Project Machine Learning
— Milestone 2 —

Ricardo Fleck, Hassan Bassiouny, Augustin Krause

January 8, 2024

1 Methodology

In this section, we will describe our chosen training approach, including the form of model-selection we opted for. The feature extraction method for both the classical MIL and `MNIST-bags` datasets remains unchanged from the previous milestone.

1.1 Data Augmentation

Originally we had planned to use data augmentation to train more robust classifiers. However, for reasons described below, this turned out to be either impossible or unnecessary. We therefore decided against augmenting any of our chosen datasets.

The classical MIL datasets contain embedding vectors with pre-computed features. This renders any form of data augmentation challenging. The difficulties arise from the ambiguity regarding what defines positive/negative instances. Because of this, transformations to individual instances within a bag are problematic, since they would result in an unlabeled bag. Shuffling within a bag is also ineffective, as the label remains invariant to permutations. Consequently, the only viable option is shuffling between different bags, which makes the labels of the shuffled bags uncertain as well.
Hence, we are left with the option to augment the `MNIST-bags` dataset. This is a trivial task, since it is clear which instances constitute a positive bag. For example, by rotating or color perturbing images, we could create new instances, from which we could form new bags. However, we did not find it necessary to pursue augmentation here, since the `MNIST-bags` dataset already makes an abundance of training data available. For more information on the datasets see our previous report.

1.2 Hyper-parameters

Model hyper-parameters are external configurations of a machine learning model that are set before the training process begins. They are distinct from the parameters of the model, which are learned during training. In our case, we needed to optimize the following hyper-parameters:

Number of Epochs The number of times the entire training dataset is passed through the model during training. Too few epochs may result in underfitting, while too many may lead to overfitting.

Learning Rate A hyper-parameter that determines the size of the steps taken during optimization. It influences the convergence speed and the risk of overshooting the optimal solution.

Weight Decay It controls the strength of the penalty applied to the magnitude of the weights in a neural network. The penalty is added to the loss function during training to prevent the model from becoming too complex and overfitting the training data.

Optimizer The algorithm used to update the model parameters during training. We will consider Adam (Kingma and Ba, 2017) and Stochastic Gradient Descent (SGD) (Sutskever et al., 2013).

Momentum This hyper-parameter only plays a role if we choose SGD as the optimizer. It essentially helps the optimizer to continue moving in the same direction as the previous iterations, smoothing out the oscillations that may occur during training.

Beta-values The hyper-parameters β_1 and β_2 in Adam control the exponential decay rates for the running averages of gradients and squared gradients, respectively. They essentially influence the adaptive learning rate used for updating the model parameters.

1.3 Model Selection Methodology

To fit our hyper-parameters to our models and learning tasks we have performed a Grid-Search on a set of parameters listed in Table 1 and evaluated the resulting models with 10-fold Cross-Validation (CV) on the 0-1 loss.

Grid Search This involves specifying a range of values for each hyper-parameter. We then compute the CV for each possible combination and choose the one with the lowest loss. Furthermore, we have chosen a logarithmic scale to cover a wide range of possible hyper-parameter values.
We decided against including the hyper-parameters β_1 and β_2 in our Grid Search, as the default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are commonly used (Kingma and Ba (2017)). The running time of the Grid Search scales exponentially with each additionally tested value. Including them would have extended the already hefty computation time (see Section 4.1) without a significant expected benefit.
Our chosen grid of hyper-parameter values we tested for is given in Table 1.

Number of Epochs	Learning Rate	Weight Decay	Momentum	Optimizer
1	0.1	0	0	Adam
10	0.01	0.05	0.9	SGD
100	0.001	0.005	0.09	
	0.0001	0.0005		

Table 1: Hyper-Parameter Grid

Cross-Validation The n-fold CV divides the available dataset into n disjoint sets (folds). In each iteration the model is trained on $n-1$ of those folds, to be then tested on the remaining fold. This process is repeated n times, so that each fold serves as a test set exactly once. The average performance over the CV results for one combination of hyper-parameters provides a general estimate of the model's performance using that set of hyper-parameters.
To test the model during CV we used the 0-1-loss:

0-1-Loss Let $\hat{y}_1, ..., \hat{y}_n$ be a sample of predicted labels with true labels $y_1, ..., y_n$, then the 0-1 loss is defined as:

$$\mathcal{L}(\hat{y}_1, ..., \hat{y}_n | y_1, ..., y_n) = \frac{1}{n} \sum_{j=1}^{n} d(\hat{y}_j, y_j), \quad \text{where } d(\hat{y}, y) = \begin{cases} 1 & \hat{y} \neq y \\ 0 & \hat{y} = y \end{cases}$$

Thus, it measures the average number of falsely predicted labels.

1.4 Model Selection Results

The model-selection process described above yielded the hyper-parameter values shown in Table 2. We can observe that the Adam optimizer performed on average better on all datasets except the `TIGER` dataset. This is different to the work of Ilse et al. (2018), who use SGD to train their model on the classical MIL datasets. However, in their paper, they did not include the optimizer as a to-be-optimized hyper-parameter in the Grid Search.
Note that in certain cases, the momentum parameter for the Adam optimizer is not set to zero, even though its value does not influence the optimization of Adam. This occurred because we tested all possible combinations of hyper-parameters, without ignoring irrelevant ones. Due to the random initialization of weights, combinations corresponding to the same optimization procedure can yield different final models.

Also noteworthy is that for the ELEPHANT dataset most MIL approaches exhibit better performance with 10 epochs. This observation suggests that 100 epochs might lead to overfitting here. Conversely, the TIGER and MUSK2 datasets exhibit better performance with 100 epochs, suggesting that these datasets are harder to learn.

(a) ELEPHANT dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	100	0.001	0.0005	0.09	Adam
	max	10	0.001	0.0	0.0	Adam
embedding	mean	10	0.0001	0.005	0.9	Adam
	max	10	0.0001	0.005	0.0	Adam
	attention	10	0.0001	0.0005	0.0	Adam
	gated attention	10	0.0001	0.0	0.0	Adam

(b) FOX dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	10	0.0001	0.0	0.0	Adam
	max	100	0.01	0.0	0.0	SGD
embedding	mean	10	0.0001	0.0005	0.9	Adam
	max	10	0.001	0.0	0.0	Adam
	attention	100	0.001	0.0005	0.0	Adam
	gated attention	10	0.0001	0.0	0.0	Adam

(c) TIGER dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	100	0.0001	0.0005	0.9	SGD
	max	100	0.0001	0.0	0.9	SGD
embedding	mean	100	0.001	0.005	0.0	Adam
	max	100	0.01	0.0	0.9	SGD
	attention	100	0.01	0.0	0.0	SGD
	gated attention	10	0.01	0.0005	0.09	SGD

(d) MUSK1 dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	10	0.001	0.0	0.09	Adam
	max	100	0.01	0.005	0.09	SGD
embedding	mean	10	0.001	0.0	0.0	Adam
	max	100	0.001	0.05	0.0	Adam
	attention	100	0.0001	0.005	0.9	Adam
	gated attention	10	0.1	0.005	0.0	SGD

(e) MUSK2 dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	10	0.001	0.005	0.0	Adam
	max	10	0.001	0.0005	0.0	Adam
embedding	mean	100	0.001	0.0005	0.9	Adam
	max	100	0.001	0.0	0.0	SGD
	attention	100	0.001	0.0005	0.09	Adam
	gated attention	100	0.0001	0.0	0.9	Adam

(f) MNIST-bags dataset hyper-parameters

MIL	pooling type	epochs	learning rate	weight decay	momentum	optimizer
instance	mean	10	0.1	0.005	0.0	SGD
	max	100	0.0001	0.0	0.0	Adam
embedding	mean	100	0.0001	0.0	0.9	Adam
	max	100	0.01	0.0	0.0	SGD
	attention	100	0.001	0.0	0.9	SGD
	gated attention	100	0.01	0.0	0.0	SGD

Table 2: Model hyper-parameter selection

2 Model Evaluation Methodology

In this section we will showcase the evaluation metrics we used to asses model generalization. We evaluated the selected models (refer to Section 1.4) on a test set that was generated from the original datasets and contains only data that is hidden during training. However, originally we planned to use Nested Cross-Validation for model evaluation, but ultimately decided against it. The details on this are also explained in this section.

2.1 Evaluation Metrics

In multiple instance learning (MIL) problems, several error measures can be used depending on the specific characteristics of the problem. Some MIL problems might require more weight to be given to reducing false negatives, while in others boosting true positives might be of higher importance. We therefore evaluated multiple different metrics on our models:

Accuracy Test accuracy measures the overall correctness of predictions made by a model on a test dataset. It calculates the ratio of correctly predicted instances to the total number of instances in the dataset. Accuracy alone may not suffice to fully evaluate the model's performance, as it does not tend to a need to minimize either false positives or false negatives.

Precision Precision evaluates the relevancy of the model's positive predictions. It is the ratio of correctly predicted positive instances to the total instances predicted as positive. This metric plays an important role in systems where false positives are more critical than false negatives. This can be important for recommender-systems. Here, a false positive represents a recommendation the user does not like, which could potentially deter the user from the platform.

Recall Recall measures the completeness of the model's positive predictions. It is the ratio of correctly predicted positive instances to the total actual positive instances. Recall holds significance in systems where the impact of false negatives outweighs that of false positives. In such scenarios, the system prioritizes identifying all positive instances accurately rather than incorrectly labeling negative examples. This is, e.g., of relevance for the diagnosis of patients, since not diagnosing an illness is a grave error.

F-Score The F-Score is the harmonic mean of precision and recall. It balances precision and recall in a single metric.

AUC (Area Under the Curve) The AUC scores the system's ability to distinguish positive and negative examples. It evaluates the classification performance by varying the margin for classification. This involves tracking the true positive rate against the false positive rate at different margin values. AUC is a metric confined between 0 and 1.

Trade-offs between Recall and Precision As discussed above, the importance of precision and recall can vary based on the specific application constraints. Trade-offs between the two are achievable through various means.
For example, some of the models from the ones we evaluate may excel more in either precision or recall. One could pick the model that suits the application best.
Alternatively, one could tweak the classification margin. To achieve higher precision, widening the margin for positive classification can enhance accuracy in identifying true positives. Conversely, narrowing the margin can ensure the identification of all positives, which would boost recall.

2.2 Nested Cross-Validation

We originally wanted to use Nested Cross-Validation (NCV) to estimate the generalization error. We fully implemented a working version of this algorithm, but found the running time of it, in combination with our selected grid of hyper-parameters, to make the application of NCV infeasible. We therefore opted to just evaluate on a standard test set.
NCV can estimate the generalization error when no selection of optimized hyper-parameters is available. This is done by performing an "inner" CV and an "outer" CV in a nested fashion. The inner CV selects a model on the $n-1$ training folds of each iteration of the outer CV. The selected model is then tested on the held out fold of the outer CV and all the obtained test scores of the outer CV are ultimately averaged. For a more detailed description of the algorithm, see the work of Wainer and Cawley (2021).

3 Model Evaluation Results on Classical MIL Datasets

In this section, we will showcase our results and compare them with the baselines achieved in the first milestone. We will also compare our different models between each other, as well as compare them to the results of Ilse et al. (2018).
Overall we trained six different setups for each dataset by modifying both the MIL type and pooling type. We will refer to these different configurations by their MIL approach ("E" for embeddings-based and "I" for instance-based), connected with a dash to the name of their pooling method (e.g., "E-attention" for an embeddings-based model using the attention pooling).

3.1 Comparison with baseline

From the results in Tables 3 and 4 it is evident that the inclusion of a model selection layer utilizing CV can notably enhance certain metrics. This is particularly remarkable for the I-max and I-mean models trained on the `ELEPHANT` dataset.
However, in some cases the use of CV did not yield substantial differences or even worsened the results compared to the baselines. This can be explained by the fact that the baseline hyper-parameter values were also obtained with CV by Ilse et al.

Model	Configuration	Test Accuracy	Precision	Recall	F-Score	AUC
ELEPHANT(E-max)	Baseline	0.95000	0.95238	0.95238	0.95238	0.94987
ELEPHANT(E-max)	CV	0.92500	0.90909	0.95238	0.93023	0.92356
ELEPHANT(E-mean)	Baseline	0.90000	0.90476	0.90476	0.90476	0.89975
ELEPHANT(E-mean)	CV	0.90000	0.90476	0.90476	0.90476	0.89975
ELEPHANT(I-max)	Baseline	0.82500	0.85000	0.80952	0.82927	0.82581
ELEPHANT (I-max)	CV	0.92500	0.87500	1.00000	0.93333	0.92105
ELEPHANT(I-mean)	Baseline	0.90000	0.90476	0.90476	0.90476	0.89975
ELEPHANT(I-mean)	CV	0.92500	0.90909	0.95238	0.93023	0.92356

Table 3: ELEPHANT Model Performance

Model	Configuration	Test Accuracy	Precision	Recall	F-Score	AUC
TIGER(E-max)	Baseline	0.77500	0.77273	0.80952	0.79070	0.77318
TIGER(E-max)	CV	0.82500	0.79167	0.90476	0.84444	0.82080
TIGER(E-mean)	Baseline	0.75000	0.78947	0.71429	0.75000	0.75188
TIGER(E-mean)	CV	0.82500	0.79167	0.90476	0.84444	0.82080
TIGER(I-max)	Baseline	0.80000	0.78261	0.85714	0.81818	0.79699
TIGER(I-max)	CV	0.75000	0.73913	0.80952	0.77273	0.74687
TIGER(I-mean)	Baseline	0.80000	0.78261	0.85714	0.81818	0.79699
TIGER(I-mean)	CV	0.77500	0.77273	0.80952	0.79070	0.77318

Table 4: TIGER Model Performance

(2018). They, however, tested over other hyper-parameter values. Therefore, their results can be different and better in certain cases.

3.2 Comparison of the Attention Mechanism with the paper

METHOD	MUSK1	MUSK2	FOX	TIGER	ELEPHANT
Attention-paper	0.892 ± 0.040	0.858 ± 0.048	0.615 ± 0.043	0.839 ± 0.022	0.868 ± 0.022
Gated-Attention-paper	0.900 ± 0.050	0.863 ± 0.042	0.603 ± 0.029	0.845 ± 0.018	0.857 ± 0.027
Attention-cv	0.84211	0.71429	0.57500	0.80000	0.95000
Gated-Attention-cv	0.73684	0.71429	0.60000	0.77500	0.95000

Table 5: Method Performance on Various Datasets

When comparing our results to the ones reported by Ilse et al. (2018), one observation of note is that our trained model outperformed the ones from the original paper on the ELEPHANT dataset. Contrarily, across other datasets, particularly MUSK1, MUSK2, FOX, and TIGER, our models exhibited slightly diminished performance compared to the reported findings. This discrepancy can be explained by the fact that the results from the paper were achieved with hyper-parameter values we did not test for. Additionally, slight differences can also occur from the randomly and therefore potentially differently initialized weights of each of the models.

3.3 Evaluation of the Attention Mechanism

In this subsection, we will assess the attention and gated attention mechanism in comparison to other pooling methods. The Figures 1, 2 and 3 depict the outcomes obtained from the test data across all the different approaches on a selection of the classical MIL datasets. The shown plots contain results we deem interesting and representative of the overall outcome.
Across classical MIL datasets, the performance of all methods remained closely aligned. Notably, in the ELEPHANT dataset (refer to Figure 1), the attention mechanism exhibited strong performance compared to other mechanisms. On the remaining datasets (for representative results, refer to Figures 2 and 3) its impact was less

pronounced. This could be due to a general lack of suitability of neural networks to these comparatively small datasets, especially when they are outfitted with as many trainable parameters as the ones using variations of the attention mechanism.
In addition, it is worth noting that the gated attention consistently provided higher precision across most datasets, while the attention mechanism exhibited superior recall. This distinction implies their suitability for different applications based on specific requirements and preferences.

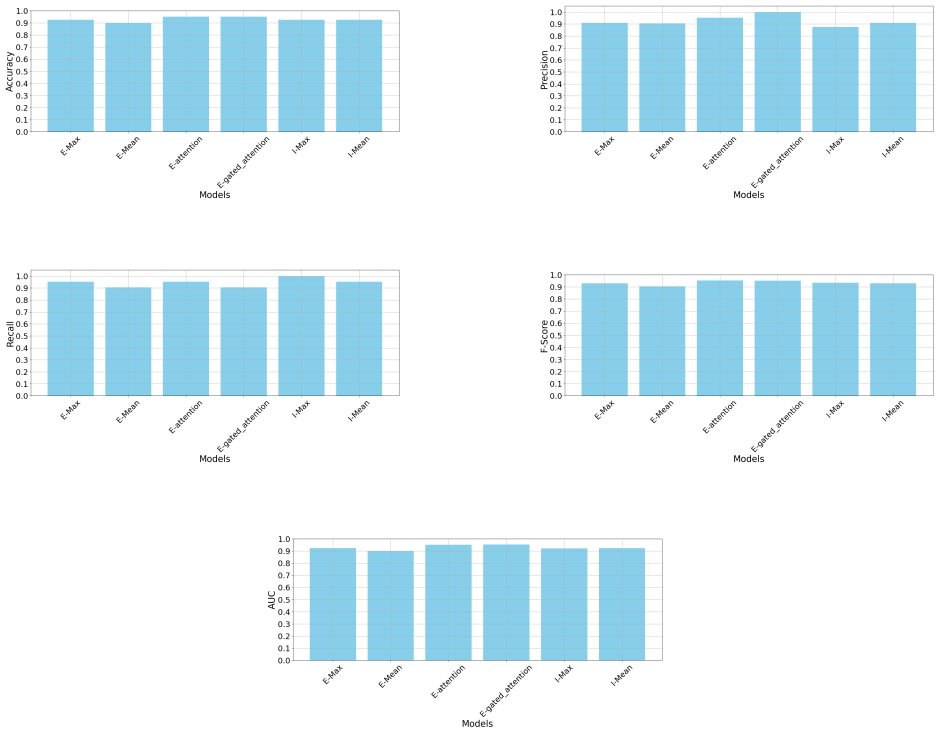

Figure 1: `ELEPHANT` dataset metrics

3.4 Comparison with the MNIST-Bags Results

Model	Test Accuracy	Precision	Recall	F-Score	AUC
MNIST-Bags (E-Max)	0.96000	0.98936	0.93000	0.95876	0.96000
MNIST-Bags (E-Mean)	0.97000	0.97959	0.96000	0.96970	0.97000
MNIST-Bags (E-attention)	0.97500	1.00000	0.95000	0.97436	0.97500
MNIST-Bags (E-gated_attention)	0.98000	0.98980	0.97000	0.97980	0.98000
MNIST-Bags (I-Max)	0.96500	0.97938	0.95000	0.96447	0.96500
MNIST-Bags (I-Mean)	0.72500	0.86885	0.53000	0.65839	0.72500

Table 6: Performance Metrics for MNIST-Bags Models

Table 6 showcases the outcomes on the `MNIST-bags` dataset. Within this subsection, we will elaborate on a few of the findings from these results, as well as delve into a comparison between the results of the classical MIL datasets and the `MNIST-bags` one.
The first striking observation is the clear superiority of the attention mechanism over other methods in the `MNIST-Bags` results, surpassing the performance achieved in all metrics when compared to the classical datasets. This may stem from its higher complexity compared to other methods. Its heightened complexity poses a risk of overfitting, which is more likely to occur due to the limited number of training samples in the classical datasets, while allowing it to model more complex function classes. The larger training dataset size in the `MNIST-bags` case might have allowed the attention mechanism based models (E-attention and E-gated_attention) to learn

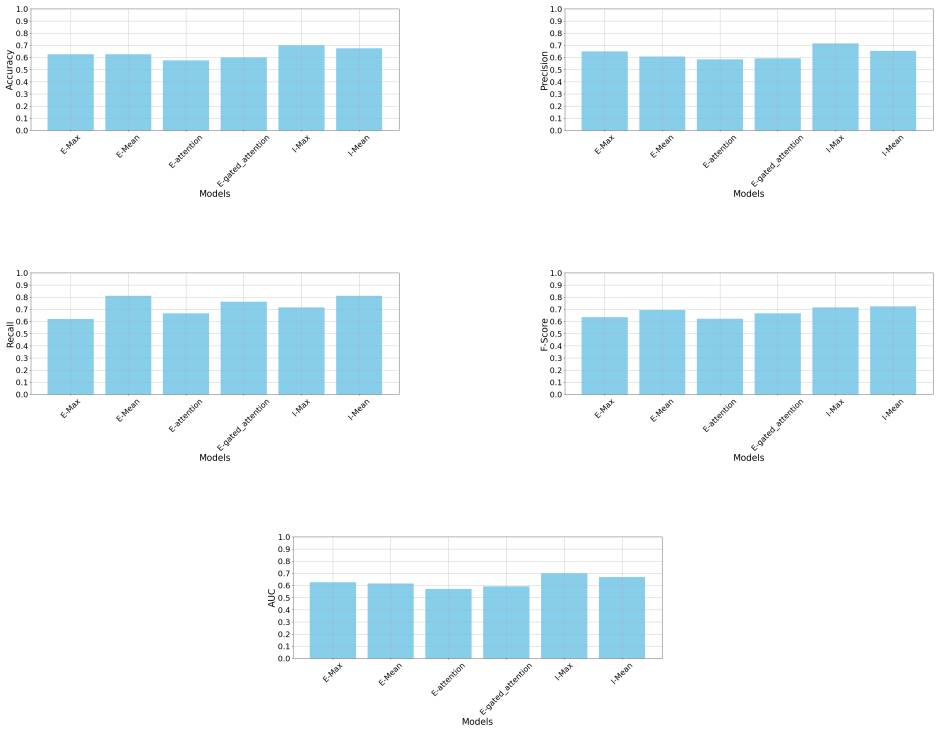

Figure 2: `FOX` dataset metrics

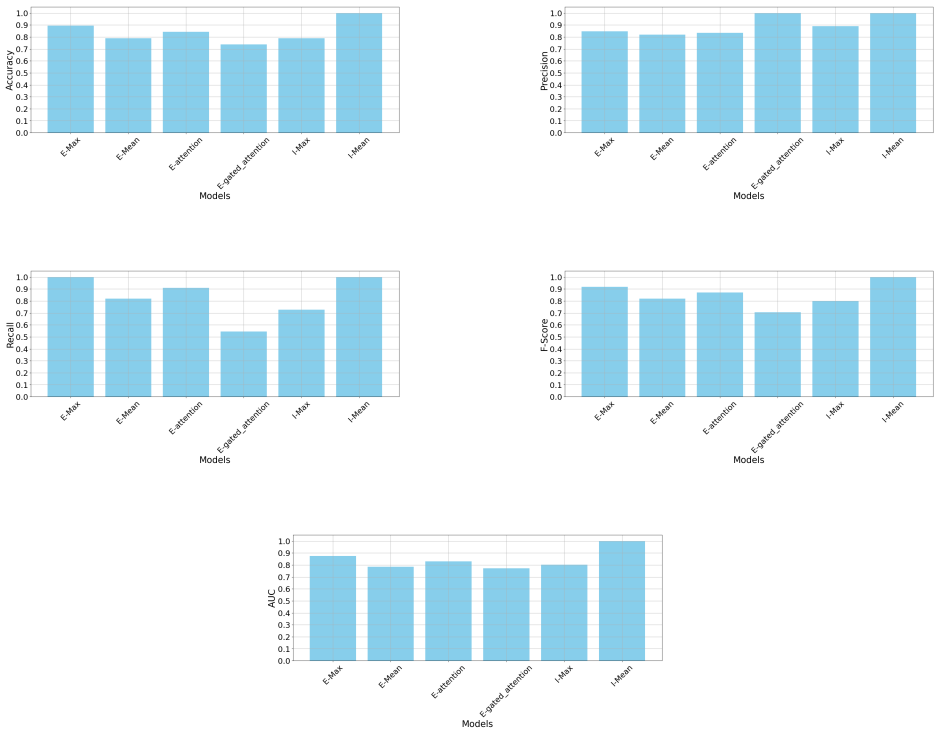

Figure 3: `MUSK1` dataset metrics

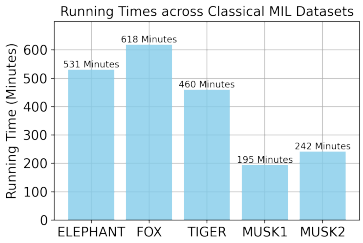

Figure 4: The running time that the training (including model-selection) took on average on the classical MIL datasets.

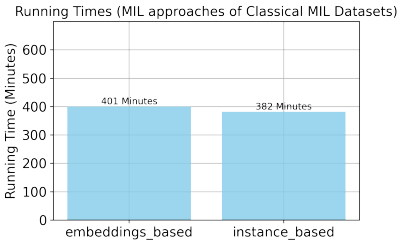

Figure 5: The average running time that the training (including model-selection) of the MIL approaches took on classical MIL datasets.

a much more robust function to classify positive bags. However, it could also be that the problem of separating positive MNIST bags from negative ones is much simpler to learn, compared to the learning problems of the classical MIL datasets.
Also, the `MNIST-bags` dataset contained a significantly larger volume of testing data. The increased test data volume allowed for a more accurate estimation of the generalization error and metrics on unseen data within this context.

4 Discussion

4.1 Training Time

In this subsection, we will be discussing the computational effort of training all of the presented models. This will include the analyses of running times of our training method, as well as possible explanations of these running times.

Classical MIL Datasets As explained in Section 1.2, all of our models contain a lot of hyper-parameters. Our chosen parameter grid led to 288 different hyper-parameter combinations that we needed to find the optimal one from.
On average the whole training process, including model-selection and evaluation on unseen test data, took 394 minutes. However, this number was obtained by excluding two outlier running time results. Training the attention and gated attention mechanisms on the `FOX` dataset took 1117 minutes and 1434 minutes, respectively. This is, on average, three times longer than the average training time of all the other models. It is also more than two times as long as the average training time for all the non-outlier models trained on one of the `FOX`, `ELEPHANT` and `TIGER` datasets, which all are more suitable to compare to the two outliers, as they all have the same dataset size.
The training of the other, non-outlier models that were trained on the `FOX` dataset did not significantly stand out in terms of their training time, which leads us to believe that this great increase in training time for the two mentioned outlier models stems from the execution using the "SLURM"[1] workload manager on the cluster the models were trained on. A closer investigation of the scheduling statistics showed that the training of these two models was never "suspended" for any reasons, such as an unavailability of necessary resources.
However, the two models were trained simultaneously with at least three other models on the same compute-head of the cluster. On all the other compute-heads a maximum of three simultaneous model trainings took place. We could imagine that this sharing of resources caused these two jobs to have had to halt their trainings for significant amounts of time, until they were able to continue training uninterruptedly. This may have caused the outstandingly long training times for just these two model configurations.

The plots in Figures 4, 5 and 6, depicting running times for different groups of models, are excluding the mentioned two outlier results. From Figure 4 we can observe that the dataset size seems to have a relatively strong effect on the training time. The `MUSK1` and `MUSK2` training datasets consist of 73 and 81 bags respectively,

[1] slurm.schedmd.com

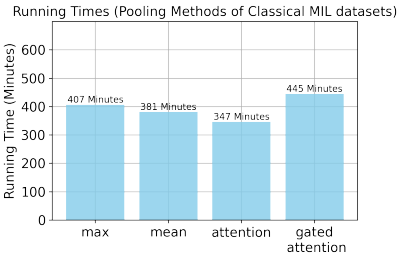

Figure 6: The average running time that the training (including model-selection) of the pooling methods took on classical MIL datasets.

which is, on average, about half of the 160 bags contained within any of the `FOX`, `ELEPHANT`, and `TIGER` datasets.
This makes sense, since the training for one hyper-parameter combination should scale as $O(N \cdot N_{epochs} \cdot \sum_{i=1}^{L-1} W_i)$, where N is the number of bags in the dataset, N_{epochs} is the number of epochs chosen as one of the hyper-parameters, L is the number of layers in the networks and W_i is the number of weights connecting layer i to layer $i+1$ of the model. Since we always test over the same set of N_{epochs}, and all the different configurations for the W_i are also the same over all of the classical MIL datasets, N is the biggest determinant behind the differences in measured running times for the different datasets.
Since for the embeddings- and instance-based approach the sets of different values for N and N_{epochs}, as well as the different configurations of the W_i, are the same, we are getting very similar average training times for the two approaches. This can be observed in Figure 5.
The attention and gated attention mechanism have an additional $64 \cdot 64 + 64$ and $2 \cdot (64 \cdot 64) + 64$ parameters respectively compared to the max- and mean-pooling-based models. This corresponds to a proportionally very small increase in the overall number $\sum_{i=1}^{L-1} W_i$ of parameters of the models, which, without the attention and gated attention layers, already is of size $S_{input} \cdot 256 + 256 \cdot 128 + 128 \cdot 64$. Here S_{input} corresponds to the input size of the dataset. Therefore, as apparent from Figure 6, the choice of the pooling operation did not play a significant role for the training time.

MNIST-bags The models of the `MNIST-bags` dataset had to be selected from the same grid of hyper-parameters as laid out in Section 1.2. However, in this scenario we are training on an order of magnitude more training bags. Here we trained on 2000 training bags, where as the classical MIL datasets had an average training set size of 127. This is reflected in a great increase in the overall training time:
As can be seen from Figures 7 and 8, the training of each `MNIST-bags`-model took on average several days. However, these massive increases can be explained purely as a result of the increased training set size. Dividing the average amount of time the training took in each of the two cases by the amount of parameter combinations trained on $(288 + 1)$, and further dividing by the (average) training set size leads to approximately 0.11 minutes as the average training time per bag in the case of the classical MIL datasets and 0.12 minutes for the `MNIST-bags` dataset.

Retraining the Models In the real world, it is desirable to be able to efficiently incorporate new training data into the models predictions, once such data has become available. For our model, this is possible, under a (potentially strong) assumption: The found hyper-parameters still work well also for the new data.
If this assumption is fulfilled, we can just additionally feed the new data forward through the model we want to re-train and perform error backpropagation using the optimizer that was found in the model-selection process.
If this assumption is not fulfilled, it is better to re-run the whole model-selection process outlined in Section 1.3. For reasons described in the preceding part of this subsection, this is not particularly efficient.

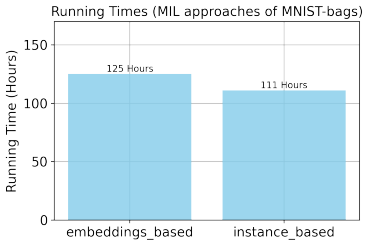

Figure 7: The average running time that the training (including model-selection) of the MIL approaches took on the `MNIST-bags` dataset.

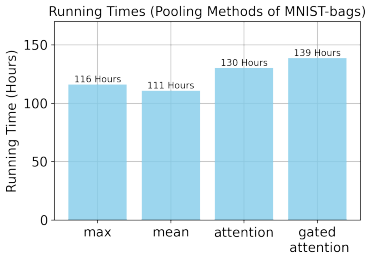

Figure 8: The average running time that the training (including model-selection) of the pooling methods took on the `MNIST-bags` dataset.

4.2 Confidence of the Models

The confidence of a model is a likelihood-value output by the model, which indicates how sure the model is about its prediction. However, this is only really useful if this confidence is "well-calibrated", i.e., it is representative of the "true correctness likelihood" (Guo et al. (2017)).
For all of our models either the last or the second to last layer consist of a "sigmoid" activation function. This function outputs values between 0 and 1. Therefore, this output can under certain circumstances be interpreted as a probability of how likely the model "considers" the input bag to be positive.
However, as Guo et al. (2017) describe in their paper, it is unlikely that our models will be well-calibrated after training without any further adjustments. Several calibration methods exist that can be applied after the training has been completed. This is not something we will be attempting in this milestone and is thus left as a potential future investigation.

4.3 Conclusion

Regarding the classical MIL datasets, the detailed analyses in Section 3 showed that performing CV was able to improve many or our chosen evaluation metrics in comparison to the results from our baselines (see Tables 3 and 4). Additionally, our selected models for the attention and gated-attention mechanisms achieved comparable results to the ones from the paper of Ilse et al. (2018). However, on the classical MIL datasets the two forms of the attention mechanism did not improve upon the results of the max- and mean-pooling models.
In part we think that this is caused by a general ill-suitedness of (deep) neural networks to these datasets. Due to the large number of trainable parameters and the small size of the training set, these models should already have trouble generalizing. The attention mechanism, with its additional trainable parameters, only amplifies this problem.
Additionally, it must also be said that these results were obtained on small test sets. Therefore they might generally lack strong validity.
On the `MNIST-bags` dataset, the attention and gated-attention mechanisms showed much more promise. Their accuracy values exceeded the average accuracy value of the mean-pooling based models (E-mean and I-mean) by an absolute value of about 0.13. They also exceeded the average accuracy of the instance-based models by a similar margin. We deem the larger training volume to be an important reason behind this. Unfortunately, as Section 4.1 showed, this increased efficacy comes at the cost of an increased training time.

References

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

M. Ilse, J. Tomczak, and M. Welling. Attention-based deep multiple instance learning. In *International conference on machine learning*, pages 2127–2136. PMLR, 2018. URL https://arxiv.org/abs/1802.04712.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/sutskever13.html.

J. Wainer and G. Cawley. Nested cross-validation when selecting classifiers is overzealous for most practical applications. *Expert Systems with Applications*, 182:115222, 2021.