Image restoration project

Alexey, Romanov Andjela, Todorović

13 December 2021

1 Introduction

Image restoration is the process of recovering an image from a degraded version—usually a blurred and noisy image. Image restoration is a fundamental problem in image processing, and it also provides a good ground for more general inverse problems. [10]

There are several approaches to the image restoration problem, starting from the traditional methods that revolve around the matrix completion problem, as far as the deep learning approaches (e.g. LDNN [13]) that are mainly based on the U-Net [11] backbone architecture.

In this report, we will tackle a special subclass of image restoration problem sometimes referred in the literature as pixel interpolation. The research approach as well as utilized algorithms to solve the given image restoration problem will be tackled in the following chapters.

Matrix completion problem is a promising technique which is able to recover an intact matrix with low-rank property from incomplete data. It is widely used in wireless communications, traffic sensing and video industry. One example is the movie-ratings matrix, as appears in the Netflix problem. The matrix of ratings is given, in which each entry (i, j) represents the rating of movie j by customer i, if customer i has watched movie j and is otherwise missing. The task is to predict the remaining entries in order to make good recommendations to customers on what to watch next.

The general problem is the following. The task of restoration matrix Y is considered. Let E be set of known indexes. In our case the values Y_{ij}, $\forall$(i, j) $\in E$ are known and Y_{ij} $\forall$(i, j) $\notin E$ are not known.

Without any additional information on matrix Y that task is trivial: each matrix X, that $X_{ij} = Y_{ij}, (i, j) \in E$ is the answer. There is unlimited number of that matrices. As a consequence, the task of completion of matrix is considered as the special case of the matrix approximation. The most popular formulation of the problem is

$$\begin{gathered} \min_X rk(X) \\ X_{i,j} = Y_{i,j}, \quad (i,j) \in E \end{gathered} \tag{1}$$

That task is NP-hard. In order to solve rank of the matrix is approximated with convex function of X. The standard choice is to take [14] trace norm.

$$\begin{gathered} \min_X ||X||_*, ||X||_* = \sum \sigma_i(X) \\ X_{i,j} = Y_{i,j}, \quad (i,j) \in E \end{gathered} \tag{2}$$

In order to solve the matrix completion problem, several algorithms of gradient boosting, as far as the deep learning approach were implemented. Some of them use proximal operator in order to make the problem convex. Others use proximal mapping. Deep learning approach is based on a simple autoencoder CNN architecture.

In the second chapter, these algorithms are explained and in the third chapter the results of that implementation are described.

2 Algorithms for matrix completion

All the algorithms deal with (2). The cost function in (2) is nondifferentiable and convex, and it is minimized by the subgradient methods [12]. A subgradient of (2) is computed from the singular value decomposition [3]. Let $X = P\Sigma Q^T$ be the singular value decomposition of X, with $P \in R^{p*r}$, $\Sigma \in R^{r*r}$, and $Q \in R^{q*r}$, where r is the rank of X. Then $P * Q^T$ is a subgradient of f at x. Since the main computation in each iteration in the subgradient method involves one singular value decomposition, the convergence of subgradient method is slower than in the next method.

In the second method the cost function is replaced with a smooth approximation. Then that approximation is minimized by a fast gradient method [8]. The smooth approximation of $||X||_*$ is obtained by taking SVD $X = \sum_{i=1}^{r} \sigma_i u_i v_i^T$. The singular values are replaced with huber loss function, in order to be able to get derivatives in gradient descent algorithm.

$$h_\mu(\sigma_i) = \begin{cases} \frac{\sigma_i^2}{2\mu} & \sigma_i \leq \mu \\ \sigma_i - \frac{\mu}{2} & \sigma_i > \mu \end{cases} \tag{3}$$

where μ is a small positive parameter. Define proximal operator [1]

$$f_\mu(X) = \sum_{i=1}^{r} h_\mu(\sigma_i) u_i v_i^T = sup_{||Y||_2 \leq 1} tr(X^T Y) - \frac{\mu * ||Y||_F^2}{2} \tag{4}$$

The gradient can be computed from (4). Actually, the main point of proximal operator is to make sure that the gradient are computed well. It means, that the proximal operator should be smooth.

$$\nabla f_\mu(X) = \begin{cases} \frac{X}{\mu}, & \text{if } \sigma_{max}(X) \leq \mu \\ 1, & \text{if } \sigma_{max}(X) > \mu \end{cases} \tag{5}$$

After that derivation fast nesterov's algorithm can be found in figure 1 [9].

For $k \geq 0$ **do**
1. Compute $f(x_k)$ and $\nabla f(x_k)$.
2. Find $y_k = T_Q(x_k)$.
3. Find $z_k = \arg\min_x \left\{ \frac{L}{\sigma} d(x) + \sum_{i=0}^{k} \frac{i+1}{2} [f(x_i) + \langle \nabla f(x_i), x - x_i \rangle] : \ x \in Q \right\}$.
4. Set $x_{k+1} = \frac{2}{k+3} z_k + \frac{k+1}{k+3} y_k$.

Figure 1: Nesterov's algorithm of fast GD

Where $T_Q(x)$ is the optimal solution of the following minimization problem. It is slight reformulation of the original task (2).

$$min_y \{ < \nabla f(x), y - x > + \frac{1}{2} L ||y - x||^2 : y \in Q \} \tag{6}$$

.

The input of the algorithm takes the initial matrix with deleted pixels and outputs for each iteration vector of restored matrix values.

Another algorithm is called FISTA [4]. The first difference with the previous method is that it has other proximal operator. The second is that there is no sum, as in the Nesterov's gradient algorithm, so the computations are much faster. It will be shown futher.

Implemenatation of FISTA's algorithm.
Initialization: $R_1 = 0, t_1 = 1$
for k = 1, 2, ..., T do

$$1.X_k = D(R_k - \frac{1}{L}A^*(A(R_k) - y))$$
$$2.t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2} \tag{7}$$
$$3.R_{k+1} = X_k + \frac{t_k - 1}{t_k + 1}(X_k - X_{k-1})$$

Original task:
$\hat{X} = argmin_X ||X||_*$,s.t $y = A(X)$
$\hat{X} = argmin_X \frac{1}{2}||y - A(X)||_2^2 + \tau||X||_*$
proximal mapping
$D_\tau(Y) = argmin_Z \frac{1}{2}||Y - Z||_F^2 + \tau||Y||_*$
If SVD is $Y = U\Sigma V^T$, where $\Sigma = diag[\sigma_1, \sigma_2, ..]$, then $D_\tau(Y) = U\Sigma^{'}V^T$, where $\Sigma^{'} = diag[\sigma_1^{'}, \sigma_2^{'}, ..]$

$$\sigma_k^{'} = \begin{cases} \sigma_k - \tau, & \text{if } \sigma_k \geq \tau \\ 0, & \text{if } \sigma_k < \tau \end{cases}$$

It is important to adjust hyperparameter, which is t_k. It corresponds to the step in (7), R_{k+1}. Original hyperparameter, that was recommended in paper, was $t_1 = 1$. However, by our implementation, the better hyperparameter was $t_1 = 0.5$. The quality in 1 metric increases by 20%. Big value of t_1 corresponds to bad behaviour of the algorithm and it loses quality quite fast.

Fista makes use of Nesterov's momentum acceleration to speed up the convergence. In each iteration, only a partial SVD is evaluated. Doing so for large-scale problems may still be too slow and require large memory. In this case, randomized techniques from numerical linear algebra can be computed to speed up the computation of SVD. But for our pictures, that we use, SVD works fast.

3 Convolutional autoencoders for matrix completion

In machine learning, autoencoder is an unsupervised learning algorithm with the input value as the same as the output value aiming to transform the input to output with least distortion[5]. Autoencoders have shown to be promising in many relevant image restoration tasks, such as image denoising and image inpainting, thus justifying their usage in matrix completion tasks.

In this context, we will define **encoder** as a mapping $f(x)$: $R^d \to R^m$ and **decoder** as a mapping $f(x)$: $R^m \to R^d$, where R^d denotes input space and R^d denotes latent space. The layers in both the encoder and decoder are fully connected:

$$l^{i+1} = (W^i l^i + b^i) \tag{8}$$

Here, l^i is the activation vector in the i-th layer, W^i and b^i are the trainable weights and is an element-wise nonlinear activation function.

The convolutional autoencoder extend the basic structure of the traditional autoencoder (i.e. feed-forward neural network) by changing the fully connected layers to convolution layers in the following manner:

$$L^{i+1} = (W^i * L^i + b^i) \tag{9}$$

where denotes the convolution operation and the bias b^i is broadcast to match the shape of L such that the j-th entry in b^i is added to the $j - th$ channel in L^i. Whereas before the hidden code was an m-dimensional vector, it is now a tensor with a rank equal to the rank of the input tensor. [7]

3.1 Training data and preprocessing

In order to explore the behavior of convolutional autoencoders, we have used three different datasets with different image resolution and number of channels. The datasets that we used are:
Dataset descriptions. In contrast to the traditional matrix completion methods that were designed in this paper to work only with grayscale images, deep learning methods handle working with 3-channel images well. Moreover, most of the training data that is publicly available contains only RGB images, making it difficult to approach single-channel data. We have decided to work with Landscape Grayscale dataset [2] that is commonly used for image colorization tasks [6], adjusting it to our needs by using only the grayscale

Dataset Name	Properties	Dataset size
CIFAR 10	$32x32$ color images	60000 (50000 training, 10000 test)
Linnaeus 5	$128x128$ color images	8000 (6000 training, 2000 test)
Grayscale landscapes	$128x128$ grayscale images	>7000 (4907 training, 2204 test)

Table 1: Overview on the datasets used for preforming matrix completion tasks.

images as our train and test data, performing image downsampling from 150 x 150 to 128 x 128 resolution, and performing a 80-20 split on the dataset.

The input data is loaded into Keras data generator. Since our code is multicore-friendly, we perform masking operation and rescaling before forming the batches without worrying that data generation becomes a bottleneck in the training process. Output of these procedure are the batches of masked images, and original images used for computing error from reconstructed image.

Justification of image sizes. Image restoration task would ideally complete large matrices and images of large resolution (eg.1024 x 1024) However, according to Yu et al.[15], to allow a pixel being influenced by the content 64 pixels away, it requires at least 6 layers of 3×3 convolutions with dilation factor 2. This justifies the overall better performance on the small CIFAR10 dataset (32 x 32), accompanying the fact that training data is magnitude larger. In addition to this, working with large resolution of images would require not only more complex architecture, but would also result in significantly larger number of features to train, leading to high training times. Considering hardware limitations, we are content with using toy datasets of image size of at most 128x128.

Data preparation. In our case as mentioned we need to add artificial deterioration to our images. This can be done using the standard image processing idea of masking an image. Since it is done in a self-supervised learning setting, we need X and y (same as X) pairs to train our model. Here X will be batches of masked images, while y will be original/ground truth image.

During data preparation, we faced couple of challenges on how to represent the imagery data in proper range. *plt.imshow* would often clip the existing photos into [0,1] range and thus many of the pixel values would become almost 0, thus resulting in completely black image. In order to prevent feeding the network with too many features close to zero that would result in misbehavior of the model (ex. oversaturated, high contrast photo in notebook trained Linneaus 5 dataset), we skip the pixel normalization, and do clipping at the very start of the project instead of the data generator.

There have been many attempts to tackle image restoration problem like image inpainting one. However, the difference is that most of the publications related to image inpainting only tackle regular small rectangle holes. In reality deterioration is highly unlikely a regular hole. Thus inspired by this paper we implemented irregular holes as mask of the image.

3.2 Convolutional autoencoder

Architecture. Architecture of this model was majorly inspired by the U-Net architecture. U-Net [11] is a convolutional neural network that was developed for biomedical image segmentation It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling.

At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU.

Input	*n*, 32,32,3	*n*, 128,128,(1/3)			
Layer	**Filters 32x32**	**Filters 128x128**	**Kernel size**	**Pool size**	**Activations**
ConvBlock	*n*, 32	*n*, 128	(3,3)	(2,2)	*ReLU*
ConvBlock	*n*, 64	*n*, 256	(3,3)	(2,2)	*ReLU*
ConvBlock	*n*, 128	*n*, 512	(3,3)	(2,2)	*ReLU*
ConvBlock	*n*, 256	*n*, 1024	(3,3)	(2,2)	*ReLU*
Layer	**(Up)filters 32x32**	**(Up)filters 128x128**	**Kernel size**	**(Up)kernel,stride**	**Activations**
UpConvBlock	*n*, 256,256	*n*, 1024,1024	(3,3)	(2,2)	*ReLU*
UpConvBlock	*n*, 128,128	*n*, 512,512	(3,3)	(2,2)	*ReLU*
UpConvBlock	*n*, 64,64	*n*, 256,256	(3,3)	(2,2)	*ReLU*
UpConvBlock	*n*, 32,32	*n*, 128,128	(3,3)	(2,2)	*ReLU*
Layer	**Filters 32x32**	**Filters 128x128**	**Kernel size**	**Pool size**	**Activations**
ConvBlock	*n*, 32	*n*, 128	(3,3)	(2,2)	*ReLU*
Conv2D	*n*, 3	*n*, (1/3)	(3,3)		*ReLU*

Table 2: Model architecture

ConvBlock architecture. We will use ConvBlock layers to extract features from the input images, and generate new features to feed into the next UpConvBlock module. ConvBlock consists of two convolutions, each followed by a rectified linear unit (ReLU). In case that the pooling is present, ConvBlock ends with MaxPool2D layer.

Layer	**Filters 32x32**	**Filters 128x128**	**Kernel size**	**Paddings**	**Activations**
Conv2D	*filters*	*filters*	(3,3)	*same*	*ReLU*
Conv2D	*filters*	*filters*	(3,3)	*same*	*ReLU*
MaxPooling2D (*pool size*)			(3,3)		

Table 3: Architecture of the ConvBlock layer

UpConvBlock architecture. UpConvBlock consists of two convolutional layers with ReLU as activation function, followrd by a Conv2DTranspose layer, which essentially doubles the spatial dimensions of it's output compared to the input. The overlapping values of the kernel multiplication results accross the image are then summed up, resulting in an output twice the lenght and twice the breadth of the input image.

Layer	**Filters 32x32**	**Filters 128x128**	**Kernel size**	**Pool size**	**Activations**
Conv2D	*filters*	*filters*	(3,3)	*same*	*ReLU*
Conv2D	*filters*	*filters*	(3,3)	*same*	*ReLU*
Conv2DTranspose	*filters*	*filters*	(3,3)	*same*	*ReLU*

Table 4: Architecture of the UpConvBlock layer

Training and evaluation. Beyond tracking only standard metrics on how well the model is fitting the training and validation data (losses), we go step further and compute Sørensen–Dice coefficient (also referred to as F1 score). The Sørensen–Dice coefficient index equals twice the number of elements common to both sets divided by the sum of the number of elements in each set. For training, we use batch size of 32 and Adam as an optimizer. Number of epochs has not been fixed, rather explored until the loss converges, but 8-10 epochs have shown to be more than enough for tasks like these.

Example of results: Grayscale landscapes During training, training loss converged to the value of 0.0266, validation loss converged to 0.0320, while F1 scores were 0.5936 and 0.5992 for train and validation set, respectively. In general, the training time has been determined by hardware specifications, but we managed to get the model to fit in quite small amount of time (approx. 95 minutes on Google Colab GPU).

4 Measuring performance of the traditional approach

In that section we will measure the performance of gradient methods in terms of mathematical norms. We worked on picture with 70% deleted pixels and then restored the image with gradient methods. It can be seen from figure 2 that from 4 iteration number of the algorithm all the gradient methods that were implemented, have better quality compared to baseline solution. FISTA algorithm has the lowest error after 100 iterations. The rate of the slope decreases faster than in DRS and Fast gradient method.After 300 iterations Fast gradient method has the same quality in Euclidean metric as FISTA. However, DRS has lower quality. The time which was spent on experiment with 300 iterations and 70% of deleted pixels equals to 20 seconds for FISTA algorithm, 19 seconds for DRS and 115 seconds for fast-gradient algorithm.The fast-gradient method works slower than others because of cumulative sum in the formula.

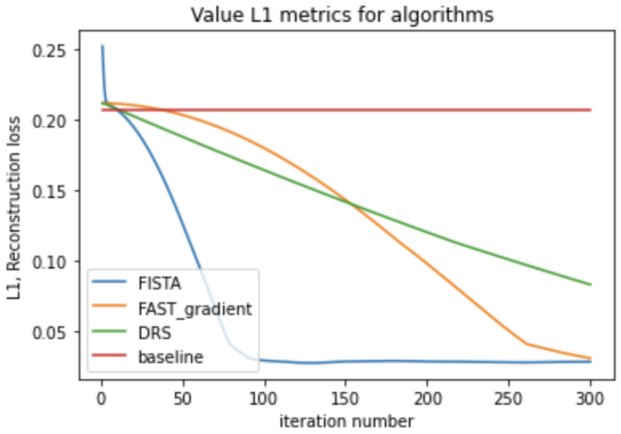

Figure 2: Quality on iteration number of the algorithms

5 Comparison of the models

In that section we will compare behaviour of the previous algorithms. We will compare results with baseline solution in 1 and 2 metrics. Baseline solution simply recovers the maximal value of pixel from the left and the right pixel.

Original picture

Picture with deleted pixels

Baseline solution

Fista algorithm

Table 5: Analyzed models

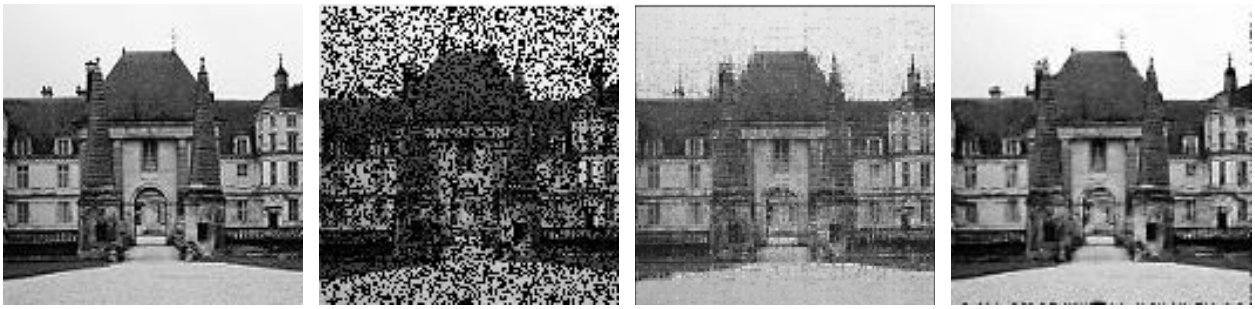

Figure 3: Original picture/Picture with deleted pixels/Fista algorithm/Conv Autoencoder

In table 5 70% of pixels were deleted in the picture. The deletion was made randomly. It can be seen that baseline solution has problems with corners of the image, whereas picture of fista's algorithm has no such problem, but is more blurred.However, the picture is restored well, with all reproduction of boundaries and thin lines. The restoration by fast gradient algorithm and DRS has the same result as Fista's algorithm, so we do not show it there.

The resolution of that picture is quite big, so it is not possible to use autoencoder algorithm on it. In order to compare the quality of both autoencoder and gradient algorithms, we will use picture with smaller resolution. In figure 2 is an example of image restoration by fista algorithm and convolution autoencoder at the same picture. It can be seen that both algoritms restore the image pretty well. In comparison with Fista, picture by autoencoder's algorithm is more bright picture and less blurred. We calculated Euclidean norm and 2-norm on both of pictures. The results can be seen in table 6. Both algorithms has low value of the norm, however convolutional autoencoder has norm value 10 times lower than fista's algorithm in both metrics.

Algorithm, norm	**Value of norm**
Conv autoencoder, 1	0.0011
Conv autoencoder, 2	0.00023
Fista, 1	0.01419
Fista, 2	0.00418

Table 6: Value in norms for algorithms

6 Conclusion

During the course of the project, we implemented different methods for restoring the pixel values for images with deleted pixels. We tried several methods: convolutional autoencoders, gradient methods. The algorithms has been tested on three different datasets(CIFAR10,Linneaus, Landscape grayscale). The fixed amount of pixels was deleted and then original picture was recovered. The final picture in gradient algorithms has the same contour structure, but is a little bit blurry. However, the final picture by autoencoders is very similar to the original one and is less blurred.

It's important to notice that many image processing tasks are similiar to the image restoration, more precisely inpainting, denoising and deblurring, to name a few. Methods for solving those problems usually rely on an Autoencoder – a neural network that is trained to copy it's input to it's output.

For further research we suggest using different proximal smooth operators in gradient methods, or using partial convolutions or replacing CNN architecture by say ResNet when speaking about deep learning approach. For large-scale matrices modern techniques of linear algebra can be used. In addition, hyperparameter tuning for the FISTA model needs to be done very carefully as the model can have different results.

References

[1] *https://math.stackexchange.com/questions/1158798/show-that-the-dual-norm-of-the-spectral-norm-is-the-nuclear-norm.*

[2] *https://www.kaggle.com/theblackmamba31/landscape-image-colorization.*

[3] M. Fazel B. Recht and P. A. Parrilo. . *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization.* 2007.

[4] Yudong Chen and Yuejie Chi. *Harnessing Structures in Big Data via Guaranteed Low-Rank Matrix Estimation.* 2018.

[5] Tom Gedeon and D. Harris. Progressive image compression. pages 403 – 407 vol.4, 07 1992.

[6] Manoj Kumar, Dirk Weissenborn, and Nal Kalchbrenner. Colorization transformer. *CoRR*, abs/2102.04432, 2021.

[7] Ilja Manakov, Markus Rohm, and Volker Tresp. Walking the tightrope: An investigation of the convolutional autoencoder bottleneck. *CoRR*, abs/1911.07460, 2019.

[8] Y. Nesterov. *Introductory Lectures on Convex Optimization.* Kluwer Academic Pub- lishers, 2004.

[9] Y. Nesterov. *Smooth minimization of non-smooth functions.* Springer-Verlag, 2005.

[10] Stanley J. Reeves. Chapter 6 - image restoration: Fundamentals of image restoration. In Joel Trussell, Anuj Srivastava, Amit K. Roy-Chowdhury, Ankur Srivastava, Patrick A. Naylor, Rama Chellappa, and Sergios Theodoridis, editors, *Academic Press Library in Signal Processing: Volume 4*, volume 4 of *Academic Press Library in Signal Processing*, pages 165–192. Elsevier, 2014.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[12] N. Z. Shor. *N. Z. Shor. Minimization Methods for Non-differentiable Functions. Springer.* 1985.

[13] Seok-Jeong Song, Young In Kim, Jina Bae, and Hyoungsik Nam. Deep-learning-based pixel compensation algorithm for local dimming liquid crystal displays of quantum-dot backlights. *Opt. Express*, 27(11):15907–15917, May 2019.

[14] Senior Member IEEE Hing Cheung So Fellow IEEE Xiao Peng Li, Lei Huang and IEEE Bo Zhao, Member. *A Survey on Matrix Completion: Perspective of Signal Processing.* 2019.

[15] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention, 2018.