Distinguishing 4-top from background events in the Large Hadron Collider

Aucke Bos
s4591496
(Dated: April 28, 2021)

A 4-top event is an extremely rare event where four top quarks are produced in one event. The search for 4-top events is a hot topic. This is an interesting area of research since if we would be able to detect and measure it correctly, it might give rise to new particles beyond the standard Model[1]. In this experiment, we use an artificially created dataset to train several neural networks to distinguish 4-top events from background. We elaborate on challenges encountered in this task, performance, and compare results.

I. INTRODUCTION

This paper is part of an assignment for the course Machine Learning in Particle Physics and Astronomy at the Radboud[1]. The assignment provides a simulated dataset of 2.000.000 records. Each record represents the measurement of a single event. For each event, the magnitude and azimuthal angle of the missing traverse energy vector is provided, along with the detected particles. For each particle, we get the particle type with its charge, and the 4-vector: the energy, the traverse component of the momentum and the theta, and the phi angle.

The task is to design, train, and evaluate a model of our own choice, to distinguish the background events from 4-top candidate events. A 'baseline' has to be established by a *binary classification neural network*. Another network must be trained as a *multi-label classifier*; its classifications must be used with Bayes formula to derive posterior probabilities for the event types. These posteriors will be used to optimally distinguish 4top candidates from background. In the last part of the assignment, the task is to improve the performance of the network in any way that is found effective. The source code, along with the best performing models, can be found on Github[2]. Any reference to `python classes` or code will refer to files in this repo unless specified otherwise.

The second part of the assignment is this paper, in which we will describe our process, evaluation, and conclusions. In the next section, we describe the dataset in more detail, we also state what preprocessing we have applied. Next, we describe three network structures with which we try to tackle the problem. The first classifier servers as a baseline. The second classifier is a little more sophisticated, as it learns to do multi-classification and includes priors about data during the test phase; it's also able to convert the class labels into binary classification labels. The last network is an attempt to improve performance by using a Recurrent neural network, which has been shown to be successful in related research [2][3]. The next section elaborates on design choices that are made and discusses some of the aspects of the network in more detail. Finally, we dedicate a section to the results of the best-performing model, compare and discuss the model performances, and finally state the conclusions we can draw from this research. Supplementary figures can be found in the appendix.

II. METHODS

A. Preprocessing

The `DataLoader` uses Pandas[3] to load the dataset. Each event-level characteristic has its own column, they are initially stored as comma-separated string in their own column. Since each event has a variable number of objects, we applied zero-padding up to the number of 4-vectors of the event with the most objects. We add an extra column that holds the number of non-null vectors, e.g. the actual object count. We split each 4-vector into separated columns, and extract the charge $\in [-1, 0, 1]$ of an object to a new column. We use one-hot encoding for the object types and labels (Process IDs), each component of the one-hot vector gets its own column. Finally, we fill NaNs (events with fewer measured objects than the maximum object count) and normalize the data to range $[0, 1]$. The resulting dataset has shape [2.000.000 x 212]: We have 19 objects per event since the event with the most measured objects counts 19 objects. There are 6 different object types measured, plus the charge and 4-vector values, this gives 11 values per object. There are 2 event-level values, plus the VectorCount column. This gives us $11 \cdot 19 + 3 = 212$ values.

The `DataLoader` can process the labels as either multi-label or binary. In the case of binary, `y` is a single column, where a value of 1 means an event is a 4-top event. In the case of multi-label, the labels are one-hot-encoded. The resulting `y` is a 5 column one-hot encoded DataFrame for events 4top, ttbarHiggs, ttbarW, ttbarZ and ttbar.

[1] `https://www.ru.nl/courseguides/science/vm/osirislinks/nm-0/nwi-nm116b/`
[2] `https://github.com/AuckeBos/MLiPPaA`
[3] `https://pandas.pydata.org/`

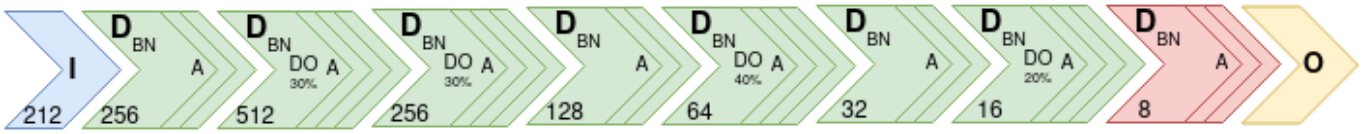

FIG. 1. The structure of the `BaseClassifier`. I, O, D_{XXX}, BN DO and A are Input, Output, Dense, Batch Normalization, Dropout, and Activation layers respectively. Output sizes, as well as the dropout rates, are provided. For all activation layers, we use Leaky Relu activation functions.

B. The classifiers

1. *The BaseClassifier*

The `BaseClassifier` is used as a basis for both the baseline model and the first multi-classification model. Its structure is used as a basis for the next two networks and is shown in Figure 1. It provides functionality for training and testing the network, as well as generating train/validation/test sets. Finally, it provides the option to apply Bayes during testing, which we will discuss more extensively in Section III C.

2. *The BinaryClassifier*

The `BinaryClassifier` serves as a baseline. It uses the structure of the `BaseClassifier` but includes a `Dense(1)` layer with sigmoid activation just before the output layer. It uses Adam optimizer, binary cross-entropy as a loss function, and measures accuracy and F1-score to visualize performance. In Appendix B we compare the performance of this classifier to the performance of our `MultiClassifier` when used as a binary classifier.

3. *The MultiClassifier*

The `MultiClassifier` Inserts a `Dense(5)` layer with softmax activation before the output layer. It uses Adam with categorical cross-entropy, and measures F1 and accuracy.

4. *RecurrentClassifier*

The structure of the `RecurrentClassifier` is shown in Figure 2. It instructs the `DataLoader` to mask non-existing vectors per event with a special masking value. The masking layer filters out these vectors. We then add three LSTM layers with Tanh activation that return sequences of the output length. A final LSTM is added that only passes on the final output. This output is concatenated with the three event-level inputs. We downsample to an output size of 5 using two dense layers, with a final softmax activation. This network too uses Adam with categorical cross-entropy and measures accuracy and F1.

III. DESIGN CHOICES

A. Rebalancing the dataset

As mentioned, the dataset provided for the assignment has been artificially created. The major problem with this dataset is shown in Figure 3. We see that the class imbalance is not in line with real-world data. 4top events are extremely rare, and thus definitely do not occur in 50% of the cases! The classifiers therefore provide the possibility to rebalance the datasets. We can provide the desired class probability, and whether we want to rebalance the test set, and/or the train and validation sets. It then rebalances the set to match this distribution by downsampling the majority classes. We elaborate on the influence of rebalancing on the performance of the network in Section IV.

B. Performance metric

Accuracy is one of the most used performance metrics in machine learning[4][5]. We therefore initially used this metric to evaluate performance. As also shown in the literature[6][7], accuracy is not suitable for imbalanced data sets. A more natural balance of 4-top vs non-4-top events would be 4% against 96% respectively. We rebalanced our complete (train/validation/test) set to this distribution and trained our binary classifier on this dataset. Training and validation accuracy is shown in Figure 4. Our network learns to simply always predicts 'no 4 top', hereby achieving an accuracy of 96% on the validation set! This same score is achieved on the rebalanced test set.

Therefore we should not use accuracy as the main and sole evaluation metric for this problem. We chose for F1 score instead. This metric is able to deal with class imbalance, as it takes into account the relation between precision and recall. Since the network described above always predicts '0', the recall, and hereby the F1 score, would be 0. Thus our network can only achieve a decent F1 score if it predicts '1' for at least some data points.

At the very end of the assignment, when the test set was provided, we were told that the metric used for final evaluation of performance is AUC[4]. At this point we had

[4] `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`

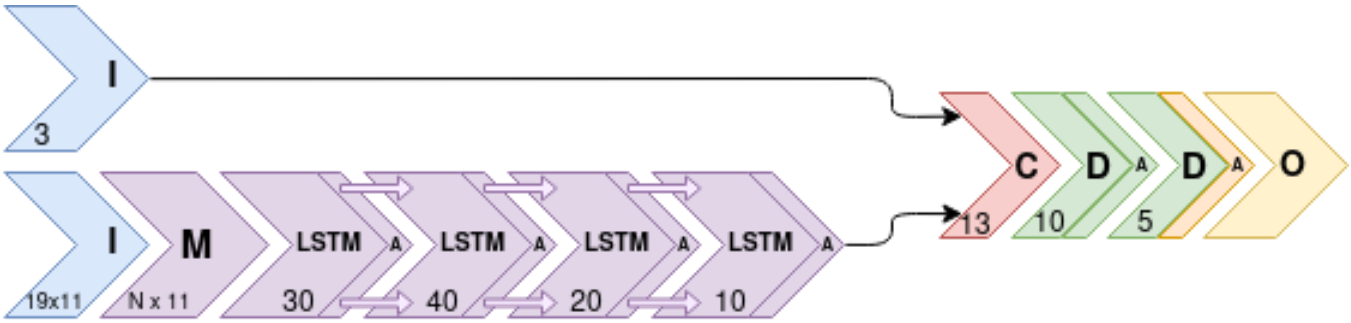

FIG. 2. The structure of the `RecurrentClassifier`. I, O, M, LSTM, C, D, and A are Input, Output, Masking, LSTM, Concatenate, Dense, and Activation layers respectively. Output sizes are provided. All but the last LSTMs return sequences instead of single outputs. The masking layer handles variable length input. The LSTMs use Tanh activation, The first dense uses Leaky Relu, the second dense uses a softmax activation.

already run all our evaluations, therefore we decided not to include this metric in our discussion.

C. From classification output to posterior probabilities

As described in Section III A, we provide the possibility to rebalance the test and/or train+validation sets. When we do so, we change the class distribution heavily. If we only rebalance the test set without rebalancing train+validation sets, we can apply Bayes Theorem during testing to improve performance on the test set. Our network is predicting the probabilities for each class given the data: $P(C|D)$. Bayes tells us that this value is given by $P(C|D) = P(C) \cdot \frac{P(D|C)}{P(D)}$. This equation obviously holds during the training and testing phase:

$$P_{tr}(C|D) = P_{tr}(C) \cdot \frac{P_{tr}(D|C)}{P_{tr}(D)}$$
$$P_{te}(C|D) = P_{te}(C) \cdot \frac{P_{te}(D|C)}{P_{te}(D)}$$

Both $P(D|C)$ and $P(D)$ are intrinsic to the data, thus these values will not change when we change the distribution our train/validation/test sets. Hence we have $\frac{P_{tr}(D|C)}{P_{tr}(D)} = \frac{P_{te}(D|C)}{P_{te}(D)}$. As such we can derive that during testing phase, we can compute $P_{te}(C|D)$, which is our prediction for the posterior distribution for each class, via

$$P_{te}(C|D) = P_{te}(C) \cdot \frac{P_{tr}(C|D)}{P_{tr}(C)} \tag{1}$$

$P_{te}(C)$ and $P_{tr}(C)$ are the priors of our test and training set respectively. We can compute these values via the distribution of our sets (for the final test set the prior is provided). $P_{tr}(C|D)$ is the prediction our network makes for a data point in the test set after we have trained on the training set. We elaborate on the influence of applying this formula during the testing phase in Section IV.

D. Weighted loss function

Because the class distribution is so imbalanced by nature, we might be able to improve performance using a weighted loss function[8]. Instead of the standard categorical cross-entropy, we also provide a weighted version of the same loss function. In this version each class gets a certain weight; before summing the losses for each class, we multiply each loss with the weight of that class. Thus classes with higher weights participate more in the final loss. We elaborate on the influence of weighting in the loss function in the next section.

IV. RESULTS

In this section, we will produce the results on the evaluation of different design choices we have made. While evaluating our choices, we split our data set into a train, validation, and test set, containing respectively 70%, 20%, and 10% of the data. Note that the actual test set will be provided later on during this experiment; since labels won't be provided, we can't use it in our evaluation. During the experiments, we train the network on the training set; the validation set is used to determine when to Early stop and Reduce the learning rate (see Section IV A). Both the validation and test set are used to evaluate our performance. We use two sets such that we can accurately measure the influence of rebalancing none, one, or all of these sets.

A. Configuration

We provide and elaborate upon two tables. Table I shows results for our standard `MultiClassifier` (Section II B 3); Table II shows results for our `RecurrentClassifier` (Section II B 4). For both of these classifiers, we use the following configuration:

1. Train for at most 150 epochs, the batch size is 64.

2. Apply an EarlyStopping callback with patience of 8, and a ReduceLearningRateOnPlateau with a factor of 40% and patience of 3.

3. Evaluate performance on both the validation and test set after each epoch. Above callbacks are based on the loss of the validation set. The test set is evaluated solely for performance evaluation

4. During testing, we convert the predicted class probabilities to a single class by selecting the class with the highest probability.

We define four binary variables. We compare performance on the validation and test set on all combinations of the values:

1. Apply Bayes Theorem during the testing phase. If true, we apply Equation 1 when calculation posterior class probabilities.

2. Use a weighted version of categorical cross-entropy. To stimulate the network to focus on 4-top events, we increase the weight for this class to 10 times the weight for the other classes, if this variable is true.

3. Rebalance the test set. If true, we rebalance the test set to a distribution of [4%, 23.75%, 23.75%, 23.75%, 23.75%], to be more in line with the natural occurrence of the classes.

4. Rebalance the train and validation set. If true, we rebalance the sets to the same distribution as above.

B. Evaluation

We investigate the results of Tables I-II in this section. Noteworthy observations are stated per classifier. A general discussion of these results, along with possible improvements and a discussion is provided in Section V.

1. MultiClassifier

We see that the best F1 score for our test set is much lower than the best F1 score for the validation set, although the lowest loss value is almost equal for both sets. It's also shown that the F1 score is lower than the accuracy in most cases for both sets. The network stops early on average at 56 epochs, thus training does not improve the performance any further at that point. We also see that the configuration with the lowest loss does not necessarily provide the highest F1 score, although in general, a lower loss correlates with higher F1 and accuracy. Although applying Bayes and rebalancing the test set should hypothetically affect performance on the test set heavily, both of these variables do not systematically increase nor decrease performance.

2. RecurrentClassifier

Although we would expect this classifier to achieve a better performance, Table II shows us the opposite: it systematically performs worse than our `MultiClassifier`. On average the training curve flattens with fewer epochs, the best loss is higher, and our best F1 is lower. The best performing models also have different configurations than those in Table I, indicating that no configuration is clearly better than the others.

V. DISCUSSION

Unfortunately, the results of both networks are quite disappointing. An F1 score of about 0.6 is not very promising. We also had higher hopes for increased performance with a weighted loss function and applying Bayes in combination with a re-balanced test set. Both of these configurations do not show foolproof and increased performance. The weighted version of categorical cross-entropy also doesn't seem to consistently increase F1 nor accuracy. Although we have yet tested several different network structures with varying depths and provided performance for the best versions, we might be able to get a higher score using other network structures. Future work includes using GNNs, CNNS, or CRNNs. Testing all these different approaches falls outside of the scope of this project, but specifically CNNs or even CRNNs might be well suited for this problem. We might also try to include more domain knowledge into the training data via preprocessing, for example by extracting more features out of the training data. This was hard in our case, due to the lack of domain experience.

VI. CONCLUSION

We conclude this paper by saying that our experiments provide a good baseline and starting point to solve this problem, but further research is required to tackle this problem with decent performance. Our models do not perform well enough to be usable in practice, but they can be extended and improved using different network structures, more hyperparameter tuning, and more data pre-processing. Most advantages can probably be achieved by more including more domain knowledge in the training data and investigating more suitable network structures.

[1] A. Collaboration, Evidence for $t\bar{t}t\bar{t}$ production in the multilepton final state in proton-proton collisions at $\sqrt{s}$=13 TeV with the ATLAS detector, arXiv:2007.14858 [hep-ex] 10.1140/epjc/s10052-020-08509-3 (2020), arXiv: 2007.14858.
[2] J. Pearkes, W. Fedorko, A. Lister, and C. Gay, Jet Constituents for Deep Neural Network Based Top Quark Tagging, arXiv:1704.02124 [hep-ex, physics:hep-ph, stat] (2017), arXiv: 1704.02124.
[3] S. Egan, W. Fedorko, A. Lister, J. Pearkes, and C. Gay, Long Short-Term Memory (LSTM) networks with jet constituents for boosted top tagging at the LHC, arXiv:1711.09059 [hep-ex, physics:hep-ph, stat] (2017), arXiv: 1711.09059.
[4] M. F. Uddin, Addressing Accuracy Paradox Using Enhanched Weighted Performance Metric in Machine Learning, in *2019 Sixth HCT Information Technology Trends (ITT)* (IEEE, Ras Al Khaimah, United Arab Emirates, 2019) pp. 319–324.
[5] A. K. Gopalakrishna, T. Ozcelebi, A. Liotta, and J. J. Lukkien, Relevance as a Metric for Evaluating Machine Learning Algorithms, in *Machine Learning and Data Mining in Pattern Recognition*, Vol. 7988, edited by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and P. Perner (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013) pp. 195–208, series Title: Lecture Notes in Computer Science.
[6] N. Japkowicz, enWhy Question Machine Learning Evaluation Methods ? (An illustrative review of the shortcomings of current methods) (2006).
[7] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, A new evaluation measure for learning from imbalanced data, in *The 2011 International Joint Conference on Neural Networks* (IEEE, San Jose, CA, USA, 2011) pp. 537–542.
[8] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, Multi-Task Learning and Weighted Cross-Entropy for DNN-Based Keyword Spotting (2016) pp. 760–764.

Appendix A: An imbalanced dataset

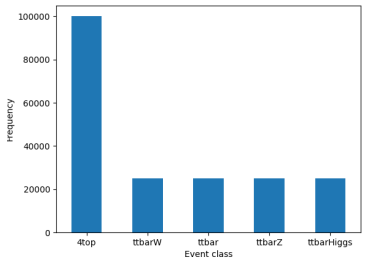

FIG. 3. The class distribution of the dataset. It is not a good representation of actual measurements.

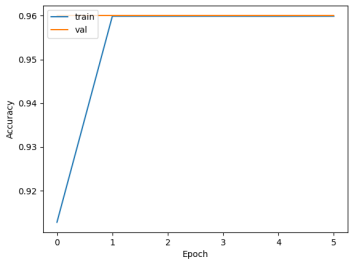

FIG. 4. Accuracy plot for the binary classifier with the total dataset rebalanced to [96% (no-4-top), 4% (4-top)]. Accuracy is not a good metric for imbalanced datasets.

Appendix B: Comparing binary evaluation

As part of the assignment, we evaluate whether our `MultiClassifier` outperforms our `BinaryClassifier`. To do so, we also train the binary classifier with the sixteen different configurations. We have trained both classifiers on all configurations and evaluated their performance on the same test set. We use two versions of this set: one that is rebalanced to [96%, 4%], and one that is not rebalanced; we do not apply Bayes during the comparison. Therefore the two variables 'Rebalance test' and 'Apply Bayes' are irrelevant for this comparison, which gives us 4 different configurations. The class label of the `MultiClassifier` is converted to '1' if the class is 'htop', else to '0'. We present F1 on both sets for both classifiers in Table III. As we can see, if we use the `MultiClassifier` as a binary classifier, it almost always outperforms our baseline `BinaryClassifier`. The best scores are mostly higher, and on average it also scores much better. We also see that rebalancing the train and validation set improves performance on the rebalanced test set. Although the scores are still not very good, we clearly see that our `MultiClassifier` is better in distinguishing 4-top events from background.

Appendix C: Comparing network configurations

TABLE I. Performance comparison for the `MultiClassifier` for all combinations of 4 design choices (see Section IV A). Bold cells indicate the lowest (loss, epochs, LR) or highest (accuracy, F1) values of the row.

Configuration index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Apply Bayes	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	
Rebalance test	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	
Rebalance train/val	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	
Weighted loss	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	AVG
Validation loss	1.620	1.309	1.448	0.944	1.621	1.301	1.449	0.946	1.607	1.305	1.448	0.949	1.599	1.301	1.449	**0.944**	1.327
Test loss	1.317	1.367	1.428	1.295	0.979	1.416	1.117	0.942	1.410	1.292	1.900	1.559	0.984	1.397	1.121	**0.940**	1.279
Validation accuracy	0.345	0.407	0.582	**0.624**	0.351	0.412	0.590	0.623	0.351	0.412	0.584	0.620	0.349	0.411	0.586	0.624	0.492
Test accuracy	0.400	0.407	0.356	0.410	0.615	0.417	0.595	0.627	0.356	0.416	0.213	0.317	0.614	0.434	0.591	**0.628**	0.462
Validation f1	0.139	0.184	0.591	0.608	0.142	0.196	0.594	**0.609**	0.150	0.187	0.590	0.604	0.142	0.199	0.596	0.608	0.384
Test f1	0.361	0.326	0.335	0.405	**0.434**	0.390	0.316	0.411	0.330	0.409	0.156	0.267	0.432	0.391	0.326	0.413	0.356
Epochs	46	77	62	61	**37**	60	44	58	39	69	56	68	48	62	47	69	56
LR	4.096e-7	**1.678e-9**	1.049e-8	4.096e-7	2.560e-6	1.024e-6	4.096e-7	4.096e-7	2.560e-6	2.621e-8	4.096e-7	**1.678e-9**	4.096e-7	1.638e-7	1.638e-7	4.096e-7	5.862E-07

TABLE II. Performance comparison for the `RecurrentClassifier` for all combinations of 4 design choices (see Section IV A). Bold cells indicate the lowest (loss, epochs, LR) or highest (accuracy, F1) values of the row.

Configuration index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Apply Bayes	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	
Rebalance test	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	
Rebalance train/val	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	
Weighted loss	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	AVG
Validation loss	1.755	1.400	1.672	1.056	1.780	1.378	1.683	1.053	1.743	1.397	1.691	**1.050**	1.832	1.400	1.676	1.058	1.476
Test loss	1.384	1.464	1.503	1.374	1.099	1.589	1.370	**1.051**	1.516	1.393	2.574	1.666	1.090	1.604	1.383	1.057	1.445
Validation accuracy	0.301	0.340	0.549	0.581	0.273	0.356	0.545	0.582	0.287	0.341	0.545	**0.584**	0.240	0.341	0.545	0.581	0.437
Test accuracy	0.350	0.344	0.304	0.359	0.570	0.343	0.551	**0.586**	0.296	0.344	0.140	0.252	0.574	0.374	0.550	0.584	0.408
Validation f1	0.060	0.012	0.554	**0.568**	0.034	0.049	0.551	0.567	0.062	0.012	0.551	0.567	0.036	0.010	0.549	0.566	0.297
Test f1	0.298	0.242	0.272	0.334	**0.354**	0.327	0.233	0.336	0.268	0.327	0.109	0.211	0.338	0.309	0.232	0.332	0.283
Epochs	52	31	42	**30**	43	74	41	50	62	40	32	48	31	39	38	47	43.75
LR	6.400e-6	2.560e-6	2.560e-6	6.400e-6	1.024e-6	6.400e-6	6.400e-6	**1.638e-7**	1.024e-6	1.024e-6	6.400e-6	1.024e-6	1.024e-6	1.024e-6	6.400e-6	1.024e-6	3.178e-6

TABLE III. Performance comparison on binary classification between our `BinaryClassifier` and our `MultiClassifier` for the combinations of 2 design choices (see Section IV A). Bold cells indicate the highest scores of the row. The rebalanced test set is rebalanced to [96%, 4%]. The `Multiclassifier` scores better.

Configuration index	3	5	6	12	
Rebalance train/val	F	T	F	T	
Weighted loss	F	F	T	T	AVG
Binary F1 rebalanced	0.612	**0.673**	0.039	0.204	0.382
Multi F1 rebalanced	0.534	**0.713**	0.346	0.599	0.548
Binary F1 not rebalanced	**0.865**	0.576	0.334	0.512	0.572
Multi F1 not rebalanced	0.839	0.656	0.668	**0.856**	0.755