Week 17

0. Table of Contents

1. Introduction

The main purpose of this week is to bring out a concrete direction in the numerical research done so far, as well as continue carrying it out.

2. Story

The aim of the second part of our project was to generate neural network automatised recovery of intrinsic properties within our persistent exclusion process system by evaluating experimental data. The motivation is twofold: pragmatically, the aim is to provide a proof of concept engineering of a tool which can evaluate and identify active matter systems. To this end, generalisation of phenomena is key; such a neural network should be able to adapt to a vast variation of parameters, namely tumbling rates, densities and sample sizes. Furthermore, there is a theoretical component to the utility of this research - on one hand, a machine learning algorithm reproducing the inherent tumbling quantity of a system can serve as external validation of active matter phenomena; with the caveat, of course, that this external validation (by which we mean externalised, removed from human judgement) is nonetheless trained and verified with human-set metrics and understanding.** On the other hand, obtaining a generalised and automated tool for evaluating active matter systems can point human focus to the interesting regions of behaviour, thus supplementing analytical examination.**

The algorithm returns various predictions for every tumbling-density-size system it is applied to; these predictions have a spread, which we intend to lower and center around the real tumbling rate, but nonetheless it is important to note that it does not provide a discrete singular prediction. One goal is to qualitatively evaluate the distributions of predictions, with the hopes of a gaussian-like organisation centred on the actual tumbling rate. In this sense the algorithm exhibits human-like prediction. The crux of the issue is determining satisfactory metrics for evaluating the systems. The goal, however, is not to perfectly fine-tune the system as much as to make it applicable enough in order to highlight different qualitative distinctions and how they influence the algorithm's output - with the explicit intent to link to qualitative phenomena which we have examined in the first part of our project.

1. Degrees of freedom

One of the important explorations is examining how changing the input degrees of freedom alters the machine output. By degrees of freedom, we specifically mean positions and orientations - the CNN can be fed either 'experiment-like' images, which present only black and white displays of our active matter system, and thus only allows the network to extrapolate on positions, or it can be fed afferent orientations as well, as colours for each particle, and henceforth weight its parameters with this extra difference. We can then explore how training the CNN on one degree of freedom setup influences its predictions on the other. Our expectations going in are that:

- training the CNN on strictly positions will have similar predictions when validating it on positions+orientations; this is because it will/might simply approach the coloured landscape the same way it approaches the monochrome landscape, by contrasting existing particles with the black background.
- training the CNN on positions and orientations will have worse preditions when validating it just on positions; this is because, having been trained to recognise colour and incorporate it in predictions, the system might struggle to 'explain' the images which now lack this degree of freedom

If these expectations prove to be correct, training the CNN on positions+orientations might be a worse prospect, due to most real life applications/evaluations of active matter not being able to access the orientation of agents through still images. Nonetheless, training on positions+orientations may give the CNN a better insight into the clustering phenomenon as a whole, due to the role particle orientations play in cluster formation and evaporation.

Once we've explored how degrees of freedom alter the CNN predictions, it is useful to explore some modifications to the orientation+position setup. An interesting extension is exploring how misinformation affects the neural network. Some active matter imaging, of, say, Janus particles rolling in 2D, can trace the orientation of a particle. But this tracing is imperfect, and there are some ambiguous cases where orientation may be misread. As such, we could engineer a dataset which 'misreads' the orientation of a certain ratio of the total particles, recolouring them.

2. Predicting untrained tumbling rates (interpolation, extrapolation)

Another important exploration is how training on a certain set of tumbling rates can then cause the CNN to be able to accurately predict tumbling rates in different regions. Our preliminary experiments have not been able to replicate such results - attempting to predict tumbling rates spaced out inbetween the training tumbling rates has caused the system to attempt to fit its predictions to the neighbouring training rates, and therefore given a very large spread to the data. This may however be strictly due to the low training data we have given the CNN so far - both low amount of original evolutions to give the system more depth of experience, as well as low amount of distinct tumbling rates. It ma

Here are our expectations regarding this part of the project:

- Using interspersed sets of training/validation will yield accurate predictions (on the validation samples) provided the spacing between values is low enough.
- Using vastly separated sets is going to fare poorly in prediction, due to different clustering behaviour which the CNN is not adequately accustomed to.

3. Predicting untrained densities

Training on a certain set of densities might also present interesting behaviour when applying the algorithm on a different set of densities.

A noteworthy extension of our research is to then apply this neural network to real active matter systems by ‘translating’ physical scenarios into the visual language our tool is familiar with.

We have experimented with various architecture models (a summary of the different architectures can be found here). From now on we will be mostly running the MN_3 (discussed in a section below). Slight variations, if existent, will be noted explicitly, as well as reversals to other architectures.

4. Gaps in data

Training the CNN on an equally (and slightly) spaced set of tumbling rates might cause it to develop a certain logarithmic bias in establishing a tumbling rate. It is therefore useful to test how the CNN develops when trained on more and/or unevenly spaced tumbling rates, in order to examine any potential misdirections in data.

5. System size dependence

To gauge how the CNN can trace the underlying behaviour of *clustering* and detailed balance breaking as it relates to the tumbling rate, it may be useful to examine how a CNN trained on a specific system size will fare in predicting tumbling rates for different system sizes.

3. Discussing Similar Research

Deep learning probability flows and entropy production rates in active matter, Boffi and Vanden-Eijnden, 2023.

4. MN_3 Discussion

Below is the CNN architecture we have settled on. As a reminder, all the architectures used in this project can be found here.

1. CONV (filters=3,kernel_size=(3,3),padding='same',input_shape=shape)
2. MAXPOOL (pool_size=(2,2),padding='same')
3. ReLU
4. BN
5. CONV (filters=4,kernel_size=(4,4),padding='same')
6. MAXPOOL (pool_size=(2,2),padding='same')
7. ReLU
8. BN
9. CONV (filters=6, kernel_size=(5,5),padding='same')
10. MAXPOOL (pool_size=(2,2),padding='same')
11. ReLU
12. BN
13. AVGPOOL
14. DO (0.1) (without layout optimiser)
15. FC (units=128,activation='relu')
16. DO (0.1) (without layout optimiser)
17. FC (units=3,activation='relu')
18. FLATTEN
19. FC (units=1,activation='linear')

We have already briefly gone over what each architecture function type does in Weeks 13-15. However, a more general justification is outlined here. The activation function is kept as ReLU althroughout (with the exception of the last function); it is overall both better at accommodating high weights in the kernel functions, as well as much less resource intensive to generate when compared to the sigma function.

Convolutional layers are our main method of extrapolating features from input data. It is generally good practice to structure architecture such that the first stages employ a small kernel, which is gradually increased with subsequent convolutions. This approach is meant to leverage a fundamental prospect of neural networks: they begin by combing and extrapolating low-level features, which in subsequent layers are meant to form into more general, 'zoomed out' features. A good analogy would be drawn number recognition (borrowed from here: the first few stages of a neural network may identify pixels or curved lines, while the latter stages might pick up on entire shapes and ultimately numbers themselves. In a similar vein, our CNN may identify individual paricles and small clustering before it identifies large clusters, and subsequently picks up on underlying tumbling rate. Therefore, the kernel sizes for our CONV layers go from (3,3) to (5,5) in increments of (1,1). The CONV layer depth (its number of filters) is similarly increasing; the first layers pick up on low-level features, which then are extrapolated to form more complex patterns in latter layers.

Each convolutional layer is followed by a maximum pooling layer. These reduce dimensionality and keep the maximal value in the subregions binned. This is a method of down-sampling, increasing the generality of the feature map in order to circumvent very slight variations significantly altering the neural network training; the pool size is almost always (2,2) in literature, due to further increases resulting in too steep a reduction of output neurons. This is then followed by explicit ReLU activations, and finally we use batch normalisation layers in order to 'standardise' the output of previous layers to be used as the input for subsequent layers. Such layers involve variance scaling and mean centering in order to circumvent co-variate shifts caused by the normalised input becoming much smaller or bigger throughout the layers. Batch normalisation layers also help prevent overfitting.

Finally, after three cycles of CONV->MAXPOOL->ReLU->BN, an average pooling layer further downsizes the system. We use two dropout layers to prevent overfitting. note their placement *after* pooling layers. Dropout layers aid in getting rid of high dependency on small sets of features (by nullifying a proportion of neurons), but a pooling layer negates some of this by synthesising areas of neurons and converging them into less neurons. Maximum pooling is much more harmful, due to it effectively invalidating any nulled neurons it 'catches' in its pool and simply picking the highest value, but average pooling also risks minimising the impact of null neurons by taking the average of its pool.

These dropout layers are interspersed with fully connected (dense) layers, which leverage all input neurons as they decrease output neurons (note the sharp decrease of 'units'). We are essentially attempting to arrive at a single tumbling rate value at the end of the architecture, and therefore will require a single unit for the final FC layer. This is, again, a movement into deeper features by increasing the generality of our layers. the flattening layer that precedes the final FC is there to convert out feature maps to one dimension for final output.

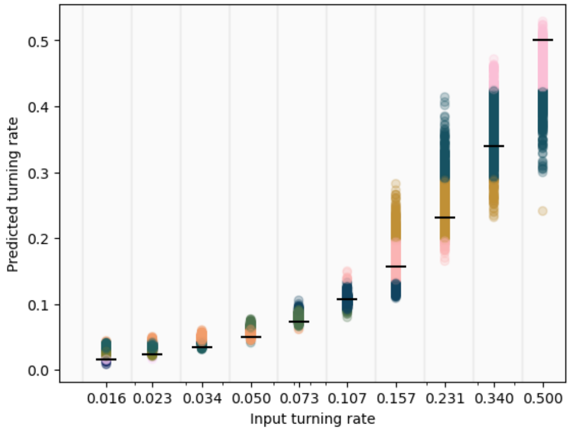
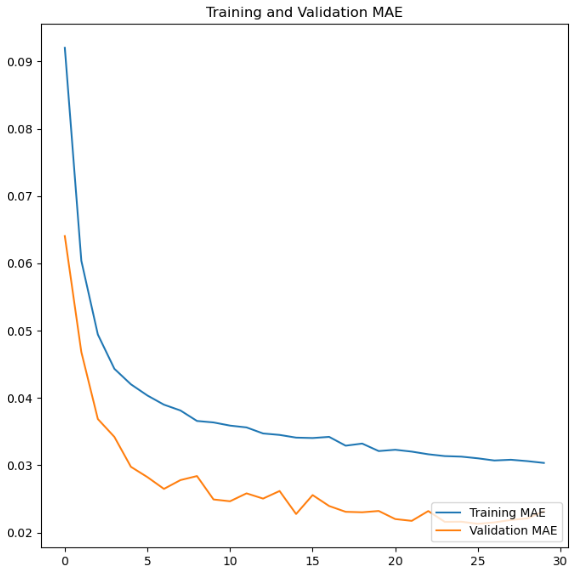
We have used 'same' padding due to the borders of the image being overall important in our case.

The general ballpark for hyperparameters, a broad perspective for architecture motivations, as well as some explanations has been taken from brief surveys of CNN literature. See, for example: 1, 2, 3, 4.

For our standard case of $N_x = N_y = 128$, the number of tuneable parameters in MN_3 is 2171.

5. Example MN_3 Application

Below is the model stage4124. It is trained on our full logspace of P_{tumble} values, and only on density $\rho = 0.15$, with 40000 total snapshots and a validation ratio of 0.2 (which is to say, 0.2*40000=8000 is the validation pool). Training was done for 30 epochs.

Predictions	Loss Evolution
	

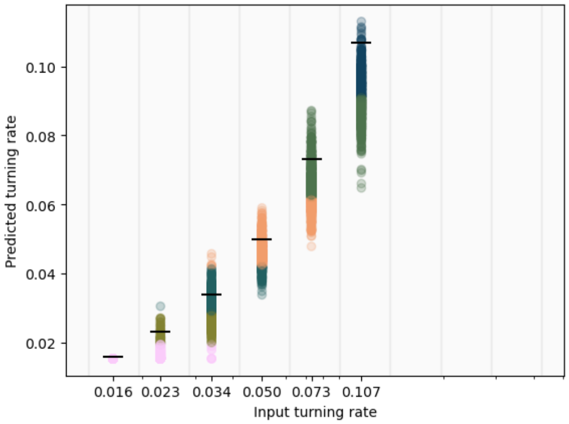
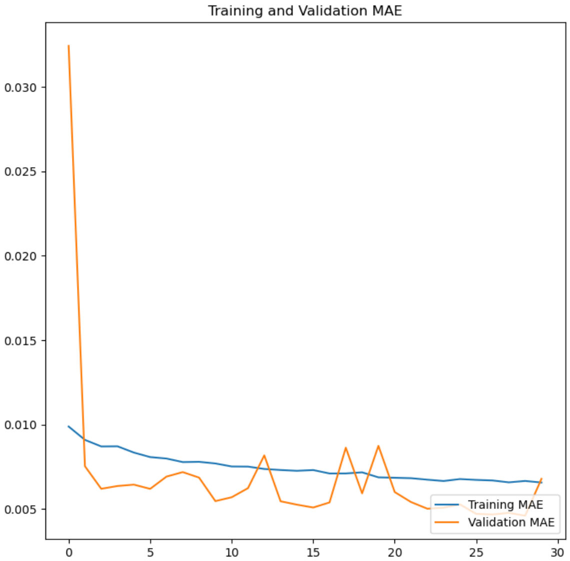
Some parameters are within acceptable margins as outlined in Week 16. Intuitively from the graph, the deviations from our margins relate specifically to the maximum spread (as can be seen in the higher tumbling rate values) and the Pearson's coefficient.

This seems to be a persisting problem for the neural network. As we will see in the section below, lower values of the turning rate tend to have less spread, whereas higher values of the turning rate tend to have more spread.

6. Gaps in P_{tumble} (monochrome)

As mentioned at the beginning of this week's log, an exploration of gaps in tumbling rate is essential for understanding the potential shortcomings of architecture MN_3. All models below are trained on **4x rolling**, with 1000 snapshots per (P_t, ρ) combination and only $\rho = 0.15$. All models are trained for **30 epochs**.

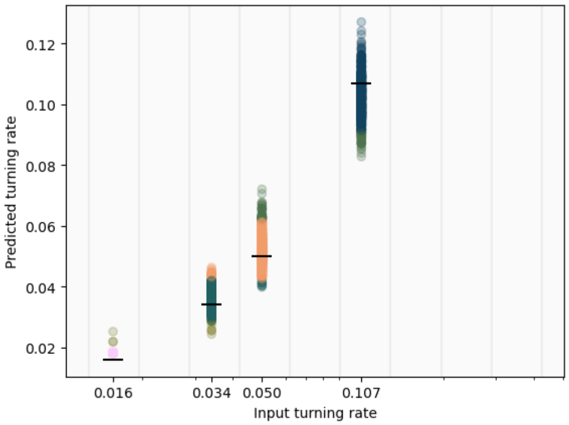
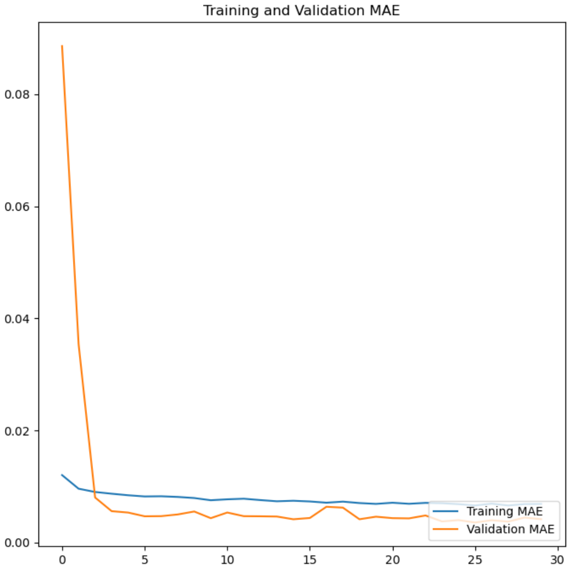
balteus3123: $P_t \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107\}$

Predictions	Loss Evolution
	

All parameters are within margins.

- ☒ MAE: 0.00679200328886509
- ☒ Min STD: 0.0000000018626451
- ☒ Avg STD: 0.00386919
- ☒ Max STD: 0.006857624
- ☒ Overlap Ratio: 1.0
- ☒ Pearson Coefficient: 0.98338868291357

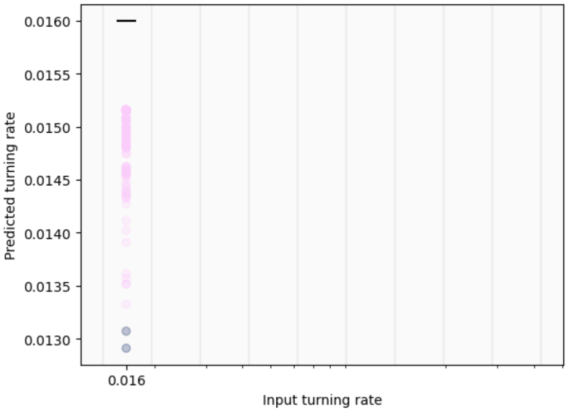
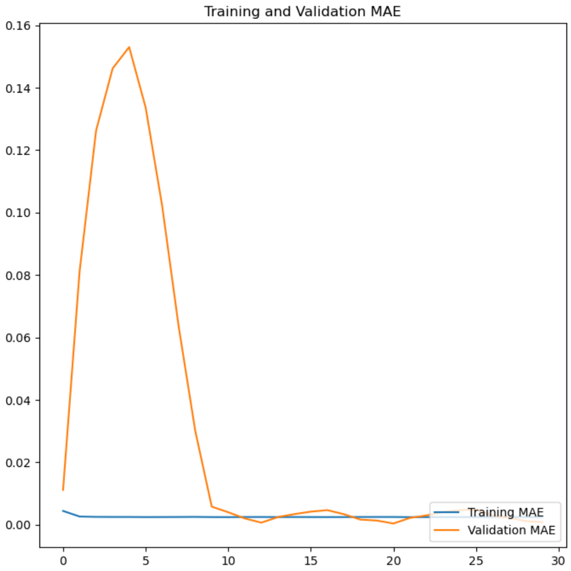
goose4421: $P_t \in \{0.016, 0.034, 0.050, 0.107\}$

Predictions	Loss Evolution
	

Most parameters are within margins, but the prediction misses the first point.

☒ MAE: 0.00417405366897583
☒ Min STD: 0.00031636294
☒ Avg STD:0.0038618112
☒ Max STD: 0.006754256
☐ Overlap ratio: 0.75
☒ Pearson Coefficient: 0.9883958630333 26

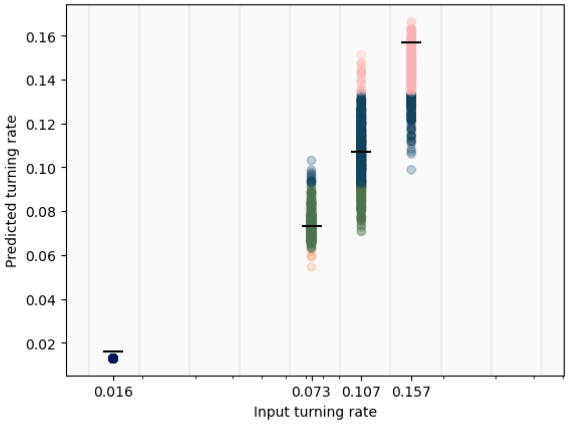
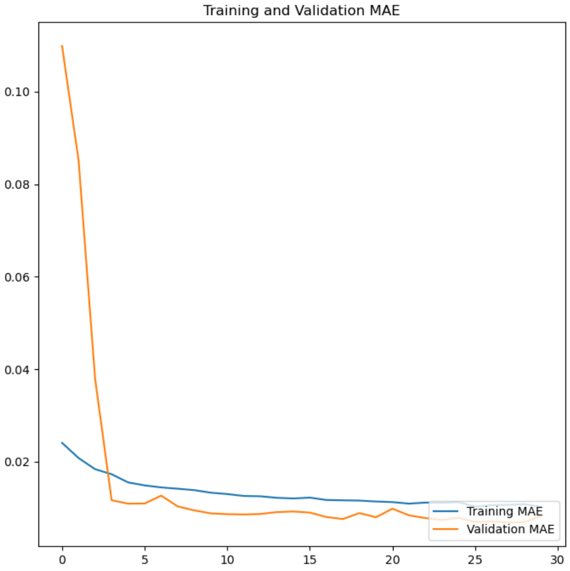
lyrical2734: $P_t \in \{0.016\}$

Predictions	Loss Evolution
	

All parameters are within margins. It's hard to gleam this from the graph, since all the prediction points seem to undershoot, but they undershoot by very little. Zoomed out to the magnitudes we usually see spread in (10^{-1} or 10^{-2}) they would look perfectly centred. Note that the Pearson coefficient is not available, since we only have one data set.

- ☒ MAE: 0.000894570257514715
- ☒ Min STD: 0.00021818299
- ☒ Avg STD: 0.00021818299
- ☒ Max STD: 0.00021818299
- ☒ Overlap ratio: 1.0
- ☒ Pearson Coefficient: nan

book1634: $P_t \in \{0.016, 0.073, 0.107, 0.157\}$

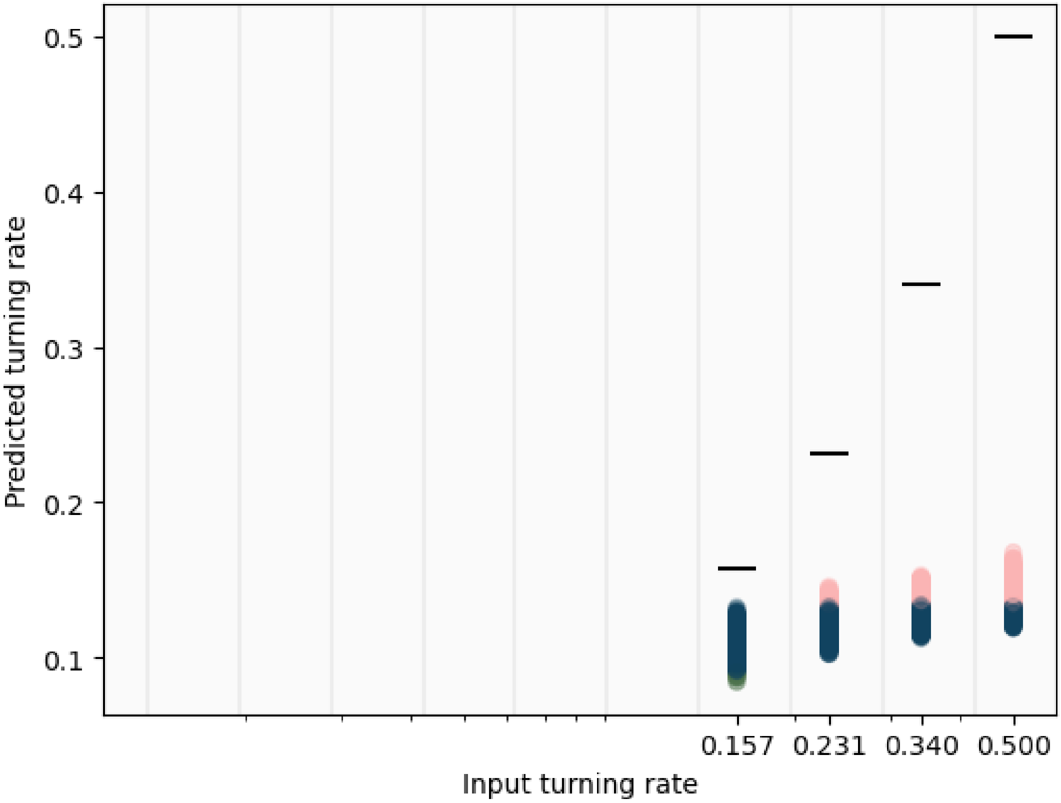
Predictions	Loss Evolution
	

Parameters

- ☒ MAE: 0.00829896610230207
- ☒ Min STD: 0.0000000037252903
- ☒ Avg STD: 0.007414392
- ☐ Max STD: 0.013894851
- ☐ Overlap ratio: 0.75
- ☒ Pearson Coefficient: 0.978325479793127

7. Preliminary Tumbling Rate Extrapolation

Model balteus3123 yielded promising data when validated on the same parameters it was trained in. By also validating it on the rest of the tumbling rate logspace, we may be able to obtain some useful information about how the neural network learns from data. We expect it to have relatively close tumbling rate predictions for those closest to the logarithmic scale it was given (so, namely, on 0.157, as it is the next value after the trained ones), but that it will break up fairly quickly when trying to predict bigger turning rates.



Parameters

- [] MAE:
- [] Min STD:
- [] Avg STD:
- [] Max STD:
- [] Overlap ratio:
- [] Pearson Coefficient:

MONOCHROME if img $>$ 0 img $=$ 1