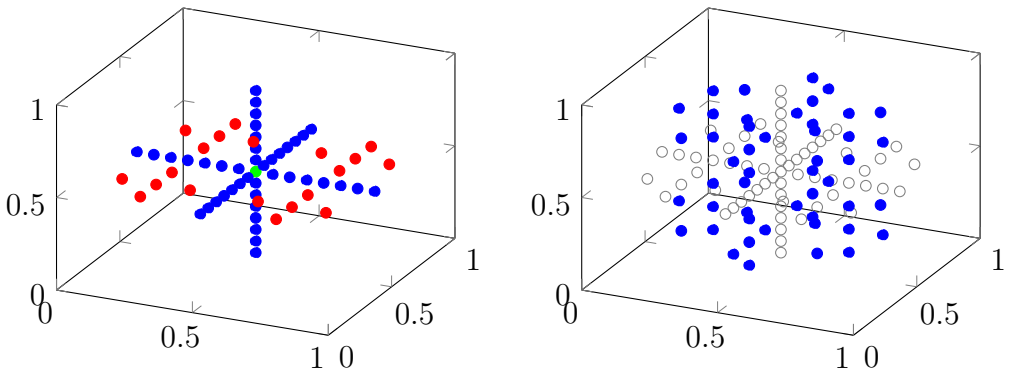


Figure 3.3: On the left: 3D interaction term sparse grid of level 4. $T = \{\varnothing, \{x\}, \{y\}, \{z\}, \{x, y\}\}$ On the right the not included point are shown.

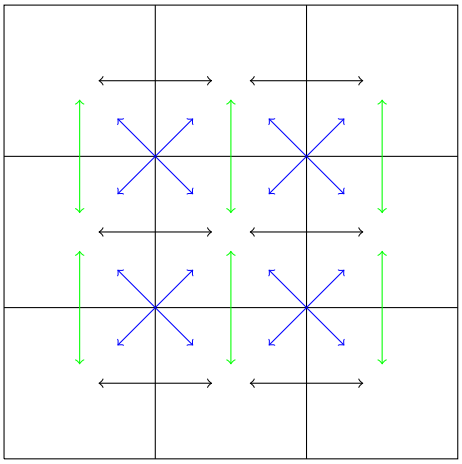


Figure 3.4: Neighbor stencil

3.1.3 Stencil choice

There are several different stencils that can be used for a geometrically aware sparse grid. The most basic one being only to consider the combination of two dimensions. One approach is, to take only the direct geometrical neighbors of a dimension. I will call this stencil **DN** for **D**irect **N**eighbor. This can be extended to include the diagonals as well (see Figure 3.4). I will refer to this as **DNDiag**. The resulting sparse grid is a highly simplified version of a regular one. The reduction of grid points is shown in Table 3.5, where 8x8x3 data is put into a sparse grid, using only the direct neighbor method.

	level 2	level 3	level 4
neighbor stencil	385	3 009	11 969
regular SG	385	74 497	9 659 649

Figure 3.5: comparison in gridsize

Possibly a more powerful stencil could compare more than two dimensions. To achieve a comparison between n different dimensions the sparse grid level has to go up to $n + 2$. So very large stencils will face the problem of increasing the number of grid points a lot.
A simple method for comparing multiple dimensions, is to take a **square** (for higher geometric dimensions a hypercube) of size n neighboring dimensions and comparing all of them. The combinations using less then n^d dimensions will be also included. This is shown in Figure 3.6 for a two by two square.

Different stencils can be combined in different geometrical dimensions. For image classification an interesting method would be to compare a square

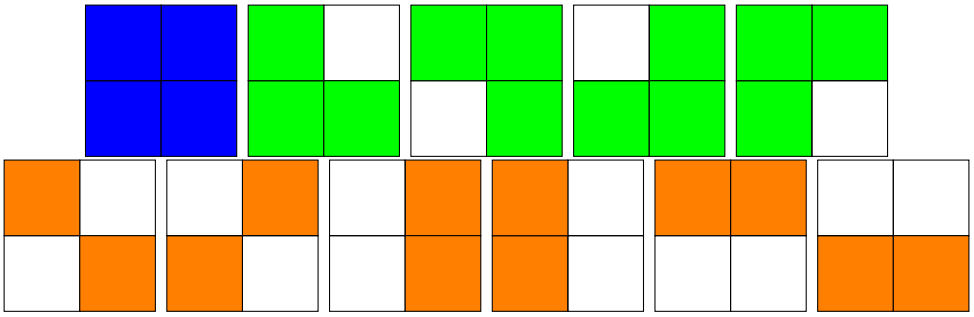


Figure 3.6: Cube/Square selection

of pixels next to each other in the same color channel, but only compare different colors channels in the same pixel location.

Grid points of different stencils for images This section will analyze the effect of stencil choice on the number of grid points. This is a crucial step in developing an adequate stencil. The number of grid points can be calculated by counting the subspaces of each level, since subspaces of the same level have the same number of grid points. A subspace of level l has 2^{l-1} grid points. The number subspaces of level l added to the grid by a interaction of k dimensions is equivalent to the possibilities of distributing $l-k-1$ not differentiable balls on k differentiable urns. This results in $\binom{l-k-1+k-1}{k-1} = \binom{l-2}{l-k-1}$ subspaces. When summing up all grid points contained in the subspaces modeling a interaction of k dimensions following formula is obtained for a sparse grid of level m:

$$g_k = \begin{cases} \sum_{i=k+1}^{m} 2^{i-1} \cdot \binom{i-2}{i-k-1} & \text{if } k \geq 1 \\ 1 & \text{if } k = 0 \end{cases} \tag{3.5}$$

For better calculation the set of interaction terms T can be partitioned by the number of dimensions contained in a interaction.

$$T = \biguplus_{i} S_i$$

$$S_i = \{t | t \in T \wedge |t| = i\}$$

The next step is to calculate the cardinality $|S_i|$ of each partition. To get the total number of grid points the sum

$$\#\text{gridpoints} = \sum_{i=0}^{\max k} |S_i| \cdot g_i$$

has to be calculated. Since S_i is not only dependent on the size of the image, but also on the stencil choice, different stencils will be investigated. g_k is solely dependent in the level of the sparse grid. The number of grid points for the following stencils are based on a image of size x x y.

Neighbor Stencil on a single color image In Figure 3.4 the neighbor stencil can be seen for one color channel. First only the direct neighbors (excluding the blue lines) will be considered.

The partition S_1 is equal to all dimensions D. In this case:

$$|S_1| = |D| = x \cdot y$$

S_2 is shown by the arrows in Figure 3.4. There are $x(y-1)$ green and $(x-1)y$ black arrows. This lead to:

$$|S_2| = x(y-1) + (x-1)y$$

When including the diagonal neighbors as well, the blue arrows have to be added too.

$$|S_2'| = x(y-1) + (x-1)y + 2(x-1)(y-1)$$

For the 32x32 images, the resulting grid sizes are shown in table 8.1.

Neighbor Stencil on a 3 color channel image When introducing color information the dimensions increase by a factor of three. If the same stencils mentioned in the paragraph above are used on each color channel individually, without modeling interactions between channels, the number of grid points will increase by the factor three. This stencil I will refer to as **NoCol** since no interactions between the color channels are included.

However interactions between the color channels might be desirable. This can be achieved by adding interactions consisting of the R,B and G values of a pixel. I will call this **Col**. Hence we have to add $\{r,g\}\{r,b\}\{g,b\}\{r,g,b\}$ to the interaction terms. This leads to the following new S_i.

$$|S_1| = |D| = 3xy$$

$$|S_2| = 3x(y-1) + 3(x-1)y + \binom{3}{2}xy$$

$$|S_3| = xy$$

In Figure 3.7 we see the grid sizes of the geometrically aware grids and the regular sparse grid. The introduction of the stencils greatly slows down the growth of the sparse grid with rising level.

3.2 Basis Choice

With the large number of dimensions of image data the choice of basis functions plays a great effect. This section will discuss what basis functions are feasible for the high dimensions and data distributions of images.

3.2.1 Linear Basis

Linear Basis function are very simple and offer a decent resolution for most classification problems. Each basis function is a scaled and shifted version of the hat-function (2.8), making the implementation simple. However there are some problems for this application.

1. For the evaluation we have to compute many evaluations of ϕ. The evaluation per grid point is dependent on the dimensions d and the number of data points M resulting in a complexity of $\mathcal{O}(d \cdot M)$.

2. With the high number of multiplications of small numbers the evaluation reaches the machine accuracy. I will illustrate this on the example of a 32x32 image resulting in 1024 dimensions. With the expected value of ϕ in one dimension being 0.5 the resulting expected value of all evaluations is $0.5^{1024} \approx 5.6 \times 10^{-309}$ easily exceeding the accuracy of a standard double.

3. If a fully black or white pixel is located in the image, all basis functions will evaluate to zero. This property is very common for images. However this problem could be avoided by scaling the image data to a smaller interval e.g. $[0.1; 0.9]$.

3.2.2 Boundary Basis

With the addition of boundary grid points it is possible to deal with fully black or white pixel. This however will drastically increase the number of grid points. $g_k^{\text{boundary}} = 3^{d-k} \cdot g_k$. Even with the interaction terms this results in an increase of order $\mathcal{O}(3^d)$. With this increase in grid points it will not be possible to reach the high dimensionality of image data.

3.2.3 Modified Linear Basis

As result of the problems with the previously mentioned basis functions I chose modified linear basis functions. The biggest benefit is that the basis function of level one evaluates to one. So the number of ϕ-evaluations per grid point is dependent on the level of the grid and not on the number of

dimensions. Each grid point needs a maximum of $l - 1$ ϕ-evaluations since all basis functions of level one do not need to be evaluated. Lower level subspaces will need even less evaluations. This tackles the problem of numerical stability and greatly decreases the complexity of the evaluation. Furthermore the modified linear basis even provides information at the boarder of the model.

3.3 Classification Method

The classification method I chose was the sparse grid density estimation, since the linear system scales with the number of grid points instead of the data points. Another advantage is the possibility of decomposing the matrix before training, since the same decomposed matrix can be used for every class. Additionally we can decompose and store the decomposed matrices for common image sizes and choices of λ. The stored matrices can then be used on different datasets.

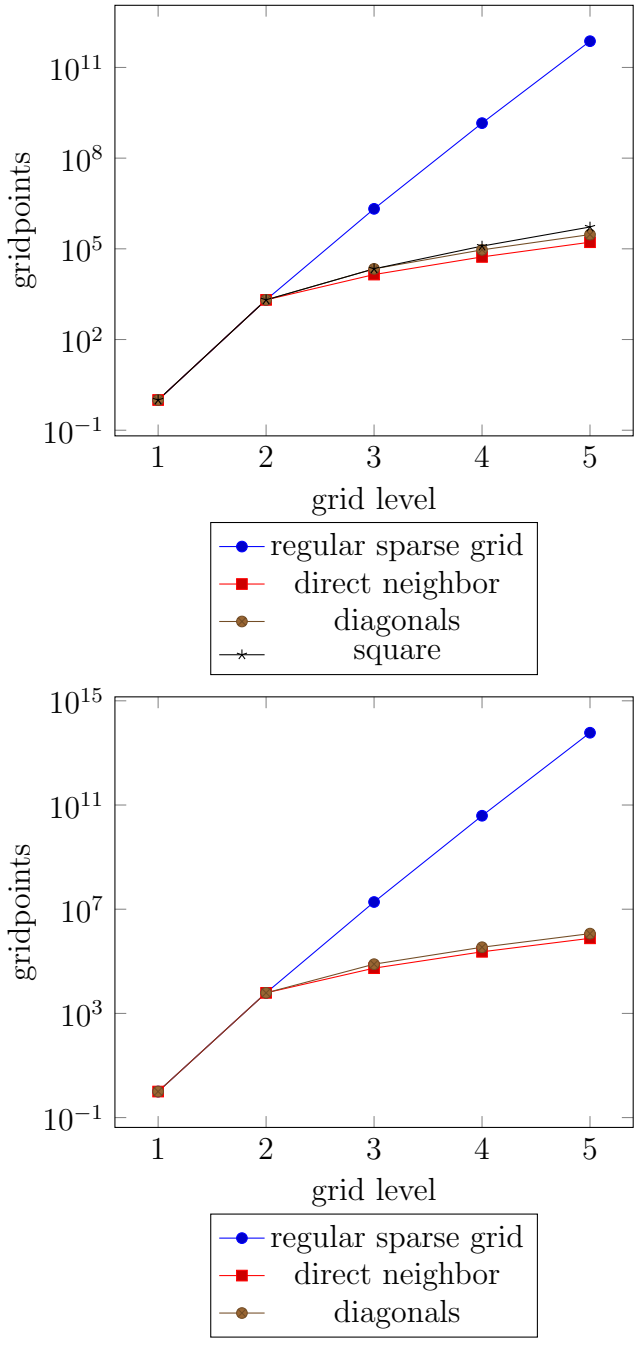


Figure 3.7: Number of gridpoints for each stencil for a 32 by 32 image. On the top single color images are shown and on the bottom the images consist of three color channels.

4 Implementation

The implementation of the geometrically aware grid was done in the library SG++ using C++ [PPB10]. I will first go over some minor adjustments and implementations I made. Then I will present an algorithm for an efficient evaluation of the grid, that helped to reduce the training time.

4.1 Density estimation with the geometrically aware grid

The functions for generating and refining a interaction based grid were already implemented. Additionally the routine of the sparse grid density estimation was implemented as well for linear basis functions. So I simply had to replace the calls to the regular generation and refinement methods with the interaction based ones.

I also split up the routine of the density estimation to the part that decomposes and stores the matrix and the part that deals with training and evaluation. This allowed me to use the same matrix multiple times without needing to recompute the decomposition.

4.2 $L2$ Scalar Product for Modified Linear Basis

The sparse grid density estimation requires the $L2$ scalar product. I had to extend the existing operation for the linear basis to incorporate the differences to the modified linear basis. The necessary modification can be derived with the Simpson rule, since the basis functions are piecewise linear.

4.3 Evaluation

The main computational cost arises from the evaluation basis functions. There are two different types of evaluation. The standard evaluation returns the density function for each data point. This is needed to classify data or for some refinement functors. The transposed evaluation returns the sum of each basis functions, which is needed for the right side b of the linear system (2.17). Both of these evaluations can be implemented in a similar fashion, only differing on where the elemental evaluations are summed up to.

SG++ already offers implementations for the evaluation, however they do not make use of the properties of modified linear basis functions and the interaction terms.

The basic idea of Algorithm 1 is to iterate over all subspaces that model a interaction, for each interaction. According to (3.4) we know that the

Algorithm 1 Evaluation with modified linear basis and interaction terms

1: **for** datapoints $d \in D$ **do**
2: **for** interaction t in T **do**
3: $l \leftarrow$ Vector of size $|t|$
4: Set all l to 0
5: $relLevel \leftarrow |l|_1$
6: **while** a point was computed **or** $|l|_1$ did not change **do**
7: **for all** $dim \notin t$ **do**
8: g.set(dim,1,1)
9: **end for**
10: **for all** $dim \in t$ **do**
11: $level \leftarrow l_{dim} + 2$
12: $index \leftarrow 1 + 2\lfloor x_{dim} 2^{level-1} \rfloor$
13: g.set(dim, $level$, $index$)
14: **end for**
15: g.hash()
16: **if** $g \in Grid$ **then**
17: $eval \leftarrow \alpha_g$
18: **for all** $dim \in t$ **do**
19: $eval \leftarrow eval \cdot \phi_g(x_{dim})$
20: **end for**
21: $result[d] \leftarrow result[d] + eval$ ▷ $result[g]$ for transposed
22: **end if**
23: Get next l with $|l|_1 = relLevel$
24: **if** not possible do $relLevel$++ first
25: **end while**
26: **end for**
27: **end for**

l_1	l_2	l_3	
0	0	0	$\|l\|_1 = 0$
0	0	1	
0	1	0	$\|l\|_1 = 1$
1	0	0	
0	0	2	
0	1	1	
0	2	0	$\|l\|_1 = 2$
1	0	1	
1	1	0	
2	0	0	

Table 1: Level surplus for a ternary interaction

level of the dimensions not in the interaction have to be 1. Since the other dimensions are of level 2 or greater, I will set them to 2 and add a level surplus to them. The surplus is stored in the vector l. During the iteration we gradually increase the level of subspaces we look at. This is done by fixing $|l|_1$. We start with $|l|_1 = 0$ which only leaves us the surplus vector $l = \vec{0}$. For higher levels we can interpret l as the digits of a number of base $|l|_1 + 1$. This allows a simple traversal of all subspaces for a given level by incrementing the number l represents. The development of l for a interaction of size 3 is shown in Table 1. With this interaction traversal of subspaces, grid points that cannot be in the Grid according to the interaction terms are not iterated over. This drastically reduces the number of hash operations needed, since we need to hash each grid point before checking if it is included in the grid. Another improvement is made in the loop starting at line 16. Instead of evaluating $\phi(x)$ in each dimension, only the dimensions in the interaction are evaluated, since the other dimensions will evaluate to one. The runtimes are plotted in Figure 4.1. We can see that Algorithm 1 performs at a faster order. This allows us to reach higher dimensions.

4.4 Data preprocessing

All data preprocessing was done in Python. Basic image manipulation like resizing images was performed using NumPy [vdWCV11] and OpenCV [Bra00].

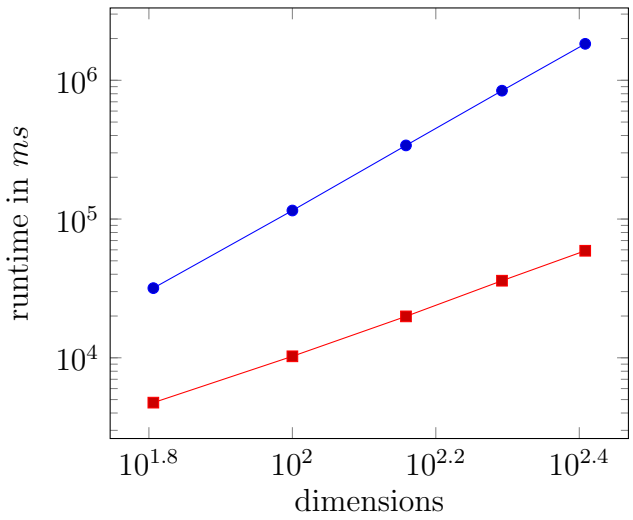


Figure 4.1: Log-log plot of the runtime. The blue line represents the standard evaluation, the red one the modified evaluation of Algorithm 1. 60 000 points were evaluated on a geometrically aware of level 3. The underlying stencil was NN with added Diagonals. Image size ranges from 8x8 to 16x16 in increments of two.

Gridtype	Imagesize	Gridpoints	Trainerror	Testerror
DN	8x8	833	31.41%	30.28%
DNDiag	8x8	1 225	36.18%	35.23%
regular SG	8x8	8 449	25.15%	24.36%
DN	28x28	10 753	27.50%	26.39%
DNDiag	28x28	16 585	28.94%	28.31%

Table 2: Classification error of different Stencils and Resolutions on the digits 2, 5 and 7 of the MNIST dataset. $\lambda = 10^{-5}$

5 Classification Results

In this section I will present the application of the geometrically aware grid. All examples are done with sparse grid density estimation using the cholesky decomposition and modified linear basis functions. This method is applied to the MNIST and CIFAR-10 datasets.

5.1 MNIST Dataset

Figure 5.1: Samples of the MNIST dataset.

The MNIST dataset is one of the most common benchmark sets for image classification [LBBH98]. The dataset consists of 60000 images of handwritten digits. Each image is of size 28x28 with only a single grayscale channel. The pixel values range from 0 (white) to 255 (black). To fit the data range to a sparse grid, we can simply divide each value by 255.

With the images scaled to different resolutions and a geometrically aware grid the results in Table 2 could be obtained. It is noticeable that the accuracy decreases even though more grid points are added to the model by including the diagonals. When we look at the corresponding confusion matrices in 3, we see that the class 7 is more dominant than 2 and 5. When adding the diagonals to the stencil, this effect only increases. This might be a problem of the interaction terms.

That class with the least variance tends to be the most dominant. This effect can be observed by stepwise removing the dominant class from the classification problem. With this stepwise removal we obtain following order of dominant classes: 1, 9, 7, 4, 6, 8, 3, 5, 2, 0 with 1 being the most dominant

		predicted		
		2	5	7
real	2	659	43	330
	5	2	374	516
	7	3	0	1 025

		predicted		
		2	5	7
real	2	578	37	417
	5	1	308	583
	7	2	0	1 026

Table 3: Confusion matrices of 8x8 Images. On the right Nearest Neighbor stencil. Added Diagonals on the left

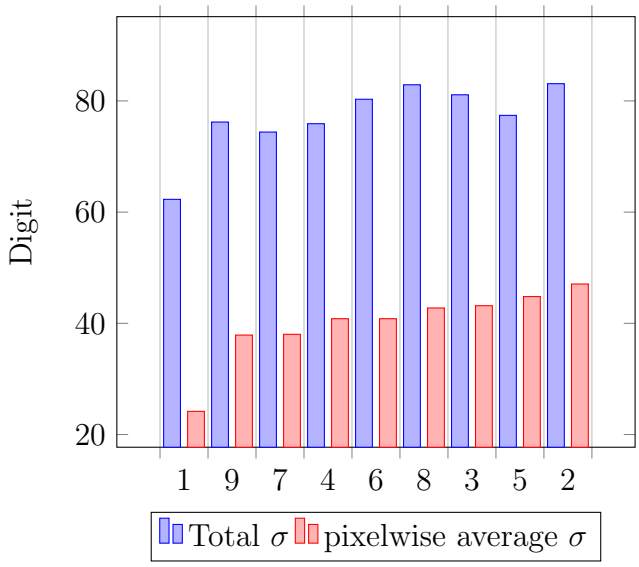


Figure 5.2: Total standard deviation and pixelwise standard deviation of the digits, sorted by dominance of the class. We see the correlation that more densely distributed digits are more dominant.

and 0 the least (see Tables 7 to 14). In Figure 5.2 we see that this ordering is very close to the ordering by standard deviation and even closer to the ordering by average pixelwise standard deviation. The pixelwise standard deviation can be seen in Figure 5.4.

When we only classify digits that have similar standard deviation, like 4 and 7, the dominant behavior of one class decreases. In the next paragraph I will have a look at the classification of those two digits.

First I will try to find a optimal λ. For this I look at the accuracy of a level 3 geometrically aware grid containing only the interactions between direct neighbors. In Figure 5.3 we see that $\lambda = 1$ is a good choice. With this λ I will compute the classification for different stencils and image resolutions. A better choice for λ may exist for each modification. However it is compu-

Stencil	image size	gridlvl	gridpoints	testacc (%)	trainacc (%)
DN	8x8	3	833	93.28	92.09
DNDiag	8x8	3	1 225	92.59	91.27
RegSG	8x8	3	8 449	96.57	95.79
DN	8x8	4	3 137	88.86	88.51
DNDiag	8x8	4	5 097	87.91	87.32
Square	8x8	4	6 665	82.64	83.35
DN	16x16	3	3 457	94.03	92.88
DNDiag	16x16	3	5 257	92.89	92.05
DN	28x28	3	10 753	94.88	93.94
DNDiag	28x28	3	16 585	93.88	93.07

Table 4: Accuracy of the geometrically aware grid for different stencils and resolutions. The digits 4 and 7 of the MNIST dataset were classified.

tationally not feasible to optimize λ every time, especially for large images. With $\lambda = 1$ the far more promising results in Table 4 could be achieved. There still is a decreace while adding additional interactions or even using a higher level. However the loss if far less then for the digits 2, 5 and 7 in Table 2.

A step to solve the dominance problem may be the normalization of the density functions. The integral of each function is already one, by solving the linear system. However through random sampling I discovered, that the density functions include very large negative evaluations of up to -60. The dominant class tends to have the most extreme minimum. In this case, normalizing the density functions might require to raise the function by the minimum of the function and then scaling it so that the integral is 1, instead of just scaling.

5.2 CIFAR-10

CIFAR-10 is a dataset of 60 000 pictures divided equally on 10 different classes [Kri09]. In Figure 5.5 we can see example pictures of all the classes. Currently the best classifier reaches an error rate of 0.21% using Neural Networks [WZZ+13]. The images are 32x32 pixel with 3 color channels. Although it is hard to classify some of these fairly small pictures as a human, current state of the art systems manage classification accuracies of up to 96.53% using Neural Networks [Gra14]. However as we can see in Table 5

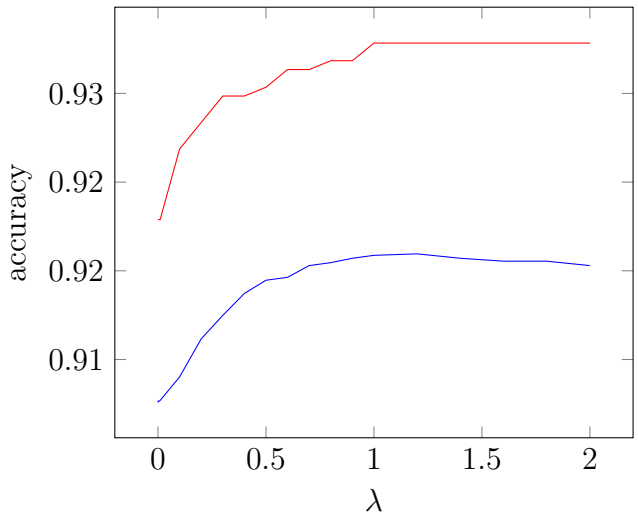


Figure 5.3: Test (red) and training (blue) compared to λ for the digits 4 and 7 on a 8×8 sparse grid of level 3 with the direct neighbor stencil. At $\lambda = 1$ the test accuracy plateaus, while the training accuracy decreaces. Hence $\lambda = 1$ is a good choice for this problem.

Figure 5.4: Pixelwise standard deviation. Digit 1 only has few pixel with a high deviation, while Digits 0 and 2 have many varying pixel.

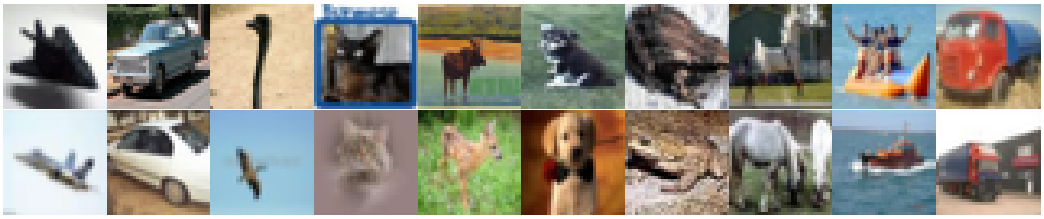


Figure 5.5: Samples images of the 10 classes of the CIFAR dataset. From left to right: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

the geometrically aware density estimation does not perform well on this dataset. This time however we do not have this strong dominance effect like for the MNIST dataset. Since the data points in each class are not simply sight modification like for the handwritten images, the data points are distributed very sparsely over the entire domain. This makes density estimation a difficult task. Even by using the gray scale images of Figure 5.6, did not improve the accuracy. The results for the gray scale images are in Table 6.

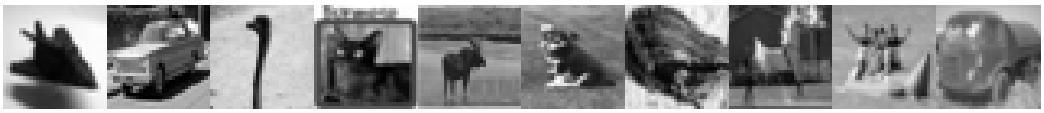


Figure 5.6: Samples grayscale images of the 10 classes of the CIFAR dataset. From left to right: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

stencil	image size	grid lvl	grid points	test accuracy (%)
regular SG	8x8	2	385	24.86
DN NoCol	8x8	3	3 673	28.01
DN Col	8x8	3	4 441	28.07
regular SG	16x16	2	1 537	25.10
DN NoCol	16x16	3	13 441	28.48

Table 5: Sparse grid density estimation on the CIFAR-10 dataset for $\lambda = 1$.

stencil	image size	grid lvl	grid points	test accuracy (%)
DN	8x8	3	833	25.28
DNDiag	8x8	3	1 225	25.76
DN	16x16	3	3 457	25.76
DNDiag	16x16	3	5 257	24.61

Table 6: Sparse grid density estimation on the grayscale images of the CIFAR-10 dataset for $\lambda = 1$.

6 Conclusion

There are two major assumptions on which the effectiveness of geometrically aware sparse grid density estimation is based. Firstly we assume that most of the information of a image lies in the comparison of neighboring pixel. This however generally is not the case. Secondly sparse grid methods work best with smooth functions. However image date will not necessarily be distributed smoothly. This seem to somewhat be the case for the MNIST dataset. The pictures of the CIFAR dataset however are more sparsely distributed.

Even though we made these assumptions, we still managed to reach an accuracy of 94.88% for the digits 4 and 7 of the MNIST dataset using all 784 pixel as input values. This is a large step from the regression of 166 dimensions of the Musk1 dataset by Pflüger in 2010 [Pfl10].

The presented method however currently still faces issues that have to be solved, before it is a reliable tool to classify images.

7 Future research

To justify geometrically aware grid a method has to be developed that avoids the dominance effect seen in the MNIST dataset. The described method of geometrically aware sparse grids only achieves a comparison between neighboring pixels. As possible way to avoid this one or more coarsened images can be added as additional dimensions. On every scaled down image the same method of geometrically aware sparse grids is applied, comparing neighboring pixels of the scaled image. Now there is a comparison between pixels that would not have been compared in the stencil of the base image.

Additionally current state of the art classifier use image augmentation and committees of classifier to reach top results. For more detail I will refer to [CMS12] and [Rom16]. These methods could be directly applied to a sparse grid method.

Further research may include the application of geometrically aware grids to non image data. A possible input are temperature measurement at certain points.

I personally think the future of image classification with sparse grids is in the development of sparse grid neural networks. A sparse grid containing only the dimensions as interaction terms and not any combination of them is fairly similar to a typical neural net with one output node and no hidden layer. The difference is that sparse grids use piecewise linear basis functions whereas neural nets typically use a single linear function. With the use of polynomial function backward propagation should be feasible as well.

8 Apendix

482	0	0	1	62	9	147	34	18	227
0	0	0	0	0	0	0	0	0	0
5	0	331	100	52	1	193	133	133	84
0	0	1	563	1	4	29	204	56	152
0	0	0	0	445	0	12	25	0	500
0	0	0	90	31	165	62	192	37	315
0	0	0	0	66	4	817	7	1	63
0	0	1	0	5	0	1	949	3	69
1	0	0	11	3	4	34	137	380	404
0	0	0	2	7	0	3	150	8	839

Table 7: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digit 1 is removed from the problem.

484	0	0	4	100	12	253	71	56	0
0	0	0	0	0	0	0	0	0	0
5	0	332	102	82	1	209	158	143	0
0	0	1	592	10	4	39	294	70	0
0	0	0	0	832	0	26	124	0	0
0	0	0	103	112	207	93	334	43	0
0	0	0	0	77	8	850	18	5	0
0	0	2	0	11	0	1	1011	3	0
1	0	0	17	34	4	56	367	495	0
0	0	0	0	0	0	0	0	0	0

Table 8: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9 are removed from the problem.

495	0	0	4	138	14	269	0	60	0
0	0	0	0	0	0	0	0	0	0
5	0	335	140	120	1	242	0	189	0
0	0	1	724	107	6	54	0	118	0
0	0	0	0	954	0	27	0	1	0
0	0	0	130	268	333	109	0	52	0
0	0	0	0	89	8	855	0	6	0
0	0	0	0	0	0	0	0	0	0
1	0	0	25	187	4	72	0	685	0
0	0	0	0	0	0	0	0	0	0

Table 9: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7 are removed from the problem.

517	0	0	6	0	41	338	0	78	0
0	0	0	0	0	0	0	0	0	0
5	0	348	143	0	4	335	0	197	0
0	0	2	760	0	13	91	0	144	0
0	0	0	0	0	0	0	0	0	0
0	0	0	136	0	495	186	0	75	0
1	0	0	0	0	11	938	0	8	0
0	0	0	0	0	0	0	0	0	0
1	0	1	32	0	5	113	0	822	0
0	0	0	0	0	0	0	0	0	0

Table 10: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7,4 are removed from the problem.

598	0	2	35	0	71	0	0	274	0
0	0	0	0	0	0	0	0	0	0
6	0	572	182	0	38	0	0	234	0
0	0	3	820	0	29	0	0	158	0
0	0	0	0	0	0	0	0	0	0
0	0	0	204	0	574	0	0	114	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
2	0	1	71	0	8	0	0	892	0
0	0	0	0	0	0	0	0	0	0

Table 11: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7,4,6 are removed from the problem.

634	0	3	131	0	212	0	0	0	0
0	0	0	0	0	0	0	0	0	0
8	0	640	307	0	77	0	0	0	0
0	0	6	918	0	86	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	2	225	0	665	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Table 12: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7,4,6,8 are removed from the problem.

654	0	5	0	0	321	0	0	0	0
0	0	0	0	0	0	0	0	0	0
8	0	885	0	0	139	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
2	0	7	0	0	883	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Table 13: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7,4,6,8,3 are removed from the problem.

824	0	156	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
15	0	1017	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Table 14: Confusion matrix of the density estimation with the DN stencil on 8x8 MNIST images. Digits 1,9,7,4,6,8,3,5 are removed from the problem.

level	1	2	3	4	5
direct neighbors, 1 color	1	2049	14081	54017	165633
added diagonals, 1 color	1	2049	21769	92457	296329
direct neighbors, 3 colors	1	6145	54529	231681	763137
added diagonals, 3 colors	1	6145	77593	347001	1155225
square selection, 1 color	1	2049	21769	123209	526969

Figure 8.1: gridpoints by stencil for a 32x32 image

References

[BG04] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[BPZ08] Hans-Joachim Bungartz, Dirk Pflüger, and Stefan Zimmer. Adaptive sparse grid techniques for data mining. In H.G. Bock, E. Kostina, X.P. Hoang, and R. Rannacher, editors, *Modelling, Simulation and Optimization of Complex Processes 2006, Proc. Int. Conf. HPSC, Hanoi, Vietnam*, pages 121–130. Springer-Verlag, August 2008.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[CMS12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[GG01] Jochen Garcke and Michael Griebel. Data mining with sparse grids using simplicial basis functions. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 87–96. ACM, 2001.

[Gra14] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[Kha16] Valeriy Khakhutskyy. *Sparse Grids for Big Data: Exploiting Parsimony for Large-Scale Learning*. Dissertation, Institut für Informatik, Technische Universität München, Munich, December 2016.

[Kre16] Lukas Krenz. Integration of prior knowledge for regression and classification with sparse grids. Bachelor's thesis, Institut für Informatik, Technische Universität München, August 2016.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Peh13] Benjamin Peherstorfer. *Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques*. Dissertation, Department of Informatics, Technische Universität München, October 2013.

[Pfl10] Dirk Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Dissertation, Institut für Informatik, Technische Universität München, München, February 2010.

[PPB10] Dirk Pflüger, Benjamin Peherstorfer, and Hans-Joachim Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, 26(5):508—-522, October 2010. published online April 2010.

[PPB14] Benjamin Peherstorfer, Dirk Pflüge, and Hans-Joachim Bungartz. Density estimation with adaptive sparse grids for large data sets. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 443–451. SIAM, 2014.

[Rom16] V.V. Romanuke. Training data expansion and boosting of convolutional neural networks for reducing the mnist dataset error rate. *Research Bulletin of the National Technical University of Ukraine" Kyiv Politechnic Institute"*, (6):29–34, 2016.

[Sie16] Adrian Sieler. Refinement and coarsening of online-offline data mining methods with sparse grids. Bachelor's thesis, Fakultät für Mathematik, Technische Universität München, March 2016.

[vdWCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, 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.