Unrolled Optimization & Matrix Completion

Advisor: Dr.Muhammad Tahir

Presented by:
Talha Ahmed (24100033)
Nehal Ahmed Shaikh (24020001)

Background

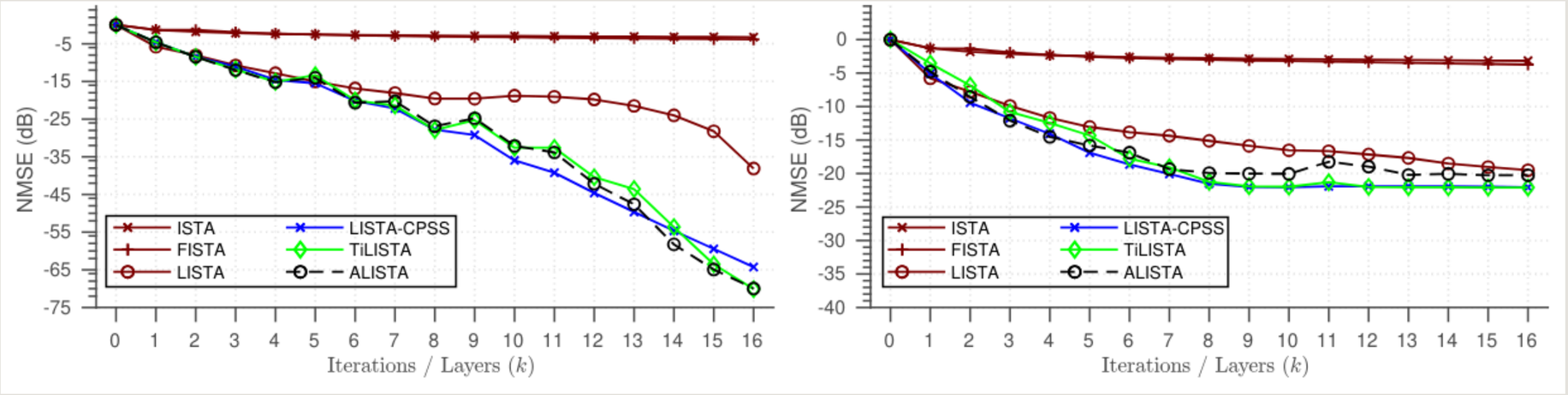
- **Interpretability of AI:**
  - The ability to understand and explain how machine learning models arrive at their conclusions.
  - As machine learning systems become more complex, understanding their decision-making processes becomes increasingly challenging.

- **Deep Unfolding:**
  - Recent approach that economizes on data and computational requirements.
  - It captures the systematic connections present in the already fine-tuned algorithms and unprecedented performance gains from DNN to give rise to interpretable, efficient models in various applied AI domains.

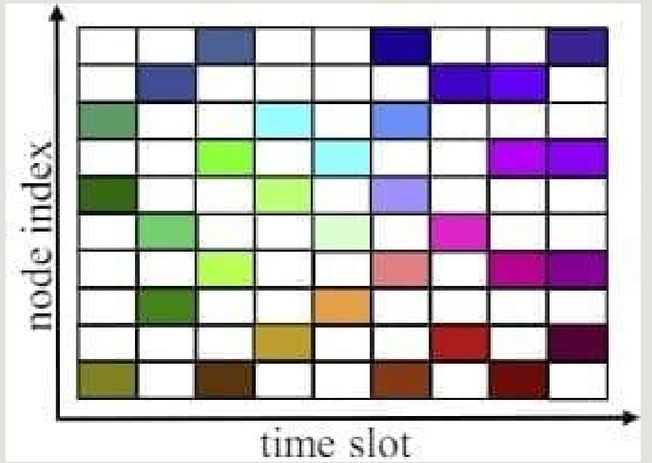
- **Matrix Completion (MC):**
  - A problem where a matrix (assumed to be low-rank) is missing some entries while the non-missing entries are possibly affected by noise.

Examples

1. LISTA and its variants as a unfolding of ISTA - predominantly used in sparse signal recovery [1].
2. Spatio-temporal (ST) data (from WSN's) recovery comes under MC [2].
3. Netflix Prize problem expressed as MC problem [3].



LISTA + variants Vs ISTA



Correlated - ST Data Recovery

Problem Statement

Our objective is to reap the benefits of unfolded optimization algorithms, namely, faster inferences and higher accuracy scores, on data that has been corrupted by white noise and impulsive GMM noise.

Presentation Timeline

01. **METHODOLOGY OVERVIEW**
Brief introduction to various MC algorithms and optimization methods

02. **DATA**
Data collection, transformation, augmentation, generation, training process

03. **PIPELINE 1**
Introduction to ConvMC-Net, formulation, performance, limitations

04. **PIPELINE 2**
Introduction to ConvHuberMC-Net, formulation, performance, limitations

05. **CONCLUSION**
Project timelines, distribution of work, and future recommendations.

Methodology Overview

Matrix Completion

- Original problem:

$$X_\Omega = H_\Omega \odot X + N,\ H(i,j) = \begin{cases} 1 & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

- Rank-minimization formulation:

$$\min_M \operatorname{rank}(M), \quad \text{s.t.} \quad \|M_\Omega - X_\Omega\|_F \leq \delta$$

- Unfortunately, the rank-minimization problem is NP-hard, as all exactly solving algorithms are doubly exponential in the maximum dimension of the matrices. Therefore, all state-of-the-art algorithms solve the approximate problem, i.e., nuclear norm minimization (NNM).

Nuclear Norm Minimization

- The nuclear norm is the convex envelope of rank and this convex relaxation enables us to efficiently solve the rank-minimization problem [4].

- The singular value thresholding (SVT) approach uses a proximal objective of NNM, which is often minimized by setting up the Lagrangian:

$$L(M, Y) = \tau \|M\|_* + \frac{1}{2} \|M\|_F^2 + \langle Y, M_\Omega - X_\Omega \rangle, \ \tau \geq 0$$

- Here, the parameter tau balances between the non-smooth nuclear norm and the smooth Frobenius norm, while Y is the Lagrange multiplier. The Lagrangian is, then, solved by introducing a soft thresholding operator.

Singular Value Thresholding (SVT)

- SVT is defined as follows:

$$\Psi_\alpha(X) = \mathrm{U}\mathrm{diag}(\mathrm{ReLU}(\sigma_i - \alpha))$$

Competing Algorithms

- We compare our models against the tabulated algorithms, all of which use matrix factorization (MF).
- The basic idea behind MF is to utilize two low-rank matrices to represent the objective matrix with an assumption that the rank of original matrix is known.

Algorithms	(Hyper)Parameters
L0-BCD [5]	epsilon = 1e-20
M-estimation [6]	c = 1.345
Lp-reg (p = 1) [7]	maxiter = 1000, num_trials = 1
Lp-ADMM (p = 1) [7]	maxiter = 500, num_trials = 1
ORMC	maxiter = 500, num_trials = 1

Matrix Factorization (MF)

- MF is superior to NNM in that it circumvents the issues of low computational efficiency and hence limited scalability, as it does not involve singular value decomposition (SVD).

- The new optimization problem is

$$\min_{U,V,Z} \|UV^T - Z\|_F^2, \text{ s.t. } Z_\Omega = X_\Omega, U \in \mathbf{R}^{m \times r}, V^T \in \mathbf{R}^{r \times n}$$

- Here, r is the predicted rank of the objective matrix.

- MF is also more flexible when dealing with various noise distributions due to its smooth nature, unlike NNM.

Optimization Overview

- Though there are multiple optimization algorithms, as shown in the table, we are only concerned with those that are relatively popular and not gradient-based.
- This is primarily because most state-of-the-art algorithms for robust matrix completion are based on these algorithms; it is also worth noting that BCD is essentially unconstrained ADMM.

Gradient	Non-gradient
Gradient descent	Block coordinate descent
Accelerated proximal gradient	Alternating direction method of multipliers
Bregman Iteration	

Alternating Direction Method of Multipliers

- To tackle constrained large-scale optimization problems, Gabay and Mercier [8] introduced ADMM. According to the principle of ADMM, the constrained problem to be optimized can be expressed as

$$\min_{X,Z} F(X) + G(Z) \quad \text{s.t.} \quad AX + BZ = C,$$

$$X \in \mathbb{R}^{m \times r},\ Z \in \mathbb{R}^{n \times r},\ A \in \mathbb{R}^{p \times m},\ B \in \mathbb{R}^{p \times n}$$

- Here, F(X) and G(Z) are convex but the former is non-smooth while the latter is smooth and r is the rank of the objective matrix.

ADMM (Continued)

- The optimization problem is initially converted into the augmented Lagrangian:

$$L_\delta(X, Y, Z) = F(X) + G(Z) + Y^T(AX + BZ - C) + \frac{\delta}{2}\|AX + BZ - C\|_F^2,\ \delta > 0$$

- Here, the last term is employed for regularization.
- Finally, the BCD approach separately optimizes each of the three variables.

ADMM Algorithm

Algorithm 1 ADMM

Input: Maximum iteration N, X_0, Z_0 and δ
for $k = 0, 1, ..., N$ **do**
    $X_{k+1} = \arg\min_{X_k} L_\delta(X_k, Y_k, Z_k)$
    $Z_{k+1} = \arg\min_{Z_k} L_\delta(X_{k+1}, Y_k, Z_k)$
    $Y_{k+1} = Y_k + \delta(AX_{k+1} + BZ_{k+1} - C)$
end for
Output: X_{k+1}, Z_{k+1}

Data

Data sets for MC

- Intel Berkeley [9]
  - 468 ground truth and low-rank matrices, each of shape (49x60).
  - 400 of these were used to train ConvMC-Net and 68 to test it.

- CSDS 300 [10]
  - 300 RGB images, each resized either to (150x300) or (400x500).
  - 200 of these were used to train the models.
  - 100 to test them for image inpainting.
    - Eight of those 100 images were used to compare performances of all algorithms.

- Synthetic: matrices of either shape (150x300) or (400x500)
  - 20 were used to train the models.
  - 10 were used to test the models.

- Note: two shapes were used for the last two data sets because we discovered that the shape significantly affects the performance of some algorithms.

Data set Specifics

- Intel Berkeley: each matrix was corrupted by zero-mean Gaussian noise.

- For CSDS 300, each image was:
  - grayed
  - sampled (20-80%)
  - corrupted with GMM noise (3, 5, 6, or 9 SNR).

- Synthetic: each matrix was generated by multiplying two matrices of shapes (mx10) and (10xn), where 10 is the rank and (mxn) is either (150x300) or (400x500).
  - Sampling and corruption were the same as they were for CSDS 300.

CSDS 300 (Inference)








Hyperparameters

- Shared between ConvMC-Net and ConvHuberMC-Net:
  - TrainInstances = 20
  - ValInstances = 10
  - BatchSize = 20
  - ValBatchSize = 10
  - num_epochs = 20

- Different between ConvMC-Net and ConvHuberMC-Net:
  - learning_rate (ConvMC-Net) = 0.12
  - learning_rate (ConvHuberMC-Net) = 0.001

Pipeline 1 - ConvMC-Net

Proposed Method

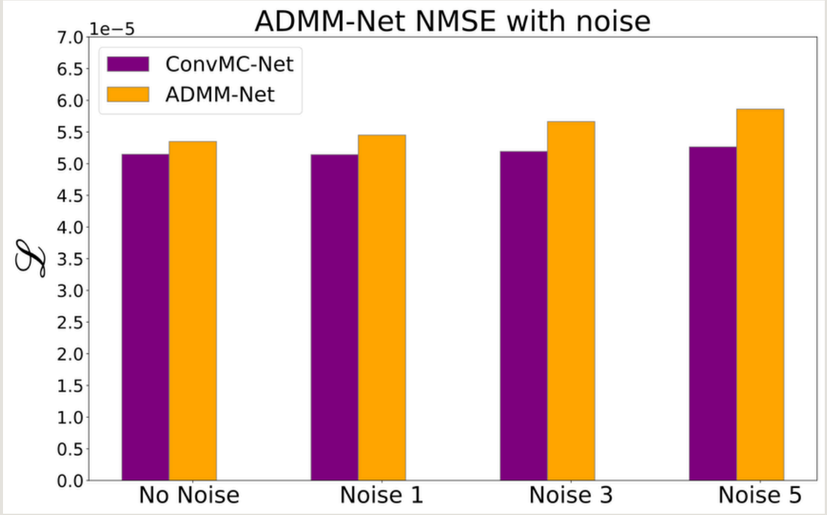
- **Augmented Lagrangian:** $\mathcal{L}(L, Y) = \|L\|_* + \mathrm{Tr}(Y^T \cdot (D_\Omega - L_\Omega)) + \frac{\mu}{2}\|D_\Omega - L_\Omega\|_F^2$
- **Updates:**
  - $L^{k+1} = \min_L \frac{1}{\mu}\|L\|_* + \frac{1}{2}\|D_\Omega - L_\Omega + \frac{Y}{\mu}\|_F^2$
  - $Y^{k+1} = Y^k + \mu \nabla_Y L(L^{k+1}, Y^k)$

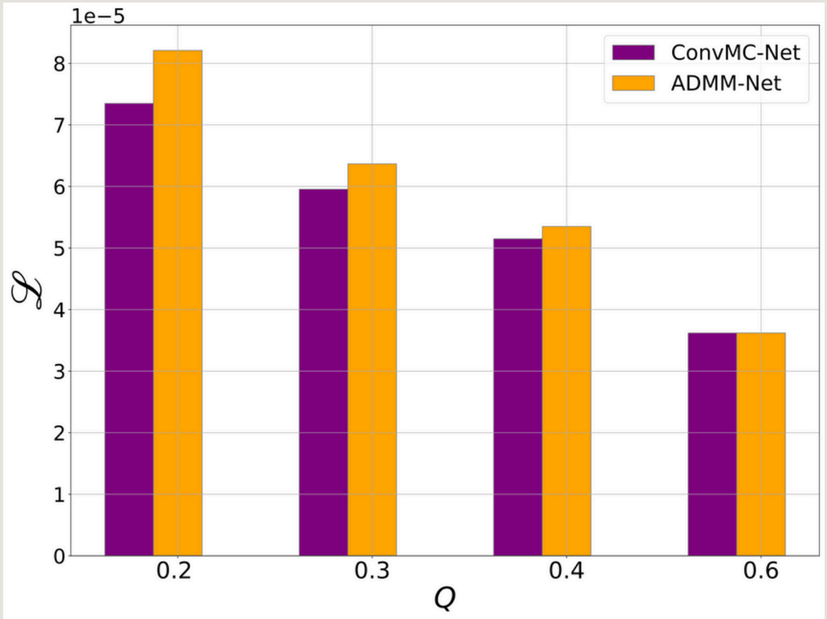
- **Proximal Step:** $L^{k+1} = \Psi_{\mu^{-1}}\left\{L_{\Omega^C} + D_\Omega + \mu^{-1} Y_\Omega\right\}$
- **Measurement Matrix Replacement [11]:** $\min_L \frac{1}{\mu}\|L\|_* + \frac{1}{2}\|D_\Omega - (HL)_\Omega + \frac{Y_\Omega}{\mu}\|_F^2$
- **We further replace *HL* with GL**
- **Simplification:** $L^{k+1} = \Psi_{\frac{1}{\mu^k}}\{L^k + G^T D_\Omega - G^T G L_\Omega^k + \frac{G^T}{\mu^k} Y_\Omega^k\}$
- **Introduce convolutional kernels and transformation:** $G^T Y_\Omega = W \circ Y_\Omega + B$
- **Finalized Update:** $L^{k+1} = \Psi_{\frac{1}{\mu^k}}\{L^k + (C_1^k * D) + (C_2^k * L_\Omega^k) + (W_k \circ Y_\Omega^k + B^k)\}$
- **Optimizer: Adam ; Epochs: 40, LR: 0.12, Stride/Padding: (1, 1), Kernel Size: (3,3,3)**

Performance on Sensing Data

- Fixing white noise and sampling rate
- Comparing NMSE - ConvMC performs better
- Comparing Inference time - ConvMC is faster

Model	Training Time	Testing Time
ConvMC-Net	0.02283	0.01425
ADMM-Net	0.4245	0.2629

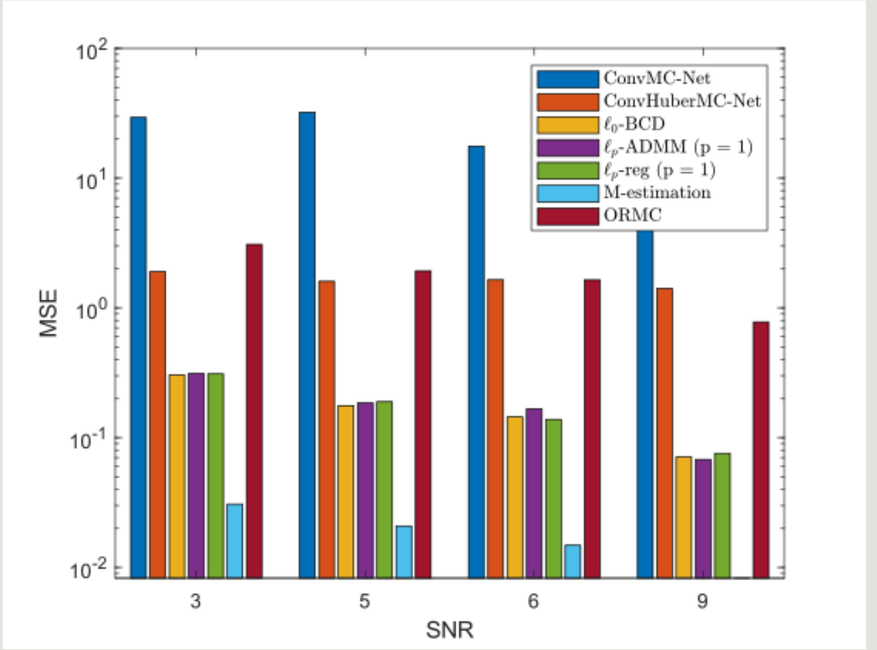


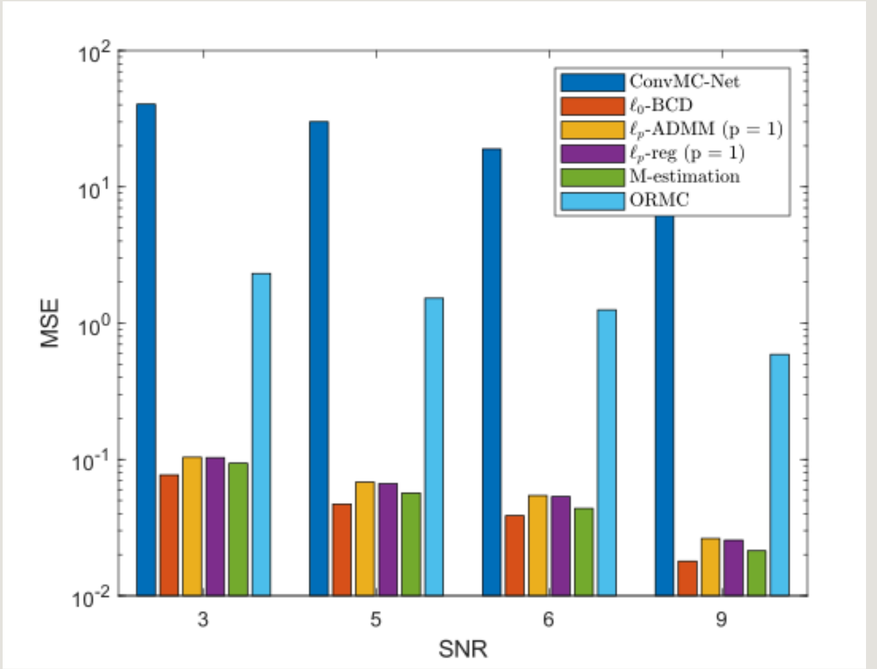


Performance on Synthetic Dataset

- **Sampling rate fixed at 40% and varying SNR**
- **Two versions: (150 x 300) & (400 x 500)**
- **Inference time: ConvMC-Net still above the rest**
- **NMSE: ConvMC-Net performs poorly.**

Algorithms	Inference/Testing Time
L0-BCD	0.12
M-estimation	2.01
Lp-reg (p = 1)	25.99
Lp-ADMM (p = 1)	9.33
ORMC	3.03
ConvMC-Net	0.06





Performance on Image Inpainting Dataset

- **Sampling rate fixed at 50% and SNR fixed at 50%**
- **Two versions: (150 x 300) & (400 x 500)**
- **Inference time: ConvMC-Net still above the rest**
- **PSNR/SSIM: ConvMC-Net performs poorly.**
- **Note however, only on this combination does this sort of performance is observable**

Image	PSNR	SSIM
1	32.98	0.57
2	33.41	0.61
3	32.83	0.52
4	33.40	0.58
5	33.50	0.60
6	33.96	0.63
7	33.53	0.59
8	34.75	0.63

ConvMC-Net with 20% sampling rate and DB 5.0

Image	Mask	Metric	ℓ_0-BCD	ℓ_p-reg	ℓ_p-ADMM	ORMC	M-Estimation	ConvMC-Net
Image-1	random	PSNR	**23.3228**	22.9612	20.9205	15.3217	21.1176	12.7777
		SSIM	**0.4845**	0.4427	0.2764	0.1409	0.3402	0.0438
Image-2	random	PSNR	**19.2602**	19.1894	18.2118	15.7579	18.1707	12.1982
		SSIM	**0.3309**	0.3240	0.2423	0.2547	0.2724	0.0412
Image-3	random	PSNR	**22.5507**	22.0577	20.0277	12.9958	20.3123	13.4528
		SSIM	**0.4012**	0.3602	0.2311	0.1054	0.2799	0.0446
Image-4	random	PSNR	**19.4893**	19.3902	18.5176	12.8923	18.8606	10.9455
		SSIM	**0.3372**	0.3226	0.2353	0.1379	0.2794	0.0355
Image-5	random	PSNR	**23.4655**	23.3291	20.9843	15.9237	21.7869	12.5822
		SSIM	**0.5054**	0.4825	0.2757	0.1643	0.3756	0.0407
Image-6	random	PSNR	**23.1620**	22.8400	20.4106	15.5223	21.5492	13.4670
		SSIM	**0.4573**	0.4256	0.2241	0.1565	0.3604	0.0535
Image-7	random	PSNR	**24.5711**	23.9382	21.4715	14.7422	23.3300	14.2580
		SSIM	**0.4820**	0.4298	0.2465	0.1195	0.3975	0.0614
Image-8	random	PSNR	**18.3704**	18.0212	17.1763	10.8709	16.9391	12.7077
		SSIM	**0.2809**	0.2534	0.1930	0.1342	0.1941	0.0730

PSNR and SSIM of different algorithms on eight 150 x 300 images with 5dB salt-and-pepper noise

Image	Mask	Metric	ℓ_0)-BCD	ℓ_p)-reg	ℓ_p)-ADMM	ORMC	M-Estimation	ConvMC-Net
Image-1	random	PSNR	18.0168	**21.2846**	17.5846	14.2193	20.1214	14.6764
		SSIM	0.3152	**0.3511**	0.1869	0.1498	0.2828	0.0197
Image-2	random	PSNR	16.9121	**18.7851**	16.2755	14.3490	17.9068	28.4482
		SSIM	0.2681	**0.2986**	0.2004	0.2139	0.2401	0.4547
Image-3	random	PSNR	15.6531	**19.7035**	15.3484	11.3027	18.7403	13.8781
		SSIM	0.2300	**0.2769**	0.1741	0.0977	0.2213	0.0524
Image-4	random	PSNR	16.5880	**18.2356**	14.4972	11.8165	17.2024	12.2581
		SSIM	0.2358	**0.2778**	0.1973	0.1385	0.2355	0.0258
Image-5	random	PSNR	20.0771	**20.9231**	17.4914	14.7859	20.8793	13.4158
		SSIM	0.3506	**0.3734**	0.2045	0.1584	0.3045	0.0587
Image-6	random	PSNR	17.8750	21.0284	15.3678	13.9751	**21.0675**	12.8591
		SSIM	0.2762	**0.3100**	0.1493	0.1243	0.2923	0.0549
Image-7	random	PSNR	18.5359	**21.4101**	16.8222	13.4270	20.9942	14.4267
		SSIM	0.2887	**0.3233**	0.1772	0.1098	0.2893	0.0453
Image-8	random	PSNR	7.6655	16.5794	10.8466	9.3512	16.0630	**37.3651**
		SSIM	0.1510	0.2106	0.1441	0.1110	0.1816	**0.8352**

PSNR and SSIM of different algorithms on eight 400 x 500 images with 5dB salt-and-pepper noise

Pipeline 2 - ConvHuberMC-Net

Proposed Method

- **Motivation: The traditional Least Squares method for solving below is sensitive to outliers and inefficient if error-terms are non-gaussian.** $y_i = \beta_0 + x_{[i]}^T \beta + v_i, \quad i = 1, \ldots, N,$
- **We use robust Huber's Loss** $\rho_c(x)$ **which acts like the L2 loss for relatively small errors and otherwise L1 loss.** $\rho_c(x) = \begin{cases} \frac{1}{2}x^2, & \text{for } |x| \le c \\ 2c|x| - c^2, & \text{for } |x| > c, \end{cases}$
- **ML approach:** $L_{\mathrm{ML}}(\beta, \sigma) = N \ln \sigma + \sum_{i=1}^{N} \rho_{\mathrm{c}} \left(\frac{y_i - x_i^T \beta}{\sigma} \right),$
- **Above approach is non-convex w.r.t when it comes to minimizing the negative log-likelihood.**
- **Alternative - Huber's Criterion** $\min_{\beta \in \mathbb{R}^{p+1}, \sigma > 0} L_{\mathrm{HUB}}(\beta, \sigma) = N(\alpha\sigma) + \sum_{i=1}^{N} \rho \left(\frac{y_i - x_{[i]}\beta}{\sigma} \right) \sigma,$ **[12]**
- **We then use the matrix factorization approach i.e. each row of U and and each column of V is sequentially updated using the Huber's Criterion (denoted as *HuberCell* in next slide)**
- **This approach is parallezible and does not require any computationally expensive operations.**

HuberCell

- **Solves Huber's criterion using MM algorithm**
- **Introducing convolutional operation for the calculation of the psuedo-inverse initialized to identity maps for the unfolding process.**
- **The iterative version uses around 500 iterations for convergence.**
- **We hope by introducing learnable kernels, the number of iterations are drastically reduced.**

Algorithm 3 The `hubreg` algorithm for solving $L_{\mathrm{HUB}}(\beta, \sigma)$ when $\rho = \rho_{H,c}$ using the MM algorithm.

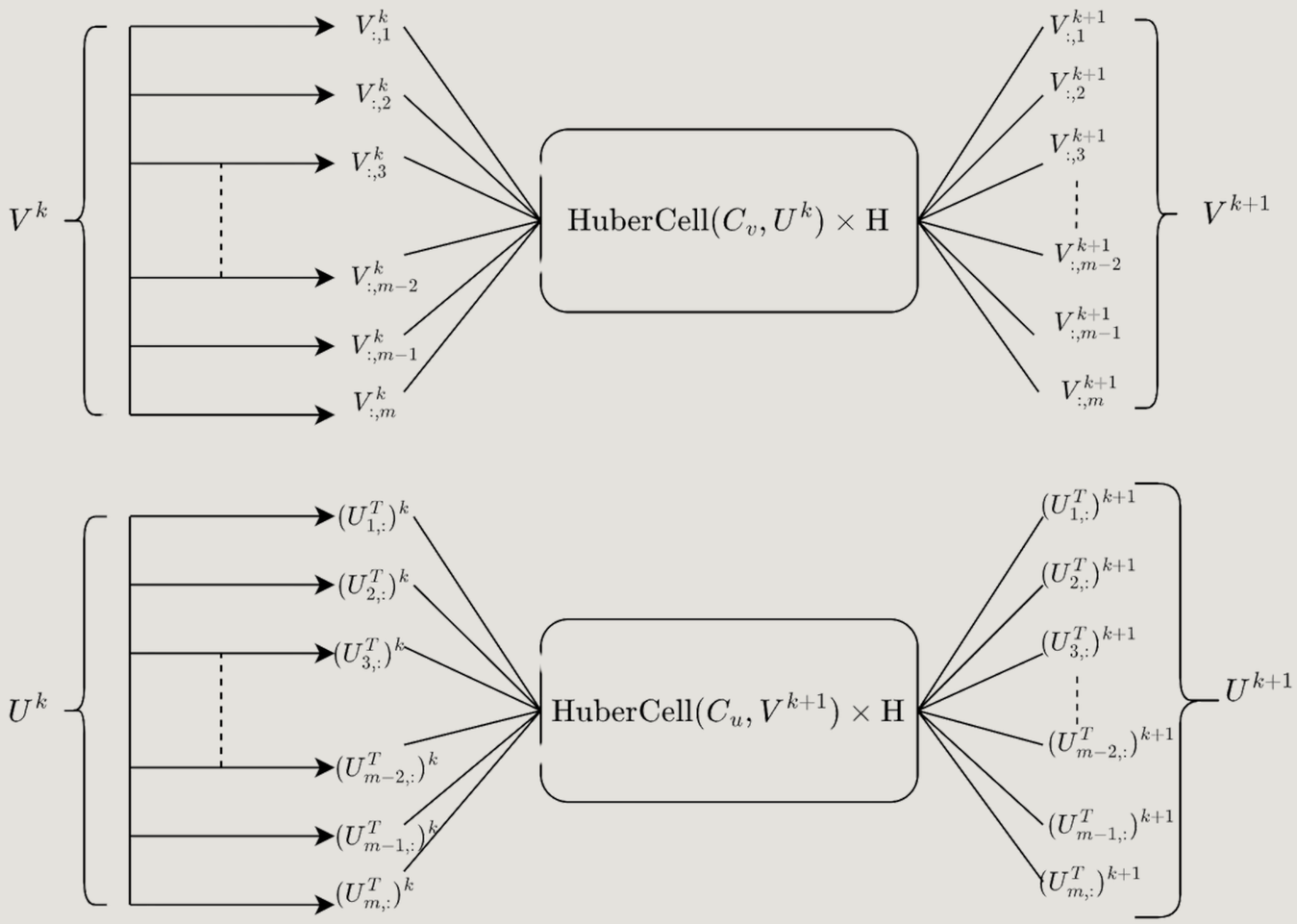
Require: $y \in \mathbb{F}^N$, $X \in \mathbb{F}^{N \times (p+1)}$, c, $\beta^{(0)}$, $\sigma^{(0)}$

Ensure: $(\hat{\beta}, \hat{\sigma})$, the minimizer of Huber's criterion $L_{\mathrm{HUB}}(\beta, \sigma)$ when $\rho = \rho_{H,c}$.

Initialize: $N_{\mathrm{iter}} \in \mathbb{N}$, $\delta > 0$, $\alpha = \alpha(c)$, $X^+ = (X^H X)^{-1} X^H$.
for $n = 0, 1, \ldots, N_{\mathrm{iter}}$ **do**
    Update residual: $r^{(n)} = y - X\beta^{(n)}$
    Update pseudo-residual: $r_\psi^{(n)} = \psi_{H,c}\left(\frac{r^{(n)}}{\sigma^{(n)}}\right) \cdot \sigma^{(n)}$
    Update scale: $\sigma^{(n+1)} = \frac{1}{\sqrt{N(2\alpha)}} \|r_\psi^{(n)}\|_2$
    (Re)update the pseudo-residual: $r_\psi^{(n+1)} = \psi_{H,c}\left(\frac{r^{(n)}}{\sigma^{(n+1)}}\right) \cdot \sigma^{(n+1)}$
    Regress X on $r_\psi^{(n+1)}$: $\delta^{(n)} = X + r_\psi^{(n+1)}$
    Update regression vector: $\beta^{(n+1)} = \beta^{(n)} + \delta^{(n)}$
    if $\frac{\delta^{(n)}}{\beta^{(n)}} < \delta$ **then**
        return $(\hat{\beta}, \hat{\sigma}) \leftarrow (\beta^{(n+1)}, \sigma^{(n+1)})$
    end if
end for

Architecture

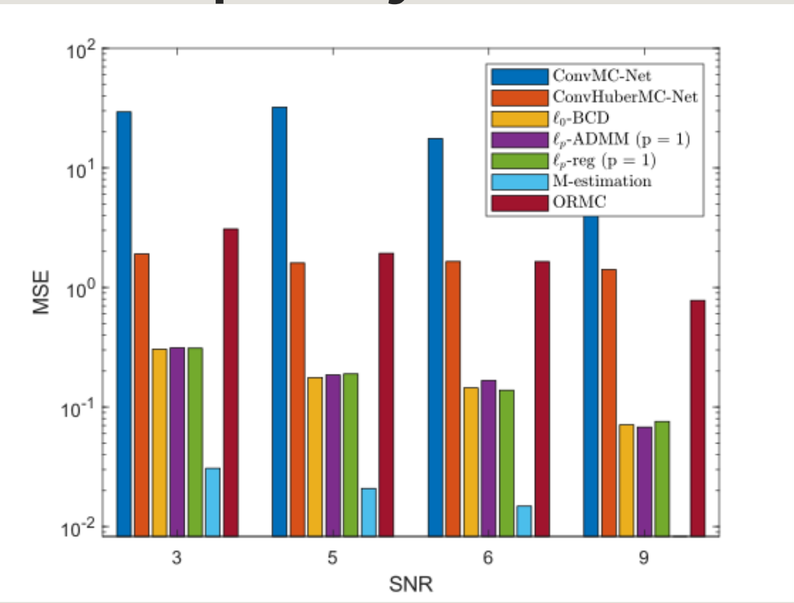
- **HuberCell updates sequentially each column/row of V/U sequentially by taking the number of iterations for each *hubreg* (denoted as *H*) and its specific convolution map.**



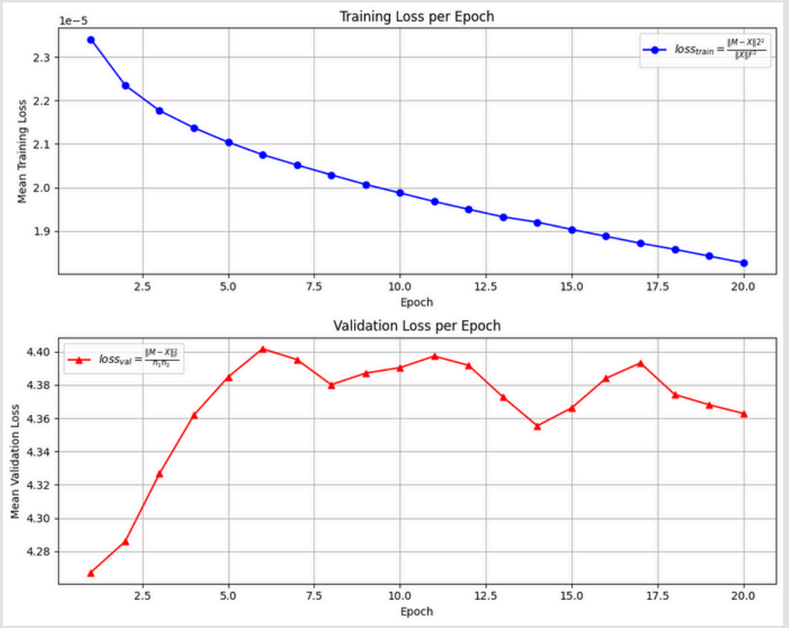
Performance on Synthetic Dataset

- Fixing H = 2 and number of layers = 3
- We test on (150 x 300)
- MSE: Pipeline 2 is better than Pipeline 1 however still not better than the other algorithms.
- Inference Time: 5 - 7 seconds on average. Comes out to be more than Pipeline 1.
- More importantly, its much unpredictable in terms of its trajectories of training and validation loss.
- For combinations of around 30% to 60% sampling rate, the trajectories start of with an increase in the validation loss then a decrease but still ends up at a loss higher than it started with.
- For combinations of 20% and 80% i.e. the extremes for our sampling rate settings the losses behave quite nicely.
- We also tried as a result learning the psuedo-inverse however, that resulted in all combinations having a validation loss in the range (7 - 8)
- Next slide shows the synthetic data performance + a case of its unpredictability + a case where it shows a sound result.
- Due to this unpredictability, we did not go ahead with further experiments like we did for ConvMC-Net

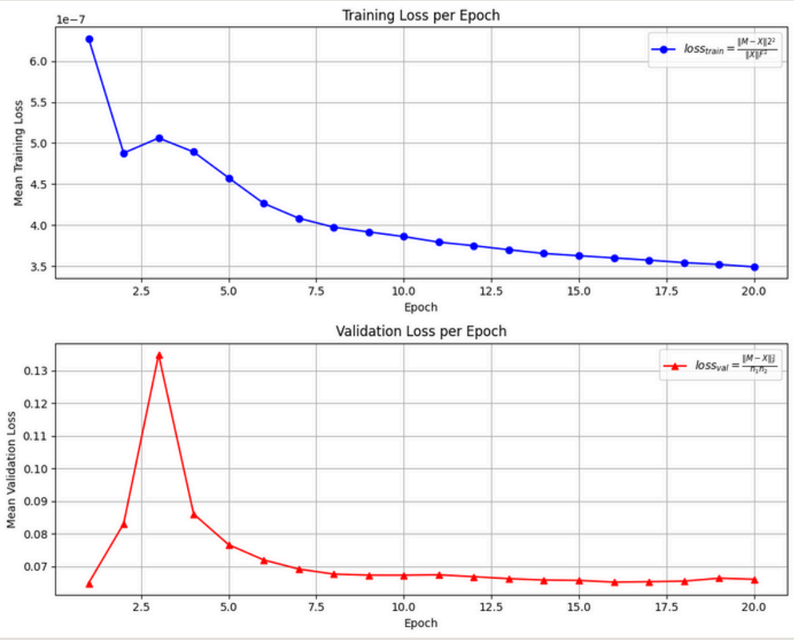
Close Up Analysis



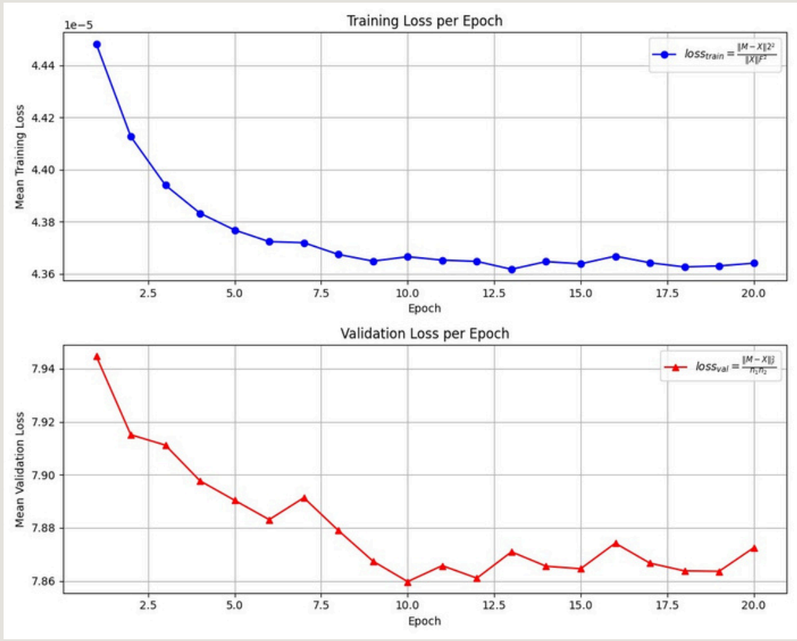
MSE vs SNR: sampling rate = 0.4, shape = (150 × 300)



Training and validation loss for 30% sampling rate and 6 SNR



Training and validation loss for 80% sampling rate and 5 SNR



Training and validation loss for 20% sampling rate and 3 SNR

Conclusion

Conclusion and Future Improvements

- We came up with ConvMC-Net in our pipeline 1, an algorithm with great inference but poor performance for impulsive GMM noise.
- ConvHuberMC-Net unpredictable performance for middle sampling rates. While its better comparable with the top state of the art methods for extreme combinations.
- For ConvMC-Net, we can consider including noise in its formulation (such as APG) but such PCP methods are not built for GMM type of noise.
- For ConvHuberMC, we might need to consider more sophisticated unfolded operations besides simple convolutional operation. However, this can get tricky with little to no past material on unfolded approaches for matrix factorization based algorithms.
- Consider taking inspiration from [13], [14] to expand on both pipelines.

References

[1] K. Gregor and Y. Lecun, “Learning fast approximations of sparse coding,” 08 2010.

[2] L. Yang, Y. C. Eldar, H. Wang, K. Kang, and H. Qian, “An ADMM-Net for data recovery in wireless sensor networks,” in 2020 28th European Signal Processing Conference (EUSIPCO), 2021, pp. 1712–1716.

[3] https://www.thrillist.com/entertainment/nation/the-netflix-prize.

[4] M. Fazel, “Matrix rank minimization with applications,” Ph.D. dissertation, Dept. Elect. Eng, Stanford Univ, California, USA, 2002.

[5] X. P. Li, Z.-L. Shi, Q. Liu, and H. C. So, “Fast robust matrix completion via entry-wise L0-norm minimization,” IEEE Transactions on Cybernetics, vol. 53, no. 11, pp. 7199–7212, 2023.

References (Continued)

[6] M. Muma, W.-J. Zeng, and A. M. Zoubir, “Robust m-estimation based matrix completion,” in ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 5476–5480.

[7] W. J. Zeng and H. So, “Outlier-robust matrix completion via Lp-minimization,” IEEE Transactions on Signal Processing, vol. 66, no. 5, pp. 1125–1140, Mar. 2018.

[8] D. Gabay and B. Mercier, “A dual algorithm for the solution of nonlinear variational problems via finite element approximation,” Computers & Mathematics with Applications, vol. 2, pp. 17–40, 1976.

[9] “Intel berkeley research lab,” http://db.lcs.mit.edu/labdata/labdata.html, 2004.

References (Continued)

[10] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in Proc. 8th Int’l Conf. Computer Vision, vol. 2, July 2001, pp. 416–423.

[11] O. Solomon, R. Cohen, Y. Zhang, Y. Yang, Q. He, J. Luo, R. J. G. van Sloun, and Y. C. Eldar, “Deep unfolded robust PCA with application to clutter suppression in ultrasound,” IEEE Transactions on Medical Imaging, vol. 39, no. 4, pp. 1051–1063, 2020

[12] A. M. Zoubir, V. Koivunen, E. Ollila, and M. Muma, Robust Statistics for Signal Processing. Cambridge, UK: Cambridge University Press, 2018.

References (Continued)

[13]https://stwisdom.github.io/assets/ICASSP2017_SparseCodingUnfoldsToRNN_final.pdf
[14] https://arxiv.org/pdf/2108.09138

Thank You!