Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation

Markus Götz, **Hartwig Anzt**
ScalA'18: 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, WS at SC18

Steinbuch Centre for Computing (SCC)

Motivation: Block Jacobi Preconditioning

- Jacobi method based on diagonal scaling: $P = diag(A)$

- Can be used as
  - Iterative solver

$$x^{(k+1)} = x^{(k)} + P^{-1}b - P^{-1}Ax^{(k)}$$

  - Preconditioner

$$\tilde{A} = P^{-1}A, \qquad \tilde{b} = P^{-1}b$$

$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$

Motivation: Block Jacobi Preconditioning

- Jacobi method based on diagonal scaling: $P = diag(A)$

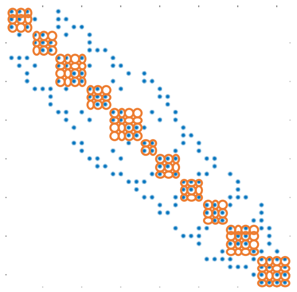
- Can be used as
  - Iterative solver

$$x^{(k+1)} = x^{(k)} + P^{-1}b - P^{-1}Ax^{(k)}$$

  - Preconditioner

$$\tilde{A} = P^{-1}A, \qquad \tilde{b} = P^{-1}b$$

$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$



- Extension: block-Jacobi: $P = diag_B(A)$
  - Set of diagonal blocks
  - Treat each block as linear system
  - Larger blocks
    - better convergence,
    - more expensive to compute

Motivation: Block Jacobi Preconditioning

- Jacobi method based on diagonal scaling: $P = diag(A)$
- Can be used as
  - Iterative solver

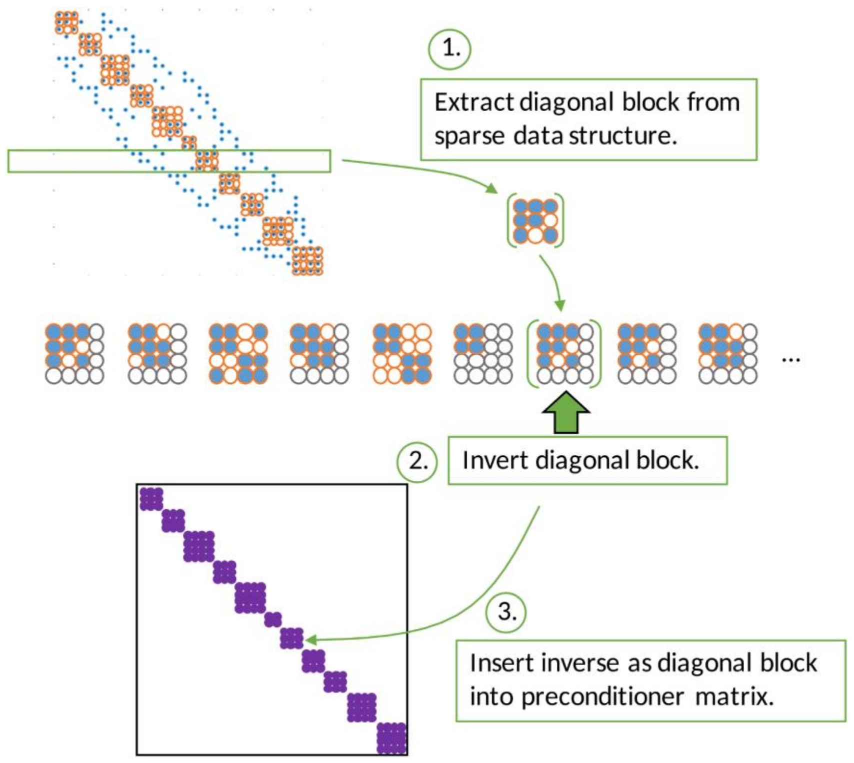
$$x^{(k+1)} = x^{(k)} + P^{-1}b - P^{-1}Ax^{(k)}$$

  - Preconditioner

$$\tilde{A} = P^{-1}A, \qquad \tilde{b} = P^{-1}b$$

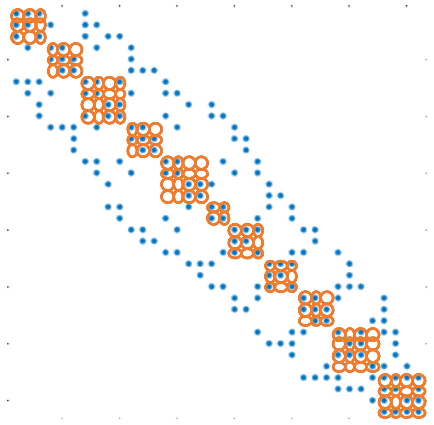
$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$

- Extension: block-Jacobi: $P = diag_B(A)$
  - Set of diagonal blocks
  - Treat each block as linear system
  - Larger blocks
    - better convergence,
    - more expensive to compute


1.
Extract diagonal block from sparse data structure.
...
2.
Invert diagonal block.
3.
Insert inverse as diagonal block into preconditioner matrix.

Motivation: Block-Jacobi Preconditioning

- Determination of block necessary
- Unproblematic if information about system is known apriori
- **Cluster Analysis**
  - Works well
  - Highly expensive to compute



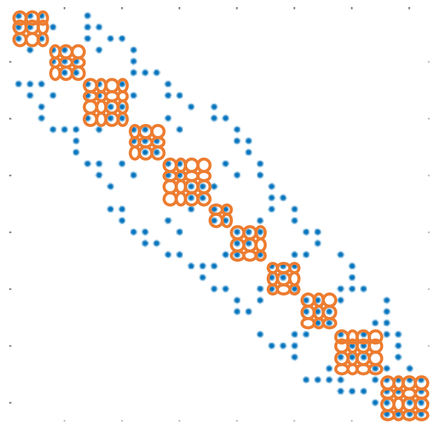
Motivation: Block-Jacobi Preconditioning

- Determination of block necessary

- Unproblematic if information about system is known apriori

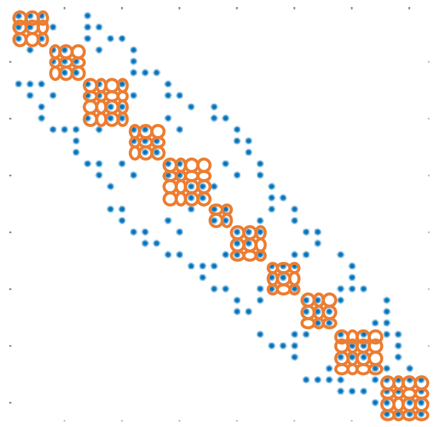
- **Cluster Analysis**
  - Works well
  - Highly expensive to compute

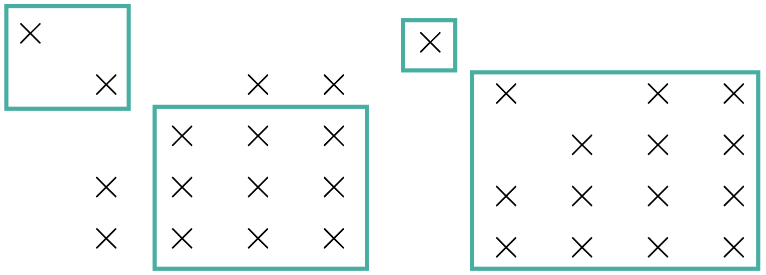
- **Supervariable Agglomeration (SVA)**
  - State-of-the-art
  - Results are okay
  - Sequential by definition



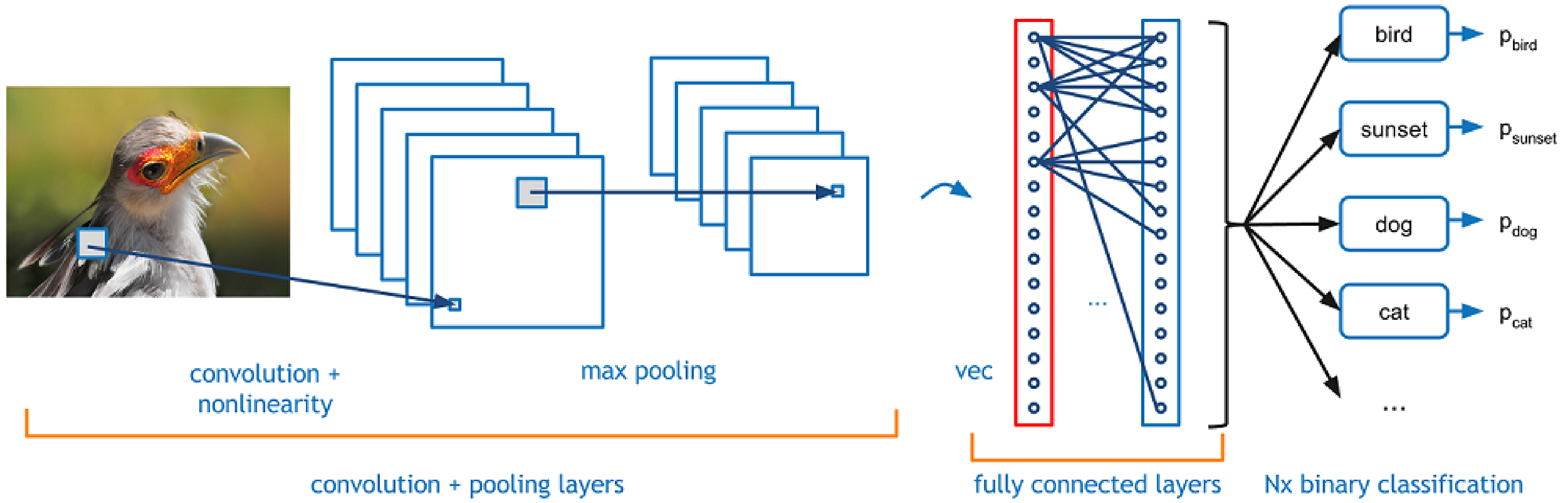
Motivation: Block-Jacobi Preconditioning

- Determination of block necessary
- Unproblematic if information about system is known apriori
- **Cluster Analysis**
  - Works well
  - Highly expensive to compute
- **Supervariable Agglomeration (SVA)**
  - State-of-the-art
  - Results are okay
  - Sequential by definition
- **„Looking at it“**
  - Heuristic
  - Feeling for natural blocks





Crash course: Convolutional Neural Networks (CNN)



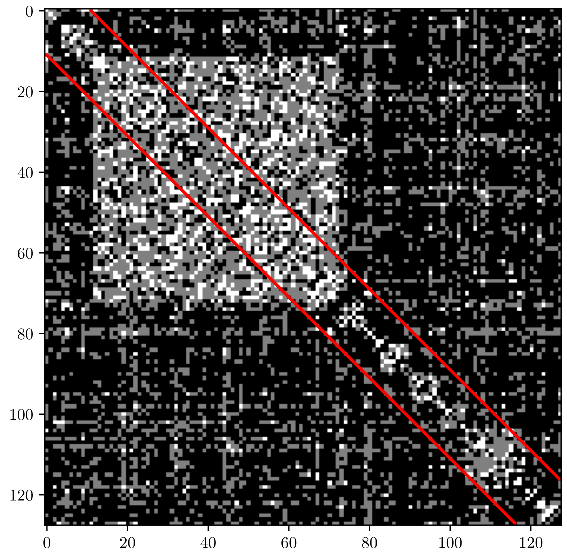
© https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

- Mimics behaviour of biological eyes
- Efficient in detecting recurring patterns

Idea – Let CNN „look at“ sparsity pattern

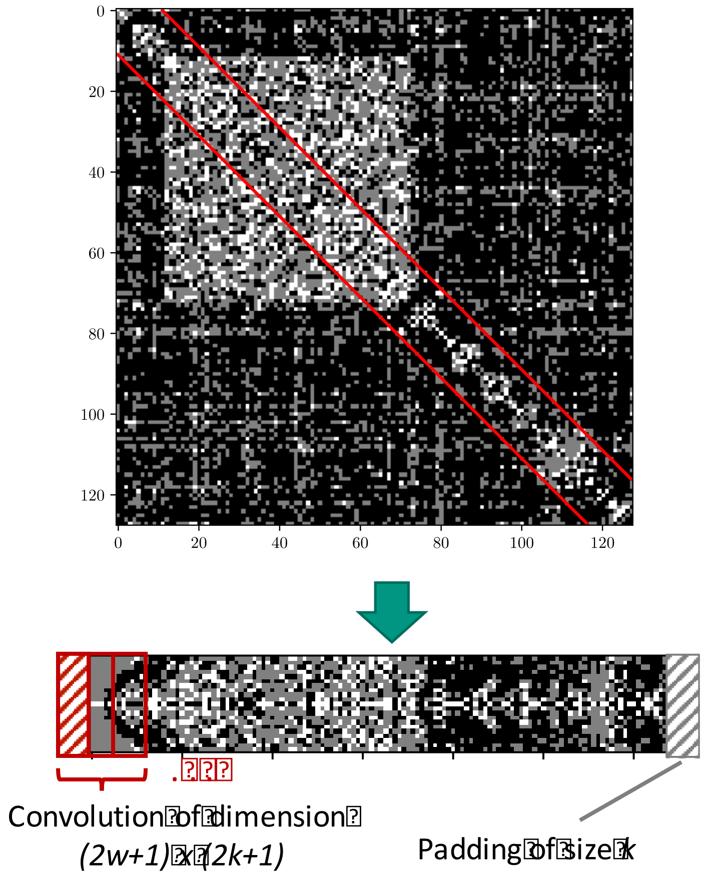
- Interpret matrix as image

- Mainly interested in non-zero **patterns close to main diagonal**



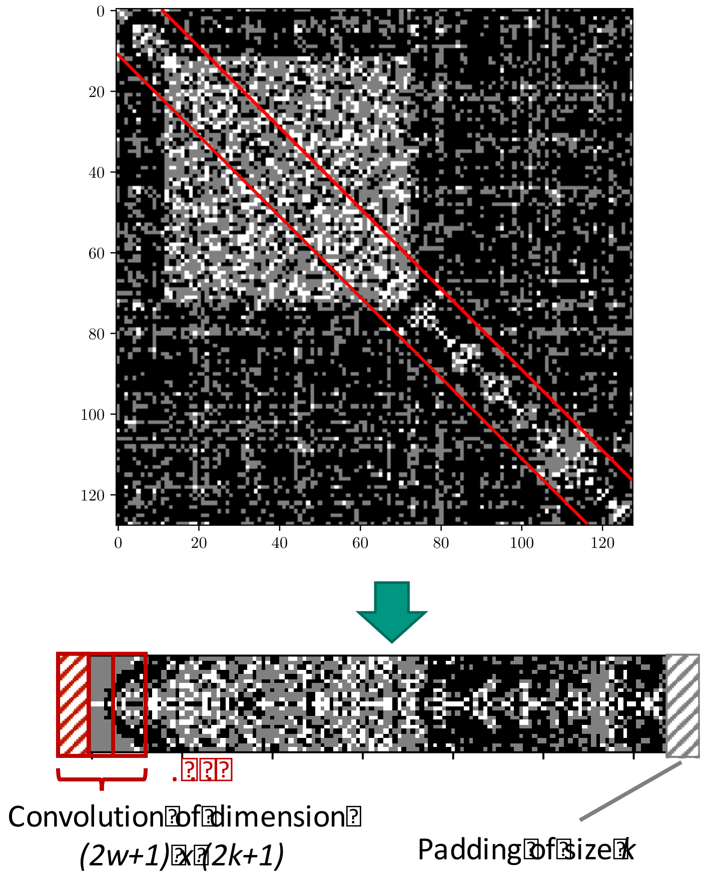
Idea – Let CNN „look at“ sparsity pattern

- Interpret matrix as image
- Mainly interested in non-zero **patterns close to main diagonal**
- Need **dataset with labels**
  - Other algorithms set upper bound
  - No credit card for Amazon Mturk



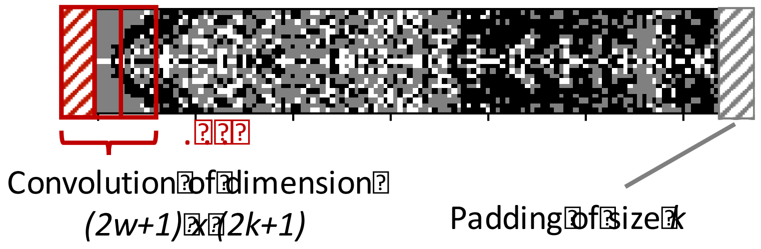
Idea – Let CNN „look at“ sparsity pattern

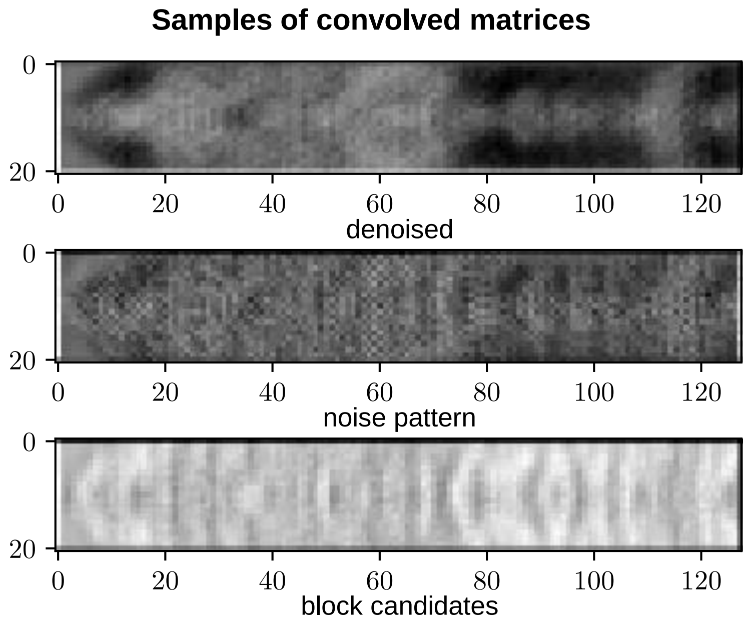
- Interpret matrix as image
- Mainly interested in non-zero **patterns close to main diagonal**
- Need **dataset with labels**
  - Other algorithms set upper bound
  - No credit card for Amazon Mturk
- Generate **artificial data**
  - Boundaries known
  - 3000 matrices
  - Uniform size, 128x128
  - $w = 10$
  - Random block + gaussian noise ($\mu = 10$)



Implementation

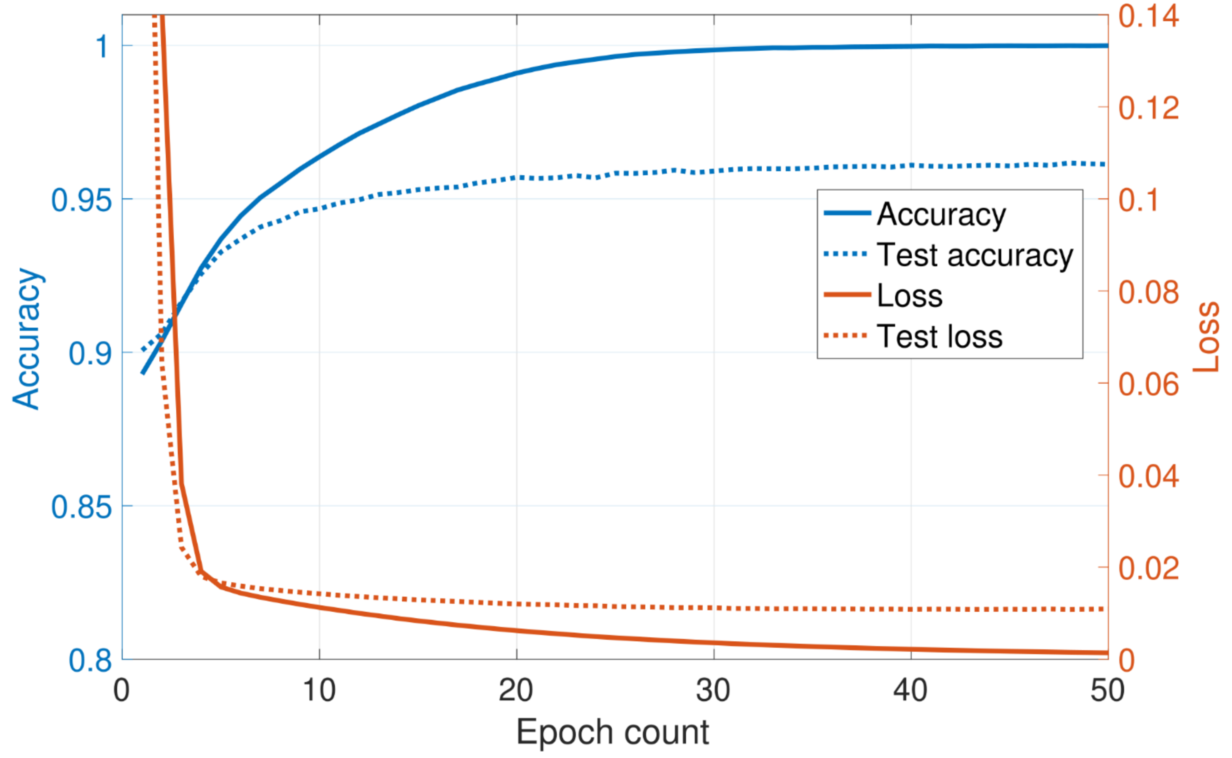
- Single block **ResNet** architecture
  - 4 Convolutional layers
  - 1 Fully-connected prediction layer
  - Dropout and L2 regularization
- **Open-source** script in Python
  - https://github.com/Markus-Goetz/block-prediction
  - Keras + TensorFlow for CNN
  - HDF5 I/O
- Processing performance
  - Single nVidia K80
  - Batch size=1, 3000 matrices in 2.5s
  - Batch size=1500, **3000 matrices in 0.6ms**





CNN Training Process

- Loss function: $\mathcal{L}(y,\hat{y}) = -\sum_{s=1}^{S}\sum_{i=1}^{n} y_{s,i} * log(\hat{y}_{s,i}) + (1-y_{s,i}) * log(1-\hat{y}_{s,i}).$



Prediction Performance Evaluation

$$precision(y,\hat{y}) = \frac{tp(y,\hat{y})}{tp(y,\hat{y}) + fp(y,\hat{y})}$$

$$recall(y,\hat{y}) = \frac{tp(y,\hat{y})}{tp(y,\hat{y}) + fn(y,\hat{y})}$$

$$F1(y,\hat{y}) = 2 * \frac{precision(y,\hat{y}) * recall(y,\hat{y})}{precision(y,\hat{y}) + recall(y,\hat{y})}$$

y	**Labels**
$\hat{y}$	**Prediction**
tp	**True-positives**
fp	**False-positives**
fn	**False-negatives**

CNN

		Actual		
Acc.:	0.9617	no block	block	precision
Pred.	no block	68010	553	0.9919
	block	2389	5848	0.7100
	recall	0.9661	0.9136	**F1:** 0.7990

SVA10

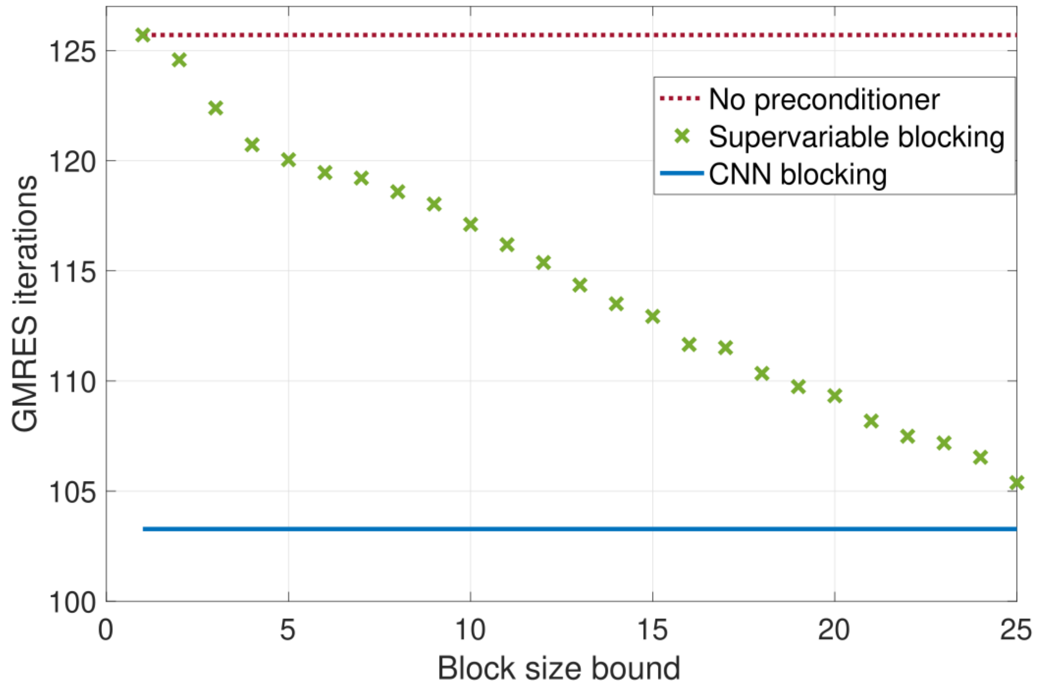
		Actual		
Acc.:	0.8261	no block	block	precision
Pred.	no block	62105	6458	0.9107
	block	6895	1342	0.1547
	recall	0.9001	0.1721	**F1:** 0.1673

SVA25

		Actual		
Acc.:	0.8695	no block	block	precision
Pred.	no block	65872	2691	0.9608
	block	7328	909	0.1103
	recall	0.8998	0.2525	**F1:** 0.1535

High-level Performance Analysis

- Used predicted block boundaries in a Jacobi preconditioner
- 600 test matrices
- Average iteration count
- ~22% less iteration with CNN compared to no blocks



Summary and Outlook

Summary

- Used **CNN** to predict **blocks** for **Jacobi** block preconditioners
- Reduction in solver iterations
- **Parallel** and fast **prediction of blocks**, usable on **GPUs**

Next Steps

- Manually label matrices
- Adaptability to other preconditioners (ILU/ILUT)
- Robustness study of CNN architecture and input data