ICL Lunch Talk

November 2nd 2018

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

Markus Götz, Hartwig Anzt

"*Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation,*" accepted at ScalA'18: 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems WS at SC18.

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

KIT Karlsruher Institut für Technologie

Motivation: Block-Jacobi Preconditioning

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

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

  - Can be used as preconditioner: $\tilde{A} = P^{-1}A,\ \tilde{b} = P^{-1}b.$

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

Motivation: Block-Jacobi Preconditioning

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

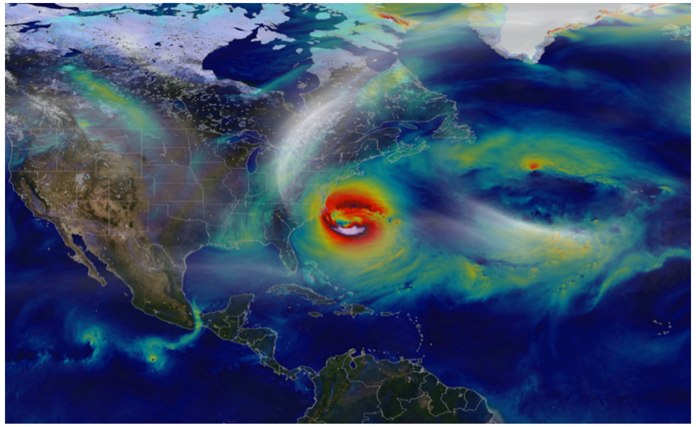
$$x^{(k+1)} = x^{(k)} + P^{-1}b - P^{-1}Ax^{(k)}$$

  - Can be used as preconditioner: $\tilde{A} = P^{-1}A,\, \tilde{b} = P^{-1}b.$

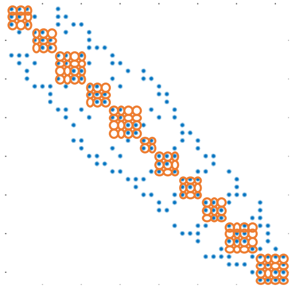
$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$

- **Block-Jacobi** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Large set of small diagonal blocks.
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically **improve convergence**.
    - *Larger* blocks make block-Jacobi **more expensive**.

      Extreme case: one block of matrix size.



https://science.nasa.gov/earth-science/focus-areas/earth-weather



Motivation: Block-Jacobi Preconditioning

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

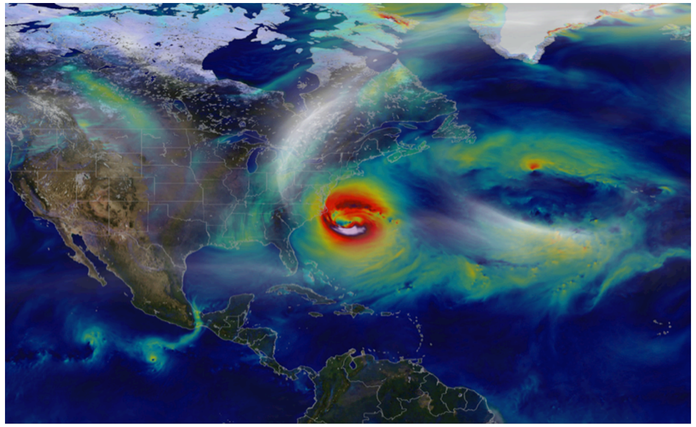
$$x^{(k+1)} = x^{(k)} + P^{-1}b - P^{-1}Ax^{(k)}$$

  - Can be used as preconditioner: $\tilde{A} = P^{-1}A,\ \tilde{b} = P^{-1}b.$

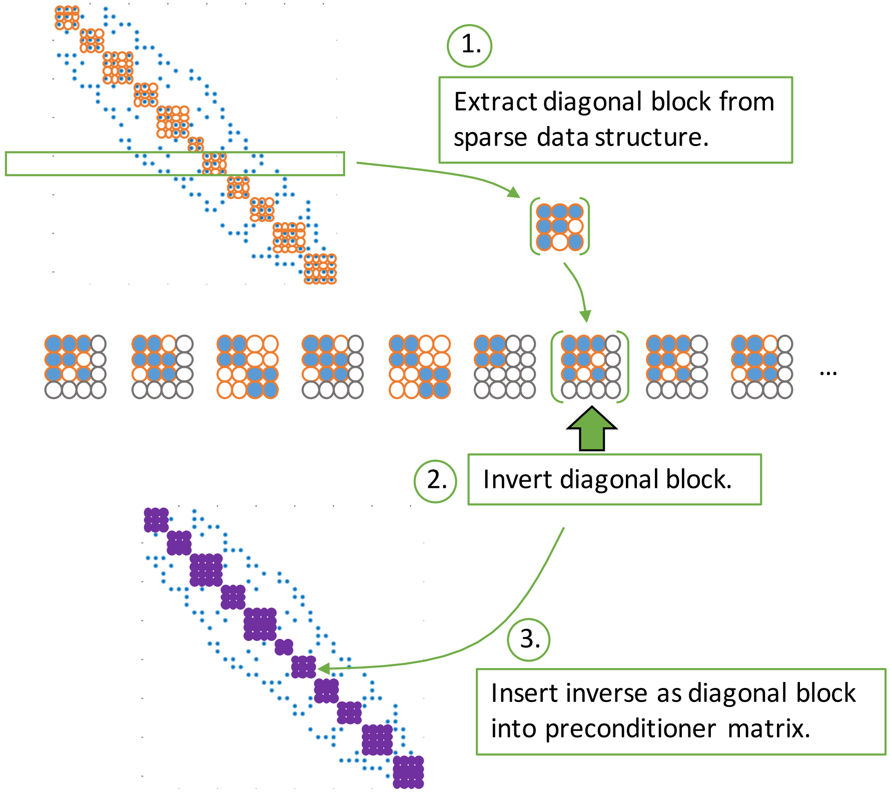
$$Ax = b \Leftrightarrow \tilde{A}x = \tilde{b}$$

- **Block-Jacobi** is based on **block-diagonal scaling**: $P = diag_B(A)$
  - Large set of small diagonal blocks.
  - Each block corresponds to one (small) linear system.
    - *Larger* blocks typically **improve convergence**.
    - *Larger* blocks make block-Jacobi **more expensive**.

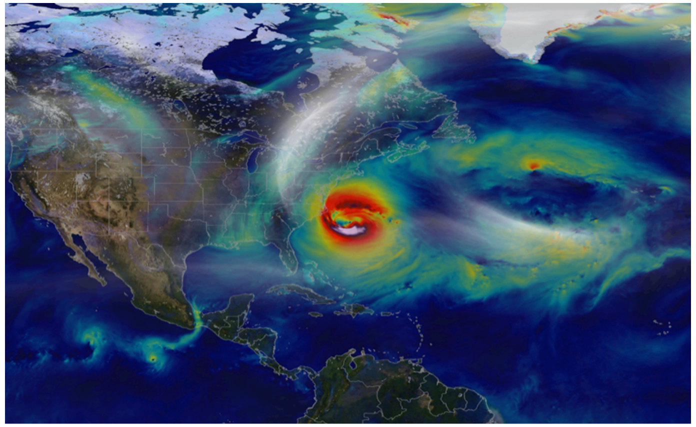
    Extreme case: one block of matrix size.



https://science.nasa.gov/earth-science/focus-areas/earth-weather



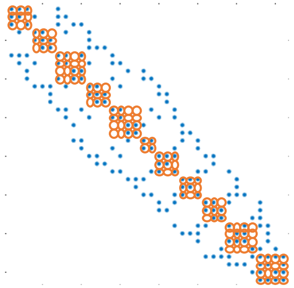
Motivation: Block-Jacobi Preconditioning



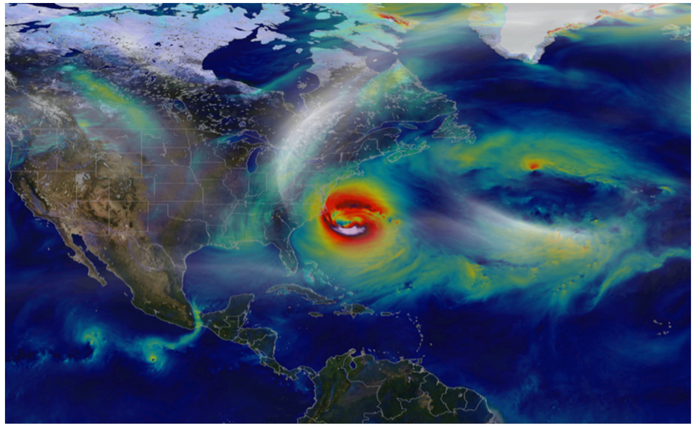
https://science.nasa.gov/earth-science/focus-areas/earth-weather

How do we determine efficient diagonal blocks?

- Strongly connected components.
- Variables associated to same discretization element.



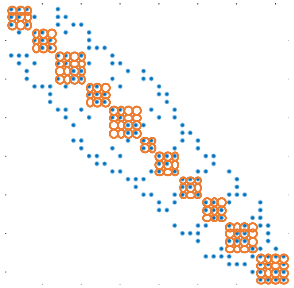
Motivation: Block-Jacobi Preconditioning



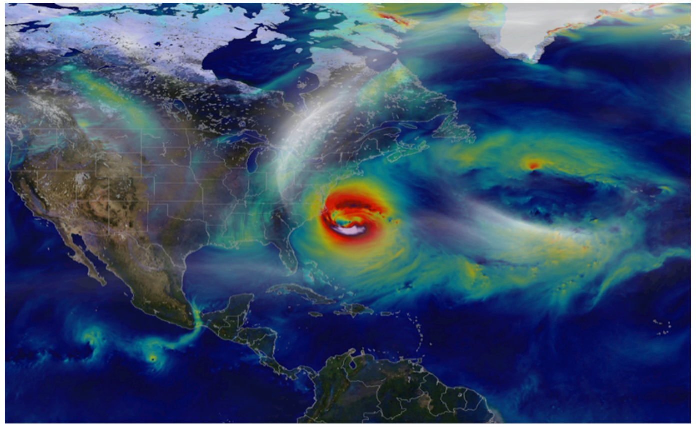
https://science.nasa.gov/earth-science/focus-areas/earth-weather

How do we determine efficient diagonal blocks?

- If we know the discretization method, information is available.
- What if the problem's origin / discretization scheme is unknown?



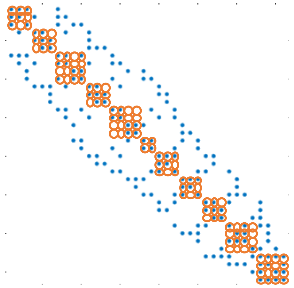
Motivation: Block-Jacobi Preconditioning



https://science.nasa.gov/earth-science/focus-areas/earth-weather

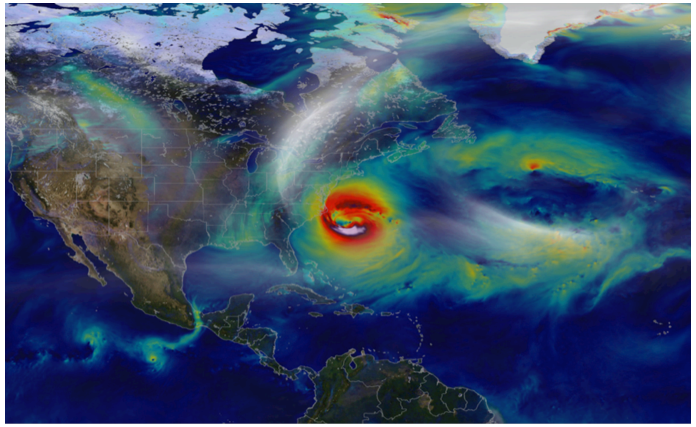
How do we determine efficient diagonal blocks?

- If we know the discretization method, information is available.

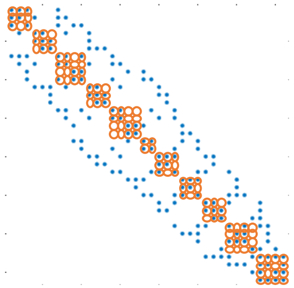


- What if the problem's origin / discretization scheme is unknown?
  - Clustering algorithms (Big Data analytics)
    "work well"
    "extremely expensive"

Motivation: Block-Jacobi Preconditioning

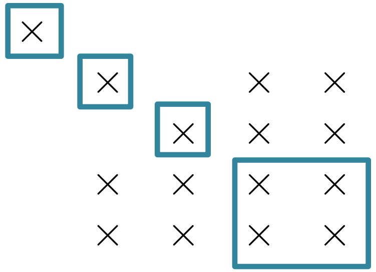


https://science.nasa.gov/earth-science/focus-areas/earth-weather

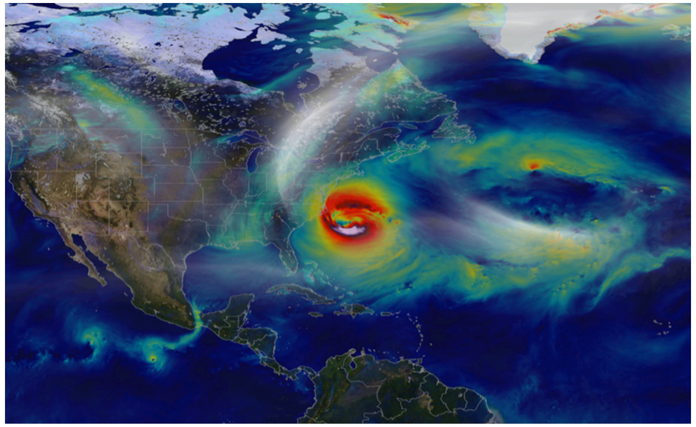


How do we determine efficient diagonal blocks?

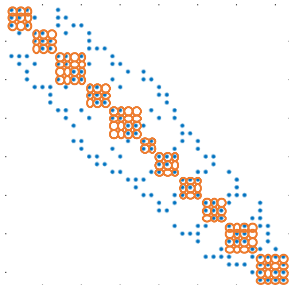
- If we know the discretization method, information is available.
- What if the problem's origin / discretization scheme is unknown?
  - Clustering algorithms (Big Data analytics)
    "work well"
    "extremely expensive"
  - Supervariable agglomeration:
    "stat-of-the-art"
    "works somewhat well"
    1. detect rows with same pattern
    2. accumulate to block size bound (4)



Motivation: Block-Jacobi Preconditioning

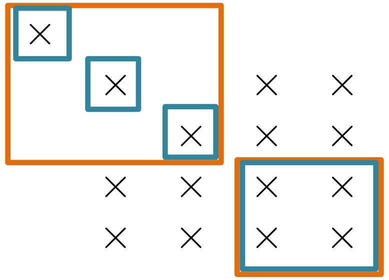


https://science.nasa.gov/earth-science/focus-areas/earth-weather

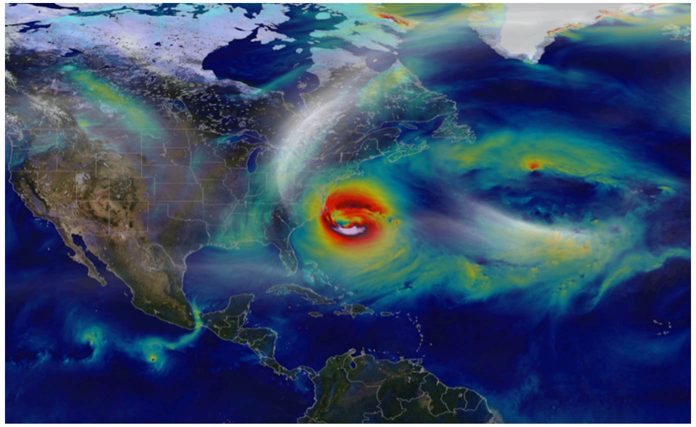


How do we determine efficient diagonal blocks?

- If we know the discretization method, information is available.
- What if the problem's origin / discretization scheme is unknown?
  - Clustering algorithms (Big Data analytics)
    "work well"
    "extremely expensive"
  - Supervariable agglomeration:
    "stat-of-the-art"
    "works somewhat well"
    1. detect rows with same pattern
    2. accumulate to block size bound (4)

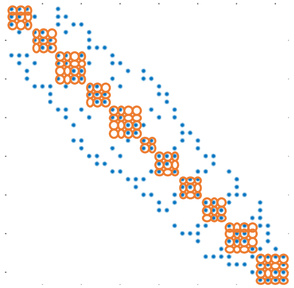


Motivation: Block-Jacobi Preconditioning

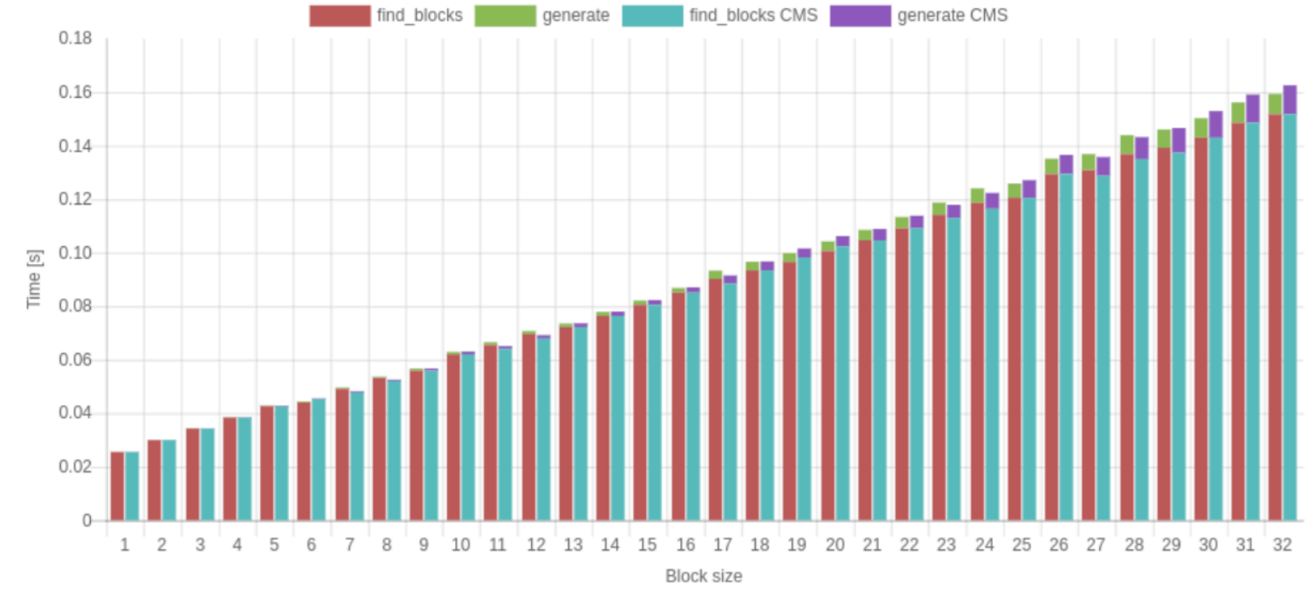


https://science.nasa.gov/earth-science/focus-areas/earth-weather

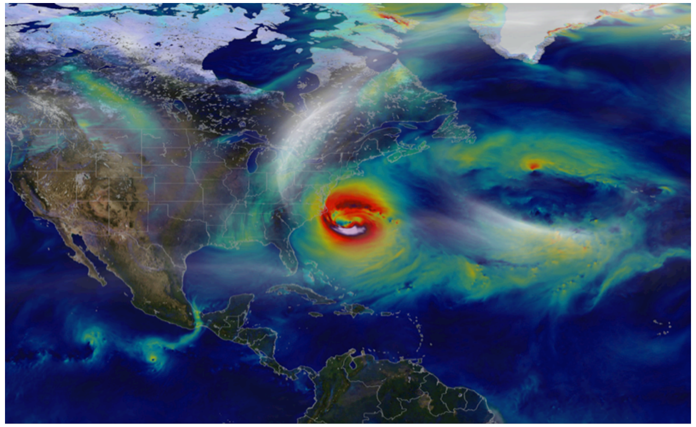
How do we determine efficient diagonal blocks?



- If we know the discretization method, information is available.
- What if the problem's origin / discretization scheme is unknown?
  - Clustering algorithms (Big Data analytics)
    "work well"
    "extremely expensive"
  - Supervariable agglomeration:
    "stat-of-the-art"
    "works somewhat well"
    "sequential character makes it unattractive & expensive"



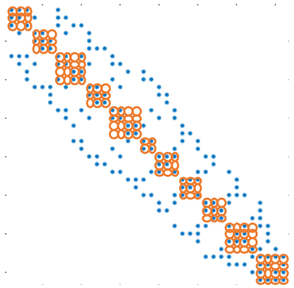
Motivation: Block-Jacobi Preconditioning



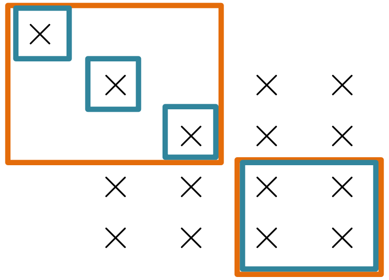
https://science.nasa.gov/earth-science/focus-areas/earth-weather

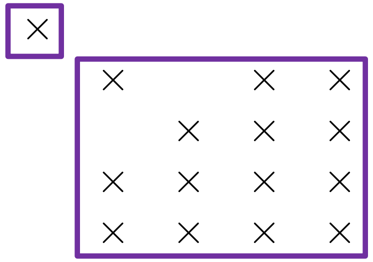
How do we determine efficient diagonal blocks?

- If we know the discretization method, information is available.



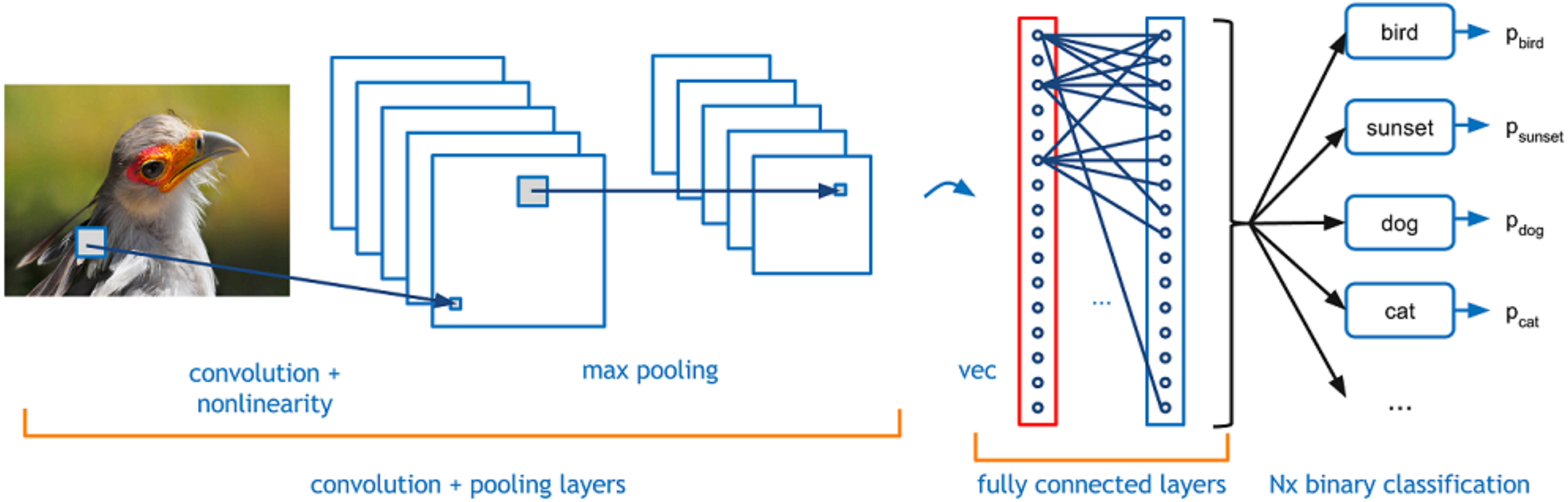
- What if the problem's origin / discretization scheme is unknown?
  - Clustering algorithms (Big Data analytics)
    "work well"
    "extremely expensive"
  - Supervariable agglomeration:
    "stat-of-the-art"
    "works somewhat well"
    "sequential character makes it
    unattractive & expensive"
  - "Look at it"
    "we have a feeling for what belongs together"





Motivation: Convolutional Neural Networks (CNN)

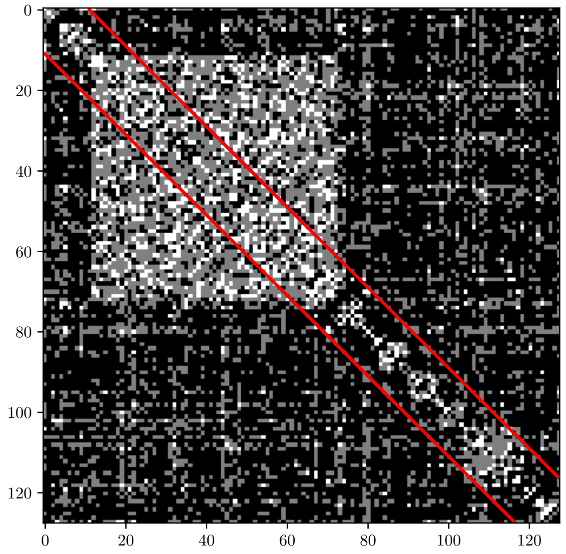
Very efficient in detecting recurring patterns.



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

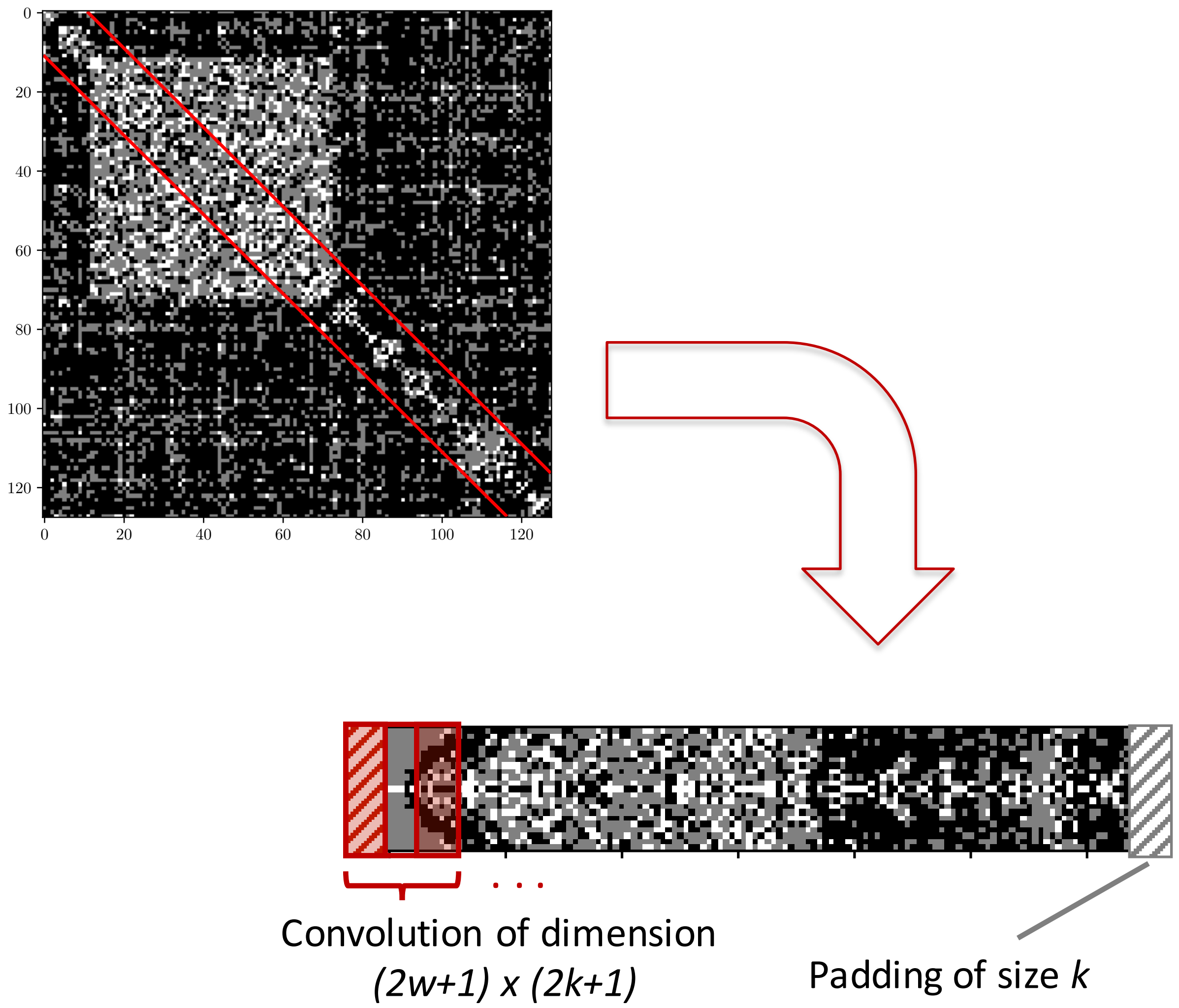
Idea: Take matrix sparsity pattern as input for CNN

- We are in particular interested in the nonzero pattern close to the main diagonal.



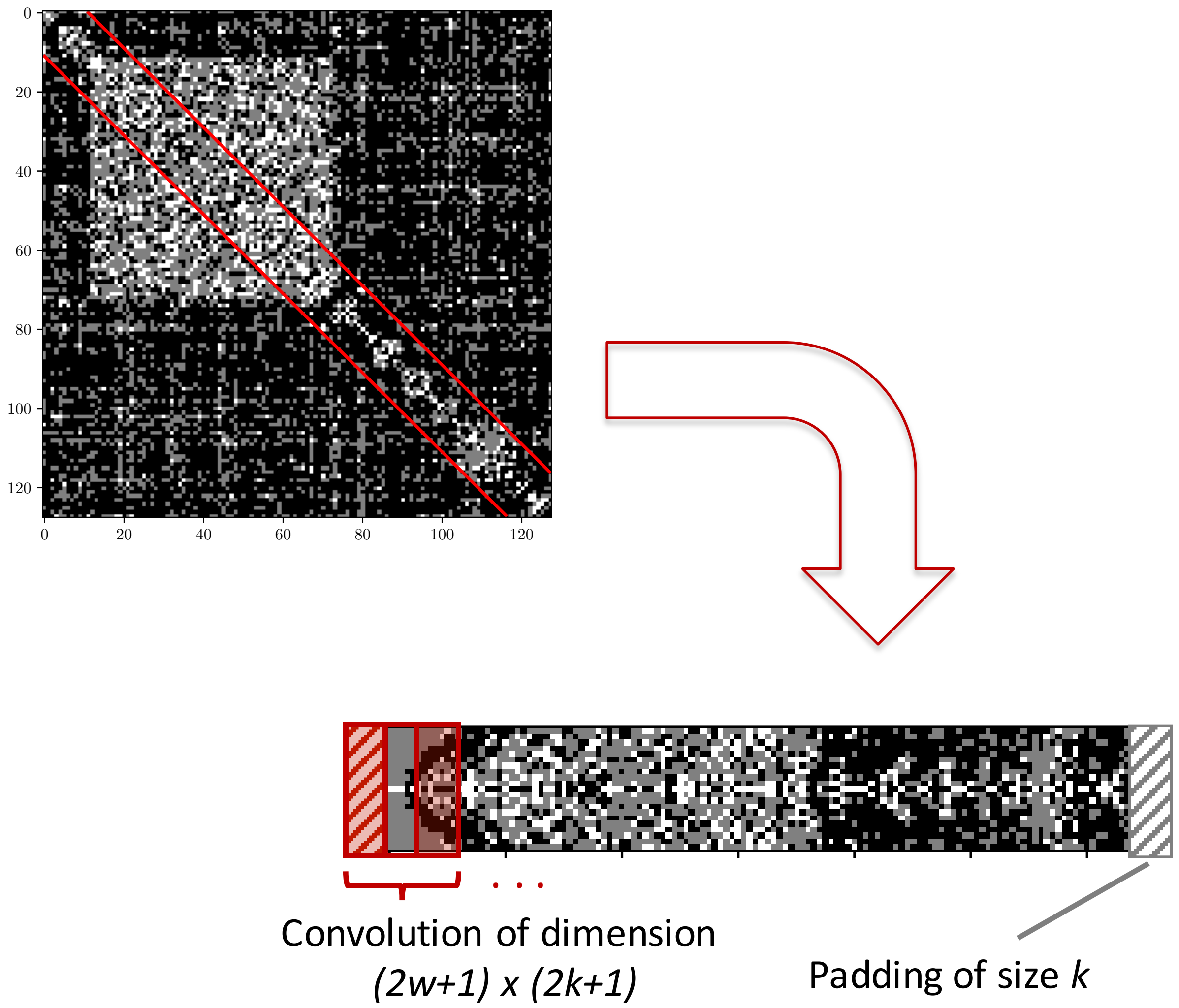
Idea: Take matrix sparsity pattern as input for CNN

- We are in particular interested in the nonzero pattern close to the main diagonal.



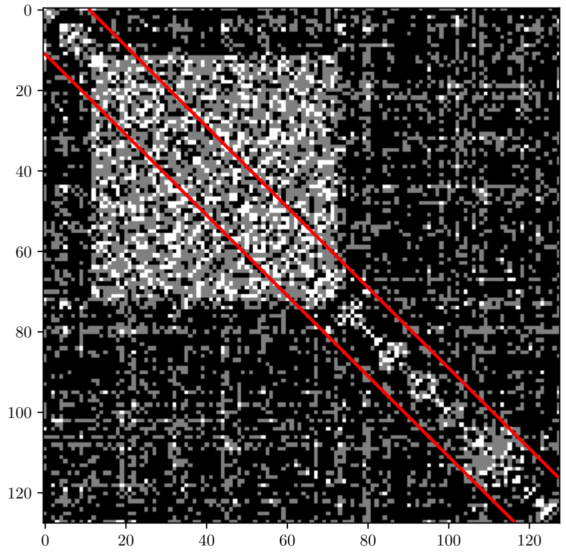
Idea: Take matrix sparsity pattern as input for CNN

- We are in particular interested in the nonzero pattern close to the main diagonal.
- We need uniform-sized input.
- For training the CNN, we need labeled data with block annotated.



Idea: Take matrix sparsity pattern as input for CNN

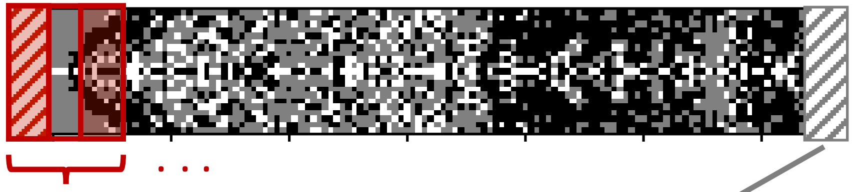
- We are in particular interested in the nonzero pattern close to the main diagonal.
- We need uniform-sized input.
- For training the CNN, we need labeled data with block annotated.



Hire Student Assistants for annotating real-world test problems.

worker unions

Generate artificial test matrices.

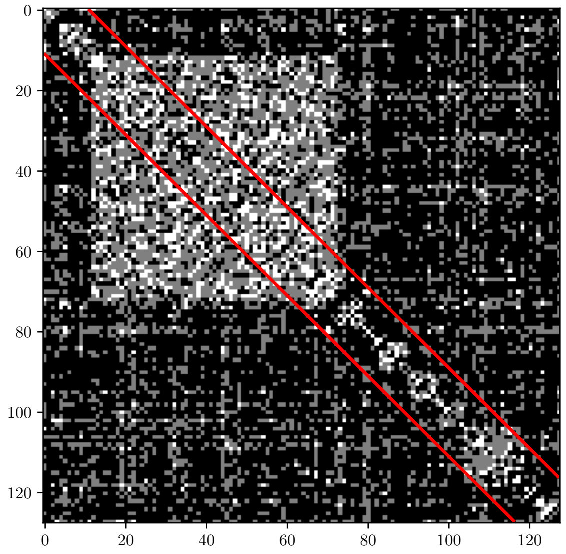


Convolution of dimension *(2w+1) x (2k+1)*

Padding of size *k*

Idea: Take matrix sparsity pattern as input for CNN

- We are in particular interested in the nonzero pattern close to the main diagonal.
- We need uniform-sized input.
- For training the CNN, we need labeled data with block annotated.

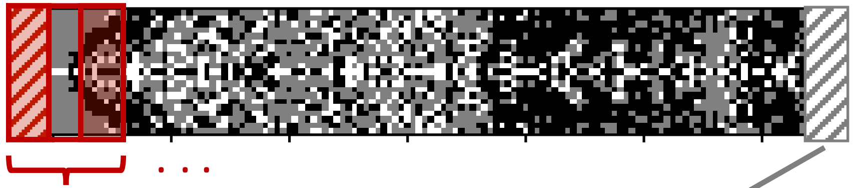


Hire Student Assistants for annotating real-world test problems.

worker unions

Generate artificial test matrices.

- 3,000 matrices of size 128x128
- *w=10*
- *random blocks with varying density distributions*
- *average size of blocks is 10*

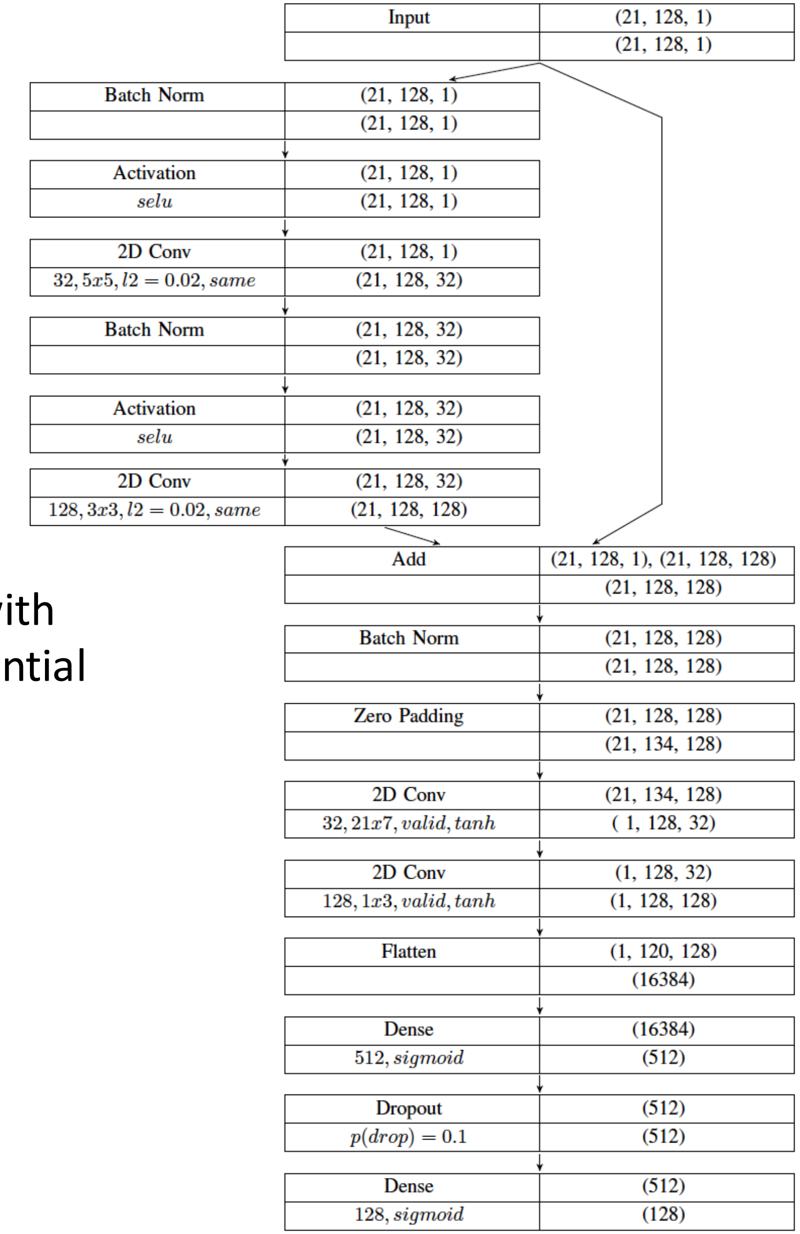


Convolution of dimension *(2w+1) x (2k+1)*

Padding of size *k*

Design a CNN for detecting diagonal blocks

- Keras v. 2.1.6 based on tensorflow v. 1.8.0
- numpy backend
- Nesterov-momentum optimizer
- Feed-forward CNN with 128 layers
- *More details in the paper*



1st block denoises image

two two-dimensional convolutional layers with post-batch normalization and scaled exponential linear units (SELU)

2nd block

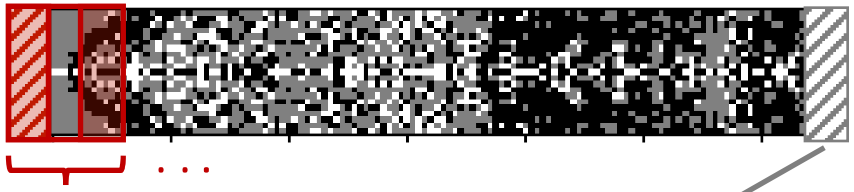
non-standard discrete convolutions,
activation via tanh function

3rd block identifies block starts

fully-connected dense layer,
identification via argmax function

Design a CNN for detecting diagonal blocks

- Keras v. 2.1.6 based on tensorflow v. 1.8.0
- numpy backend
- Nesterov-momentum optimizer
- Feed-forward CNN with 128 layers
- *More details in the paper*

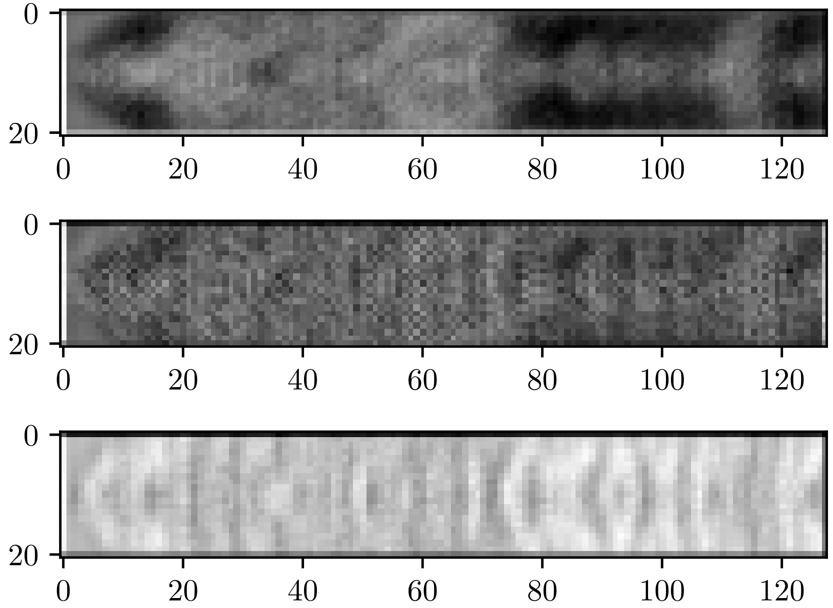


Samples of different layers:

denoised image

inverse content

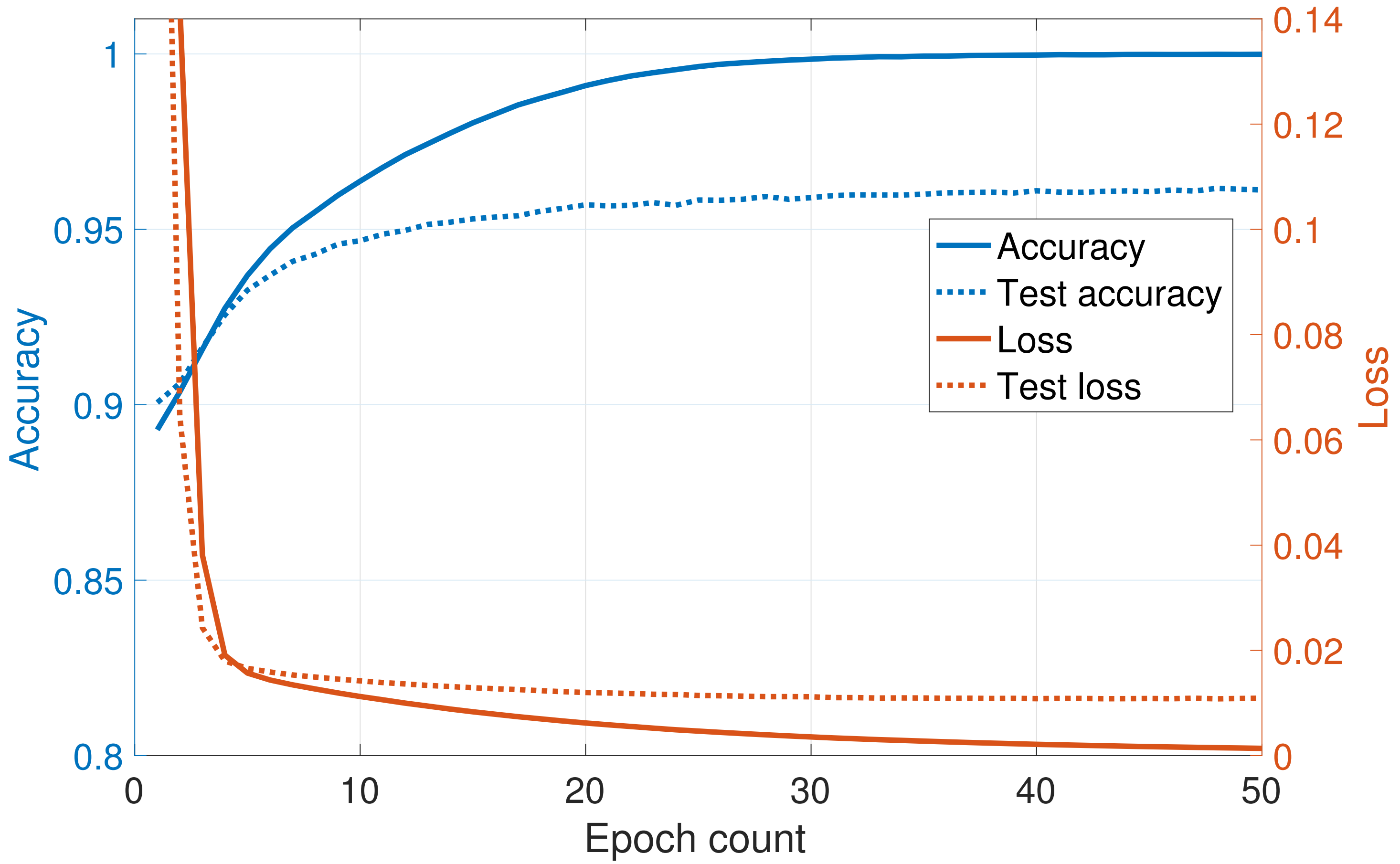
potential block boundaries



CNN training process

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

Training / Test distribution: 80% / 20%



CNN high-level quality analysis

Label vector y

Prediction vector $\hat{y}$

True positives tp

False positive fp

False negative fn

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

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

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

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

SVA-10

		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

SVA-25

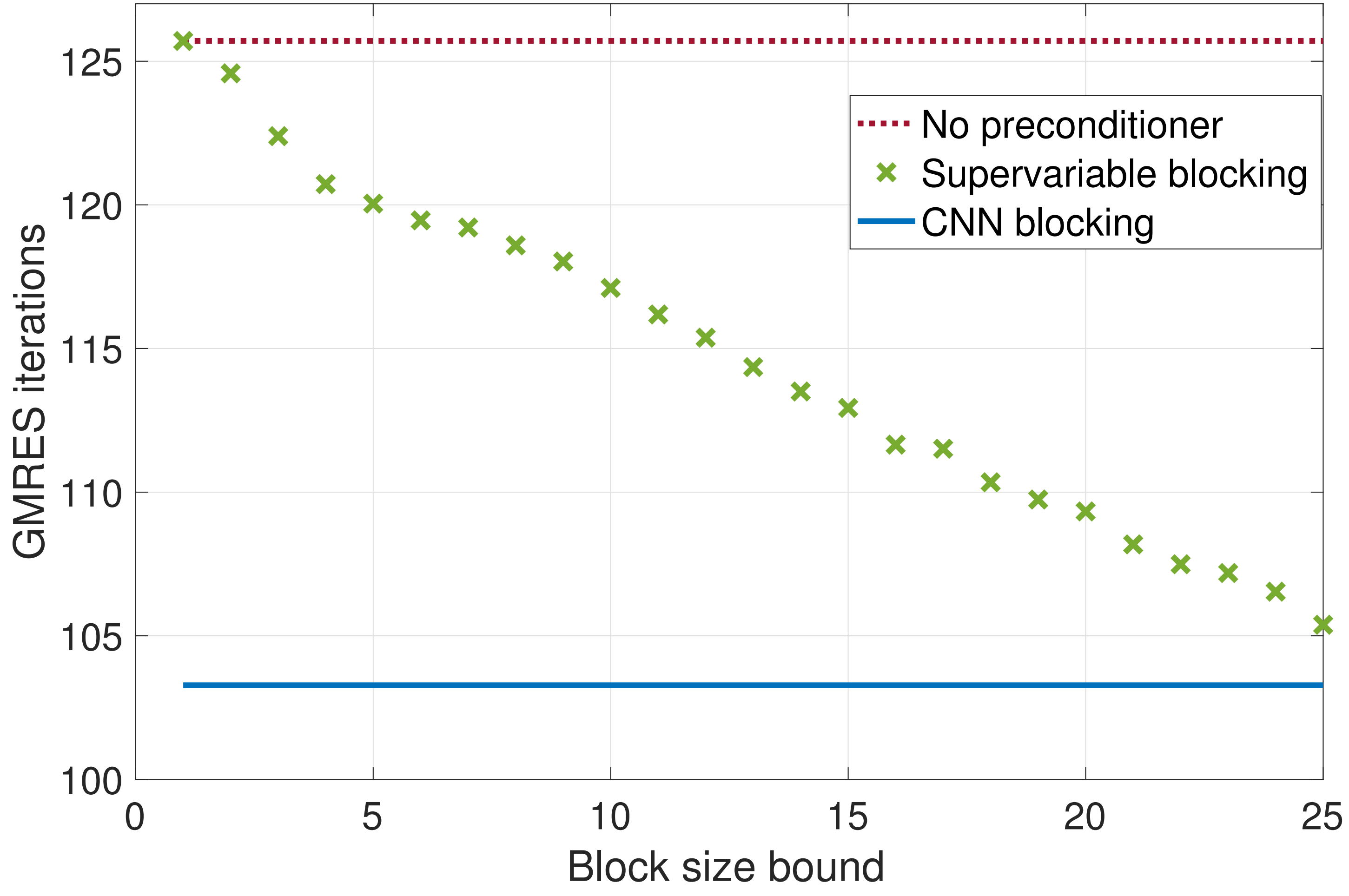
		Actual		
Acc.:	0.8695	no block	block	precision
Pred.	no block	65872	2691	0.9608
	block	7328	909	0.1103
	recall	0.8998	0.2525	**F1:** 0.1535

CNN high-level quality analysis

Use the predicted blocks for a block-Jacobi preconditioner.

Compare against uniform blockings.

Iterations averaged over test data (600 matrices).



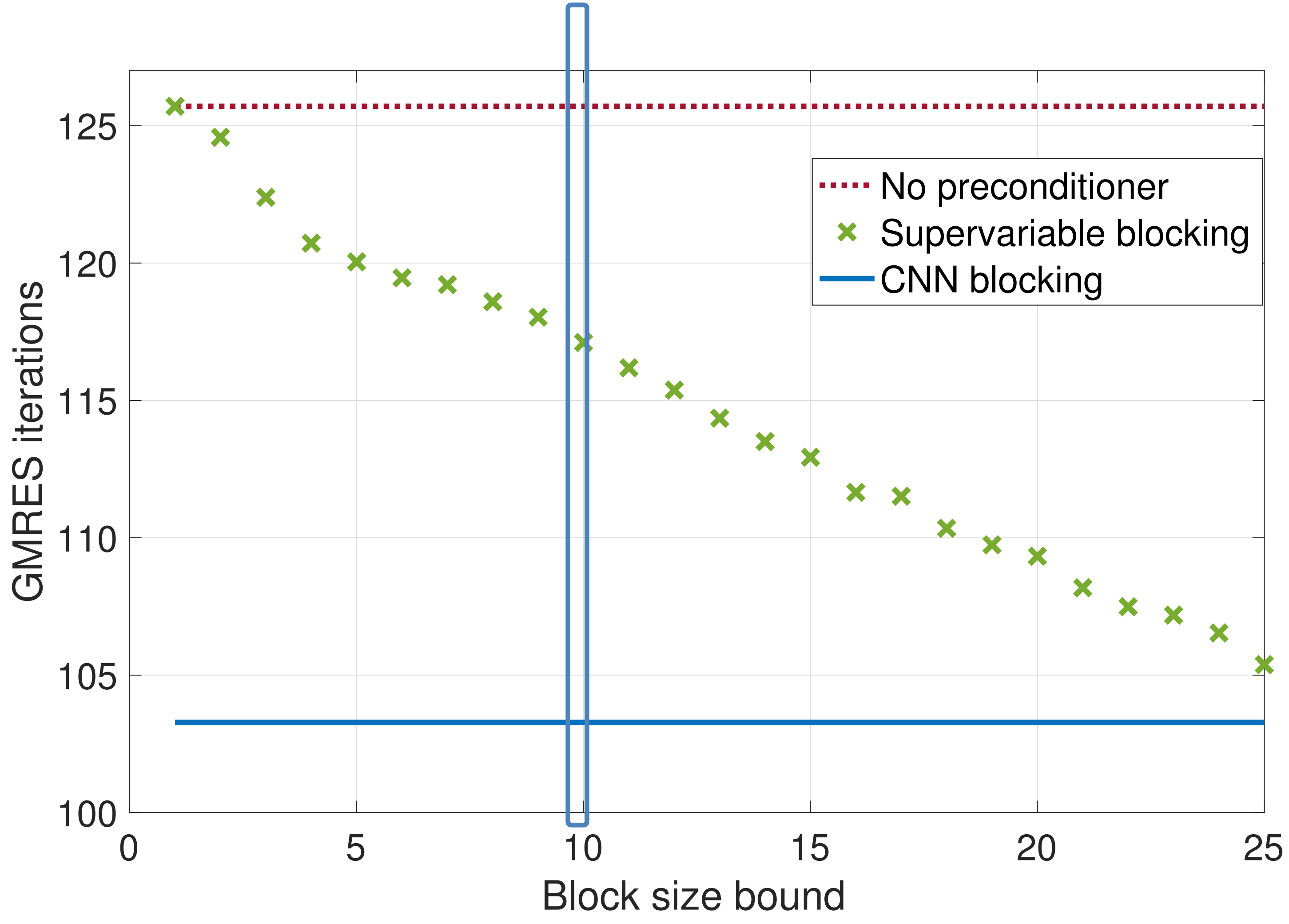
CNN high-level quality analysis

Use the predicted blocks for a block-Jacobi preconditioner.

Compare against uniform blockings.

Iterations averaged over test data (600 matrices).

9.8 is average block size predicted by CNN



CNN high-level quality analysis

Use the predicted blocks for a block-Jacobi preconditioner.

Compare against uniform blockings.

Iterations averaged over test data (600 matrices).

Next steps:

- **Real data (manually label data?)**
- **Other preconditioners (ILU/ILUT?)**