Fast Incremental Image Reconstruction with CNN-enhanced Poisson Interpolation

Blaž Erzar
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana, Slovenia
be6384@student.uni-lj.si

Žiga Lesar
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana, Slovenia
ziga.lesar@fri.uni-lj.si

Matija Marolt
University of Ljubljana
Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana, Slovenia
matija.marolt@fri.uni-lj.si

ABSTRACT

We present a novel image reconstruction method from scattered data based on multigrid relaxation of the Poisson equation and convolutional neural networks (CNN). We first formulate the image reconstruction problem as a Poisson equation with irregular boundary conditions, then propose a fast multigrid method for solving such an equation, and finally enhance the reconstructed image with a CNN to recover the details. The method works incrementally so that additional points can be added, and the amount of points does not affect the reconstruction speed. Furthermore, the multigrid and CNN techniques ensure that the output image resolution has only minor impact on the reconstruction speed. We evaluated the method on the CompCars dataset, where it achieves up to 40% error reduction compared to a reconstruction-only approach and 9% compared to a CNN-only approach.

Keywords
Image reconstruction, numerical interpolation, multigrid method, convolutional neural networks, autoencoder.

1 INTRODUCTION

Image reconstruction is a process used to recover the complete data from the incomplete ones that form scattered data. Reconstruction can be applied to both three-dimensional point clouds and two-dimensional images. In this paper we focus on reconstructing images which are generated out of input scattered data. Hereafter, we refer to these images as *corrupted*, although the data may be missing for a variety of reasons, e.g., errors in transfer between different systems, missing data before the transfer.

Missing data can be the result of a desire to save time or space, since in some cases the generation of data is resource heavy. One such example is ray tracing for rendering three-dimensional data. Despite the current powerful graphics processors, the method is still time consuming. The reason for this is the need to simulate the reflections of the light rays for each pixel in order to calculate its colour in the final image. This need for computation power could be lowered by simulating only a small fraction of the rays and reconstructing the rest of the pixels.

Similar idea is used in *foveated rendering* [Jab+22], which is used in virtual reality. Here eye tracking is used to monitor where the user's view is focused. Most of the pixels on the screen are in the peripheral vision, where the sharpness is lower as in the central area. This means that fewer rays could be sent over those areas (or areas containing less details) and the reduction of image quality will not be noticed by the user.

We solve the reconstruction problem using Poisson interpolation, which allows us to use numerical methods for solving linear systems. These methods run fast on the GPU and also converge in small number of iterations, given we choose multigrid method used in this paper. Poisson interpolation allows us to use some other method for generating the first approximation of the solution, e.g. *ray tracing*. Image generated on a very small amount of rays could be interpolated to generate a fast preview or noise in scattered data could be removed since interpolation creates a smoother image. This way we would get a direct preview of the rendering process, since we can run reconstruction concurrently along with ray tracing.

2 THEORETICAL BACKGROUND

First we introduce some theoretical background and notation used in solving the reconstruction problem, which is solved using linear systems of equations.

2.1 Reconstruction

Let $\hat{\varphi} : \mathbb{R}^2 \to \mathbb{R}^3$ be an image made of colour pixels $\mathbf{u}_{ij} \in \mathbb{R}^3$. It is defined on a rectangular grid of size $n \times m$ in points (x_i, y_j):

$$\hat{\varphi}(x_i, y_j) = \mathbf{u}_{ij}, \tag{1}$$

for indices $i, j \in \mathbb{Z}$, $0 \le i < m$ and $0 \le j < n$. Distance between neighbouring pixels in x dimension is defined as $h_x = 1/(m-1)$ and analogous for the y dimension.

We have a corrupted image φ, which we want to reconstruct. Because the image is corrupted, it contains actual pixel values only for points in the set of boundary conditions R, $|R| \ll nm$. The corrupted image is then defined as

$$\varphi(x_i, y_j) = \begin{cases} \mathbf{u}_{ij}, & \text{if } (x_i, y_j) \in R, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

As the result of the reconstruction, we want a smooth image. We can model this using **Laplace's equation** $\Delta\varphi = \nabla^2\varphi = \varphi_{xx} + \varphi_{yy} = 0$, which is a partial differential equation [Str07b]. Multigrid solver actually solves the nonhomogenous version of this equation – **Poisson's equation**:

$$\Delta\varphi = f, \tag{2}$$

which we will be solving from here on out.

2.2 Linear system

To be able to write Poisson's equation as a system of linear equations, it needs to be discretized first. This can be achieved using *finite difference* method for approximations of the partial derivatives. Using *backward difference* followed by the *forward difference*, the second partial derivative φ_{xx} can be approximated as

$$\varphi_{xx} \approx \frac{\varphi(x_i + h_x, y_j) - 2\varphi(x_i, y_j) + \varphi(x_i - h_x, y_j)}{h_x^2}$$

an analogous for φ_{yy}.

Using these approximations we can evaluate the left-hand side of (2) where we use h instead of h_x and h_y since grid size in both dimensions is the same:

$$\begin{aligned} \Delta\varphi(x_i, y_j) \approx \frac{1}{h^2}[&\varphi(x_i + h, y_j) + \varphi(x_i - h, y_j) \\ &+ \varphi(x_i, y_j + h) + \varphi(x_i, y_j - h) \\ &- 4\varphi(x_i, y_j)]. \end{aligned} \tag{3}$$

This way we get *five-point centered approximation of the Laplacian* that can also be evaluated using a convolution with a 2D kernel.

By approximation (3) and (1) – which also hold for the function f – we write the **discrete Poisson's equation**:

$$\frac{\mathbf{u}_{i+1,j} + \mathbf{u}_{i-1,j} + \mathbf{u}_{i,j+1} + \mathbf{u}_{i,j-1} - 4\mathbf{u}_{ij}}{h^2} = \mathbf{f}_{ij} \tag{4}$$

that can be solved as a linear system.

After multiplying (4) by $-h^2$ on both sides, we write the system $\mathbf{Au} = \mathbf{b}$. Vectors $\mathbf{u}$ and $\mathbf{b}$ are formed by writing the image and function $\mathbf{f}$ as a vector row wise:

$$\begin{aligned} \mathbf{u} &= [\mathbf{u}_{11}, \mathbf{u}_{21}, \dots, \mathbf{u}_{m1}, \mathbf{u}_{12}, \dots, \mathbf{u}_{m2}, \dots, \mathbf{u}_{mn}]^T, \\ \mathbf{b} &= -h^2[\mathbf{f}_{11}, \mathbf{f}_{21}, \dots, \mathbf{f}_{m1}, \mathbf{f}_{12}, \dots, \mathbf{f}_{m2}, \dots, \mathbf{f}_{mn}]^T. \end{aligned}$$

The matrix of the system $\mathbf{A}$ is a tridiagonal block matrix [GV96]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{C} & -\mathbf{I} & & \\ -\mathbf{I} & \mathbf{C} & \ddots & \\ & \ddots & \ddots & -\mathbf{I} \\ & & -\mathbf{I} & \mathbf{C} \end{bmatrix} \in \mathbb{R}^{n \times m},$$

where $\mathbf{I} \in \mathbb{R}^{m \times m}$ denotes the identity matrix and $\mathbf{C}$ the tridiagonal matrix:

$$\mathbf{C} = \begin{bmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix} \in \mathbb{R}^{m \times m}.$$

Matrix $\mathbf{A}$ can also be decomposed as a sum of three matrices, diagonal matrix $\mathbf{D}$, lower triangular matrix $\mathbf{L}$ and upper triangular matrix $\mathbf{U}$:

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}. \tag{5}$$

This decomposition is used in defining iterative methods.

3 RELATED WORK

The linear system $\mathbf{Au} = \mathbf{b}$ can be solved using two different approaches. *Direct methods* [Wen17, chap. 3] solve it in a finite number of steps. Examples of direct methods are Gaussian elimination, LU decomposition, pivoting etc. These methods always return a solution – as long as the system has a solution – but they have high time and space complexity. The latter is in this case more problematic, since the matrix $\mathbf{A}$ is a very sparse matrix. For this system of size nm, the Gaussian elimination has time complexity $\mathcal{O}(n^3m^3)$ and space complexity $\mathcal{O}(n^2m^2)$. On the other hand, *iterative methods* [Wen17, chap. 4] solve the system by generating approximate solutions that converge towards the final one. In each iteration, the next approximation is generated from the previous one. For a system of size nm each iteration typically has time complexity $\mathcal{O}(n^2m^2)$. This means that the iterative methods returns a solution faster than the direct method, as long as a good solution is obtained in less than nm steps. Moreover, to use iterative methods, we do not need to explicitly generate the matrix $\mathbf{A}$, because we only need it to derive a formula to generate the next approximations.

Defining iterative methods as in [GQ20], each iterative method has its *iteration matrix* $\mathbf{B}$ and vector $\mathbf{c}$, which are used to calculate every successive approximation given the previous one:

$$\mathbf{u}_{k+1} = \mathbf{B}\mathbf{u}_k + \mathbf{c}, \quad k \in \mathbb{N}_0.$$

The sequence of approximation converges towards the true solution $\tilde{\mathbf{u}}$, which minimizes the *residual* defined for the approximation $\mathbf{u}$ as

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{u}.$$

3.1 Jacobi method

One of the simplest iterative methods is the *Jacobi method* [DF18]. Its iteration matrix $\mathbf{B}$ and vector $\mathbf{c}$ are

$$\mathbf{B} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}),$$
$$\mathbf{c} = \mathbf{D}^{-1}\mathbf{b}.$$

Given this we can write down the rule for updating pixel values:

$$\mathbf{u}_{ij}^{(k+1)} = \frac{1}{4}\left(\mathbf{u}_{i-1,j}^{(k)} + \mathbf{u}_{i,j-1}^{(k)} + \mathbf{u}_{i+1,j}^{(k)} + \mathbf{u}_{i,j+1}^{(k)} - h^2\mathbf{f}_{ij}\right).$$

3.2 Gauss-Seidel method

The *Gauss-Seidel method* [DF18] is similar to the Jacobi method, but it has slightly better convergence. This is achieved by using values from iteration $k+1$ when calculating the values of iteration $k+1$. The matrix $\mathbf{B}$ and vector $\mathbf{c}$ are

$$\mathbf{B} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U},$$
$$\mathbf{c} = (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b},$$

while the update rule is

$$\mathbf{u}_{ij}^{(k+1)} = \frac{1}{4}\left(\mathbf{u}_{i-1,j}^{(k+1)} + \mathbf{u}_{i,j-1}^{(k+1)} + \mathbf{u}_{i+1,j}^{(k)} + \mathbf{u}_{i,j+1}^{(k)} - h^2\mathbf{f}_{ij}\right).$$

It is almost exactly the same as Jacobi's update, but here the first two terms are from the currently calculated iteration $k+1$ instead of the already calculated iteration k.

3.3 Successive over-relaxation

By introducing the relaxation parameter ω the *successive over-relaxation (SOR)* [QSS07] can be derived, whose matrix $\mathbf{B}$ and vector $\mathbf{c}$ are

$$\mathbf{B} = -(\mathbf{D} + \omega\mathbf{L})^{-1}[(\omega - 1)\mathbf{D} + \omega\mathbf{U}],$$
$$\mathbf{c} = (\mathbf{D} + \omega\mathbf{L})^{-1}\omega\mathbf{b}.$$

While deriving the update rule it can be shown, that the next SOR approximation is actually a linear combination of the previous approximation and approximation calculated using the Gauss-Seidel method:

$$\mathbf{u}_{ij}^{(k+1)} = (1-\omega)\mathbf{u}_{ij}^{(k)} + \omega\mathbf{u}_{ij,GS}^{(k+1)}.$$

By theorem 9.6 in [GQ20], the method converges under the condition $0 < \omega < 2$.

3.4 Conjugate gradient method

The *conjugate gradient method (CG)* [Wen17, chap. 6] is different from the previous ones, since it is not derived from the matrix decomposition (5). It is based on the same idea as gradient descend.

In every iteration the next approximation is calculated by

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k\mathbf{p}_k,$$

which represents a move in the direction of the **conjugate gradient** $\mathbf{p}_k$, that is defined to be $\mathbf{A}$-conjugate to all other conjugate gradients. After defining $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{u}_k$ as the residual of the current approximation, the length of the next move can be calculated using

$$\alpha_k = \frac{\mathbf{p}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}.$$

The next residual can then be calculated as

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$$

and the next conjugate gradient as

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$$

using

$$\beta_k = \frac{\mathbf{r}_{k+1}^T\mathbf{r}_{k+1}}{\mathbf{r}_k^T\mathbf{r}_k}.$$

It is chosen that $\mathbf{p}_0 = \mathbf{r}_0$, like in the gradient descend case.

3.5 Neural networks

Recently many new neural networks based on diffusion have been released. Models like Stable Diffusion [Rom+21], DALL-E [Ram+22] or Imagen [Sah+22] take a text prompt and generate an image based on its input. Some of these models even allow users to input an image and generate similar images. This approach could be used for some sort of reconstruction, but these models are usually very big. Stable Diffusion for example has around 890 million parameters, but some models are even bigger. Only the forward pass on these models requires a powerful computer. In contrast the model developed for this paper has less than a million parameters and can be run on a desktop computer with a dedicated graphics card. A smaller network runs faster and can be used alongside a fast reconstruction method.

4 METHOD

4.1 Multigrid solver

The problem of the methods described till now is slow convergence, which is the consequence of errors consisting of high and low frequencies. High frequencies are removed in a few iterations, while the low ones are being removed slowly. The idea of the *multigrid method (MG)* [Str07a, chap. 7.3] is to use grids of multiple resolutions, where low frequencies become high. The multigrid method is not a standalone method, but rather a high-level scheme for using the existing relaxation methods.

Since the method operates on grids of different resolutions, an operation is needed which generates a grid of lower resolution – **coarse grid** – from a higher resolution grid – **fine grid** – and the other way around. For simplicity, we will restrict ourselves to $n \times n$ quadratic images whose size is a power of 2. This means that we can reduce the image down to a 1×1 grid.

It is a recursive method, as it is also used to solve systems at lower levels. Depending on the order in which these systems are solved, we obtain different iteration schemes called *cycles*.

4.1.1 Restriction

The operation that reduces the size of the grid is called *restriction*. For an image $\mathbf{u}$ (or residual $\mathbf{r}$), it returns an image $\mathbf{u}'$, which has a fourth of the input image pixels:

$$\text{restriction} : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2/4}.$$

It works by calculating one pixel value in the restricted image as the mean of four pixel values from the input image. This operation can be performed efficiently on a GPU.

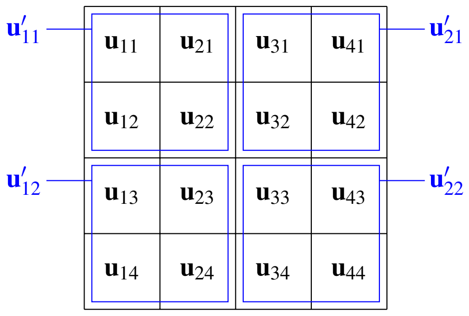

Figure 1: Restriction on image of size 4×4.

4.1.2 Interpolation

To increase the size of the grid, *interpolation* operation is used, which creates a grid twice the resolution:

$$\text{interpolation} : \mathbb{R}^{n^2} \to \mathbb{R}^{4n^2}.$$

This is done using bilinear interpolation. This operation can be performed efficiently on a GPU.

4.1.3 Boundary conditions

In single-grid methods the boundary conditions are simple to account for – we do not update the pixels $\mathbf{u}_{ij}$ for which $(x_i, y_j) \in R$, but we still use them for updating other pixels. In the multigrid method, this only works on the first grid level. On other levels, a different system is solved, which also has different dimensions. As we will see, this is an error system, so the boundary conditions are homogeneous – their value is 0.

However, because of the different dimensions, we have a problem, because we do not know which pixels fall under the boundary conditions. The solution, as proposed in [GT11], is that a point on the coarse grid becomes a boundary condition if it has been constructed from at least one boundary condition point on the fine grid in the restriction process, as it can be seen in Figure 2.

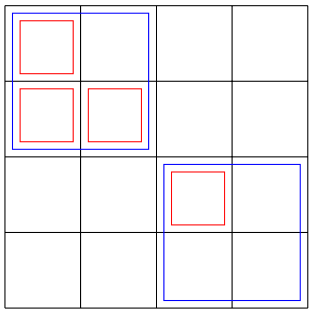

Figure 2: Example of boundary conditions on fine (red) and coarse (blue) grid.

Implementing this logic is fairly simple. Along the image $\mathbf{u}$ we also need an image $\mathbf{m}$ of the same dimensions, which holds the boundary conditions. Its elements are -1 where pixel is a boundary condition and 1 everywhere else. During the iteration of the multigrid method, we also apply restriction operation over $\mathbf{m}$. The result is that for boundary conditions (x_i, y_j) on all levels, $\mathbf{m}_{ij} < 1$, as the result of restriction will be 1 only if it has been calculated from four pixels which are not a boundary condition and have a value of 1. If pixel is built from at least one pixel which is a boundary condition, the mean of elements of $\mathbf{m}$ during restriction will certainly be less than 1.

4.1.4 V-cycle

There exist different cycles (iteration schemes) of the multigrid method. The individual steps are the same for all cycles, but they differ in the order in which grids of different resolutions are considered. We use the V-cycle.

It is the simplest of all cycles and has very good convergence. Other cycles increase the running time of one iteration, but do not bring much improvement in convergence. We implement it on the GPU and it runs upwards of 100 times faster than the CPU implementation. Its steps are:

1. **Pre-smoothing:** Perform N_{sm} iterations of a single-grid method over the system $\mathbf{Au} = \mathbf{b}$.
2. **Restriction:** Restrict (downsample) the residual $\mathbf{r}$ to a coarse grid to get $\mathbf{r}'$.
3. **Solve:** We solve $\mathbf{Ae} = \mathbf{r}'$. If the size of grid is larger than n_{min}, the V-cycle is invoked recursively, otherwise N_{so} iterations of a single-grid method are performed.
4. **Interpolation:** Interpolate (upsample) solution $\mathbf{e}$ to a fine grid to obtain correction $\mathbf{p}$.
5. **Post-smoothing:** Perform N_{sm} iterations of smoothing over the improved solution $\mathbf{u}+\mathbf{p}$.

For the smoothing we use SOR implemented since its iteration takes similar amount of time as the simple Jacobi method, but converges much faster. We implement it on the GPU using *red-black ordering* [Str07a, p. 568], because we cannot read and write to the same GPU memory at the same time. The value of N_{sm} is 20, while N_{so} is 10. For n_{min} we take 1. Another thing we need to be careful about is step 5 – since the correction is interpolated, it does not take boundary conditions into account, so we only need to apply the correction to points that are **not** boundary conditions on the fine grid.

Algorithm 1 V-cycle

```
Input: $\mathbf{u}_k, \mathbf{f}, \mathbf{m}, n$
Output: $\mathbf{u}_{k+1}$

$\mathbf{u}_{k+1} \leftarrow smoothing(\mathbf{u}_k, \mathbf{f}, \mathbf{m}, n, N_{sm})$

$\mathbf{r} \leftarrow residual(\mathbf{u}_{k+1}, \mathbf{f}, \mathbf{m}, n)$
$\mathbf{r}' \leftarrow restriction(\mathbf{r})$

$\mathbf{e} \leftarrow$ empty image
$\mathbf{m}' \leftarrow restriction(\mathbf{m})$
if $n \leq n_{min}$ then
    $\mathbf{e} \leftarrow smoothing(\mathbf{e}, \mathbf{r}', \mathbf{m}', n/2, N_{so})$
else
    $\mathbf{e} \leftarrow V\text{-}cycle(\mathbf{e}, \mathbf{r}', \mathbf{m}', n/2)$
end if

$\mathbf{p} \leftarrow interpolation(\mathbf{e})$

$\mathbf{u}_{k+1}[\mathbf{m} == 1] \leftarrow \mathbf{u}_{k+1}[\mathbf{m} == 1] + \mathbf{p}[\mathbf{m} == 1]$
$\mathbf{u}_{k+1} \leftarrow smoothing(\mathbf{u}_{k+1}, \mathbf{f}, \mathbf{m}, n, N_{sm})$
```

The algorithms is written using pseudocode in Algorithm 1. Both functions *restriction* and *interpolation* take a single image to process as input, while functions *V-cycle* and *residual* take approximation $\mathbf{u}$, right side $\mathbf{f}$, boundary conditions $\mathbf{m}$ and image size n. Function *smoothing* takes the same parameters as *V-cycle* with another parameter for number of iterations to perform.

4.2 Detail recovery

Since the corrupted images contain only a small percentage of the original pixels, much of the detail in the image is lost despite the reconstruction. For the recovery of these details we have used a neural network. The neural network was trained with the reconstructed images at the input and the original images at the output. This gave us a model into which we could later feed new reconstructed images, resulting in images with recovered details. We used an autoencoder architecture depicted in Figure 3. In the figure, N stands for the number of filters, n for their size and m for down or upsampling size.

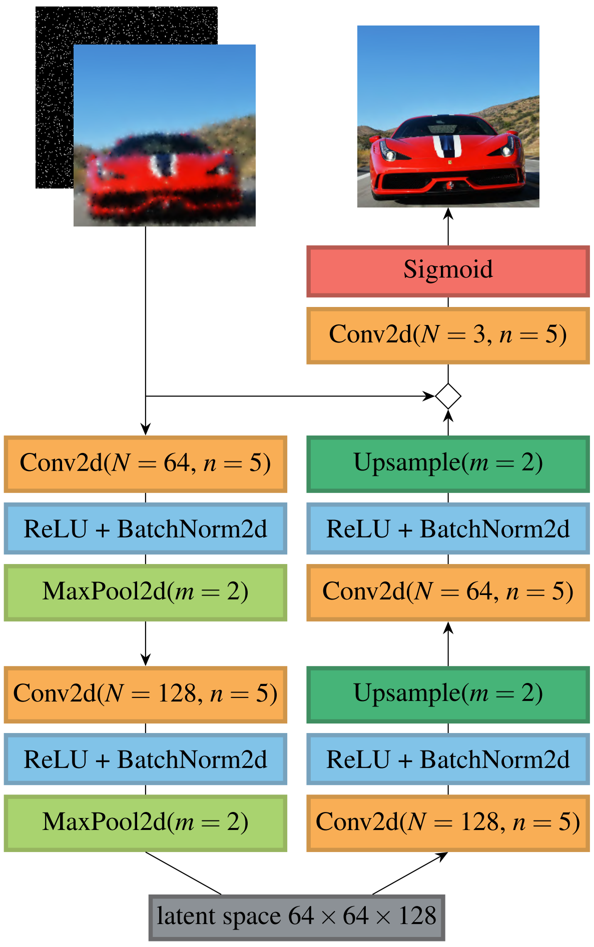

Figure 3: Architecture of the implemented neural network.

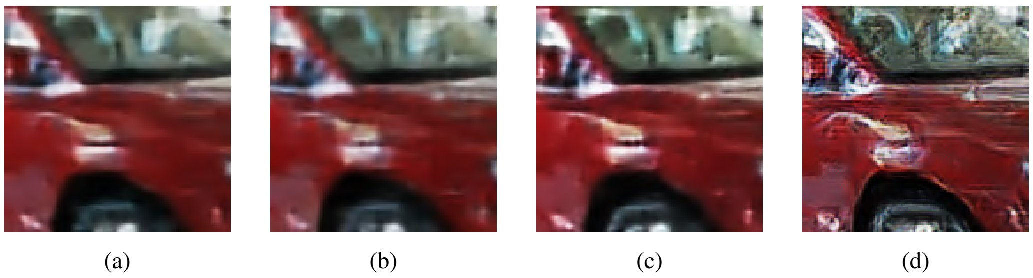

Figure 4: Outputs of model trained using (a) L1, (b) L2, (c) SSIM and (d) LPIPS loss.

For the training and evaluation we used the *CompCars Dataset* [Yan+15]. Out of all the available images, 27 thousand were used. They were cropped and scaled to size 256×256. Then we selected 5% of pixels for each image and generated the reconstructed image as well. An example of images used can be seen in Figure 3. The input into the network is the reconstructed image with an additional fourth channel – a binary mask representing the pixels selected as boundary conditions. On the output we put the original image.

For the loss function we chose the L1 loss, because it produced the least amount of image artifacts among of the loss functions considered. The comparison of four loss functions is shown in Figure 4. The last two loss functions, SSIM [Wan+04] and LPIPS [Zha+18], are actually perceptual metrics, but they do not provide better reconstruction results.

The dataset of images was split into train (80%), validation (10%) and test (10%) sets. The sets were then further divided into batches of 64 images. The Adam optimizer was used for model parameters optimization and the training lasted for 100 epochs. The validation set was used to select the best model, and the evaluation was performed on the test set.

5 RESULTS

We present the reconstruction and detail recovery results separately. First we evaluate the process of reconstruction of basic iterative methods in comparison with the multigrid solver and then show the capability of the neural network. At the end we also present a few examples of the full pipeline – reconstruction and detail recovery.

5.1 Reconstruction

We evaluated all reconstruction methods using the baboon image and two types of boundary conditions (see Figure 5). For the metric we used the *relative residual* defined as $\|\mathbf{r}_k\|/\|\mathbf{r}_0\|$.

We only use the baboon image, because we are only interested in the convergence process. All methods are solving the same linear system, which means they all converge to the same solution. The quality of the reconstruction is only dependent on the boundary conditions.

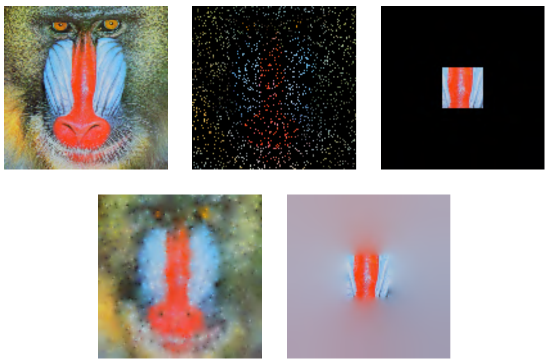

Figure 5: Images used for the evaluation of reconstruction methods: original (top left), corrupted with random boundary conditions (top middle), corrupted with center boundary conditions (top right), reconstructed with random, with center boundary conditions.

As shown in Figures 6 and 7, the multigrid method reduces the reconstruction error the fastest out of all compared methods. We evaluated the methods using two different boundary condition configurations, as shown in Figure 5, which results in vastly different performance. In both configurations, the multigrid method outperforms the other reconstruction methods.

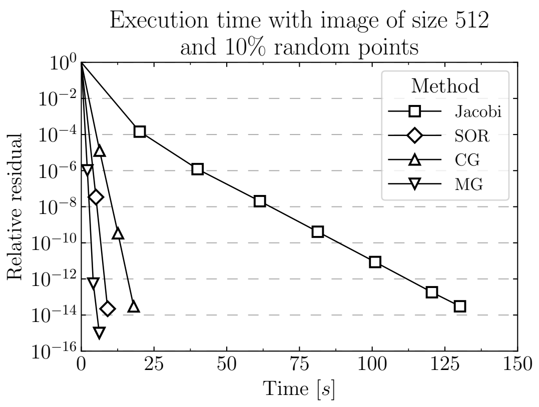

Figure 6: Comparison of reconstruction error reduction w.r.t. time, evaluated on random boundary conditions.

Another improvement given by the multigrid method is that the convergence is much less dependant on the image size when using center boundary conditions, which can be seen in Figure 8. Using other methods the recon-

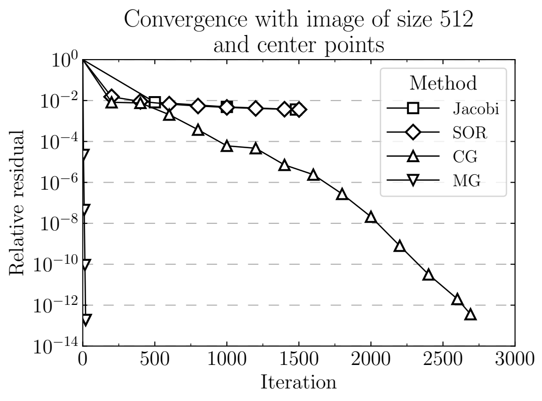

Figure 7: Comparison of reconstruction error reduction w.r.t. time, evaluated on center boundary conditions. Jacobi and SOR reconstruction were stopped early because of their slow convergence.

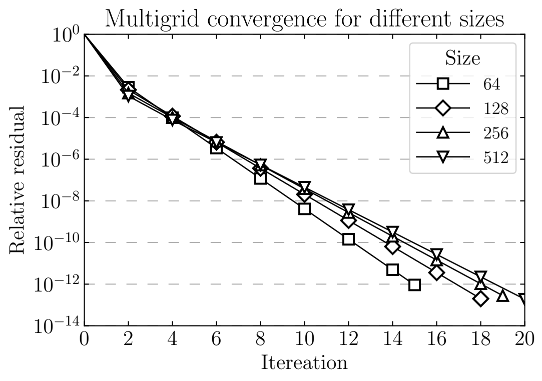

Figure 8: Comparison of multigrid reconstruction error reduction on different image sizes.

struction is propagated outward starting at the boundary conditions, but multigrid updates the whole image in one iteration.

5.2 Detail recovery

The results of the evaluation for the model trained on the reconstructed images are shown in Table 1 and its learning curve in Figure 9. Additionally, we trained the model directly on the corrupted images, but it achieved worse performance compared to training on the reconstructed images.

	Reconstructed			Corrupted	
Metric	Input	Prediction	Change	Input	Prediction
L1	0.066	**0.048**	27%	0.434	0.051
L2	0.013	**0.009**	31%	0.238	0.010
SSIM	0.575	**0.679**	18%	0.030	0.654
LPIPS	0.514	**0.308**	40%	1.082	0.339

Table 1: Values of four different metrics for the input images (reconstructed and corrupted) and predictions of both models.

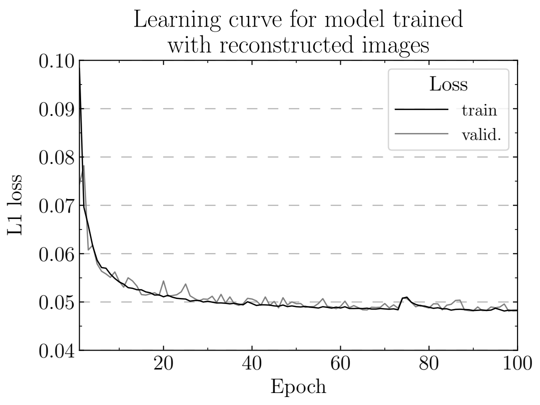

Figure 9: Model learning curve.

This can also be seen from the outputs of the models in Figure 10. Model trained on the reconstructed images produces images with less noise, better edge definition and localization. It also improves all images in the test set regarding all four used metrics.

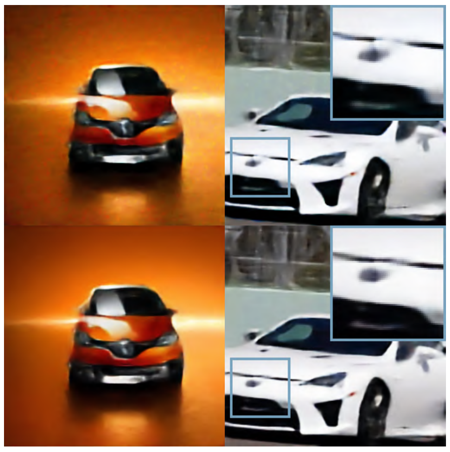

Figure 10: Outputs of model trained on corrupted (top) and reconstructed (bottom) images.

In Figure 12 results for two more instances are shown. They represent the reconstructed images which have been most and least improved by the model. As it can be seen, both predicted images show an improvement over the reconstructed ones which were input into the model.

At the end we show seven examples of the complete reconstruction and details restoration process in Figure 11. The corrupted images are first reconstructed and then fed into the model to produce the predicted images with restored details – these can then be compared to the original images. We show the outputs of both models, trained on corrupted and reconstructed images.

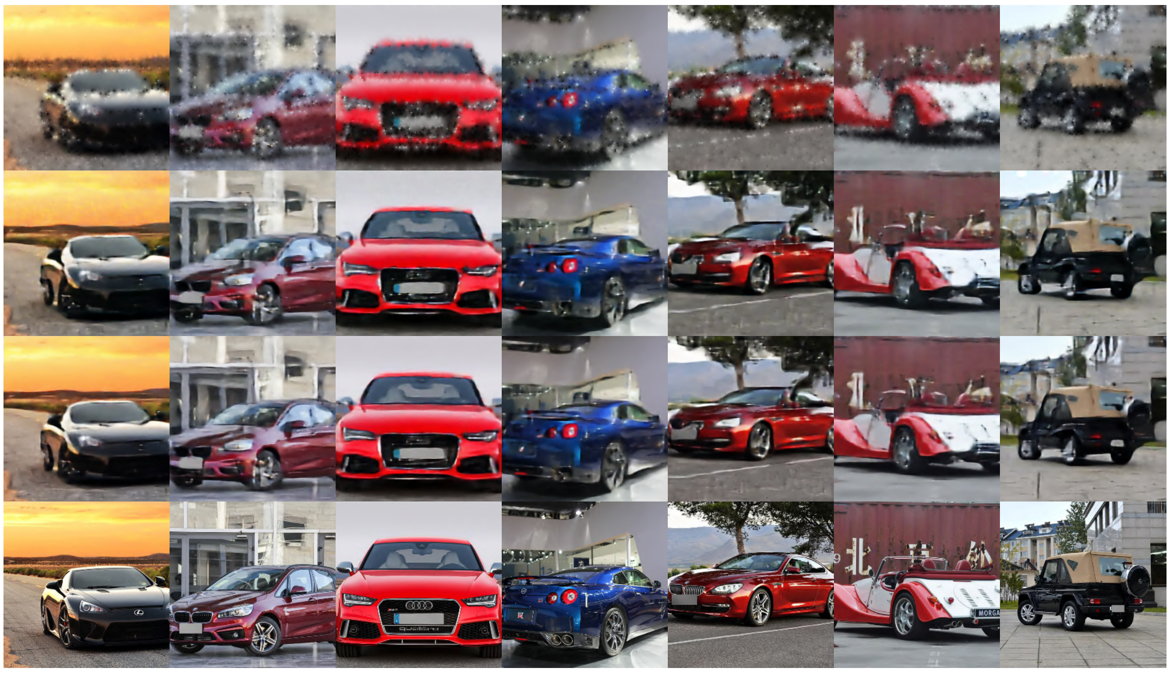
Figure 11: Comparison between reconstruction and models, from top to bottom: reconstruction, model trained with corrupted images, model trained with reconstructed images, original image.

6 CONCLUSION

In this paper we dealt with the problem of reconstruction from scattered data. We focused on images and presented usage of the multigrid method to solve the differential equation used to model this problem. This method is built upon the more basic iterative methods and improves their convergence rate, which becomes much less dependent on the image size.

Since the corrupted images contain only a small fraction of the original points, the reconstructed images contain less details. Because of this we also employed a neural network model, which is capable of restoring lost details. It is based on the autoencoder architecture and trained using a dataset of reconstructed images. The combined multigrid and neural network methods outperformed the individual methods in terms of reconstruction quality.

Using both of these methods, we can generate reconstructed images faster and improve the quality of the reconstruction itself, but we are not able to use the neural network on general images, because it was trained on only one domain. This is a classic problem of convolutional neural networks, which could be resolved by using a larger dataset containing images from multiple domains. This small dataset of cars was used only for illustration purposes. A further study using more diverse dataset like ImageNet [Den+09] would be beneficial.

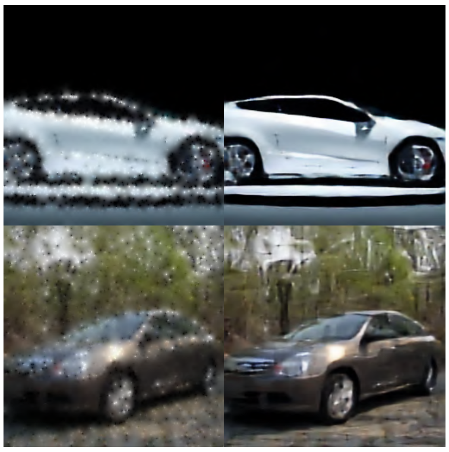
Figure 12: Reconstructed and predicted images for best (top) and worst (bottom) model improvement.

7 REFERENCES

[Den+09] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[DF18] Matías Di Martino and Gabriele Facciolo. “An Analysis and Implementation of Multigrid Poisson Solvers With Verified Linear Complexity”. In: *Image Processing On Line* 8 (2018), pp. 192–218. URL: https://doi.org/10.5201/ipol.2018.228 (visited on 01/15/2023).

[GQ20] Jean Gallier and Jocelyn Quaintance. *Linear Algebra and Optimization with Applications to Machine Learning: Volume I: Linear Algebra for Computer Vision, Robotics, and Machine Learning*. English. New Jersey: WSPC, Jan. 2020. Chap. 9. ISBN: 9789811207716.

[GV96] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)* USA: Johns Hopkins University Press, 1996, pp. 177–180. ISBN: 978-0-8018-5414-9.

[GT11] Thomas Guillet and Romain Teyssier. “A simple multigrid scheme for solving the Poisson equation with arbitrary domain boundaries”. In: *Journal of Computational Physics* 230.12 (June 2011). arXiv:1104.1703 [astro-ph, physics:physics], pp. 4756–4771. ISSN: 00219991. DOI: 10.1016/j.jcp.2011.02.044. URL: http://arxiv.org/abs/1104.1703 (visited on 01/15/2023).

[Jab+22] Susmija Jabbireddy et al. *Foveated Rendering: Motivation, Taxonomy, and Research Directions*. 2022. DOI: 10.48550/ARXIV.2205.04529. URL: https://arxiv.org/abs/2205.04529 (visited on 01/15/2023).

[QSS07] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. “Iterative Methods for Solving Linear Systems”. en. In: *Numerical Mathematics*. Ed. by Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. Texts in Applied Mathematics. Berlin, Heidelberg: Springer, 2007, pp. 126–132. ISBN: 978-3-540-49809-4. DOI: 10.1007/978-3-540-49809-4_4.

[Ram+22] Aditya Ramesh et al. “Hierarchical text-conditional image generation with clip latents”. In: *arXiv preprint arXiv:2204.06125* (2022).

[Rom+21] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV].

[Sah+22] Chitwan Saharia et al. “Photorealistic text-to-image diffusion models with deep language understanding”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36479–36494.

[Str07a] Gilbert Strang. *Computational Science and Engineering*. English. 1st edition. Wellesley, MA: Wellesley-Cambridge Press, Nov. 2007, pp. 283–284, 568, 571–583. ISBN: 978-0-9614088-1-7.

[Str07b] Walter A. Strauss. *Partial Differential Equations: An Introduction*. English. 2nd edition. New York: Wiley, Dec. 2007, pp. 165–172. ISBN: 978-0-470-05456-7.

[Wan+04] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[Wen17] Holger Wendland. *Numerical Linear Algebra: An Introduction*. Cambridge Texts in Applied Mathematics. Cambridge: Cambridge University Press, 2017. Chap. 3, 4, 6. ISBN: 978-1-107-14713-3. DOI: 10.1017/9781316544938.

[Yan+15] Linjie Yang et al. “A large-scale car dataset for fine-grained categorization and verification”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3973–3981. DOI: 10.1109/CVPR.2015.7299023.

[Zha+18] Richard Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.