

# Operator Action Predictions

Dominykas Asauskas, Christian Contreras,  
Jean-Roch Vlimant

# Short Introduction

- Common issues are errors in grid jobs, that may be due to missing/corrupt input files, high memory usage, etc.
- Currently all handled manually in Operations:
  - An operator must look at the error codes and decide on what appropriate actions to take on the workflow
- General Goal:
  - Deliver error handling predictions and mature towards moving manual operator intervention into automated actions

For more info look at Chrisians slides (slides 2-4):

[https://github.com/chrisjcc/WorkflowWebTools/blob/ErrorHandlingAI/docs/Christian\\_Workflow\\_Handling\\_PR.pdf](https://github.com/chrisjcc/WorkflowWebTools/blob/ErrorHandlingAI/docs/Christian_Workflow_Handling_PR.pdf)

# Short Introduction

Having errors that happened and where happened  
(at which site) predict (total examples = 9522):

- Which *action* to choose
- Requested *memory*
- Job *splitting*
- Enabling *xrootd* or not

# Inputs / X

- Each example consists of matrix w/ shape (errors,sites)
- On average 99.75% of all numbers in matrix will be 0
- 66% out of all combinations (error x site) are never used (are always 0) in dataset with 9522 examples

	site 1	site 2	site 3	site 4	...	site 140	site 141	site 142
error 1	1	1	1	1	1	1	1	1
error 2	4	0	0	0	0	0	0	0
error 3	8	1	0	0	0	0	0	0
...	0	2	0	0	0	0	2	0
error 52	0	0	0	0	0	0	0	5
error 53	0	0	0	0	0	0	0	1
error 54	1	0	0	0	0	0	0	0

single example shape = (54,142)

total examples = 9522

For more stats which errors are most common or examples look at Chrisians slides 5-7:

[https://github.com/chrisjcc/WorkflowWebTools/blob/ErrorHandlingAI/docs/Christian\\_Workflow\\_Handling\\_PR.pdf](https://github.com/chrisjcc/WorkflowWebTools/blob/ErrorHandlingAI/docs/Christian_Workflow_Handling_PR.pdf)

# Outputs / Y Targets

## Action

1. acdc (88%)
2. clone (10%)
3. special (2%)

## Splitting

1. 98.5% default
2. 0.59% 10x
3. 0.31% 2x
4. 0.19% max
5. 0.12% 100x
6. 0.09% 20x
7. 0.05% 3x
8. 0.03% 50x

## Memory

- |     |               |     |            |
|-----|---------------|-----|------------|
| 1.  | 92.6% default | 14. | 0.03% 2k   |
| 2.  | 2.36% 20k     | 15. | 0.03% 19k  |
| 3.  | 1.55% 18k     | 16. | 0.03% 16k  |
| 4.  | 0.59% 12k     | 17. | 0.02% 11k  |
| 5.  | 0.54% 4k      | 18. | 0.01% 40k  |
| 6.  | 0.51% 9k      | 19. | 0.01% 32k  |
| 7.  | 0.40% 6k      | 20. | 0.01% 30k  |
| 8.  | 0.37% 10k     | 21. | 0.01% 3k   |
| 9.  | 0.25% 5k      | 22. | 0.01% 28k  |
| 10. | 0.19% 8k      | 23. | 0.01% 25k  |
| 11. | 0.17% 14k     | 24. | 0.01% 20k  |
| 12. | 0.06% 7k      | 25. | 0.01% 180k |
| 13. | 0.05% 15k     | 26. | 0.01% 15k  |

## Xrootd

1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

# Outputs / Y Targets (merged)

## Action

1. acdc (88%)
2. clone (10%)
3. special (2%)

## Memory

1. default (92%)
2. 18k-20k (4%)
3. 2k-9k (2%)
4. 10k-16k (1.2%)

## Splitting

1. default (98.5%)
2. 10x-100x (1%)
3. 2x-3x (0.4%)

## Xrootd

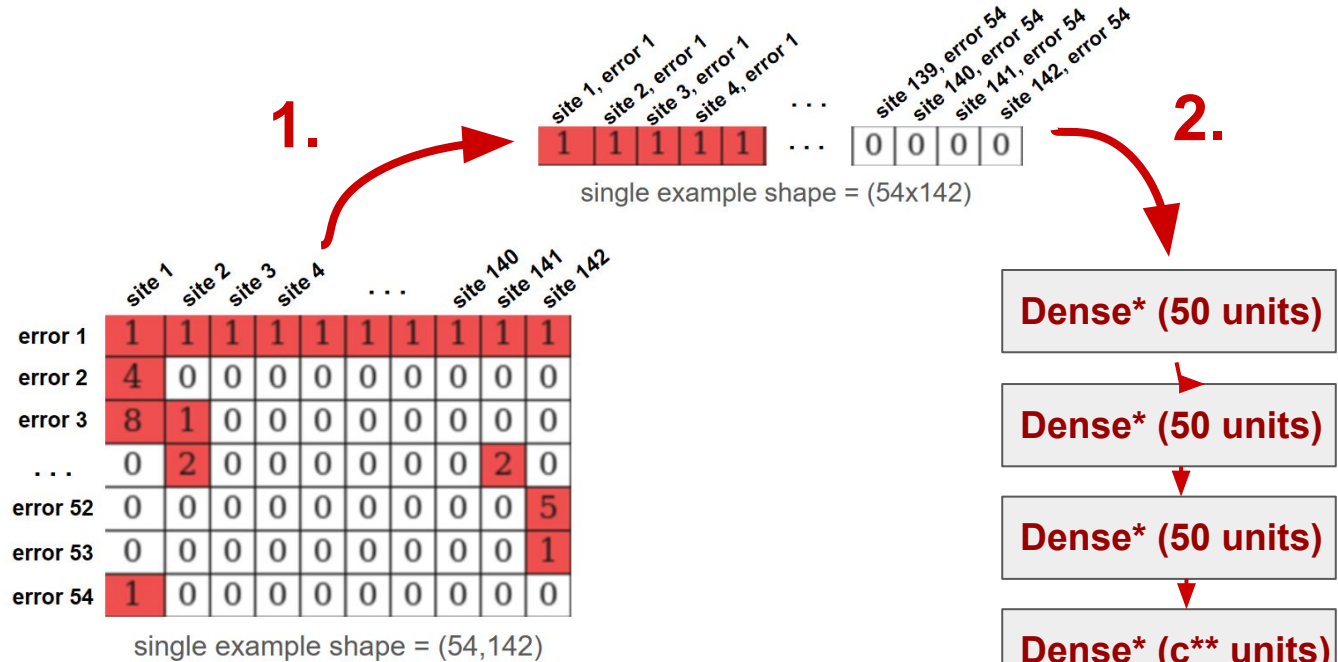
1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

# Models Ideas

- Because lack of examples, try to minimize model parameters (CNN Model)
- Change input so that model would generalize better (Create embeddings for each site and error)
- Try different ways of fighting class imbalance (SMOTE resampling, weighted cross entropy)
- Try to find a way to explain the output (by adding additional layers to network)
- Compare everything with simple feed forward network (Simple model)

# Simple Model

1. inputs are flattened
2. fed to feed forward neural network
3. everything is optimized with weighted cross entropy (wCE)



\* - All Dense layers are with 0.2 dropout w/ relu activations

\*\* - number of classes

\*\*\* - 
$$W_{y_i} = \frac{\text{total examples}}{\# \text{ classes}} \times \frac{1}{\# y_i \text{ examples}}$$

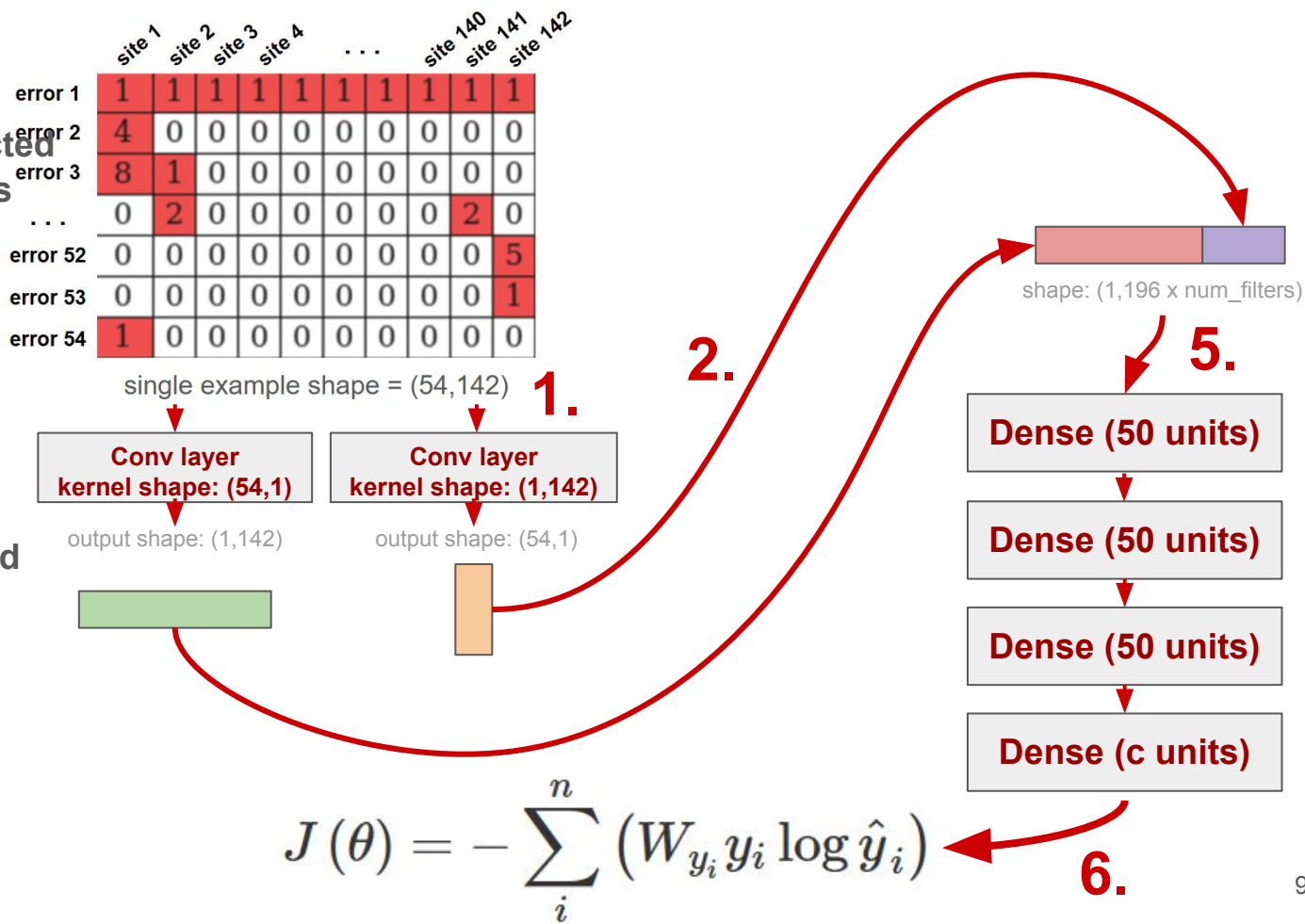
$$J(\theta) = - \sum_i^n (W_{y_i}^{***} y_i \log \hat{y}_i)$$

\*\* when resampling (using SMOTE) all classes become same size, which results in  $W = 1$  and loss = cross entropy<sub>8</sub>



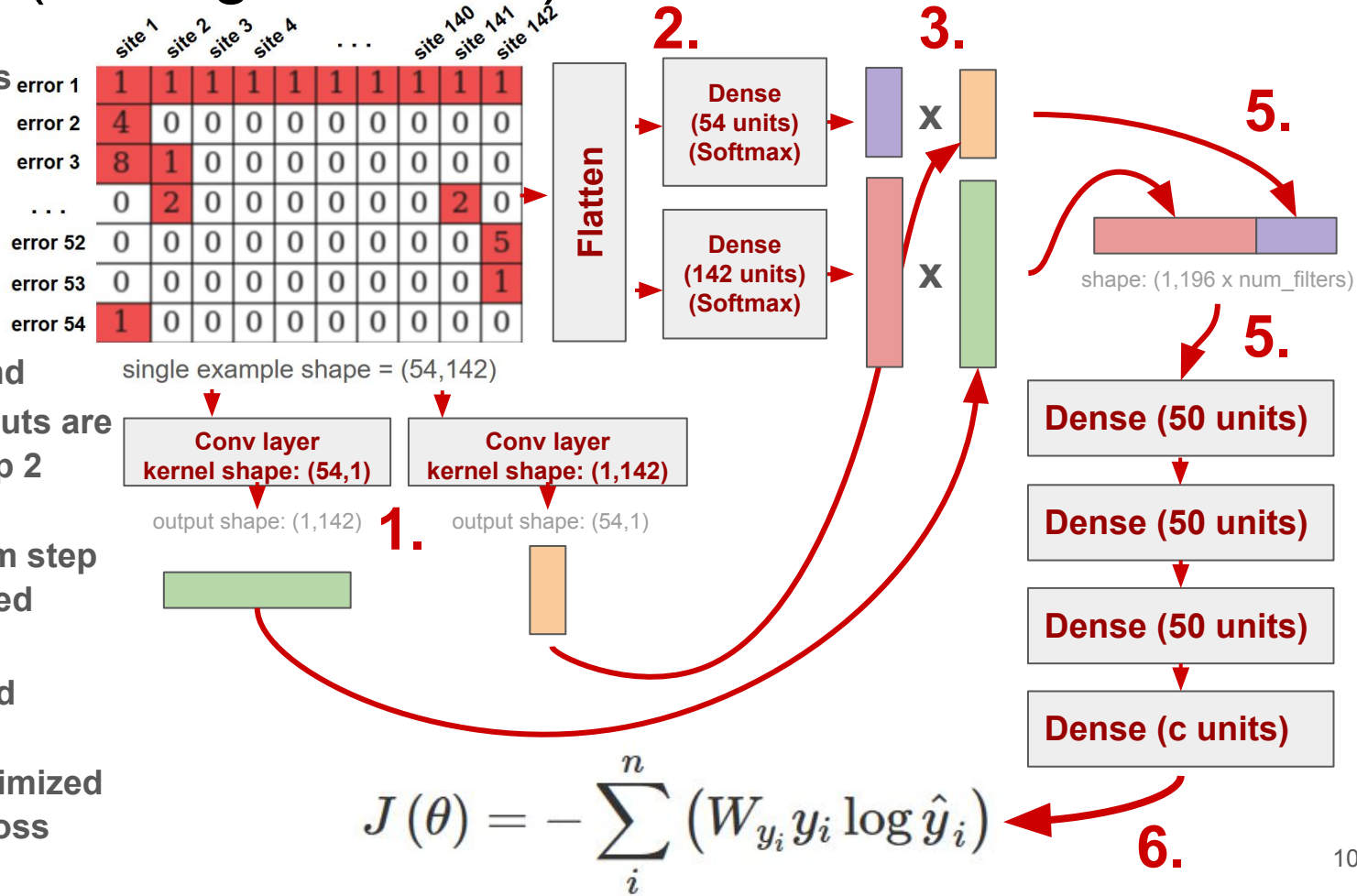
# CNN Model

1. input matrix is connected to 2 convolution layers across both axis
2. convolution layer outputs for each filter are concatenated
3. concat vector is connected to feed forward layers
4. everything is optimized with weighted cross entropy (wCE)



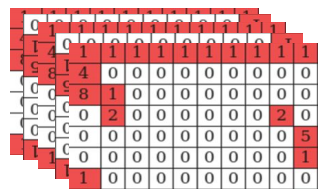
# CNN Model (adding Attention)

1. input matrix is convolved across both axis
2. input matrix is flattened and connected to 2 feed forward layers with softmax at the end
3. convolution outputs are multiplied by step 2 outputs
4. both outputs from step 3 are concatenated
5. concat vector is connected to feed forward layers
6. everything is optimized with weighted cross entropy (wCE)

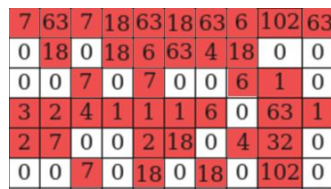


# Error and Site embeddings

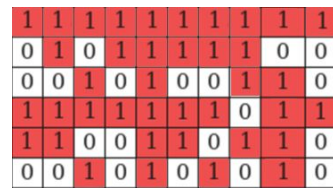
1. all examples are summed across axis 0
2. all numbers that are greater than 0 are replaced by 1
3. by doing dot product\* of error embedding matrix times sites embedding matrix try to recreate matrix from step 2
4. everything is optimized with mse



shape = (9522,54,142)



shape = (54,142)

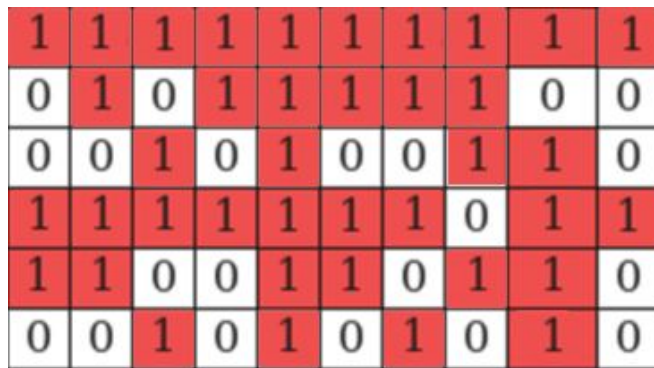


shape = (54,142)

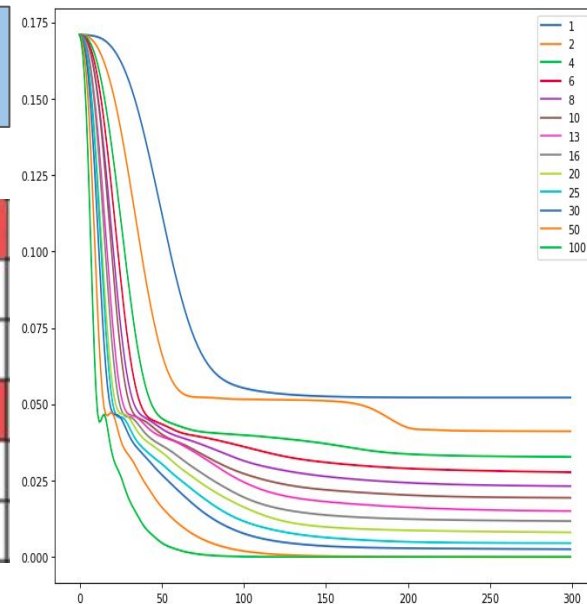
3.



sites embedding matrix with shape (e,142) \*\*



error embedding matrix \*\*  
with shape (54,e)



mse error with different embedding sizes 11

\* -  $(54,e) \cdot (e,142) = (54,142)$

\*\* -  $e = 20$

# Pseudo-Embedding Model

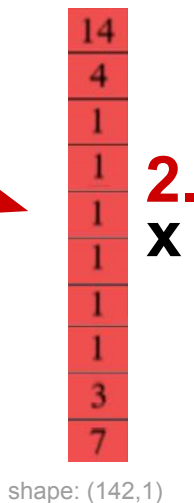
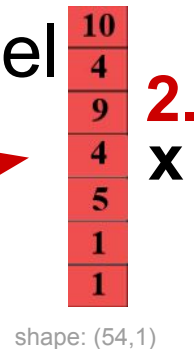
	site 1	site 2	site 3	site 4	...	site 140	site 141	site 142
error 1	1	1	1	1	1	1	1	1
error 2	4	0	0	0	0	0	0	0
error 3	8	1	0	0	0	0	0	0
...	0	2	0	0	0	0	0	2
error 52	0	0	0	0	0	0	0	5
error 53	0	0	0	0	0	0	0	1
error 54	1	0	0	0	0	0	0	0

single example shape = (54,142)

1. count number of errors
2. count errors in each site
3. site embeddings and error embeddings are concatenated
4. concat vector is fed to feed forward layers
5. everything is optimized with weighted cross entropy (wCE)

1.1.

1.2.



2. x

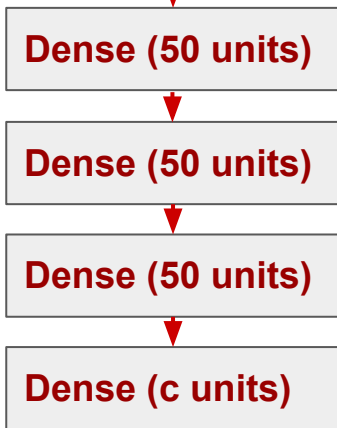
2. x



3.



4.



5.

$$J(\theta) = - \sum_i^n (W_{y_i} y_i \log \hat{y}_i)$$

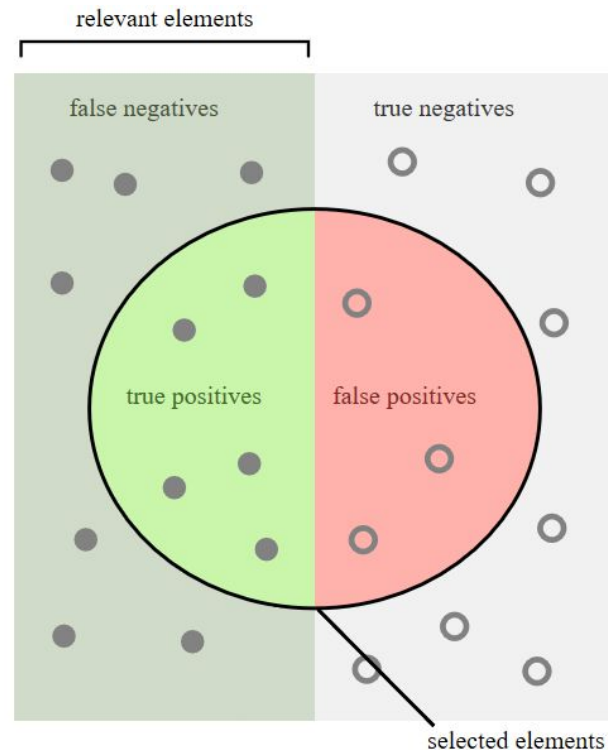
# Evaluation Metrics

- **Precision** (for each class) \*  
how many selected items are relevant?
- **Recall** (for each class) \*\*  
How many relevant items are selected?
- **Confusion MSE (main)** \*\*\*  
Mean squared error of normalized confusion matrix and identity matrix

Macro average is calculated for Recall and Precision !

$$* - \text{precision} = \frac{\text{relevant elements}}{\text{selected elements}}$$

$$** - \text{recall} = \frac{\text{relevant elements}}{\text{total elements}}$$



$$*** - \text{confusion mse} =$$

mse

0.74	0.21	0.05
0.37	0.61	0.02
0.26	0.03	0.71

,

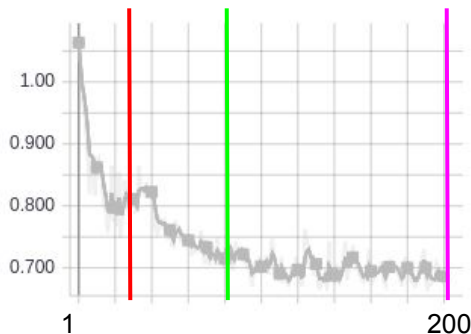
1	0	0
0	1	0
0	0	1

# Optimal Stopping when Training

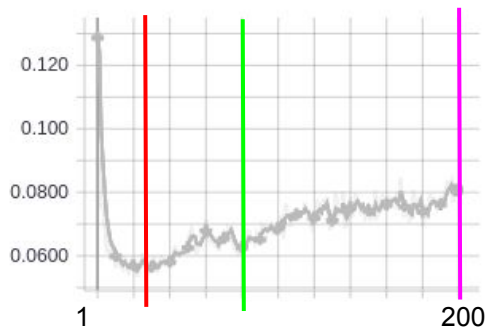
- Early stopping with single metric can lead training to stop too fast
- Looking at multiple metrics as equal can lead to stopping too late
- Optimal stopping point should be somewhere in between, that main metric doesn't deviate from best score and other metrics get better

- - too early stopping
- - optimal stopping
- - too late stopping

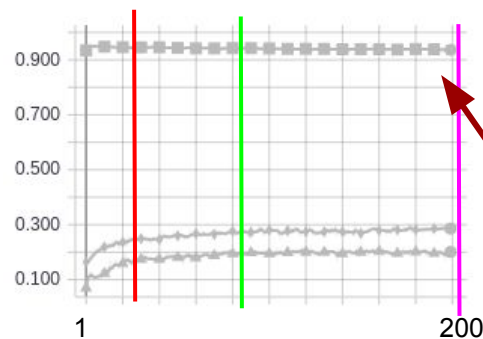
validation cross entropy



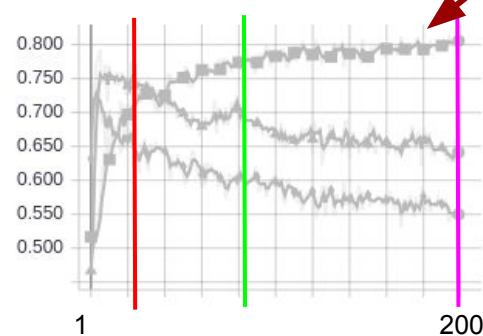
validation confusion mse (main metric)



validation precision (for each class)



validation recall (for each class)



dominant class (88%)

examples are taken when predicting action

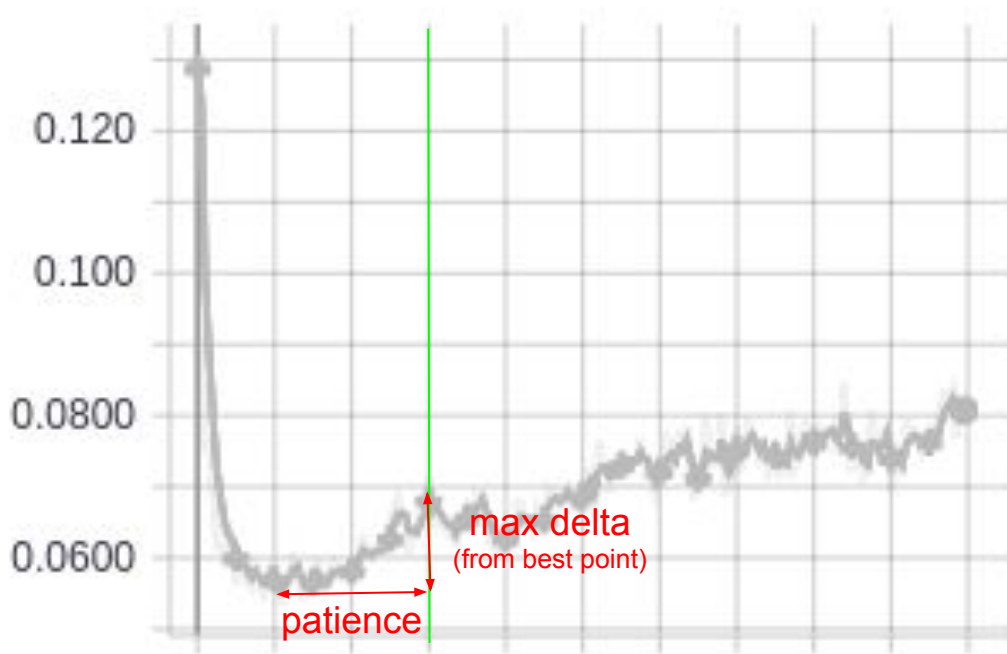
# Multiple Metric Early Stopping

1. At each epoch set best scores from all picked metrics
2. If none of the metrics gets better in  $n^*$  epoch then stop
3. If main score is worse compared to best score by  $T^{**}$  then stop

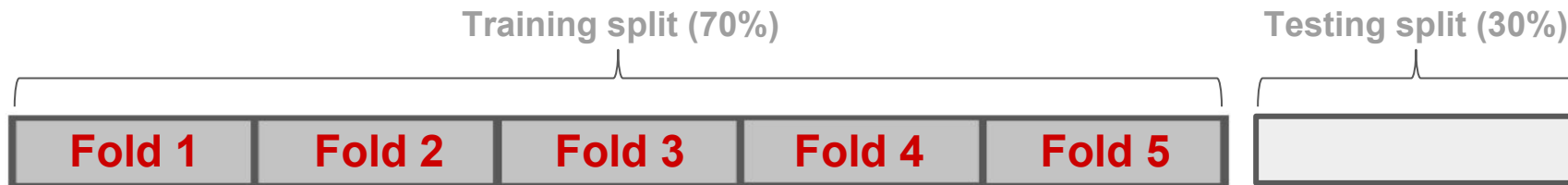
| - stopping point

\* -  $n$  - patience

\*\* -  $T$  - max delta, maximum allowed decrease of score from best score (in %)



# Picking Single Best Model



- Splitting\* dataset into testing and training datasets
- For each model\*\*:
  - Split training dataset into 5 equal size folds
  - Train on 4 Folds and test on 1 Fold (do this for each fold)
- Average each fold results into mean and std of each metric
- Model that results with best confusion mse on most outputs is considered best

\* - numpy seed = 42

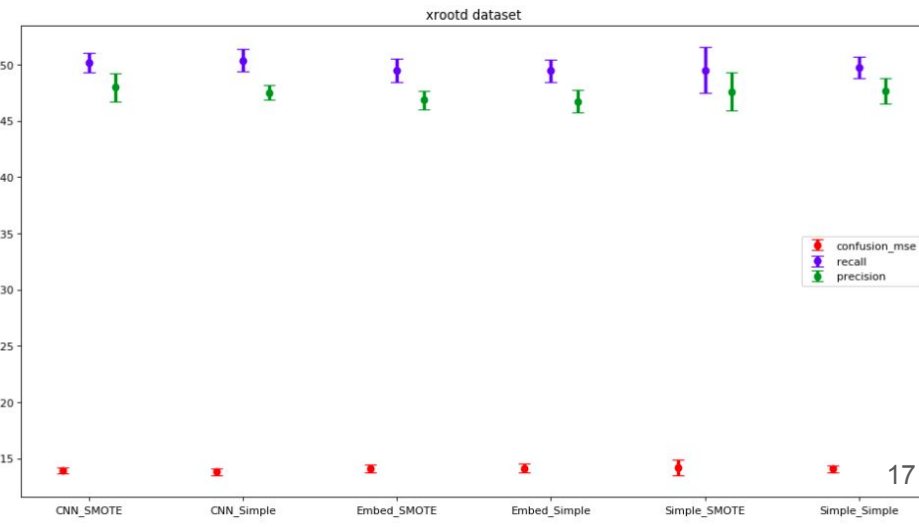
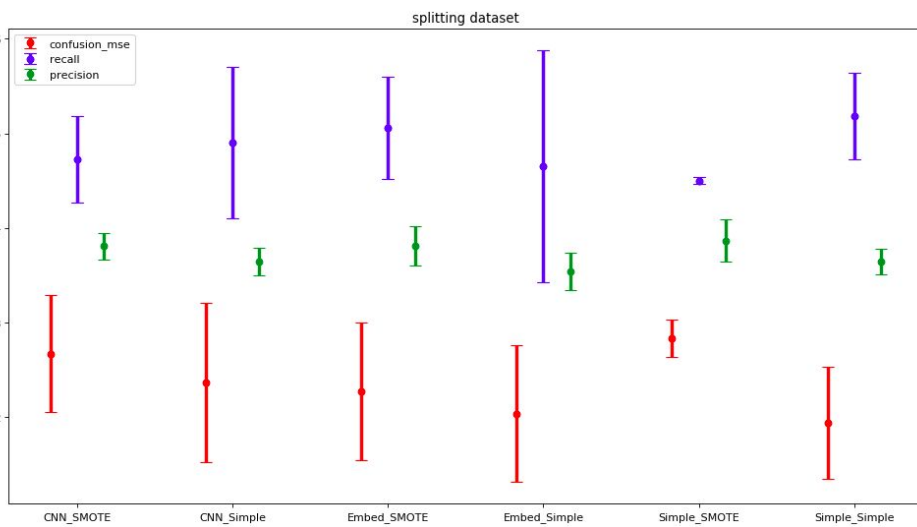
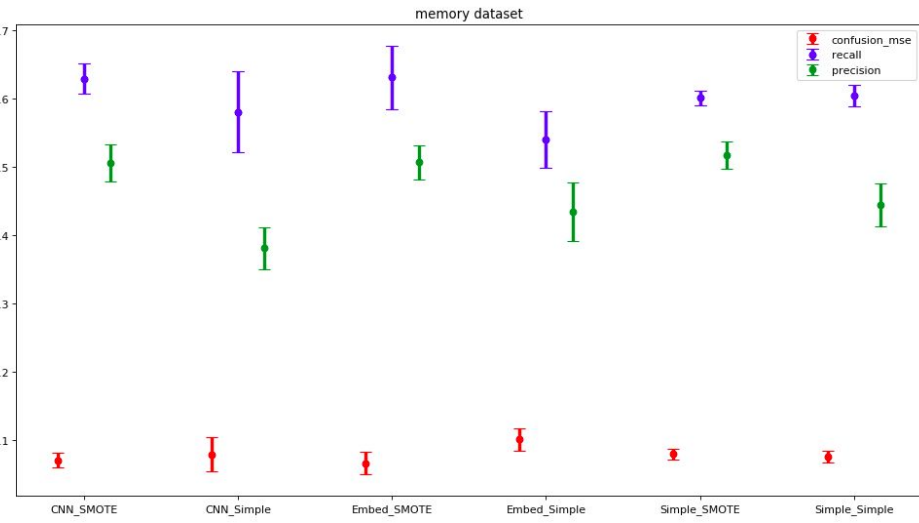
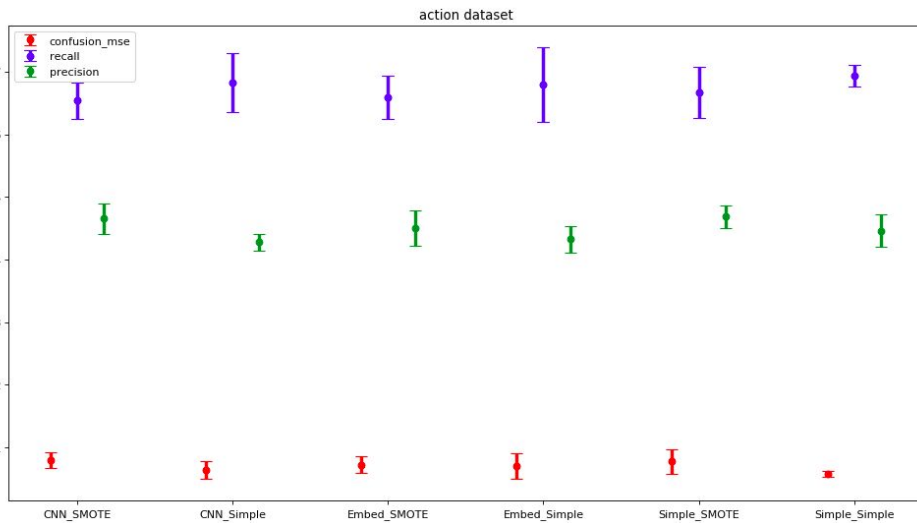
\*\* - Models that were looked at were: Total of  $3 \times 4 = 12$  Models

- Simple, CNN, Embedding with different methods:
  - trained using SMOTE resampling
  - trained with Attention
  - trained using both
  - trained with no SMOTE resampling and no Attention

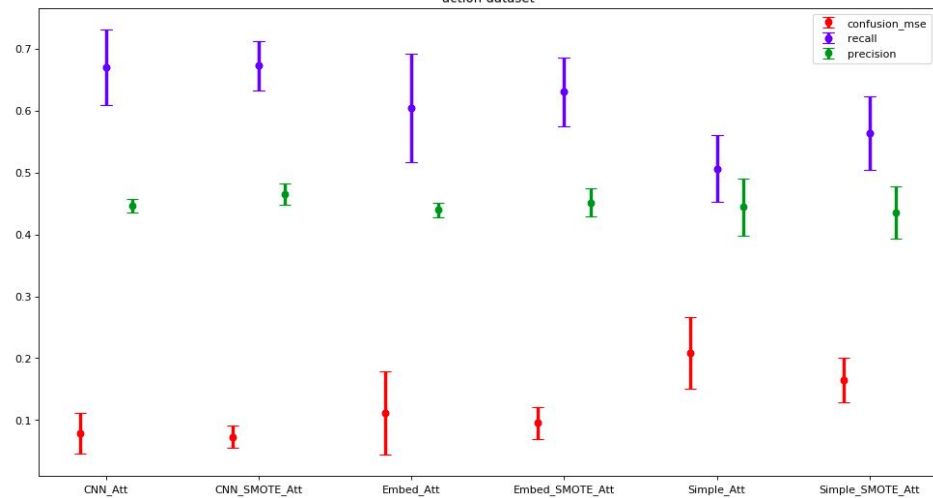
Used Early Stopping Parameters:

- patience = 7
- maximum percentage delta = 30%
- moving average length = 2

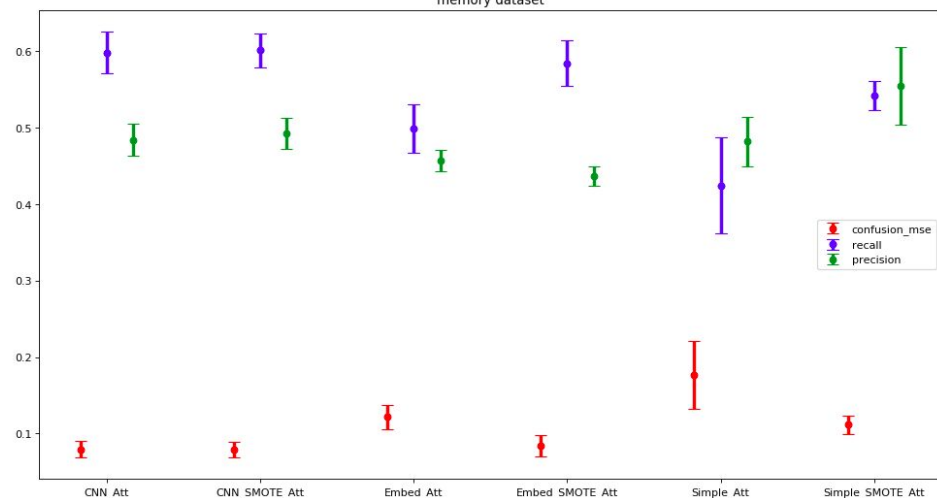




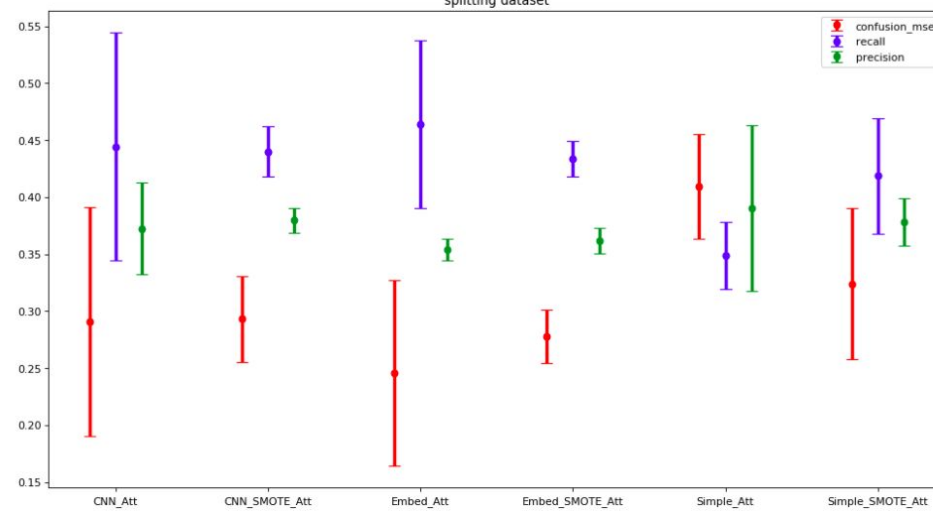
action dataset



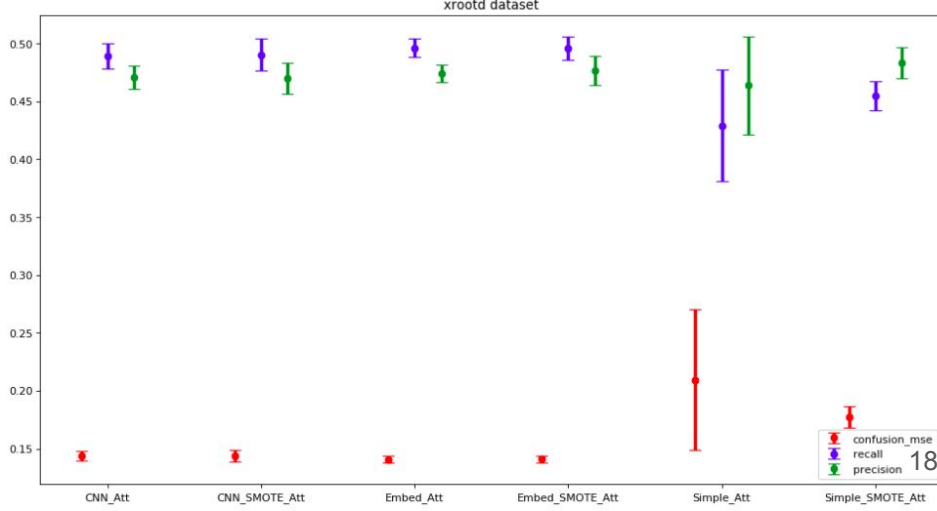
memory dataset



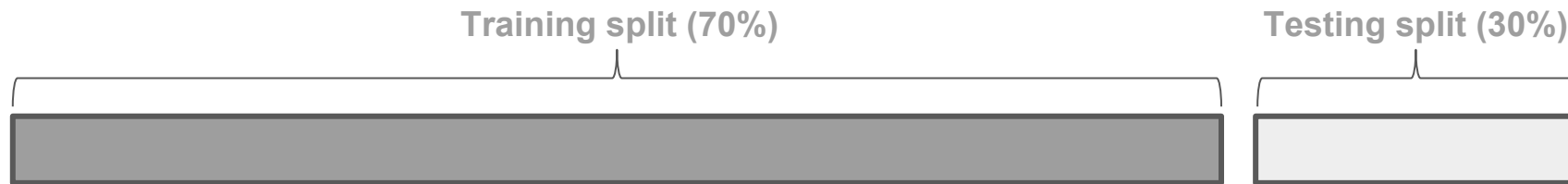
splitting dataset



xrootd dataset



# Testing Models



- Splitting\* dataset into testing and training datasets
- For 5 times:
  - Train chosen model on training dataset
  - Get all metrics results with this model on testing dataset
- Average each results into mean and std of each metric

\* - numpy seed = 42

# Best Model Results on Test Set

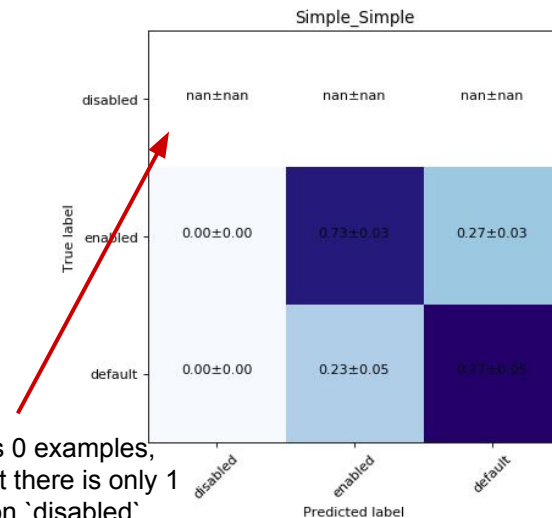
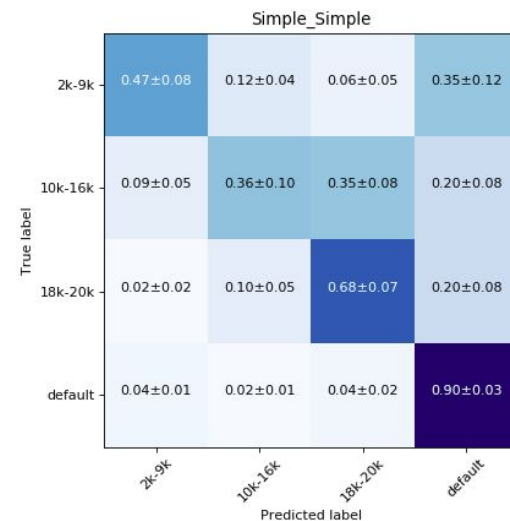
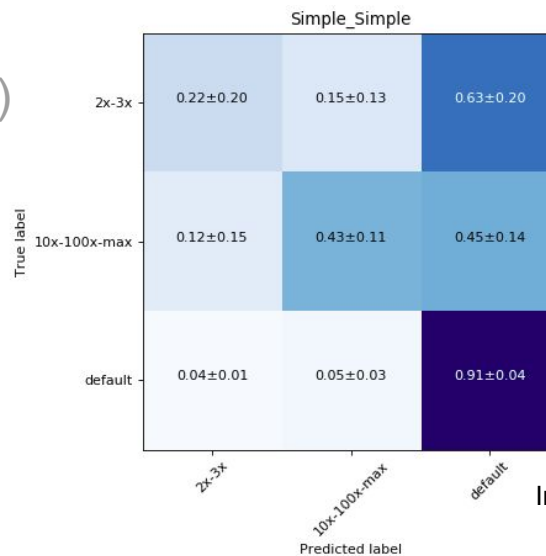
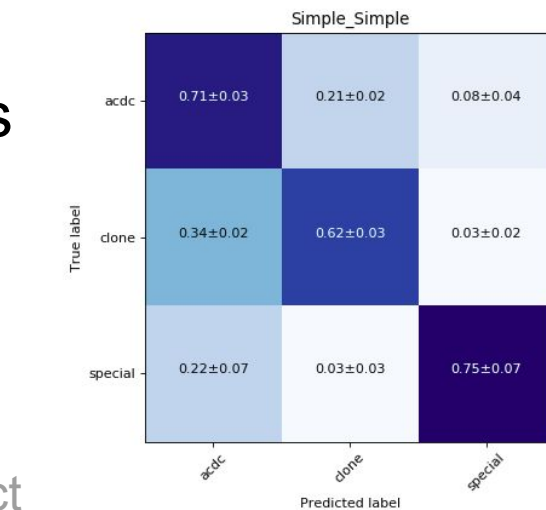
Best model - Simple\_Simple

Dataset	Model	SMOTE	Attention	conf mse	recall	precision
action	simple	-	-	.0576 ± .0068	.691 ± .017	.447 ± .005
memory	simple	-	-	.0873 ± .0160	.582 ± .028	.450 ± .022
splitting	simple	-	-	.2112 ± .0321	.482 ± .028	.366 ± .008
xrootd	simple	-	-	.1456 ± .0027	.497 ± .009	.476 ± .011

# Best Model Results on Test Set

Note: model always prioritizes on bigger classes except in `action` target: it is easier to predict special (138 examples) then clone (934 examples)

Y targets:  
action - top left  
memory - top right  
splitting - bottom left  
xrootd - bottom right



There was 0 examples,  
In all dataset there is only 1  
sample on `disabled`

# Bonus Slides

- Other Results ( Best of SMOTE, Attention, XGBoost)
- SMOTE vs weighted cross entropy (Metric evolution plots)
- Best of Attention and no Attention (confusion matrices)

# Other Results

Following slides show:

- Results using SMOTE resampling (results and confusion matrices)
- Results using Attention (results and confusion matrices)
- XGBoost model results (results)

# SMOTE Results

Best model - Simple\_SMOTE

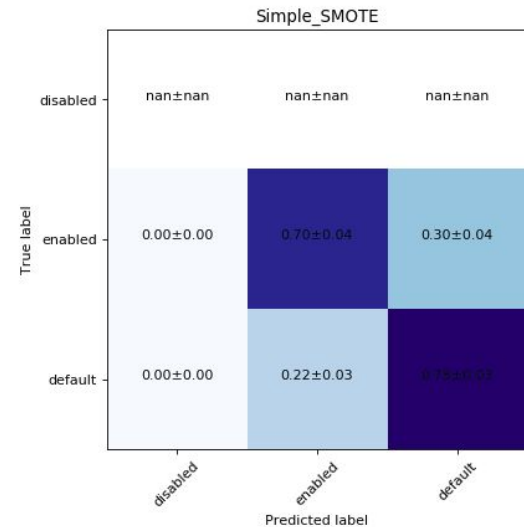
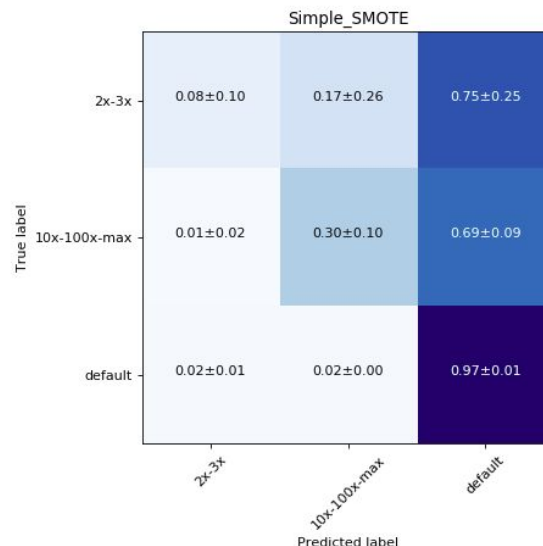
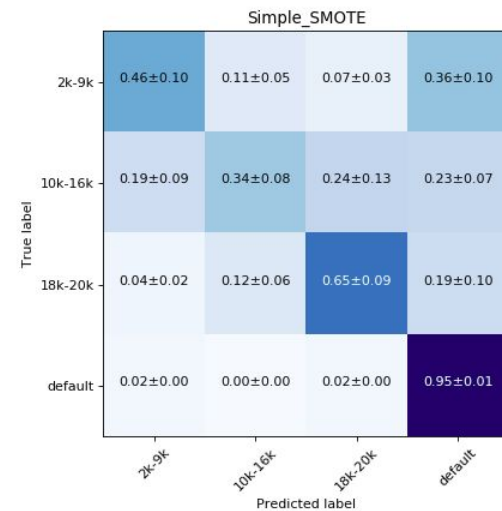
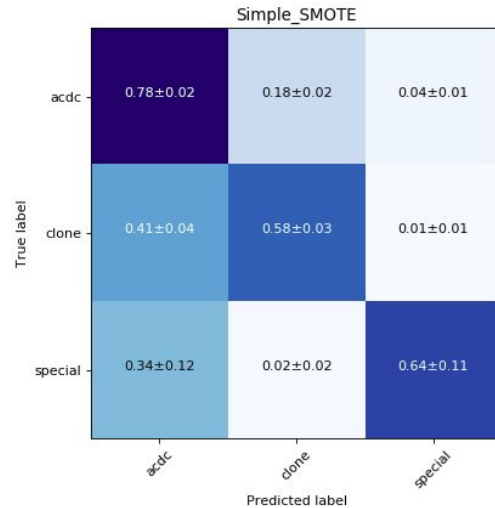
Gets higher precision then without SMOTE, but recall and conf mse are worse

Dataset	Model	SMOTE	Attention	conf mse	recall	precision
action	simple	+	-	.0780 $\pm$ .0195	.666 $\pm$ .040	.468 $\pm$ .018
memory	simple	+	-	.0759 $\pm$ .0073	.611 $\pm$ .011	.515 $\pm$ .019
splitting	simple	+	-	.2834 $\pm$ .0200	.449 $\pm$ .003	.386 $\pm$ .022
xrootd	simple	+	-	.1417 $\pm$ .0070	.495 $\pm$ .020	.475 $\pm$ .017

everything is trained with early stopping on confusion mse  
with patience = 10, maximum percentage delta = 30%, moving average length = 2



# SMOTE Results



Y targets:  
 action - top left  
 memory - top right  
 splitting - bottom left  
 xrootd - bottom right

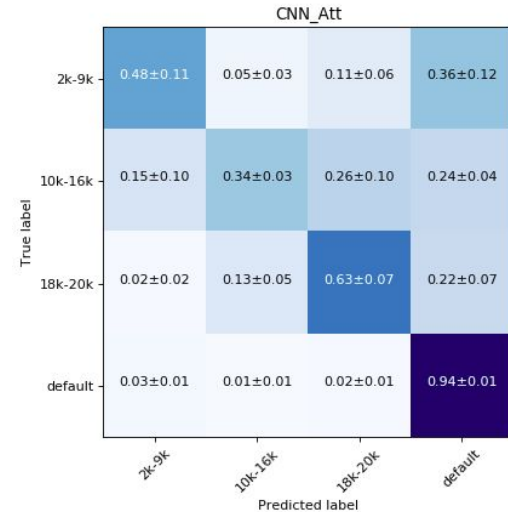
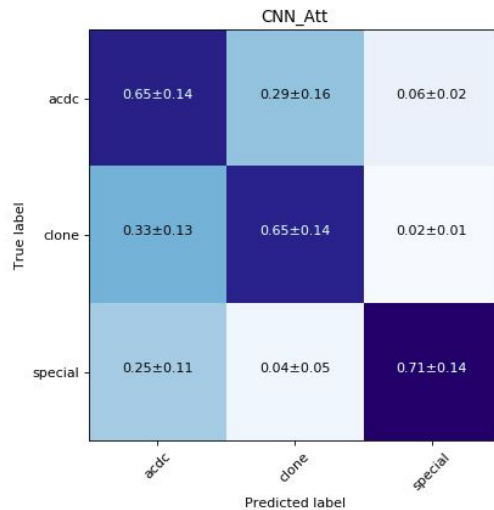
# Attention Results

Best model - CNN\_Att

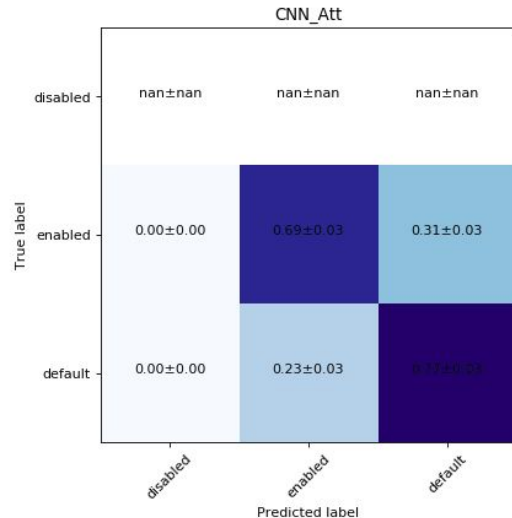
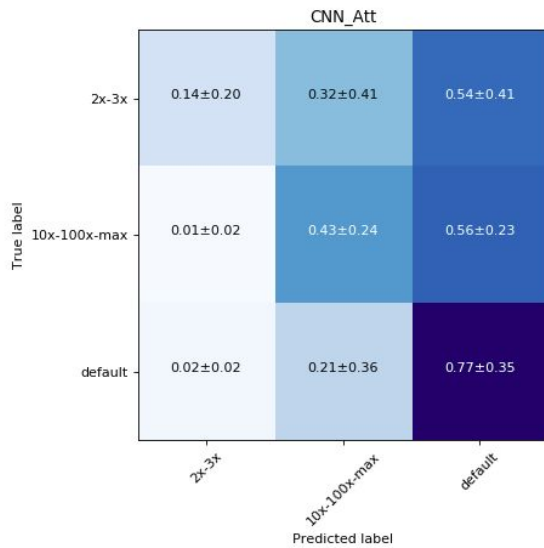
Still most metrics are worse in comparison with models without Attention

Dataset	Model	SMOTE	Attention	conf mse	recall	precision
action	cnn	-	+	.0690 ± .0159	.684 ± .016	.445 ± .022
memory	cnn	-	+	.0985 ± .074	.567 ± .027	.474 ± .021
splitting	cnn	-	+	.2907 ± .1007	.395 ± .100	.321 ± .040
xrootd	cnn	-	+	.1437 ± .0039	.489 ± .010	.470 ± .010

# Best model with Attention (confusion matrices)



Y targets:  
action - top left  
memory - top right  
splitting - bottom left  
xrootd - bottom right



# XGBoost Results

When training with max depth 4 all metrics are worse

Dataset	SMOTE	conf mse	recall	precision
action	-	.308	.443	.754
action	+	.095	.633	.456
memory	-	.129	.536	.751
memory	+	.090	.593	.594
splitting	-	.395	.362	.528
splitting	+	.384	.368	.371
xrootd	-	.291	.459	.508
xrootd	+	.252	.507	.507

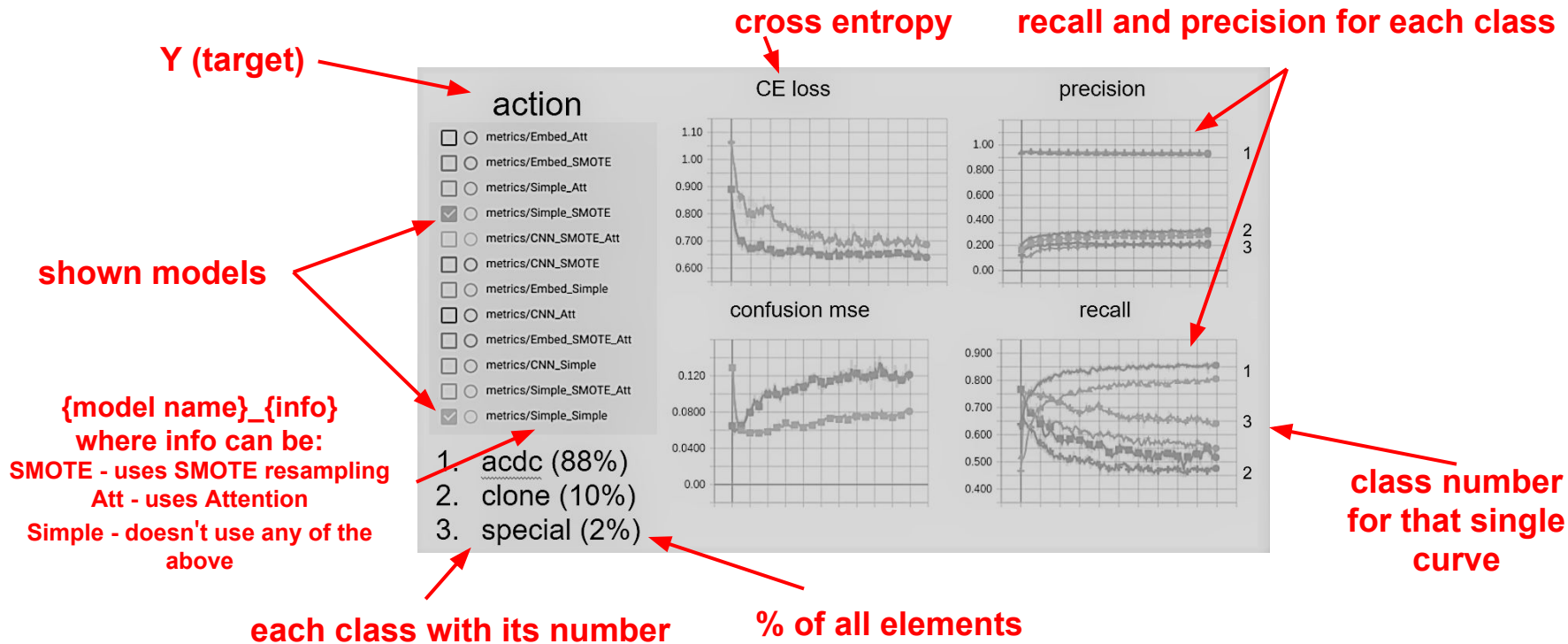
everything is trained with early stopping on confusion mse with patience = 10

# SMOTE vs weighted cross entropy

Following slides show: Metrics evolution on different Models over 200 epochs

Each dataset has 3 slides for models: (Simple, CNN, Embedding)

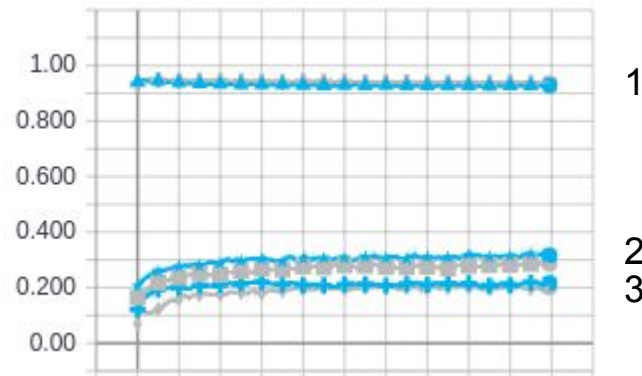
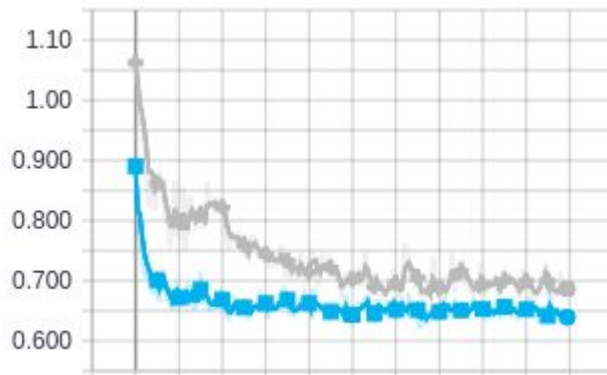
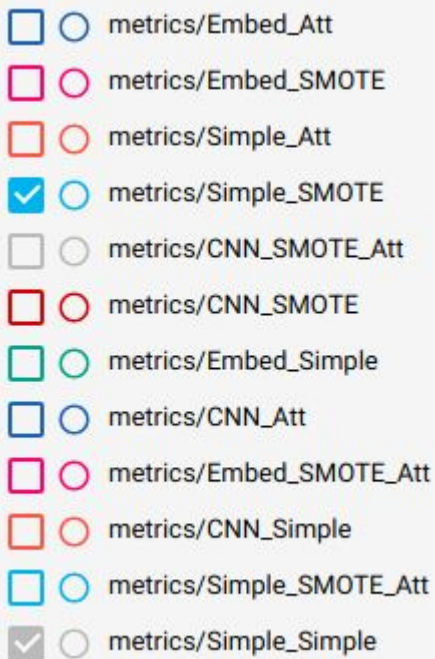
Single slide has evolution of metrics for model using and not using SMOTE resampling



# action

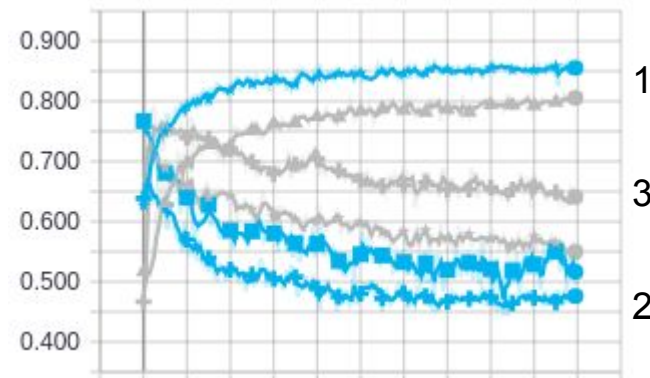
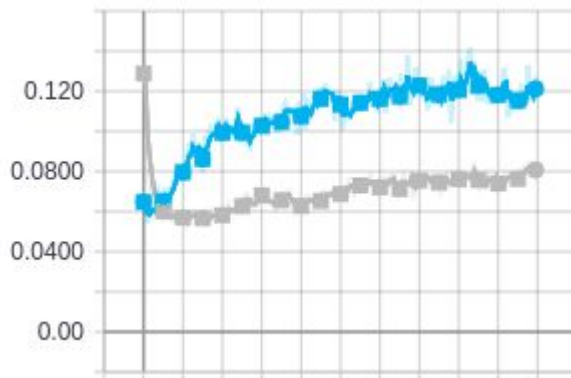
## CE loss

## precision



## confusion mse

## recall



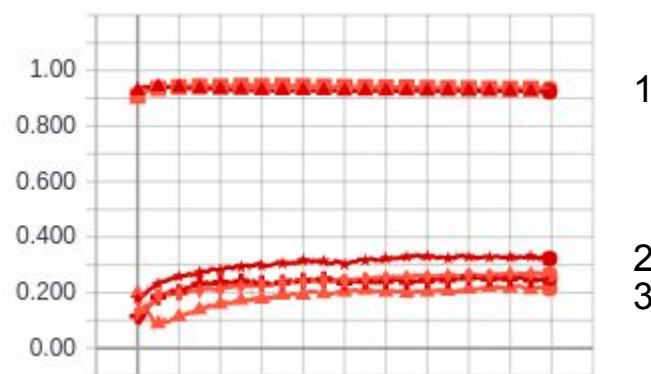
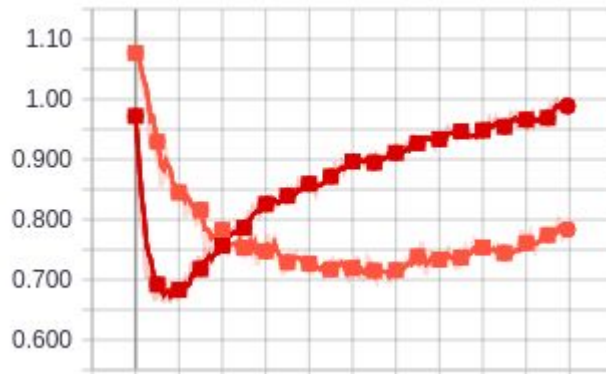
1. acdc (88%)
2. clone (10%)
3. special (2%)

# action

## CE loss

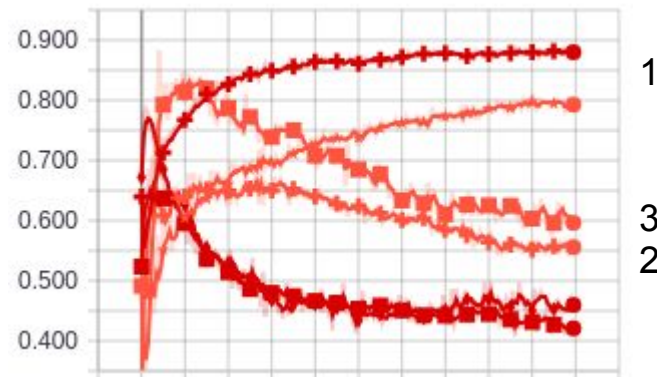
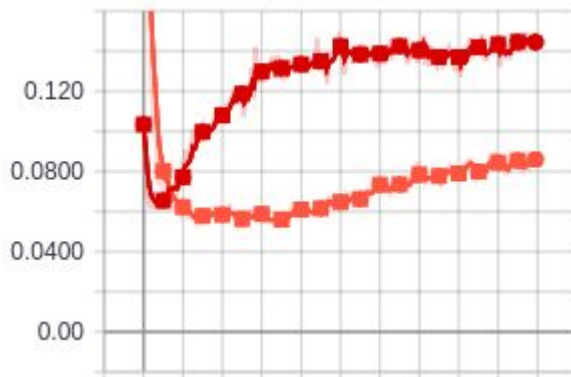
## precision

- ○ metrics/Embed\_Att
- ○ metrics/Embed\_SMOTE
- ○ metrics/Simple\_Att
- ○ metrics/Simple\_SMOTE
- ○ metrics/CNN\_SMOTE\_Att
- ☑ ○ metrics/CNN\_SMOTE
- ○ metrics/Embed\_Simple
- ○ metrics/CNN\_Att
- ○ metrics/Embed\_SMOTE\_Att
- ☑ ○ metrics/CNN\_Simple
- ○ metrics/Simple\_SMOTE\_Att
- ○ metrics/Simple\_Simple



## confusion mse

## recall



1. acdc (88%)
2. clone (10%)
3. special (2%)

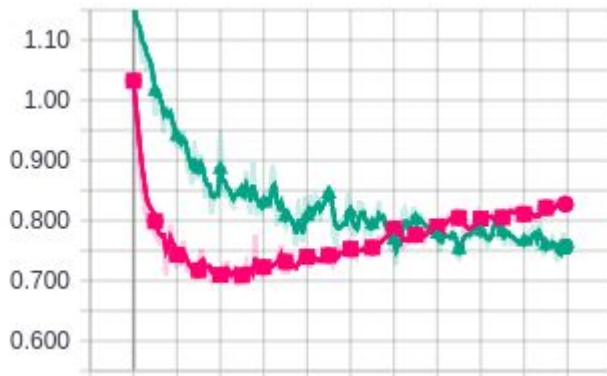


# action

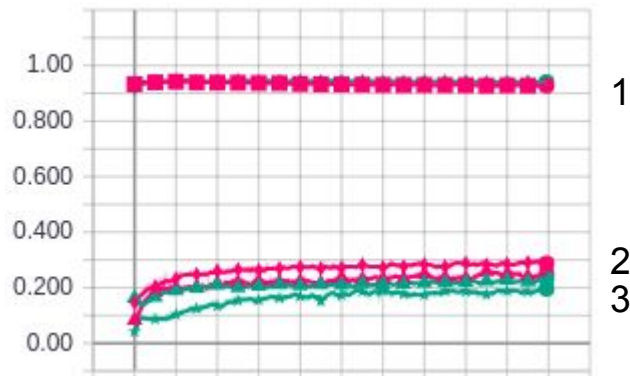
- metrics/Embed\_Att
- ✓ metrics/Embed\_SMOTE
- metrics/Simple\_Att
- metrics/Simple\_SMOTE
- metrics/CNN\_SMOTE\_Att
- metrics/CNN\_SMOTE
- ✓ metrics/Embed\_Simple
- metrics/CNN\_Att
- metrics/Embed\_SMOTE\_Att
- metrics/CNN\_Simple
- metrics/Simple\_SMOTE\_Att
- metrics/Simple\_Simple

1. acdc (88%)
2. clone (10%)
3. special (2%)

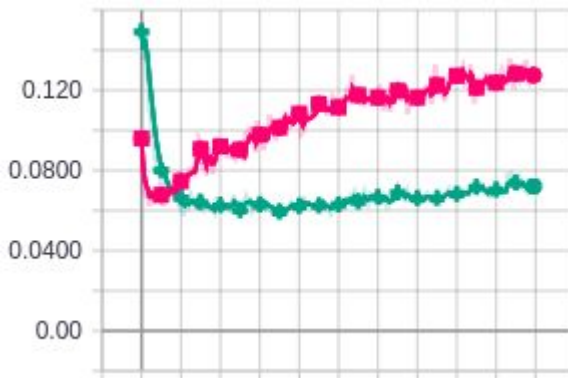
## CE loss



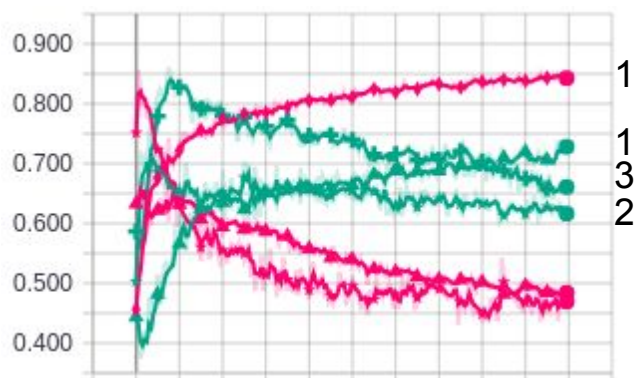
## precision



## confusion mse

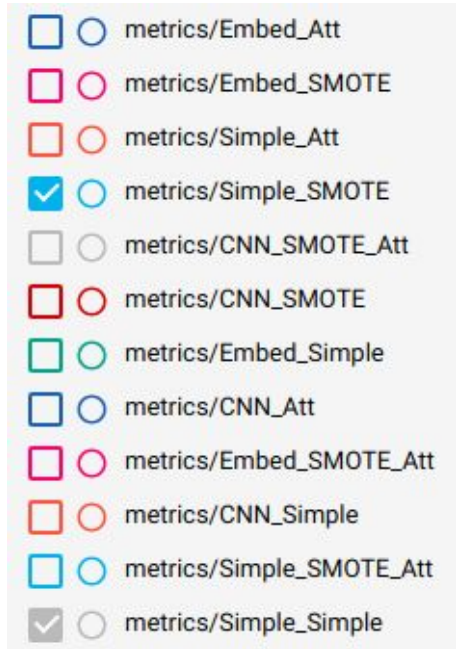


## recall

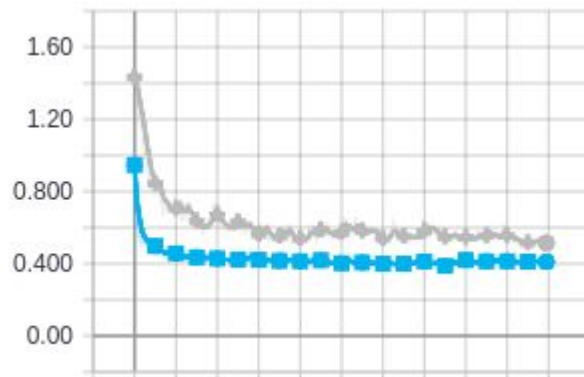




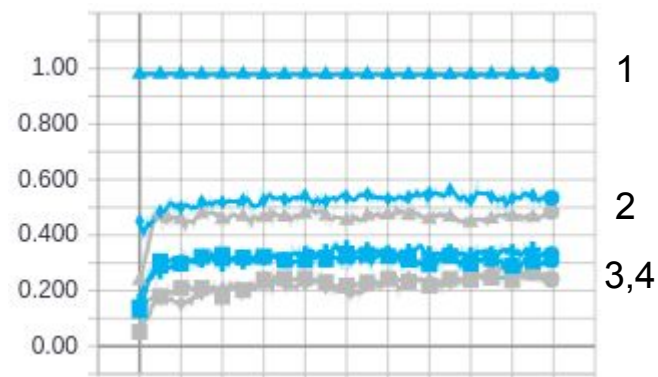
# memory



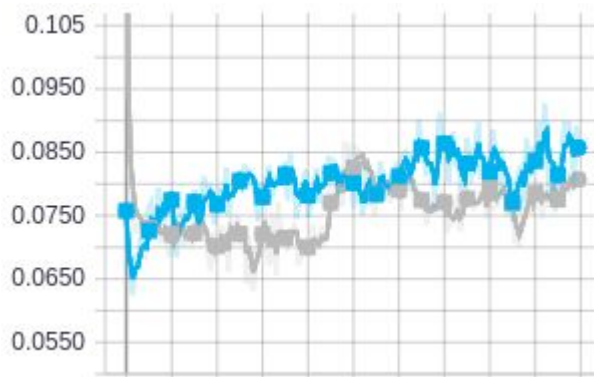
CE loss



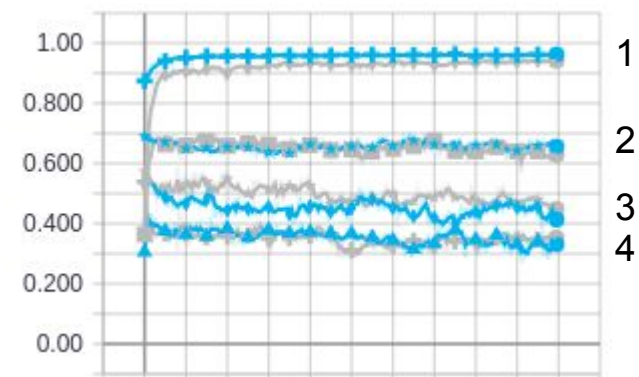
precision



confusion mse

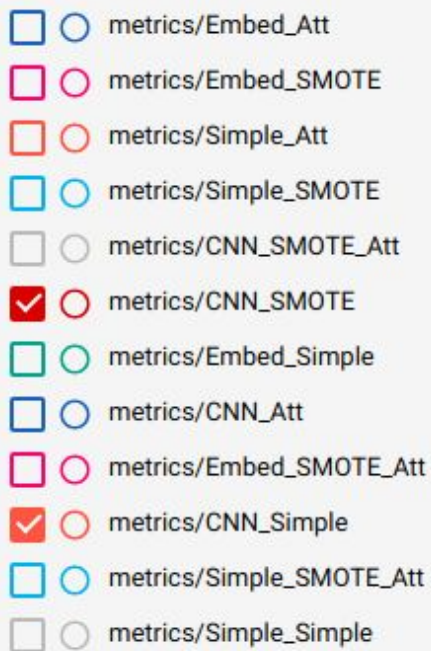


recall



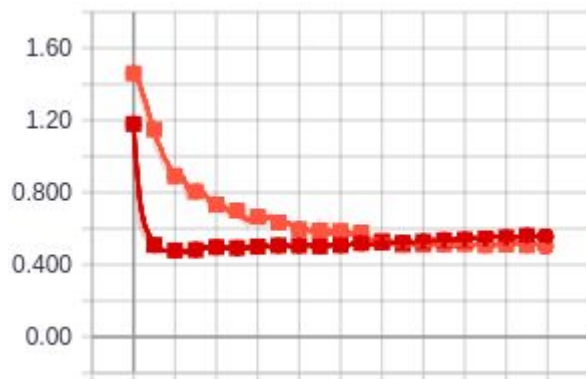
1. default (92%)
2. 18k-20k (4%)
3. 2k-9k (2%)
4. 10k-16k (1.2%)

# memory

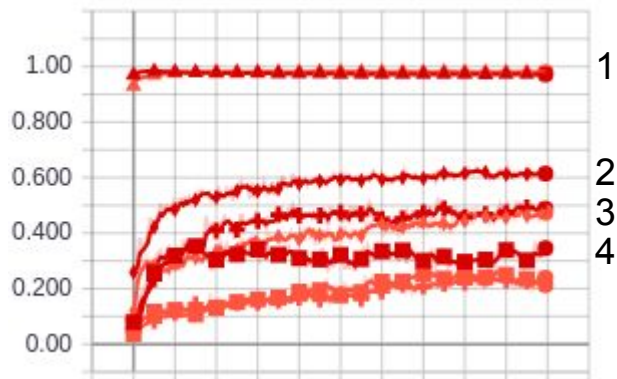


1. default (92%)
2. 18k-20k (4%)
3. 2k-9k (2%)
4. 10k-16k (1.2%)

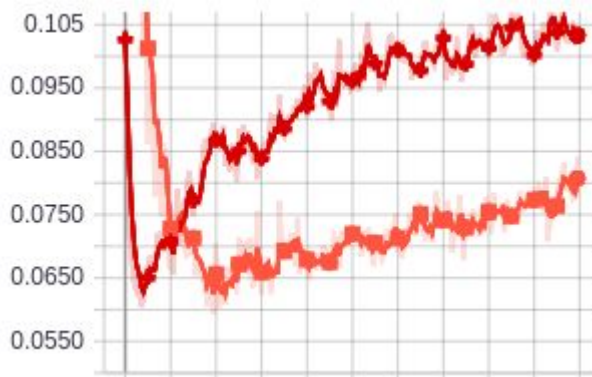
CE loss



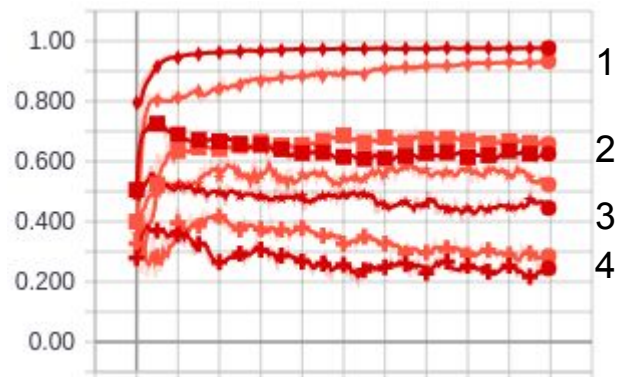
precision



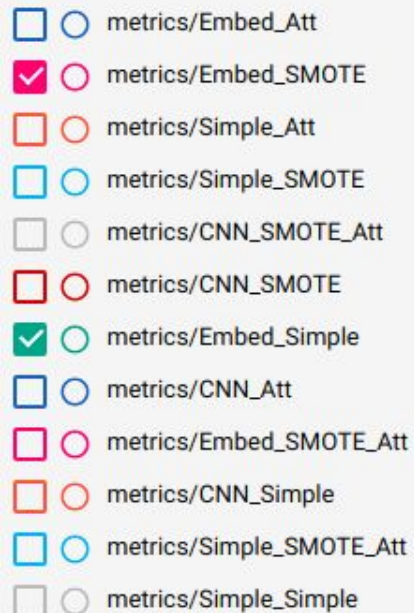
confusion mse



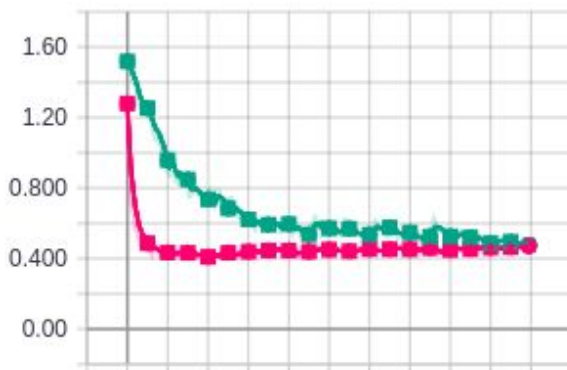
recall



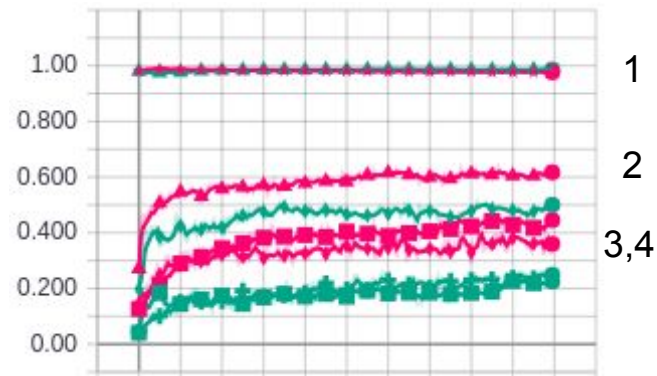
# memory



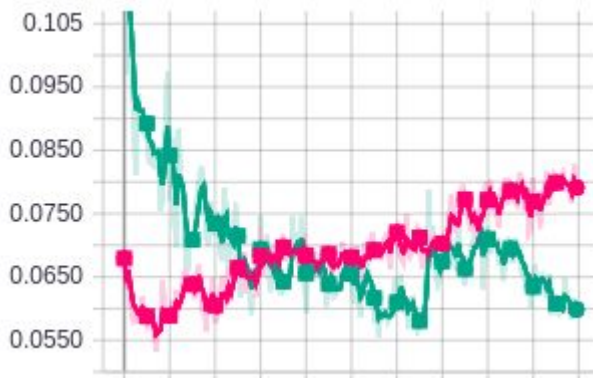
## CE loss



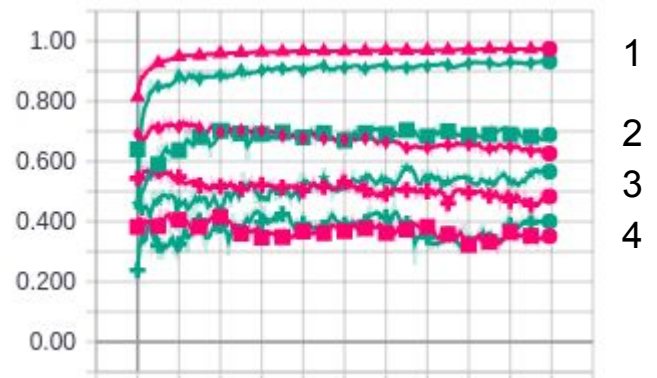
## precision



## confusion mse



## recall



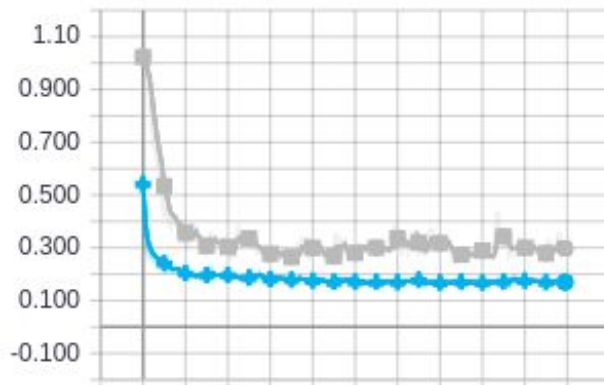
1. default (92%)
2. 18k-20k (4%)
3. 2k-9k (2%)
4. 10k-16k (1.2%)



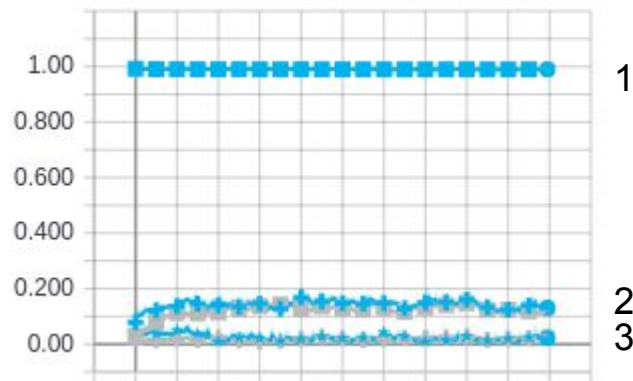
# splitting

- metrics/Embed\_Att
- metrics/Embed\_SMOTE
- metrics/Simple\_Att
- ✓ metrics/Simple\_SMOTE
- metrics/CNN\_SMOTE\_Att
- metrics/CNN\_SMOTE
- metrics/Embed\_Simple
- metrics/CNN\_Att
- metrics/Embed\_SMOTE\_Att
- metrics/CNN\_Simple
- metrics/Simple\_SMOTE\_Att
- ✓ metrics/Simple\_Simple

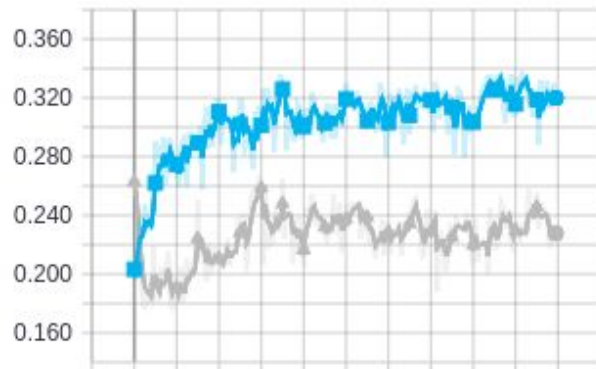
CE loss



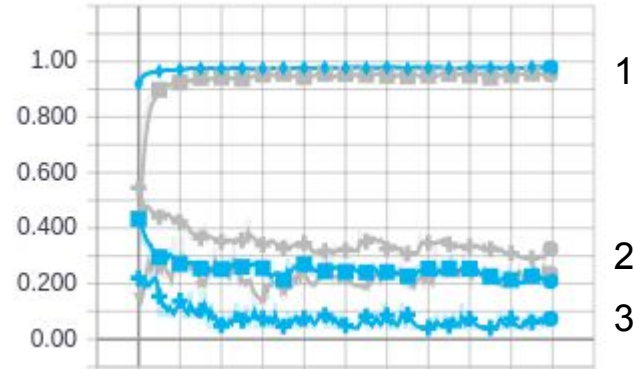
precision



confusion mse



recall



1. default (98%)
2. 10x-100x (1%)
3. 2x-3x (0.4%)

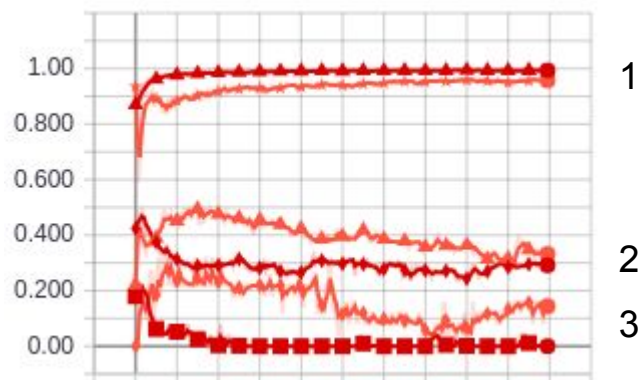
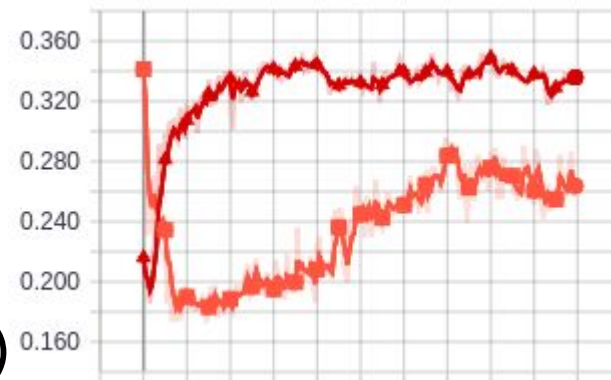
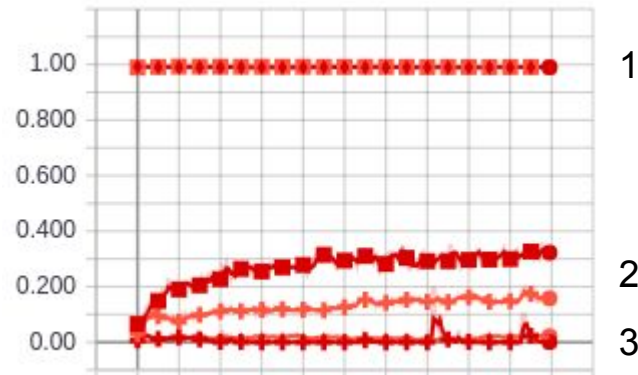
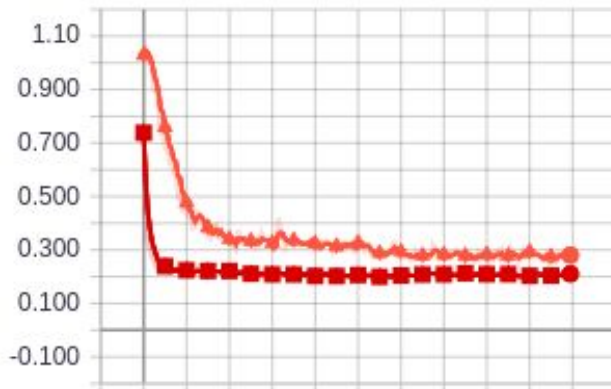
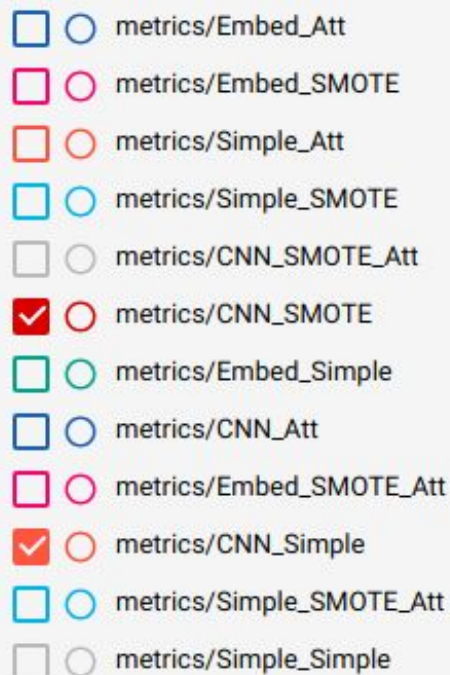
# splitting

## CE loss

## precision

## confusion mse

## recall

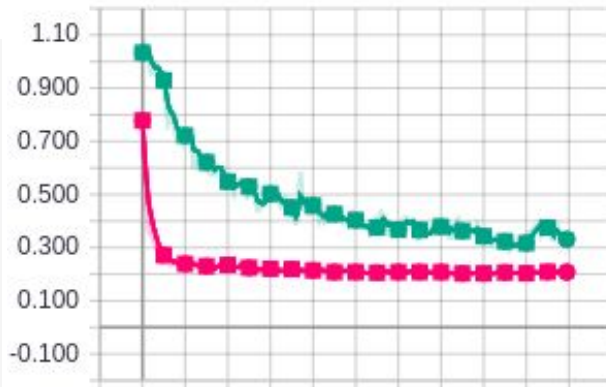


1. default (98%)
2. 10x-100x (1%)
3. 2x-3x (0.4%)

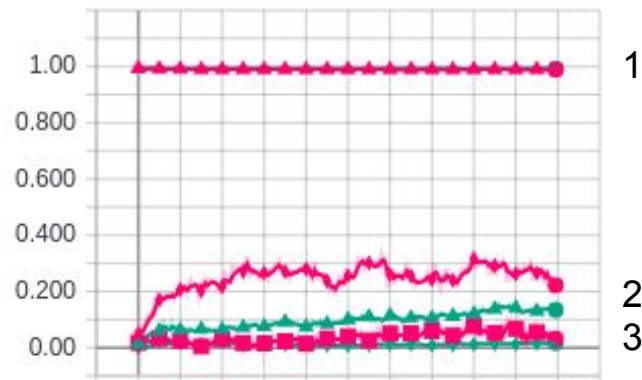
# splitting

- ☐ ☐ metrics/Embed\_Att
- ☒ ☐ metrics/Embed\_SMOTE
- ☐ ☐ metrics/Simple\_Att
- ☐ ☐ metrics/Simple\_SMOTE
- ☐ ☐ metrics/CNN\_SMOTE\_Att
- ☐ ☐ metrics/CNN\_SMOTE
- ☒ ☐ metrics/Embed\_Simple
- ☐ ☐ metrics/CNN\_Att
- ☐ ☐ metrics/Embed\_SMOTE\_Att
- ☐ ☐ metrics/CNN\_Simple
- ☐ ☐ metrics/Simple\_SMOTE\_Att
- ☐ ☐ metrics/Simple\_Simple

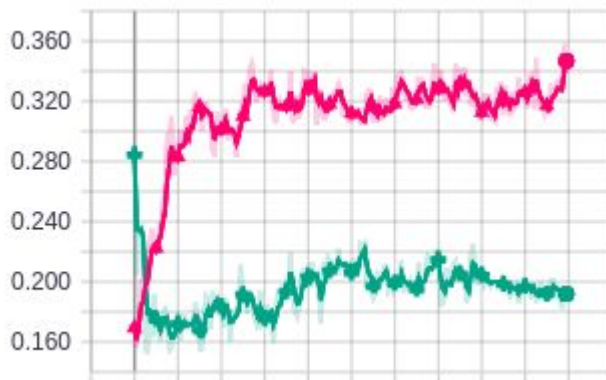
CE loss



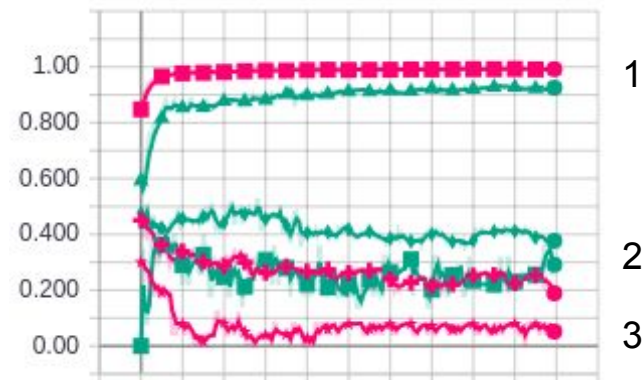
precision



confusion mse



recall



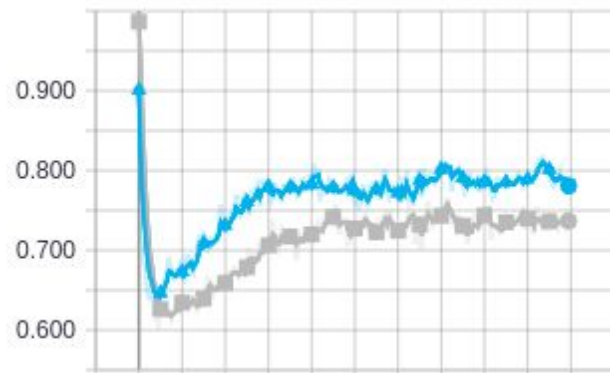
1. default (98%)
2. 10x-100x (1%)
3. 2x-3x (0.4%)

# xrootd

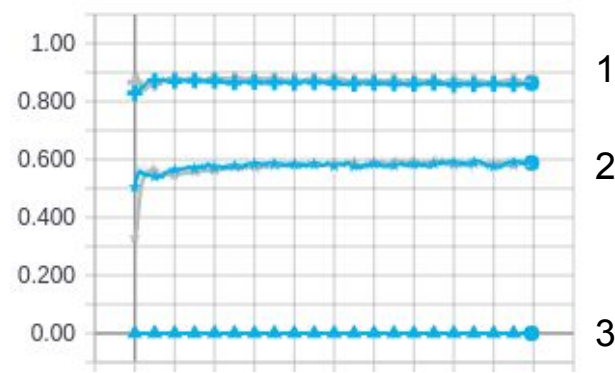
- metrics/Embed\_Att
- metrics/Embed\_SMOTE
- metrics/Simple\_Att
- ☒ metrics/Simple\_SMOTE
- metrics/CNN\_SMOTE\_Att
- metrics/CNN\_SMOTE
- metrics/Embed\_Simple
- metrics/CNN\_Att
- metrics/Embed\_SMOTE\_Att
- metrics/CNN\_Simple
- metrics/Simple\_SMOTE\_Att
- ☒ metrics/Simple\_Simple

1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

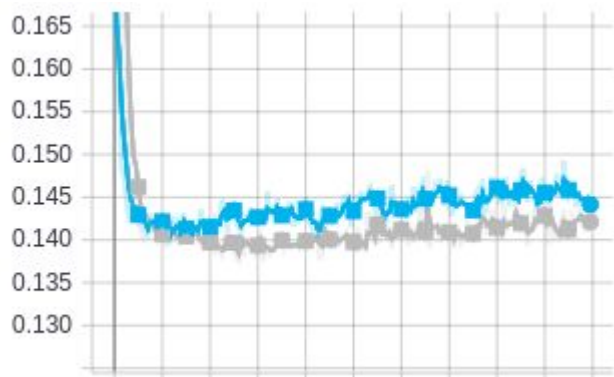
CE loss



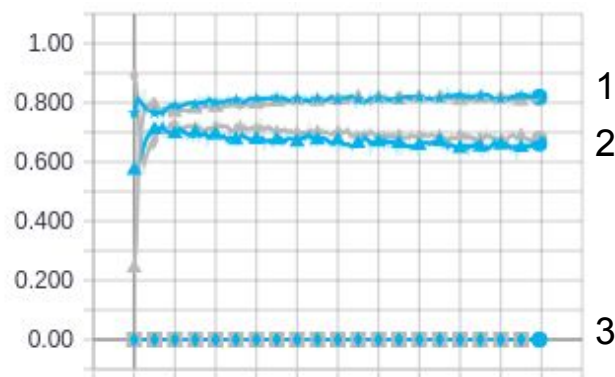
precision



confusion mse



recall



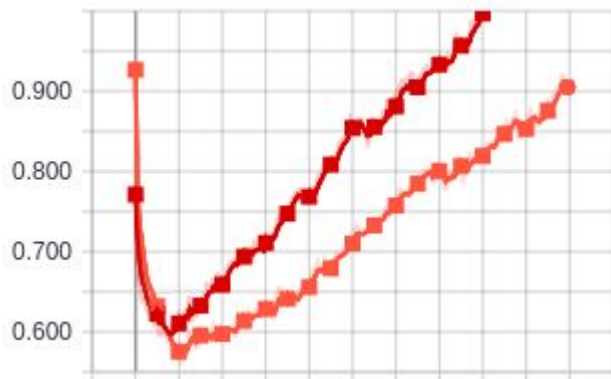


# xrootd

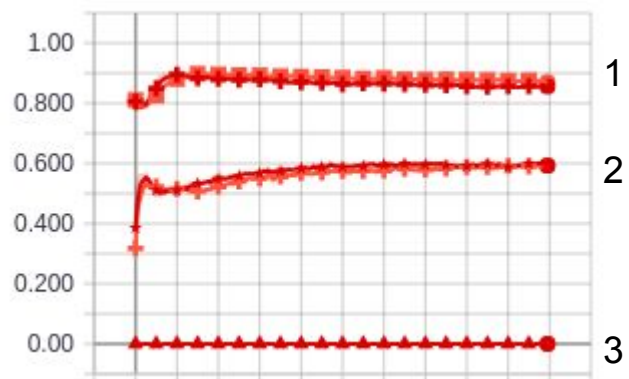
- metrics/Embed\_Att
- metrics/Embed\_SMOTE
- metrics/Simple\_Att
- metrics/Simple\_SMOTE
- metrics/CNN\_SMOTE\_Att
- metrics/CNN\_SMOTE
- metrics/Embed\_Simple
- metrics/CNN\_Att
- metrics/Embed\_SMOTE\_Att
- metrics/CNN\_Simple
- metrics/Simple\_SMOTE\_Att
- metrics/Simple\_Simple

1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

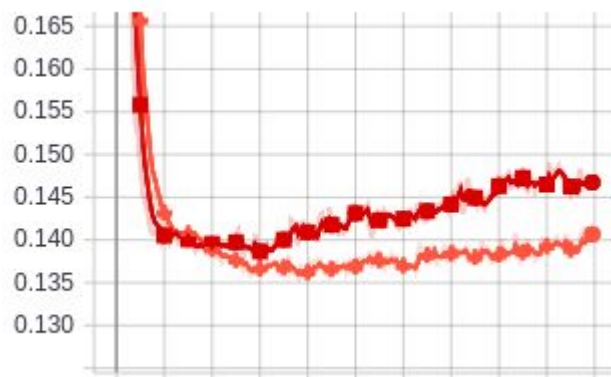
CE loss



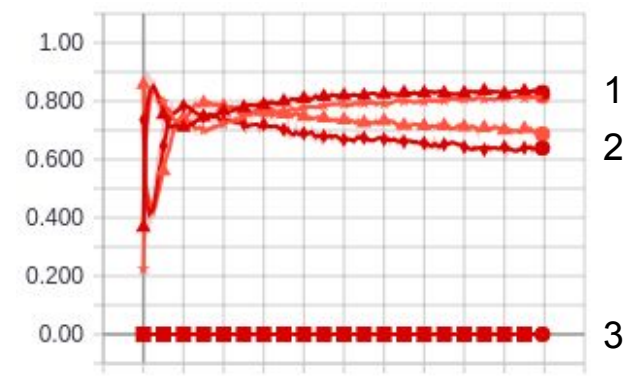
precision



confusion mse



recall



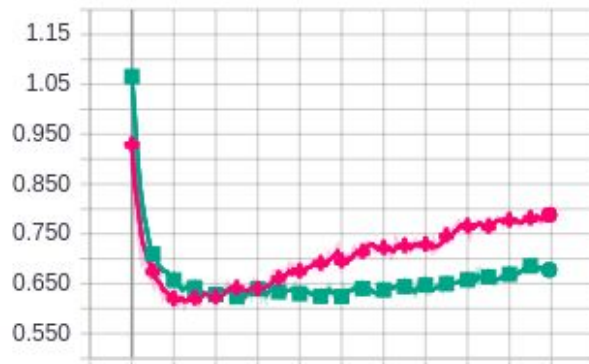


# xrootd

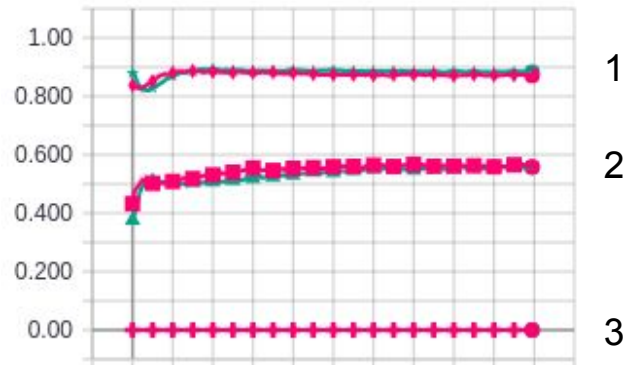
- metrics/Embed\_Att
- metrics/Embed\_SMOTE
- metrics/Simple\_Att
- metrics/Simple\_SMOTE
- metrics/CNN\_SMOTE\_Att
- metrics/CNN\_SMOTE
- metrics/Embed\_Simple
- metrics/CNN\_Att
- metrics/Embed\_SMOTE\_Att
- metrics/CNN\_Simple
- metrics/Simple\_SMOTE\_Att
- metrics/Simple\_Simple

1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

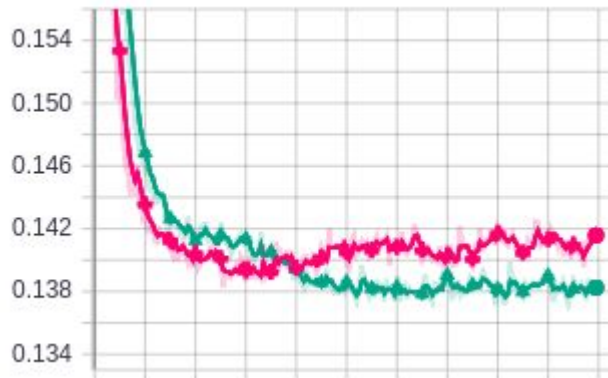
CE loss



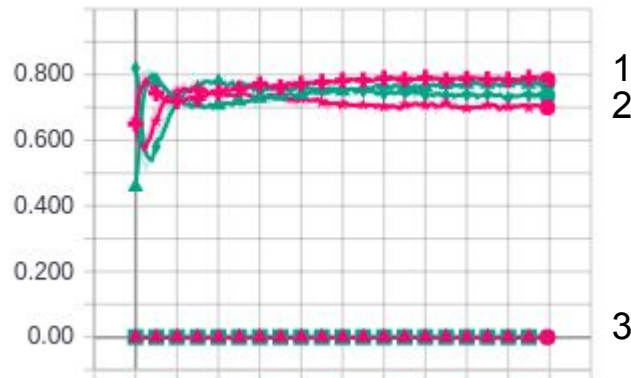
precision



confusion mse



recall



# Attention vs no Attention (confusion matrices)

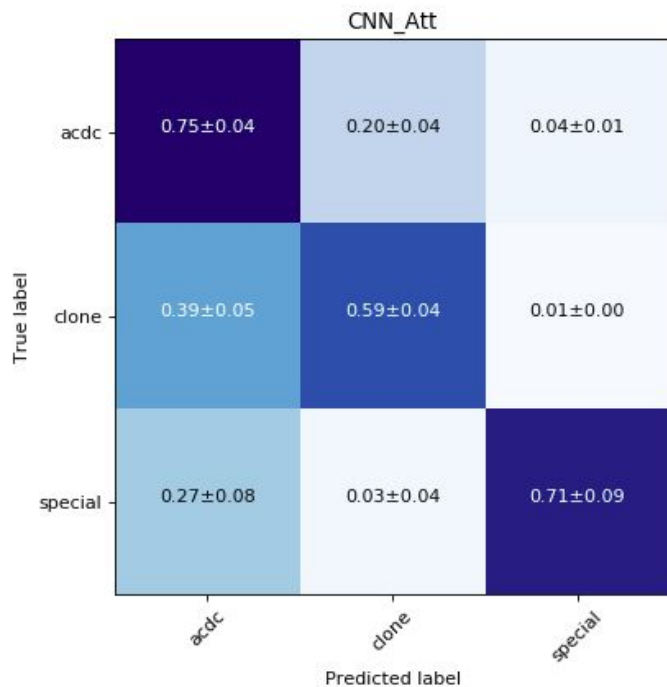
Following slides show:

- Best confusion matrix on each Y target on validation dataset (averaged across 5 folds) using Attention (left matrix) and comparison on right with model without using Attention

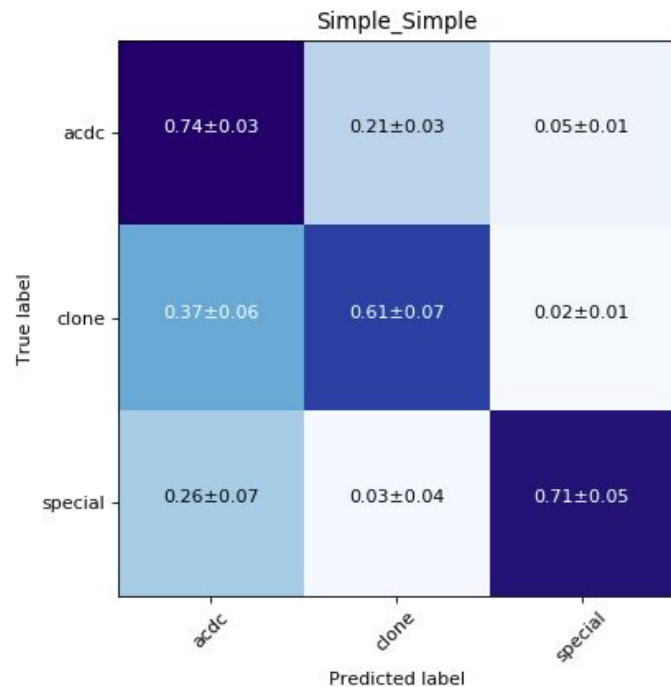
# action

1. acdc (88%)
2. clone (10%)
3. special (2%)

best: CNN model  
w/ weighted CE  
w/out SMOTE



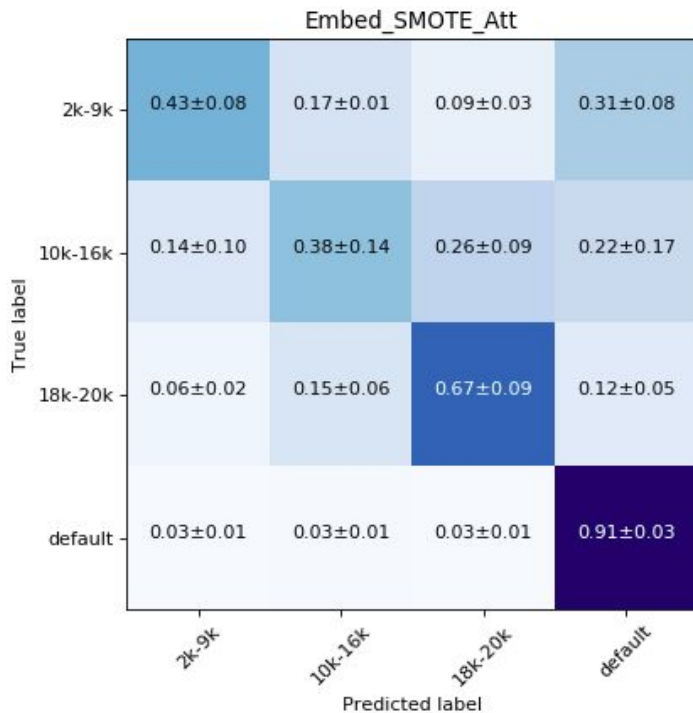
comparison w/ best model  
w/out Attention:  
Simple w/ weighted CE  
w/out SMOTE



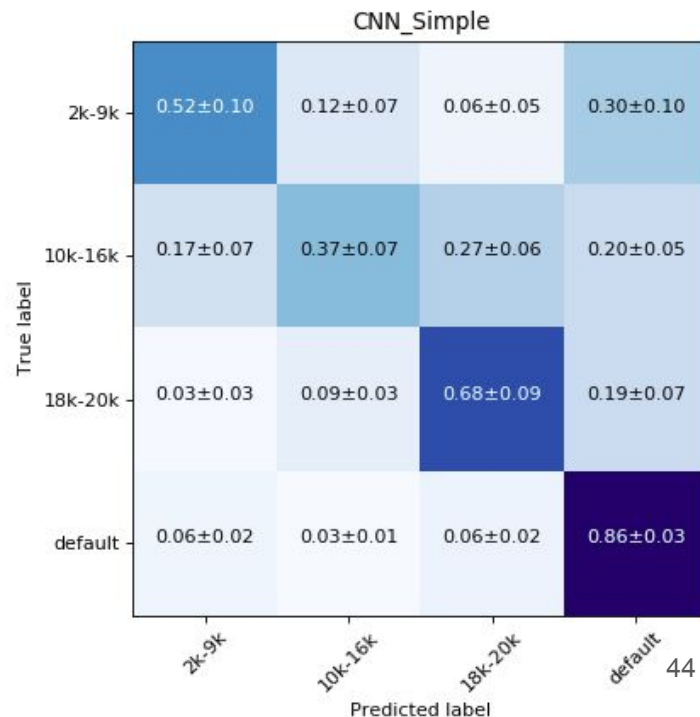
# memory

1. default (92%)
2. 18k-20k (4%)
3. 2k-9k (2%)
4. 10k-16k (1.2%)

best: Embed model  
w/ SMOTE



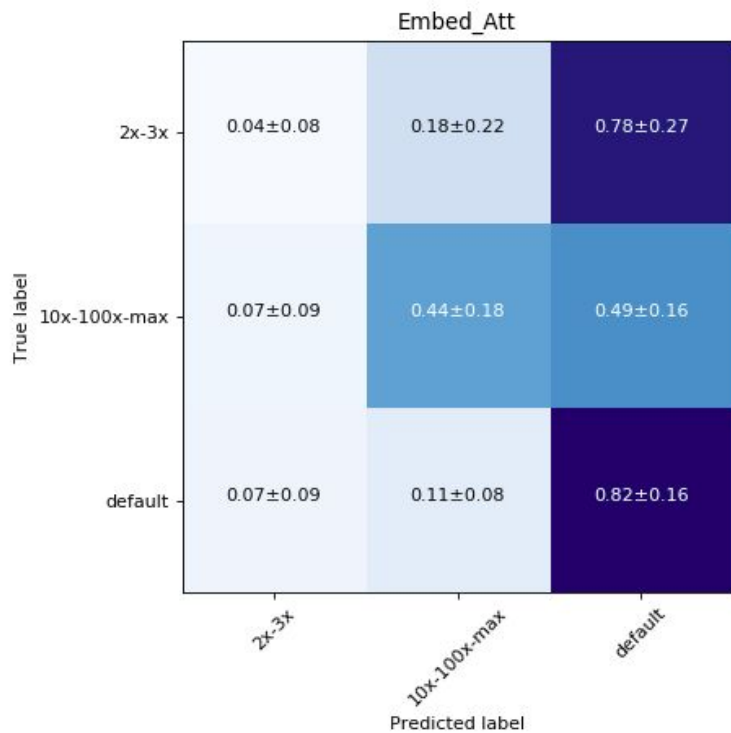
comparison w/ best model  
w/out Attention:  
CNN w/ weighted CE  
w/out SMOTE



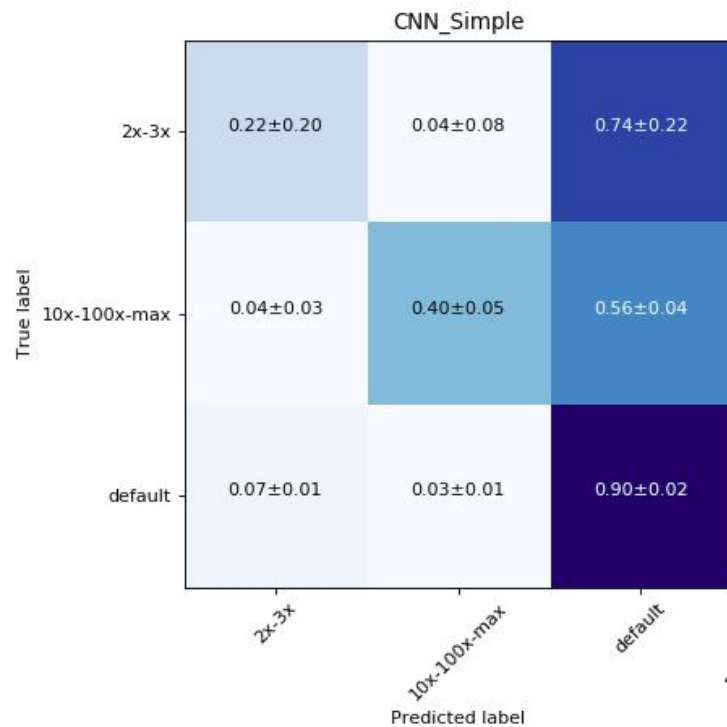
# splitting

1. default (98%)
2. 10x-100x (1%)
3. 2x-3x (0.4%)

best: Embed model  
w/ weighted CE  
w/out SMOTE



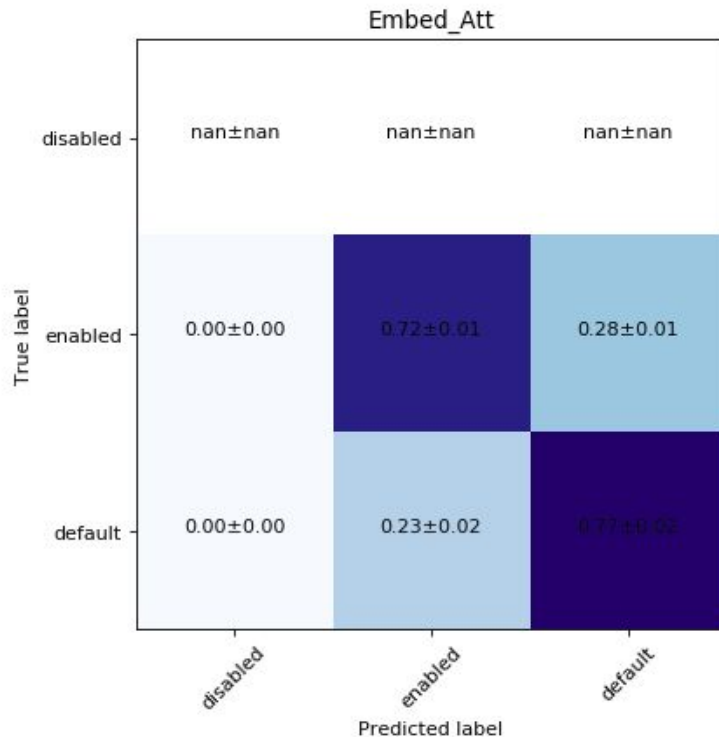
comparison w/ best model  
w/out Attention:  
CNN w/ weighted CE  
w/out SMOTE



# xrootd

1. default (72%)
2. enabled (28%)
3. disabled (0.01%)

best: Embed model  
w/ weighted CE  
w/out SMOTE



comparison w/ best model  
w/out Attention:  
CNN w/ weighted CE  
w/out SMOTE

