

Dissipation Learning in Active Matter

Table of contents

1	Home	11
1.1	Information	11
2	Week 1	12
3	0. Table of Contents	13
4	1. Goals	14
5	2. Communication/Recording Method	15
6	3. Information Gathering	16
7	Week 2	20
8	0. Table of Contents	21
9	1. Introduction	22
10	2. Deadline Schedule	23
11	3. Text information	24
12	4. Motivation Report	26
13	5. Langevin Equation	28
14	Week 3	29
15	0. Table of Contents	30
16	1. Goals	31
17	2. Configuring BlueCrystal4	32
18	3. Small Github Digression	33

19 4. Persistent Exclusion Process Code	34
19.0.1 $P_{tumble} = 0.0005$	34
19.0.2 $P_{tumble} = 0.001$	34
19.0.3 $P_{tumble} = 0.01$	34
19.0.4 $P_{tumble} = 0.1$	35
19.0.5 $P_{tumble} = 0.33$	35
20 5. Code Examination	36
21 Week 4	38
22 Table of Contents	39
23 1. Introduction	40
24 2. Motivation Report Feedback	41
25 3. Varying Sample Size and Tumble Probability	44
26 4. Miscellaneous and Unsorted Notes	46
27 Weeks 5-6	47
28 0. Table of Contents	48
29 1. Introduction	49
30 2. Persistent Exclusion Process Visualisation	50
31 3. Utility Functions	52
32 4. Total Orientation Against Time	54
32.0.1 $\rho = 0.3; P_{tumble} = 0.05$	54
32.0.2 $\rho = 0.3; P_{tumble} = 0.2$	54
33 5. Cluster Analysis	56
34 6. Updated Motivational Report	59
35 Week 7	62
36 0. Table of Contents	63
37 1. Introduction	64
38 2. Potential Troubles with Tumbling	65

39	3. Revisions for Last Week	66
40	4. Interim Report Research	67
41	5. Interim Report Progress	68
42	Week 8	69
43	0. Table of Contents	70
44	1. Introduction	71
45	2. Project Plan	72
46	3. Tumbling Rate Clarifications	74
47	4. Dissipation and Structure	75
48	6. Bin Count Grid	76
49	7. Interim Report	77
50	Weeks 10-13	78
51	0. Table of Contents	79
52	1. Introduction	80
53	2. Quantifying Percolation	81
54	Weeks 13-15	84
55	0. Table of Contents	85
56	1. Introduction	86
57	2. Cluster Convergence	87
58	3. Machine Learning	88
59	4. Convolutional Neural Networks (CNNs)	90
60	5. Dataset Rolling	92
61	6. Preliminary CNN Experiment	93
62	7. CNN Layers	96

63 8. Preliminary CNN Tests on Rolled Data	97
64 9. Training and Predicting on Mixed Densities	98
65 10. Current Problems	99
66 11. Misc Notes	100
67 Week 16	101
68 0. Table of Contents	102
69 1. Introduction	103
70 2. Checklist	104
71 3. Model List and Storage	105
72 4. Shorthand Model List	107
72.0.1 MN_1	107
72.0.2 MN_2	107
73 3. New Architecture	109
74 4. More Tumbling Rates	110
75 4. Bigger N Values	111
76 5. Model Summaries	115
77 Week 17	116
78 0. Table of Contents	117
79 1. Introduction	118
80 2. Story	119
81 3. Discussing Similar Research	122
82 4. MN_3 Discussion	123
83 5. Example MN_3 Application	125
84 6. Gaps in P_{tumble} (monochrome)	126
84.0.1 balteus3123: $P_t \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107\}$	126
84.0.2 goose4421: $P_t \in \{0.016, 0.034, 0.050, 0.107\}$	127

84.0.3 lyrical2734: $P_t \in \{0.016\}$	127
84.0.4 book1634: $P_t \in \{0.016, 0.073, 0.107, 0.157\}$	128
85 7. Preliminary Tumbling Rate Extrapolation	130
86 Week 18	132
87 0. Table of Contents	133
88 1. Introduction	134
89 2. Refining CNN Architecture	135
90 3. Degrees of Freedom Discussion	136
91 4. Density Discussion in Big Tumbling Rate Spread	137
92 5. Gaps in Lower Tumbling Rates for Higher Densities	141
93 6. Gaps in Higher Tumbling Rates for Higher Densities	146
94 7. Omitting Highest Tumbling Rate	149
95 8. Different Density Comparison (Omitting Highest Tumbling Rate)	152
95.0.1 Comparison	154
96 9. Multiple Nearby Densities	155
97 10. Epoch Numbers	157
97.0.1 Comparison	158
98 11. Monochrome Interpolation (Low Tumbling Rates)	159
99 12. Monochrome Interpolation (High Tumbling Rates)	161
100 13. Monochrome Extrapolation	164
101 Week 19	167
102 0. Table of Contents	168
103 1. Introduction	169
104 2. Analysis Update	170
105 3. Higher Number Interpolation	172

1064. Preliminary Confusion Code	174
1075. Improved Full Tumbling Rate Analysis (more rolling, more epochs)	176
1086. Running Same Parameters Multiple Times	177
109Weeks 20-22	182
1100. Table of Contents	183
1111. Introduction	184
1122. Cluster Periodicity	185
112.0.1 Import Libraries	185
112.0.2 Set Analysis Conditions	185
112.0.3 Run Analysis and Plot	186
112.0.4 Array of Cluster Analysis	187
1133. Cluster Number and Size Examination	191
113.1 Import Libraries	191
113.2 Get Biggest Cluster + Cluster Count Details	191
113.3 Create Dataframe Storage	192
113.4 Plot Results	192
1144. Percolation Analysis	195
114.0.1 Import Libraries	195
114.0.2 Define Auxiliary Functions	195
114.0.3 Establish Percolation Values	196
114.0.4 Read Percolations	197
114.0.5 Plot Results	198
1155. Cluster Orientation Analysis	200
115.1 Import Data	200
115.2 Define Functions	200
115.3 Plot Cluster Map	202
115.4 Get Frequency of Up-Down Edge Orientations	203
115.5 Get Frequency of Left-Right Edge Orientations	204
115.6 Get Frequency of Up-Down Scrambled Edge Orientations	205
115.7 Get Frequency of Left-Right Scrambled Edge Orientations	206
115.8 Plot Cluster Edge Orientations	206
115.9 Plot Waffle Plot Distribution For Horizontal Filter	209
115.10 Frequency Comparison	213

1166. Feature Map Analysis (Orientation Data Type)	216
116.1Import Libraries	216
116.2Load Datasets	217
116.3Example Datamaps ($P_t \in \{0.050, 0.157\}$, $\rho = 0.25$)	217
116.4Set Up GPU and Load Model	217
116.5Plot Feature Maps	220
116.5.11. CONV (filters=3,kernel_size=(3,3),padding='same',input_shape=shape)	222
116.5.22. MAXPOOL (pool_size=(2,2),padding='same')	226
116.5.33. ReLU	226
116.5.44. BN	226
116.5.55. CONV (filters=4,kernel_size=(5,5),padding='same')	227
116.5.66. MAXPOOL (pool_size=(2,2),padding='same')	227
116.5.77. ReLU	233
116.5.88. BN	233
116.5.99. CONV (filters=6, kernel_size=(5,5),padding='same')	233
116.5.100. MAXPOOL (pool_size=(2,2),padding='same')	241
116.5.111. ReLU	241
116.5.122. BN	241
116.5.133. AVGPOOL	241
116.5.144. DO (0.1) (without layout optimiser)	243
116.5.155. FC (units=128,activation='relu')	243
116.5.166. DO (0.1) (without layout optimiser)	243
116.5.177. FC (units=3,activation='relu')	243
116.5.188. FLATTEN	248
116.5.199. FC (units=1,activation='linear')	248
116.6Some notes	248
1177. Feature Map Analysis (Monochrome Data Type)	250
117.1Import Libraries	250
117.2Load Datasets	251
117.3Example Datamaps ($P_t \in \{0.050, 0.157\}$, $\rho = 0.25$)	251
117.4Set Up GPU and Load Model	251
117.5Plot Feature Maps	254
117.5.11. CONV (filters=3,kernel_size=(3,3),padding='same',input_shape=shape)	256
117.5.22. MAXPOOL (pool_size=(2,2),padding='same')	260
117.5.33. ReLU	260
117.5.44. BN	260
117.5.55. CONV (filters=4,kernel_size=(5,5),padding='same')	261
117.5.66. MAXPOOL (pool_size=(2,2),padding='same')	261
117.5.77. ReLU	267
117.5.88. BN	267
117.5.99. CONV (filters=6, kernel_size=(5,5),padding='same')	267
117.5.100. MAXPOOL (pool_size=(2,2),padding='same')	275

117.5.11.1. ReLU	275
117.5.12. BN	275
117.5.13. AVGPOOL	275
117.5.14. DO (0.1) (without layout optimiser)	277
117.5.15. FC (units=128,activation='relu')	277
117.5.16. DO (0.1) (without layout optimiser)	277
117.5.17. FC (units=3,activation='relu')	277
117.5.18. FLATTEN	282
117.5.19. FC (units=1,activation='linear')	282
117.6 Some notes	282
118. Final Averaged Predictions	284
118.1 Import packages	284
118.2 Set seed (optional)	285
118.3 Setup GPU	285
118.4 Define Functions	286
118.5 Import and prepare data	288
118.6 Predict multiple models	289
118.7 Combined plots	290
119. Interpolation	298
119.1 Interpolation Predictions on Same Data Types	298
119.2 Interpolation Predictions on Other Data Types	298
120. Extrapolation	300
120.1 Low-bound Extrapolation	300
120.1.1 Predictions on Same Data Types	300
120.1.2 Predictions on Different Data Types	302
120.2 High-bound Extrapolation	302
120.2.1 Predictions on Same Data Types	302
120.2.2 Predictions on Different Data Types	302
121 Motivational Report	304
122. Table of Contents	305
123. Introduction	306
124. Original Motivational Report (Week 2)	307
125. Supervisor Feedback (Week 3)	309
126. Responding to Supervisor Feedback Week 4	311

127	CNN Models	316
127.0.1	MN_1	316
127.0.2	MN_1.5	316
127.0.3	MN_2	317
127.0.4	MN_3	317
127.0.5	MN_3*	318
128	Preliminary CNN Training and Analysis	319
129	Import packages	320
130	Setup GPU	321
131	Import and prepare data	322
132	Setup and train our model	324
132.1	Setting up the model's architecture	325
132.2	Optimizer	327
132.3	Training and evaluation	327
132.4	Run the training	329
133	Analyse training results	331
133.0.1	Kernel analysis	332
134	Save model (if needed)	336

1 Home

1.1 Information

Welcome to the home page (or landing spot) for the Dissipative Learning in Active Matter project! This page links to a few useful pages for the project as a whole. For weekly lab notes, see the [activity log](#), also available on the sidebar.

Some generally useful links for day to day sharing and work:

- [Deadlines](#) : A quick list of deadlines for this project.
- [Repository information](#) : Info on how to set up the repository.
- [Commands Information](#) : Useful commands for navigating the repository and publishing pages.

2 Week 1

3 0. Table of Contents

- 0. Table of Contents
- 1. Goals
- 2. Communication/Recording Method
- 3. Information Gathering

4 1. Goals

The main goal of week 1 is to get an introductory grasp of the literature regarding active matter.

The secondary goal of week 1 is to set up a communication and recording method for sharing files and information easily throughout this project.

5 2. Communication/Recording Method

Communication is done through a github repository. Both myself and my lab partner have personal branches: cp and np respectively. I will try to stick to the convention of referencing my contributions in github commits and code comments as “cp”. We also have a shared branch, where we upload collective work. Per the demands of this assignment, all progress will be documented individually, despite the collaborative nature of the project. There may therefore be repeats between the activity log and various files in the shared branch; wherever possible, the shared folder will be referenced rather than copied (such as the storing of programs and data).

There is also the individual gh-pages branch, which hosts a github website which should contain both shared and individual work. This may be the cleanest method of accessing the activity log, though presumably an exported pdf will be required for examination. Nonetheless, I link (embedded) all the relevant pages below:

- [github repository](#) (note that this is private, so it requires an access invite)
- [website landing page](#)
 - [personal activity log](#)

There is also the matter of convention. We have set up weekly supervisor meetings on Thursday. As such, the way weeks are kept track of is slightly unconventional, purely for pragmatic reasons. Week 1, for instance, began on the first Thursday of the university year, and ended on the second Thursday of it. It might therefore seem strange to see major edits in the github repository for, say, the Week 2 activity log on Wednesday night in what would normally be called week 3. I will think of ways to make this easier to keep track of, but the system works for now. There will (hopefully) be an addendum to this paragraph if/when I clarify the record-keeping system.

6 3. Information Gathering

Taking information from the following articles:

- I. The 2020 motile active roadmap
 - a. Introduction
 - b. Active Brownian particles: from collective phenomena to fundamental physics
- II. Run-and-tumble dynamics in a crowded environment: Persistent exclusion process for swimmers

6.0.0.1 I. The 2020 motile active roadmap, Gompper et al.

6.0.0.1.1 a. Introduction, Gompper & Roland

Active matter is a class of nonequilibrium systems composed of a large number of autonomous agents

- persistently out of equilibrium (constituents continuously consume energy)
 - absence of equilibrium concepts
 - * detailed balance
 - * Gibbs ensemble & free energy
 - * time-reversal symmetry
- therefore theories must be constructed on:
 - symmetries:
 - * polar shape (regarding polarity of molecules)
 - * nematic shape (regarding molecules that are aligned loosely parallel)
 - * interactions of agents
 - conservation laws
 - dynamic rules

- examples are agent-based standard models
 - active Brownian particles
 - squirmers (complemented by continuum field theory)
- aim is creation of artificial active matter (synthetic micro/nanomachines)
 - look to biological active matter
 - * propulsion mechanisms (rotation, translation and periodic altering of shape)
 - cilia
 - flagella
 - * navigation strategies
 - chemotaxis: movement/orientation along chemical concentration gradient (toward or away from stimulus)
 - phototaxis: movement/orientation towards or away from light
 - (what is the scale lower limit of such behaviour?)
 - as such, suggested synthetic micro/nanomachines can utilise:
 - * phoresis
 - diffusiophoresis: motion of species A in response to concentration gradient in colloidal species B
 - thermophoresis: motion in mixture of particles along temperature gradient (tendency of light molecules to hot and heavy particles to cold)
- swarming: spontaneous self-organisation of active agents in large numbers -> emergent coordinated collective motion on various length scales
 - determined by
 - * agent shape
 - * steric interactions
 - * sensing
 - * fluctuations
 - * environmentally-mediated interactions
- novel phenomena:
 - * motility-induced phase separation

* active turbulence

6.0.0.1.2 b. Active Brownian particles: from collective phenomena to fundamental physics, Speck

Active matter makes use of “persistence of motion”; locally broken symmetry, rather than a global preferred direction.

Synthetic active matter employs particle shape, ultrasound, etc. to facilitate movement.

Janus particles as important experimental strategy of locomotion: two hemispheres with different surface properties. Example: **colloidal particles**

- solvent containing H₂O₂
 - coat one hemisphere in catalyst for H₂O₂
 - resulting local concentration leads to individual particle propulsion along symmetry axis
 - axis undergoes rotational diffusion due to fluctuation
- * single-particle trajectories with a *persistence length* analogous to polymers(?)

Active Brownian Particles (ABPs)

- persistent motion
- particle interactions
 - short range
 - typically repulsive
- particles aggregate into clusters (even without cohesive forces)
 - dynamic feedback between speed and density: motility-induced phase separation (MIPS)

6.0.0.2 II. Run-and-tumble dynamics in a crowded environment: Persistent exclusion process for swimmers, Soto & Golestanian

Bacteria biofilms constitute development into multicellular communities through aggregation; exhibit novel properties:

- differentiation
- delegation of function

Free bacteria, upon interacting with surfaces, adapt to the new conditions by adopting different motility modes

- by contrast, biofilms can **nucleate** when a number of bacteria settle down near a surface and become completely localised
 - **nucleation**: first step in formation of a new thermodynamic phase or structure via self-assembly/self-organisation within a substance or mixture

7 Week 2

8 0. Table of Contents

8.0.0.0.0.1 0. Table of Contents

8.0.0.0.0.2 1. Introduction

8.0.0.0.0.3 2. Deadline Schedule

8.0.0.0.0.4 3. Text Information

8.0.0.0.0.5 4. Motivation Report

9 1. Introduction

The goal of week 2 is to further familiarise ourselves with the literature. We are to write a quick 500-word referenced report detailing motivations for studying active matter. Notes for this week will therefore be slightly sparser, primarily sifting through the goals of various papers (their detailed processes are to be examined at a later date). Some of these tidbits of information will be taken from the texts explored in week 1; as such, overlap with week 1 information is not only possible, but very likely. Repetition was preferred over disorganised and uncatalogued information, as these notes serve both as an activity log (which must document repeats occasionally) and as a catalogue of information as it is found.

Another essential goal of weeks 1 and 2 was to figure out the general examination schedule of the project. This in turn would influence planning the project out more properly, in anticipation of the deadline for the interim report (other influences, naturally, are personal academic schedules; I personally anticipate being much busier during the first term, and therefore expect higher activity during the latter half of the project). This schedule was not given out to the 30cp version of the project page until very recently, and has therefore been appended in the activity log for this week.

Above is a table of contents. Below is a list of a few of the utilised texts, with some key information extracted. The 500-word referenced essay is given afterwards. At the very bottom is a very brief introduction to the Langevin equation. On an informational level, this is meant as a way to ease in into Brownian motion theory; on a technical level, this also gives a brief occasion to experiment with LaTex in markdown.

10 2. Deadline Schedule

- Week 1: Start of project
- Week 9: Optional interim report submitted. Formative only. Deadline is Thursday 23rd November @ 12.30pm.
- Week 19: Practical work (experimental or computational/theoretical) must finish by the end of the week. Friday 8th March.
- Weeks 20/21: Analysis of results obtained. Start write up of report. Results to be presented to supervisor and analysis discussed.
- Week 22: Final Report submission. Deadline is Thursday 18th April @ 12.30pm.
- Weeks 23/24: Final interviews with Assessors and supervisors. **Assessment:** The final assessment is worth 100% of the total marks available.

(This schedule will be added on the website as a separate tab as well.)

11 3. Text information

11.0.0.1 The Mechanics and Statistics of Active Matter

This is a 2010 review of (at the time) recent progress within the field. Its main role for my current purposes is to obtain references to older papers (and therefore to their motivations), while also providing a starting definition for active matter.

- active matter can be considered as a type of material
- take active matter as condensed matter in a **nonequilibrium regime**
 - each (autonomous/active) constituent takes direct energetic input
 - * energy input is therefore homogenously distributed in system
 - * compare to fluid motion, for instance: energy is not supplied to each individual particle, but rather is applied (e.g. kinetically) at the boundaries: this then causes particles to push others forward, but they don't all have direct access to energy
 - * slightly related, Ramaswamy argues in one of his [lectures](#) that this is the key distinction of active matter, phrased as **direct access to energy** (in the fluid example above, the bulk particles have **indirect** access to energy)
 - force-free: forces exerted between particle and fluid cancel
 - (self-propelled) motion is set by particle, not external fields

11.0.0.2 Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton, Kistler et al.

This is a 2004 paper experimentally confirming a solution to the convection-diffusion relation applied to rod-shaped particles (with Pt and Au segments) moving autonomously in hydrogen peroxide solutions.

- one of the big nanotechnology challenges is the conversion of stored chemical technology to motion
 - this is precisely what many biological active matter systems do

- studying this would yield useful artificial active matter systems

11.0.0.3 Active matter: quantifying the departure from equilibrium, Flenner & Szamel

This is a 2020 paper examining active matter systems as they are moved further away from thermodynamic equilibrium.

- the main motivational point here is that quantifying departure from equilibrium helps understand the difference between active matter and equilibrium systems, and thus can help chart generalised models of how they both work

12 4. Motivation Report

Active matter is, broadly, a subcategory of matter systems distinguished primarily by energy input homogenously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a force-free medium (for instance, the forces between particles and the fluid they move through cancel)[1]. The agents therefore have *direct* access to energy, and use it to autonomously propel and direct themselves (with different models and situations allowing for various degrees of freedom). The study of active matter is generally grounded in (but not limited to) observed behaviour of biological agents, as they are the primary (though not only) examples of active matter in nature.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the natural world. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2]. This is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3] (note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics under duress might deal holistically, rather than individually, with other human agents[4]). Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by medicine[5].

Outside of biology, active matter research serves to emulate, or otherwise learn from, naturally-occurring behaviours in order to analyse a potentially more general thermodynamic state. Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibria can be modified and generalised to non-equilibrium states, and how these generalisations hold as departure from equilibrium through various means is increased[6]. These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative,

ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[7].

The feature in active matter of converting stored and homogenously available energy, such as chemical potential, into mechanical work is also of great importance to the field: understanding how this can work and how to facilitate, among other things, long-term energy access across the active matter substance is a key pursuit of nanotechnology[8]. Statistical and computational models can lend insight into individual and collective dynamics, and in turn give way to new experimental designs of nano/micromechanical systems.

499 words.

12.0.0.1 References

1. Active matter: quantifying the departure from equilibrium. Flenner & Szamel
2. From Disorder to Order in Marching Locusts. Buhl et al. (2006)
3. Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts. Silverberg et al. (2013)
4. How simple rules determine pedestrian behavior and crowd disasters. Moussaid, Helbing & Theraulaz (2011)
5. Active matter at the interface between materials science and cell biology. Needleman & Zvonimir (2017)
6. Phase Separation and Multibody Effects in Three-Dimensional Active Brownian Particles. Turci & Wilding (2021)
7. Nano/Micromotors in Active Matte. Lv, Yank & Li (2022)
8. Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton et al. (2004)

13 5. Langevin Equation

The **Langevin equation** represents the partly-random particle movement of a particle within a fluid:

$$m \frac{dv}{dt} = -\lambda \mathbf{v} + \eta(t)$$

where \mathbf{v} is particle velocity, λ is the damping coefficient, m is the particle mass, η is the **noise term**.

The **noise term** indicates collisions of the given particle with other molecules within the fluid. It is determined by a Gaussian distribution, and for the Boltzmann constant k_B , temperature T and η_i being the i-th component of vector $\eta(t)$, it is described by the following correlation function:

$$\langle \eta_i(t) \eta_j(t') \rangle = 2\lambda k_B T \delta_{i,j} \delta(t - t')$$

This approximates that any given force (at time t) is uncorrelated with a force at any other time. The collision time with other molecules indicates this is not the case. However, within great collective motion this is broadly the case. The appearance of the damping coefficient λ in the correlation function is, *within an equilibrium system*, an expression of the **Einstein relation**.

14 Week 3

15 0. Table of Contents

- 0. Table of Contents
- 1. Goals
- 2. Configuring BlueCrystal4
- 3. Small Github Digression
- 4. Persistent Exclusion Process Code
- 5. Code Examination

16 1. Goals

The main aim of this week is to begin reproducing a simple active matter system in code. To this end, a suitable environment to work in must be found and utilised. For now we are primarily sticking to python.

We have gained access to the Bristol supercomputers BlueCrystal4 and BluePebble (I have access to the former, my lab partner has access to the latter). We will attempt to apply the scripts to the supercomputers.

In [Week 2](#) it was suggested that I would go more in depth in statistical analysis. This has been put aside for now, in the interest of getting a feel for how ABP simulations work in practice.

17 2. Configuring BlueCrystal4

I already have access to BlueCrystal4 (BC4) due to one of my other modules. For that particular module, I am programming in C/C++; as such, I have already configured (installed and loaded) modules compatible with it. These modules are generally installed on a user instance, and loaded using the following bash command: `module load <path>`. For example, for loading the necessary files for Intel's C/C++ compiler (icc), I use the following: `module load languages/intel/2020-u4`.

The issue that arises is that, at least initially, we have decided to work in Python. While I plan to attempt to translate (and speed up) our later codes in C, it is preferable to agree on a shared language while we get our bearings (and Python is easier, for all its faults). I therefore need use both C and python modules; hopefully this will not cause conflicts as long as they are not loaded simultaneously. The loading can easily be done in the `.bashrc` script on the user home instance, which will keep the module loaded for any and all processes. This is inadvised in the BC4 user manual, however - like I said, conflicts can occur. It is much healthier and safer to instead load modules in the `.sh` script that will send a request to the job queue. This will be elaborated on later.

You load/add modules with `module load` or `module add`. I have used the former for C and the latter for Python, though strictly to differentiate the two better. Below is a quick list of the different module commands I use:

```
module avail #Lists available modules; can combine with grep to search for a specific one

module load languages/intel/2020-u4 #Loads icc for C/C++
module add languages/anaconda3/2022.11-3.9.13-tensorflow-2.11 #Loads anaconda tensorflow
module add languages/anaconda3/2020-3.8.5 #Loads full anaconda package
curl https://pyenv.run | bash #Loads python environment (non-anaconda alternative)

#NOTE: only one of these should be done for any instance!

module list #Lists loaded modules
```

Tested a quick “Hello, World!” program just to make sure that python runs correctly; it does, at least for now.

18 3. Small Github Digression

We have added a shared file regarding various commands for Github and Quarto. It can now be found [here](#), as well as in the [landing page](#), currently under the name ‘Commands Information’.

19 4. Persistent Exclusion Process Code

The code models particles moving across various lattice sites, with a ‘tumble’ probability. This is the probability that the particle will change its direction at every iteration (or time ‘tick’). This means that the tumble variable is inversely proportional to the **persistence length** of the environment.

Note that this is a (relatively) simple algorithm: every particle has the same chance to change direction (in a more accurate model they would more likely follow a probability distribution instead). Nonetheless, it’s a good start for getting a feel for one of the core topics of this project: the way **persistence length influences collective behaviour**. In this case, the **clustering** phenomena is heavily affected.

Below are some GIF files with varying tumble speeds P_{tumble} applied to the model. Note that the animation runs at 6 frames per second, and computes 50 frames total.

19.0.1 $P_{tumble} = 0.0005$

(Placeholder text: .gif can obviously not be rendered in pdf form. Consult website for better layout.)

The persistence length is so small here that almost all particles form into clusters.

19.0.2 $P_{tumble} = 0.001$

(Placeholder text: .gif can obviously not be rendered in pdf form. Consult website for better layout.)

The clustering is still noticeable here, but there are many more free particles roaming already (with a persistence length twice as big as the previous one).

19.0.3 $P_{tumble} = 0.01$

(Placeholder text: .gif can obviously not be rendered in pdf form. Consult website for better layout.)

Clusters are far less frequent, but still noticeable. This is a tenfold increase in persistence length compared to the previous case.

19.0.4 $P_{tumble} = 0.1$

(Placeholder text: .gif can obviously not be rendered in pdf form. Consult website for better layout.)

Barely any clusters form here, and when they do, it's only for a few frames.

19.0.5 $P_{tumble} = 0.33$

(Placeholder text: .gif can obviously not be rendered in pdf form. Consult website for better layout.)

There are no noticeable clusters.

Note that if $P_{tumble} = 1$, there is no autonomous activity within the system. In other words, particles exercise simple inertial movement in frictionless environment.

20 5. Code Examination

We have started looking at the code. It's hard to standardise everything we have talked about into notes format; most of it is commented on the [repository](#) (formalised as in-line comments and docstrings). As the status of this repository is unknown at this time (whether it will be included in the final project submissions, that is; it is, after all, an example supplied by the supervisor, which we are analysing to get a better understanding of the topic), I will summarise to the best of my ability. (Note that this will be vastly incomplete, as we have only discussed a few scripts, and only partially!).

20.0.0.0.1 lattice.py

- establishes a lattice class
 - call each instance that fits within a class an **object**
- establishes various attributes for the class
 - **Nsites**: the total amount of lattice sites within system
 - ‘**Nparticles**’: the total amount of particles moving around the system (with the stipulation that only one particle can fit in a lattice site at a given time)
 - **connectivity=4**: we are unsure what this is as of yet; operating under the assumption that this is the amount of neighbours a lattice site is connected to (and thus the amount of places a particle occupying a lattice site can jump to, depending on its orientation)
- **set_square_connectivity** subfunction
 - requires values of **Nx** and **(number of sites along x axis and number of sites along y axis) to rectangularly fit the number of total sites (**Nx*Ny==Nsites**)**
 - creates a ‘neighbor table’ using 2D numpy arrays, with each row being an individual lattice site and each column being an individual neighbour (along collectivity)
 - this is flattened into a one-dimensional array, ran through **construct_square_neighbor_table** (discussed below), then the result is obtained in both a one-dimensional (flattened) version and in a restored two-dimensional version

20.0.0.0.0.2 construct_square_neighbor_table

- takes the neighbor table from `lattice.py` as outlined above
 - the fastest way to store data in C is through a one dimensional array called with pointers, as it ensures that all the data is contiguously stored in the same memory address area
 - the indexing is of note (and may be useful later)
- * i,j lattice sites
- i goes through Nx positions (rows)
 - j goes through Ny positions (columns)
- * index: $i * Ny + j$

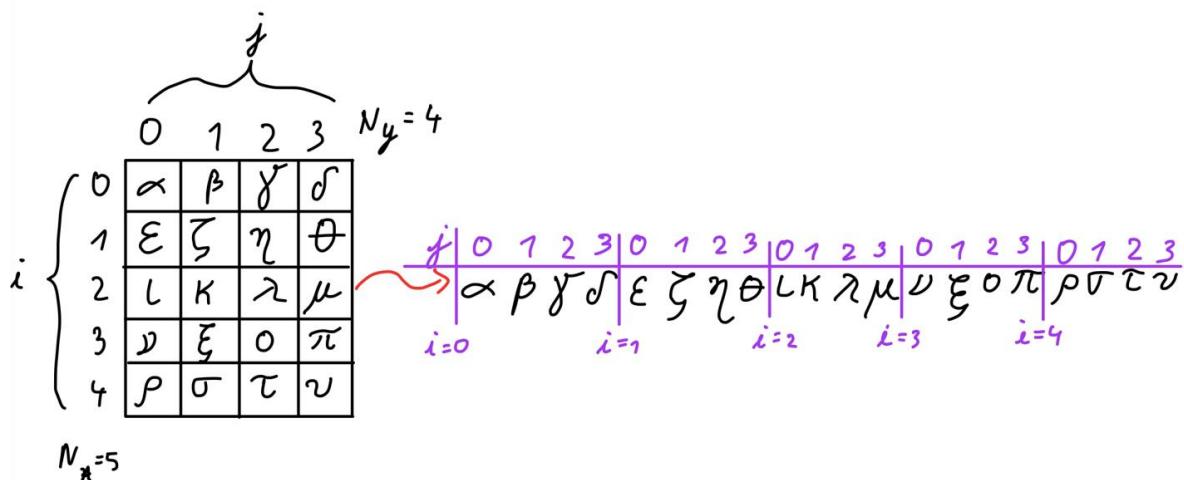


Figure 20.1: 2D to 1D array sketch example

21 Week 4

22 Table of Contents

- 0. Table of Contents
- 1. Introduction
- 2. Motivational Report Feedback
- 3. Varying Sample Size and Tumble Probability
- 4. Miscellaneous and Unsorted Notes

23 1. Introduction

The aim of this week is to consolidate the work of previous weeks and to build up qualitative understanding from it. The tasks for the following few weeks are:

1. Look over supervisor feedback for the motivational report from last week
2. Make a 5x10 configuration list of persistent exclusion process particle system from [Week 3](#), varying tumble probability and particle number.
3. Combine results with lab partner into a 10x10 grid (qualitatively showing how clustering changes as particle density and tumble probability are altered)
4. Plot total orientation against time for various cases
5. Look into clustering analysis

Only the first two points are expected to be done this week.

24 2. Motivation Report Feedback

I have created a separate website for monitoring development of the Motivational Report [here](#), also accessible through the sidebar. I will address every piece of feedback below.

My original context is presented, with the supervisor comments hyperlinked and prefaced by “FT”. Underneath I add my own comments in the form of bullet points. Only the commented parts of the report are shown.

Active matter is, broadly, a subcategory of matter systems FT “matter systems is unclear distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a force-free medium (for instance, the forces between particles and the fluid they move through cancel FT not exactly. Theres i no mechanical equilibrium. On the contrary, there is dissipation

- Here I was looking for a broad category to place active matter into; matter systems is indeed too vague. I would have been better off calling it a subcategory of soft matter systems.
- I don’t know exactly where I got the mechanical equilibrium confusion. I may have read some very specific thing that I generalised, but yes, dissipation ought to happen - one of the most important aspects of active matter is the requirement of supplying each autonomous agent with a steady energy supply which they steadily (or perhaps not so steadily in more complex models) use up.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the natural world FT be more precise: it is the world of living organisms, which constantly dissipate energy to perform their biological functions. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2]. This is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. FT: You are onto something here. Physicist Andrea Cavagna likes to say that “*Physics gauges the surprise in biology*” Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3] (note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics under duress might

deal holistically, rather than individually, with other human agents[4] FT not clear to me, please explain). Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by the medical sciences[5].

- I forgot that ‘natural world’ in English tends to refer more to general physical processes rather than specifically living organisms; I’ll try to be more specific regarding what active matter models help with understanding.
- From the brief look I managed to take at the literature, it seems that discussion of human behaviour in terms of physical systems is quite contentious. In hindsight, I should spend more than a sentence explaining this: the ‘cognitive heuristics’ argument for holism refers to the way humans deal with other humans in immediate crises. Many models will have an individual agent deal with other (in some way) adjacent agents individually; that is to say, it defines its relationship to each agent in turn, and then computes its behaviour. There are psychological arguments that this is not the case, and that instead humans might under duress conceptualise crowds (still) as a collective, and take actions in relation to the collective itself. At the time of writing this, it is unclear to me whether there are any active matter models that apply this ‘holistic’ method; the writers I cited, I believe, were criticising the models that do not attempt to do so. This is the case with the basic models I have engaged with so far (such as ABPs). It’s hard to imagine (though not impossible) how such a model can be implemented, but I don’t doubt that newer human-tracking physical models might work in this direction.
- Either way, I’ll look into Andrea Cavagna’s work. I’m interested in exploring this point more in detail.

Outside of biology, active matter research serves to emulate, or otherwise learn from naturally-occurring behaviours in order to analyse a potentially more general thermodynamic state FT “state” is not a good word. Are you thinking about a more general thermodynamic framework? . Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibrium can be modified and generalised to non-equilibrium states, and how these generalisations hold as departure from equilibrium through various means is increased[6] FT: not easy to read, but the idea is important: we can be just slight off equilibrium, and have a so-called linear-response regime, or we could be beyond linear response . These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[7].

- I take the point that ‘state’ is the wrong word; another loss in translation. I did mean a more general thermodynamic framework; thermodynamic ‘state’ implies thermal equilibrium, which is exactly what active matter does not have!
- I do get a bit long-winded here; I’ll try to rephrase this paragraph a bit and make sentences more readable

FT: You could get into more specifics, illustrating some examples of interesting behavior such as pattern formation or phase separation

- Yes, I’ll look into examples of pattern formation, as those tend to be quite demonstrative of what active matter study can do.

25 3. Varying Sample Size and Tumble Probability

There are two independent ways of generating graphical results. The first is to generate frames and animate them into a GIF file, which is done through the script `video.py` (see [Week 3](#)). The other way is what is explored this week: generate a data set with `sampler.py` and then use `view.py` to obtain snapshots.

The benefits of this are that datasets are stored for reference alongside the images, unlike the gif computation (which only stored the animation). This is useful for record keeping, as well as for understanding the way file storage is organised. The datasets are stored in h5py formats, a way to store huge amounts of data in a compressed manner - the downside is that the understanding process of how data is stored is less straightforward. So far, we have elected to keep a quite unoptimised version of the code for our purposes until we get a firmer grasp of the way the datasets work.

We have modified the code slightly to fit our purposes - made a few variables more explicit and standardised some parts. Furthermore, we've made the sampler vary with density and tumble probability - our aim is for each of us to generate 50 datasets, with 10 shared density values varied over 5 individual tumble probability values. The end goal, as stated in point 3 in the [Introduction](#), to combine these into a 100 dataset grid.

Each combination of data is stored in its own .h5 file. They are all indexed by a pandas dataframe, which keeps track of their ascribed particle density, tumble probability, particle speed and iteration count.

The particle density ρ_p is defined as a percentage, such that the total number of particles n_p is defined by:

$$n_p = \rho_p n_x n_y$$

where n_x and n_y are the dimensions of the lattice sites: the number of lattice sites along the x and y directions, respectively.

As defined in [Week 3](#), the tumble probability P_{tumble} is the probability at any given time cycle that a specific particle will change direction. } Below are some preliminary examples of varying the density $\rho_p \in [0.1, 0.3, 0.5]$ for a constant tumble probability of $P_{tumble} \cong 0.34$.

This browser does not support PDFs..

This browser does not support PDFs..

This browser does not support PDFs..

It's hard to see exactly how clustering forms at such a high particle density. As such, below are some more examples that vary the density $\rho \in [0.1, 0.2, 0.3]$, under the constant tumble probability of $P_{tumble} \cong 0.073$. The reasoning is that checking lower particle densities will help avoid noise that gets in the way of clustering, and checking lower tumble probabilities will allow more clustering to occur.

This browser does not support PDFs..

This browser does not support PDFs..

This browser does not support PDFs..

Clusters can be observed, especially in the $\rho_p = 0.2$ case. They are arguably also visible in the $\rho_p = 0.3$ case, although it's hard to disentangle clusters from one another - this will be very important later, once we start employing cluster analysis.

Some other things that were not mentioned last week. In all these diagrams and animations, colours indicate orientations. Clustering can therefore be seen by observing the orientation of exterior particles pushing them into an existing chunk, which is then consequently 'trapped'.

26 4. Miscellaneous and Unsorted Notes

There are a few overarching (long-term) goals to pursue right now.

26.0.0.1 0. Prepare a dataset for CNN

26.0.0.2 1. grid 2D ABP and bin the density

26.0.0.3 2. Explore PDEs

$$\begin{aligned}\rho_1 &= D_1 \nabla \rho_1 + f_1(\rho_1, \rho_2, \rho_3) \\ \rho_2 &= D_2 \nabla \rho_2 + f_2(\rho_1, \rho_2, \rho_3) \\ \rho_3 &= D_3 \nabla \rho_3 + f_3(\rho_1, \rho_2, \rho_3)\end{aligned}$$

Some other short-term ideas to explore in the following weeks:

- See radial distribution function (probability) - see for Leonard Jones Fluid
- Attempt a reconstruction of features of systems without breaking any symmetries.
- Plot umble rate against predicted tumble rate

27 Weeks 5-6

28 0. Table of Contents

- 0. Table of Contents
- 1. Introduction
- 2. Persistent Exclusion Process Visualisation
- 3. Utility Functions
- 4. Total Orientation Against Time
- 5. Cluster Analysis
- 6. Updated Motivational Report

29 1. Introduction

These two weeks broadly overlapped with reading week in the arts department; I used a significant portion of this time to catch up on other projects. As such, I have elected to combine these two weeks. This specific log is nonetheless expected to be shorter than the previous ones.

The goals for this week are as follows:

- obtaining a working graph of varying tumble probability and orientation (by arranging screenshots with various values next to each other)
 - mention a few issues in the code
- begin cluster analysis
- continue work on the motivational report

30 2. Persistent Exclusion Process Visualisation

It became apparent that the images obtained in [Week 4](#) using the `sampler.py` script to generate static images and the `view.py` script to visualise certain of these static images showed less clustering when compared to the animation in [Week 3](#). Upon closer analysis, the reasoning is due to the way orientation is set up. `lattice.py` (the script that sets up the lattice in which particles move) assigns orientation ‘0’ to the background, in order to assign it a color (black). It then assigns orientations to the moving particles, depending on orientation. However, the orientations assigned to particles are ‘0’, ‘1’, ‘2’, ‘3’. What this effectively does is give a quarter of the particles the same orientation as the background, thus rendering them invisible. This is a significant information loss.

Another issue is the way I have visualised static lattices in the past. In [Week 4](#) I have opted for downloadable .pdf files, which comes in handy for quick downloading in an unchangeable format (with no potential information loss due to poor quality), but is slightly harder to format. As such the titles are slightly cut off (something I did not notice until much later). I will try to fix it, or otherwise I will switch pack to .png formats.

Below is a general graph (before having fixed the orientation problem) showing how clustering varies with varying tumble probability.

123 124

This browser does not support PDFs..

125 126

For contrast, this is how a similar graph looks like after having fixed the orientation issues:

123 124

This browser does not support PDFs..

125 126

These are different samples generated for the same values of P_{tumble} and ρ . The contrast given by recovering the last 25% of the particles is very stark - some graphs that formerly displayed individual clusters now prove to have them be connected.

Some qualitative analysis:

- low densities do not display clustering for any P_{tumble} , as there are not enough particles in the environment to actually cluster
- once density is increased, clustering does occur for certain tumble probability values, as long as density is not too large
- when density ρ gets too large, clustering cannot be said to occur because particles occupy most of the simulation area (and in a sense, they form one big cluster which is trivial for our considerations; we're exploring how autonomous behaviour creates clustering, whereas this clustering is given simply by sheer quantity)
 - this can be counterbalanced by increasing P_{tumble} up to a point - for the selected ρ values this is (mostly) effective, but for a large enough ρ no amount of tumbling will preserve discrete and separate clustering

There is still the task of making a 10x10 grid which shows a more elaborate change in clustering across a wider range of values. This can get quite cumbersome and needs to be rendered in a large enough size for the quality to be preserved.

31 3. Utility Functions

Below is the script `utils.py` written by my lab partner, which features two utility functions which will come up occasionally in code. They unpack the h5py files in which the persistent exclusion process data is stored, and obtain unique iteration numbers or mean orientations.

```
"""
Utility functions
"""

import re
import h5py
import numpy as np

def get_ds_iters(key_list: list) -> list:
    """
    Get all the unique iteration numbers

    :param key_list: a list of all the possible dataset keys/names

    :returns: a list of unique iterations
    """
    iter_n = []
    for val in key_list:
        if re.search("^conf_\d+$", val):
            iter_n.append(int(val[5:]))
    return sorted(iter_n)

def get_mean_orientation(file) -> list:
    """
    Get the mean orientation at each iteration

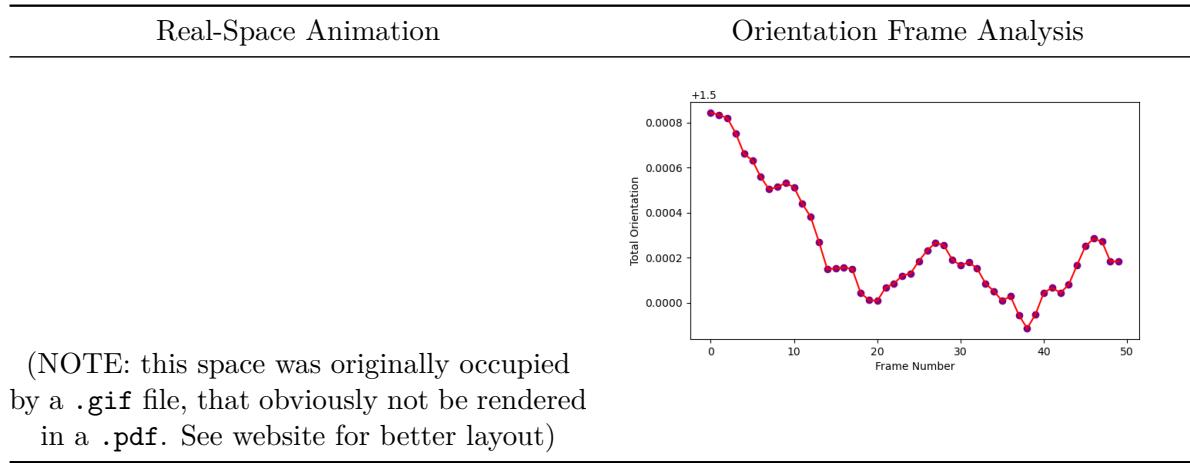
    :param file: the h5 file to open [str]
    :returns: mean orientation [list] of length 1000
```

```
Go through all iteration
"""
hf = h5py.File(file, "r")
key_list = list(hf.keys())
iter_n = get_ds_iters(key_list)
ori = []
ori_acm = []
for idx, val in enumerate(iter_n):
    sshot = np.array(hf[f"conf_{val}"]).flatten()
    avg_ori = np.average(sshot[np.where(sshot != 0)[0]] - 1)
    ori.append(avg_ori)
    ori_acm.append(np.mean(ori))
return ori_acm
```

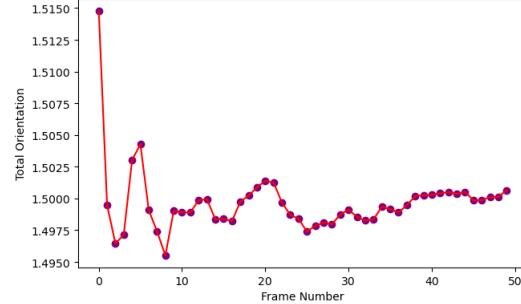
32 4. Total Orientation Against Time

The next thing to do is to show how the total orientation of the system differs with time. As a quick preliminary demonstration, we can make use of the `video.py` function and the total orientation it displays for every frame. Below are two demonstrations for the same particle density, $\rho = 0.3$

32.0.1 $\rho = 0.3; P_{tumble} = 0.05$



32.0.2 $\rho = 0.3; P_{tumble} = 0.2$



(NOTE: this space was originally occupied by a .gif file, that obviously not be rendered in a .pdf. See website for better layout)

The animations only run over 50 frames - this may be too small a sample size to gauge what's actually going on. Nonetheless, some basic traits can be inferred:

- for the $P_{tumble} = 0.05$ case, the persistent length is long enough that the system does not fundamentally change; fluctuations in total orientation are quite small, of the order 10^{-4}
- for the $P_{tumble} = 0.2$ case, the persistent length is shorter - this means that the initial state of the system is heavily chaotic, and begins changing rapidly as clusters form. We can observe a stabilisation effect, which slowly brings fluctuations to about the same order of magnitude as the one above
 - note that this is only qualitative analysis so far

33 5. Cluster Analysis

The process consists of splitting the particles using neighbour analysis. This is done with `scipy.ndimage`, using a kernel that only engages with vertical and horizontal neighbours (the '1' values) and disregards diagonal neighbours (the '0' values). The justification for this is that particles only have freedom of movement along vertical and horizontal directions; clustering as such emerges when inner particles are prohibited from moving by outer particles. If a particle is present on the diagonal, it cannot be said to contribute to the cluster; as such, it is disregarded.

The code below constitutes the main analysis done in this section. It leverages h5py file manipulation and uses a locally-created function (`utils.get_ds_iters`) for obtaining individual iterations from within it. This code is heavily borrowed from my lab partner's [version](#) (external link), which in turn makes use of our supervisor's [code](#).

```
from scipy import ndimage
import numpy as np
import matplotlib.pyplot as plt
import h5py
import utils
import cmcrameri #for different cmaps

Pt=0.157 #tumble probability
rho=0.3 #particle density
file = ("..../data/dataset_tumble_{}_{}.h5".format(Pt,rho)) #change this to analyse different files

cmap1 = plt.get_cmap(name="gnuplot",lut=5) #cmap for first picture
cmap2 = plt.get_cmap(name="cmc.tokyoS") #cmap for second picture
hfile = h5py.File(file,"r")

fig, (regplot, clusterplot, clusterhistogram) = plt.subplots (1,3,figsize=(9,3),width_ratios=[1,1,1])

iters = utils.get_ds_iters(hfile.keys())
fig.suptitle("Cluster Analysis (P={}; rho={})".format(Pt,rho))

#plot regular graph
image = hfile[f"conf_{iters[-1]}"]
```

```

regplot.matshow(image,cmap=cmap1)

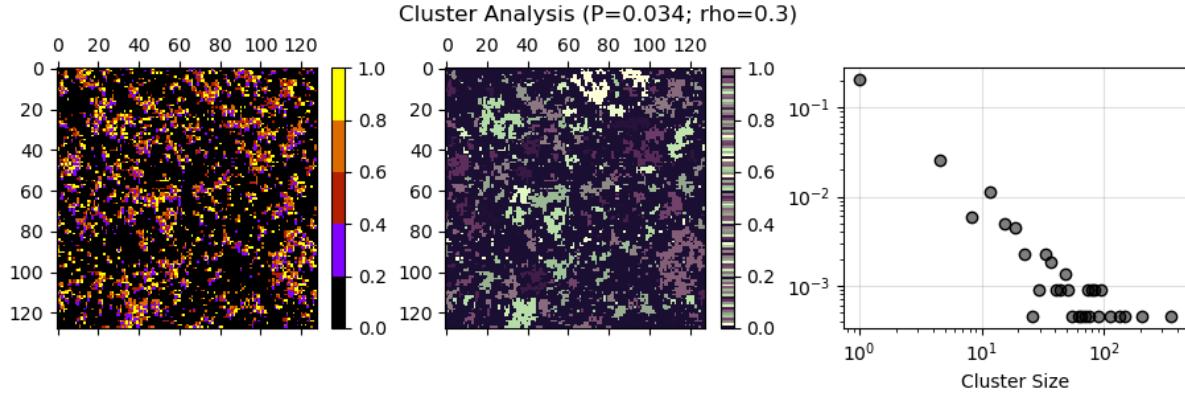
#plot cluster separation graph
kernel = [[0,1,0],
           [1,1,1],
           [0,1,0]]
labelled, nlabels = ndimage.label(image,structure=kernel)
clusterplot.matshow(labelled,cmap=cmap2)

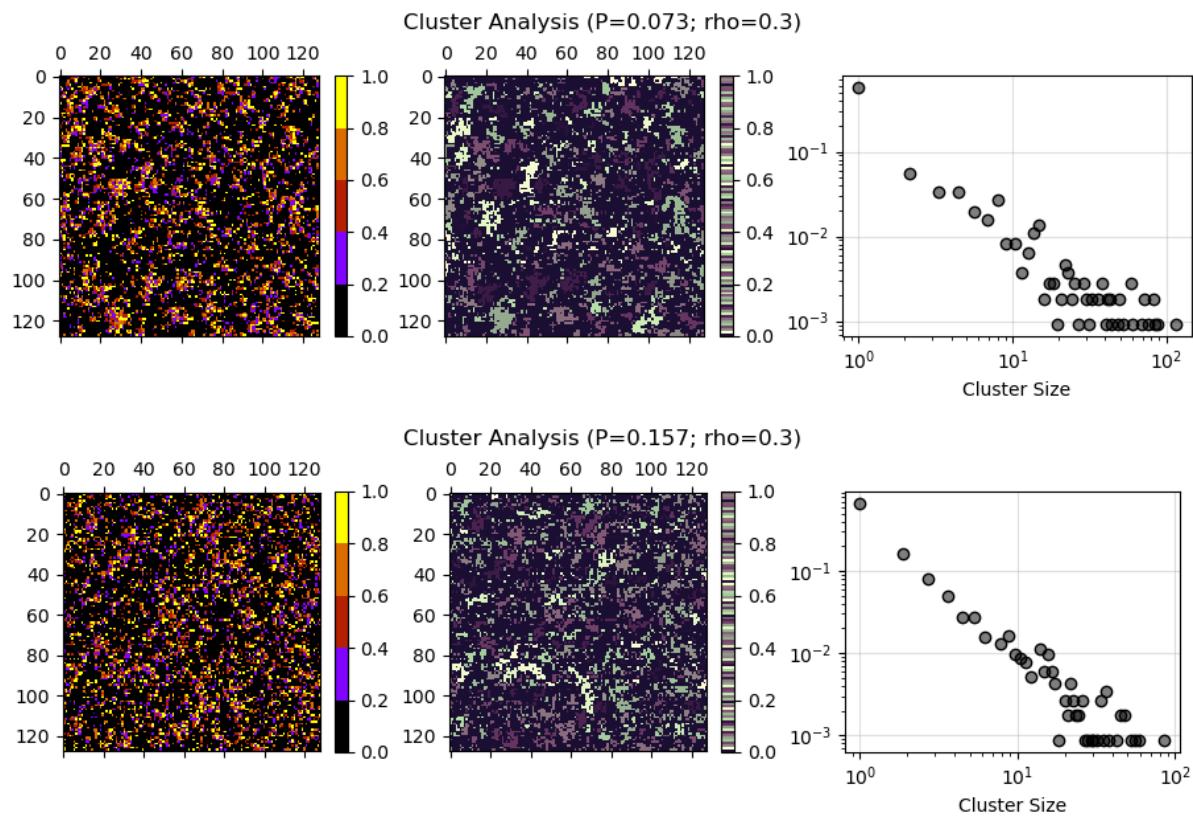
#plot histogram of obtained clusters
cluster_sizes = np.bincount(labelled.flatten())[1:]
bin_edges = np.linspace(cluster_sizes.min(),cluster_sizes.max(),100)
counts, _ = np.histogram(cluster_sizes,bins=bin_edges,density=True)
clusterhistogram.grid(alpha=.4)
clusterhistogram.set_axisbelow(True)
clusterhistogram.scatter(bin_edges[:-1],counts,edgecolor=(0,0,0,1),facecolor=(0,0,0,.5))
clusterhistogram.set_yscale("log"), clusterhistogram.set_xscale("log")
clusterhistogram.set_xlabel("Cluster Size")

fig.colorbar(plt.cm.ScalarMappable(cmap=cmap1),ax=regplot)
fig.colorbar(plt.cm.ScalarMappable(cmap=cmap2),ax=clusterplot)
plt.show()

```

Below are a few results with varying P_{tumble} and a fixed $\rho = 0.3$





34 6. Updated Motivational Report

34.0.0.0.1 For full evolution history of the report, see the [Motivational Report](#) page.

Active matter is, broadly, a subcategory of condensed matter systems distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a medium. The agents therefore have *direct* access to energy, and use it to autonomously propel and direct themselves (with different models and situations allowing for various degrees of freedom). The study of active matter is generally grounded in (but not limited to) observed behaviour of biological agents, as they are the primary (though not only) examples of active matter in nature.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the world of living organisms, where energy is constantly dissipated in order to perform various biological functions. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2].

This biological emulation through physical models is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3]. Note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics (the autonomous individual behaviour of a human) under duress might deal holistically, rather than individually, with other human agents[4]. The issue is that most active matter systems only form individual relationships between agents, and do not account for the way an agent interacts with the group as a whole - the resulting individual behaviour is merely a summation of the agent’s response to each other agent around it. There are psychological arguments that this is not the case, and that instead humans might under duress conceptualise crowds as a collective, and take actions in relation to the collective itself. This objection rests on the assumption that this holistic heuristic does not *emerge* from individual relations, of course (in which case mapping relationships strictly between individuals is unproblematic).

These insights lead to the exploration of various models. For flocks of birds, individual cognitive heuristics tend to suffice - self-propelled particles with adaptive movement patterns based

on neighbours can accurately reproduce some migrational patterns [5]. Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by medicine[6]. Bacteria lends a great example for exploring the intertwining of phenomena to be emulated by active matter. Some strains (such as *Bacillus subtilis*) can be modelled using both direct physical interaction (between individuals) and long-distance biochemical signalling (within the collective), with complexity and clustering developing in response to harsh external conditions [7]. The latter interaction is called quorum sensing, the adaptation of the individual to local population density; this has developed into its own active matter branch of individual-to-collective behaviour [8]. Using such models, it is possible to recover the aforementioned human holistic cognitive heuristics [9].

Outside of biology, active matter research serves to emulate, or otherwise learn from, naturally-occurring behaviours in order to analyse a potentially more general thermodynamic framework. Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibria can be modified and generalised to non-equilibrium states. Exploring how these generalisations would hold as departure from equilibrium through various means is increased is then paramount[10]. These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[11].

The feature in active matter of converting stored and homogenously available energy, such as chemical potential, into mechanical work is also of great importance to the field: understanding how this can work and how to facilitate, among other things, long-term energy access across the active matter substance is a key pursuit of nanotechnology[12]. Statistical and computational models can lend insight into individual and collective dynamics, and in turn give way to new experimental designs of nano/micromechanical systems.

756 words.

34.0.0.1 References

1. Active matter: quantifying the departure from equilibrium. Flenner & Szamel
2. From Disorder to Order in Marching Locusts. Buhl et al. (2006)
3. Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts. Silverberg et al. (2013)

4. How simple rules determine pedestrian behavior and crowd disasters. Moussaid, Helbing & Theraulaz (2011)
5. Novel Type of Phase Transition in a System of Self-Driven Particles, Vicsek et al. (1995)
6. Active matter at the interface between materials science and cell biology. Needleman & Zvonimir (2017)
7. Formation of complex bacterial colonies via self-generated vortices, Czirok et al. (1996)
8. Self-organization of active particles by quorum sensing rules, Bäuerle et al. (2018)
9. Quorum sensing as a mechanism to harness the wisdom of the crowds, Moreno-Gámez et al. (2023)
10. Phase Separation and Multibody Effects in Three-Dimensional Active Brownian Particles. Turci & Wilding (2021)
11. Nano/Micromotors in Active Matte. Lv, Yank & Li (2022)
12. Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton et al. (2004)

35 Week 7

36 0. Table of Contents

- 0. Table of Contents
- 1. Introduction
- 2. Potential Troubles with Tumbling
- 3. Revisions for Last Week
- 4. Interim Report Research
- 5. Interim Report Progress

37 1. Introduction

The aim of this week is mostly to work on the upcoming interim report. As such, most of this week's activity constitutes in gathering data from various papers. A further potential problem in the code was also spotted with regards to how the tumbling rate is established. Beyond that, some minor revisions for the previous week were mentioned.

38 2. Potential Troubles with Tumbling

Examining the code a bit further, we noticed a peculiarity of the tumbling process - on the offchance that a particle does decide to tumble (the probability of which is P_{tumble} , it to picks one of the four orientations (0,1,2,3) with equal chance to switch its orientation to. This is an issue, in that the particle has a 25% chance to “tumble in place”. This means that P_{tumble} has been consistently off by a factor. The real tumbling rate is:

$$P_{tumble}^{actual} = \frac{3}{4} P_{tumble}$$

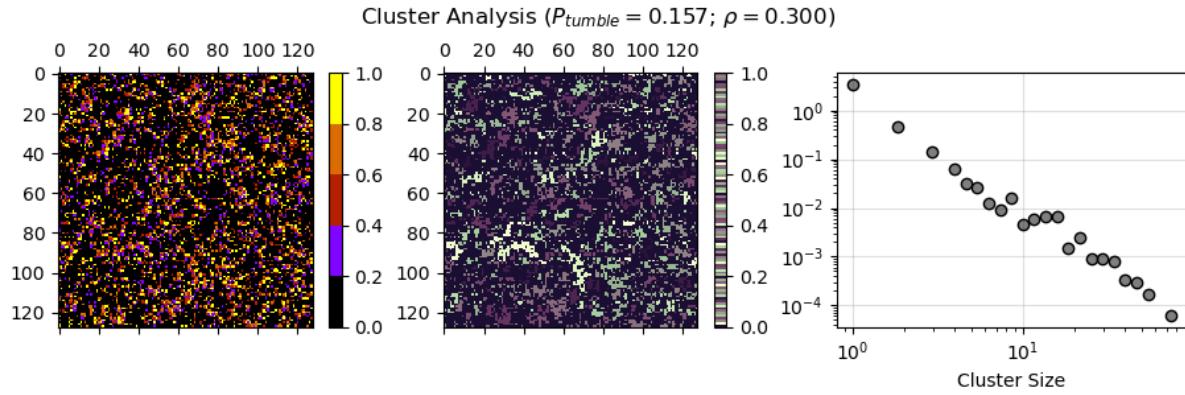
For now, we will operate with the flawed P_{tumble} . I will mention when this is fixed; for this week, at least, P_{tumble} will refer to the flawed tumbling rate, and P_{tumble}^{actual} will refer to the actual tumbling rate.

39 3. Revisions for Last Week

I misunderstood the effect of large densities within run-and-tumble models and what it means for clusters: it isn't that cluster analysis cannot be done due to every particle being joined in the same cluster. Rather, this is where percolation analysis becomes important - will look into it next week further.

There is also the matter of units be established: length can be measured in lattice sites, such as by establishing a lattice site convention as a .

This is technically not a revision, but I've updated the cluster analysis grouping graph from last week to also work in log scale. Here is an example, with the histogram bin fitted for log 2:



This is quite similar with the fitting done by Soto and Golestanian in their 2014 paper “Run-and-tumble dynamics in a crowded environment: Persistent exclusion process for swimmers”. More comparison and research is needed, though.

40 4. Interim Report Research

I've done a lot of research into the Persistent Exclusion Process, some of its underlying principles (such as the simple exclusion interaction, run and tumble dynamics and the zero range process), Active Brownian Particles (ABP), and the like. I would usually list a classificatory list with sorted information; however, due to illness and most of my project time being allotted to working on the interim report itself, I could not make the list this week.

I will instead make a small list of what I have to look into next week, in anticipation of the deadline:

- general convolutional neural network usage
- further active matter theories
 - active matter field theories
 - * Model B
 - * Model B+
 - * Model H, potentially
 - collision active matter theories
 - * Active Ornstein Uhlenbeck Particles (AOUP)
 - compare and contrast with ABPs
- correlation between dissipation and spatial patterns
 - how to study entropy through clusters
 - link structure with activity
- further research percolation

41 5. Interim Report Progress

123 124

This browser does not support PDFs..

125 126

Many things here are subject to change. This is a combination of the [Motivational Report](#) from before and the research we have done this week. There are a few places which have to be completed, and the Discussion section is particularly lackluster as of now. The report is currently 1695 words without references.

42 Week 8

43 0. Table of Contents

- 0. Table of Contents
- 1. Introduction
- 2. Project Plan
- 3. Tumbling Rate Clarifications
- 4. Dissipation and Structure
- 5. Further Information
- 6. Interim Report

44 1. Introduction

The main aim of this week is to finish the Interim Report. As this is not summative for us, the deadline has proven to not be strict - nonetheless I aim to finish it in time.

The secondary aim is to clarify various things that have come up throughout the past weeks, first and foremost the general plan of this project. The project plan is outlined below; we are currently still in the Characterisation phase. Percolation and pair-correlations are high priority in order to understand energy dissipation and to then transition into CNN construction.

45 2. Project Plan

These are some notes taken during open discussion between us and our supervisor.

1. Characterisation - set the landscape of physical properties by analysing the given Persistent Exclusion Process model
 - timescales
 - ρ - P_{tumble} diagram
 - cluster distribution
 - percolation
 - **pair-correlations** -> link to dissipation
2. Construction - make a minimal Convolved Neural Network (CNN) model; retrieve physical properties from data analysis
 - input details
 - ignore orientations (experiment-like): only use particle positions
 - include everything: particle orientations and positions
 - pull information from steady-state systems
 - validate the architecture
 - how do we choose architecture options?
 - * map size
 - * layer number
 - how much data? can we minimise it?
 - how does it extrapolate?
 - explainability
 - compare with feature maps
 - check with explicit ways of measuring dissipation

3. Extension

- use CNN on non-steady state systems
 - chart applicability on transition into steady state
- extend CNN use to off-lattice models

46 3. Tumbling Rate Clarifications

In Week 7, I mentioned some potential issues with P_{tumble} . This was due to a misunderstanding of the role this variable plays. To reiterate, P_{tumble} is the probability that a particle will tumble in any direction, **including** the one it is already going into. This means that P_{tumble} is related, but not synonymous, to the **rate of changing orientation**, because the former has a 25% chance to effectively ‘tumble in place’. This is not an oversight of the model, it is a characteristic of analogising to random thermal fluctuations. Departure from equilibrium is caused as P_{tumble} departs from $P_{tumble}^{max} = 1$, irrespective of the probability of actually *changing* direction.

This clarification is useful to keep in mind as we begin talking about energy dissipation.

47 4. Dissipation and Structure

There is a prominent link between the structure of an active matter system and energy dissipation. Here energy dissipation is understood as the breaking of detailed balance; we can follow the argument below to trace expression of energy dissipation in terms of structural characteristic P_{tumble} :

$$P_{tumble} \sim \frac{1}{\tau} \sim \frac{1}{P_e} \sim \frac{1}{v_{active}}$$

where τ is the persistence time, P_e is the Peclet number and v_{active} is active velocity; the latter can be rephrased as a force using a friction coefficient, thereby giving us a ‘negative friction’. Thus the link between dissipation and the tumbling rate is established: a decrease in tumbling rate is an increase in detailed balance breaking, which is an increase in dissipation.

There are specific relations we can use to infer dissipation; these will be analysed in subsequent weeks.

48 6. Bin Count Grid

Using the cluster analysis shown in [Weeks 5-6](#), we can obtain cluster distribution relations as we vary the tumbling rate and the density. For the following grid of particle distributions:

Could not load PDF..

we get the following cluster grid:

Could not load PDF..

49 7. Interim Report

This is an almost finished version of the interim report, after applying some preliminary supervisor feedback. The finalised version will be done next week. There are some points that should be elaborated regarding CNN functionality, MIPS, percolation, pair correlation function as well as slightly revising the Discussion.

Could not load PDF..

</embed>

50 Weeks 10-13

51 0. Table of Contents

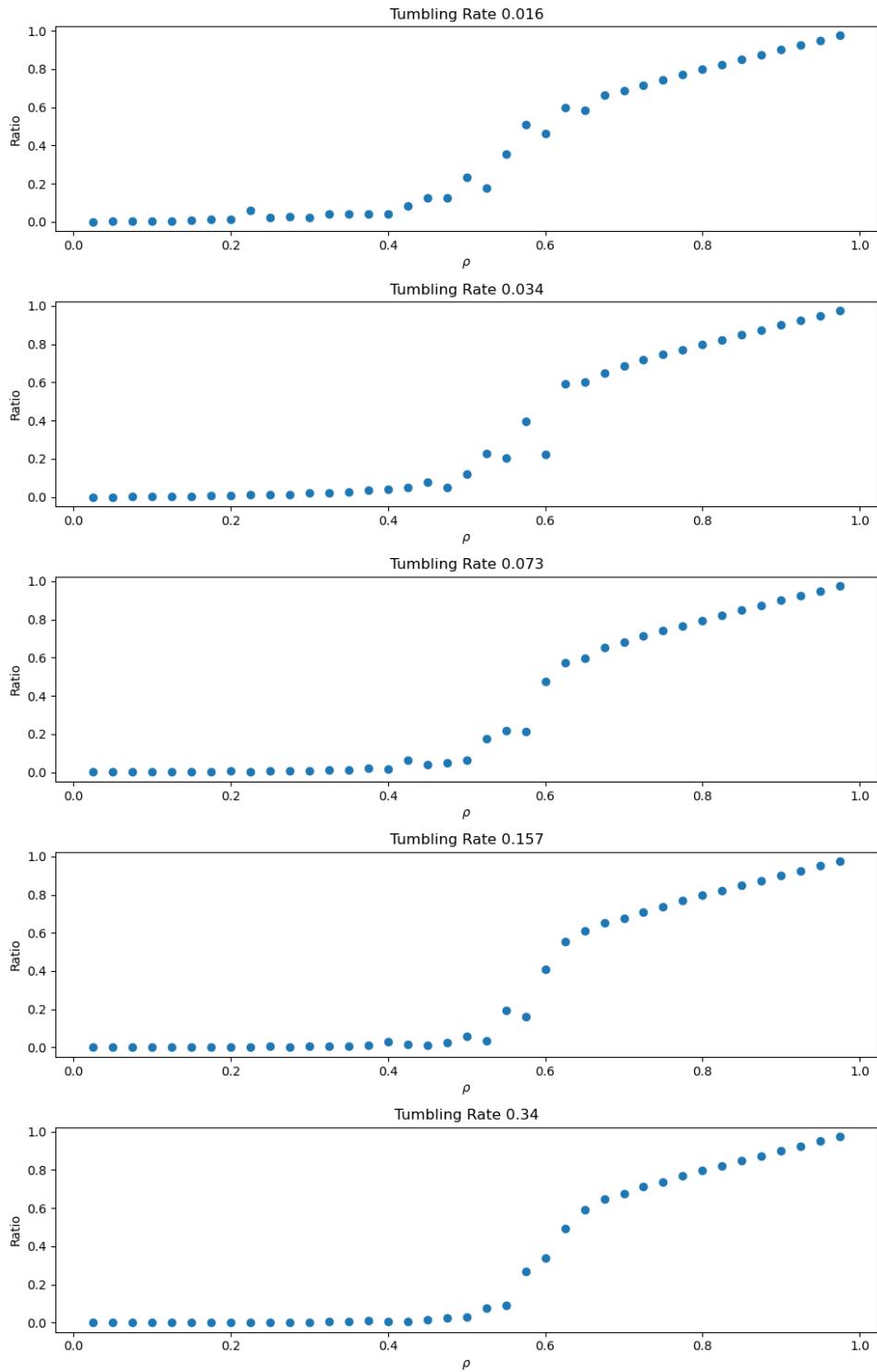
- 0. Table of Contents
- 1. Introduction
- 2. Quantifying Percolation
- 3. Cluster Convergence

52 1. Introduction

53 2. Quantifying Percolation

We can begin to quantify percolation by applying the cluster analysis outlined in [Weeks 5-6](#) (specifically subsection 5). By obtaining the size of the biggest cluster, we can see how it varies for different densities - percolation occurs at higher sizes relative to the entire system size (which as of now is still 128x128 lattice sites). The figure below shows such an analysis for different tumbling rates P_t .

Biggest Cluster Sizes Relative to System Size Against System Density for Different Tumbling Rates



We can see that the highest possible cluster stagnates around zero for lower densities in all graphs. This means that, while clustering does begin to occur, there are many unconnected and small clusters rather than one big cluster reaching across the entire system. Take the clustering analysis picture from Weeks 5-6 below, for instance; we notice obvious non-zero clusters, but they are all individually relatively small to the entire system. This is an evident non-percolation regime.

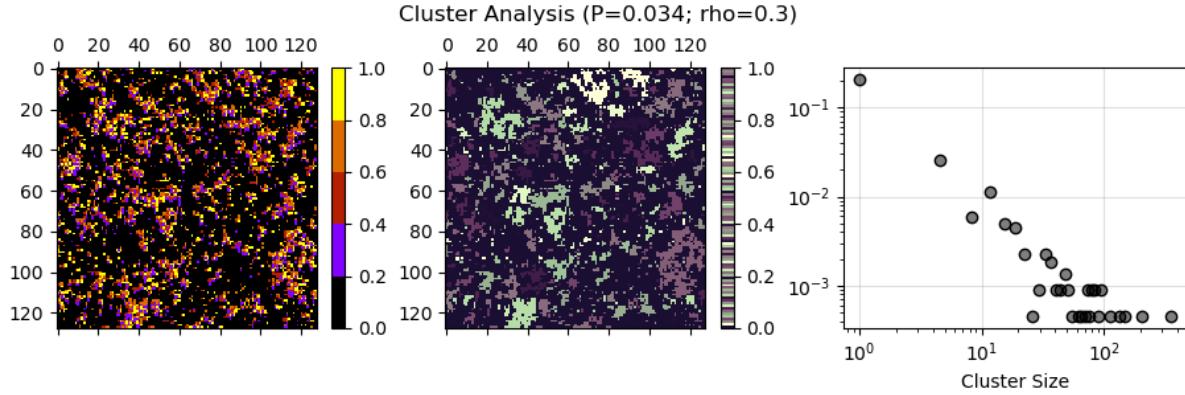


Figure 53.1: test

Where percolation does start being called into question is when the relative percentage of the cluster becomes comparable to the set density. Sticking with the $P_t = 0.034$ case, we can see that by $\rho = 0.8$, the gradient of the plotted curve becomes distinctly linear. This is precisely because the biggest cluster in the system covers the exact density thereof.

Note of course that the ratio need not exactly cover the density - there is an ambiguous regime where percolation feasibly occurs with most clustering being enmeshed within the same connected cluster, but simply with a lot of agents in constant movement. We can characterise this as the zone around the ‘inflection point’ within the above graphs.

Note, as well, the inflection region, which is more spread out for a smaller tumbling rate, and conversely is sharper for higher tumbling probabilities.

To further examine percolation,

54 Weeks 13-15

55 0. Table of Contents

1. Introduction
2. Cluster Convergence
3. Machine Learning
4. Convolutional Neural Networks (CNNs)
5. Dataset Rolling
6. Preliminary CNN Experiment
7. CNN Layers
8. Preliminary CNN Tests on Rolled Data
9. Training and Predicting on Mixed Densities
10. Current Problems
11. Misc Notes

56 1. Introduction

The main purpose of this week is to begin machine learning tests. The secondary purpose is to flesh out some details a

57 2. Cluster Convergence

Our cluster analysis code currently outputs various clusters organised by their sizes. In practice, we have analysed them by plotting their distributions within a snapshot using histograms (see, for example, [Weeks 5-6 Cluster Analysis](#)).

The initial thought was to maintain the bin distributions and chart how each individual bin varies over time. This would essentially involve a ‘rolling average’, where the bin values of iteration n would be compared with the average of the previous k iterations for *every bin*. This would mean a constant account of standard deviations for moving averages, and setting an acceptable threshold beyond which the system can be considered to stabilise. This is obviously complicated - firstly, the histogram bins will not always be constant; there are bins which will have zero clusters, or which will fluctuate even within the steady state regime by virtue of clusters slightly evaporating (past a bin threshold) and then growing back. This is also quite difficult to visualise.

Instead, the simpler method is to plot the mean of these cluster sizes (weighted by virtue of their categorisation into clusters) and chart how this mean evolves in time. The principle of monitoring stabilisation is the same, but it can be done much simply and visually through a two-dimensional image.

58 3. Machine Learning

With tools of analysis developed over the past 12 weeks (and the last few able to be fleshed out in the following weeks), we can move into the machine learning part of the project. This section has some general details about machine learning as a whole and the CNN we are using.

The principle of layered machine learning is that of receiving an input vector of datapoints (a ‘layer’ of ‘neurons’) which is fitted with a weight matrix and calibrated with a bias in order to generate successive learning layers; the whole process is given an ‘activation function’, which alters the results after a desired rule (which will be explained below). These weights are trained on the input data in order to correspond to some desired output data, with the intent of generalising this process to work on extended environments and cases. A useful way to express this in linear algebraic notation is:

$$\vec{a}^{(n+1)} = \sigma(\mathbf{W}\vec{a}^{(n)} + \vec{b})$$

where we take $\vec{a}^{(n)}$ to be the n layer of neurons, \mathbf{W} to be the weight matrix, \vec{b} to the associated bias vector, and σ to be the activation function. The weight matrix \mathbf{W} applies different weights to every neuron of the previous layer for *each* neuron of the current layer.

The activation function σ is a function which is meant to bind the neuron values within certain restrictions. A common function, for instance, is the sigmoid function, which binds neurons between 0 and 1, flattening very high values in both negative and positive directions towards the former and the latter, respectively. It is illustrated below:

$$\sigma_{sigmoid} = \frac{1}{1 + e^{-a}}$$

The reasoning behind the sigmoid function is to treat neurons on a continuous gradient between fully activated (1) and fully deactivated (0), in loose analogy with biological neurological systems. The problem with the sigmoid function is precisely the flattening behaviour it displays at high values - a big part of the machine learning process (precisely the ‘learning’ aspect) is the adjustment of weights with the purposes of minimising the cost of deviation from the expected values (which will also be explained below). High weight values will not produce meaningful values which can then be fine-tuned by the system, so the fluctuations in learning improve much less in practice, especially with a lot of inputs and layers involved.

What is often opted for instead, which is the activation function we are using in this project as well, is the **Rectified Linear Unit** (ReLU) function σ_{ReLU} ; this function sets any negative

number to 0, and any positive number to its own value. Under this function, varying the weights will always yield some feedback, i.e. learning, on how the algorithm can improve to better match expectations.

How does a machine learning algorithm figure out how to alter its weights? The brief answer is through an analysis of deviation, or cost. The machine is fed training data alongside the expected values, and it generates predicted values in turn based on the final layer of neurons. The algorithm obtains a cost function by summing the squares of differences between the expected output value of a neuron and its actual value. Not only does this function convey how well an algorithm performs on its training data (the bigger the value of the cost function is, the worse the algorithm is), but its principle can be used to compute a cost gradient - essentially, the algorithm can compute a cost function vector across all its weights and biases (which constitute its ‘dimensions’, in a calculus sense) and then take the gradient to determine how much each weight/bias parameter ought to change.

59 4. Convolutional Neural Networks (CNNs)

For our purposes, running a machine learning algorithm for 128x128 lattice sites will get very resource-intensive very quickly. The input layer alone requires more than 16000 neurons, and subsequent layers will require an extremely high number of weights to compile into new neurons, as, per the equation above, each new neuron $a_k^{(n)}$ will depend on all previous neurons $\vec{a}^{(n-1)}$. This results in poor scaling with increases in image size.

For this reason, alongside considerations of accuracy, this project employs a convolutional neural network. From hereon I will be denoting the number of a layer with l . The main difference in principle is that the main successive layer within the network is a *convolutional layer*, wherein each neuron at layer l is only connected to a given neighbourhood of neurons in layer $l - 1$ through a *kernel/filter* (the entity analogous to the aforementioned weight matrix \mathbf{W}). This kernel ‘moves’ across the input map in *strides*, generating one neuron value by adding the sum of its respective weights.

In broad strokes, the weight function of a standard machine learning algorithm applies holistically to the entire previous layer map, whereas the kernel of a convolutional neural network applies locally to a region of the previous layer map. Both are subsequently subjected to an activation function before yielding a neuronal value.

The consequence of a moving kernel is that the (output) feature map of a convolutional layer will be smaller than the input map it analyses. Using many convolutional layers would therefore mean that the edges of the input map would be factored considerably less in the learning algorithm compared to the centre. The feature maps would get smaller and smaller - crucially, the more layers there are, the more focus is shifted towards the central part of the initial map.

One approach is to introduce *padding*. Essentially, padding constitutes inserting values at the edges of the output feature map in order to increase its overall size. In most cases, the aim is to preserve the size of the input map across convolution, and the edges are padded with zeroes (“zero-padding”). Note that the amount of padding P must align with a relational formula of the different sizes of the system in order to be able to construct a valid convolutional layer:

$$\frac{V_i - F + 2P}{S} - 1 \in \mathbb{Z}$$

where V_i is the input volume size (for square images), F is the receptive field size, and S is the stride.

The **receptive field size**, in turn, can be calculated generally using the following formula (taken from Araujo et al, 2019):

$$F_0 = \sum_{l=1}^L ((k_l - 1) \prod_{i=1}^{l-1} s_i) + 1$$

where k_l is the kernel size of layer l , s_l is the stride size of layer l , and L is the final layer of the system.

Or recursively using the following formula:

$$F_{l-1} = S_l F_l + (k_l - s_l)$$

Finally, the amount of kernels determines the **depth** of a feature map. Essentially, we can picture each kernel as generating a two-dimensional grid, which can be stacked along the z-axis (the depth-axis, in other words). This is important in order to preserve the depth of the original input map, pertinently stored in our case in *colour*, through an RGB depth of 3. Therefore, for all CNN algorithms which make use of particle orientation in our PEP system, we will need three filters in our convolutional layers.

60 5. Dataset Rolling

One key trait of the in-house code has been a ‘rolling’ of the dataset. Essentially, once the iteration of a dataset is generated by the `sampler.py` function, it is stored under the header `conf_{iteration}`. This iteration is then ‘rolled’ (using the `np.roll` function) 12 times along the x-axis, essentially creating new datasets from the same generative process. So far, we have not been making use of this in-house code property in our data at all - this was intended specifically for the machine learning part of the project, so it was not brought up in our analysis (which was strictly focused on the non-rolled datasets). For reference, the rolled datasets are stored within the `h5py` file as `conf_{iteration}_{roll}`.

But this brings up an interesting point now that we are moving into the machine learning parts. The CNN will be training on all available datasets in order to draw its inferences. To what extent is introducing rolling data favourable to our predictions? On one hand, training the model on recycled data may give it a weighing bias towards the specific conditions of the datasets we present it with. There is the worry of overfitting, or overspecialisation to strictly the circumstances (P_t , ρ and (N_x, N_y) combinations) it receives as input. On the other hand, the rolling provides more samples on which the CNN can train, with a much smaller computational cost. The end of this experiment involves large scale data manipulation and generation, and if rolling proves to be a reliable tool to train a model on, it will aid immensely in the generation aspect.

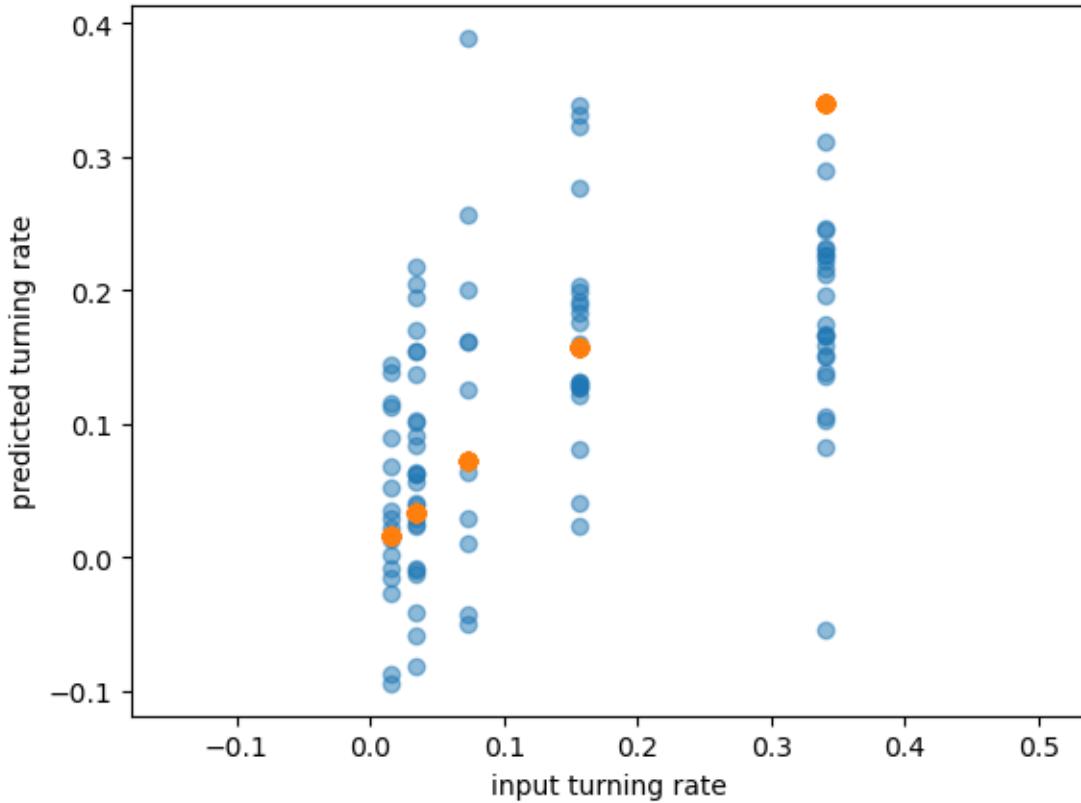
On this note, I have done some quick analysis on the data. As a rule of thumb, datasets with rolling take approximately 10 extra seconds to generate in comparison to the strictly unrolled datasets. This has been tested on my personal computer - it is possible the time difference is much smaller once a supercomputer is employed.

The file size contrast is considerable. Datasets which have surplus rolled iterations have approximately 284MB each. Datasets which do not only have about 66MB. Since all the files have been stored on my computer, this is considerable insofar as so far around 215MB per dataset have gone unused (though they will now be employed by the CNN). For example, to obtain the biggest cluster size distribution in [Weeks 9-12](#) required 100 different datasets, resulting in about 19GB of data that was not called.

Data analysis aside, we will run the CNN using both options, and contrast the results for different scenarios. I have rewritten `sampler.py` to accommodate the requirements of unrolled dataset files (as `sampler_no_roll.py`). This code stores its datasets in a separate folder (`data/no-roll`).

61 6. Preliminary CNN Experiment

We can start with a simple case to illustrate the principle. Taking a single density value of $\rho = 0.15$, we can train the model on a few probability distributions. Running for 5 epochs and using only one convolutional layer, with 3 5x5 kernels with 3x3 strides, we get the following predictions:

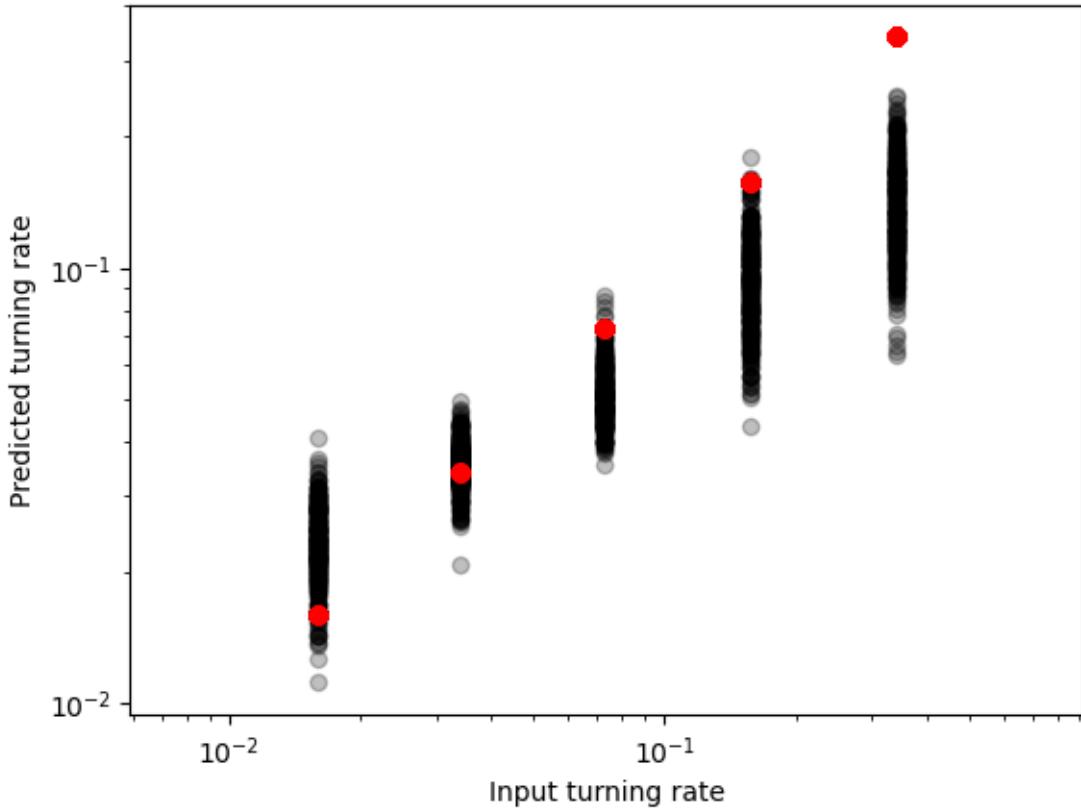


where the orange data points are the input turning rates, and the blue data points are the model predictions.

The spreads are quite big on most probability values, and for the highest one the model is clearly not fitting properly at all. Nontheless, this is a good start for how simple the layer diagram is:

INPUT => CONV => FLATTEN => FC (optimiser: adam ; 5 epochs)

After some experimenting, below are the predictions made on the unrolled data, running for 10 epochs:

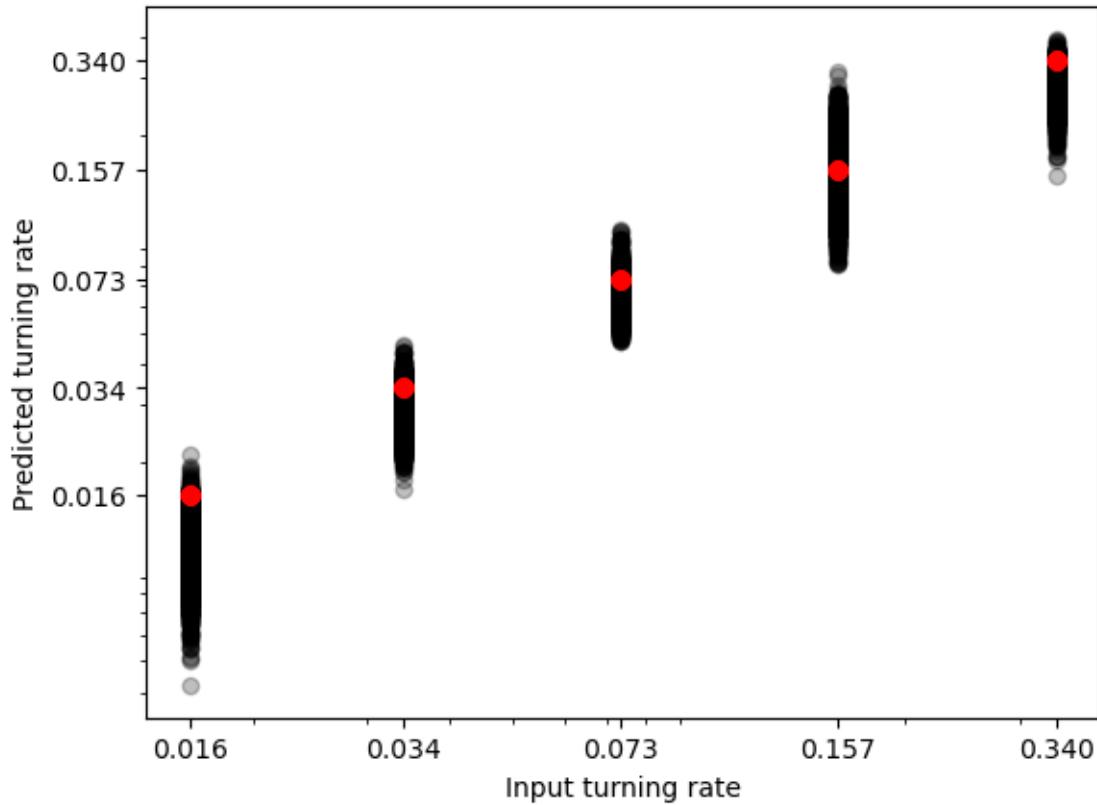


With the following layer diagram:

INPUT => CONV => BN => CONV => BN => MAXPOOL => CONV => BN => CONV => BN => MAXPOOL => FC => DO => FC => FLATTEN => FC (optimiser: adam ; 10 epochs)

The training size is also important; this model is trained on one dataset for each probability distribution (so 5 datasets total). Each of these datasets has 1000 snapshots, leaving a total of 5000 data. We trained the model on 3000 snapshots, and then applied it to the last 2000.

The natural question is how the predictions change if we train on longer evolutions. Below is the graphical result of this, with the same $(\rho, P_t, (N_x, N_y))$ configuration, but with 10 times the generated evolution steps (and therefore snapshots). This means the mode is now trained on 30000 snapshots, and then applied to the last 20000.



We can see the fitting drastically improve for the higher tumbling rates, though it's clearly still lacking at the extremes of our turning rates.

To understand the general principle and methodology of our preliminary analysis, see the uploaded [Preliminary Analysis Notebook](#) example.

62 7. CNN Layers

Quick definitional list for the most commonly used CNN layers.

- Convolutional Layer (CONV): Standard convolution layer, mapping the input map through kernels in strides, and generating a feature map of lower dimensionality as a result.
- Activation Function Layer (RELU): Applies the activation function to the emergent neurons. Often left implicit after a CONV or FC layer.
- Batch Normalisation Layer (BN): Normalises layer inputs through re-centring and re-scaling.
- Pooling Layer (POOL): Separates input into patches; replaces each patch in input with single value in output, which is often either...
 - MAXPOOL: the maximum value within the pool
 - AVGPOOL: the average value within the pool
- Dense/Fully Connected Layer (FC): Standard neural network layer; maps each neuron of the feature map to every neuron of the input map.
- Dropout Layer (DO): Nullifies contribution of some neurons towards next layer while keeping the rest intact.
- Flattening Layer (FLATTEN): Flattens feature map into a one-dimensional column.

63 8. Preliminary CNN Tests on Rolled Data

So far we have trained the model strictly on non-rolled datasets. Below is an example of the same 1000 iteration per dataset scenario, but used on rolled datasets to facilitate more samples for the model to train on. As a result, there are 70000 samples in the dataset, 50000 of which are training data, and 20000 of which are validation data.

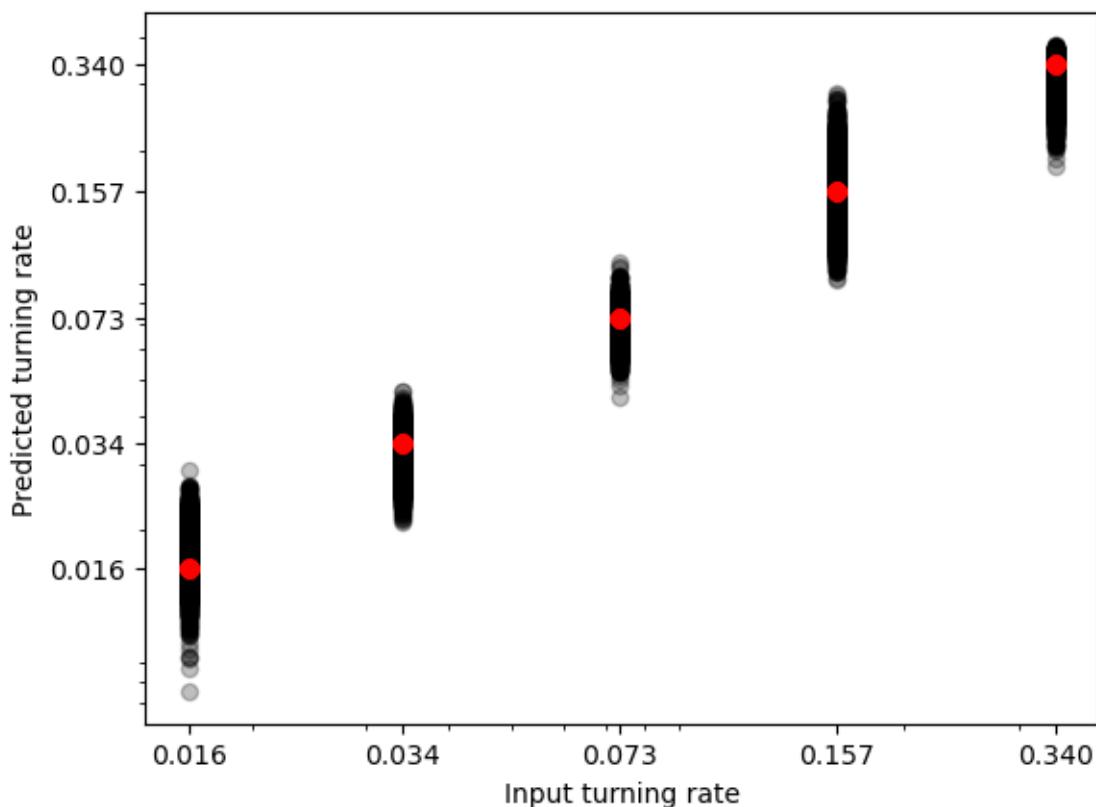


Figure 63.1: Pearson's correlation coeff: 0.98276

We can see that the fitting is even better in the low iteration rolling case than in the high iteration no-rolling case. However, the $P_t = 0.34$ values are consistently predicted incorrectly across these two examples, which might suggest an underlying problem with the system.

64 9. Training and Predicting on Mixed Densities

The next evident question is how the model holds up when we apply the above principles to a broad range of densities. In the figure below, the CNN runs the same framework on 100 different densities. Unfortunately, I have not been able to construct predictions for such a model yet, as the CUDA memory storage overwhelms my current machine and crashes. As a result, this will be done once the supercomputer system is fully set up.

65 10. Current Problems

We are currently training on every snapshot of a lattice's evolution. This means that our model trains on both steady state and non-steady state segments of a system. In the following weeks it will be worth separating these segments for the purposes of analysing how training the CNN system only on steady state postively or negatively affects its ability to derive tumbling rates.

66 11. Misc Notes

J. Chodera - Equilibration in Monte Carlo and molecular dynamics

67 Week 16

68 0. Table of Contents

69 1. Introduction

Now that we have set up the machine learning system and begun familiarising ourselves with its architecture, the aim of the next few weeks (this one included) is to begin experimenting on different training and validation data, as well as tuning the architecture.

70 2. Checklist

Our evolving checklist of things to explore regarding CNNs can be found [here](#) (Sheet 1). The highlight of what has been currently done and what is currently aimed at has been transcribed below.

- Data augmentation (rolled): noticeable improvement
- More samples: noticeable improvement
- Different alphas (training): noticeable improvement
- Gaps in alphas (training)
- Different densities
- Data augmentation (rotate)
- Tune learning rate/optimiser
- Tune batch size

71 3. Model List and Storage

In order to keep track of our many models, involving different layers and different data training sets, we opted to create a standardised table detailing the different parameters going into the system, such as training data snapshots, ratio of training to prediction data, etc. Below is a quick (preliminary) snapshot of the first few lines of the table, for reference. Note that a lot of columns have been skipped, which detail the utilised densities, tumbling rates and system sizes used for training and validation.

1	RUN ID	# SNAPSHOT	AUGMENTATION	OPTIMIZER	LEARNING RATE	BATCH SIZE	EPOCHS	MAE	STD MIN	STD AVG	STD MAX	OVERLAP R	R
2	headgear2718	20000 (0.2)	none	adam	0.001	64	10	0.02	0.001	0.05	0.02	0.80	0.98
3	atreus1273	70000 (0.29)	rolling (12)	adam				0.01					0.98
4	michigan2241	10000 (0.2)	none	adam				0.03	0.000	0.03	0.01		
5	kangaroo2519							0.07	0.038	0.07	0.05	0.90	0.78
6	sandpaper7571							0.20	0.000	0.06	0.01	0.20	0.89
7	corned1441						20	0.03	0.000	0.06	0.01	0.80	0.98
8	nearness8197						10	0.02	0.001	0.05	0.02	1.00	0.99
9	affected1468						20	0.04	0.001	0.05	0.02	0.80	0.97
10	implement6908						10	0.06	0.001	0.02	0.01	0.10	0.97
11	culture7803						20	0.02	0.001	0.06	0.02	0.70	0.98
12	subwoofer0187							0.02	0.001	0.06	0.02	0.70	0.98
13	gone0635						15	0.02	0.001	0.05	0.01	0.80	0.99
14	posh3633					32		0.02	0.002	0.05	0.02	0.70	0.97
15	multitask7949					64	30	0.04	0.001	0.06	0.02	0.40	0.96
16	awry4366	40000 (0.2)					10	0.02	0.001	0.05	0.02	0.80	0.98
17	lying1339						20	0.02	0.002	0.06	0.02	0.90	0.99
18	recite9661	6400 (0.2)	rolling (6)				70	0.00	5.93E-05	0.00	0.00	1.00	1.00
19	entrench0085	12800 (0.2)				0.0005	30	0.01	0.006	0.01	0.01	1.00	0.39
20	pruning0896					0.001		0.00	0.004	0.00	0.00	1.00	0.50
21	shamrock2212			sgd	0.0003 (def 0.01)		180	0.01	0.007	0.01	0.01	1.00	0.10
22	astronaut1046					0.005		0.01	0.011	0.02	0.01	1.00	0.98

The growing model list can be found [here](#), on the same link as the checklist above (Sheet 2). Below are transcribed our current sought metrics (coloured in green within the table above).

- Maximum STD: 0.02
 - This is the maximum standard deviation a particular prediction spread can have. For example, given a model, the maximum STD determines the biggest spread in the output predictions (which are generated for some specific tumbling rates). Ideally the spreads would be similar across, but we can consider the learning to have failed if any individual prediction has too big a spread.
- Average STD: 0.01
 - This is the average standard deviation a particular prediction spread can have. Taking the same example as above, this averages over all predicted tumbling rates to obtain an average spread deviation. This naturally has to be small as well in order for the model to succeed.

- Mean Average Error: 0.01
- Pearson's Coefficient: 0.975
- Overlap Ratio: 1
 - This ratio indicates the amount of tumbling rate prediction spreads which overlap at all with the expected tumbling rates.

Combining a ubiquitous overlap ratio with a very small average and maximum standard deviation yields results which are both very accurate and very precise.

72 4. Shorthand Model List

While each model is referred to by the aforementioned standardisation, it is also useful to communicate architecture more quickly. Here's the current list of different architectures being used, all indexed by a model number (MN). The latter predictions generated in [Weeks 13-15](#) are all done with MN_1, for example.

72.0.1 MN_1

1. CONV (filters=3,kernel_size=(3,3),padding='same',strides=(3,3),activation="relu", input_shape=shape)
2. BN
3. CONV (filters=3,kernel_size=(3,3),padding='same')
4. BN
5. MAXPOOL (pool_size=(3,3))
6. CONV (filters=6,kernel_size=(3,3),padding='same')
7. BN
8. CONV (filters=6,kernel_size=(3,3),padding='same')
9. BN
10. MAXPOOL (pool_size=(3,3))
11. FC (units=128,activation='relu')
12. DO (0.2) (without layout optimiser)
13. FC (units=10,activation='softmax')
14. FLATTEN
15. FC (units=1,activation='linear')

72.0.2 MN_2

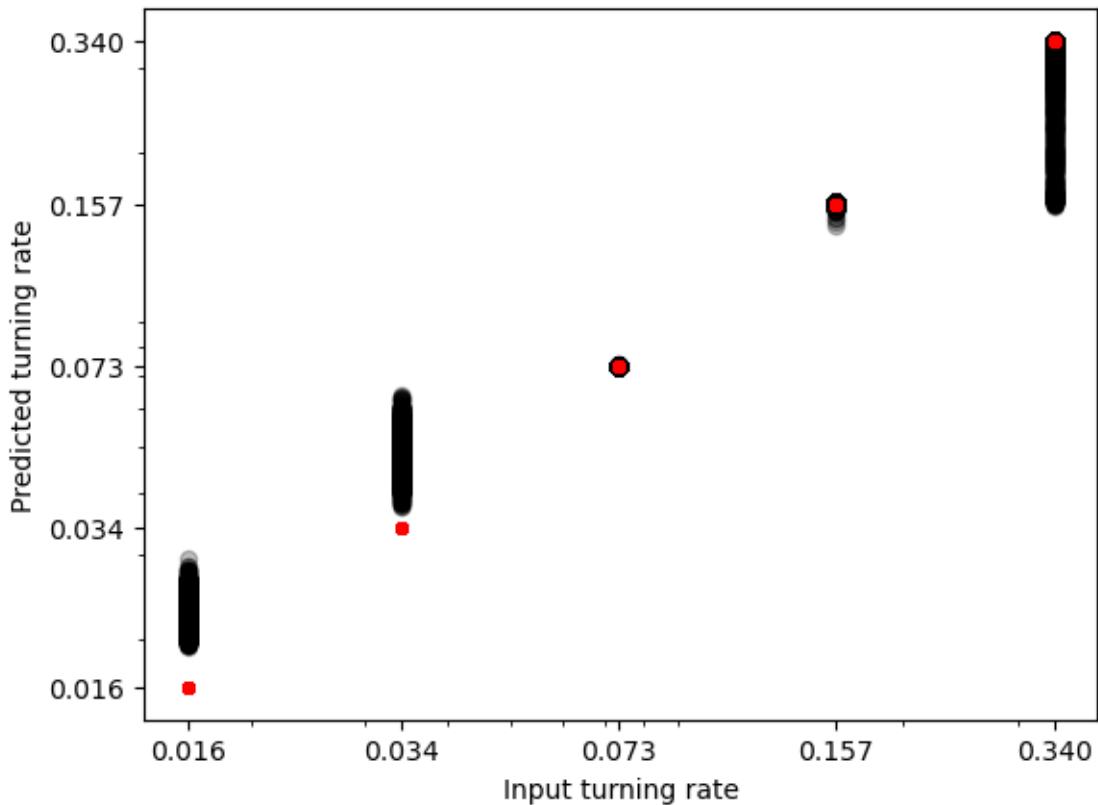
1. CONV (filters=3,kernel_size=(3,3),padding='same',input_shape=shape)
2. MAXPOOL (pool_size=(2,2),padding='same')
3. ReLU
4. BN
5. CONV (filters=6,kernel_size=(5,5),padding='same')
6. MAXPOOL (pool_size=(2,2),padding='same')
7. ReLU

8. BN
9. AVGPOOL
10. DO (0.2) (without layout optimiser)
11. FC (units=64,activation='relu')
12. DO (0.2) (without layout optimiser)
13. FC (units=10,activation='softmax')
14. FLATTEN
15. FC (units=1,activation='linear')

As more architectures are employed, the full list will be updated [here](#).

73 3. New Architecture

Using the architecture of MN_2, we can try to examine a similar case to which we applied MN_1. Below is the prediction of model atreus1273, applied to a sample size of 70000 iterations spread equally among 5 turning rates (50000 training, 20000 validation), for density 0.15.

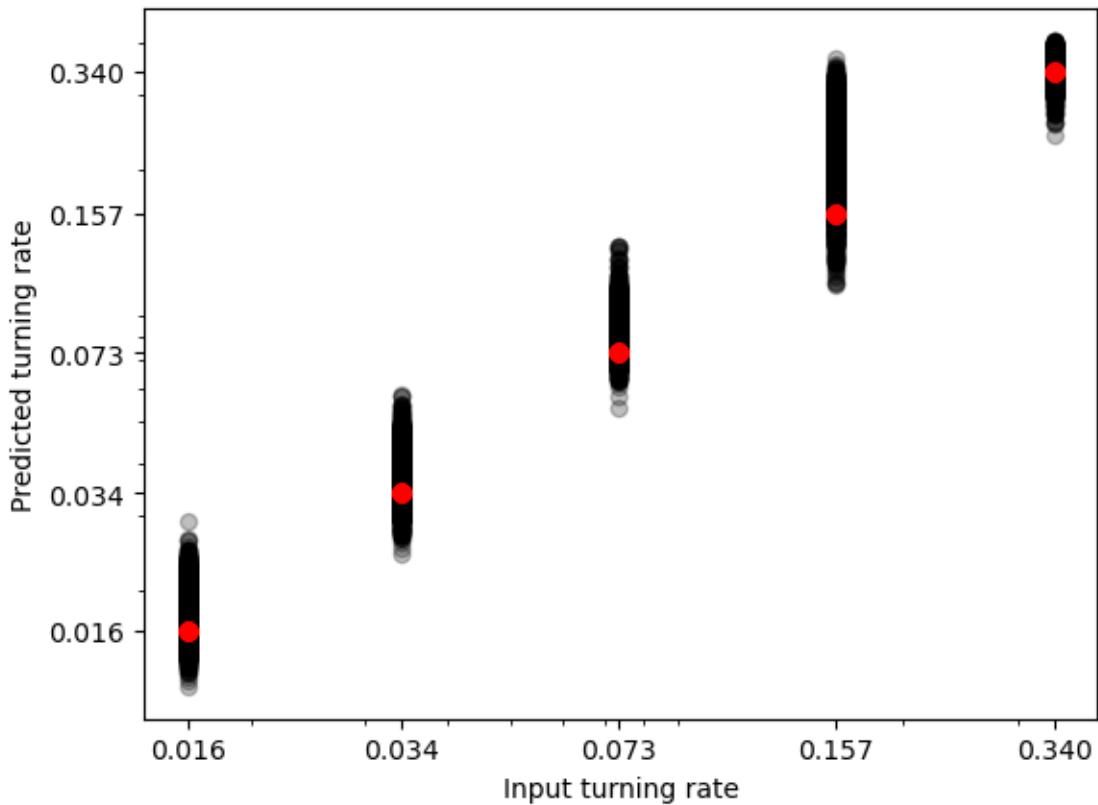


The fitting overall seems to be worse, but the top middle values are almost perfectly fitted. My lab partner has gotten better results with this model, so I will continue exploring it in parallel with MN_1 for now.

74 4. More Tumbling Rates

There is an obvious extension in increasing the amount of tumbling rates utilised. This may help the CNN pick up on the clustering/dissipation pattern better by providing more scenarios in which it can be examined.

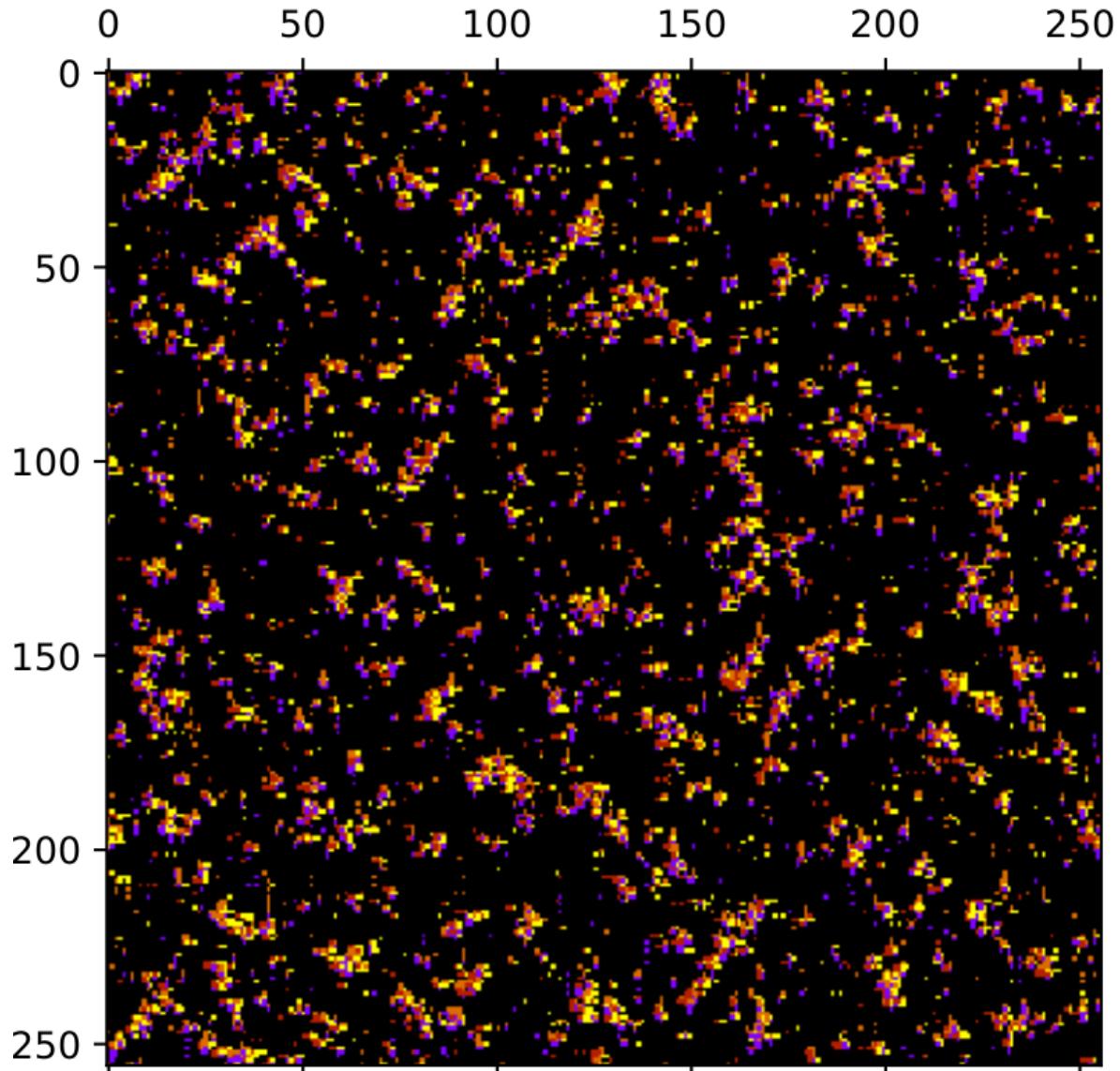
Below is an example reproduction of MN_1 trained on the same values as the last example in [Weeks 13-15](#), model median4431. It has now been standardised to be easily referred to in our table.



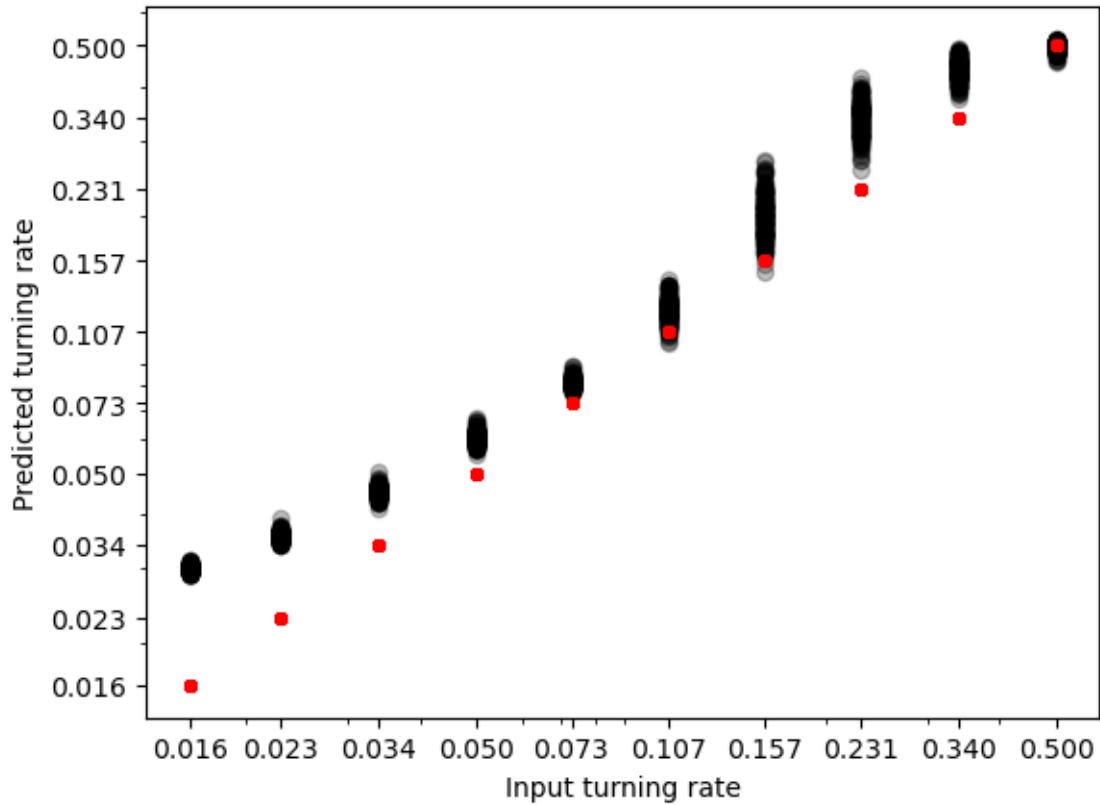
Note that this is using rolled data, so it is bound to be more accurate than the unrolled variants we will consider below.

75 4. Bigger N Values

Here is a brief attempt to train the CNN on broader N values. This will be investigated more in depth at the end of the project. Model michigan2241 is trained on $N_x = N_y = 256$; an example screenshot can be found below.



Running MN_2 architecture, the training yields the following predictions:



We can see the fitting does not match

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 3)	30
max_pooling2d (MaxPooling2D)	(None, 128, 128, 3)	0
re_lu (ReLU)	(None, 128, 128, 3)	0
batch_normalization (Batch Normalization)	(None, 128, 128, 3)	12
conv2d_1 (Conv2D)	(None, 128, 128, 6)	456
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 6)	0

g2D)

re_lu_1 (ReLU)	(None, 64, 64, 6)	0
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 6)	24
global_average_pooling2d (GlobalAveragePooling2D)	(None, 6)	0
dropout (Dropout)	(None, 6)	0
dense (Dense)	(None, 64)	448
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650
flatten (Flatten)	(None, 10)	0
dense_2 (Dense)	(None, 1)	11

=====

Total params: 1631 (6.37 KB)
Trainable params: 1613 (6.30 KB)
Non-trainable params: 18 (72.00 Byte)

76 5. Model Summaries

77 Week 17

78 0. Table of Contents

1. Introduction
2. Story
3. Discussing Similar Research
4. MN_3 Discussion
5. Example MN_3 Application
6. Gaps in P_{tumble} (monochrome)

79 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.

80 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.

80.0.0.0.1 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 predictions 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.

80.0.0.0.2 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.

80.0.0.0.3 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.

80.0.0.0.4 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.

80.0.0.0.5 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.

81 3. Discussing Similar Research

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

82 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 throughout (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 combining 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 particles 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.

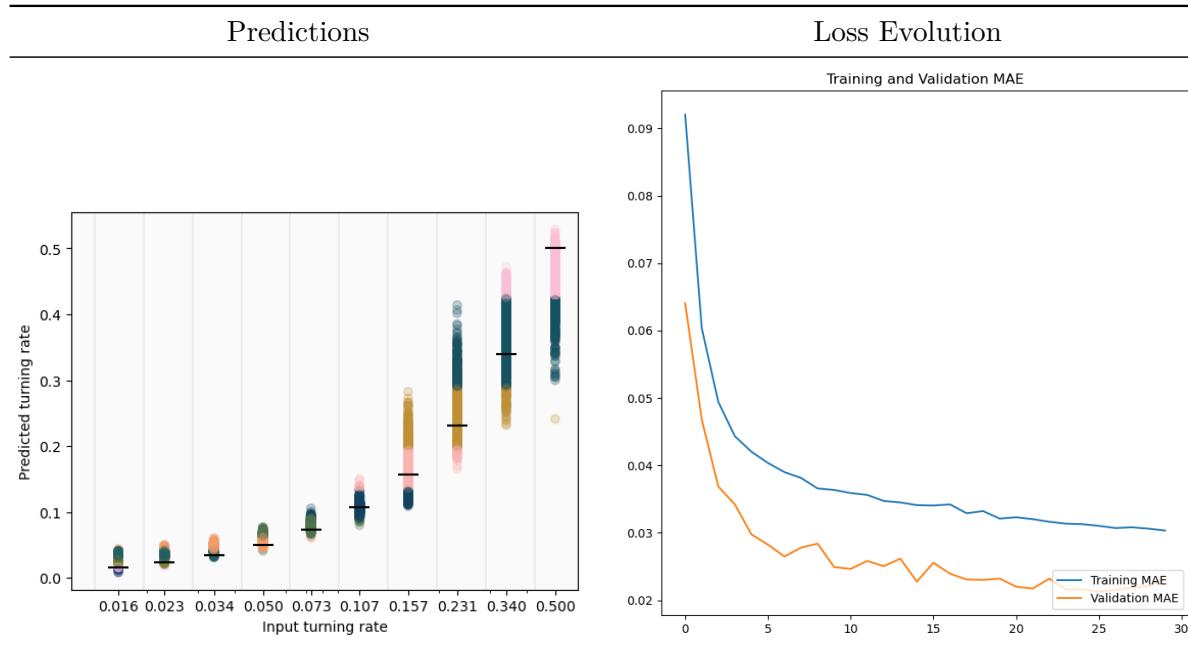
We have used ‘same’ padding due to the borders of the image being overall important in our case.

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.

83 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, $0.2 \times 40000 = 8000$ is the validation pool). Training was done for 30 epochs.



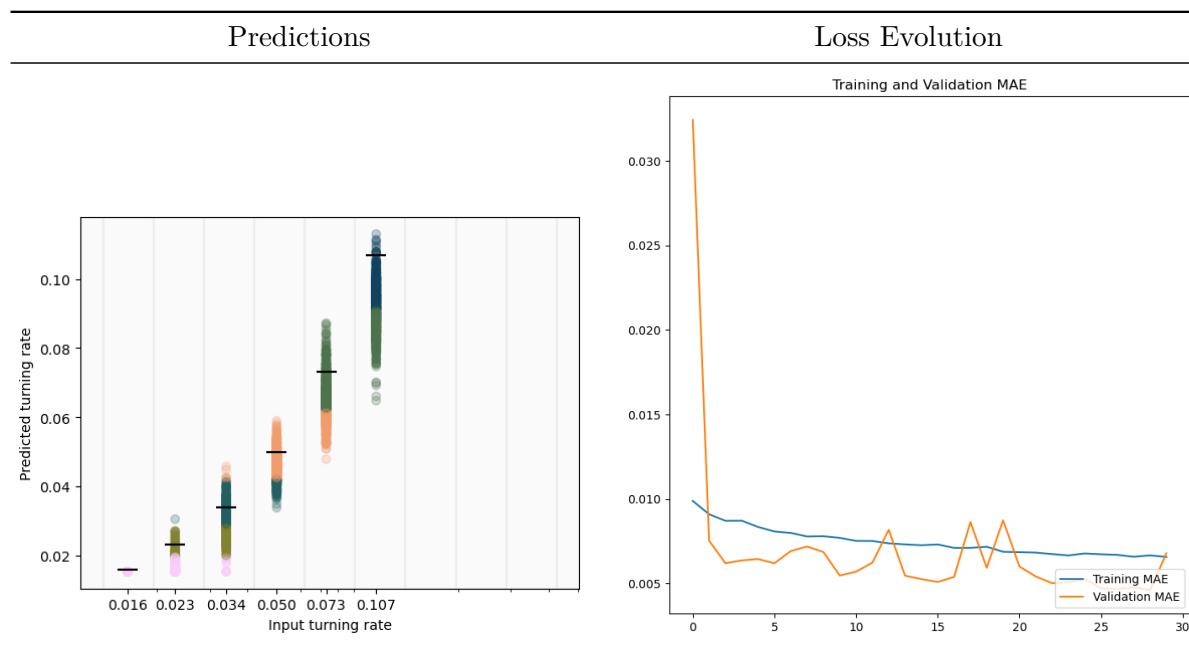
Some parameters are within acceptable margins as outlined in [Week 16](#). Intuitively from the graph, the deviations from our margins relate specifically to the maximum spread (as can be seen in the higher tumbling rate values) and the Pearson's coefficient.

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.

84 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**.

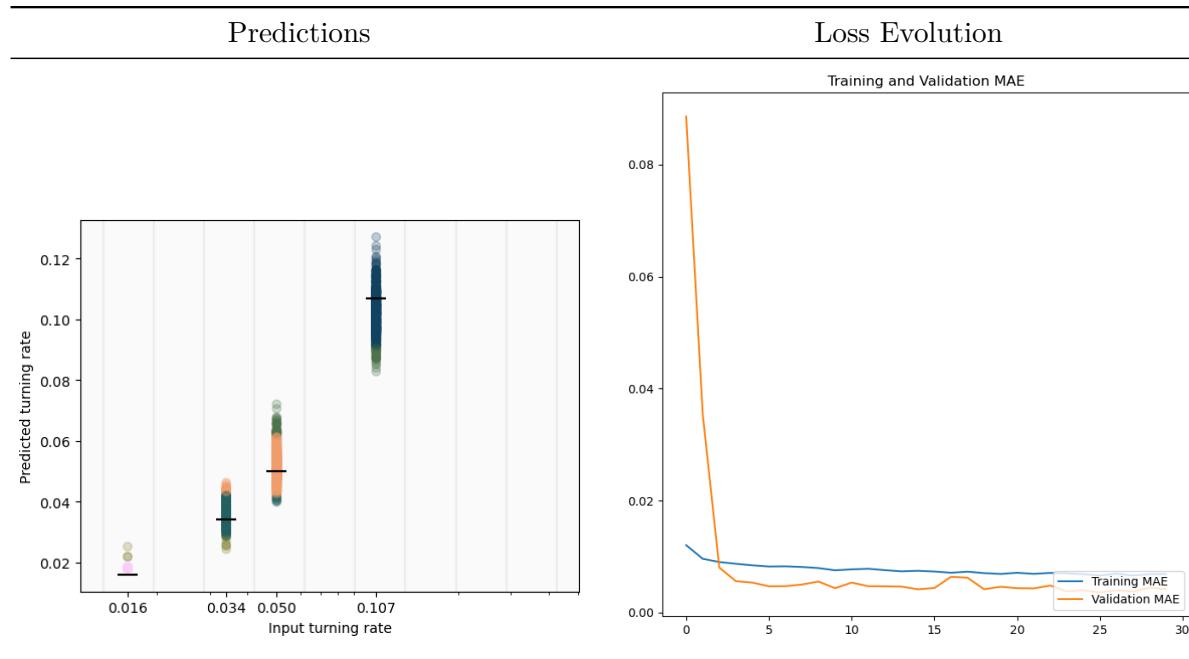
84.0.1 balteus3123: $P_t \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107\}$



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

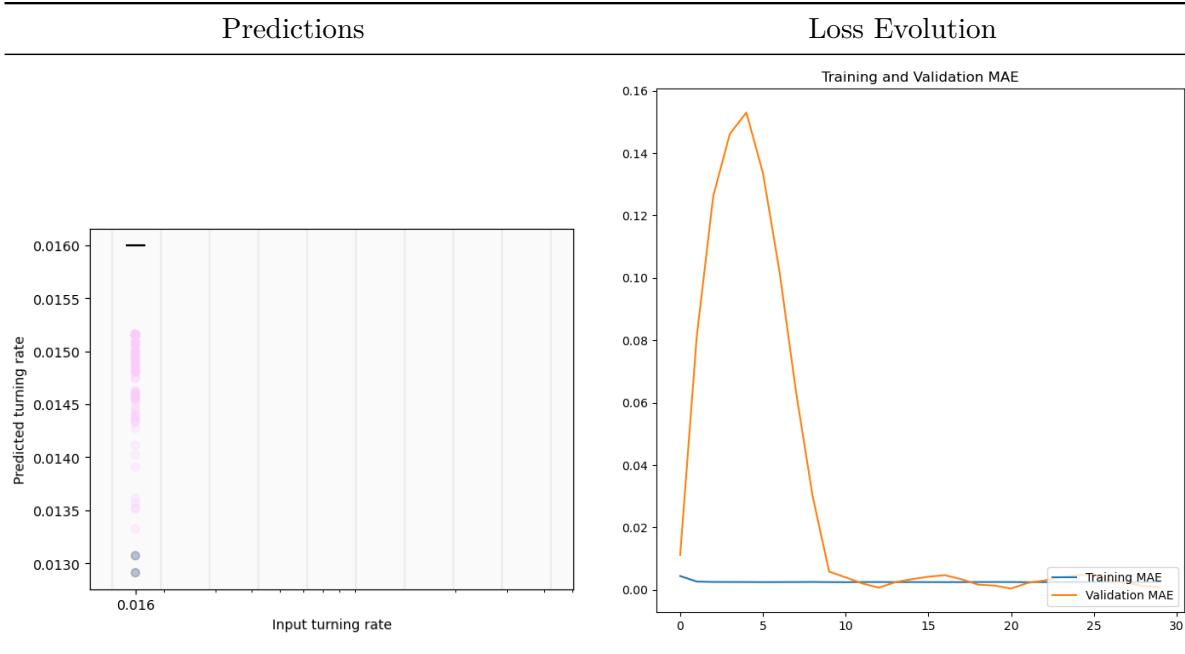
84.0.2 goose4421: $P_t \in \{0.016, 0.034, 0.050, 0.107\}$



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: 0.988395863033326

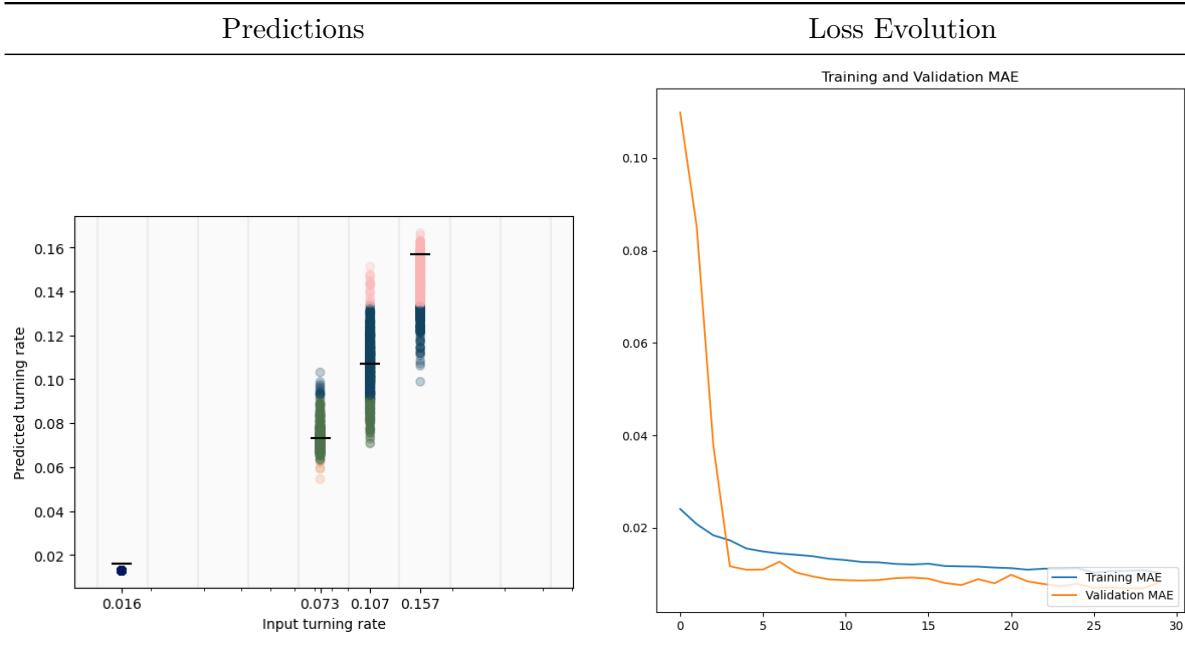
84.0.3 lyrical2734: $P_t \in \{0.016\}$



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

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

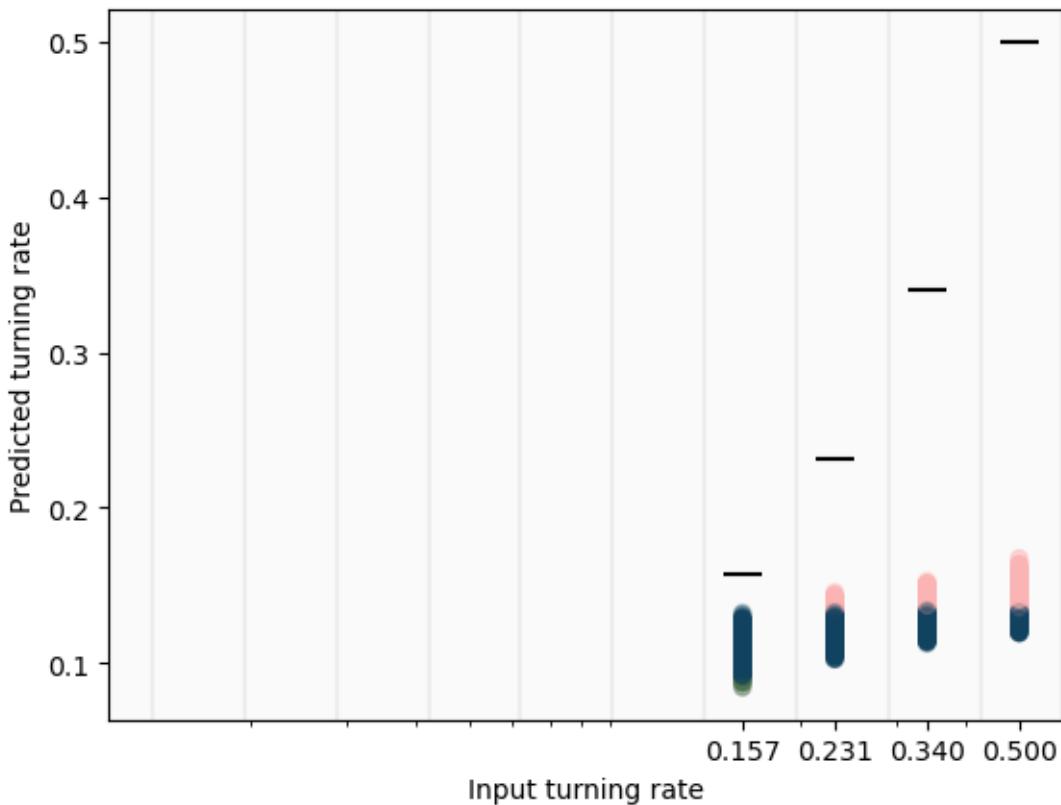


Parameters

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

85 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 $\text{img} > 0$ $\text{img} = 1$

86 Week 18

87 0. Table of Contents

1. Introduction
2. Refining CNN Architecture
3. Degrees of Freedom Discussion
4. Density Discussion in Big Tumbling Rate Spread
5. Gaps in Lower Tumbling Rates for Higher Densities
6. Gaps in Higher Tumbling Rates for Higher Densities
7. Omitting Highest Tumbling Rate
8. Different Density Comparison (Omitting Highest Tumbling Rate)
9. Multiple Nearby Densities
10. Epoch Numbers
11. Monochrome Interpolation (Low Tumbling Rates)
12. Monochrome Interpolation (High Tumbling Rates)
13. Monochrome Extrapolation

88 1. Introduction

The main purpose of this week is to further explore gaps in tumbling rates for our CNN predictions, alongside discussions of densities and clusters. The auxiliary purpose of this week is to flesh out the discussion around degrees of freedom.

89 2. Refining CNN Architecture

The intermediary convolutional layer in MN_3 was running a (4,4) kernel size; this is uncentered, and therefore slightly hinders the model. We have swapped it out for a (5,5) kernel size.

We could do ‘contour plots’ of individual clusters to map out how their orientation

90 3. Degrees of Freedom Discussion

We have identified three different changes we could make to the system images before experimenting with the CNN that could pose interesting results.

- Orientation case
- Monochrome case
- Confusion case: This case consists of a random scrambling orientations. The orientation case keeps its orientation categories, but we alter the image such that these categories do not mean anything. Our expectations are that:
 - Training on Confusion case will yield the same result whether validated on Confusion or Orientation case.
 - Training on Orientation case will yield better results when validated on Orientation case, rather than when validated on Confusion case. We expect this because the Orientation case training *should* prime the model to detect an intrinsic feature of the system, which is then completely scrambled by the Confusion case. If our hypothesis is incorrect, and validating on Orientation in fact yields similar results, it would mean that the Orientation case does not pick up on this degree of freedom in its analysis.
 - The Orientation case should overall produce better results than the Confusion case, unless our hypothesis in the last bullet point is false.
- Misinformation case: This case consists of misattributing a random percentage of particle orientations in an Orientation case image. This has physical parallels to misidentifying the orientation of active matter particles from a two-dimensional perception (as they are three-dimensional swimmers).
- Noise case: This case consists of giving a random (float) noise distribution in a Monochrome case image. The reasoning behind this is granting the CNN the ability to distinguish

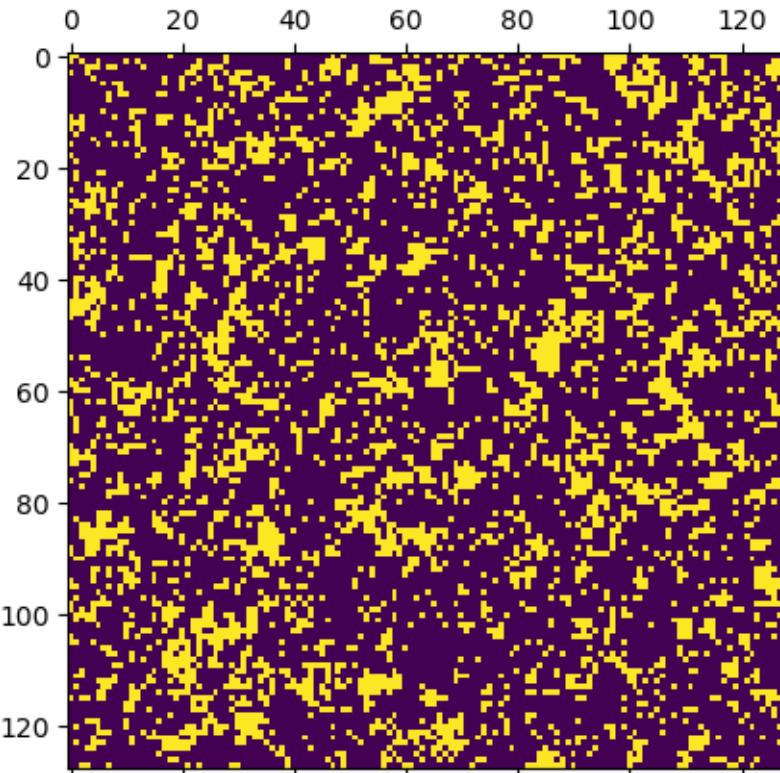
91 4. Density Discussion in Big Tumbling Rate Spread

A natural question that arises out of the density analysis done in [Week 17](#) is why the prediction distributions get skewed by the upper probability values. That is to say, why does adding bigger tumbling rates significantly decrease the prediction accuracy and precision?

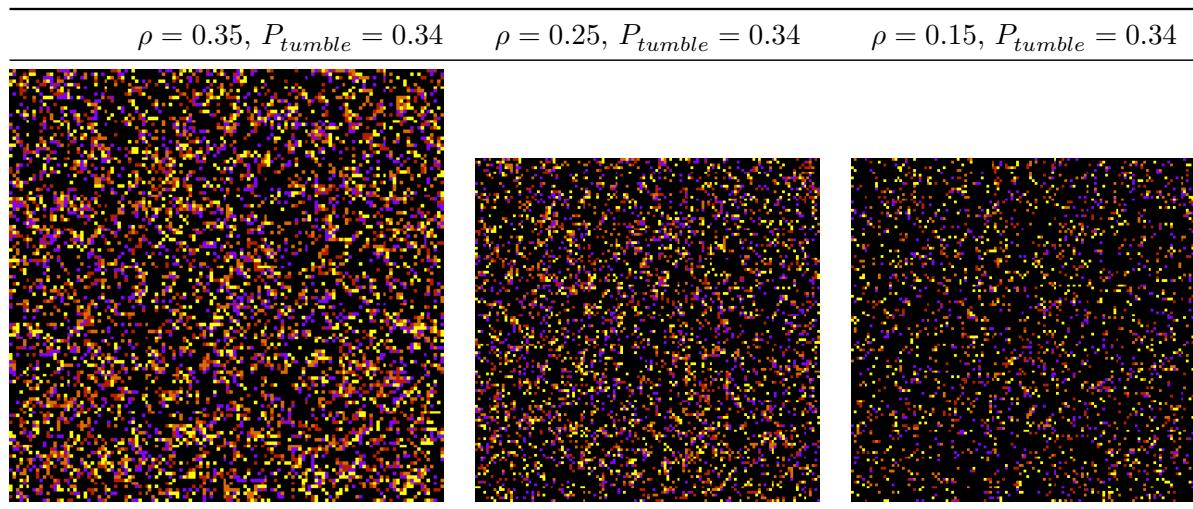
Our current theory is that higher tumbling rates do not exhibit the clustering behaviour which the CNN is tracking in order to assert its predictions. Once the tumbling rate reaches a certain amount for our $\rho = 0.15$ case, it is more difficult for the CNN to draw comparisons, due to the feature landscape dramatically changing. This essentially causes the CNN to misrecognise these different (clustering and non-clustering) ranges of data, across both cases having too small a training sample size to effectively predict the probabilities.

The natural fix for such a problem is more data, but there is some other analysis that can be done to further explore the situation. As stated above, we have so far been working on a density of $\rho = 0.15$. Provided our theory is correct, we might notice changes to predictions by *increasing* the density, thus allowing for clustering at higher tumbling rates.

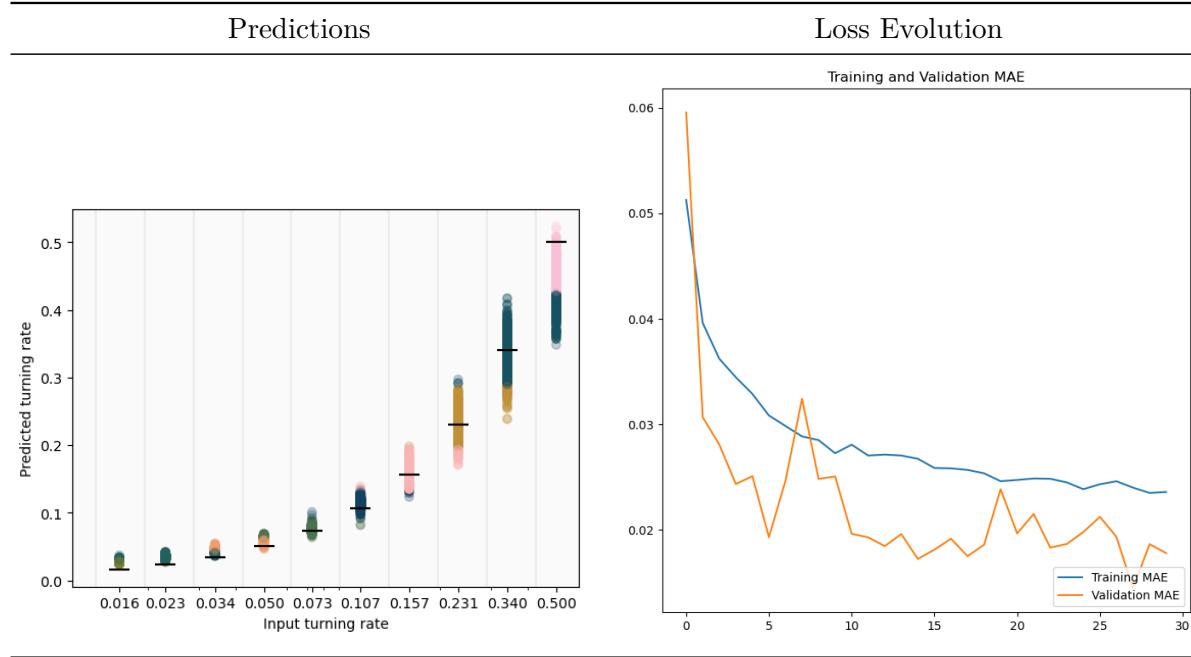
Below is an example of how the landscape looks for $\rho = 0.25$. This is a randomly selected image from the pool of utilised probabilities, so its tumbling rate is unknown; nonetheless this image gives a visual idea of the amount of particles on the screen.



And below is a side by side comparison of the last screenshots of an evolution using $P_{tumble} = 0.34$, for $\rho = 0.35$ (left), $\rho = 0.25$ (center) and $\rho = 0.15$ (right). We can see that in the center case the density is hitgh enough for clusters to begin forming (though only barely), whereas the left case already has more noticeable clusters.

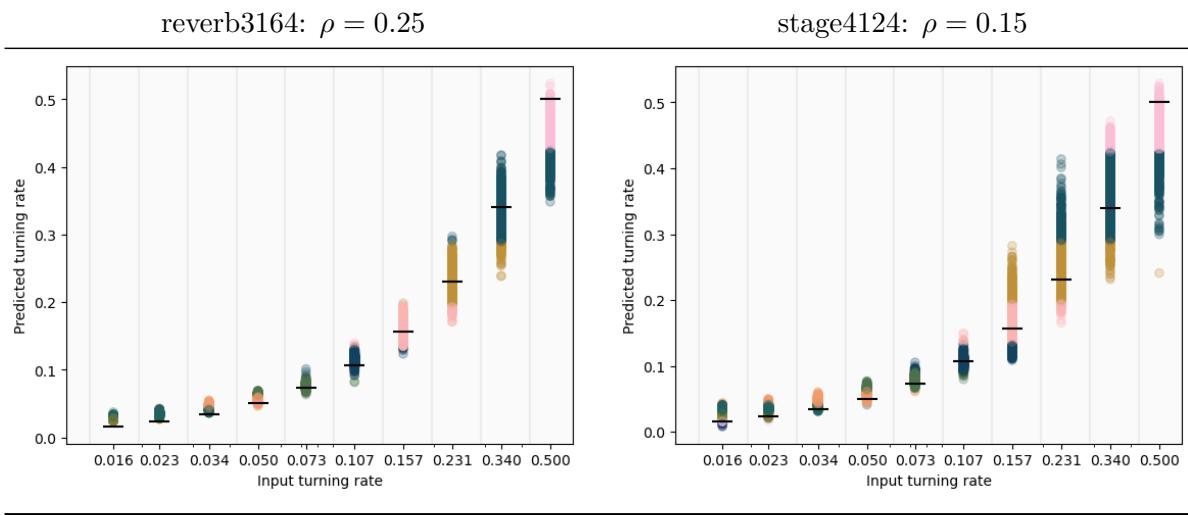


91.0.0.1 reverb3164: $\rho = 0.25$,
 $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.157, 0.231, 0.340, 0.500\}$, **30 epochs**



- MAE: 0.0177948232740164
- Min STD: 0.001830725
- Avg STD: 0.011430472
- Max STD: 0.030022161
- Overlap Ratio: 0.7 (acc 1e-3)
- Pearson Coefficient: 0.990001865469512

We can also visually compare reverb3164 (the $\rho = 0.25$ case) with stage4124 (the analogous $\rho = 0.15$ case):



Given that the scales are the same, we can qualitatively notice a decrease in spread (i.e. an increase in accuracy) while jumping from a smaller density to a larger density. This is reflected in our quantitative results:

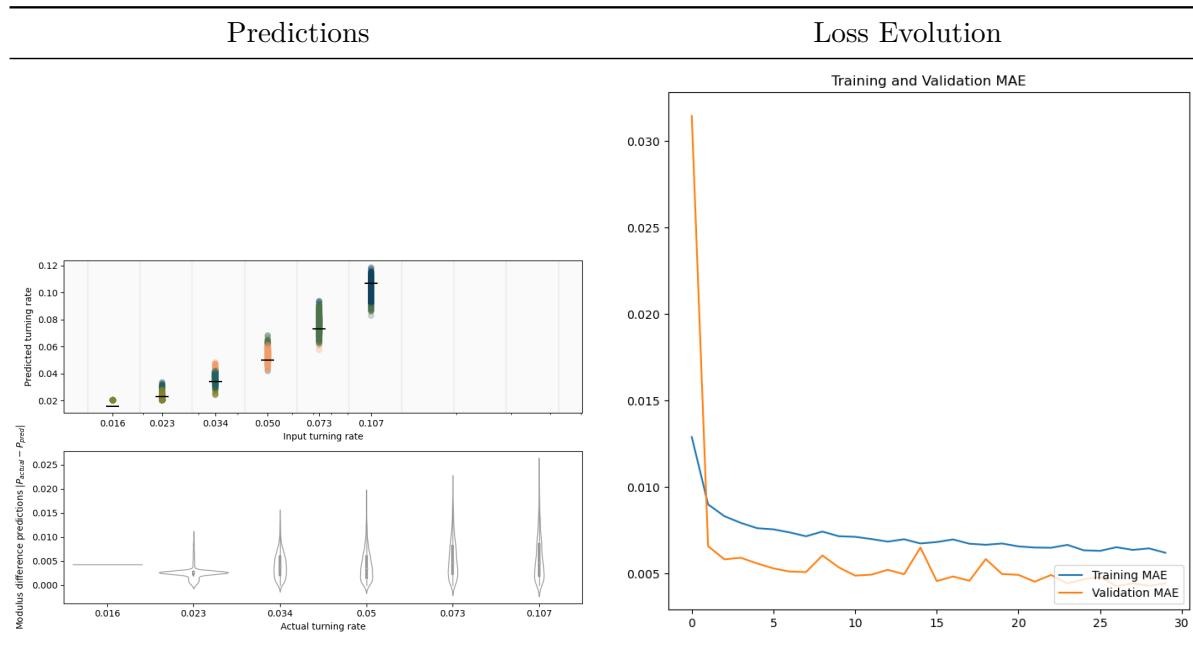
Parameter	reverb3164	stage4124
MAE	0.017795	0.000895
Avg STD	0.011430472	0.019767912
Max STD	0.030022161	0.04721327
Overlap Ratio	0.7	1.0

We can see that the standard deviations are lower for a bigger density, both on average and regarding maxima. We have, however, included two parameters which are in fact worse in the higher density case: the overlap ratio and the mean absolute error. Regarding the overlap ratio, this is directly tied to the lowering of spread: if we look to the lower density case, we can see that the lower values which are ‘hit’ there (and which are barely missed in the higher density case) do not hit so with the centre of their distribution, but rather only with the spread periphery. There is also the issue of the accuracy we’ve been employing so far for our overlap ratio: 1e-3 is simply too small to account for distributions which do not have a big spread, but are nonetheless within the vicinity of the guessed distribution. We later decided to increase the accuracy to 5e-3; there is an argument to be made that it should be increased even further; in reality the *contents* of these probability distributions matter much more: are they Gaussian? We will see once we introduce violin plots and absolute deviation considerations that this is indeed the case. Furthermore, how does the *mean* of the distribution relate to the expected value? Within how many standard deviations are they from each other? This will also start to be factored in in the analysis below.

92 5. Gaps in Lower Tumbling Rates for Higher Densities

We begin by mirroring the cases we explored with $\rho = 0.15$. The thought is that we can show the prior results (again, see [Week 17](#)) are somewhat general by doing so, while also exploring how allowing more clusters to be picked up on across the tumbling rate distributions slightly improves our data.

92.0.0.1 salad8110: $\rho = 0.25$, $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107\}$, **30 epochs**

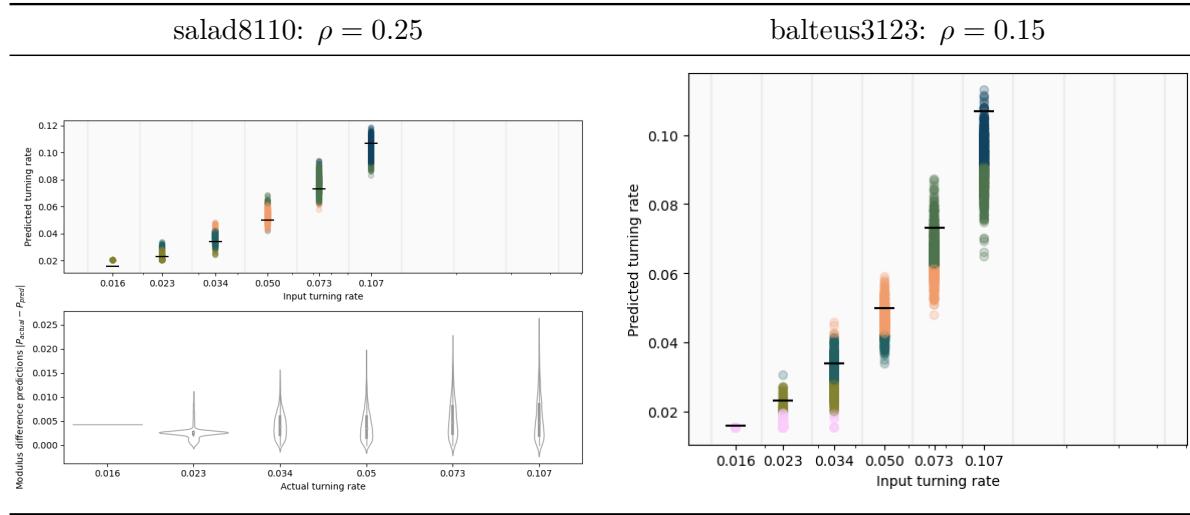


- MAE: 0.004445969592779875
- Min STD: 1.8626451e-09
- Avg STD: 0.0035765618
- Max STD: 0.0058416952
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.9865499725921921

We can now see the prediction distributions that the CNN makes, and get confirmation that they are broadly Gaussian in form. This suggests that a mean analysis of our predictions would accurately reflect the prediction dynamics at play. Note that the violin plots show specifically the absolute difference between the expected and predicted values, and so values are better the closer they are to zero. We can also see what was previously intuited from the original distribution graphs: the spread does get larger with increased tumbling rates.

We can also see exactly how precise the predictions of the CNN are regarding the lowest value. Rather than a Gaussian, the $P_{tumble} = 0.016$ case appears at our scope to be a constant distribution (it is, in actuality, still a Gaussian distribution with an extremely narrow standard deviation; the CNN prediction naturally never hits the exact same real number twice).

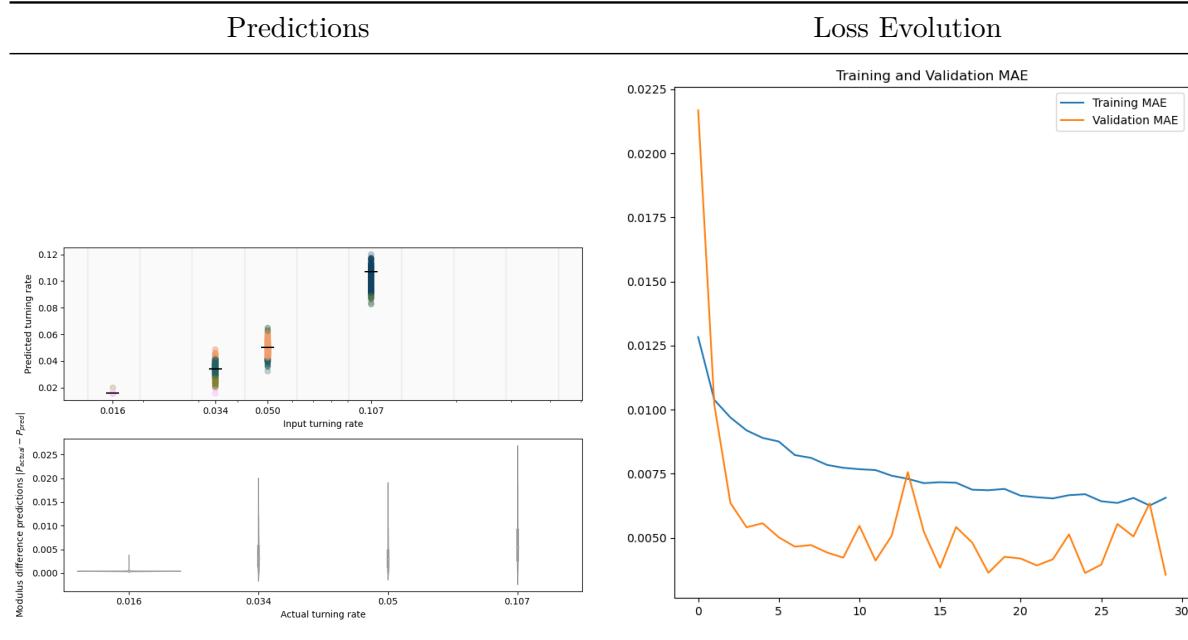
Beyond that, we can see that the above parameters are still very good. We can once again visually and quantitatively compare with the $\rho = 0.15$ case:



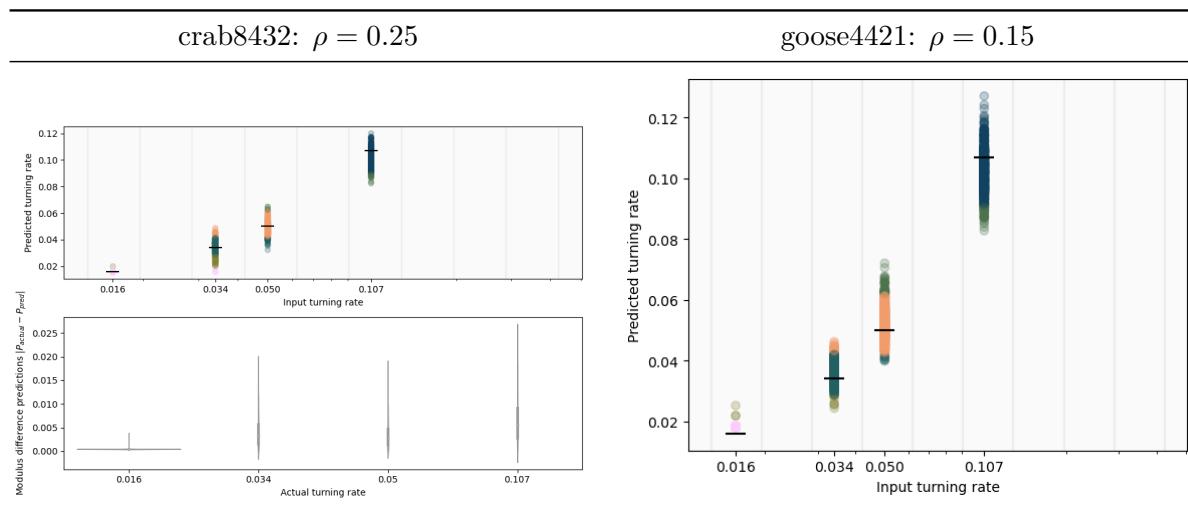
Parameter	salad8110	balteus3123
MAE	0.004446	0.006792
Avg STD	0.0035765618	0.00386919
Max STD	0.0058416952	0.006857624
Overlap Ratio	1.0	1.0

And see that once again, the higher density case yields better results over all. Note, of course, that we are discussing differences of magnitude 10^{-3} (for MAE and Max STD) and 10^{-4} (for Avg STD).

92.0.0.2 crab8432: $\rho = 0.25$, $P_{tumble} \in \{0.016, 0.034, 0.050, 0.107\}$, **30 epochs**



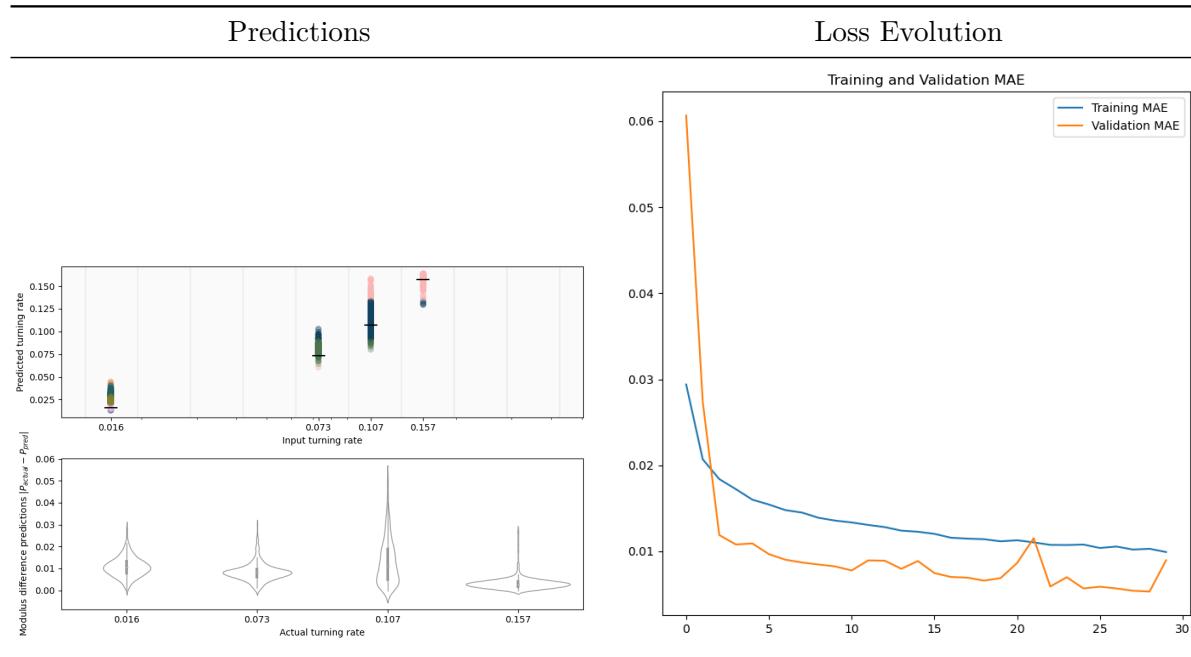
- MAE: 0.003553365357220173
- Min STD: 0.00015766203
- Avg STD: 0.0037065577
- Max STD: 0.005531624
- Overlap Ratio: 1.0 (acc 5e-3)
- Pearson Coefficient: 0.9910517414919411



Parameter	crab8432	goose4421
MAE	0.003553	0.004174
Avg STD	0.0037065577	0.0038618112
Max STD	0.005531624	0.006754256
Overlap Ratio	1.0	1.0

(Note: the overlap ratio for goose4421 was adapted to the new criterion of accuracy 5e-3)

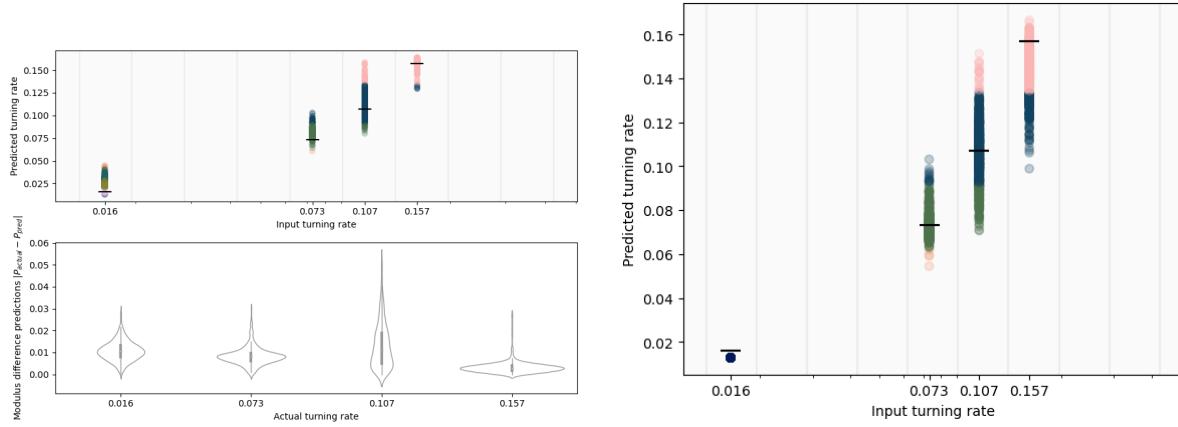
92.0.0.3 summer6911 $\rho = 0.25$, $P_{tumble} \in \{0.016, 0.073, 0.107, 0.157\}$



- MAE: 0.008954823948442936
- Min STD: 0.004150052
- Avg STD: 0.0066630687
- Max STD: 0.013605628
- Overlap Ratio: 1.0 (acc 5e-3)
- Pearson Coefficient: 0.9869872375647769

summer6911: $\rho = 0.25$

book1634: $\rho = 0.15$

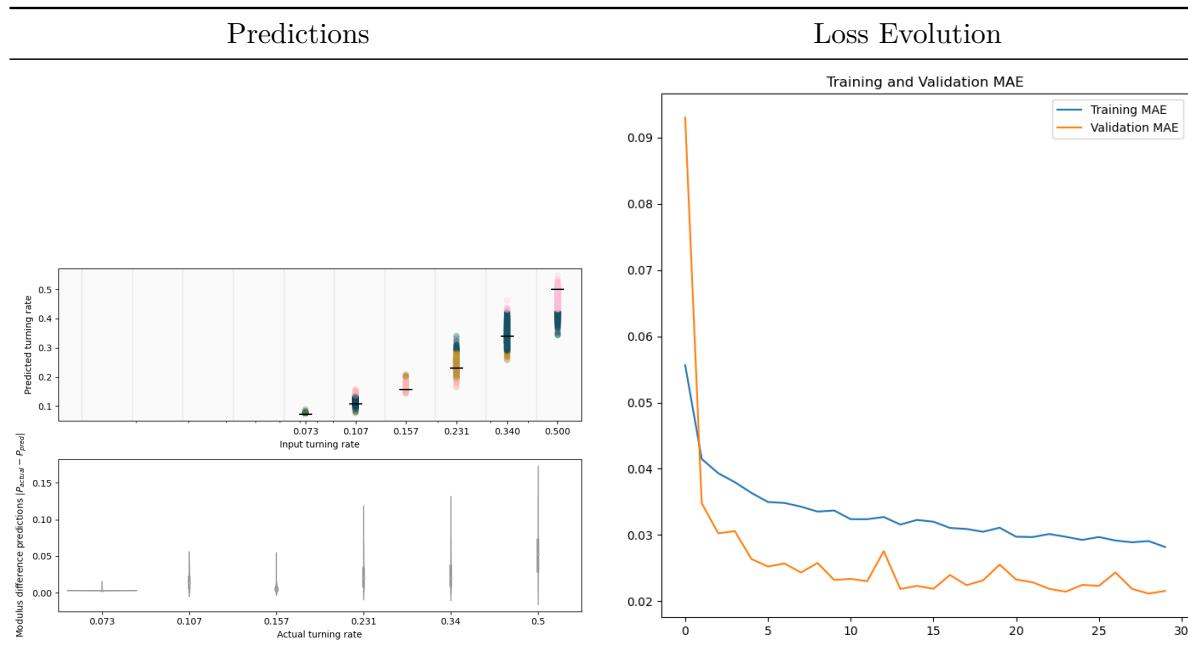


Parameter	crab8432	book1634
MAE	0.008955	0.008299
Avg STD	0.0066630687	0.007414392
Max STD	0.013605628	0.013894851
Overlap Ratio	1.0	0.75

(Note: the overlap ratio just barely misses book1634 in the $P_{tumble} = 0.016$ case even with the 5e-3 extension)

93 6. Gaps in Higher Tumbling Rates for Higher Densities

93.0.0.1 salmon9100: $\rho = 0.25$, $P_{tumble} \in \{0.073, 0.107, 0.157, 0.231, 0.34, 0.5\}$, **30 epochs**



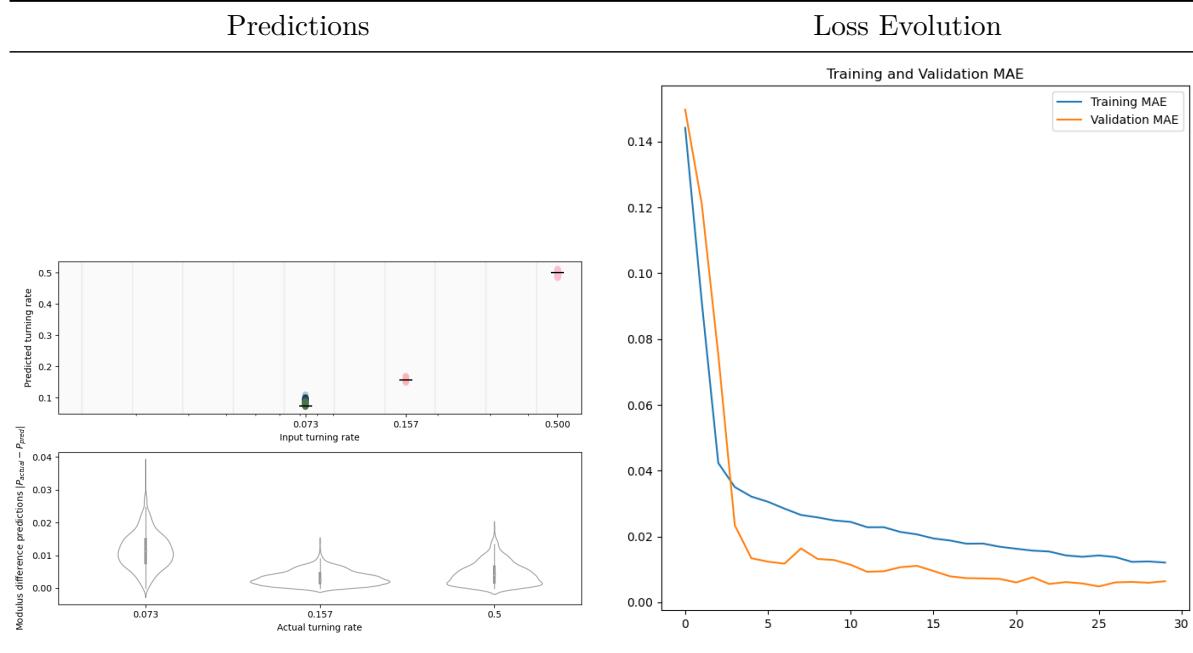
Prediction means and standard deviations.

Actual value 0.073: Average = 0.07637 +- 0.00064; Expected value within 5.309 stdevs of mean
 Actual value 0.107: Average = 0.11958 +- 0.01252; Expected value within 1.005 stdevs of mean
 Actual value 0.157: Average = 0.16357 +- 0.00663; Expected value within 0.990 stdevs of mean
 Actual value 0.231: Average = 0.24277 +- 0.02668; Expected value within 0.441 stdevs of mean
 Actual value 0.34: Average = 0.34620 +- 0.03178; Expected value within 0.195 stdevs of mean
 Actual value 0.5: Average = 0.44783 +- 0.03338; Expected value within 1.563 stdevs of mean

- MAE: 0.021534917876124382
- Min STD: 0.0006356345
- Avg STD: 0.01860332

- Max STD: 0.033375088
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.9793251812649026

93.0.0.2 tread4399: $\rho = 0.25$, $P_{tumble} \in \{0.073, 0.157, 0.5\}$, 30 epochs



Prediction means and standard deviations.

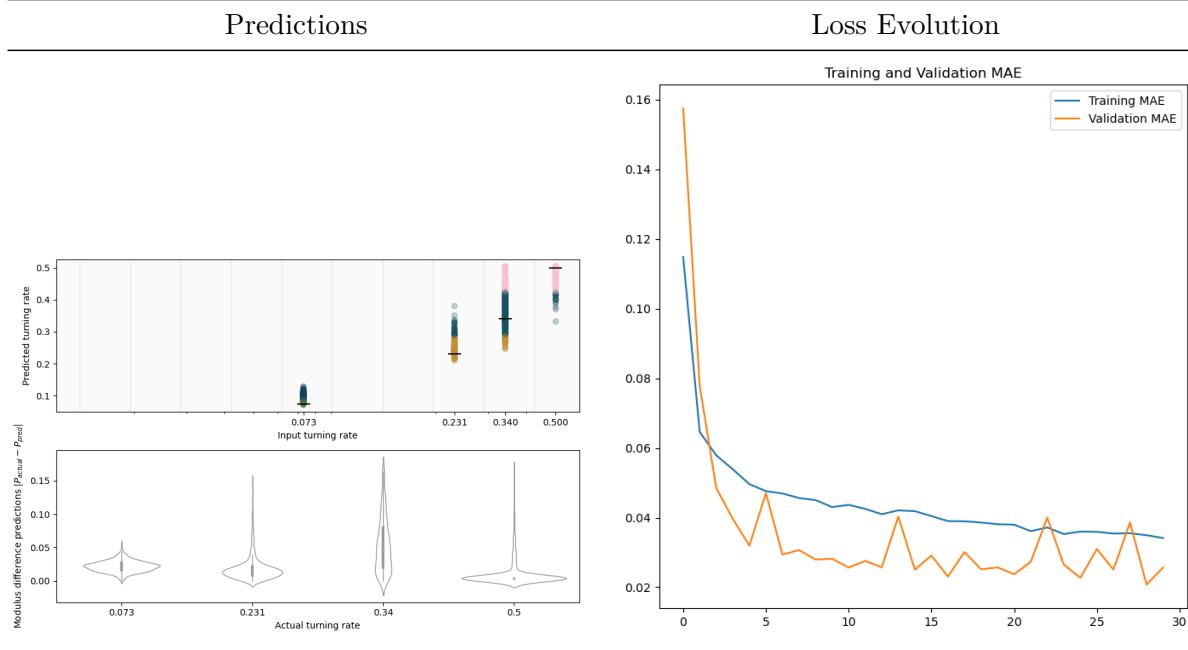
Actual value 0.073: Average = 0.08449 +- 0.00540; Expected value within 2.128 stdevs of mean

Actual value 0.157: Average = 0.15972 +- 0.00279; Expected value within 0.973 stdevs of mean

Actual value 0.5: Average = 0.49756 +- 0.00513; Expected value within 0.475 stdevs of mean

- MAE: 0.00639208871871233
- Min STD: 0.0027909318
- Avg STD: 0.004439787
- Max STD: 0.0054007815
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.999563884360183

93.0.0.3 revolve8117: $\rho = 0.25$, $P_{tumble} \in \{0.073, 0.231, 0.340, 0.500\}$, 30 epochs



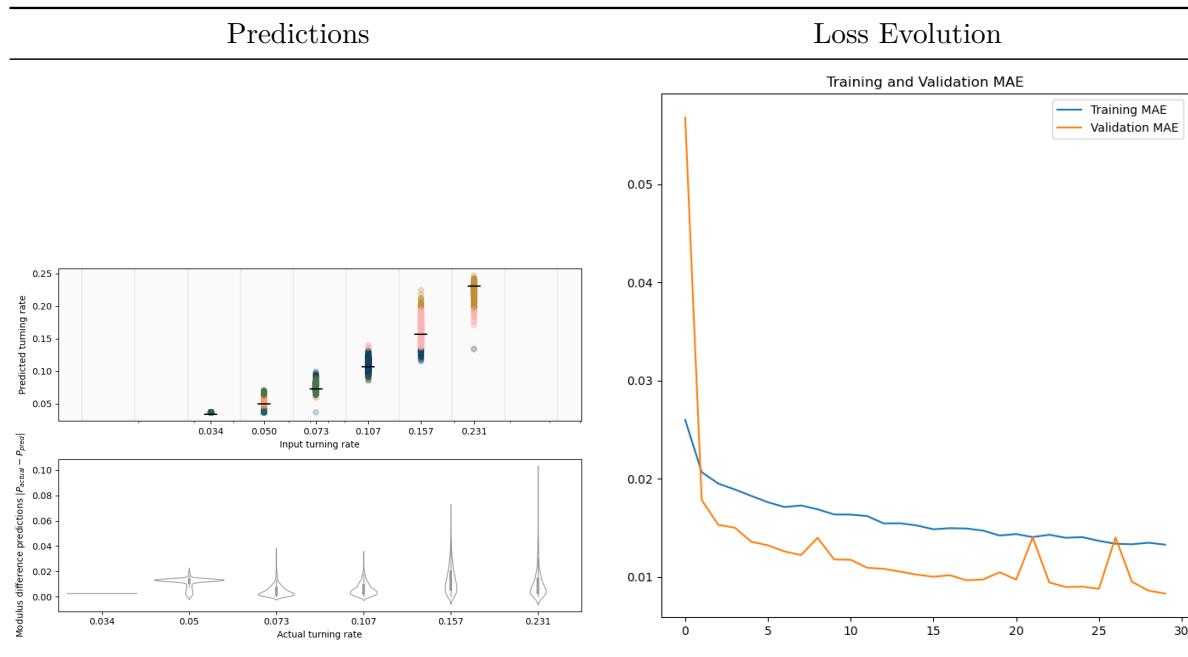
Prediction means and standard deviations.

Actual value 0.073: Average = 0.09487 +- 0.00837; Expected value within 2.613 stdevs of mean
 Actual value 0.231: Average = 0.24794 +- 0.01834; Expected value within 0.924 stdevs of mean
 Actual value 0.34: Average = 0.37743 +- 0.05689; Expected value within 0.658 stdevs of mean
 Actual value 0.5: Average = 0.49567 +- 0.02125; Expected value within 0.204 stdevs of mean

- MAE: 0.0256530288606882
- Min STD: 0.008370637
- Avg STD: 0.026211156
- Max STD: 0.05688635
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.974904538431155

94 7. Omitting Highest Tumbling Rate

94.0.0.1 flag1899: $\rho = 0.25$, $\$P_{\{tumble\}} \{0.034, 0.050, 0.073, 0.107, 0.157, 0.231\}$, 30 epochs



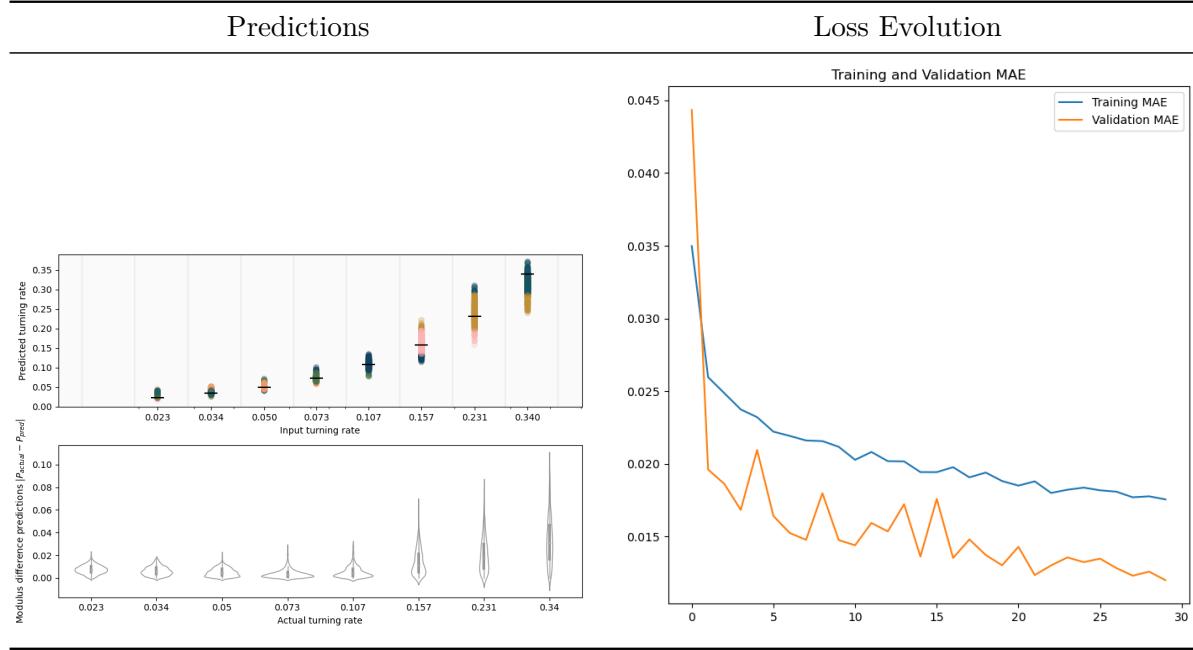
Prediction means and standard deviations.

Actual value 0.034: Average = 0.03666 +- 0.00000; Expected value within inf stdevs of mean
 Actual value 0.05: Average = 0.04100 +- 0.00806; Expected value within 1.116 stdevs of mean
 Actual value 0.073: Average = 0.07631 +- 0.00551; Expected value within 0.600 stdevs of mean
 Actual value 0.107: Average = 0.11042 +- 0.00746; Expected value within 0.458 stdevs of mean
 Actual value 0.157: Average = 0.16183 +- 0.01717; Expected value within 0.282 stdevs of mean
 Actual value 0.231: Average = 0.22189 +- 0.01270; Expected value within 0.718 stdevs of mean

- MAE: 0.00829365104436874
- Min STD: 0.0 !!
- Avg STD: 0.008482912
- Max STD: 0.017168295

- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.984593225567436

94.0.0.2 candy8131: $\rho = 0.25$,
 $P_{tumble} \in \{0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.231, 0.340\}$, **30 epochs, 32000 (0.2) snapshots**

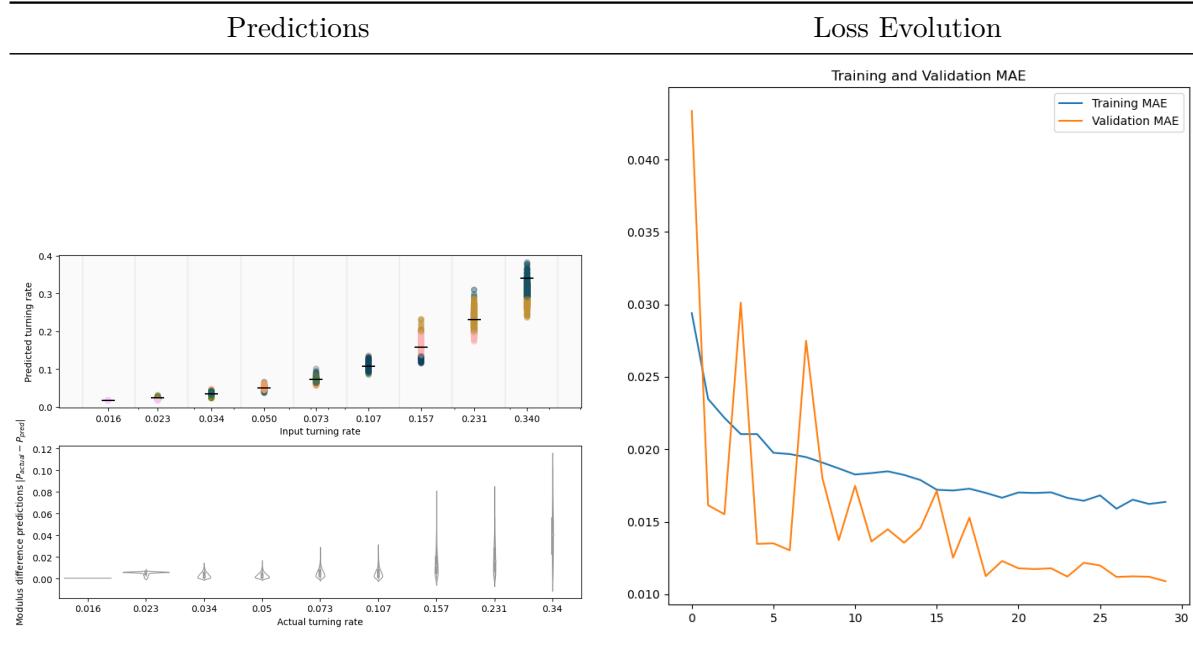


Prediction means and standard deviations.

Actual value 0.023: Average = 0.03053 +- 0.00393; Expected value within 1.917 stdevs of mean
 Actual value 0.034: Average = 0.04008 +- 0.00437; Expected value within 1.393 stdevs of mean
 Actual value 0.05: Average = 0.05421 +- 0.00473; Expected value within 0.891 stdevs of mean
 Actual value 0.073: Average = 0.07196 +- 0.00521; Expected value within 0.199 stdevs of mean
 Actual value 0.107: Average = 0.10870 +- 0.00724; Expected value within 0.235 stdevs of mean
 Actual value 0.157: Average = 0.15856 +- 0.01793; Expected value within 0.087 stdevs of mean
 Actual value 0.231: Average = 0.23771 +- 0.02513; Expected value within 0.267 stdevs of mean
 Actual value 0.34: Average = 0.30826 +- 0.02299; Expected value within 1.381 stdevs of mean

- MAE: 0.0119977109134197
- Min STD: 0.0039281165
- Avg STD: 0.011441505
- Max STD: 0.025132488
- Overlap Ratio: 1.0 (acc 5e-3)
- Pearson Coefficient: 0.985809870908463

94.0.0.3 briar9222: $\rho = 0.25$,
 $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.231, 0.340\}$, **30 epochs**,
36000 (0.2) snapshots



Prediction means and standard deviations.

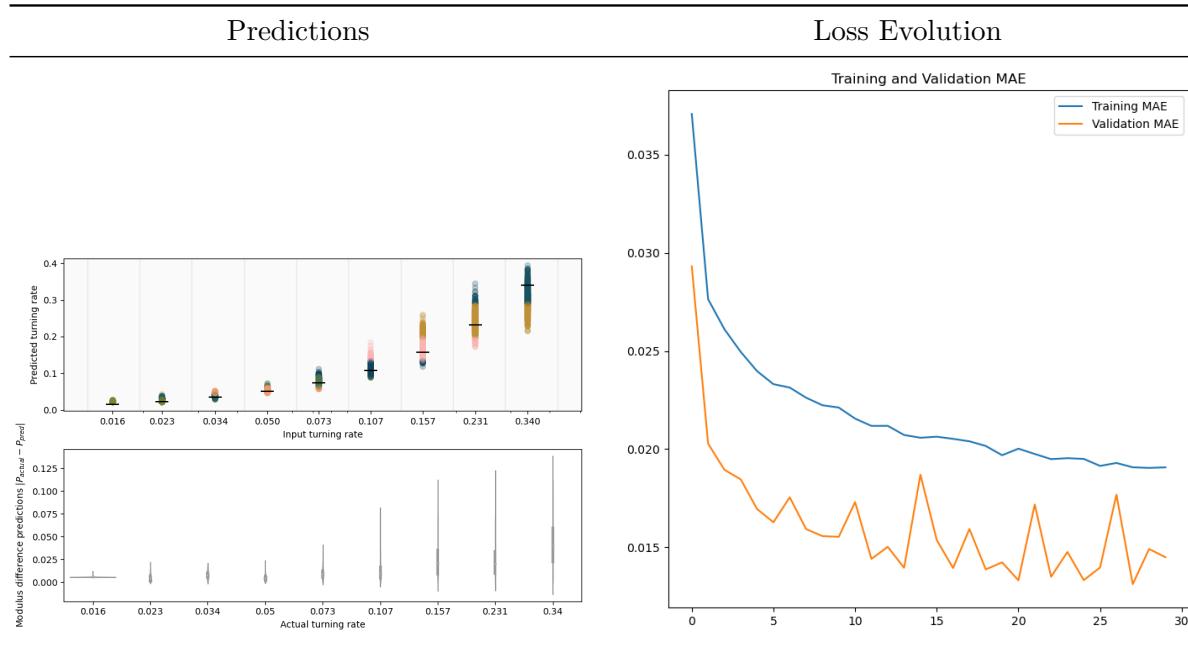
Actual value 0.016: Average = 0.01687 +- 0.00000; Expected value within 469738.408 stdevs of mean
 Actual value 0.023: Average = 0.01806 +- 0.00219; Expected value within 2.259 stdevs of mean
 Actual value 0.034: Average = 0.03374 +- 0.00421; Expected value within 0.061 stdevs of mean
 Actual value 0.05: Average = 0.05006 +- 0.00412; Expected value within 0.015 stdevs of mean
 Actual value 0.073: Average = 0.06890 +- 0.00514; Expected value within 0.798 stdevs of mean
 Actual value 0.107: Average = 0.10614 +- 0.00696; Expected value within 0.124 stdevs of mean
 Actual value 0.157: Average = 0.15741 +- 0.01823; Expected value within 0.022 stdevs of mean
 Actual value 0.231: Average = 0.22961 +- 0.02319; Expected value within 0.060 stdevs of mean
 Actual value 0.34: Average = 0.30098 +- 0.02491; Expected value within 1.566 stdevs of mean

- MAE: 0.0109049286693335
- Min STD: 0.0000000018626451
- Avg STD: 0.009882737
- Max STD: 0.024909819
- Overlap Ratio: 1.0 (acc 5e-3)
- Pearson Coefficient: 0.986715154338279

95 8. Different Density Comparison (Omitting Highest Tumbling Rate)

Omitting the regime which exhibits non-clustering behaviour for both $\rho = 0.15$ and $\rho = 0.25$, we can even better highlight the difference between the two densities in generating predictions. We can better yet contrast both of them to the $\rho = 0.35$ case, as was briefly done near the beginning of this Week.

95.0.0.1 stamp5111: $\rho = 0.15$,
 $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.231, 0.340\}$, **30 epochs**,
36000 (0.2) snapshots



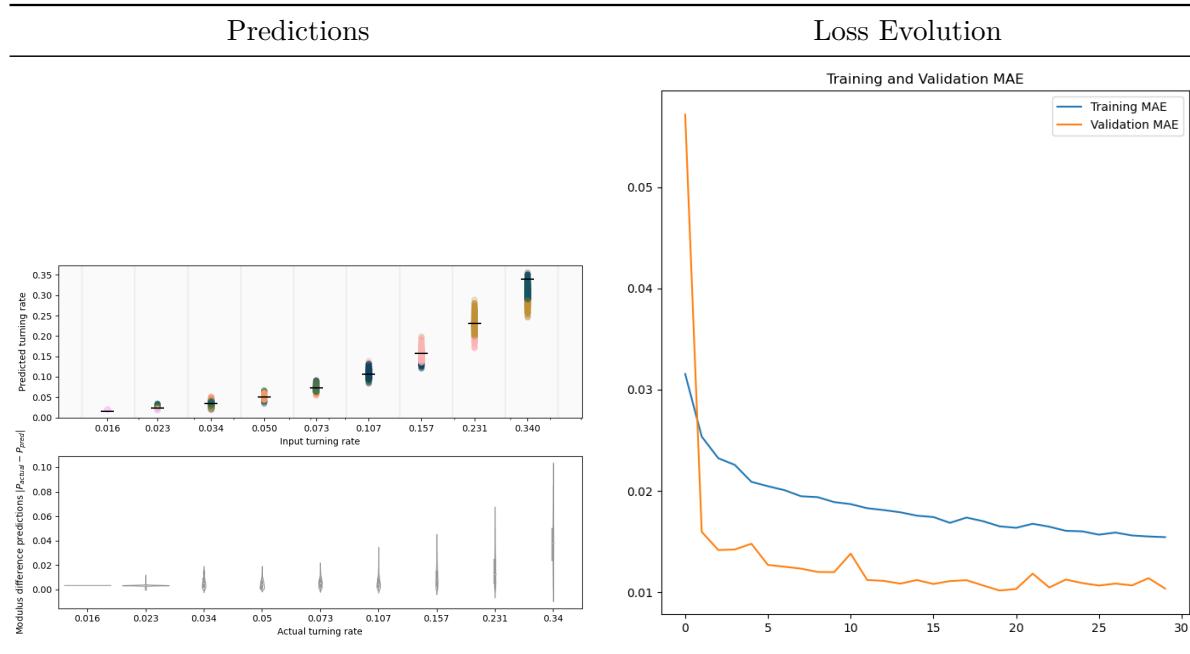
Prediction means and standard deviations.

Actual value 0.016: Average = 0.02170 +- 0.00062; Expected value within 9.130 stdevs of mean
 Actual value 0.023: Average = 0.02707 +- 0.00378; Expected value within 1.077 stdevs of mean

Actual value 0.034: Average = 0.04148 +- 0.00404; Expected value within 1.850 stdevs of mean
 Actual value 0.05: Average = 0.05424 +- 0.00314; Expected value within 1.349 stdevs of mean
 Actual value 0.073: Average = 0.08061 +- 0.00730; Expected value within 1.041 stdevs of mean
 Actual value 0.107: Average = 0.11660 +- 0.01217; Expected value within 0.789 stdevs of mean
 Actual value 0.157: Average = 0.17568 +- 0.02403; Expected value within 0.777 stdevs of mean
 Actual value 0.231: Average = 0.24407 +- 0.02617; Expected value within 0.499 stdevs of mean
 Actual value 0.34: Average = 0.30026 +- 0.02976; Expected value within 1.336 stdevs of mean

- MAE: 0.0144910635426641
- Min STD: 0.000623846
- Avg STD: 0.012336487
- Max STD: 0.029758396
- Overlap Ratio: 0.89 (acc 5e-3)
- Pearson Coefficient: 0.976878137376062

95.0.0.2 ripple9010: $\rho = 0.35$,
 $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.157, 0.231, 0.340\}$, **30 epochs**,
36000 (0.2) snapshots



Prediction means and standard deviations.

Actual value 0.016: Average = 0.01950 +- 0.00000; Expected value within 1879028.408 stdevs of mean
 Actual value 0.023: Average = 0.02014 +- 0.00208; Expected value within 1.375 stdevs of mean

Actual value 0.034: Average = 0.03159 +- 0.00694; Expected value within 0.347 stdevs of mean
 Actual value 0.05: Average = 0.05213 +- 0.00506; Expected value within 0.420 stdevs of mean
 Actual value 0.073: Average = 0.07047 +- 0.00630; Expected value within 0.401 stdevs of mean
 Actual value 0.107: Average = 0.10682 +- 0.00768; Expected value within 0.024 stdevs of mean
 Actual value 0.157: Average = 0.15306 +- 0.01171; Expected value within 0.336 stdevs of mean
 Actual value 0.231: Average = 0.22579 +- 0.01955; Expected value within 0.266 stdevs of mean
 Actual value 0.34: Average = 0.30243 +- 0.01907; Expected value within 1.971 stdevs of mean

- MAE: 0.010349047370255
- Min STD: 0.0000000018626451
- Avg STD: 0.008708556
- Max STD: 0.019550083
- Overlap Ratio: 1.0 (acc 5e-3)
- Pearson Coefficient: 0.991194498813799

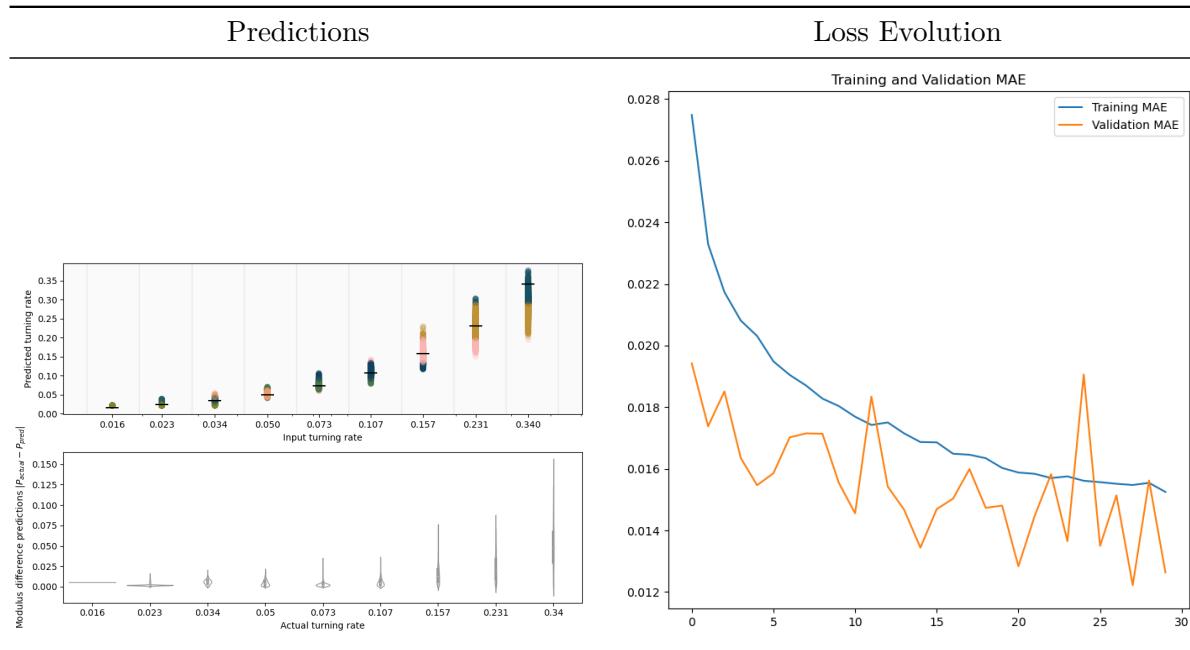
95.0.1 Comparison

Parameter	ripple9010 ($\rho = 0.35$)	briar9222 ($\rho = 0.25$)	stamp5111 ($\rho = 0.15$)
MAE	0.010349	0.010905	0.014491
Avg STD	0.008708556	0.009882737	0.012336487
Max STD	0.019550083	0.024909819	0.029758396
Overlap Ratio	1.0	1.0	0.89

Expected Values	0.016	0.023	0.034	0.050	0.073	0.107	0.157	0.231	0.34
$\rho = 0.35$	0.01950 ± 0.02014 ± 0.03159 ± 0.05213 ± 0.07047 ± 0.10682 ± 0.15306 ± 0.22579 ± 0.30243 ± 0.00000 0.00208 0.00694 0.00506 0.00630 0.00768 0.01171 0.01955 0.01907								
$\rho = 0.25$	0.01687 ± 0.01806 ± 0.03374 ± 0.05006 ± 0.06890 ± 0.10614 ± 0.15741 ± 0.22961 ± 0.30098 ± 0.00000 0.00219 0.00421 0.00412 0.00514 0.00696 0.01823 0.02319 0.02491								
$\rho = 0.15$	0.02170 ± 0.02707 ± 0.04148 ± 0.05424 ± 0.08061 ± 0.11660 ± 0.17568 ± 0.24407 ± 0.30026 ± 0.00062 0.00378 0.00404 0.00314 0.00730 0.01217 0.02403 0.02617 0.02976								

96 9. Multiple Nearby Densities

96.0.0.1 keter3955:



Prediction means and standard deviations.

Actual value 0.016: Average = 0.02119 +- 0.00000; Expected value within 1392564.204 stdevs of mean
 Actual value 0.023: Average = 0.02253 +- 0.00321; Expected value within 0.147 stdevs of mean
 Actual value 0.034: Average = 0.03882 +- 0.00464; Expected value within 1.038 stdevs of mean
 Actual value 0.05: Average = 0.05392 +- 0.00443; Expected value within 0.886 stdevs of mean
 Actual value 0.073: Average = 0.07471 +- 0.00426; Expected value within 0.403 stdevs of mean
 Actual value 0.107: Average = 0.11074 +- 0.00665; Expected value within 0.563 stdevs of mean
 Actual value 0.157: Average = 0.15046 +- 0.01607; Expected value within 0.407 stdevs of mean
 Actual value 0.231: Average = 0.21991 +- 0.02594; Expected value within 0.427 stdevs of mean
 Actual value 0.34: Average = 0.29158 +- 0.02782; Expected value within 1.740 stdevs of mean

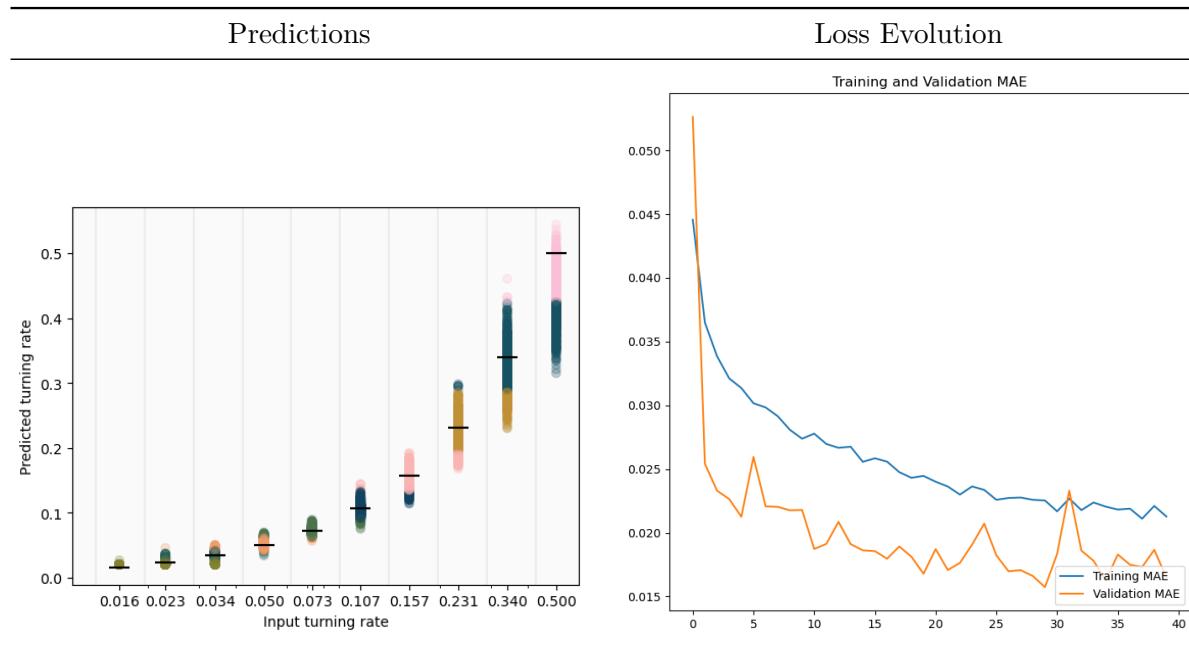
- MAE: 0.0126360701397061
- Min STD: 0.0000000037252903

- Avg STD: 0.010336722
- Max STD: 0.027823886
- Overlap Ratio: 0.89 (acc 5e-3)
- Pearson Coefficient: 0.984687332725716

97 10. Epoch Numbers

We have been mostly running 30 epochs for each CNN model. We can see a downward shift in all the MAE evolutions above, with a potential indication that more epochs might decrease it further and thus yield even better results. Below is a model ran for 40 epochs, as it compares to reverb3164, outlined above at the beginning.

97.0.0.1 remnant3992: $\rho = 0.25$,
 $P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.157, 0.231, 0.231, 0.340, 0.340, 0.5000\}$, **40 epochs, 40000 (0.2) snapshots**



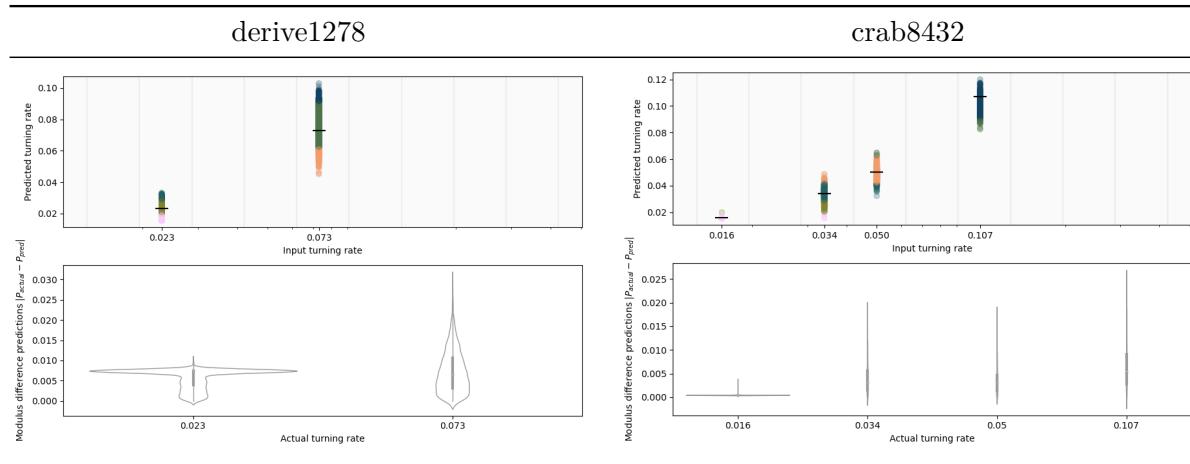
- MAE: 0.0164859797805548
- Min STD: 0.00021988283
- Avg STD: 0.013953483
- Max STD: 0.0383877
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.987502201574785

97.0.1 Comparison

Parameter	reverb3164 (30 epochs)	remnant3992 (40 epochs)
MAE	0.017795	0.016486
Avg STD	0.011430472	0.013953483
Max STD	0.030022161	0.0383877
Overlap Ratio	0.7	1.0

98 11. Monochrome Interpolation (Low Tumbling Rates)

98.0.0.1 derive1278 (crab8432) $\rho = 0.25$ $P_{val} \in \{0.023, 0.073\}$,
 $P_{train} \in \{0.016, 0.034, 0.050, 0.107\}$



Prediction means and standard deviations for derive1278.

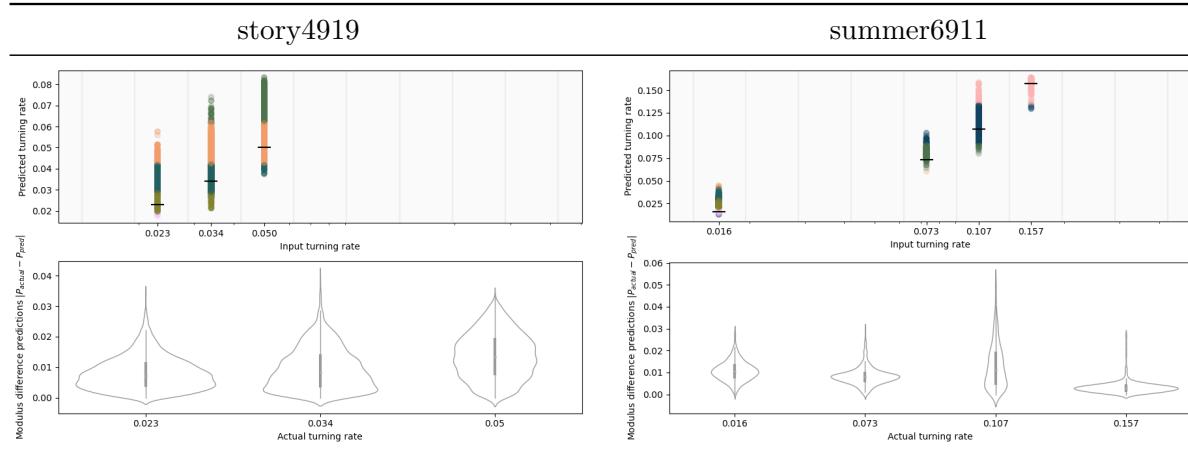
Actual value 0.023: Average = 0.01815 +- 0.00382; Expected value within 1.270 stdevs of mean
 Actual value 0.073: Average = 0.07531 +- 0.00863; Expected value within 0.268 stdevs of mean

- MAE: -
- Min STD: 0.0038166833
- Avg STD: 0.0062226485
- Max STD: 0.008628614
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.973814798645944

Parameter	derive1278	crab8432
Avg STD	0.0062226485	0.0037065577
Max STD	0.008628614	0.005531624
Overlap Ratio	1.0	1.0

Parameter	derive1278	crab8432
Pearson Coefficient	0.973815	0.991052

98.0.0.2 story4919 (summer6911): $\rho = 0.25$, $P_{val} \in \{0.023, 0.034, 0.050\}$, $P_{train} \in \{0.016, 0.073, 0.107, 0.157\}$



Prediction means and standard deviations for story4919.

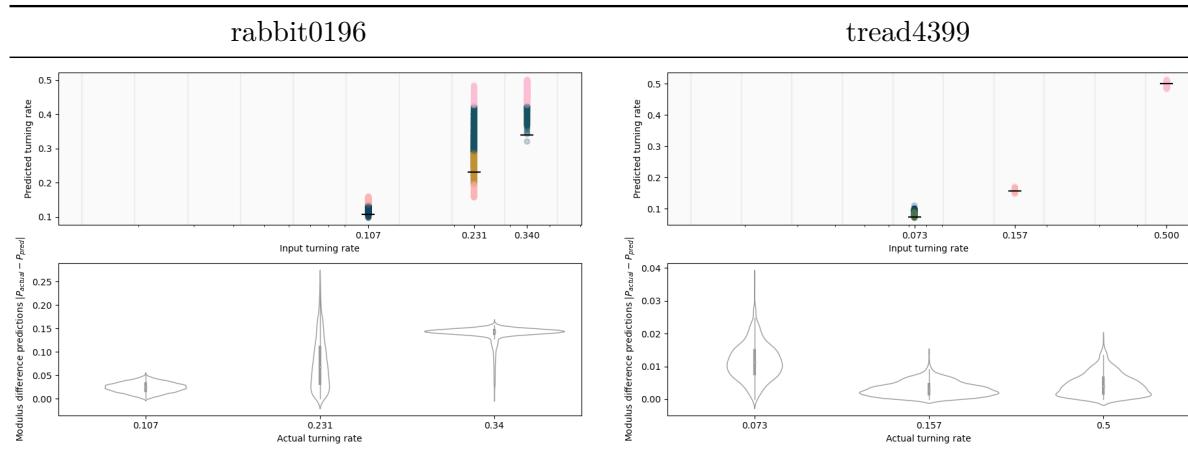
Actual value 0.023: Average = 0.03101 +- 0.00552; Expected value within 1.452 stdevs of mean
 Actual value 0.034: Average = 0.04259 +- 0.00784; Expected value within 1.096 stdevs of mean
 Actual value 0.05: Average = 0.06339 +- 0.00805; Expected value within 1.663 stdevs of mean

- MAE: -
- Min STD: 0.005518279
- Avg STD: 0.0071368576
- Max STD: 0.00805434
- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.878673132485899

Parameter	story4919	summer6911
Avg STD	0.0071368576	0.0066630687
Max STD	0.00805434	0.013605628
Overlap Ratio	1.0	1.0
Pearson Coefficient	0.878673	0.986987

99 12. Monochrome Interpolation (High Tumbling Rates)

99.0.0.1 rabbit0196 (tread4399): $\rho = 0.25$, $P_{val} \in \{0.107, 0.231, 0.340\}$,
 $P_{train} \in \{0.073, 0.157, 0.5\}$



Prediction means and standard deviations for rabbit0196.

Actual value 0.107: Average = 0.13162 +- 0.01054; Expected value within 2.336 stdevs of mean
 Actual value 0.231: Average = 0.29793 +- 0.06501; Expected value within 1.030 stdevs of mean
 Actual value 0.34: Average = 0.47849 +- 0.01906; Expected value within 7.268 stdevs of mean

Prediction means and standard deviations for tread4399.

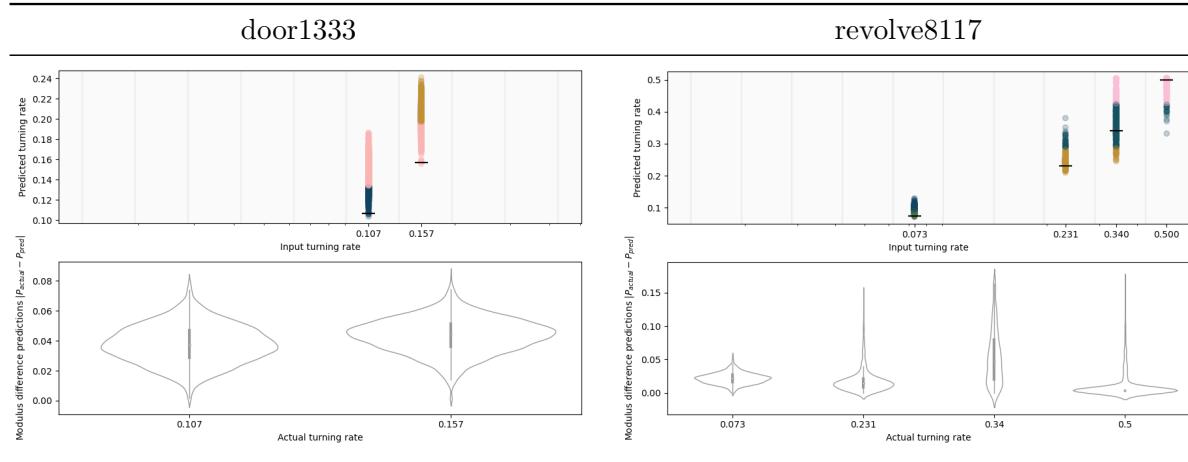
Actual value 0.073: Average = 0.08449 +- 0.00540; Expected value within 2.128 stdevs of mean
 Actual value 0.157: Average = 0.15972 +- 0.00279; Expected value within 0.973 stdevs of mean
 Actual value 0.5: Average = 0.49756 +- 0.00513; Expected value within 0.475 stdevs of mean

- MAE: -
- Min STD: 0.010536376
- Avg STD: 0.031534202
- Max STD: 0.06501088

- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.961320568504786

Parameter	rabbit0196	tread4399
Avg STD	0.031534202	0.004439787
Max STD	0.06501088	0.0054007815
Overlap Ratio	1.0	1.0
Pearson Coefficient	0.961321	0.999564

99.0.0.2 door1333 (revolve8117): $\rho = 0.25$, $P_{val} \in \{0.107, 0.157\}$, $P_{train} \in \{0.073, 0.231, 0.340, 0.500\}$



Prediction means and standard deviations for door1333.

Actual value 0.107: Average = 0.14482 +- 0.01286; Expected value within 2.942 stdevs of mean
 Actual value 0.157: Average = 0.20067 +- 0.01161; Expected value within 3.761 stdevs of mean

Prediction means and standard deviations for revolve8117.

Actual value 0.073: Average = 0.09487 +- 0.00837; Expected value within 2.613 stdevs of mean
 Actual value 0.231: Average = 0.24794 +- 0.01834; Expected value within 0.924 stdevs of mean
 Actual value 0.34: Average = 0.37743 +- 0.05689; Expected value within 0.658 stdevs of mean
 Actual value 0.5: Average = 0.49567 +- 0.02125; Expected value within 0.204 stdevs of mean

- MAE: -
- Min STD: 0.0116104465
- Avg STD: 0.0122337025
- Max STD: 0.012856959

- Overlap Ratio: 1 (acc 5e-3)
- Pearson Coefficient: 0.915762512590914

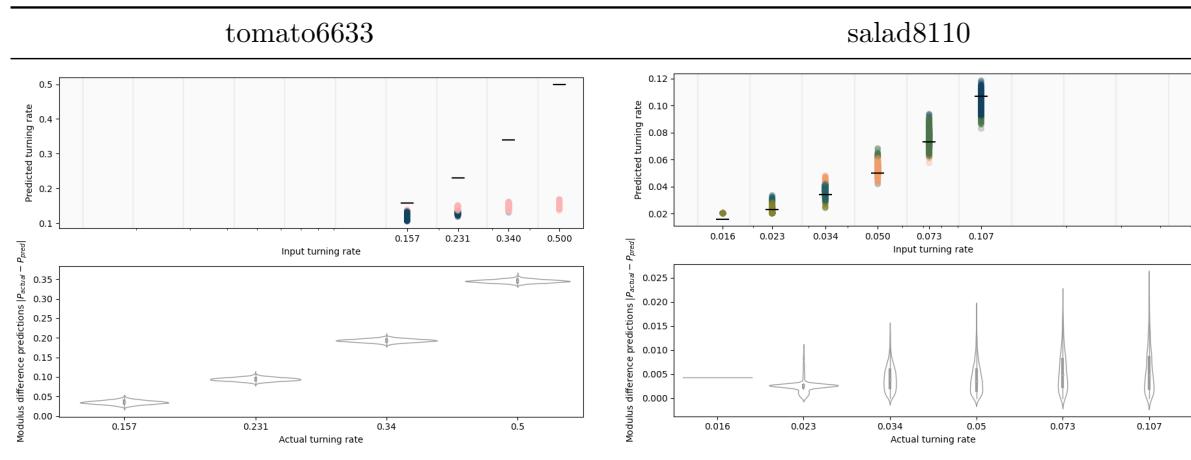
Parameter	door1333	revolve8117
Avg STD	0.0122337025	0.026211156
Max STD	0.012856959	0.05688635
Overlap Ratio	1.0	1.0
Pearson Coefficient	0.915763	0.974905

100 13. Monochrome Extrapolation

salmon9100: extrapolate downwards

100.0.0.1 tomato6633 (salad8110): $\rho = 0.25$, $P_{val} \in \{0.157, 0.231, 0.340, 0.500\}$,
 $P_{train} \in \{0.016, 0.023, 0.034, 0.05, 0.073, 0.107\}$

Extrapolating upwards.



Prediction means and standard deviations for tomato6633.

Actual value 0.157: Average = 0.12189 +- 0.00509; Expected value within 6.904 stdevs of mean

Actual value 0.231: Average = 0.13669 +- 0.00462; Expected value within 20.414 stdevs of mean

Actual value 0.34: Average = 0.14682 +- 0.00438; Expected value within 44.096 stdevs of mean

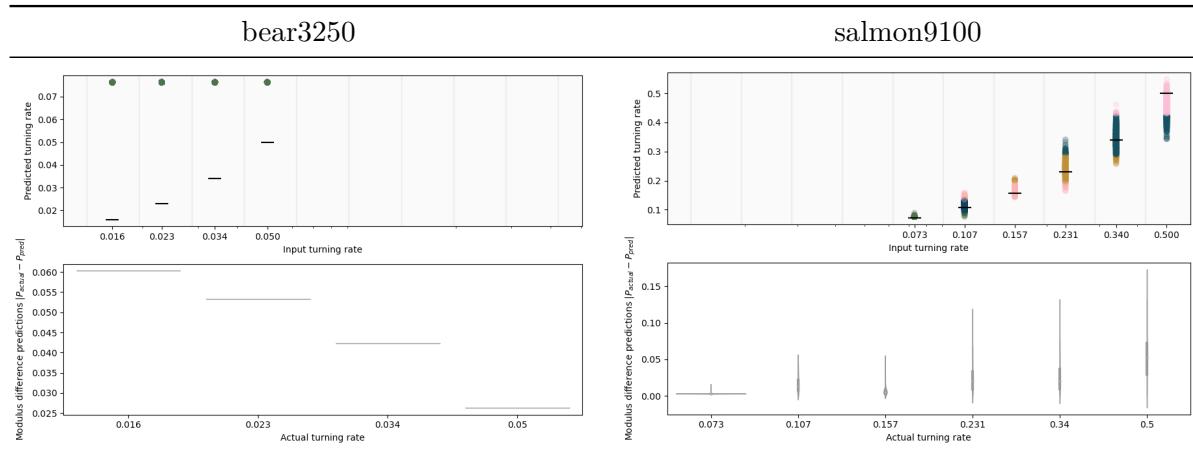
Actual value 0.5: Average = 0.15471 +- 0.00434; Expected value within 79.595 stdevs of mean

- MAE: -
- Min STD: 0.004338048
- Avg STD: 0.0046061105
- Max STD: 0.0050855814
- Overlap Ratio: 0 (acc 5e-3)
- Pearson Coefficient: 0.891692673465758

Parameter	tomato6633	salad8110
Avg STD	0.0046061105	0.0035765618
Max STD	0.0050855814	0.0058416952
Overlap Ratio	0.0	1.0
Pearson Coefficient	0.891693	0.986550

100.0.0.2 bear3250 (salmon9100): $\rho = 0.25$, \$P_{\text{val}} \{ \}

Extrapolating downwards.



Prediction means and standard deviations for bear3250.

Actual value 0.016: Average = 0.07631 +- 0.00000; Expected value within 8094958.352 stdevs of mean
 Actual value 0.023: Average = 0.07631 +- 0.00000; Expected value within 7155434.256 stdevs of mean
 Actual value 0.034: Average = 0.07631 +- 0.00000; Expected value within 5679039.248 stdevs of mean
 Actual value 0.05: Average = 0.07631 +- 0.00000; Expected value within 3531555.600 stdevs of mean

Prediction means and standard deviations for salmon9100.

Actual value 0.073: Average = 0.07637 +- 0.00064; Expected value within 5.309 stdevs of mean
 Actual value 0.107: Average = 0.11958 +- 0.01252; Expected value within 1.005 stdevs of mean
 Actual value 0.157: Average = 0.16357 +- 0.00663; Expected value within 0.990 stdevs of mean
 Actual value 0.231: Average = 0.24277 +- 0.02668; Expected value within 0.441 stdevs of mean
 Actual value 0.34: Average = 0.34620 +- 0.03178; Expected value within 0.195 stdevs of mean
 Actual value 0.5: Average = 0.44783 +- 0.03338; Expected value within 1.563 stdevs of mean

MAE: -

Min STD: 0.000000007450581

Avg STD: 0.000000007450581

Max STD: 0.000000007450581

Overlap Ratio: 0 (acc 5e-3)

Pearson Coefficient: nan

101 Week 19

102 0. Table of Contents

1. Introduction
2. Analysis Update
3. Higher Number Interpolation
4. Preliminary Confusion Code
5. Improved Full Tumbling Rate Analysis (more rolling, more epochs)
6. Running Same Parameters Multiple Times
7. Auto-Correlation Function

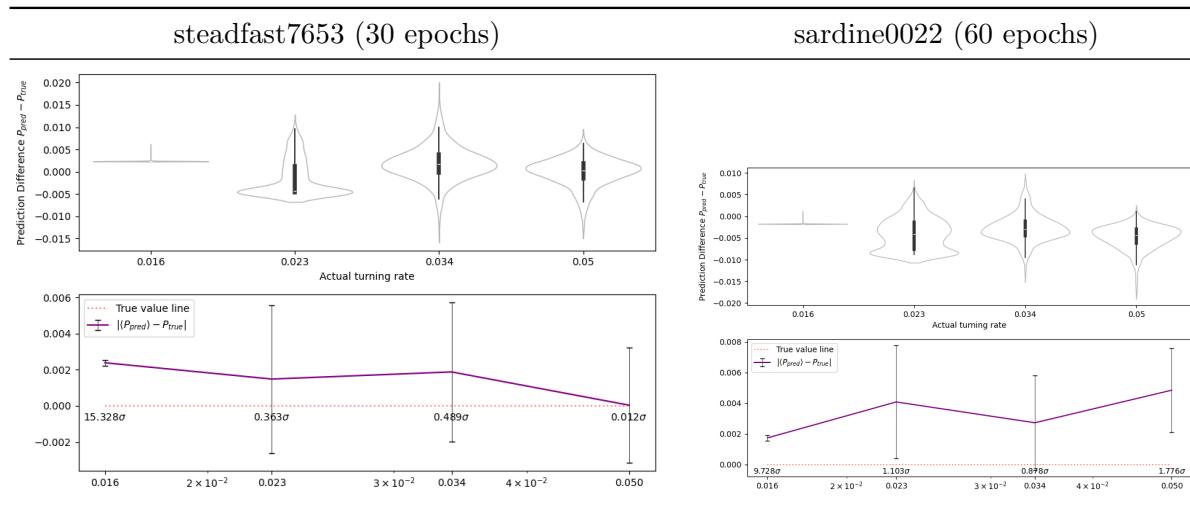
103 1. Introduction

This week consists of exploring higher range parameters in order to consolidate the results in previous weeks.

104 2. Analysis Update

We have finally fixed GPU tensorflow, which greatly accelerates model training.

104.0.0.1 steadfast7653: $P_t \in \{0.016, 0.023, 0.034, 0.050\}$, $\rho = 0.25$, **30 epochs** ;
sardine0022: same, same, **60 epochs**



Prediction means and standard deviations for steadfast7653.

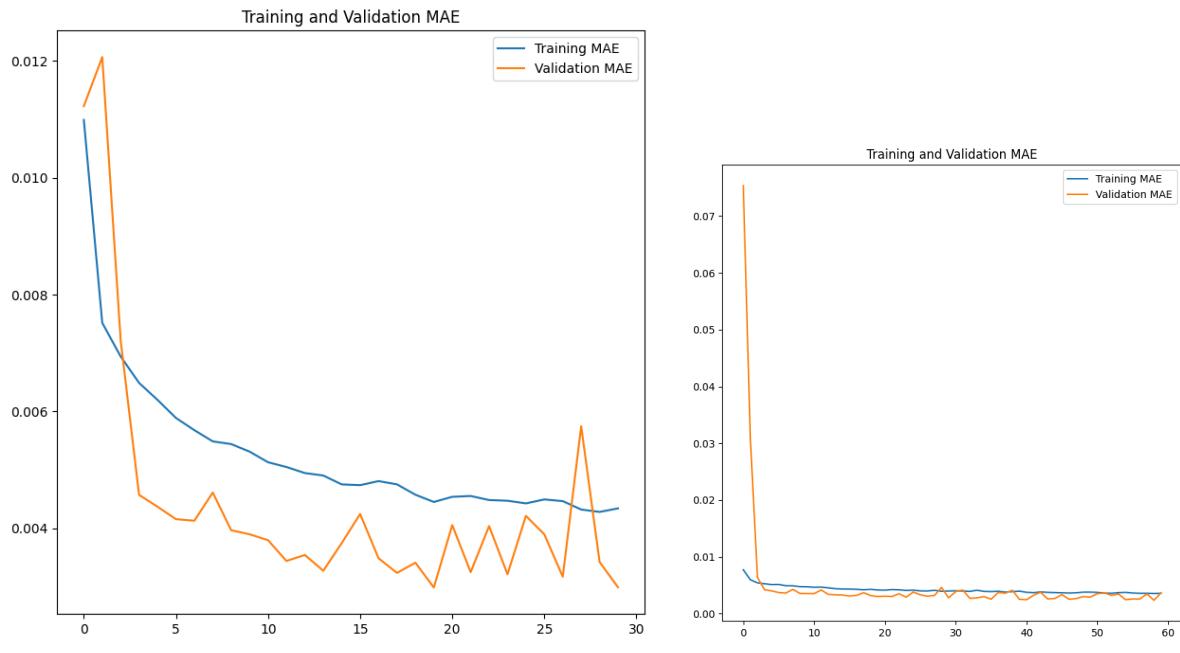
Actual value 0.016: Average = 0.01838 +- 0.00016; Expected value within 15.328 stdevs of mean
 Actual value 0.023: Average = 0.02152 +- 0.00409; Expected value within 0.363 stdevs of mean
 Actual value 0.034: Average = 0.03588 +- 0.00384; Expected value within 0.489 stdevs of mean
 Actual value 0.05: Average = 0.04996 +- 0.00318; Expected value within 0.012 stdevs of mean

Prediction means and standard deviations for sardine0022.

Actual value 0.016: Average = 0.01427 +- 0.00018; Expected value within 9.728 stdevs of mean
 Actual value 0.023: Average = 0.01892 +- 0.00370; Expected value within 1.103 stdevs of mean
 Actual value 0.034: Average = 0.03128 +- 0.00310; Expected value within 0.878 stdevs of mean
 Actual value 0.05: Average = 0.04516 +- 0.00273; Expected value within 1.776 stdevs of mean

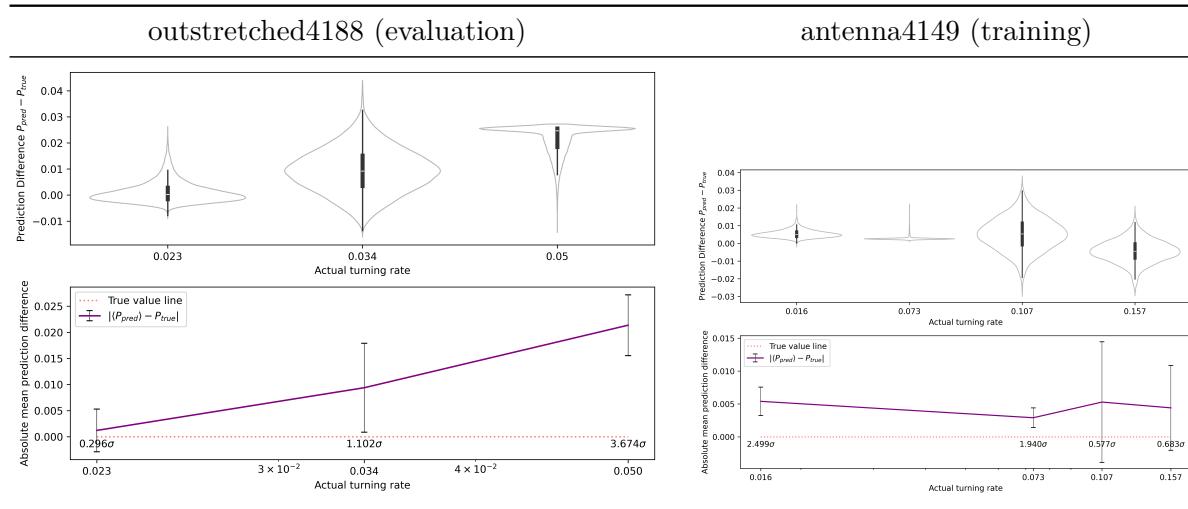
steadfast7653 (30 epochs)

sardine0022 (60 epochs)



105 3. Higher Number Interpolation

105.0.0.1 outstretched4188 (antenna4149): $P_{val} \in \{0.023, 0.034, 0.050\}$,
 $P_{train} \in \{0.016, 0.073, 0.107, 0.157\}$, $\rho = 0.25$, **30 epochs**



Prediction means and standard deviations for outstretched4188.

Actual value 0.023: Average = 0.02421 +- 0.00409; Expected value within 0.296 stdevs of mean

Actual value 0.034: Average = 0.04339 +- 0.00852; Expected value within 1.102 stdevs of mean

Actual value 0.05: Average = 0.07137 +- 0.00582; Expected value within 3.674 stdevs of mean

Prediction means and standard deviations for antenna4149.

Actual value 0.016: Average = 0.02139 +- 0.00216; Expected value within 2.499 stdevs of mean

Actual value 0.073: Average = 0.07590 +- 0.00150; Expected value within 1.940 stdevs of mean

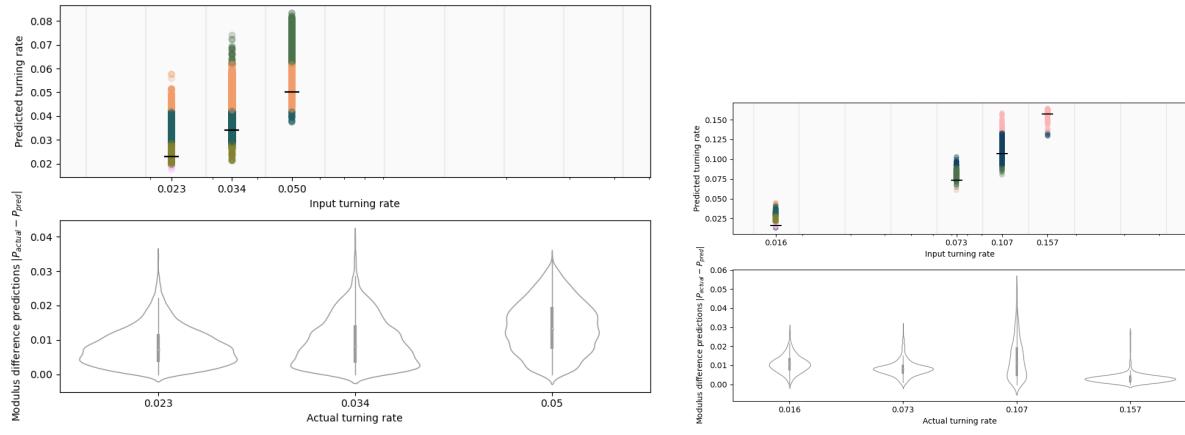
Actual value 0.107: Average = 0.11229 +- 0.00916; Expected value within 0.577 stdevs of mean

Actual value 0.157: Average = 0.15259 +- 0.00645; Expected value within 0.683 stdevs of mean

Compare to equivalent case from [last week](#):

story4919 (evaluation)

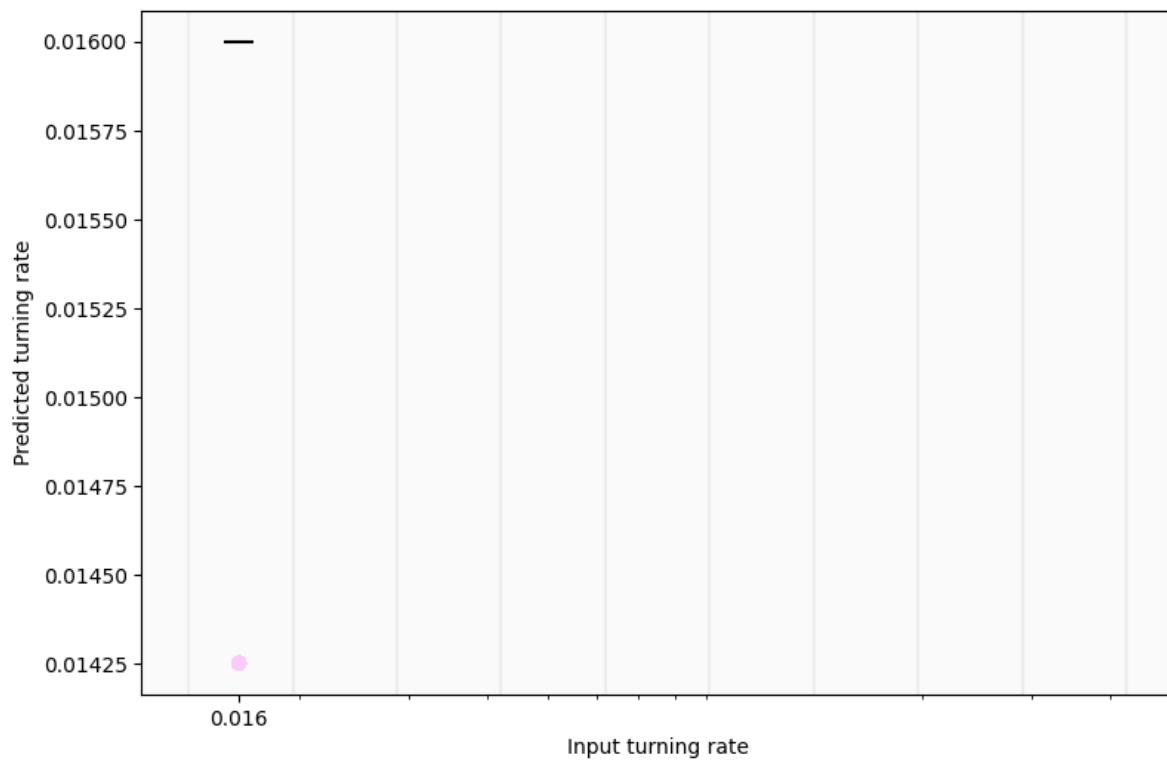
summer6911 (training)



106 4. Preliminary Confusion Code

```
img = fin[key] [:]
for i in range(128):
    for j in range(128):
        if img[i,j] > 0:
            chance=float(random.randint(1,100))/100
            if chance <= 0.05:
                while (True):
                    x = random.randint(1,4)
                    if x != img[i,j]:
                        break
                img[i,j] = x
img = img.astype(float)/4.0
```

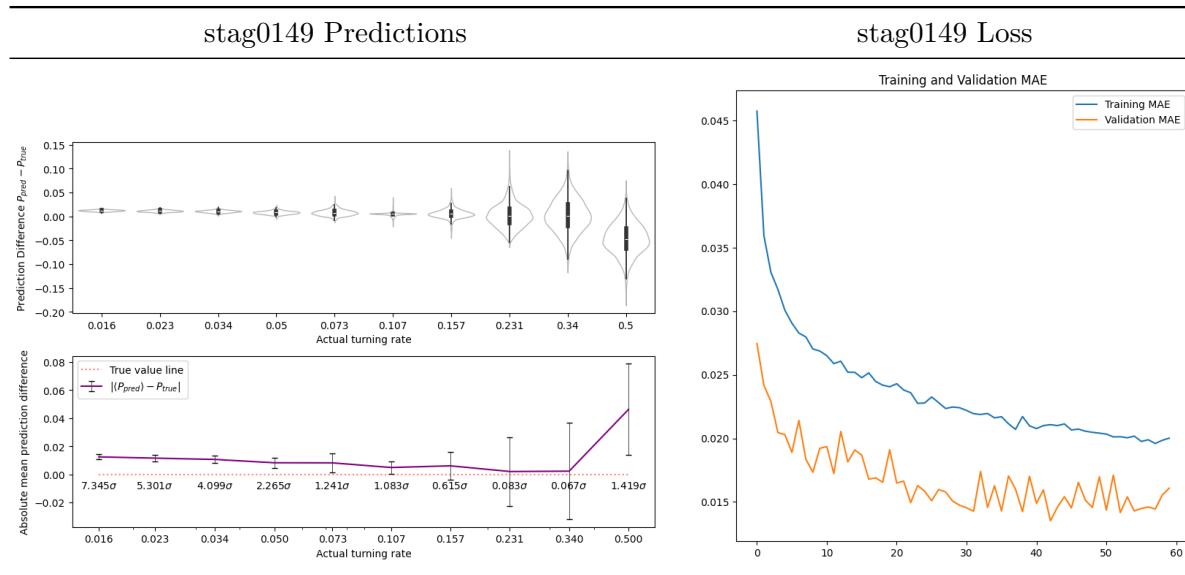
106.0.0.1 less5622: $P_{tumbke} = 0.016$, $\rho = 0.25$, **60 epochs, 4000 (0.2) snapshots**



107 5. Improved Full Tumbling Rate Analysis (more rolling, more epochs)

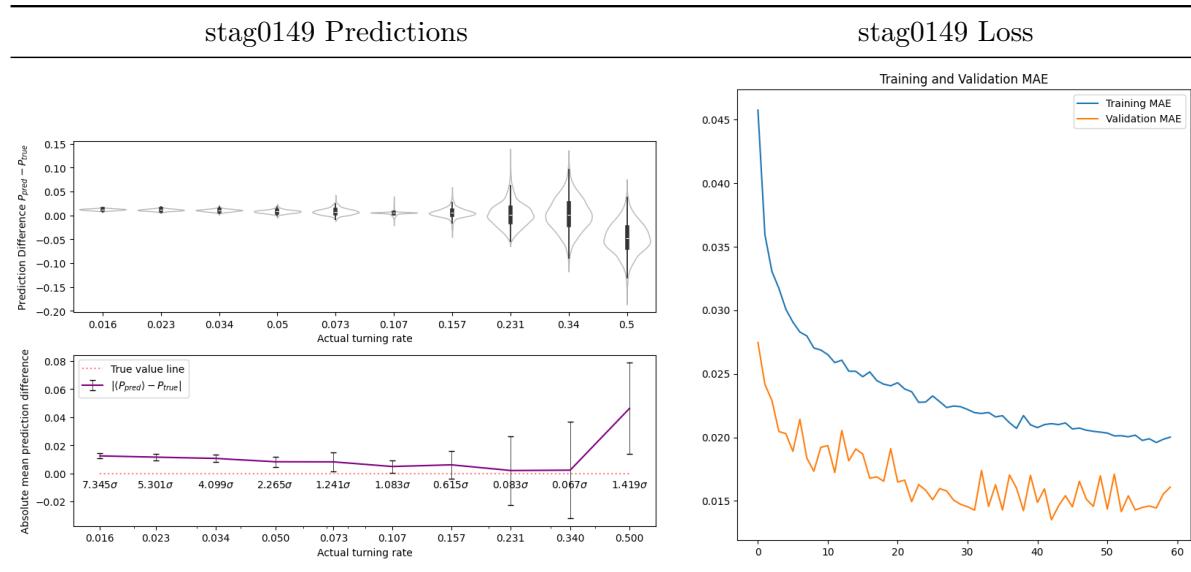
107.0.0.1 stag0149:

$P_{tumble} \in \{0.016, 0.023, 0.034, 0.050, 0.073, 0.107, 0.157, 0.231, 0.340, 0.500\}$,
 $\rho = 0.25$, **60 epochs, 80000 (0.2) snapshots**



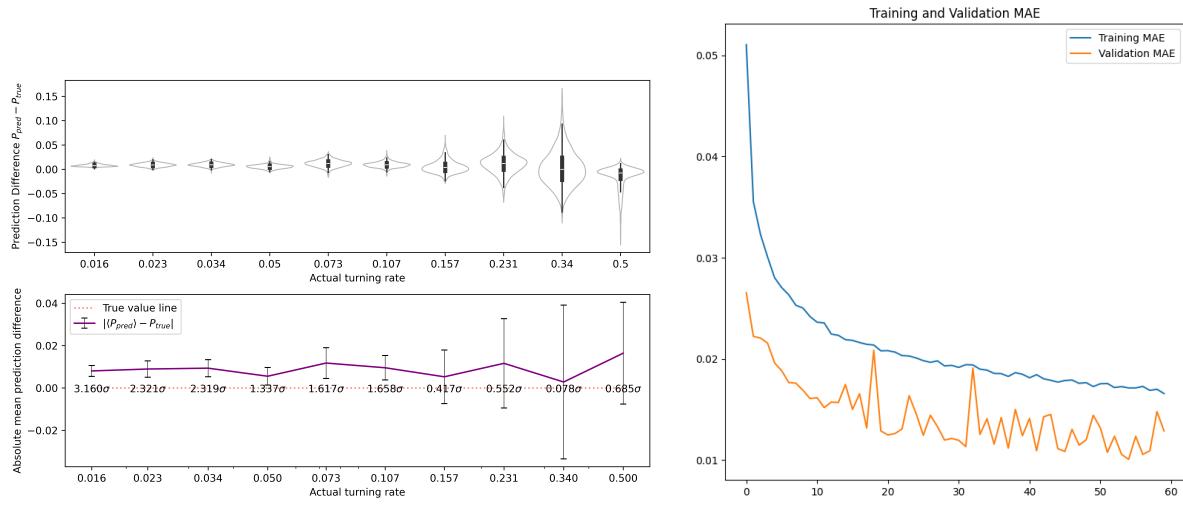
108 6. Running Same Parameters Multiple Times

stag0149, leaflet5121 and branch3151 are different runs of the same parameters (see Section 5).



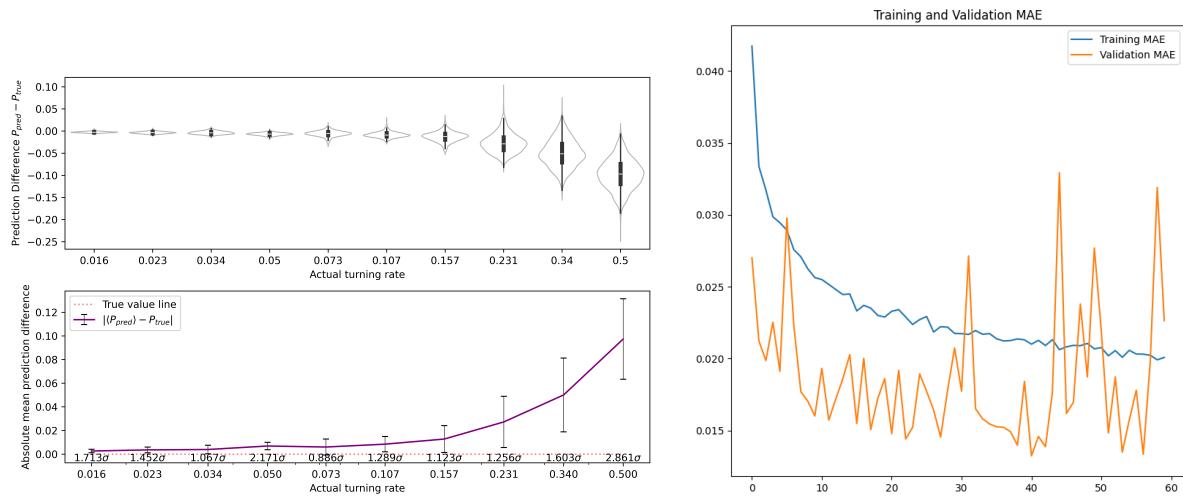
leaflet5121 Predictions

leaflet5121 Loss



branch3151 Predictions

branch3151 Loss



7. Auto-Correlation Function

```

def overlap(traj,i,j):
    N = traj[i][traj[i]>0].shape[0]
    return ((traj[i]>0)*(traj[j]>0)).sum()/N

def acf_analysis (tumble,density,lags,data):

```

```

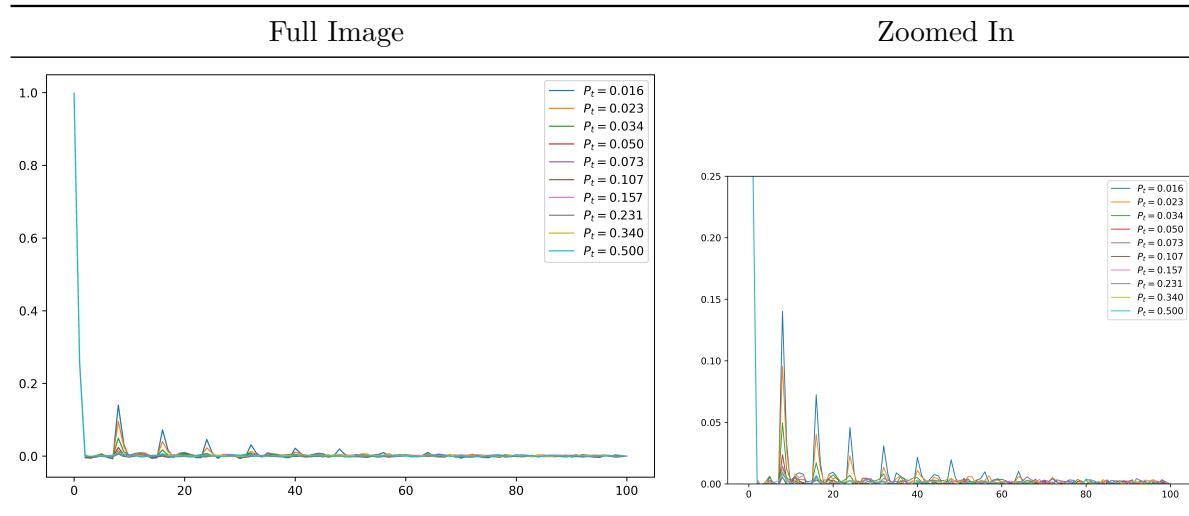
acf = []
for i in range(0,500,10):
    a = [overlap(data,i,i+lag) for lag in lags]
    acf.append(a)

acf = np.asarray(acf).mean(axis=0)
acf = acf-acf[-1]

acf/= acf.ptp()
return acf

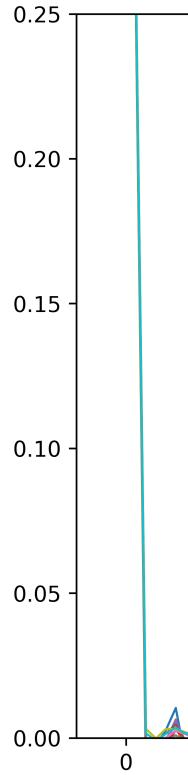
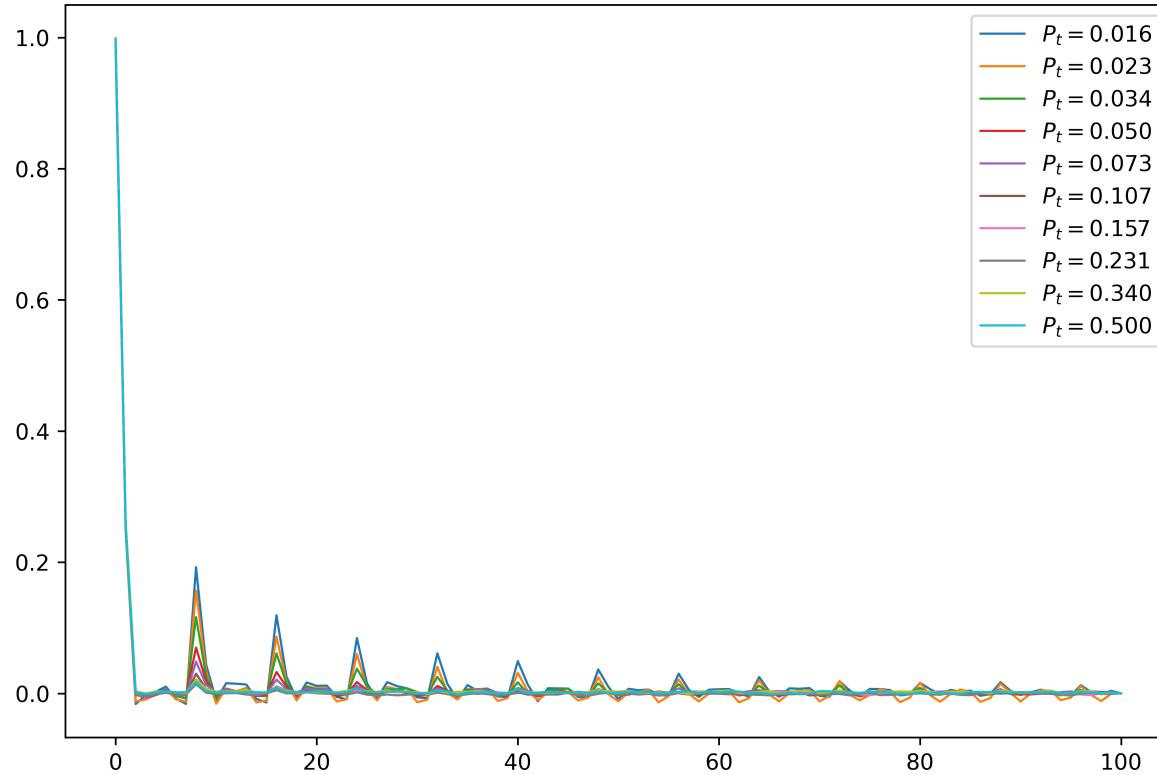
```

108.0.0.1 $\rho = 0.15$



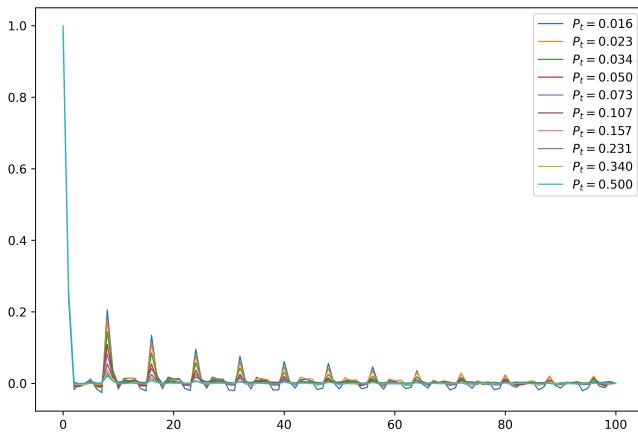
108.0.0.2 $\rho = 0.25$

Full Image

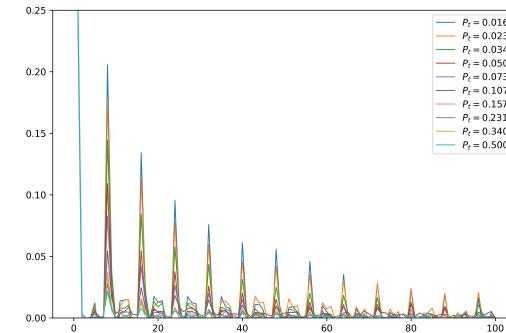


108.0.0.3 $\rho = 0.35$

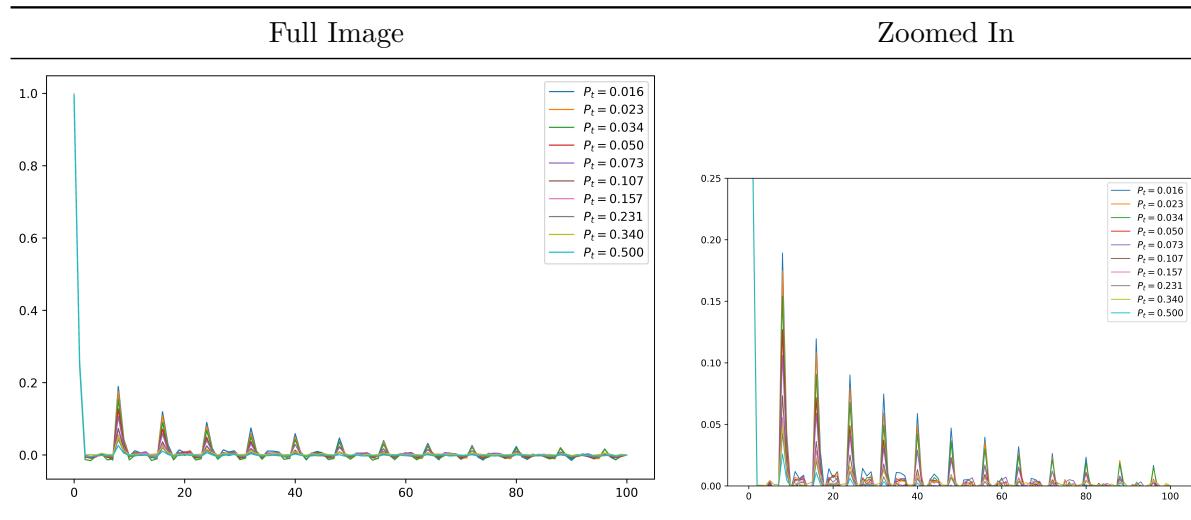
Full Image



Zoomed In



108.0.0.4 $\rho = 0.5$



109 Weeks 20-22

110 0. Table of Contents

1. Introduction
2. Cluster Periodicity and Revised Cluster Histograms
3. Cluster Number and Size Examination
4. Percolation Analysis
5. Cluster Edge Orientation
6. Feature Map Visualisation (Orientation)
7. Feature Map Visualisation (Monochrome)
8. Final Averaged Predictions
9. Interpolation
10. Extrapolation

111 1. Introduction

The aims of these last few weeks is to finish up experimental analysis, revise some mistakes in previous weeks, as well as prepare figures for the final report. These Weeks will also contain all the final results of analysis that were previously not fully included in the activity log (due to preliminary individual work which was shared at the end between lab partners).

112 2. Cluster Periodicity

This is an improved version of `cluster-analysis.py`, which can be found in the `src` folder. It uses a newer function which separates clusters while keeping boundary periodicity. The jupyter notebook can be found in `notebooks/cp_cluster_analysis.ipynb`.

112.0.1 Import Libraries

```
from scipy import ndimage
import numpy as np
import matplotlib.pyplot as plt
import h5py
import cmcrameri #for different cmaps
from mpl_toolkits.axes_grid1.inset_locator import inset_axes

import sys
sys.path.append('..')
from src.utils import get_cluster_labels, get_ds_iters
#from src.plot_utils import get_plot_configs
from src.training_utils import extract_floats
```

```
2024-04-11 12:40:58.351512: I tensorflow/core/util/port.cc:113] oneDNN custom operations are
2024-04-11 12:40:59.880218: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensor
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild Tensor
2024-04-11 12:41:01.535476: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W
```

112.0.2 Set Analysis Conditions

```
Pt=0.034 #tumble probability
rho=0.3 #particle density
file = ("..../data/no-rolling/dataset_tumble_{}_{}/density_{}.h5").format(Pt,rho) #change this

cmap1 = plt.get_cmap(name="cmc.lajolla") #cmap for first picture
```

```
cmap2 = plt.get_cmap(name="cmc.tokyoS") #cmap for second picture
```

112.0.3 Run Analysis and Plot

```
hfile = h5py.File(file,"r")

fig, (regplot, clusterplot, clusterhistogram) = plt.subplots (1,3,figsize=(9,3),width_ratios=[1,1,1])

iters = get_ds_iters(hfile.keys())
fig.suptitle(r"Cluster Analysis ($P_{tumble}=%.3f$; $\rho=%.3f$)" % (Pt,rho))

#plot regular graph
image = hfile[f"conf_{iters[-1]}"]
regplot.matshow(image,cmap=cmap1)

#plot cluster separation graph
kernel = [[0,1,0],
           [1,1,1],
           [0,1,0]]
labelled, _ = ndimage.label(image,structure=kernel)

# periodicity establishment
for y in range(labelled.shape[0]):
    if labelled[y, 0] > 0 and labelled[y, -1] > 0:
        labelled[labelled == labelled[y, -1]] = labelled[y, 0]
for y in range(labelled.shape[1]):
    if labelled[0, y] > 0 and labelled[-1, y] > 0:
        labelled[labelled == labelled[-1, y]] = labelled[0, y]

clusterplot.matshow(labelled,cmap=cmap2)

#plot histogram of obtained clusters
cluster_sizes = np.bincount(labelled.flatten())[1:]
cluster_sizes = np.delete(cluster_sizes,np.where(cluster_sizes==0))
print(cluster_sizes.min(),cluster_sizes.max())
bin_edges = np.logspace(np.log2(cluster_sizes.min()), np.log2(cluster_sizes.max()), 30, base=10)
#bin_edges = np.linspace(cluster_sizes.min(),cluster_sizes.max(),100)
counts, _ = np.histogram(cluster_sizes,bins=bin_edges,density=True)
clusterhistogram.grid(alpha=.4)
```

```

clusterhistogram.set_axisbelow(True)
clusterhistogram.scatter(bin_edges[:-1],counts,edgecolor=(0,0,0,1),facecolor=(0,0,0,.5))
clusterhistogram.set_yscale("log"), clusterhistogram.set_xscale("log")
clusterhistogram.set_xlabel("Cluster Size")

fig.colorbar(plt.cm.ScalarMappable(cmap=cmap1),ax=regplot)
fig.colorbar(plt.cm.ScalarMappable(cmap=cmap2),ax=clusterplot)
plt.show()

```

1 414

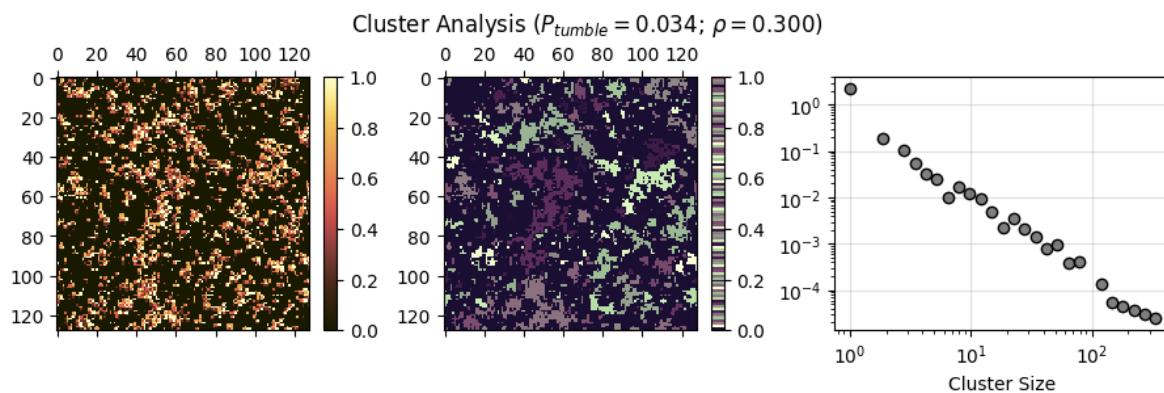


Figure 112.1: png

112.0.4 Array of Cluster Analysis

```

#plot_configs = get_plot_configs()
#plot_configs["xtick.labelsize"] = 8
#plot_configs["ytick.labelsize"] = 8
#plt.rcParams.update(plot_configs)
fig = plt.figure(figsize=(9 * 3 / 5, 9), constrained_layout=True)
gspec = fig.add_gridspec(5, 3, wspace=0.15, hspace=0.15)
cmap = "cmc.lajolla"
tumbles = [0.016,0.034,0.073,0.157,0.340]
densities = [0.15,0.25,0.35]
files = []

for tumble in tumbles:

```

```

for density in densities:
    files.append(f"..../data/no-rolling/dataset_tumble_{tumble:.3f}_{density}.h5")

ctr = 0
for idx in range(5):
    for jdx in range(3):
        axis = fig.add_subplot(gspec[idx, jdx], autoscale_on=False)
        inaxis = inset_axes(axis, width="100%", height="100%", loc="upper right", bbox_to_anchor=[0, 0, 1, 1], borderpad=0)

        with h5py.File(files[ctr], "r") as fin:
            key_list = list(fin.keys())
            iter_n = get_ds_iters(key_list)
            img = fin[f"conf_{iter_n[-2]}"]

            kernel = [[0, 1, 0],
                      [1, 1, 1],
                      [0, 1, 0]]
            labelled, _ = ndimage.label(img, structure=kernel)

            for y in range(labelled.shape[0]):
                if labelled[y, 0] > 0 and labelled[y, -1] > 0:
                    labelled[labelled == labelled[y, -1]] = labelled[y, 0]
            for y in range(labelled.shape[1]):
                if labelled[0, y] > 0 and labelled[-1, y] > 0:
                    labelled[labelled == labelled[-1, y]] = labelled[0, y]

            cluster_sizes = np.bincount(labelled.flatten())[1:]
            cluster_sizes = np.delete(cluster_sizes, np.where(cluster_sizes==0))
            print(cluster_sizes.min(), cluster_sizes.max())
            bin_edges = np.logspace(np.log2(cluster_sizes.min()), np.log2(cluster_sizes.max()))
            #bin_edges = np.linspace(cluster_sizes.min(), cluster_sizes.max(), 100)
            counts, _ = np.histogram(cluster_sizes, bins=bin_edges, density=True)
            axis.grid(alpha=.4)
            axis.set_axisbelow(True)
            axis.scatter(bin_edges[:-1], counts, edgecolor=(0, 0, 0, 1), facecolor=(0, 0, 0, .5))
            axis.set_yscale("log"), axis.set_xscale("log")
            axis.set_xlabel("Cluster Size")

            inaxis.matshow(img, cmap=cmap)
            inaxis.set_xticks([])
            inaxis.set_yticks([])


```

```

        axis.set_xlim((cluster_sizes.min()-20, cluster_sizes.max()+5))
        axis.set_ylim((-20, 200))
        axis.set_title(r"$P_{t}=%s$, $\rho = %s$ % (tumbles[idx],densities[jdx]))\n\n
        ctr += 1
fig.supylabel(r"Inverse Tumbling Probability, $P_{tumble}^{-1}$")
fig.supxlabel(r"Density, $\rho$")

```

```

1 132
1 224
1 593
1 70
1 165
1 434
1 42
1 104
1 254
1 20
1 51
1 115
1 9
1 37
1 82

```

```

/tmp/ipykernel_7618/475512023.py:56: UserWarning: Attempt to set non-positive xlim on a log-
    axis.set_xlim((cluster_sizes.min()-20, cluster_sizes.max()+5))
/tmp/ipykernel_7618/475512023.py:57: UserWarning: Attempt to set non-positive ylim on a log-
    axis.set_ylim((-20, 200))

```

```
Text(0.5, 0.01, 'Density, $\rho$')
```

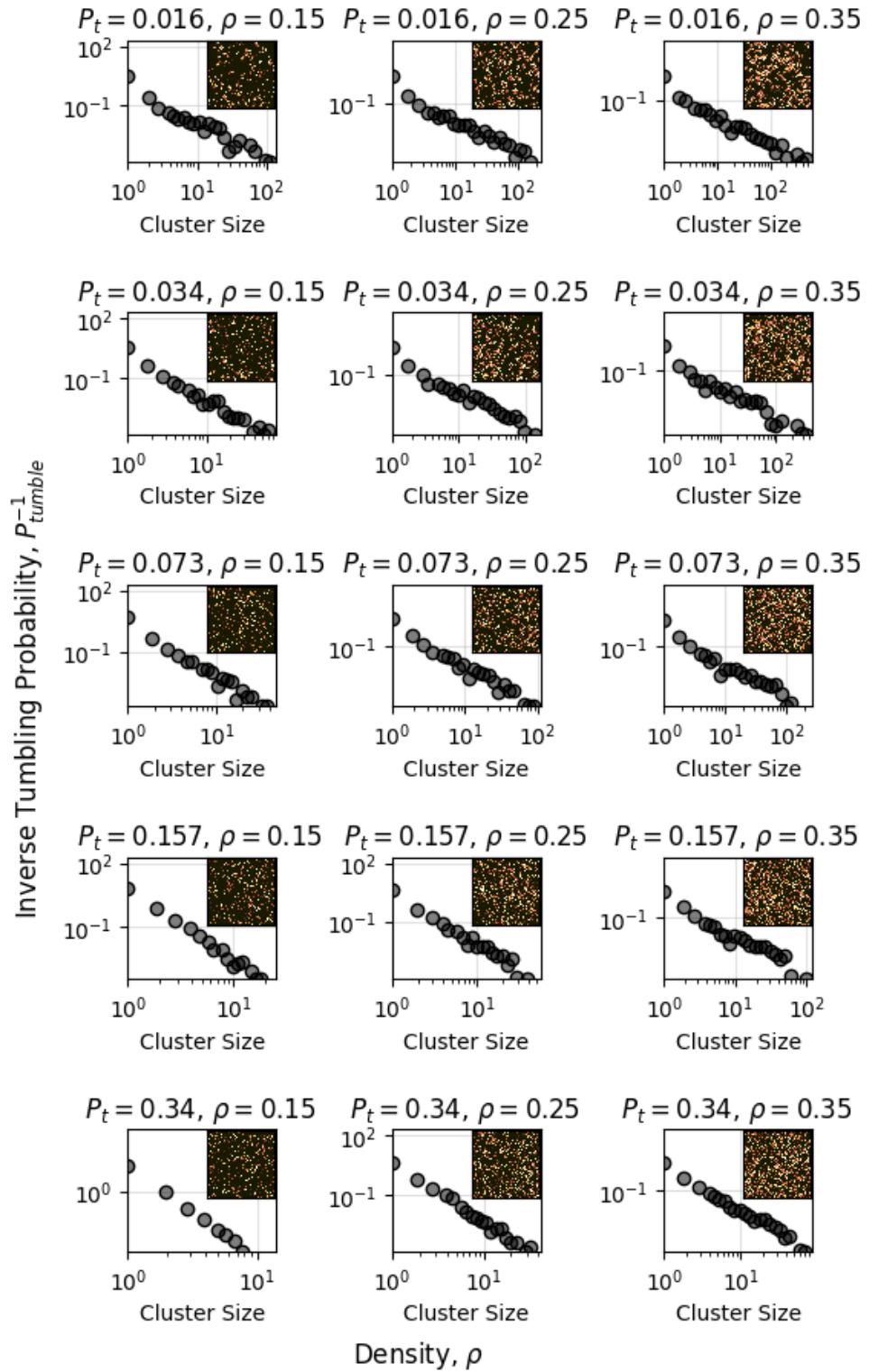


Figure 112.2: png

113 3. Cluster Number and Size Examination

113.1 Import Libraries

```
import warnings
warnings.filterwarnings("ignore")

import sys
sys.path.append('..../..')

import h5py
import glob
import re

from scipy import ndimage
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from cmcrameri import cm
import numpy as np
import seaborn as sns
import pandas as pd

from src.utils import get_ds_iters, get_cluster_labels
from src.plot_utils import get_plot_configs
```

113.2 Get Biggest Cluster + Cluster Count Details

```
densities = [0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5]
tumbles = np.logspace(-6, -1, 10, base=2)
num = []
```

```

big = []
ds = []
ts = []

for idx, d in enumerate(densities):
    for jdx, t in enumerate(tumbles):
        file = f"../data/no-rolling/dataset_tumble_{t:.3f}_density_{d}.h5"
        for idx2 in range(500,1000,1):
            ds.append(d)
            ts.append(t)
            labelled, nlabels = get_cluster_labels(file, idx2)
            lb = labelled.flatten()
            big.append(np.max(np.bincount(lb)[1:]))
            num.append(nlabels)

```

113.3 Create Dataframe Storage

```

df = pd.DataFrame()
df.insert(0, "alpha", ts,300)
df.insert(1, "numclus", num)
df.insert(2, "bigsize", big)
df.insert(0, "density", ds,300)

df.to_csv("cache/cluster_count_size.csv")

```

113.4 Plot Results

```

tumbles = np.logspace(-6, -1, 10, base=2)
df = pd.read_csv("cache/cluster_count_size.csv")
df["density"] = ["${:s}$".format(x) for x in df["density"]] #this is a rough fix for the sns hue names
sns.set_style("ticks")
sns.set_style({"xtick.direction": "in","ytick.direction": "in"})

fig, axes = plt.subplots(
    2,
    1,

```

```

    figsize=(10, 10),
    constrained_layout=True,
    dpi=600
)

ax1 = sns.lineplot(ax=axes[0], data=df, x="alpha", y="numclus", errorbar='sd', marker='o',
ax2 = sns.lineplot(ax=axes[1], data=df, x="alpha", y="bigsize", errorbar='sd', marker='o',

ax1.set_ylabel("Cluster count")
ax2.set_ylabel(r"Biggest cluster volume ( $a^2$ )")

for ax in axes:
    ax.set_xscale('log')
    ax.set_xlabel(r"Tumbling rate ( $P_t$ )")
    ax.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
    ax.set_xticks(np.round(tumbles,3)[::2])

ax2.set_yscale('log')
sns.move_legend(ax1, "upper left", title=r"Density ( $\rho$ )",ncols=2)
sns.move_legend(ax2, "upper right", title=r"Density ( $\rho$ )",ncols=3)
sns.despine()
plt.show()

fig.savefig("../plots/cluster_count_size.pdf")

```

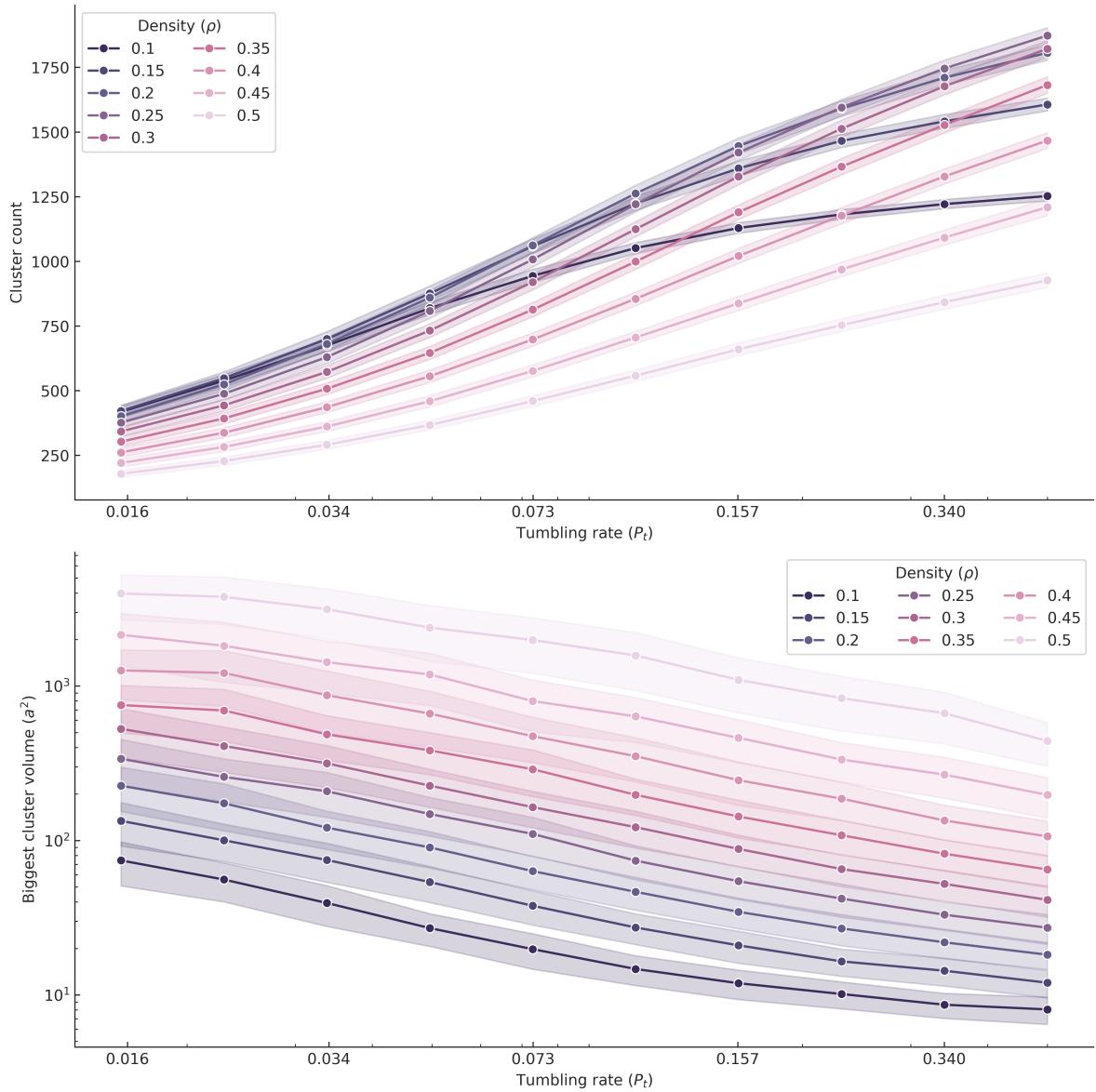


Figure 113.1: png

114 4. Percolation Analysis

114.0.1 Import Libraries

```
import matplotlib.pyplot as plt
import cmcrameri
import h5py
import numpy as np
from scipy import ndimage
import seaborn as sns
import pandas as pd
from cmcrameri import cm
import natsort

import sys
sys.path.append('..')
from src.utils import get_cluster_labels, get_ds_iters
from src.training_utils import extract_floats

np.set_printoptions(precision=5)
```

```
2024-04-11 13:23:37.221633: I tensorflow/core/util/port.cc:113] oneDNN custom operations are
2024-04-11 13:23:37.255613: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensorflow
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild Tensorflow
2024-04-11 13:23:37.701060: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Wa
```

114.0.2 Define Auxiliary Functions

```
def get_biggest_cluster_loc(file, idx):
    kernel = [[0, 1, 0], [1, 1, 1], [0, 1, 0]]
    labelled, _ = get_cluster_labels(file, idx)
    lb = labelled.flatten()
    cluster_sizes = np.bincount(lb)[1:]
    biggest_cluster_id = np.argmax(cluster_sizes) + 1
```

```

        return np.where(labelled == biggest_cluster_id)

def check_threshold(locs, threshold=20, dim=128):
    #print(np.ptp(locs[0]), np.ptp(locs[1]))
    if np.ptp(locs[0]) >= 128-threshold:
        return True
    if np.ptp(locs[1]) >= 128-threshold:
        return True
    return False

```

114.0.3 Establish Percolation Values

NOTE: this program is set to append, so it will pile on the same data if this notebook is reran.

```

tumbles = np.logspace(-6,-1,10,base=2)
densities = [0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5]
files = []

for tumble in tumbles:
    for density in densities:
        files.append(f"..../data/no-rolling/dataset_tumble_{tumble:.3f}_density_{density}.h5")

for ctr,file in enumerate(files):
    with h5py.File(file, "r") as fin:
        key_list = list(fin.keys())
        iter_n = get_ds_iters(key_list)
        percolating = 0
        total = 0
        for idx in range(50,1000): #amount of snapshots we have in the data sets
            locs = get_biggest_cluster_loc(file,idx)
            percolating += check_threshold(locs)
            total+=1
        ratio = float(percolating) / float(total)
        values = extract_floats(file)
        output_line = f"{values[0]} {values[1]} {ratio}\n"
        with open("cache/percolation.txt", "a") as cache:
            cache.write(output_line)
            print(output_line,end=' ')

```

```
0.016 0.45 0.8463157894736842
0.023 0.45 0.8010526315789473
0.034 0.45 0.651578947368421
0.050 0.45 0.5063157894736842
0.073 0.45 0.4231578947368421
0.107 0.45 0.4189473684210526
0.157 0.45 0.32105263157894737
0.231 0.45 0.3105263157894737
0.340 0.45 0.24210526315789474
0.500 0.45 0.16631578947368422
```

114.0.4 Read Percolations

```
with open("cache/percolation.txt", 'r') as cache:
    res = cache.readlines()
    perc = []
    for val in res:
        line_list = [float(i) for i in val.split(" ")]
        perc.append(line_list[2])
        if line_list[2] > 0.75: #frequency threshold for percolating condition is 75%
            print(f"tumble {line_list[0]}, dens {line_list[1]} undergoes percolation")
#perc = [float(val.strip('\n')) for val in res[2]]
perc = np.reshape(perc, (10,10))
```

```
tumble 0.016, dens 0.45 undergoes percolation
tumble 0.016, dens 0.5 undergoes percolation
tumble 0.023, dens 0.45 undergoes percolation
tumble 0.023, dens 0.5 undergoes percolation
tumble 0.034, dens 0.5 undergoes percolation
tumble 0.05, dens 0.5 undergoes percolation
tumble 0.073, dens 0.5 undergoes percolation
```

114.0.5 Plot Results

```
tumbles = np.logspace(-6,-1,10, base=2)
density = [0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5]

cmap = plt.get_cmap('cmc.acton')

xlabels = np.concatenate(([0,0],tumbles))
ylabels = np.concatenate(([0,0],density))

fig, ax = plt.subplots(1,1, figsize=(5,5), constrained_layout=True)
m = ax.imshow(perc, cmap=cmap.reversed())
cbar = plt.colorbar(ax=ax, mappable=m, location='top', aspect=20, fraction=0.046, pad=0.05)
ax.set_yticklabels(np.round(xlabels,3)[::2])
ax.set_xticklabels(np.round(ylabels,3)[::2])

ax.plot([7.5,7.5,8.5,8.5,9.5],[-0.5,1.5,1.5,4.5,4.5], c='white', linestyle='--')

cbar.ax.set_xlabel('Percolation Frequency')

ax.set_ylabel(r"Tumbling rate ($P_t$)")
ax.set_xlabel(r"Density ($\rho$)")

ax.text(s="Percolating", x=8.75, y=3, c='w', rotation=90, fontsize=16)

/tmp/ipykernel_9498/2130497943.py:12: UserWarning: set_ticklabels() should only be used with
    ax.set_yticklabels(np.round(xlabels,3)[::2])
/tmp/ipykernel_9498/2130497943.py:13: UserWarning: set_ticklabels() should only be used with
    ax.set_xticklabels(np.round(ylabels,3)[::2])

Text(8.75, 3, 'Percolating')
```

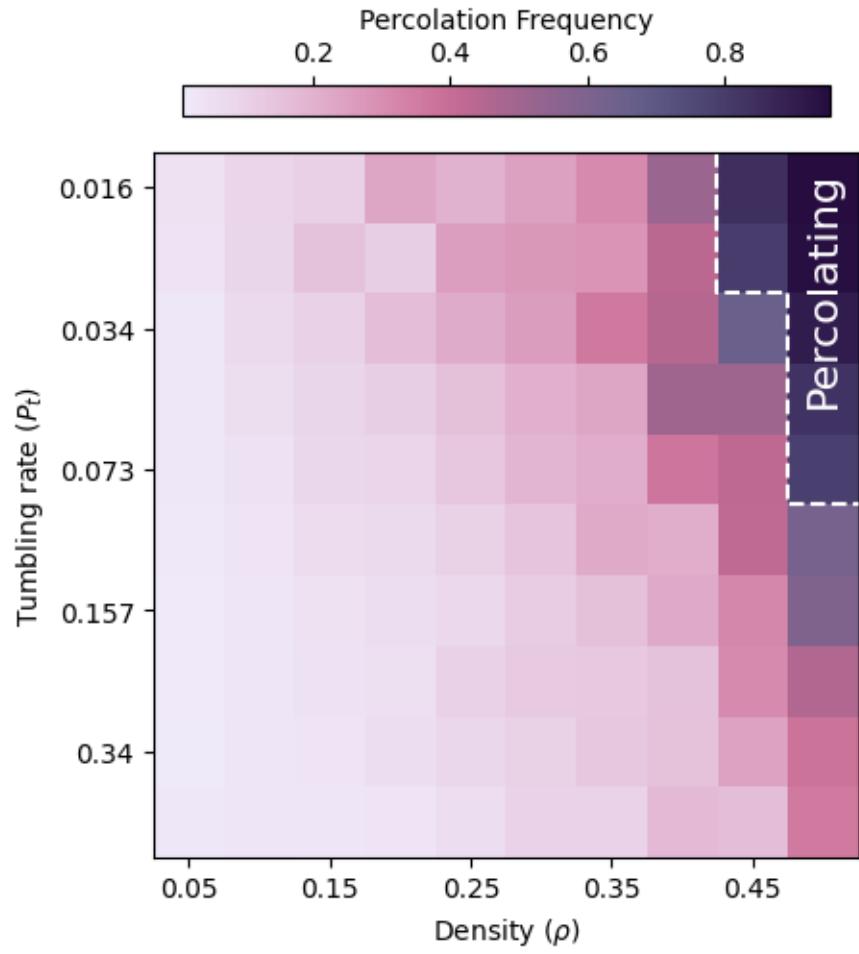


Figure 114.1: png

115 5. Cluster Orientation Analysis

115.1 Import Data

```
import matplotlib.pyplot as plt
import cmcrameri
import h5py
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import numpy as np
from scipy import ndimage
import seaborn as sns
import pandas as pd
from cmcrameri import cm

import sys
sys.path.append('..')

from src.utils import get_cluster_labels, get_ds_iters
from src.waffle_plot import waffle_plot

#from src.plot_utils import get_plot_configs

#plot_configs = get_plot_configs()
#plt.rcParams.update(plot_configs)
```

115.2 Define Functions

```
def plot_labelled_cluster(axis, file, sshot_idx):
    cmap_label = plt.get_cmap(name="gnuplot")
    #cmap_label = 'cmc.acton'
    labelled, _ = get_cluster_labels(file, sshot_idx)
    axis.matshow(labelled, cmap=cmap_label)
    return axis
```

```

def get_biggest_cluster(img):
    kernel = [[0, 1, 0], [1, 1, 1], [0, 1, 0]]
    labelled, _ = ndimage.label(img, structure=kernel)
    lb = labelled.flatten()
    cluster_sizes = np.bincount(lb)[1:]
    biggest_cluster_id = np.argmax(cluster_sizes)
    loc = ndimage.find_objects(labelled)[biggest_cluster_id]
    labelled_crop = labelled[loc]
    img_crop = img[loc]
    labelled_crop[labelled_crop != biggest_cluster_id+1] = 0
    labelled_crop[labelled_crop == biggest_cluster_id+1] = 1
    img_crop *= labelled_crop
    return img_crop, ndimage.center_of_mass(labelled_crop)

def get_edges(img, axis):
    img_threshold = np.zeros_like(img)
    img_threshold[img > 0] = 1
    edges = ndimage.sobel(img_threshold, axis=axis)
    #edges[edges > -2] = 0
    #edges[edges != 0] = 1
    edges *= img
    return edges

def map_ori(ori):
    ori_mapped = np.zeros_like(ori, dtype=np.float_)
    ori_mapped[ori == 1] = np.pi
    ori_mapped[ori == 2] = np.pi/2
    ori_mapped[ori == 3] = 0
    ori_mapped[ori == 4] = -np.pi/2
    return ori_mapped

def map_ori_human(ori):
    ori_mapped = np.zeros_like(ori, dtype=np.dtype('U100'))
    ori_mapped[ori == 3] = "Down"
    ori_mapped[ori == 2] = "Right"
    ori_mapped[ori == 1] = "Up"
    ori_mapped[ori == 4] = "Left"
    return ori_mapped

```

```

# 0: y, 1: x
def get_ori_and_loc(edges, com):
    positions = edges.nonzero()
    edges_ori = map_ori(edges[positions[0], positions[1]])
    edges_loc = np.arctan2((com[0]-positions[0]), (positions[1]-com[1]))

    # CHECK CODE
    # colors = ListedColormap(["k", "r", "yellow", "g", "b"])
    # edges[edges == 0] = -4
    # edges[positions[0], positions[1]] = edges_loc
    plt.matshow(edges)
    plt.colorbar()
    plt.axvline(com[1], c='w')
    plt.axhline(com[0], c='w')

    return edges_ori, edges_loc

```

115.3 Plot Cluster Map

```

import matplotlib as mpl
mpl.rcParams.update(mpl.rcParamsDefault)
sshot_idx = -1
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3), constrained_layout=True)
ax1 = plot_labelled_cluster(ax1, "../data/no-rolling/dataset_tumble_0.016_density_0.4.h5",
    ax1.text(
        y=-0.1,
        x=1,
        transform=ax1.transAxes,
        ha="right",
        s=r"\alpha = 0.016",
    )
ax2 = plot_labelled_cluster(ax2, "../data/no-rolling/dataset_tumble_0.340_density_0.4.h5",
    ax2.text(
        y=-0.1,
        x=1,
        transform=ax2.transAxes,
        ha="right",
        s=r"\alpha = 0.340",
    )

```

```

plt.show()
fig.savefig("../plots/cluster_orientation_analysis/cluster_map.svg")

```

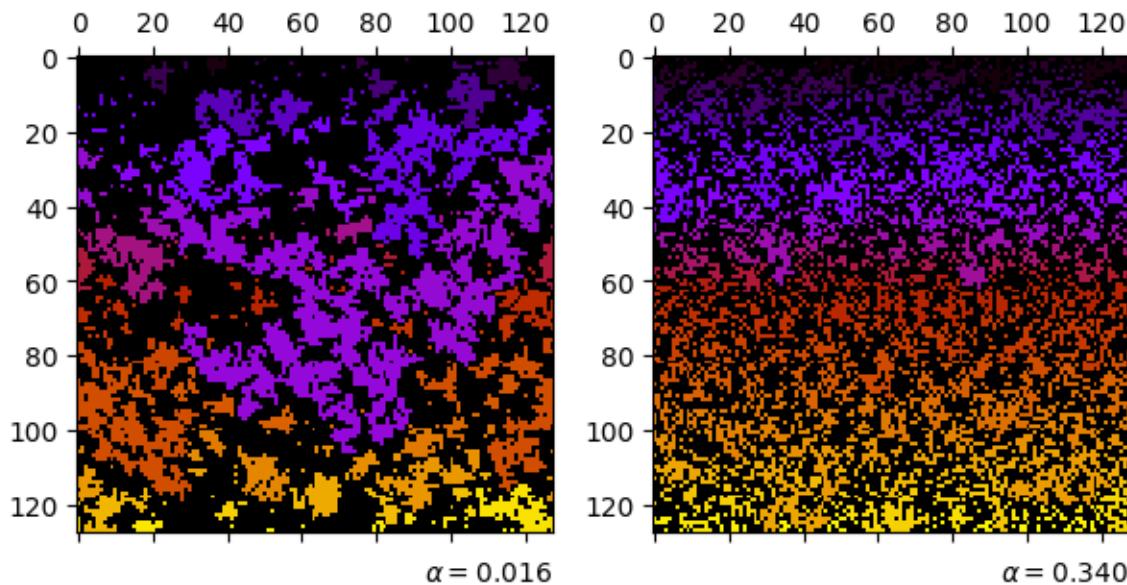


Figure 115.1: png

115.4 Get Frequency of Up-Down Edge Orientations

```

hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.016_density_0.4.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)

img2 = get_edges(img, axis=0)
img2[img2<0] = -1
img2[img2>0] = 1

locs_neg = np.where(img2 == -1)
locs_mid = np.where(img2 == 0)
locs_pos = np.where(img2 == 1)

neg = img[locs_neg[0][:], locs_neg[1][:]]

```

```

mid = img[locs_mid[0][:],locs_mid[1][:]]
pos = img[locs_pos[0][:],locs_pos[1][:]]

counts = np.column_stack((np.bincount(neg),np.bincount(pos)))[1:]

df = pd.DataFrame({
    'Grad': ([["Negative"]+["Positive"]]*4,
    'Orientation': pd.Categorical(
        ["Up", "Up", "Right", "Right", "Down", "Down", "Left", "Left"],
        categories=["Left", "Up", "Right", "Down"]
    ),
    'Frequency': counts.flatten(),
})
df.to_csv("cache/updown_ori_freq.csv")

```

115.5 Get Frequency of Left-Right Edge Orientations

```

hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.016_density_0.4.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)

img2 = get_edges(img, axis=1)
img2[img2<0] = -1
img2[img2>0] = 1

locs_neg = np.where(img2 == -1)
locs_mid = np.where(img2 == 0)
locs_pos = np.where(img2 == 1)

neg = img[locs_neg[0][:],locs_neg[1][:]]
mid = img[locs_mid[0][:],locs_mid[1][:]]
pos = img[locs_pos[0][:],locs_pos[1][:]]

counts = np.column_stack((np.bincount(neg),np.bincount(pos)))[1:]

df = pd.DataFrame({
    'Grad': ([["Negative"]+["Positive"]]*4,
    'Orientation': pd.Categorical(

```

```

        ["Up", "Up", "Right", "Right", "Down", "Down", "Left", "Left"],
        categories=["Left", "Up", "Right", "Down"]
    ),
    'Frequency': counts.flatten(),
)
df.to_csv("cache/leftright_ori_freq.csv")

```

115.6 Get Frequency of Up-Down Scrambled Edge Orientations

```

hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.016_density_0.4.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)
img[img > 0] = 1
img = img * np.random.randint(1, 5, size=(128, 128))

img2 = get_edges(img, axis=0)
img2[img2<0] = -1
img2[img2>0] = 1

locs_neg = np.where(img2 == -1)
locs_mid = np.where(img2 == 0)
locs_pos = np.where(img2 == 1)

neg = img[locs_neg[0][:], locs_neg[1][:]]
mid = img[locs_mid[0][:], locs_mid[1][:]]
pos = img[locs_pos[0][:], locs_pos[1][:]]

counts = np.column_stack((np.bincount(neg), np.bincount(pos)))[1:]

df = pd.DataFrame({
    'Grad': (["Negative"]+["Positive"])*4,
    'Orientation': pd.Categorical(
        ["Up", "Up", "Right", "Right", "Down", "Down", "Left", "Left"],
        categories=["Left", "Up", "Right", "Down"]
    ),
    'Frequency': counts.flatten(),
})
df.to_csv("cache/updown_ori_freq_scrambled.csv")

```

115.7 Get Frequency of Left-Right Scrambled Edge Orientations

```
hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.016_density_0.4.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)
img[img > 0] = 1
img = img * np.random.randint(1, 5, size=(128, 128))

img2 = get_edges(img, axis=1)
img2[img2<0] = -1
img2[img2>0] = 1

locs_neg = np.where(img2 == -1)
locs_mid = np.where(img2 == 0)
locs_pos = np.where(img2 == 1)

neg = img[locs_neg[0][:],locs_neg[1][:]]
mid = img[locs_mid[0][:],locs_mid[1][:]]
pos = img[locs_pos[0][:],locs_pos[1][:]]

counts = np.column_stack((np.bincount(neg),np.bincount(pos)))[1:]

df = pd.DataFrame({
    'Grad': (["Negative"]+["Positive"])*4,
    'Orientation': pd.Categorical(
        ["Up", "Up", "Right", "Right", "Down", "Down", "Left", "Left"],
        categories=["Left", "Up", "Right", "Down"]),
    ,
    'Frequency': counts.flatten(),
})
df.to_csv("cache/leftright_ori_freq_scrambled.csv")
```

115.8 Plot Cluster Edge Orientations

```
#mpl.rcParams.update(mpl.rcParamsDefault)

df = pd.read_csv("cache/leftright_ori_freq_scrambled.csv")
```

```

#plot_configs = get_plot_configs()
#sns.set(rc=plot_configs)
sns.set_style("ticks")
sns.set_style({"xtick.direction": "in","ytick.direction": "in"})
#plt.rcParams.update(plot_configs)
fig = plt.figure(figsize=(5,5), constrained_layout=True,dpi=600)
ax0 = fig.add_subplot(
    1, 1, 1
)
ax4 = fig.add_subplot(
    2, 2, 1
)
ax2 = fig.add_subplot(
    2, 2, 2
)
ax3 = fig.add_subplot(
    2, 2, 3
)
ax1 = fig.add_subplot(
    2, 2, 4
)

hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.016_density_0.4.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)
img_thres = img
img_thres[img_thres>0]=1
#axins = inset_axes(ax4, width="100%", height="100%", borderpad=1)
#axins.set_axes_locator(InsetPosition(ax4, [.7, 1.3, 1, 1]))
#axins.matshow(img_thres, cmap='cmc.oslo')
#axins.tick_params(
#    axis = "both",
#    which = "both",
#    length = 0,
#    labelleft = False,
#    labeltop = False,
#)
img2 = get_edges(img, axis=0)
img2[img2<0] = -1
img2[img2>0] = 1

```

```

cbar = ax2.matshow(img2, cmap=plt.get_cmap(cm.bam, lut=3))
cb = plt.colorbar(cbar, ax=ax2, ticks=np.arange(-1, 1 + 1), values=np.arange(-1, 1 + 1), 1
cb.set_ticklabels(["Bottom\nedges", "BG.", "Top\nedges"])
ax2.tick_params(
    axis = "both",
    which = "both",
    length = 0,
    labelleft = False,
    labeltop = False,
)
img2 = get_edges(img, axis=1)
img2[img2<0] = -1
img2[img2>0] = 1
cbar = ax4.matshow(img2, cmap=plt.get_cmap(cm.bam, lut=3))
cb = plt.colorbar(cbar, ax=ax4, ticks=np.arange(-1, 1 + 1), values=np.arange(-1, 1 + 1), 1
cb.set_ticklabels(["Right\nedges", "BG.", "Left\nedges"])
ax4.tick_params(
    axis = "both",
    which = "both",
    length = 0,
    labelleft = False,
    labeltop = False,
)
palette = {'Negative': '#0C4B00', 'Positive': '#65024B' }
df = pd.read_csv("cache/updown_ori_freq.csv")
sns.barplot(ax=ax1, data=df, x='Orientation',y='Frequency', hue='Grad', palette=palette, 1
ax1.set(xticklabels=[])

#df = pd.read_csv("cache/updown_ori_freq_scrambled.csv")
#sns.barplot(ax=ax2, data=df, x='Orientation',y='Frequency', hue='Grad', palette=palette)
#ax2.set(xlabel=None, xticklabels=[], ylabel=None, yticklabels[])
#sns.move_legend(ax2, "upper right", bbox_to_anchor=(1, 1.1), frameon=False, title=r"Grad

df = pd.read_csv("cache/leftright_ori_freq.csv")
sns.barplot(ax=ax3, data=df, x='Orientation',y='Frequency', hue='Grad', palette=palette, 1
#df = pd.read_csv("cache/leftright_ori_freq_scrambled.csv")
#sns.barplot(ax=ax5, data=df, x='Orientation',y='Frequency', hue='Grad', palette=palette,
#ax5.set(yticklabels[])

```

```

for idx, ax in enumerate((ax1,ax3)):
    sns.despine(ax=ax)
    ax.set_ylim(0,1100)
    ax.set(xlabel=None, ylabel=None)
    ax.set_xticklabels(["↑","↓","→","←"])
    #ax.set_ylabel('Particle count', fontsize=14)
    ax.set_xlabel("Particle orientation")

ax4.set_title("Vertical Filter")
ax2.set_title("Horizontal Filter")

#fig.text(s="(A)",x=0.17, y=0.95)
#fig.text(s="(B)",x=0.52, y=0.95, backgroundcolor=(1,1,1,0.95))
#fig.text(s="(C)",x=0.17, y=0.48)
#fig.text(s="(D)",x=0.52, y=0.48, backgroundcolor=(1,1,1,0.95))
fig.savefig("../plots/cluster_orientation_analysis/cluster_edge_orientations.png", bbox_in

```

```

/tmp/ipykernel_13085/2311038860.py:93: UserWarning: set_ticklabels() should only be used with
  ax.set_xticklabels(["↑","↓","→","←"])
/tmp/ipykernel_13085/2311038860.py:93: UserWarning: set_ticklabels() should only be used with
  ax.set_xticklabels(["↑","↓","→","←"])

```

115.9 Plot Waffle Plot Distribution For Horizontal Filter

```

df = pd.read_csv("cache/updown_ori_freq.csv")
sns.barplot(ax=ax1, data=df, x='Orientation',y='Frequency', hue='Grad', palette=palette, l
ax1.set(xticklabels=[])

waffle_plot(df[df['Grad'] == "Positive"]['Orientation'], df[df['Grad'] == "Positive"]['Fre
waffle_plot(df[df['Grad'] == "Negative"]['Orientation'], df[df['Grad'] == "Negative"]['Fre

```

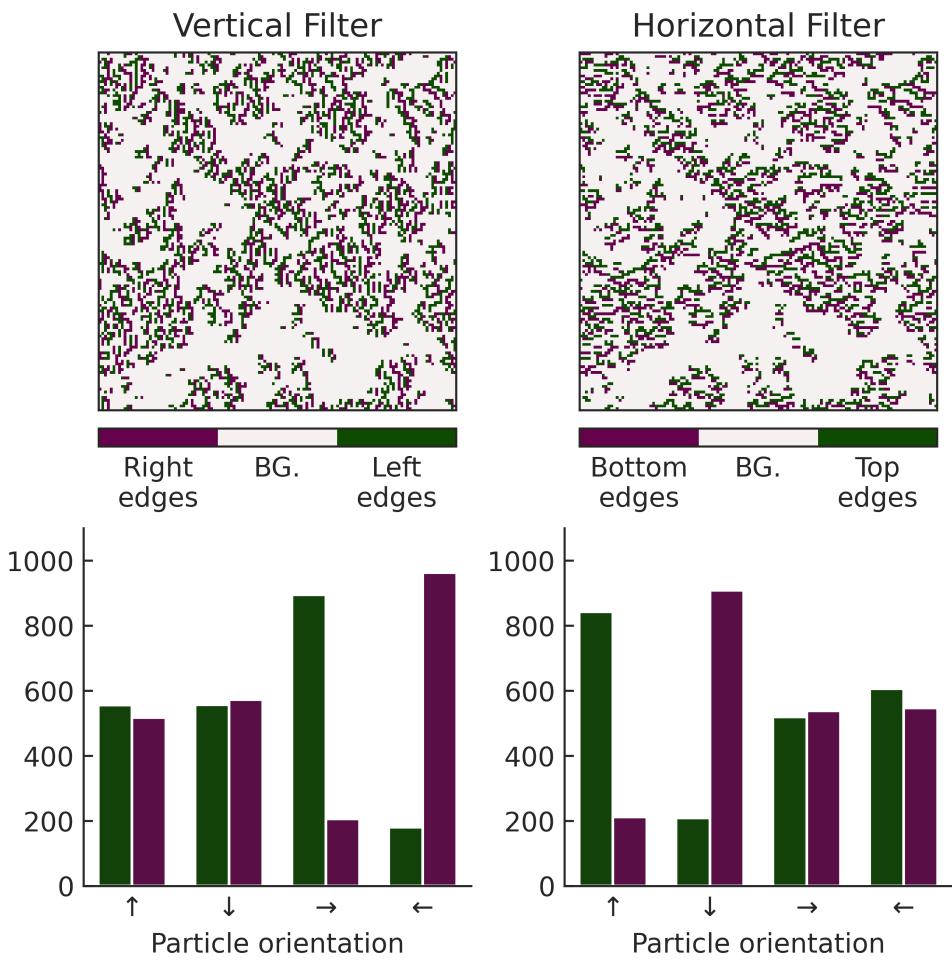


Figure 115.2: png

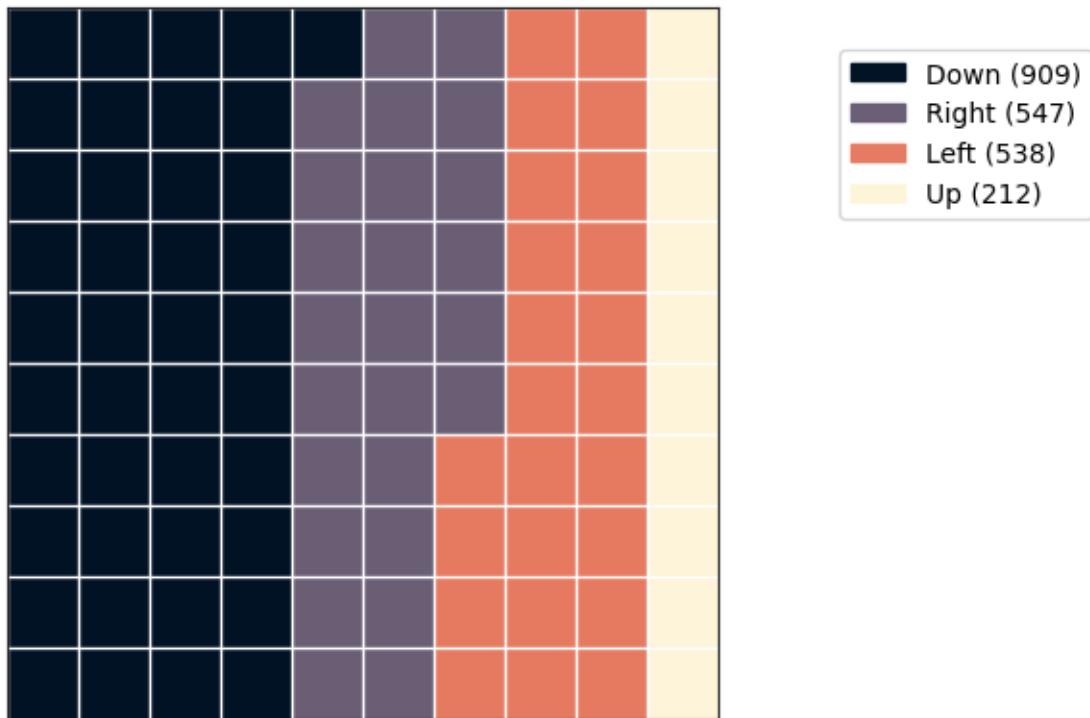


Figure 115.3: png

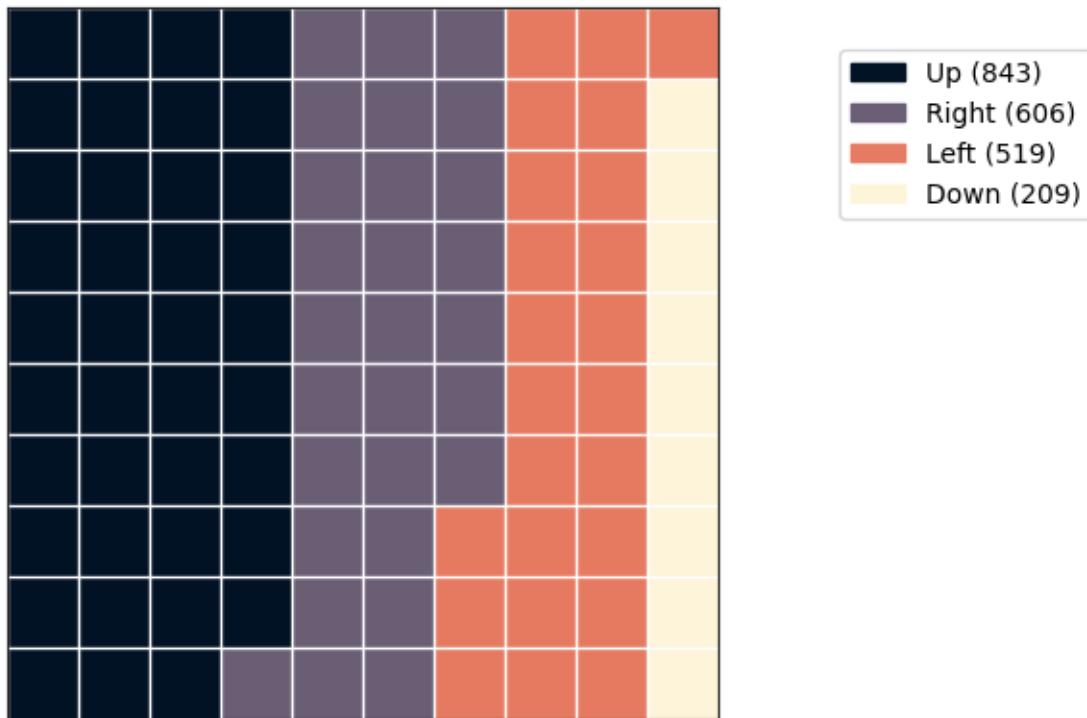


Figure 115.4: png

115.10 Frequency Comparison

```
df = pd.read_csv("cache/updown_ori_freq.csv")
df2 = pd.read_csv("cache/updown_ori_freq_scrambled.csv")

df3 = pd.read_csv("cache/leftright_ori_freq.csv")
df4 = pd.read_csv("cache/leftright_ori_freq_scrambled.csv")

diff1 = (df[df['Grad'] == "Positive"]['Frequency'].values - df[df['Grad'] == "Negative"][])
diff1[[1,2]] = diff1[[2,1]]

diff2 = (df3[df3['Grad'] == "Positive"]['Frequency'].values - df3[df3['Grad'] == "Negative"])
diff2[[1,2]] = diff2[[2,1]]

diff3 = (df2[df2['Grad'] == "Positive"]['Frequency'].values - df2[df2['Grad'] == "Negative"])
diff3[[1,2]] = diff3[[2,1]]

diff4 = (df4[df4['Grad'] == "Positive"]['Frequency'].values - df4[df4['Grad'] == "Negative"])
diff4[[1,2]] = diff4[[2,1]]

ori = np.array(["Up", "Left", "Down", "Right"])
ori[[1,2]] = ori[[2,1]]

len(np.tile(ori,2))

diff_v = pd.DataFrame({
    "Difference": np.concatenate((
        diff1,
        diff2
    )),
    "Orientation": np.tile(ori,2),
    'Context': ([["Vertical filter"]]*4+["Horizontal filter"]*4)*1,
})
}

diff_v2 = pd.DataFrame({
    "Difference": np.concatenate((
        diff3,
        diff4,
    )),
    "Orientation": np.tile(ori,2),
```

```
'Context': (["Vertical filter"]*4+["Horizontal filter"]*4)*1,  
})  
  
fig = plt.figure(figsize=(4,5), constrained_layout=True,dpi=600)  
  
ax1 = sns.barplot(data=diff_v, x='Orientation',y='Difference', palette='cmc.tokyo', hue='O  
sns.move_legend(ax1, "lower center", bbox_to_anchor=(0.5, 0.1), frameon=False, title=None)  
  
ax1.set(ylabel="Difference between opposing edges")  
  
[Text(0, 0.5, 'Difference between opposing edges')]
```

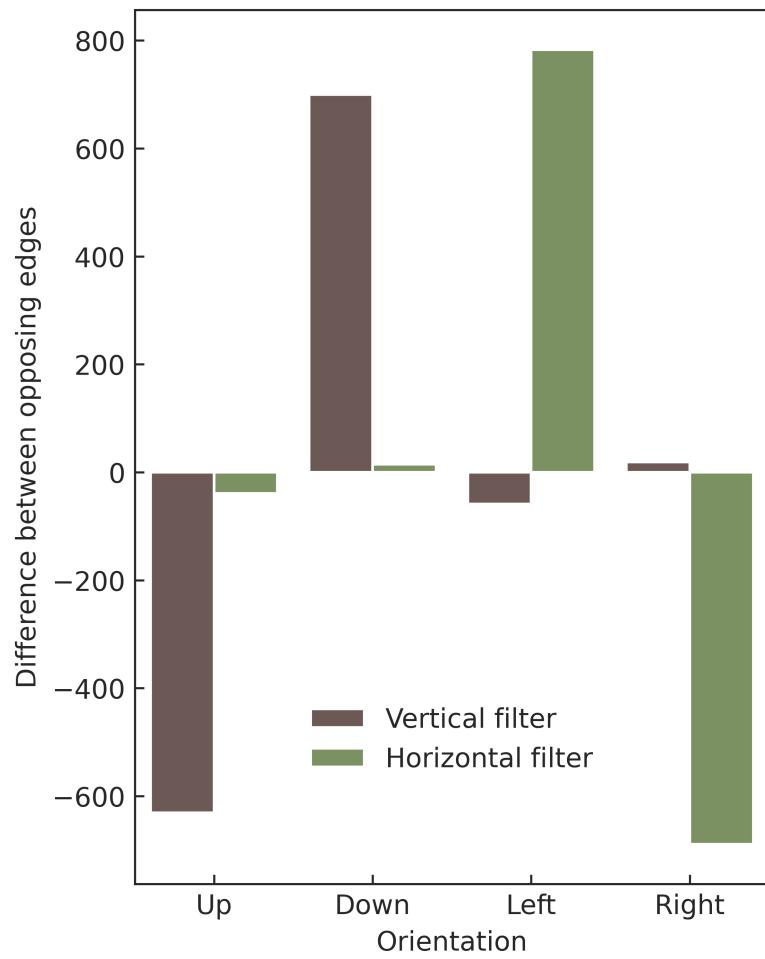


Figure 115.5: png

116 6. Feature Map Analysis (Orientation Data Type)

116.1 Import Libraries

```
import warnings
warnings.filterwarnings("ignore")

import h5py
import numpy as np

import tensorflow as tf
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import img_to_array
from keras.models import Model
import matplotlib.pyplot as plt
from cmcrameri import cm
from numpy import expand_dims

import sys
sys.path.append('..')

from src.utils import get_cluster_labels, get_ds_iters
from src.training_utils import (
    data_load,
    split_dataset,
)
from src.plot_utils import get_plot_configs
```

2024-04-08 22:29:35.720088: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary was not compiled with CPU FMA support but currently has active CPU FMA operations. To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with

2024-04-08 22:29:36.831818: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W

116.2 Load Datasets

```
hf = h5py.File(f'../data/no-rolling/dataset_tumble_0.050_density_0.25.h5', "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)
img = img.reshape((img.shape[0], img.shape[1], 1))

hf = h5py.File(f'../data/no-rolling/dataset_tumble_0.157_density_0.25.h5', "r")
iters = get_ds_iters(hf.keys())
img2 = hf[f"conf_{iters[300]}"]
img2 = np.array(img2)
img2 = img2.reshape((img2.shape[0], img2.shape[1], 1))
```

116.3 Example Datamaps ($P_t \in \{0.050, 0.157\}$, $\rho = 0.25$)

```
plt.matshow(img, cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
plt.savefig('../plots/fmaps/input_0.050.svg', bbox_inches='tight', pad_inches=-0.1)
plt.matshow(img2, cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
plt.savefig('../plots/fmaps/input_0.157.svg', bbox_inches='tight', pad_inches=-0.1)
```

116.4 Set Up GPU and Load Model

Reminder: the following commands need to be ran in console in order to employ GPU. This is not strictly necessary here, since running on the CPU only affects performance (and we are computing very little data here). This is nonetheless a good habit.

```
export CUDNN_PATH=$(dirname $(python -c "import nvidia.cudnn;print(nvidia.cudnn.__file__)")
export LD_LIBRARY_PATH=${CUDNN_PATH}/lib
export PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:$PATH

model = tf.keras.models.load_model('../models/orientation0216.keras')
```

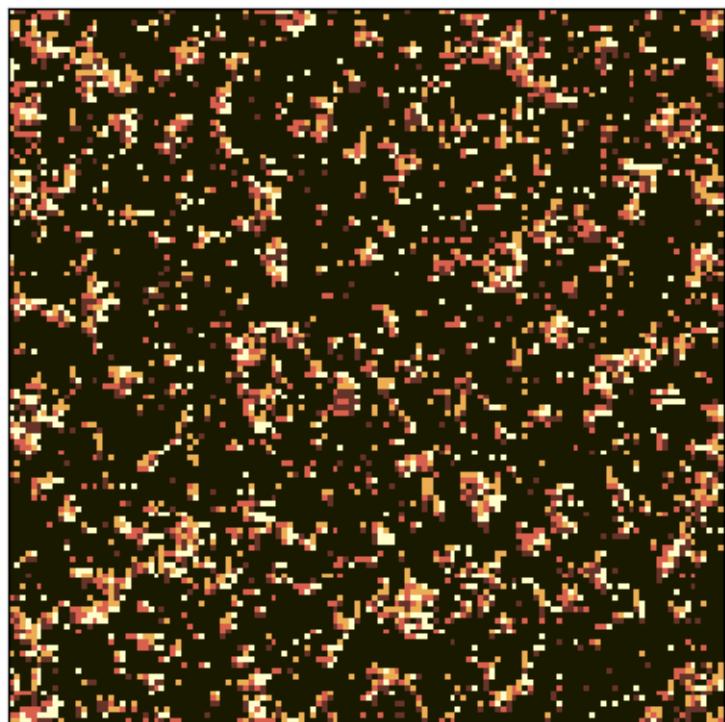


Figure 116.1: png

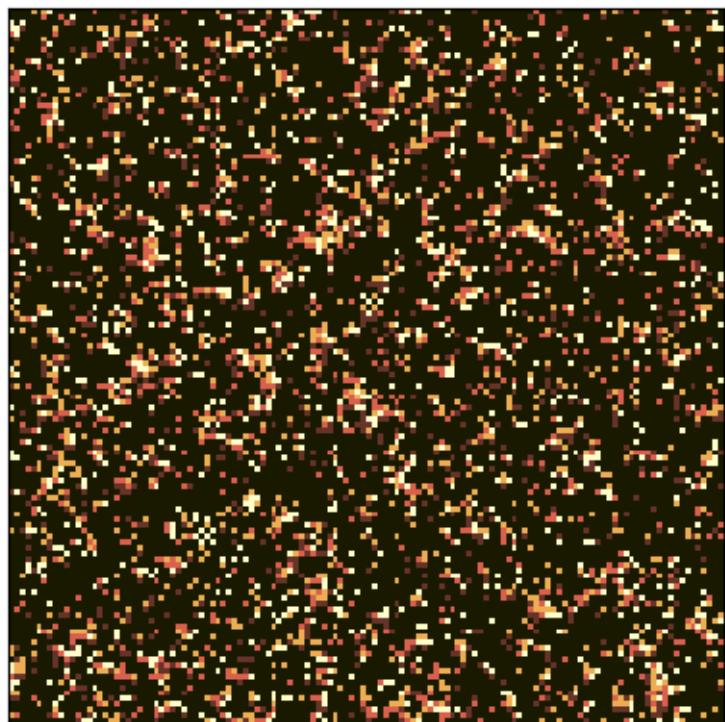


Figure 116.2: png

```

2024-04-08 22:29:40.971354: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.033494: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.034172: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.036429: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.036836: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.037016: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.144116: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.144449: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.144621: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-08 22:29:41.144752: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1928] Created

```

116.5 Plot Feature Maps

The following shows output feature maps from each architecture layer (indicated in titles). Row 0 is low tumbling rate ($P_t = 0.050$), row 1 is high tumbling rate ($P_t = 0.157$). Multiple columns indicates multiple emergent feature maps (decided by number of filters) from however many input feature maps the layer takes (dictated by previous layer output maps).

```

def model_mapper (img,img2,model,layer_number,shift=None,ncols=3,post_avgpool=False,flattened=False):
    model_mini = Model(inputs=model.inputs, outputs=model.layers[layer_number].output)

    feature_maps1 = model_mini.predict(img, verbose=0)
    feature_maps2 = model_mini.predict(img2, verbose=0)

    if post_avgpool == False and flattened == False:
        i = 2
        j = ncols
        for idx in range(i):
            for jdx in range(j):
                ax = plt.subplot(i, j, idx*j+jdx+1)
                ax.set_xticks([])
                ax.set_yticks([])
                if idx == 0:
                    plt.imshow(feature_maps1[:shift, :, 0, jdx], cmap='cmc.lajolla')
                else:
                    plt.imshow(feature_maps2[:shift, :, 0, jdx], cmap='cmc.lajolla')
        if output == True:
            plt.savefig(path+f"layer_{layer_number}.svg",bbox_inches='tight')

    if post_avgpool == True and flattened == False: #plots images which do not output mult

```

```

plt.matshow(feature_maps1[:, :], cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
if output == True:
    plt.savefig(path+f"layer_{layer_number}_im1.svg", bbox_inches='tight')
plt.matshow(feature_maps2[:, :], cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
if output == True:
    plt.savefig(path+f"layer_{layer_number}_im2.svg", bbox_inches='tight')

if post_avgpool == True and flattened == True: #rotates images which do not output mul
    plt.matshow(np.rot90(feature_maps1[:, :]), cmap='cmc.lajolla')
    plt.xticks([])
    plt.yticks([])
    if output == True:
        plt.savefig(path+f"layer_{layer_number}_im1.svg", bbox_inches='tight')
    plt.matshow(np.rot90(feature_maps2[:, :]), cmap='cmc.lajolla')
    plt.xticks([])
    plt.yticks([])
    if output == True:
        plt.savefig(path+f"layer_{layer_number}_im2.svg", bbox_inches='tight')

def kernel_printer(model, layer_number=0, path=f"../plots/fmaps/"):
    filters, biases = model.layers[layer_number].get_weights()
    print (filters.shape[-1])
    for k in range(filters.shape[-1]):
        f = filters[:, :, :, k]
        plt.matshow(f[:, :, 0], cmap="cmc.lajolla")
        plt.xticks([])
        plt.yticks([])
        plt.savefig(path+f"kernel_{k}_layer_{layer_number}.svg", bbox_inches='tight')

```

116.5.1 1. CONV

(filters=3,kernel_size=(3,3),padding='same',input_shape=shape)

```
model_mapper (img,img2,model,shift=None,layer_number=0,output=True)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1712604581.825443    9317 service.cc:145] XLA service 0x78b0500035e0 initialized
I0000 00:00:1712604581.825508    9317 service.cc:153] StreamExecutor device (0): NVIDIA Gel
2024-04-08 22:29:41.880682: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:465] L
I0000 00:00:1712604582.252356    9317 device_compiler.h:188] Compiled cluster using XLA! Th
```

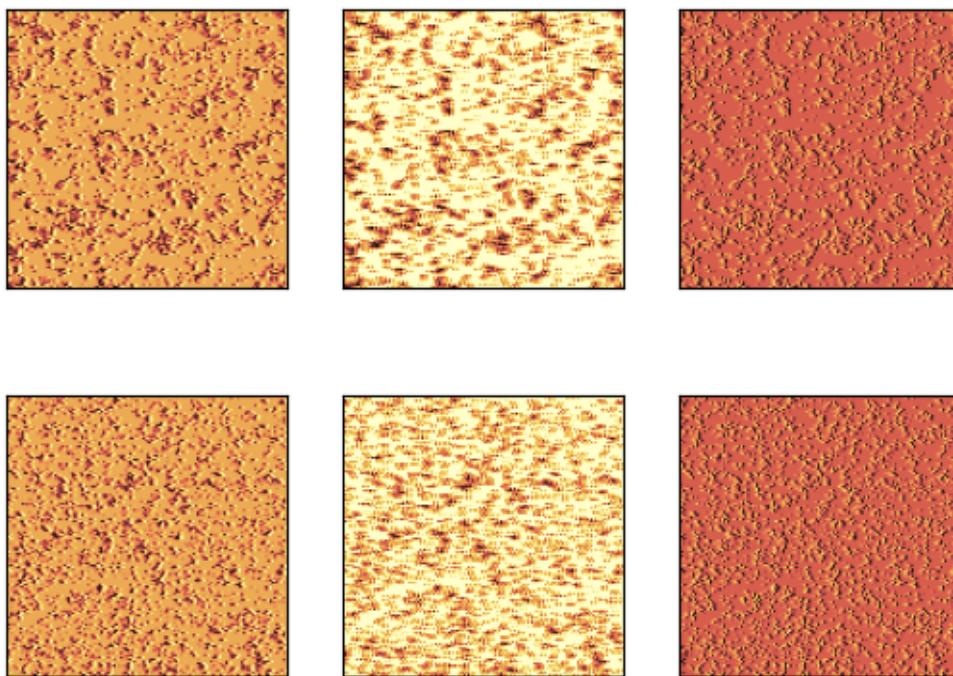


Figure 116.3: png

116.5.1.1 Computed with the Following Kernels

```
kernel_printer(model,layer_number=0,path=f"../plots/fmaps/")
```

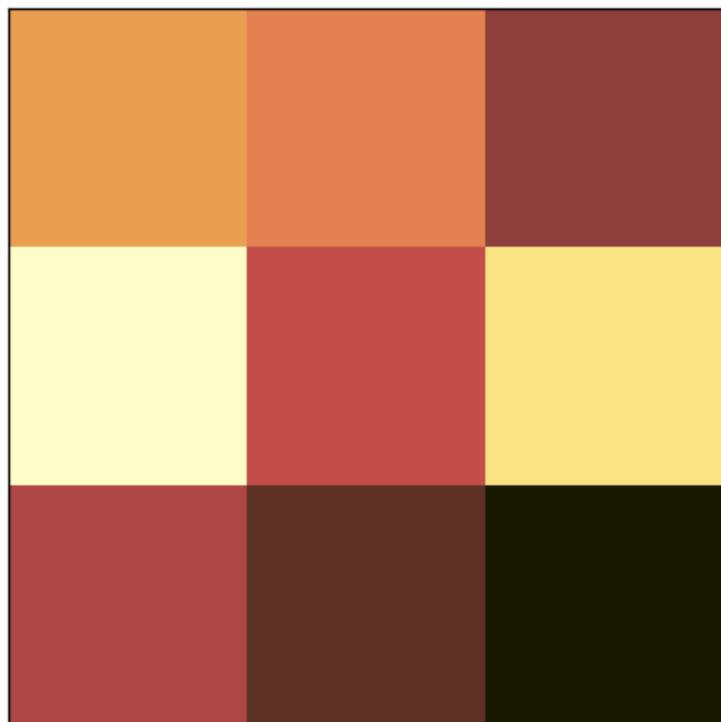


Figure 116.4: png

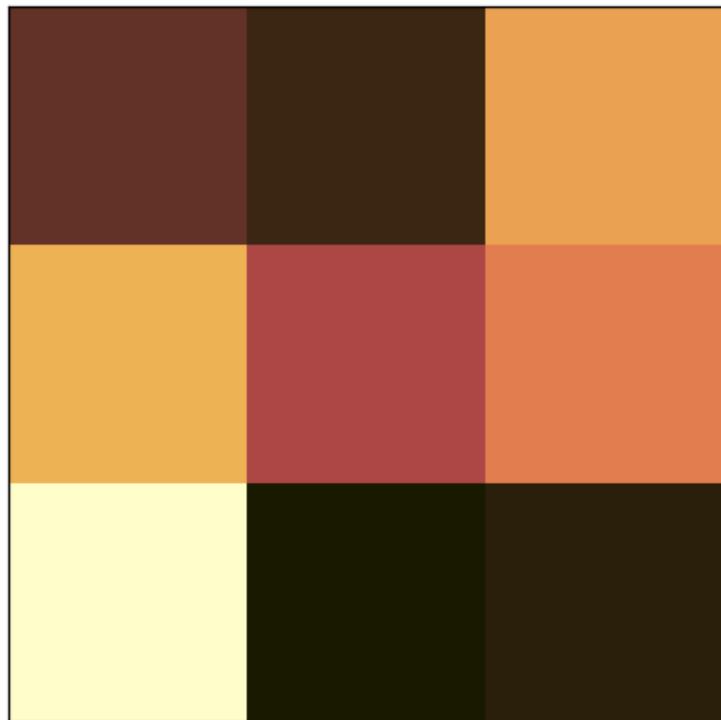


Figure 116.5: png

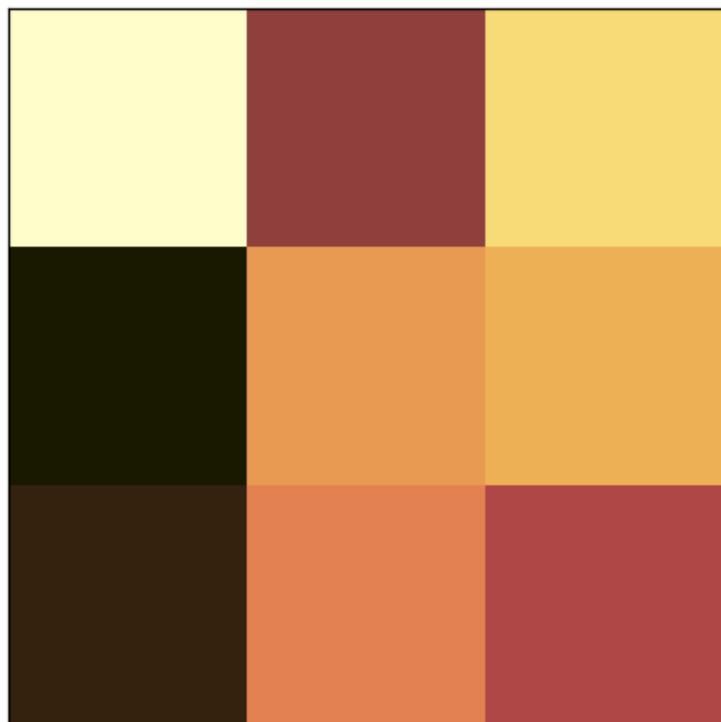


Figure 116.6: png

116.5.2 2. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=64,layer_number=1,output=True)
```

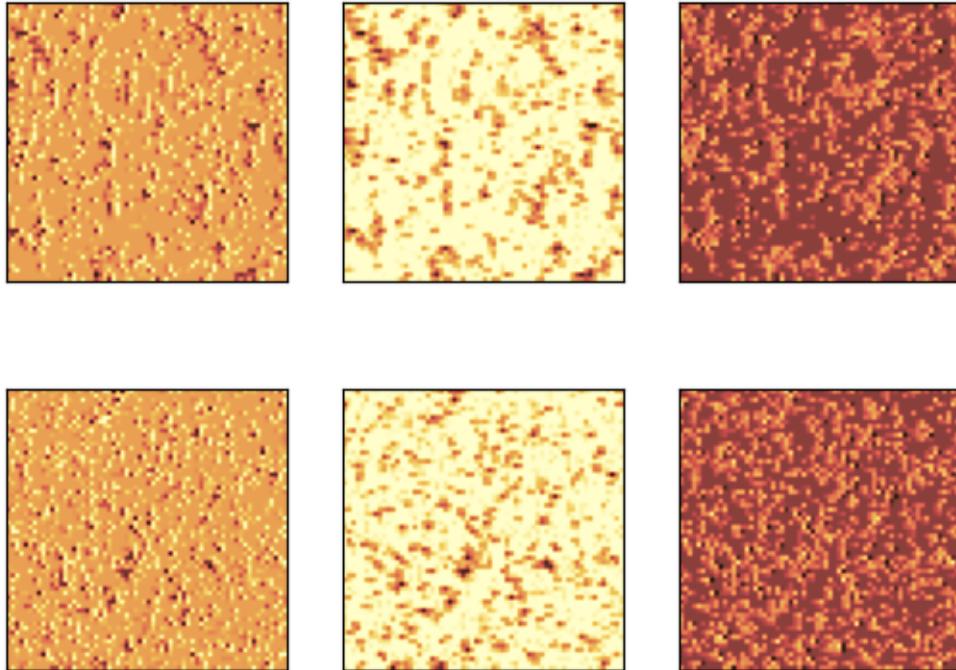


Figure 116.7: png

116.5.3 3. ReLU

```
model_mapper (img,img2,model,shift=64,layer_number=2,output=True)
```

2024-04-08 22:29:44.564777: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268

116.5.4 4. BN

```
model_mapper (img,img2,model,shift=64,layer_number=3,output=False)
```

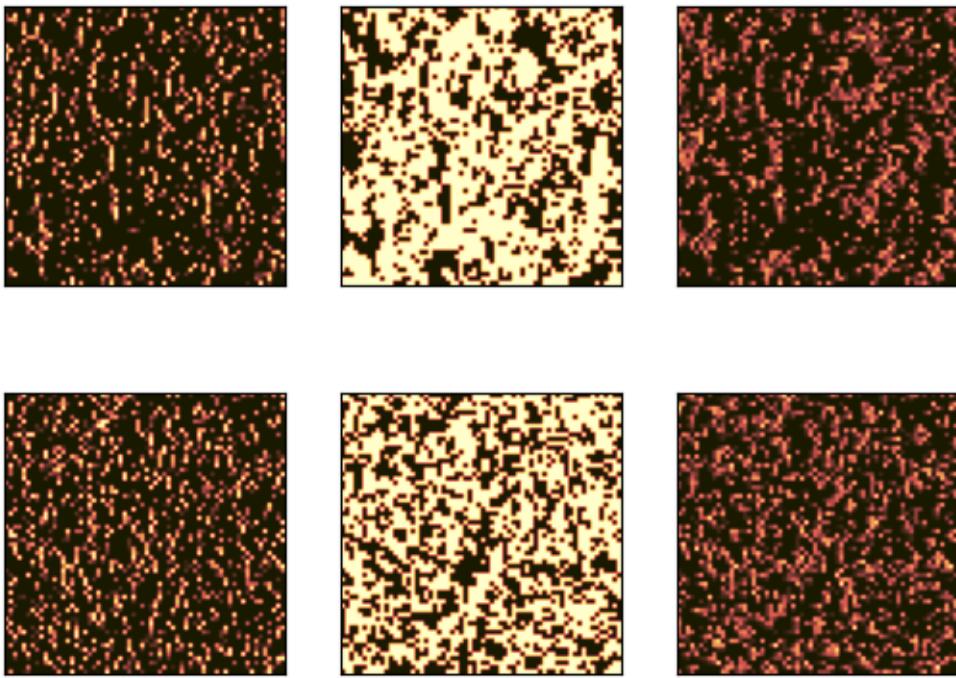


Figure 116.8: png

116.5.5 5. CONV (filters=4,kernel_size=(5,5),padding='same')

```
model_mapper (img,img2,model,shift=64,ncols=4,layer_number=4,output=True)
```

116.5.5.1 Computed with the Following Kernels

```
kernel_printer(model,layer_number=4,path=f"../plots/fmaps/")
```

4

116.5.6 6. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=5,output=True)
```

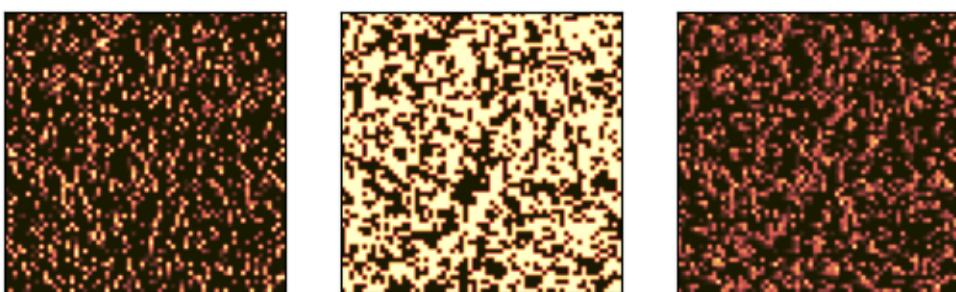
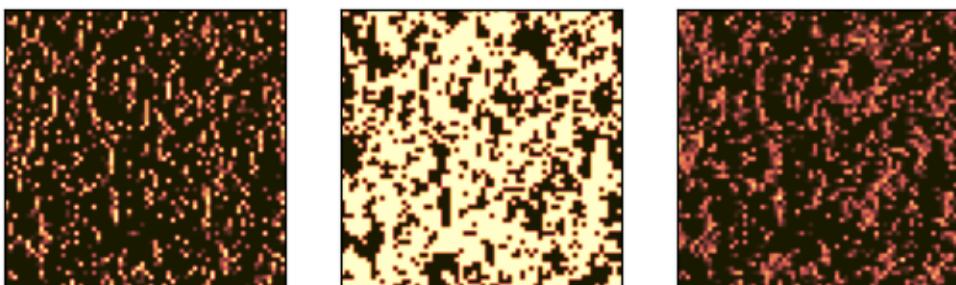


Figure 116.9: png

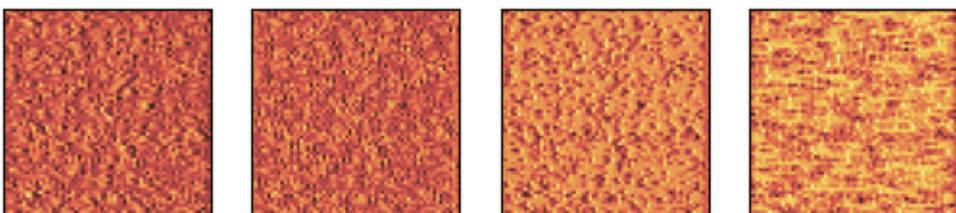
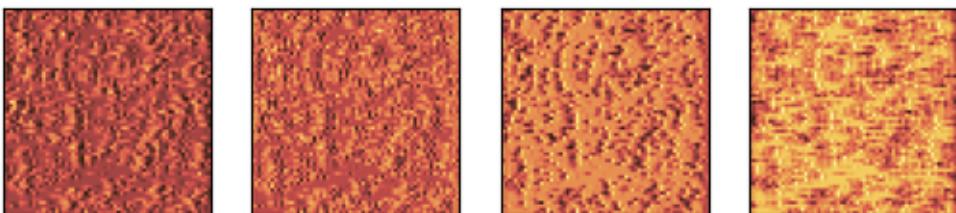


Figure 116.10: png

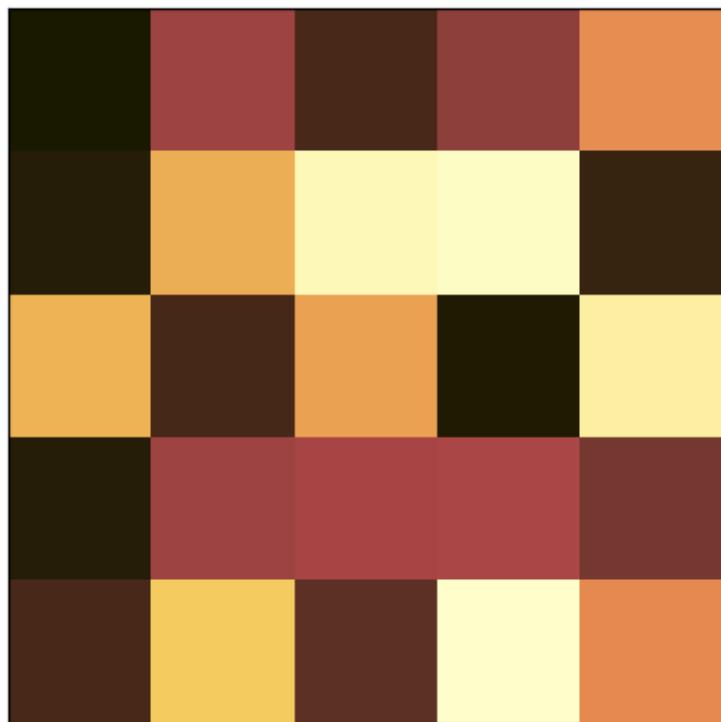


Figure 116.11: png

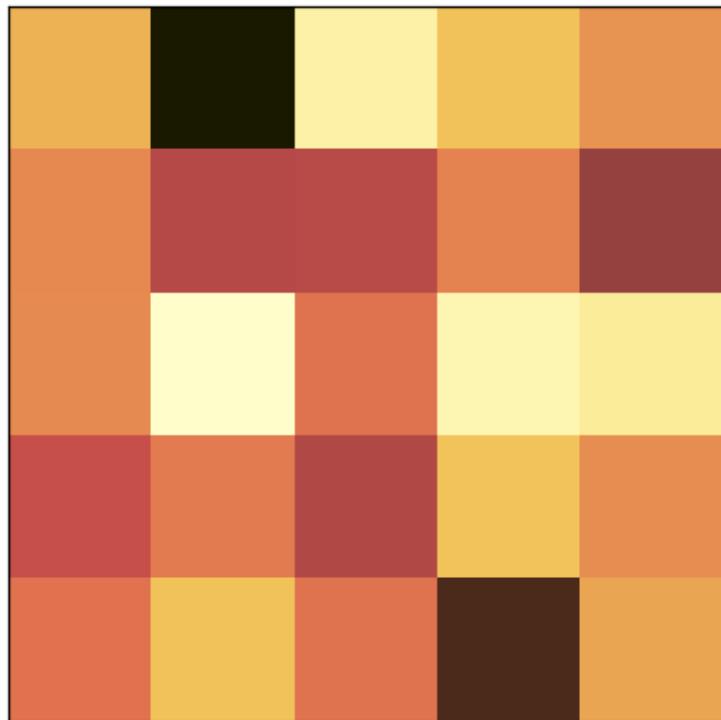


Figure 116.12: png

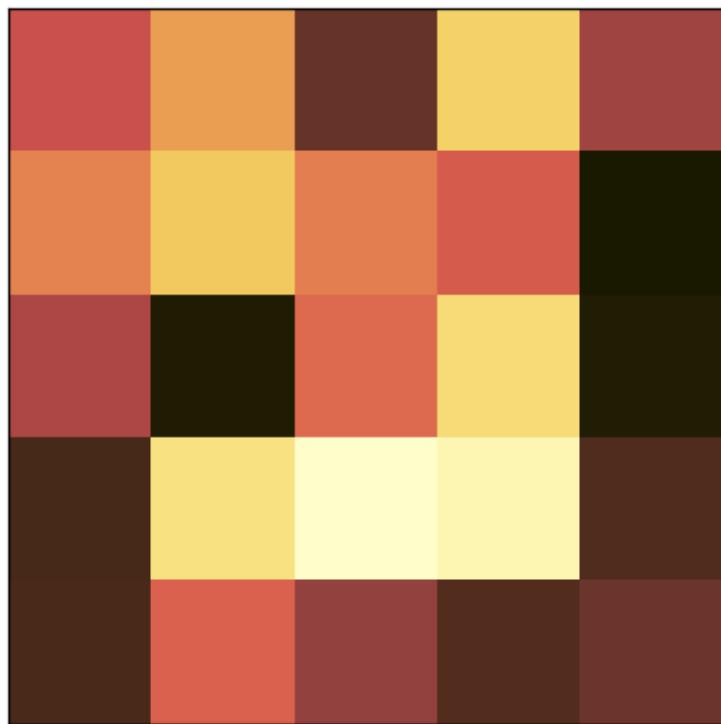


Figure 116.13: png

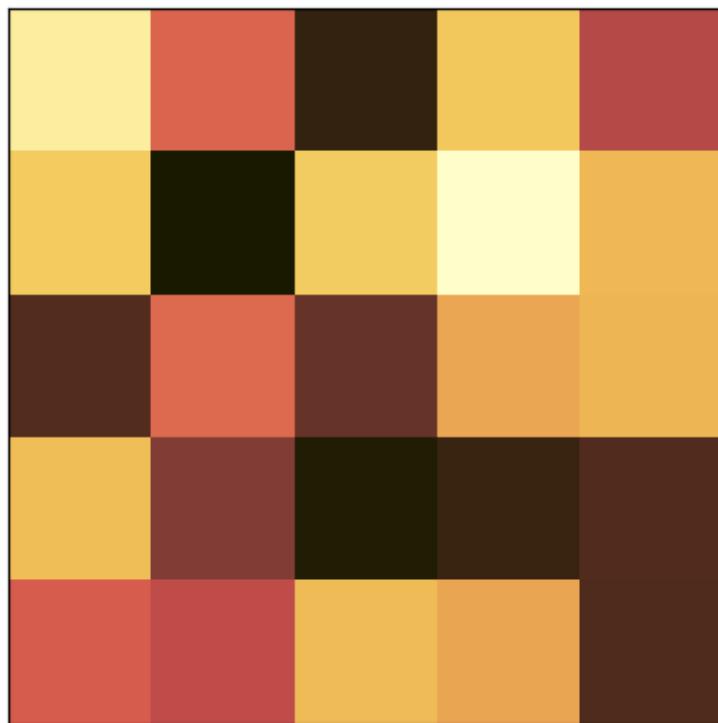


Figure 116.14: png

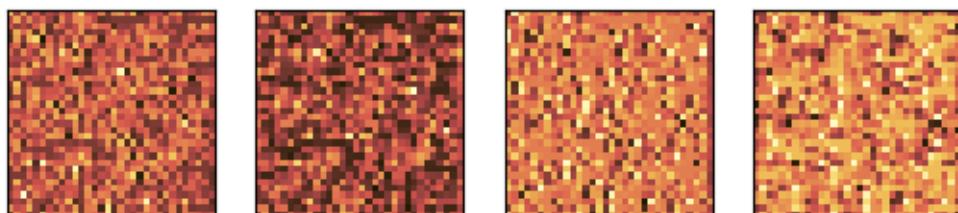
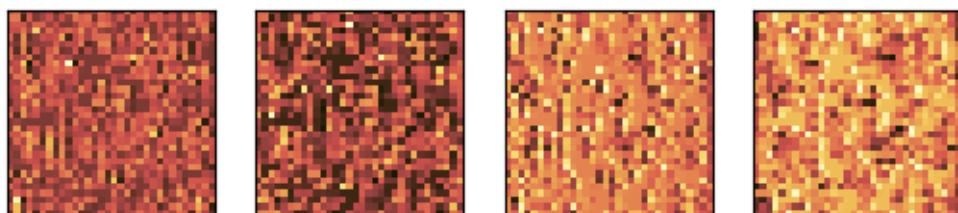


Figure 116.15: png

116.5.7 7. ReLU

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=6,output=True)
```

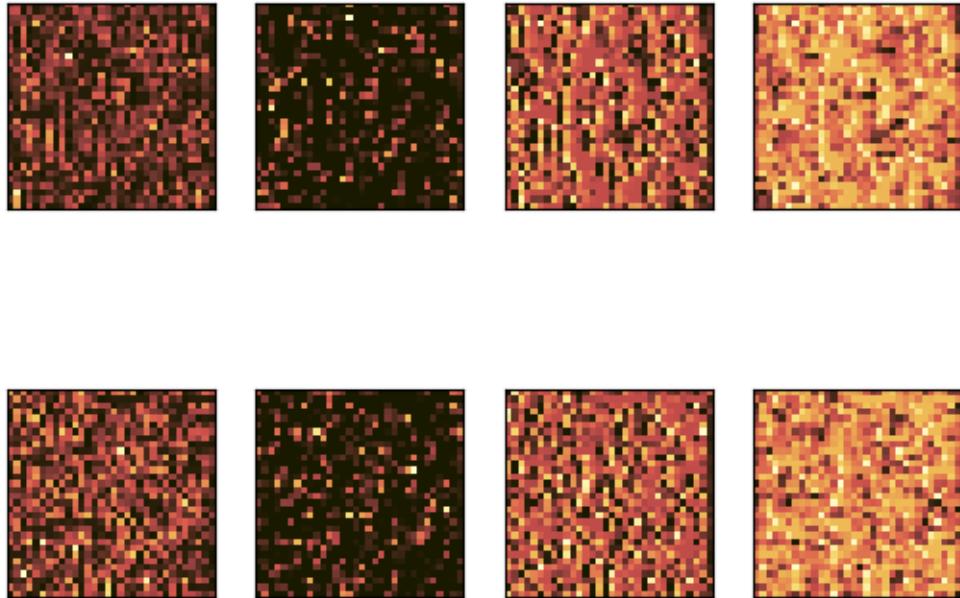


Figure 116.16: png

116.5.8 8. BN

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=7,output=False)
```

116.5.9 9. CONV (filters=6, kernel_size=(5,5),padding='same')

```
model_mapper (img,img2,model,shift=32,ncols=6,layer_number=8,output=True)
```

```
kernel_printer(model,layer_number=8,path=f"../plots/fmaps/")
```

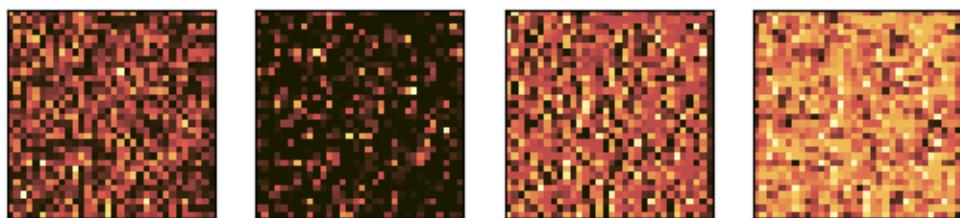
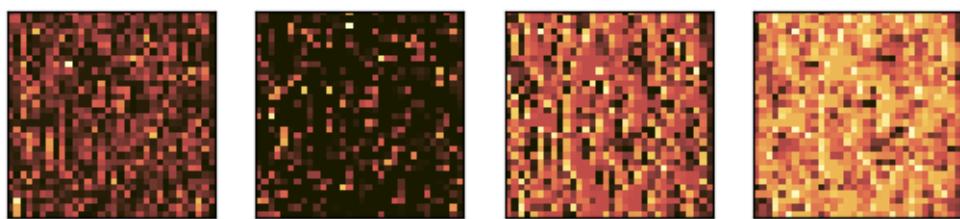


Figure 116.17: png

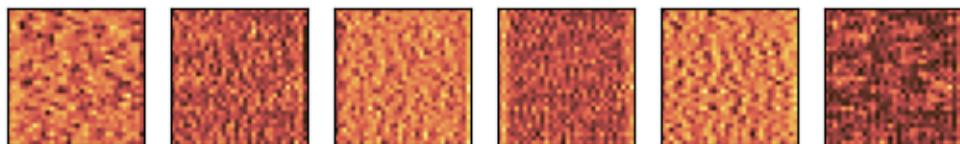
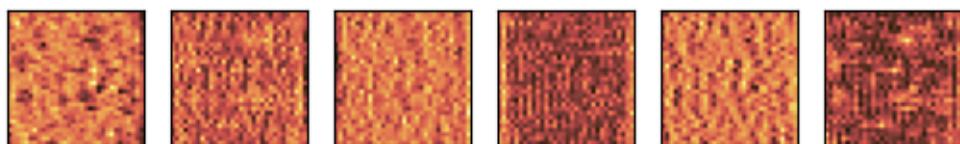


Figure 116.18: png

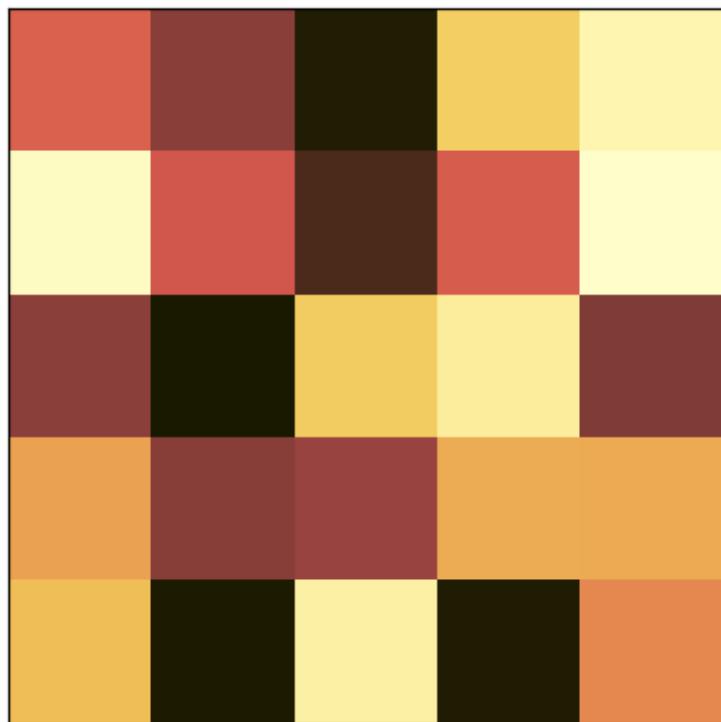


Figure 116.19: png

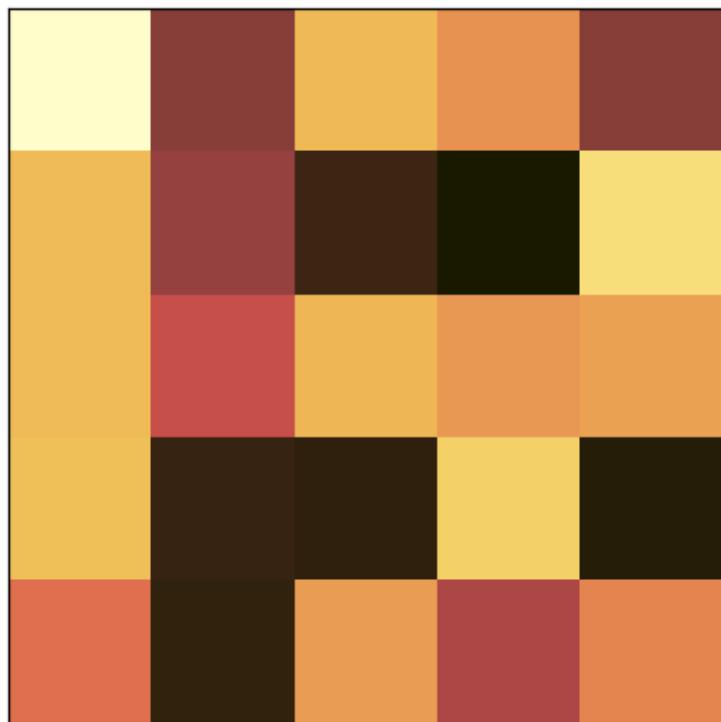


Figure 116.20: png

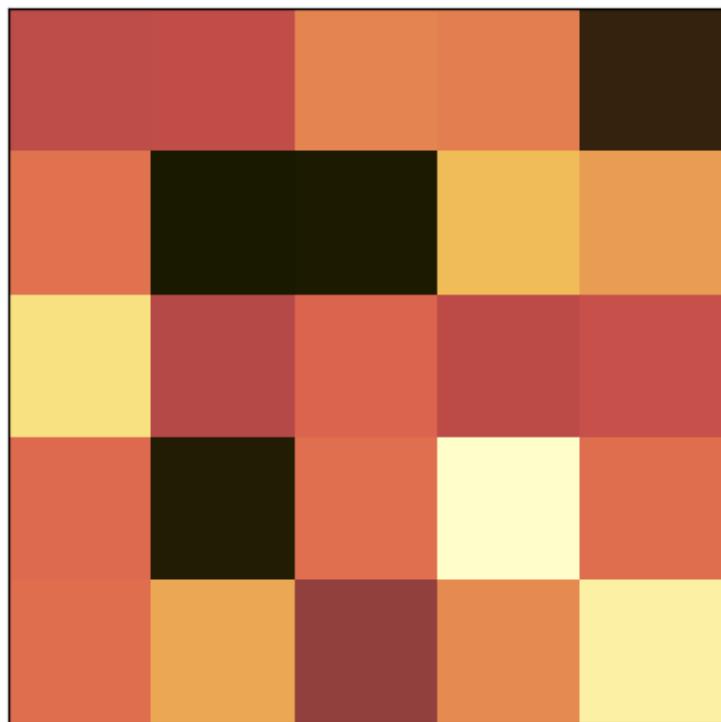


Figure 116.21: png

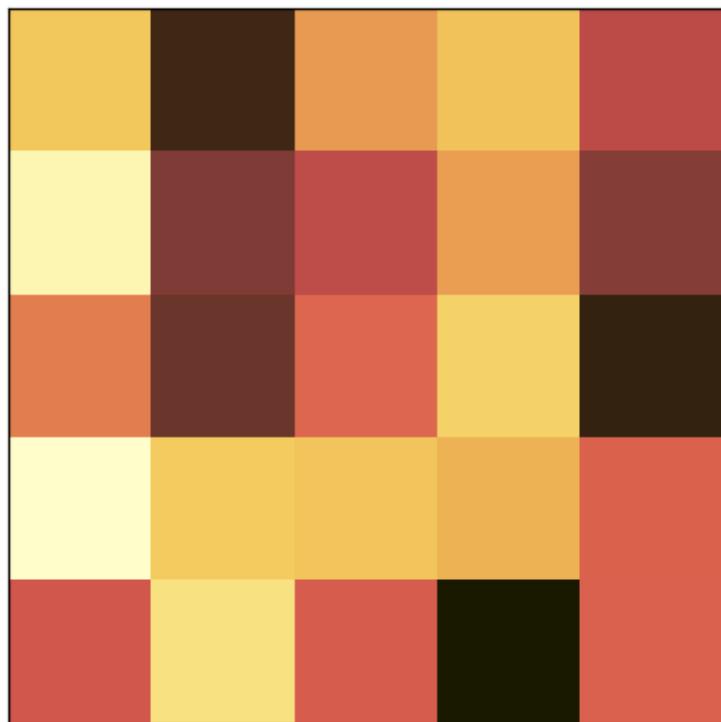


Figure 116.22: png

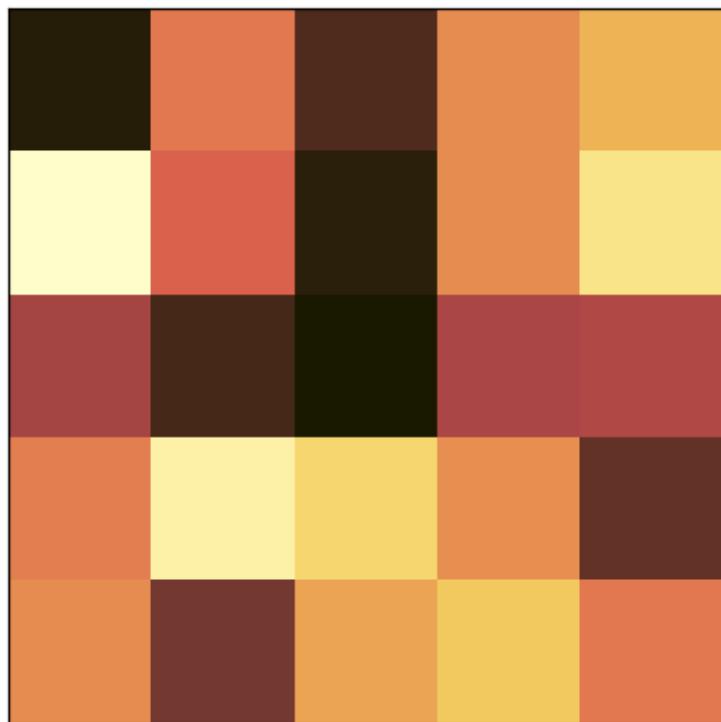


Figure 116.23: png

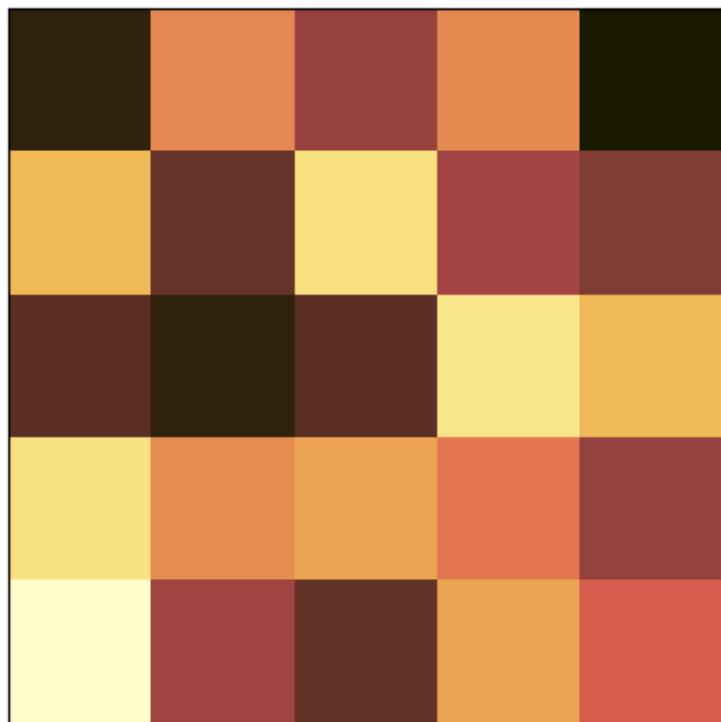


Figure 116.24: png

116.5.10 10. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=9,output=True)
```

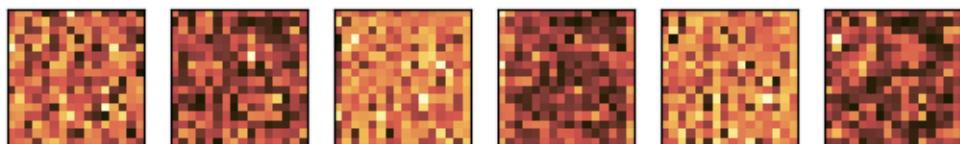
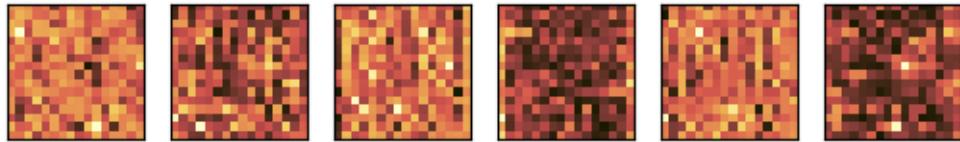


Figure 116.25: png

116.5.11 11. ReLU

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=10,output=True)
```

116.5.12 12. BN

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=11,output=False)
```

116.5.13 13. AVGPOOL

```
model_mapper (img,img2,model,layer_number=12,post_avgpool=True,flattened=True,output=True)
```

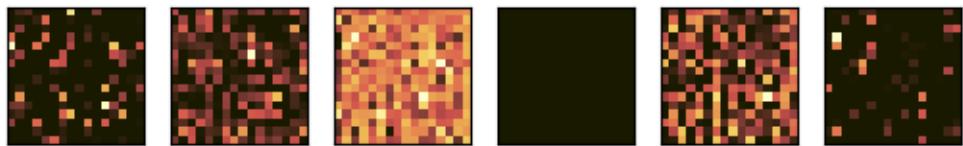
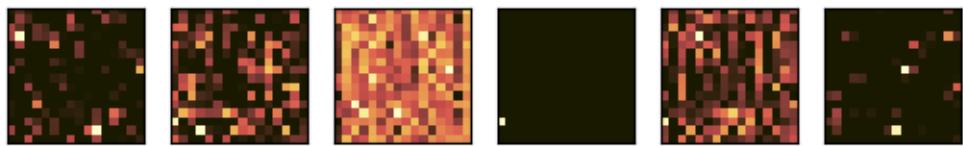


Figure 116.26: png

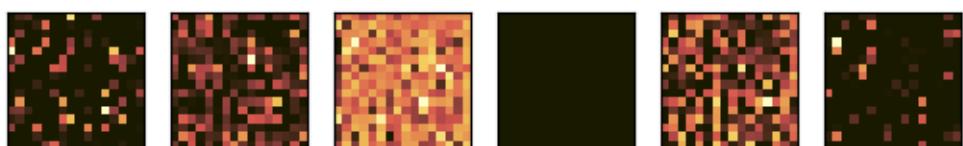
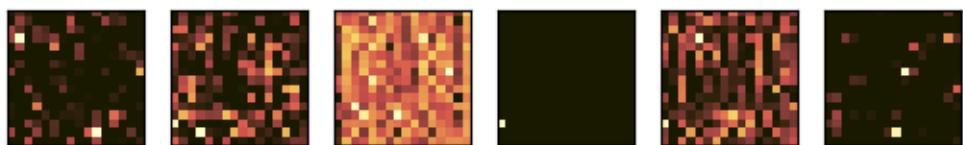


Figure 116.27: png



Figure 116.28: png



Figure 116.29: png

116.5.14 14. DO (0.1) (without layout optimiser)

```
model_mapper (img,img2,model,layer_number=13,post_avgpool=True,flattened=True,output=False)
```



Figure 116.30: png



Figure 116.31: png

116.5.15 15. FC (units=128,activation='relu')

```
model_mapper (img,img2,model,layer_number=14,post_avgpool=True,flattened=False,output=True)
```

116.5.16 16. DO (0.1) (without layout optimiser)

```
model_mapper (img,img2,model,layer_number=15,post_avgpool=True,flattened=False,output=False)
```

116.5.17 17. FC (units=3,activation='relu')

```
model_mapper (img,img2,model,layer_number=16,post_avgpool=True,flattened=True,output=True)
```



Figure 116.32: png

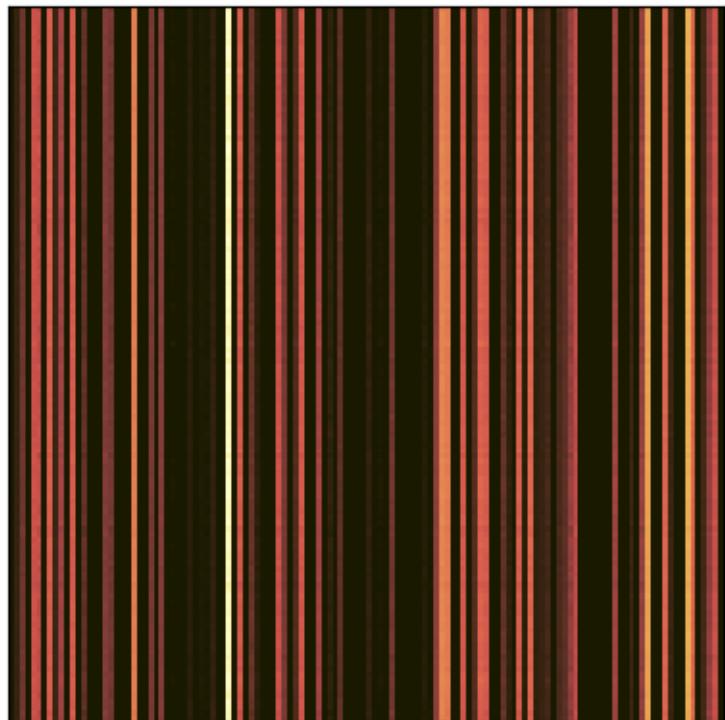


Figure 116.33: png



Figure 116.34: png

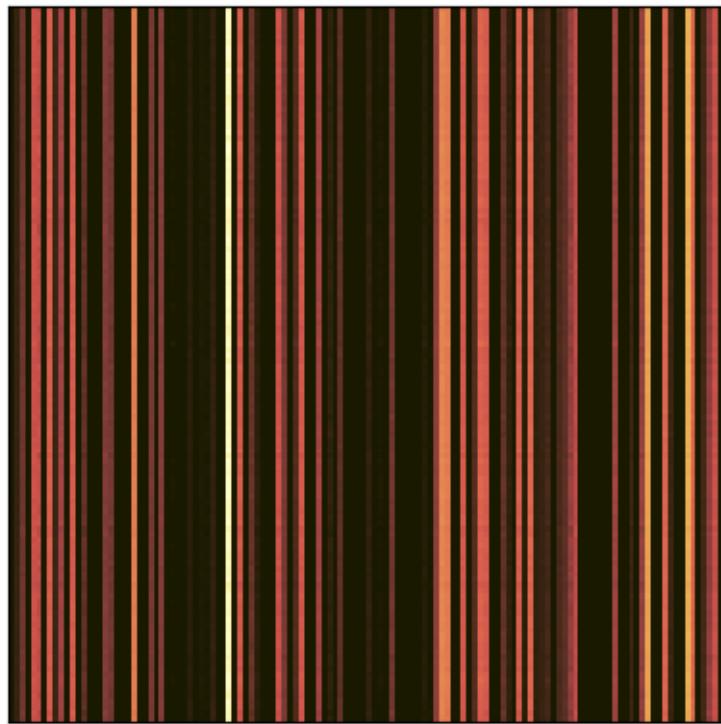


Figure 116.35: png



Figure 116.36: png



Figure 116.37: png

116.5.18 18. FLATTEN

```
model_mapper (img,img2,model,layer_number=17,post_avgpool=True,flattened=True,output=False)
```



Figure 116.38: png



Figure 116.39: png

116.5.19 19. FC (units=1,activation='linear')

```
model_mapper (img,img2,model,layer_number=18,post_avgpool=True,flattened=True,output=True)
```



Figure 116.40: png

116.6 Some notes

The batch normalisation layers do not display any change because they simply renormalise the data. They are included here for completeness.

The dropout layers don't actually activate during model predictions. This is a deliberate feature of Keras which we've leveraged; the dropout layer helps with dataset training in order to prevent overfitting, but does not eliminate further data during actual prediction applications. They are also included here for completeness.



Figure 116.41: png

117 7. Feature Map Analysis (Monochrome Data Type)

117.1 Import Libraries

```
import warnings
warnings.filterwarnings("ignore")

import h5py
import numpy as np

import tensorflow as tf
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import img_to_array
from keras.models import Model
import matplotlib.pyplot as plt
from cmcrameri import cm
from numpy import expand_dims

import sys
sys.path.append('..')

from src.utils import get_cluster_labels, get_ds_iters
from src.training_utils import (
    data_load,
    split_dataset,
)
from src.plot_utils import get_plot_configs
```

2024-04-16 16:15:26.430617: I tensorflow/core/util/port.cc:113] oneDNN custom operations are not enabled. To enable them, set the environment variable `TF_ENABLE_CUSTOM_OPS=1`.
2024-04-16 16:15:26.456785: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized with `oneDNN` data flow on `AVX2` `AVX_VNNI` `FMA`, in other operations, rebuild TensorFlow with `TF_ENABLE_CUSTOM_OPS=1`.
2024-04-16 16:15:26.925737: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warnings: The input tensor has type float32, which is not supported by the TensorRT engine. The input tensor will be converted to float16.

117.2 Load Datasets

```
hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.050_density_0.25.h5", "r")
iters = get_ds_iters(hf.keys())
img = hf[f"conf_{iters[300]}"]
img = np.array(img)
img[img>0]=1
img = img.reshape((img.shape[0], img.shape[1], 1))

hf = h5py.File(f"../data/no-rolling/dataset_tumble_0.157_density_0.25.h5", "r")
iters = get_ds_iters(hf.keys())
img2 = hf[f"conf_{iters[300]}"]
img2 = np.array(img2)
img2[img2>0]=1
img2 = img2.reshape((img2.shape[0], img2.shape[1], 1))
```

117.3 Example Datamaps ($P_t \in \{0.050, 0.157\}$, $\rho = 0.25$)

```
plt.matshow(img, cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
plt.savefig('../plots/fmaps/input_0.050.svg', bbox_inches='tight', pad_inches=-0.1)
plt.matshow(img2, cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
plt.savefig('../plots/fmaps/input_0.157.svg', bbox_inches='tight', pad_inches=-0.1)
```

117.4 Set Up GPU and Load Model

Reminder: the following commands need to be ran in console in order to employ GPU. This is not strictly necessary here, since running on the CPU only affects performance (and we are computing very little data here). This is nonetheless a good habit.

```
export CUDNN_PATH=$(dirname $(python -c "import nvidia.cudnn;print(nvidia.cudnn.__file__)")
export LD_LIBRARY_PATH=${CUDNN_PATH}/lib
export PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:$PATH
```

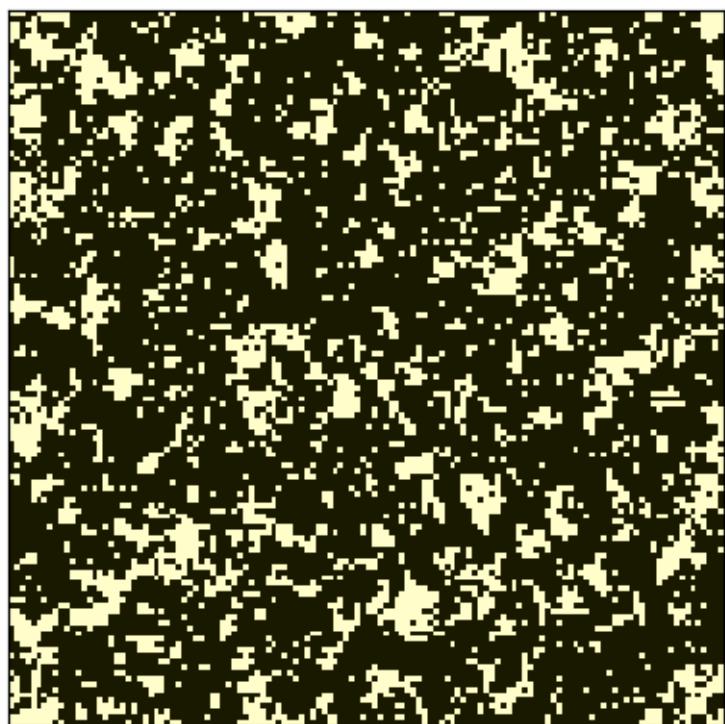


Figure 117.1: png

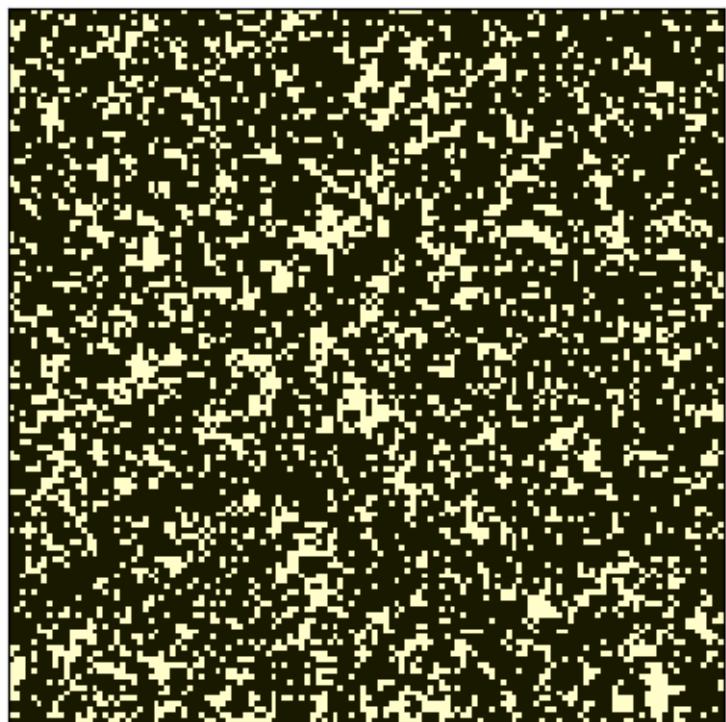


Figure 117.2: png

```
model = tf.keras.models.load_model('../models/monochrome0216.keras')
```

```
2024-04-16 16:15:29.155891: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:29.866598: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:29.866961: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:29.871670: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:29.872067: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:29.872279: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:30.087381: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:30.087495: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:30.087551: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-16 16:15:30.088606: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1928] Created
```

117.5 Plot Feature Maps

The following shows output feature maps from each architecture layer (indicated in titles). Row 0 is low tumbling rate ($P_t = 0.050$), row 1 is high tumbling rate ($P_t = 0.157$). Multiple columns indicates multiple emergent feature maps (decided by number of filters) from however many input feature maps the layer takes (dictated by previous layer output maps).

```
def model_mapper (img,img2,model,layer_number,shift=None,ncols=3,post_avgpool=False,flattened=False):
    model_mini = Model(inputs=model.inputs, outputs=model.layers[layer_number].output)

    feature_maps1 = model_mini.predict(img, verbose=0)
    feature_maps2 = model_mini.predict(img2, verbose=0)

    if post_avgpool == False and flattened == False:
        i = 2
        j = ncols
        for idx in range(i):
            for jdx in range(j):
                ax = plt.subplot(i, j, idx*j+jdx+1)
                ax.set_xticks([])
                ax.set_yticks([])
                if idx == 0:
                    plt.imshow(feature_maps1[:shift, :, 0, jdx], cmap='cmc.lajolla')
                else:
                    plt.imshow(feature_maps2[:shift, :, 0, jdx], cmap='cmc.lajolla')
    if output == True:
```

```

        plt.savefig(path+f"layer_{layer_number}.svg",bbox_inches='tight')

if post_avgpool == True and flattened == False: #plots images which do not output mult
    plt.matshow(feature_maps1[:, :],cmap='cmc.lajolla')
    plt.xticks([])
    plt.yticks([])
    if output == True:
        plt.savefig(path+f"layer_{layer_number}_im1.svg",bbox_inches='tight')
plt.matshow(feature_maps2[:, :],cmap='cmc.lajolla')
plt.xticks([])
plt.yticks([])
if output == True:
    plt.savefig(path+f"layer_{layer_number}_im2.svg",bbox_inches='tight')

if post_avgpool == True and flattened == True: #rotates images which do not output mul
    plt.matshow(np.rot90(feature_maps1[:, :]),cmap='cmc.lajolla')
    plt.xticks([])
    plt.yticks([])
    if output == True:
        plt.savefig(path+f"layer_{layer_number}_im1.svg",bbox_inches='tight')
    plt.matshow(np.rot90(feature_maps2[:, :]),cmap='cmc.lajolla')
    plt.xticks([])
    plt.yticks([])
    if output == True:
        plt.savefig(path+f"layer_{layer_number}_im2.svg",bbox_inches='tight')

def kernel_printer(model,layer_number=0,path=f"../plots/fmaps/"):
    filters, biases = model.layers[layer_number].get_weights()
    print (filters.shape[-1])
    for k in range(filters.shape[-1]):
        f = filters[:, :, :, k]
        plt.matshow(f[:, :, 0],cmap="cmc.lajolla")
        plt.xticks([])
        plt.yticks([])
        plt.savefig(path+f"kernel_{k}_layer_{layer_number}.svg",bbox_inches='tight')

```

117.5.1 1. CONV

(filters=3,kernel_size=(3,3),padding='same',input_shape=shape)

```
model_mapper (img,img2,model,shift=None,layer_number=0,output=True)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1713280530.466361    8794 service.cc:145] XLA service 0x78dc1c002cf0 initialized
I0000 00:00:1713280530.466396    8794 service.cc:153] StreamExecutor device (0): NVIDIA Gel
2024-04-16 16:15:30.560875: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:465] L
I0000 00:00:1713280531.625524    8794 device_compiler.h:188] Compiled cluster using XLA! Th
```

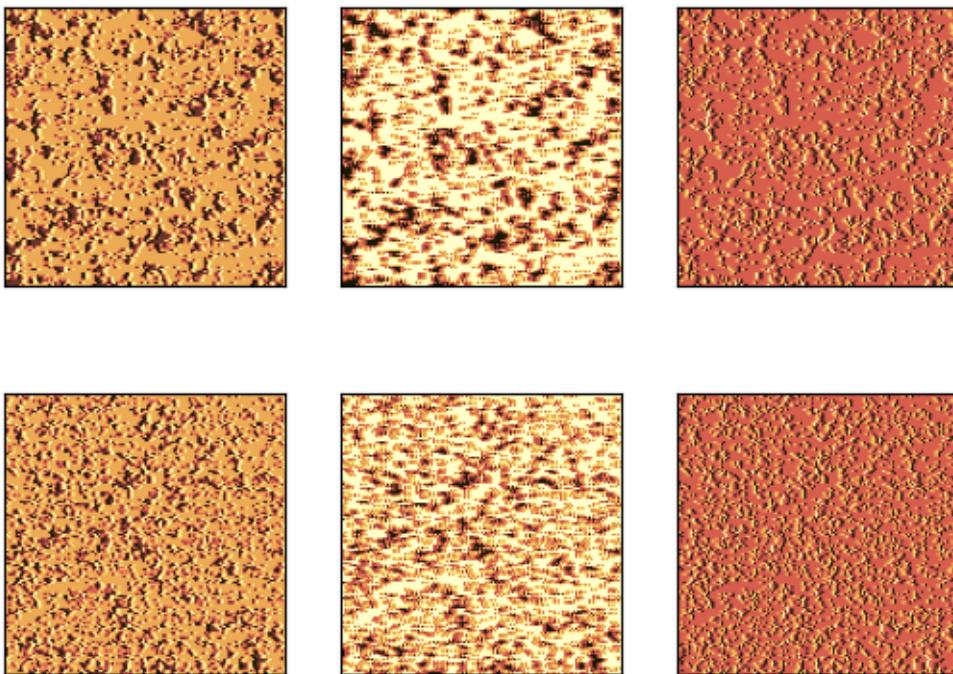


Figure 117.3: png

117.5.1.1 Computed with the Following Kernels

```
kernel_printer(model,layer_number=0,path=f"../plots/fmaps/")
```

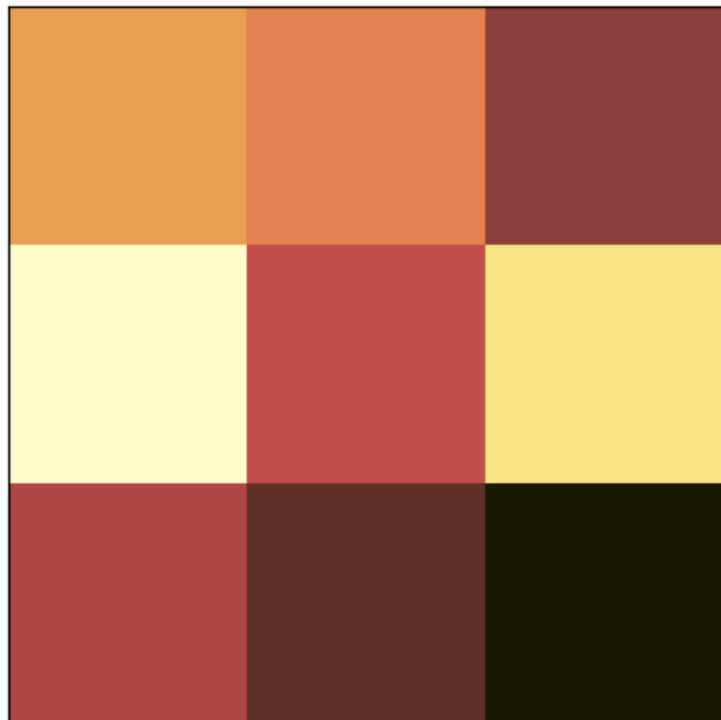


Figure 117.4: png

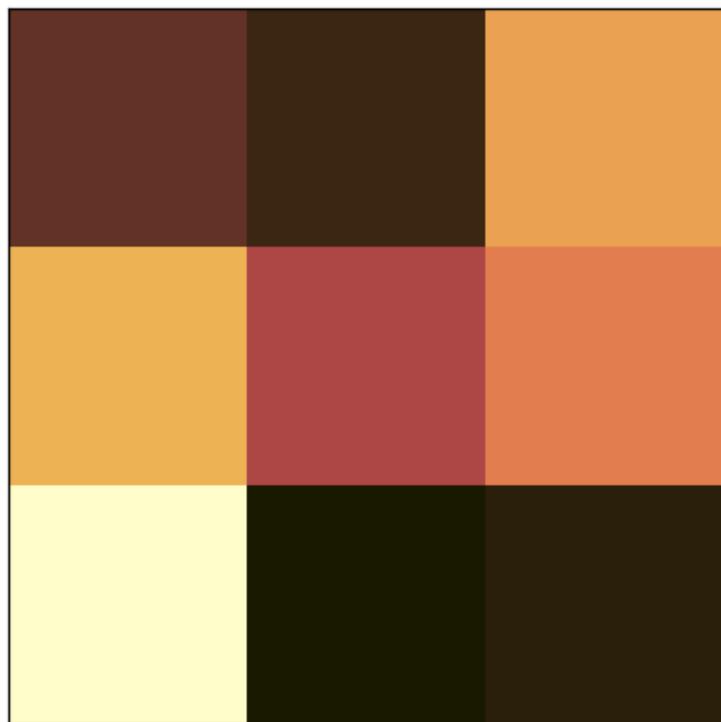


Figure 117.5: png

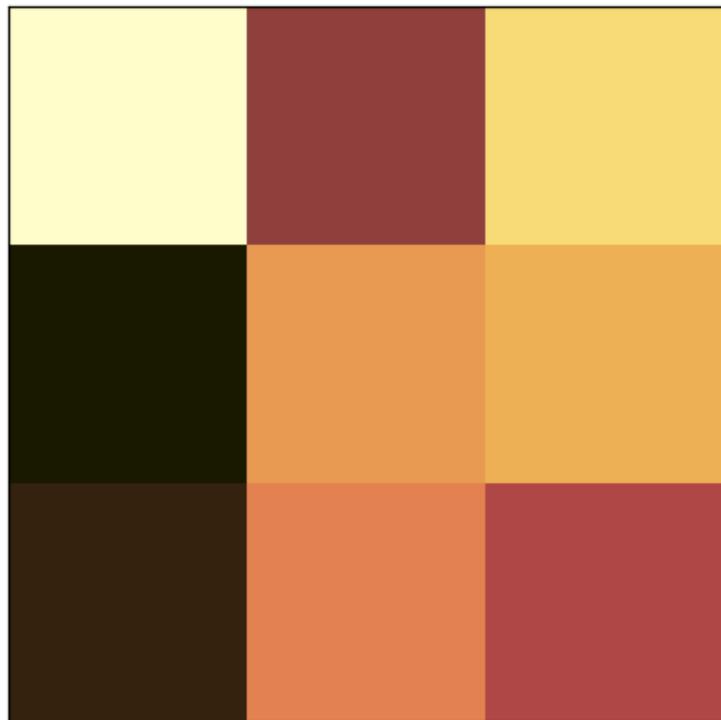


Figure 117.6: png

117.5.2 2. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=64,layer_number=1,output=True)
```

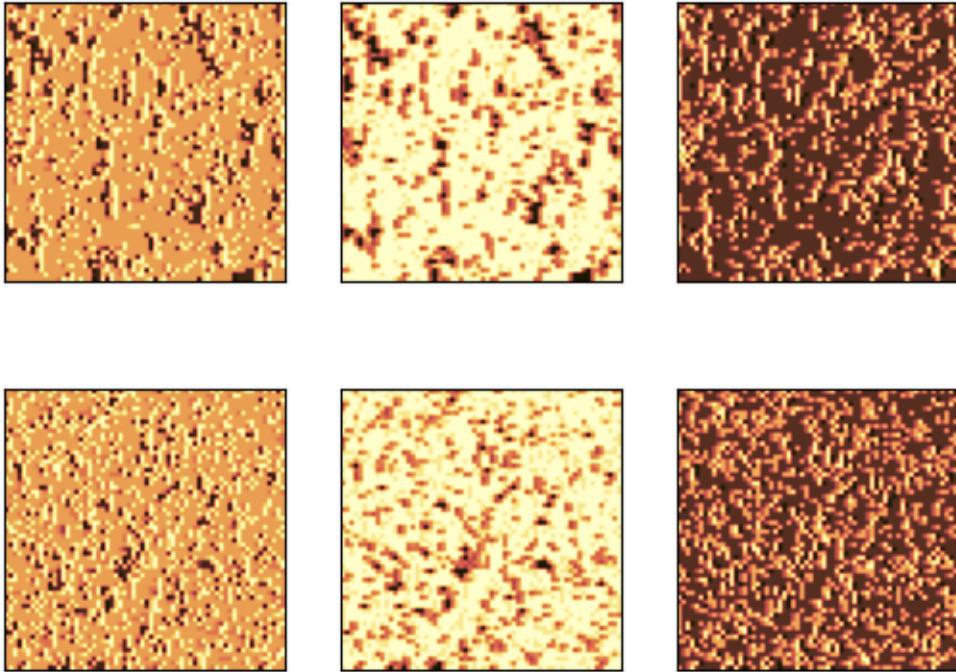


Figure 117.7: png

117.5.3 3. ReLU

```
model_mapper (img,img2,model,shift=64,layer_number=2,output=True)
```

2024-04-16 16:15:32.456222: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268

117.5.4 4. BN

```
model_mapper (img,img2,model,shift=64,layer_number=3,output=False)
```

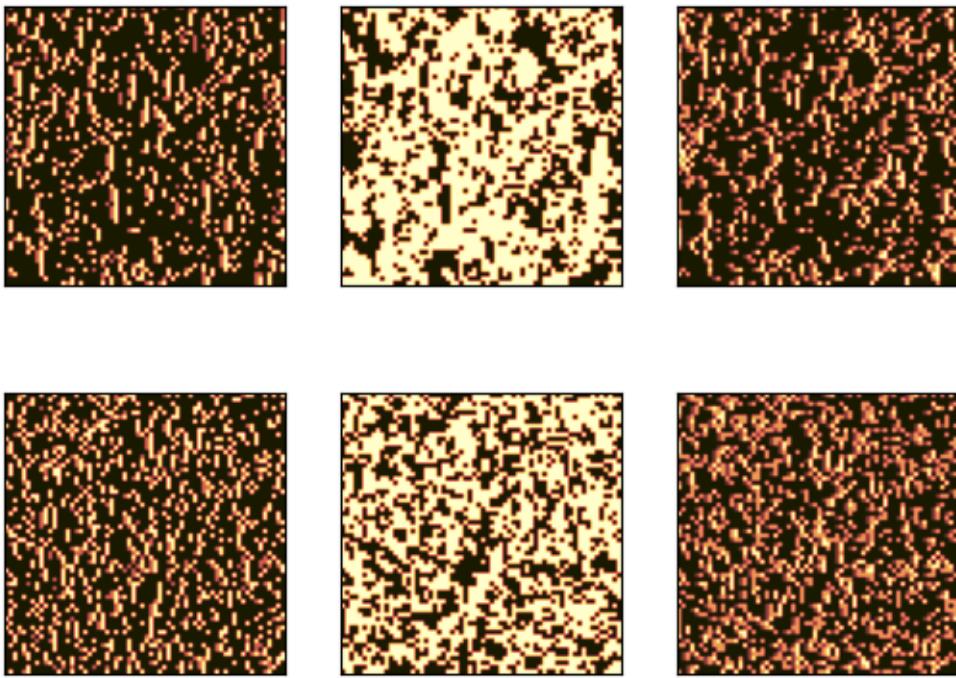


Figure 117.8: png

117.5.5 5. CONV (filters=4,kernel_size=(5,5),padding='same')

```
model_mapper (img,img2,model,shift=64,ncols=4,layer_number=4,output=True)
```

117.5.5.1 Computed with the Following Kernels

```
kernel_printer(model,layer_number=4,path=f"../plots/fmaps/")
```

4

117.5.6 6. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=5,output=True)
```

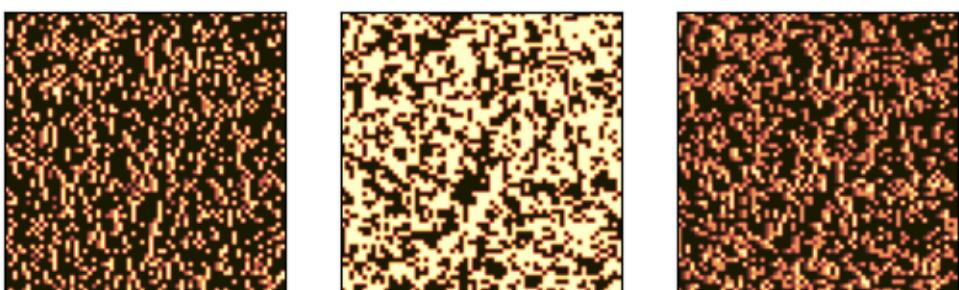
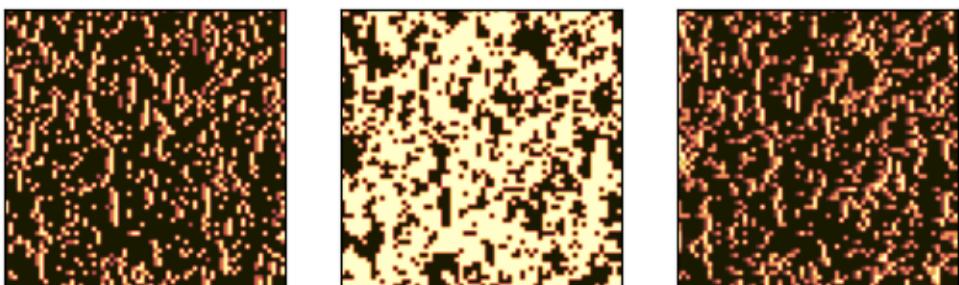


Figure 117.9: png

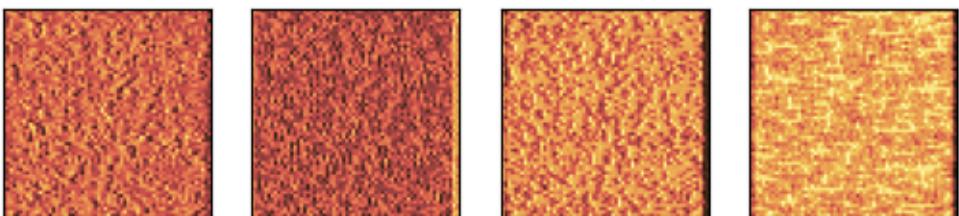
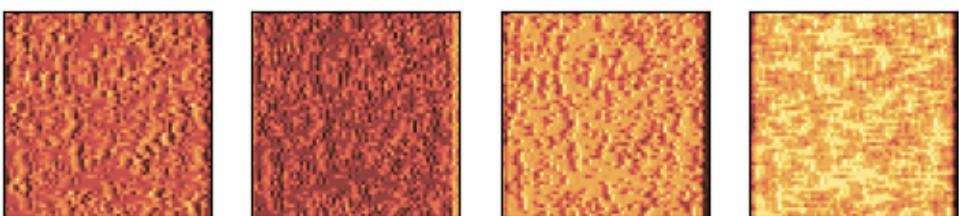


Figure 117.10: png

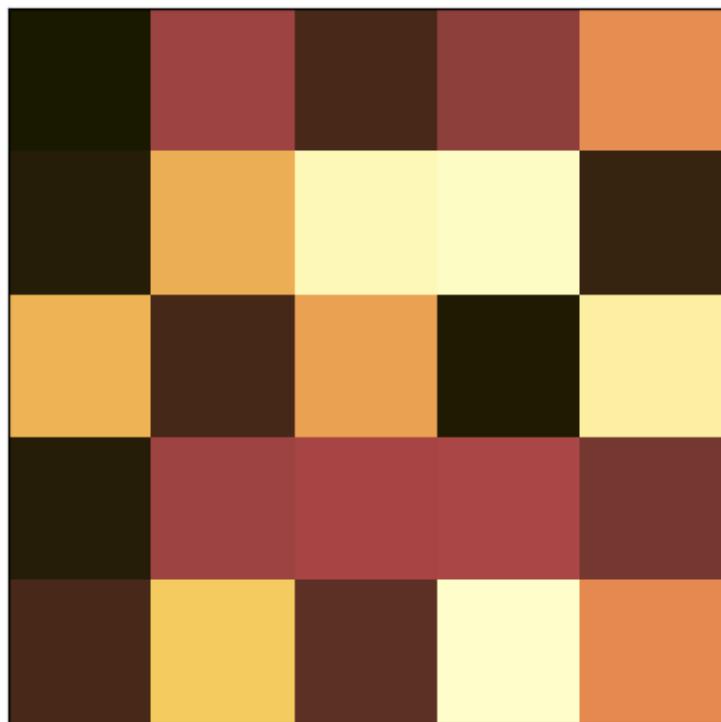


Figure 117.11: png

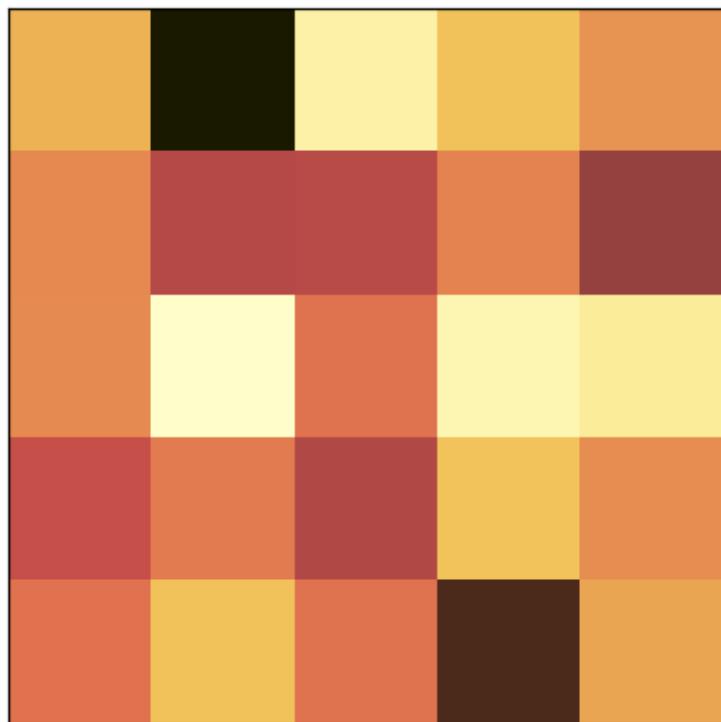


Figure 117.12: png

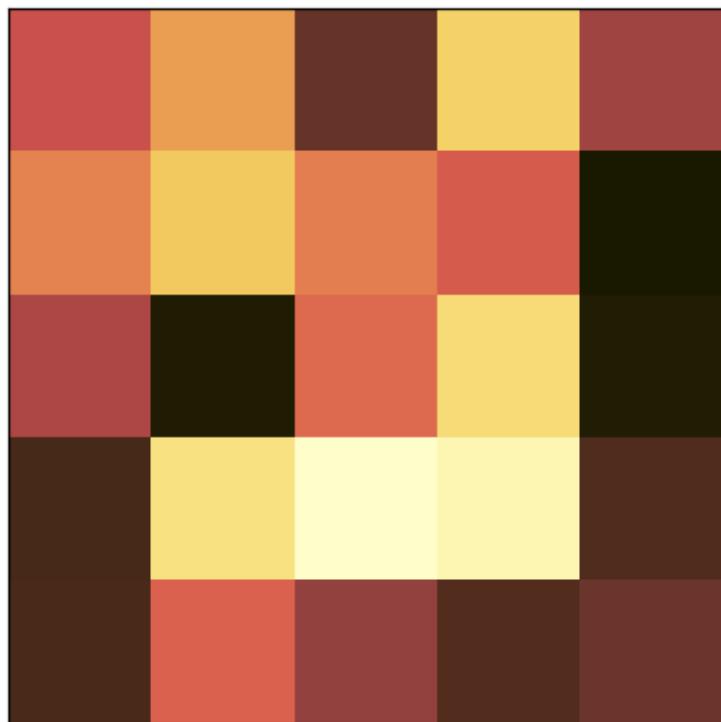


Figure 117.13: png

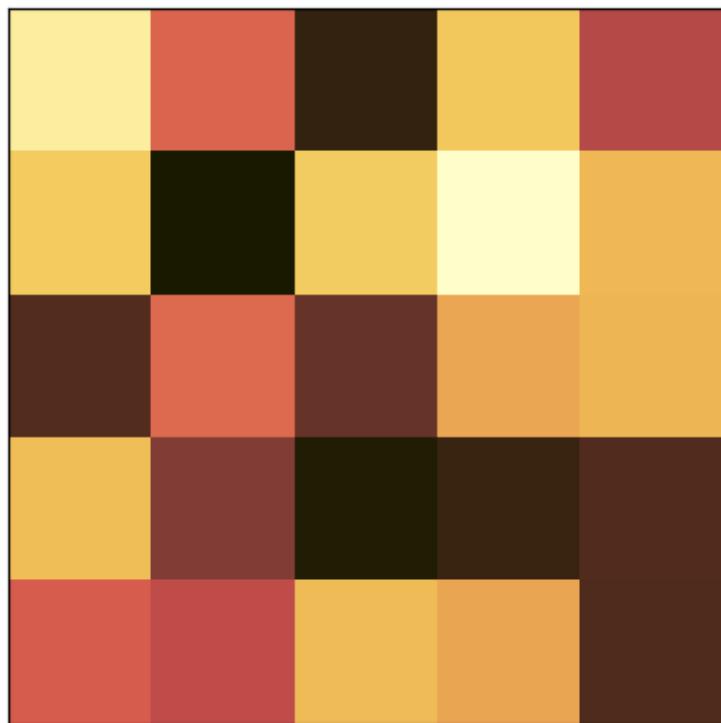


Figure 117.14: png

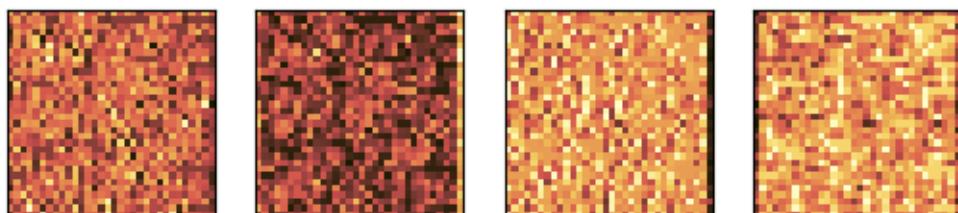
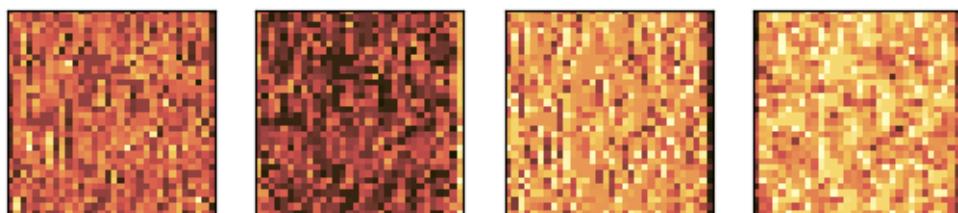


Figure 117.15: png

117.5.7 7. ReLU

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=6,output=True)
```

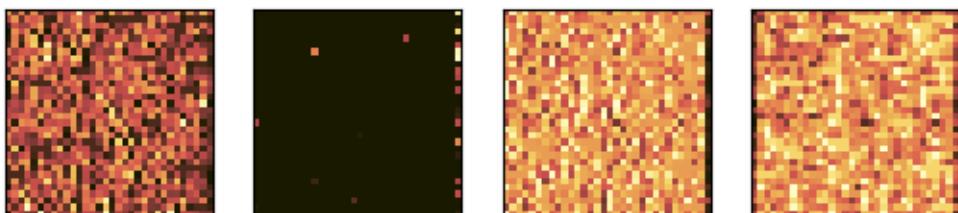
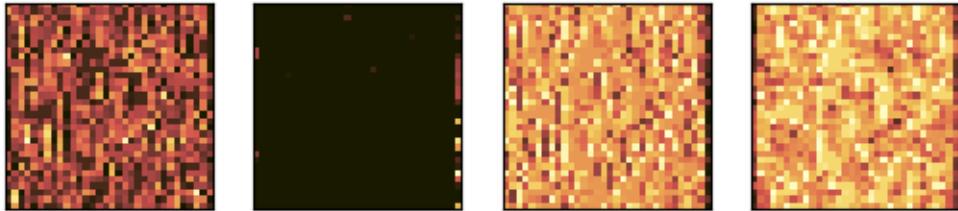


Figure 117.16: png

117.5.8 8. BN

```
model_mapper (img,img2,model,shift=32,ncols=4,layer_number=7,output=False)
```

117.5.9 9. CONV (filters=6, kernel_size=(5,5),padding='same')

```
model_mapper (img,img2,model,shift=32,ncols=6,layer_number=8,output=True)
```

```
kernel_printer(model,layer_number=8,path=f"../plots/fmaps/")
```

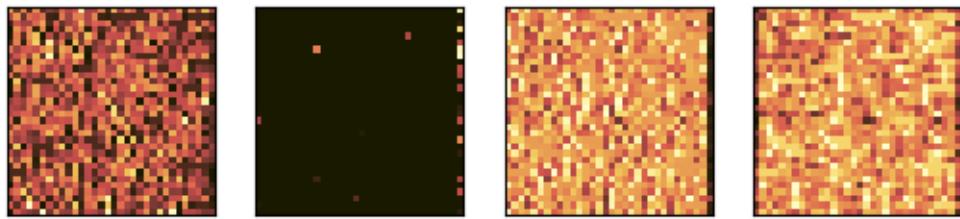
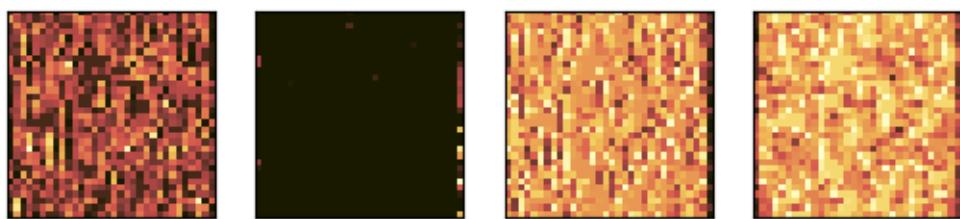


Figure 117.17: png

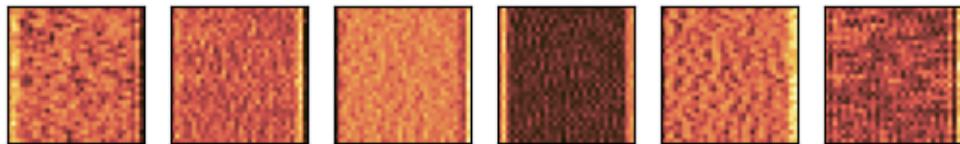
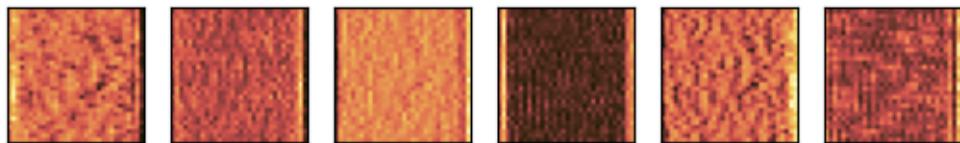


Figure 117.18: png

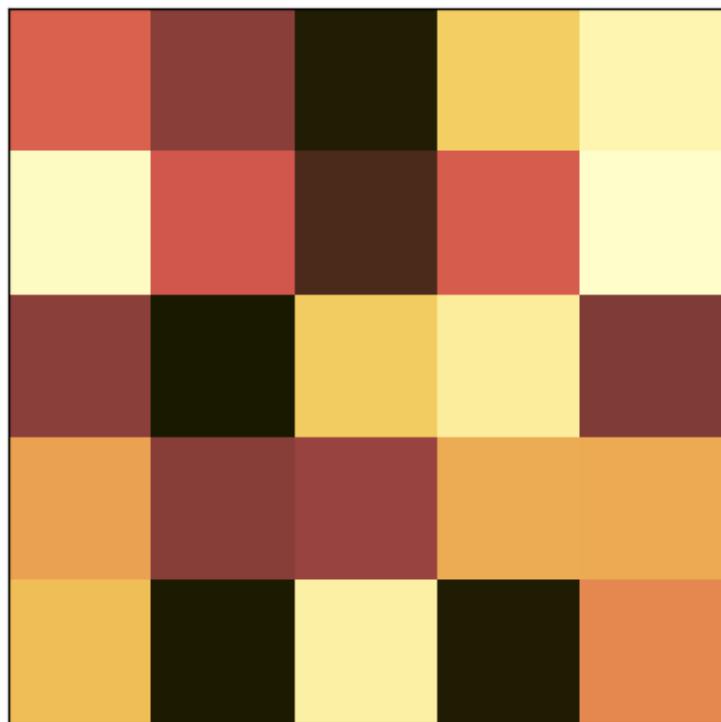


Figure 117.19: png

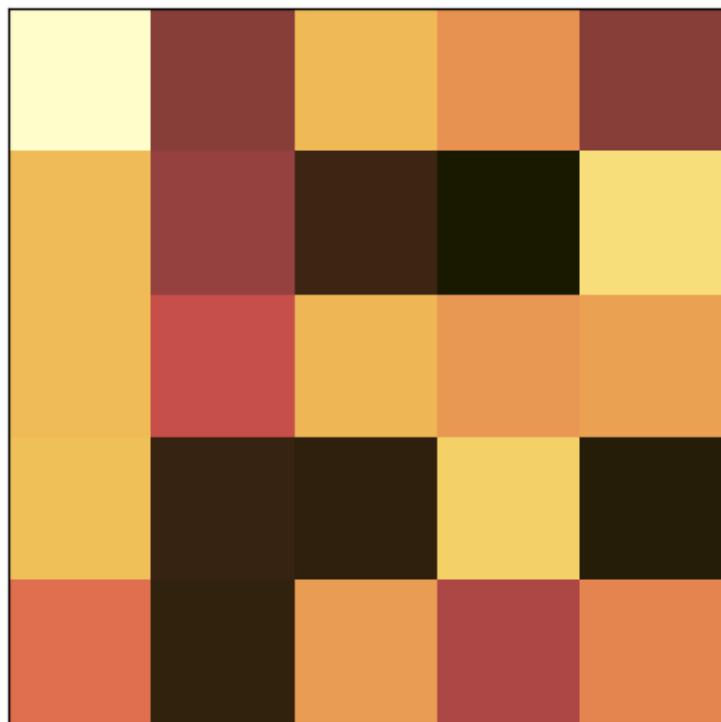


Figure 117.20: png

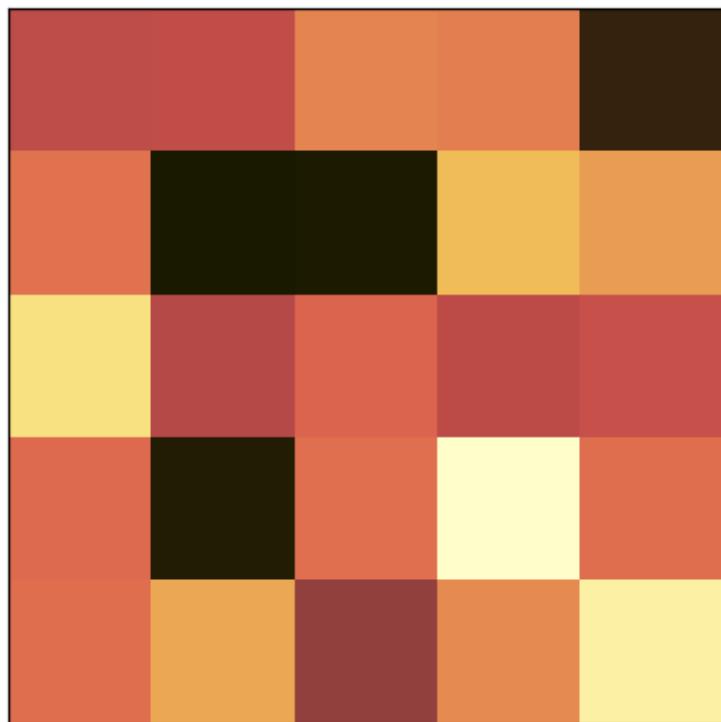


Figure 117.21: png

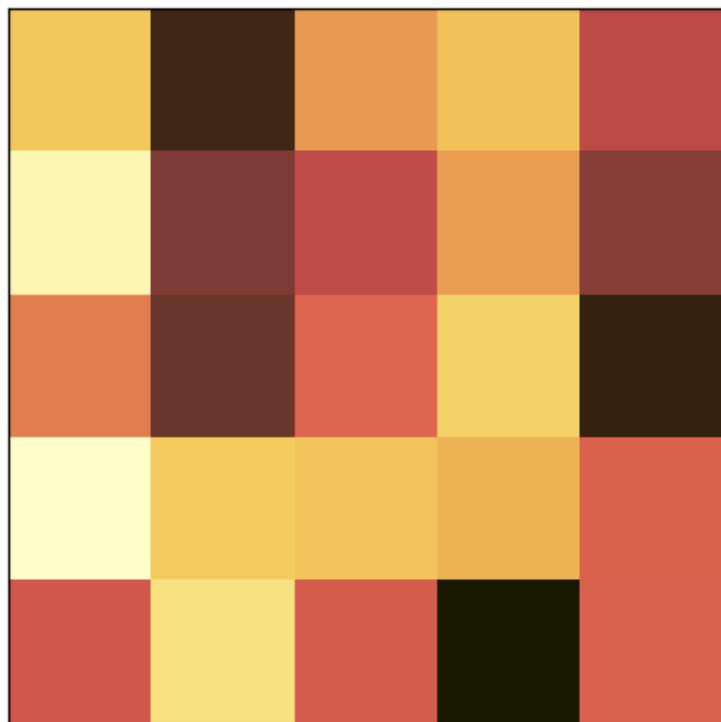


Figure 117.22: png

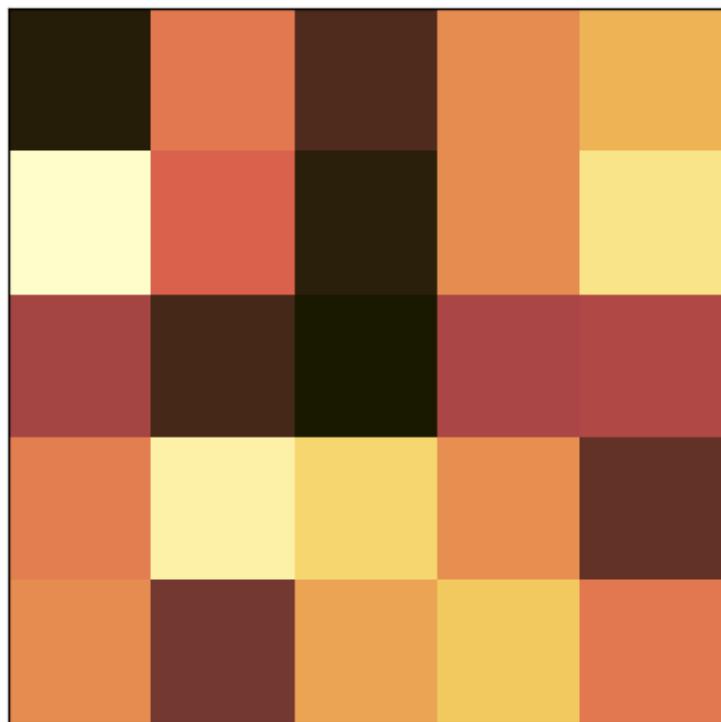


Figure 117.23: png

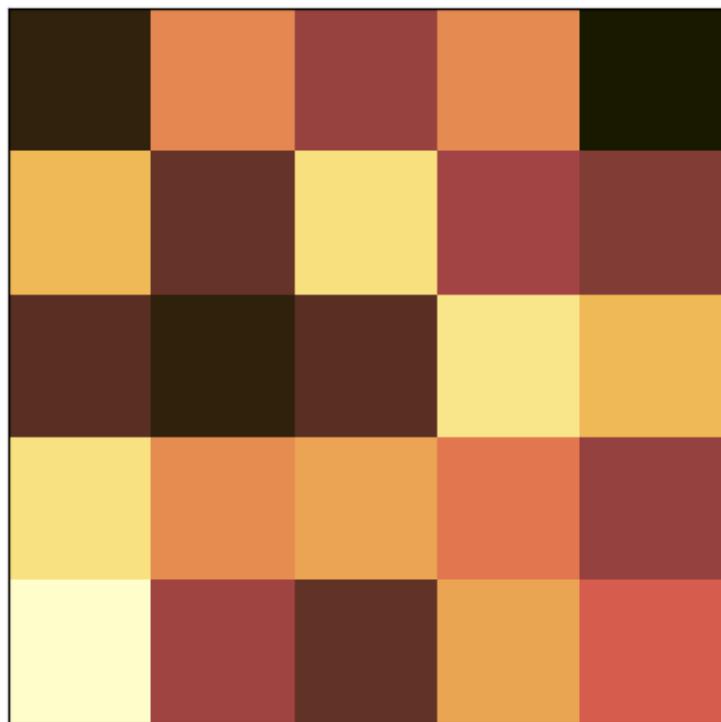


Figure 117.24: png

117.5.10 10. MAXPOOL (pool_size=(2,2),padding='same')

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=9,output=True)
```

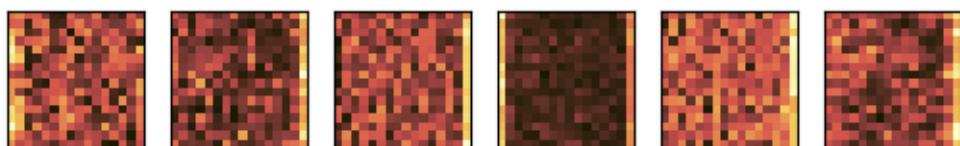
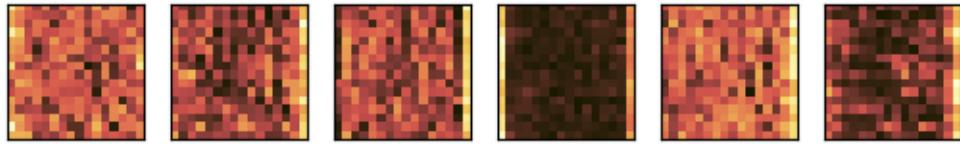


Figure 117.25: png

117.5.11 11. ReLU

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=10,output=True)
```

117.5.12 12. BN

```
model_mapper (img,img2,model,shift=16,ncols=6,layer_number=11,output=False)
```

117.5.13 13. AVGPOOL

```
model_mapper (img,img2,model,layer_number=12,post_avgpool=True,flattened=True,output=True)
```

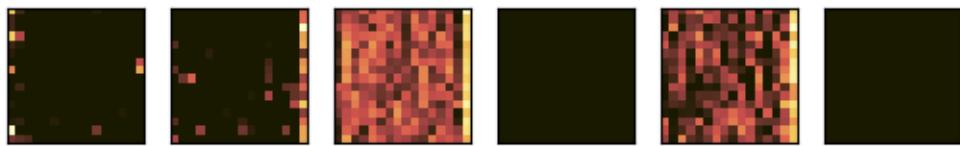


Figure 117.26: png

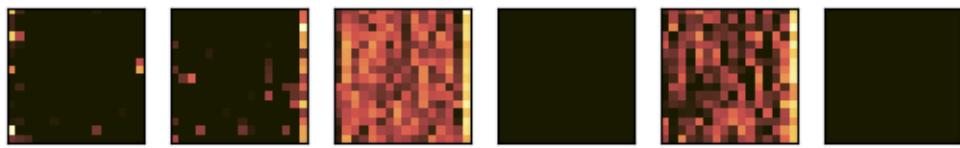


Figure 117.27: png



Figure 117.28: png



Figure 117.29: png

117.5.14 14. DO (0.1) (without layout optimiser)

```
model_mapper (img,img2,model,layer_number=13,post_avgpool=True,flattened=True,output=False)
```



Figure 117.30: png



Figure 117.31: png

117.5.15 15. FC (units=128,activation='relu')

```
model_mapper (img,img2,model,layer_number=14,post_avgpool=True,flattened=False,output=True)
```

117.5.16 16. DO (0.1) (without layout optimiser)

```
model_mapper (img,img2,model,layer_number=15,post_avgpool=True,flattened=False,output=False)
```

117.5.17 17. FC (units=3,activation='relu')

```
model_mapper (img,img2,model,layer_number=16,post_avgpool=True,flattened=True,output=True)
```



Figure 117.32: png



Figure 117.33: png



Figure 117.34: png

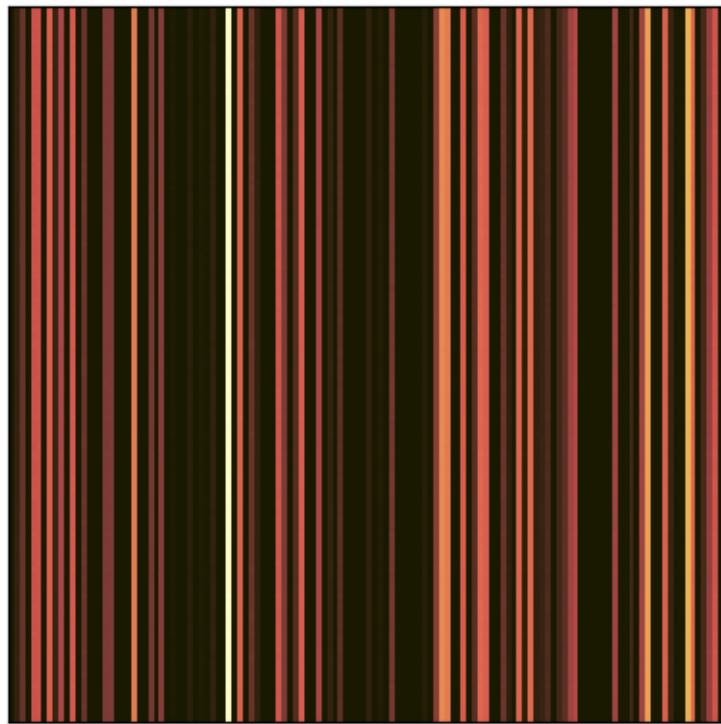


Figure 117.35: png



Figure 117.36: png



Figure 117.37: png

117.5.18 18. FLATTEN

```
model_mapper (img,img2,model,layer_number=17,post_avgpool=True,flattened=True,output=False)
```



Figure 117.38: png



Figure 117.39: png

117.5.19 19. FC (units=1,activation='linear')

```
model_mapper (img,img2,model,layer_number=18,post_avgpool=True,flattened=True,output=True)
```



Figure 117.40: png

117.6 Some notes

The batch normalisation layers do not display any change because they simply renormalise the data. They are included here for completeness.

The dropout layers don't actually activate during model predictions. This is a deliberate feature of Keras which we've leveraged; the dropout layer helps with dataset training in order to prevent overfitting, but does not eliminate further data during actual prediction applications. They are also included here for completeness.



Figure 117.41: png

118 8. Final Averaged Predictions

118.1 Import packages

```
import numpy as np
import h5py
import glob
import re
import tensorflow as tf

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.stats import pearsonr

import sys
sys.path.append('..')
from src.training_utils import data_load, extract_floats, split_dataset, predict_multi_by_

from tensorflow import keras
from keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D, BatchNormali

from cmcrameri import cm
import seaborn as sns
import pandas as pd

np.set_printoptions(precision=3, suppress=True)
```

```
2024-04-15 22:46:20.013183: I tensorflow/core/util/port.cc:113] oneDNN custom operations are
2024-04-15 22:46:20.126745: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensorflow
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild Tensorflow
2024-04-15 22:46:22.013569: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W
```

118.2 Set seed (optional)

```
fixed_seed = 216 #choose seed (comment out if not needed)

if 'fixed_seed' in locals():
    keras.utils.set_random_seed(fixed_seed)
    print("Running program with fixed seed:",fixed_seed)
else:
    print("Running program with random seed.")
```

Running program with fixed seed: 216

118.3 Setup GPU

First, follow instructions [here](#), or alternatively run:

```
for a in /sys/bus/pci/devices/*; do echo 0 | sudo tee -a $a/numa_node; done
```

We do this as a workaround for [this error](#):

```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)
print(tf.config.list_physical_devices('GPU'), tf.test.gpu_device_name())
print("TF Version:",tf.__version__)
```

[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')] /device:GPU:0
TF Version: 2.16.1

```
2024-04-15 22:46:24.894395: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.097387: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.097511: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.100146: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.100239: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.100292: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.201387: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.201471: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
```

```
2024-04-15 22:46:25.201526: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99
2024-04-15 22:46:25.201574: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1928] Created
```

118.4 Define Functions

```
def violin_plotter (v,y_val,adjustment,legloc="upper left"):  
    bins = np.logspace(-6,-1,10, base=2)*0.85  
  
    #v = prediction2.T[0]  
  
    colors = cm.batlowS(np.digitize(v, bins))  
    colors_actual = cm.batlowS(np.digitize(np.unique(y_val),bins))  
  
    fig, (ax1,ax2) = plt.subplots(nrows=2,ncols=1,figsize=(9,6),dpi=600)  
  
    df = pd.DataFrame()  
    df.insert(0, "predicted", v - y_val)  
    df.insert(1, "actual", y_val)  
  
    sns.violinplot(  
        ax=ax1,  
        data=df,  
        x="actual",  
        y="predicted",  
        color="w",  
        alpha=0.7,  
        density_norm="width",  
        linewidth=1,  
        inner="box",  
        inner_kws={"box_width": 4, "color": "0.2"},  
    )  
  
    ax1.set_xlabel("Actual turning rate")  
    ax1.set_ylabel(r"Prediction Difference $P_{\{pred\}}-P_{\{true\}}$")  
  
    std = []  
    means = []  
    overlap = []  
    std_div = []  
    accuracy = 5e-3
```

```

print ("Prediction means and standard deviations.")
for val in np.unique(y_val):
    v_mapped = v[np.where(y_val == val)]
    stdev = np.std(v_mapped)
    std.append(stdev)
    mean = np.mean(v_mapped)
    overlap.append((val + accuracy >= np.min(v_mapped)) & (val - accuracy <= np.max(v_
within_std = abs(val-mean)/stdev
print (f"Actual value {val}: Average = {mean:.5f} +- {stdev:.5f}; Expected value w
std_div.append(within_std)

print(f"With accuracy {accuracy}, overlap ratio:", np.sum(overlap)/len(overlap))
print("(Min, Max, Avg) STD:", np.min(std), np.max(std), np.mean(std))
print("Pearson's correlation coeff: ", pearsonr(y_val, v).statistic)

for val in np.unique(y_val):
    v_mapped = v[np.where(y_val == val)]
    means.append(np.mean(v_mapped))

ax2.errorbar(np.sort(np.unique(y_val)),np.abs(means-np.sort(np.unique(y_val))),yerr=(s
ax2.plot(np.sort(np.unique(y_val)),np.zeros(np.unique(y_val).shape[0]),color='red',lab

ax2.legend(loc=legloc)

counter = 0
for i in np.sort(np.unique(y_val)):
    ax2.text(i,adjustment,f"${std_div[counter]:.3f} \sigma$,ha="center")
    counter = counter + 1

ax2.set_xscale("log")
ax2.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
ax2.set_xticks(np.unique(y_val))

ax2.set_xlabel("Actual turning rate")
ax2.set_ylabel("Absolute mean prediction difference")

fig.tight_layout()

```

118.5 Import and prepare data

set model1 to have orientation, model2 to be monochrome, model3 to be scrambled

```
#all alphas: [0.016,0.023,0.034,0.050,0.073,0.107,0.157,0.231,0.340,0.500]
#all densities: [0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40,0.45,0.50,0.55,0.60,0.65,0.70,0.7
x1,y1,shape1 = data_load(alphas=[0.016,0.023,0.034,0.050,0.073,0.1067,0.157,0.231,0.340,0.
x2,y2,shape2 = data_load(alphas=[0.016,0.023,0.034,0.050,0.073,0.1067,0.157,0.231,0.340,0.
x3,y3,shape3 = data_load(alphas=[0.016,0.023,0.034,0.050,0.073,0.1067,0.157,0.231,0.340,0.
```

We have $N * \text{number of unique alpha}$ snapshots total, we split them into training set and a validation set with the ratio 80/20:

```
print("Orientation model:")
x_train1, y_train1, x_val1, y_val1 = split_dataset(x1,y1,last=int(len(x1)*1)) #len(x)*1 me
x_train2, y_train2, x_val2, y_val2 = split_dataset(x2,y2,last=int(len(x1)*1)) #len(x)*1 me
x_train3, y_train3, x_val3, y_val3 = split_dataset(x3,y3,last=int(len(x1)*1)) #len(x)*1 me
```

```
Orientation model:
Number of unique alpha: 10
Shape of x: (10000, 128, 128, 1)
Shape of y: (10000,)
Size of training data: 0
Size of validation data: 10000
Number of unique alpha: 10
Shape of x: (10000, 128, 128, 1)
Shape of y: (10000,)
Size of training data: 0
Size of validation data: 10000
Number of unique alpha: 10
Shape of x: (10000, 128, 128, 1)
Shape of y: (10000,)
Size of training data: 0
Size of validation data: 10000
```

```
fig, (ax1,ax2,ax3) = plt.subplots(nrows=1,ncols=3)
ax1.matshow(x_val1[500],cmap=plt.get_cmap(name="gnuplot",lut=5))
ax2.matshow(x_val2[500],cmap=plt.get_cmap(name="gnuplot",lut=5))
ax3.matshow(x_val3[500],cmap=plt.get_cmap(name="gnuplot",lut=5))
```

```
<matplotlib.image.AxesImage at 0x7f253eff1030>
```

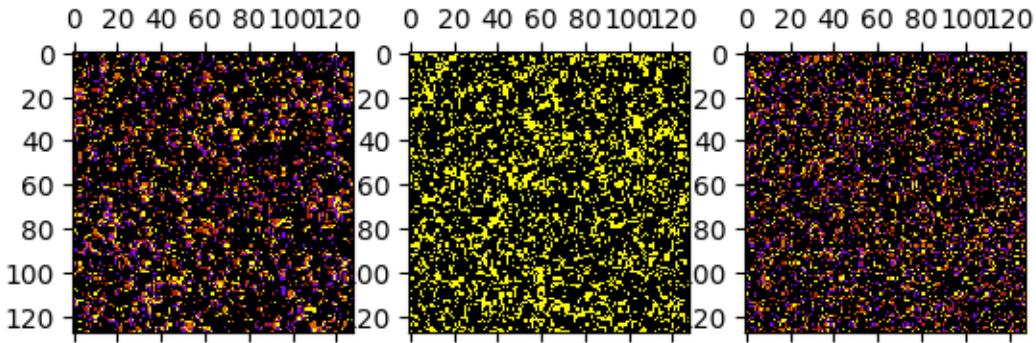


Figure 118.1: png

118.6 Predict multiple models

```
models_one = ['orientation0216','orientation0226','orientation0236','orientation0246','ori  
models_two = ['monochrome0216','monochrome0226','monochrome0236','monochrome0246','monochr  
models_three = ['scrambled0216','scrambled0226','scrambled0236','scrambled0246','scrambled  
  
one_pred_of_one, one_actualls_of_one = predict_multi_by_name(models_one,x_val1,y_val1)  
one_pred_of_two, one_actualls_of_two = predict_multi_by_name(models_one,x_val2,y_val2)  
one_pred_of_three, one_actualls_of_three = predict_multi_by_name(models_one,x_val3,y_val3)  
  
two_pred_of_two, two_actualls_of_two = predict_multi_by_name(models_two,x_val2,y_val2)  
two_pred_of_one, two_actualls_of_one = predict_multi_by_name(models_two,x_val1,y_val1)  
two_pred_of_three, two_actualls_of_three = predict_multi_by_name(models_two,x_val3,y_val3)  
  
three_pred_of_three, three_actualls_of_three = predict_multi_by_name(models_three,x_val3,y_val3)  
three_pred_of_one, three_actualls_of_one = predict_multi_by_name(models_three,x_val1,y_val1)  
three_pred_of_two, three_actualls_of_two = predict_multi_by_name(models_three,x_val2,y_val2)
```

```
2024-04-15 22:46:33.673278: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99  
2024-04-15 22:46:33.673415: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99  
2024-04-15 22:46:33.673470: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99  
2024-04-15 22:46:33.673546: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99  
2024-04-15 22:46:33.673596: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:99  
2024-04-15 22:46:33.673640: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1928] Created
```

```

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1713217594.529131    80228 service.cc:145] XLA service 0x7f223c004550 initialized
I0000 00:00:1713217594.529189    80228 service.cc:153] StreamExecutor device (0): NVIDIA Gel
2024-04-15 22:46:34.535575: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268
2024-04-15 22:46:34.563498: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:465] L
I0000 00:00:1713217595.037178    80228 device_compiler.h:188] Compiled cluster using XLA!  Th

```

118.7 Combined plots

```

#predictions on own kind
#means1,std1,means2,std2,means3,std3=cross_mean_err_calculator(one_pred_of_one,one_actuals

fig, (ax1,ax2) = plt.subplots(nrows=1,ncols=2,figsize=(11,6),dpi=600)

#NOTE: I've commented out the previous method of plotting due to its inelegance, but I have

#ONE ON ONE
#ax1.errorbar(np.sort(np.unique(orientation_actuals_of_orientation)),np.abs(means1-np.sort(n
#ax1.plot(np.sort(np.unique(one_actuals_of_one)),np.abs(means1-np.sort(np.unique(one_actua
#ax1.fill_between(np.sort(np.unique(one_actuals_of_one)),np.abs(means1-np.sort(np.unique(o

#TWO ON TWO
#ax1.errorbar(np.sort(np.unique(monochrome_actuals_of_monochrome)),np.abs(means2-np.sort(n
#ax1.plot(np.sort(np.unique(two_actuals_of_two)),np.abs(means2-np.sort(np.unique(two_actua
#ax1.fill_between(np.sort(np.unique(two_actuals_of_two)),np.abs(means2-np.sort(np.unique(t

#THREE ON THREE
#ax1.plot(np.sort(np.unique(three_actuals_of_three)),np.abs(means3-np.sort(np.unique(thre
#ax1.fill_between(np.sort(np.unique(three_actuals_of_three)),np.abs(means3-np.sort(np.unique(t

#ZERO LINE, LEGEND, AX1 PLOT CONFIG
#ax1.plot(np.sort(np.unique(one_actuals_of_one)),np.zeros(np.unique(one_actuals_of_one).sh
#ax1.legend(loc='upper left')
#ax1.set_xscale("log")
#ax1.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
#ax1.set_xticks(np.unique(y_val1))
#ax1.set_xlabel("Actual turning rate")
#ax1.set_ylabel("Absolute mean prediction difference")

df_one = pd.DataFrame()

```

```

df_two = pd.DataFrame()
df_three = pd.DataFrame()
df_one.insert(0,"predicted",np.abs(one_pred_of_one-one_actualls_of_one))
df_one.insert(1,"actualls",one_actualls_of_one)
df_two.insert(0,"predicted",np.abs(two_pred_of_two-two_actualls_of_two))
df_two.insert(1,"actualls",np.abs(two_actualls_of_two))
df_three.insert(0,"predicted",np.abs(three_pred_of_three-three_actualls_of_three))
df_three.insert(1,"actualls",np.abs(three_actualls_of_three))
df_one['Data Type']='Orientation'
df_two['Data Type']='Monochrome'
df_three['Data Type']='Scrambled'
cdf = pd.concat([df_one,df_two,df_three])
#print(cdf.head())

sns.lineplot(ax=ax1,
              x="actualls",
              y="predicted",
              hue="Data Type",
              data=cdf,
              errorbar="sd",
              palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

sns.boxplot(ax=ax2,
            data=cdf,
            x="actualls",
            y="predicted",
            hue="Data Type",
            fill=False,
            gap=.4,
            whis=(0,100),
            width=.5,
            palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

ax1.set_xlabel("Actual turning rate",fontsize=16)
ax2.set_xlabel("Actual turning rate",fontsize=16)
ax1.set_ylabel("Absolute mean prediction difference",fontsize=16)
ax2.set_ylabel("Absolute mean prediction difference",fontsize=16)
ax2.set_title("IQR Comparison of Self-Prediction",fontsize=16)
ax1.set_title("STD Comparison of Self-Prediction",fontsize=16)

ax1.set_xscale("log")

```

```

ax1.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
ax1.set_xticks(np.unique(y_val1))

#handles,labels=ax2.get_legend_handles_labels()
#ax2.legend(handles=handles[1:],labels=labels[1:]) #this should fix hue title appearing in
ax1.legend(loc="upper left",fontsize=16)
ax2.legend(fontsize=16)

#predictions on other kind
#means1,std1,means2,std2,means3,std3=cross_mean_err_calculator(one_pred_of_two,one_actuals

#ONE ON TWO
#ax2.errorbar(np.sort(np.unique(orientation_actuals_of_monochrome)),np.abs(means1-np.sort(
#ax2.plot(np.sort(np.unique(one_actuals_of_two)),np.abs(means1-np.sort(np.unique(one_actual
#ax2.fill_between(np.sort(np.unique(one_actuals_of_two)),np.abs(means1-np.sort(np.unique(o

#TWO ON ONE
#ax2.errorbar(np.sort(np.unique(monochrome_actuals_of_orientation)),np.abs(means2-np.sort(
#ax2.plot(np.sort(np.unique(two_actuals_of_one)),np.abs(means2-np.sort(np.unique(two_actua
#ax2.fill_between(np.sort(np.unique(two_actuals_of_one)),np.abs(means2-np.sort(np.unique(t

#ZERO LINE, LEGEND, AX2 PLOT CONFIG
#ax2.plot(np.sort(np.unique(one_actuals_of_two)),np.zeros(np.unique(one_actuals_of_two).sh
#ax2.legend(loc='upper left')
#ax2.set_xscale("log")
#ax2.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
#ax2.set_xticks(np.unique(y_val1))
#ax2.set_xlabel("Actual turning rate")
#ax2.set_ylabel("Absolute mean prediction difference")

fig.tight_layout()

```

```

df12=pd.DataFrame()
df21=pd.DataFrame()
df13=pd.DataFrame()
df31=pd.DataFrame()
df23=pd.DataFrame()
df32=pd.DataFrame()
df12.insert(0,"predicted",np.abs(one_pred_of_two-one_actuals_of_two))
df12.insert(1,"actuals",one_actuals_of_two)

```

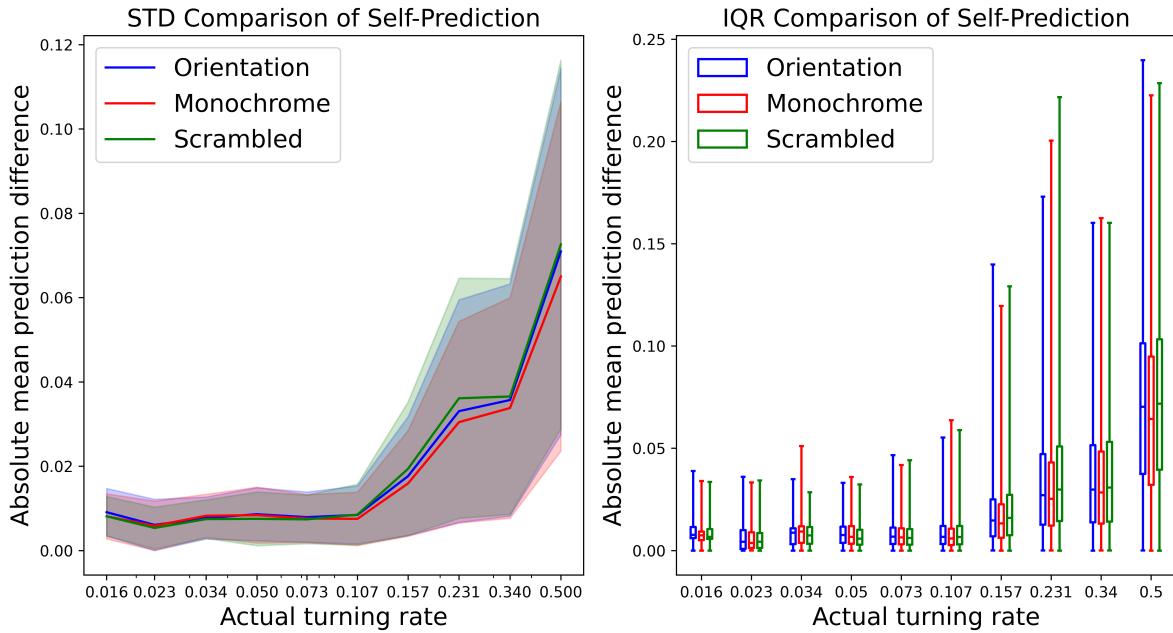


Figure 118.2: png

```

df21.insert(0,"predicted",np.abs(two_pred_of_one-two_actuals_of_one))
df21.insert(1,"actuals",two_actuals_of_one)
df13.insert(0,"predicted",np.abs(one_pred_of_three-one_actuals_of_three))
df13.insert(1,"actuals",one_actuals_of_three)
df31.insert(0,"predicted",np.abs(three_pred_of_one-three_actuals_of_one))
df31.insert(1,"actuals",three_actuals_of_one)
df23.insert(0,"predicted",np.abs(two_pred_of_three-two_actuals_of_three))
df23.insert(1,"actuals",two_actuals_of_three)
df32.insert(0,"predicted",np.abs(three_pred_of_two-three_actuals_of_two))
df32.insert(1,"actuals",three_actuals_of_two)

fig,ax = plt.subplots(nrows=3,ncols=2,figsize=(12,16),dpi=600)

#PREDICTING ONE

df21['Data Type']='Monochrome'
df31['Data Type']='Scrambled'
cdf = pd.concat([df_one,df21,df31])

sns.lineplot(ax=ax[0][0],

```

```

x="actuals",
y="predicted",
hue="Data Type",
data=cdf,
errorbar="sd",
palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

sns.boxplot(ax=ax[0][1],
            data=cdf,
            x="actuals",
            y="predicted",
            hue="Data Type",
            fill=False,
            gap=.4,
            whis=(0,100),
            width=.5,
            palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

#PREDICTING TWO

df12['Data Type']='Orientation'
df32['Data Type']='Scrambled'
cdf = pd.concat([df12,df_two,df32])

sns.lineplot(ax=ax[1][0],
             x="actuals",
             y="predicted",
             hue="Data Type",
             data=cdf,
             errorbar="sd",
             palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

ax[0][0].legend(loc='upper left', fontsize=16)
ax[0][1].legend(loc='upper right', fontsize=16)

sns.boxplot(ax=ax[1][1],
            data=cdf,
            x="actuals",
            y="predicted",

```

```

        hue="Data Type",
        fill=False,
        gap=.4,
        whis=(0,100),
        width=.5,
        palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

ax[1][0].legend(loc='upper left', fontsize=16)
ax[1][1].legend(loc='upper left', fontsize=16)

#PREDICTING THREE

df13['Data Type']='Orientation'
df23['Data Type']='Monochrome'
cdf = pd.concat([df13, df23, df_three])

sns.lineplot(ax=ax[2][0],
             x="actuals",
             y="predicted",
             hue="Data Type",
             data=cdf,
             errorbar="sd",
             palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

sns.boxplot(ax=ax[2][1],
            data=cdf,
            x="actuals",
            y="predicted",
            hue="Data Type",
            fill=False,
            gap=.4,
            whis=(0,100),
            width=.5,
            palette={"Orientation": "blue", "Monochrome": "red", "Scrambled": "green"})

ax[2][1].legend(fontsize=16)
ax[2][0].legend(loc='upper left', fontsize=16)

for i, examiner in enumerate(["STD", "IQR"]):
    for j, examined in enumerate(["Orientation", "Monochrome", "Scrambled"]):
        ax[j][i].set_xlabel("Actual turning rate", fontsize=16)

```

```
ax[j][i].set_ylabel("Absolute mean prediction difference", fontsize=16)
ax[j][i].set_title(f"{examiner} Comparison Predicting on {examined}", fontsize=16)
if i == 0:
    ax[j][i].set_xscale("log")
    ax[j][i].get_xaxis().set_major_formatter(ticker.ScalarFormatter())
    ax[j][i].set_xticks(np.unique(y_val1))

fig.tight_layout()
```

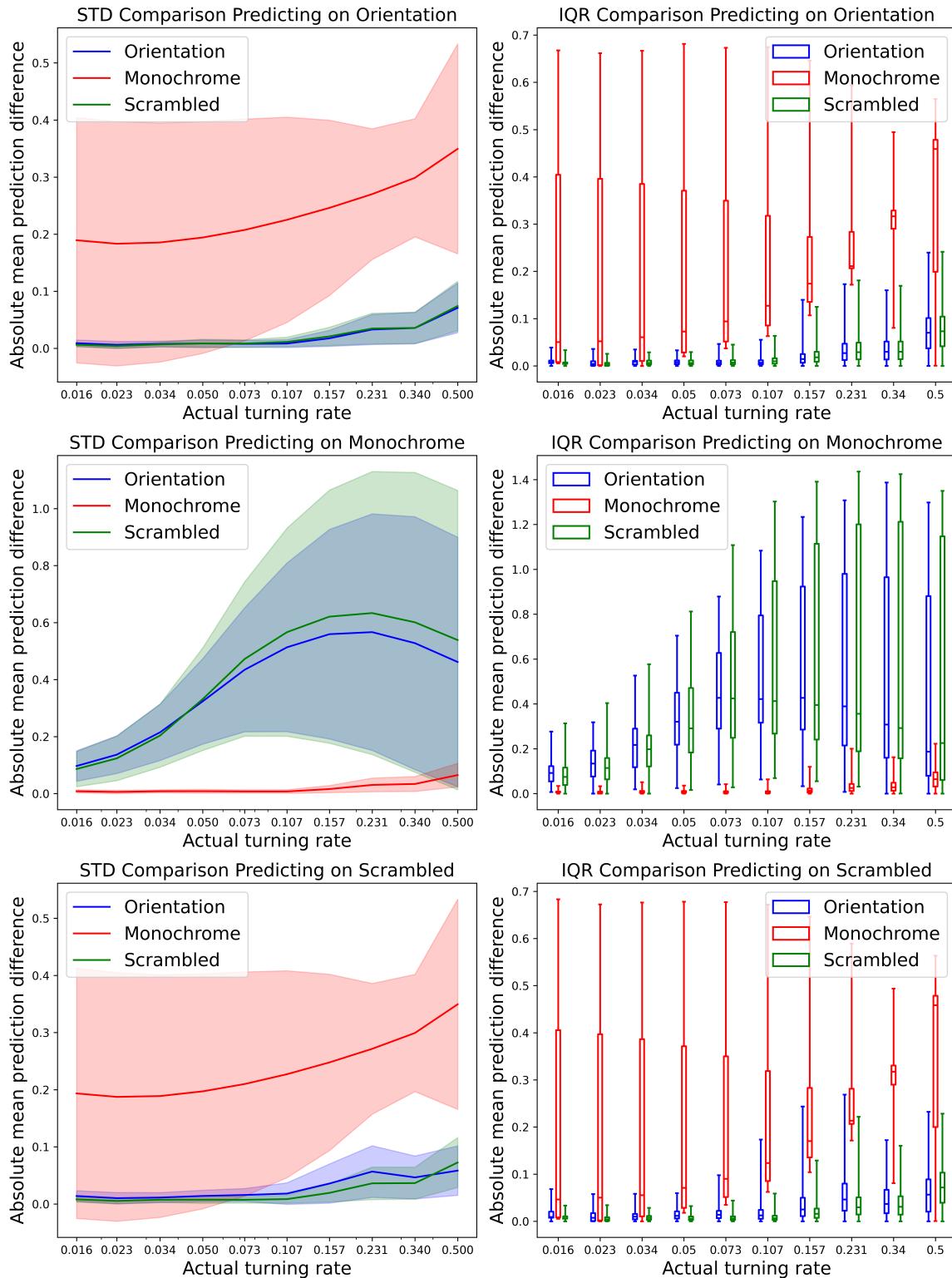


Figure 118.3: png

119 9. Interpolation

The code for interpolation is widely the same as the one in Section 8. As such, it is not pasted here (but can be found in `notebooks/cp_interpolation.ipynb`). Below are the results with arbitrarily assigned tumbling rates *between* the 10 values we have been using throughout this project.

119.1 Interpolation Predictions on Same Data Types

[png](week-20-22-files/more_interp_std_iqr_comparison_of_self_better.png)

119.2 Interpolation Predictions on Other Data Types

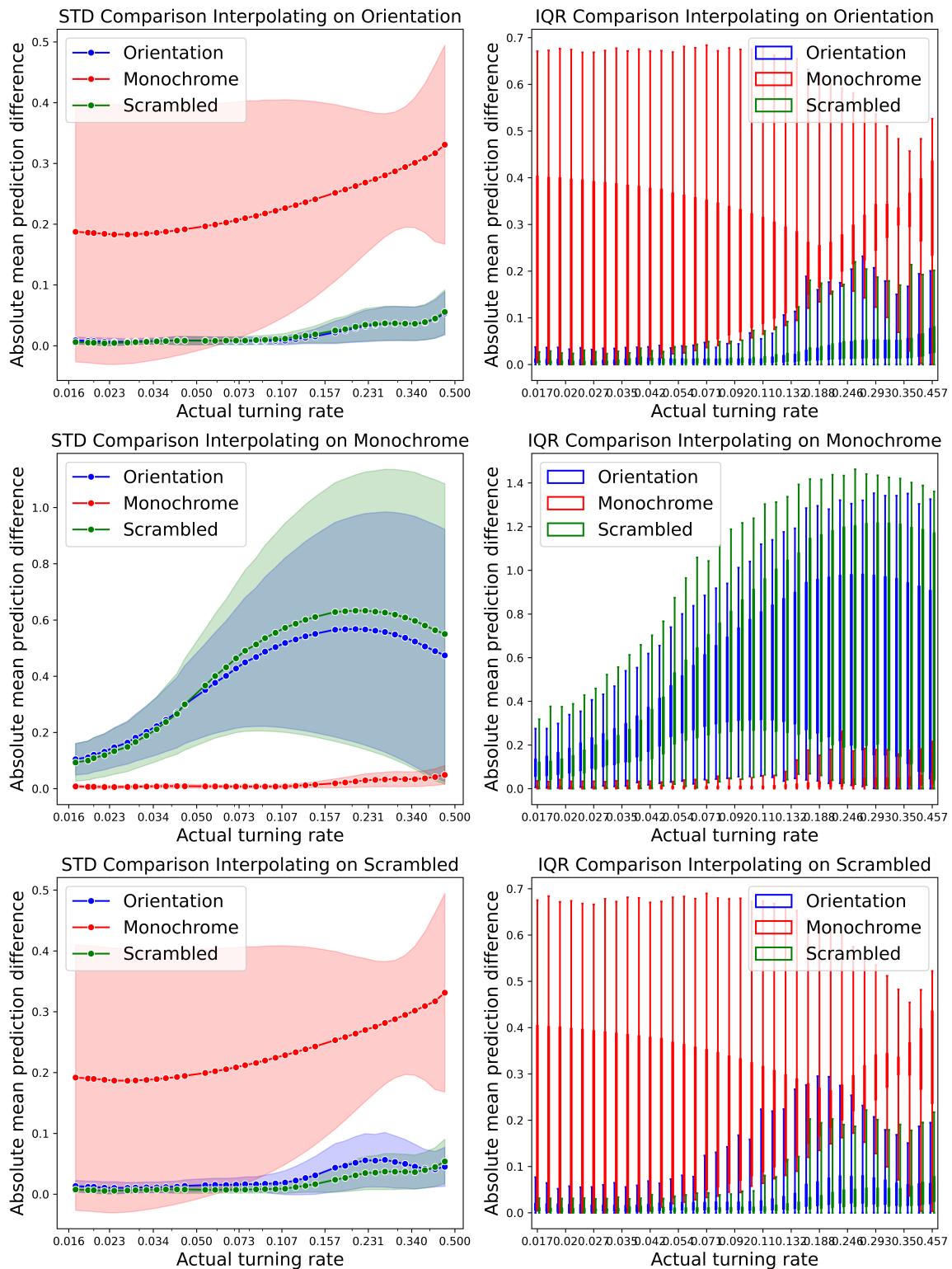


Figure 119.1: png

120 10. Extrapolation

Likewise, the code for extrapolation is widely the same as the one in Section 8. The results are plotted below, for both high-bounds extrapolation case (above the 10 training tumbling rates) and low-bounds extrapolation case (below the 10 training tumbling rates).

120.1 Low-bound Extrapolation

120.1.1 Predictions on Same Data Types

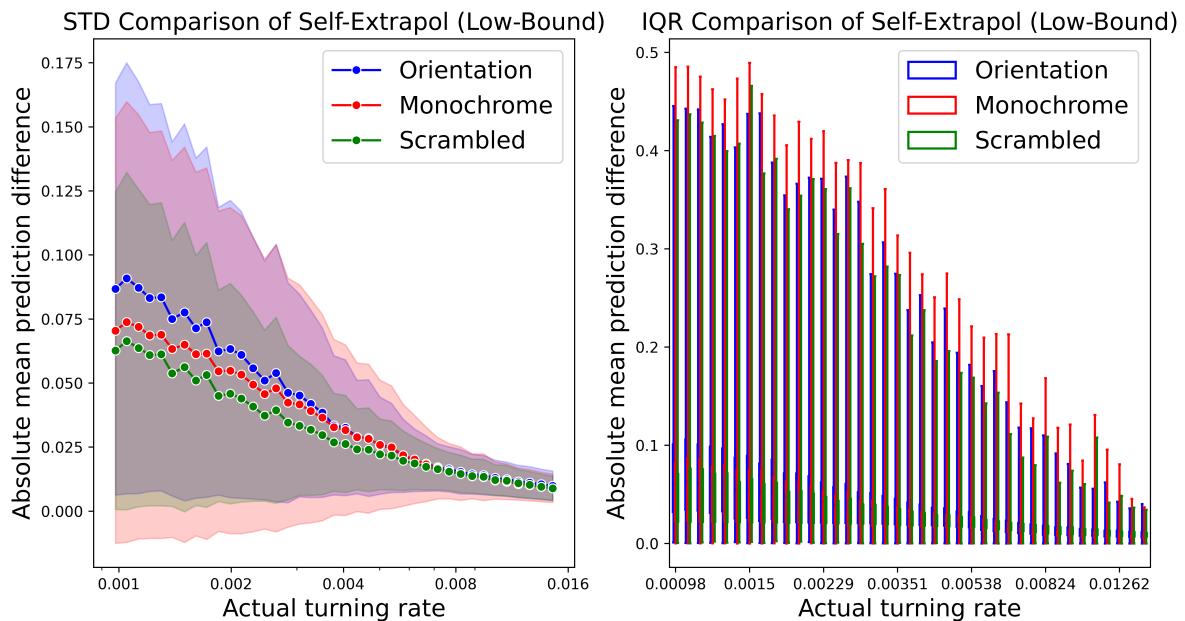


Figure 120.1: png

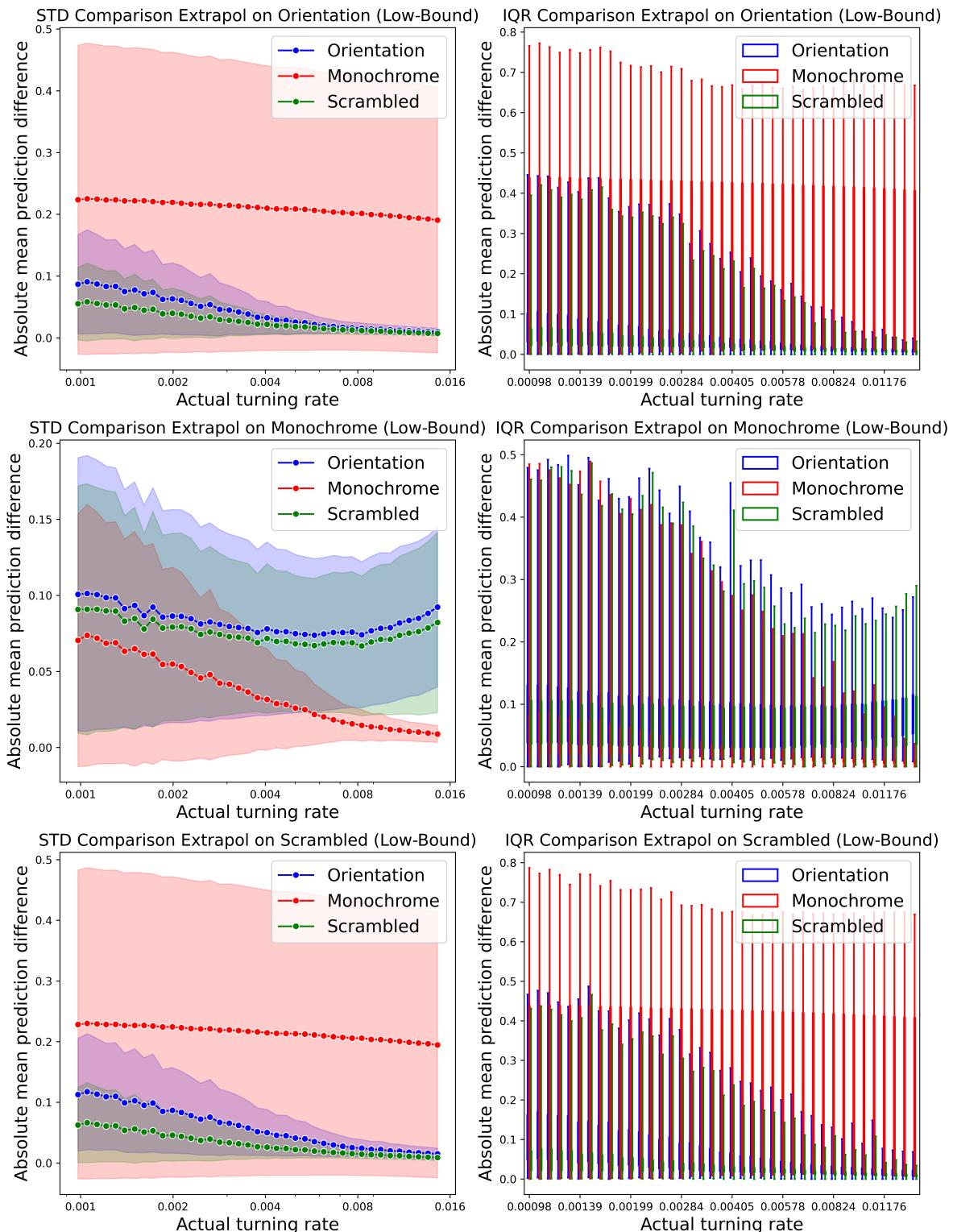


Figure 120.2: png

120.1.2 Predictions on Different Data Types

120.2 High-bound Extrapolation

120.2.1 Predictions on Same Data Types

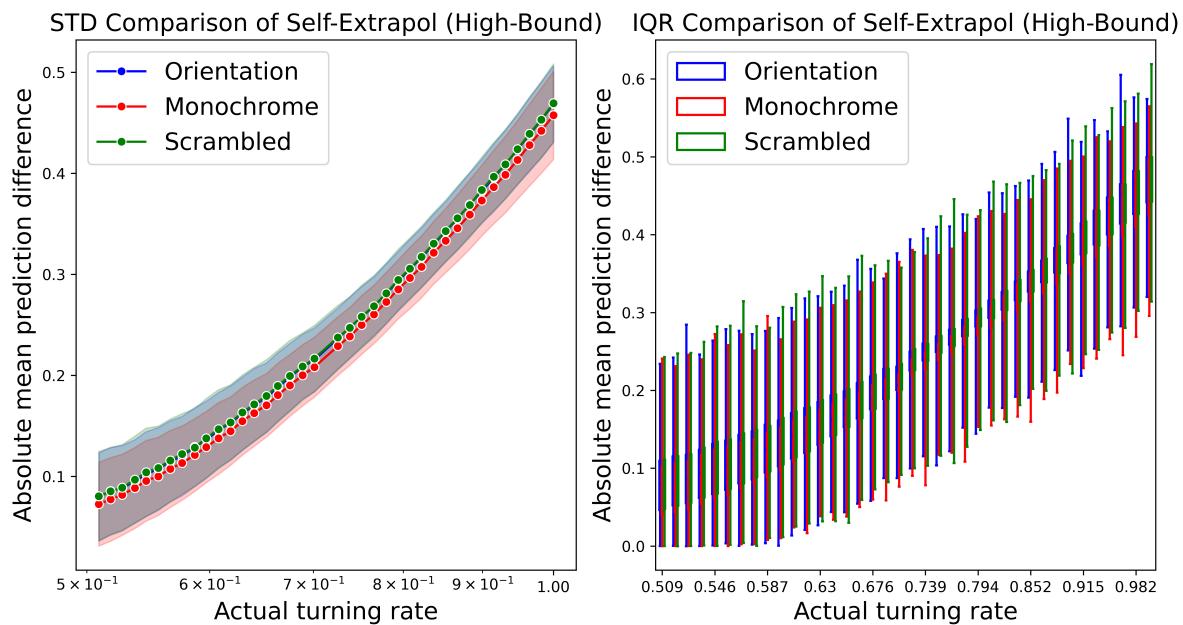


Figure 120.3: png

120.2.2 Predictions on Different Data Types

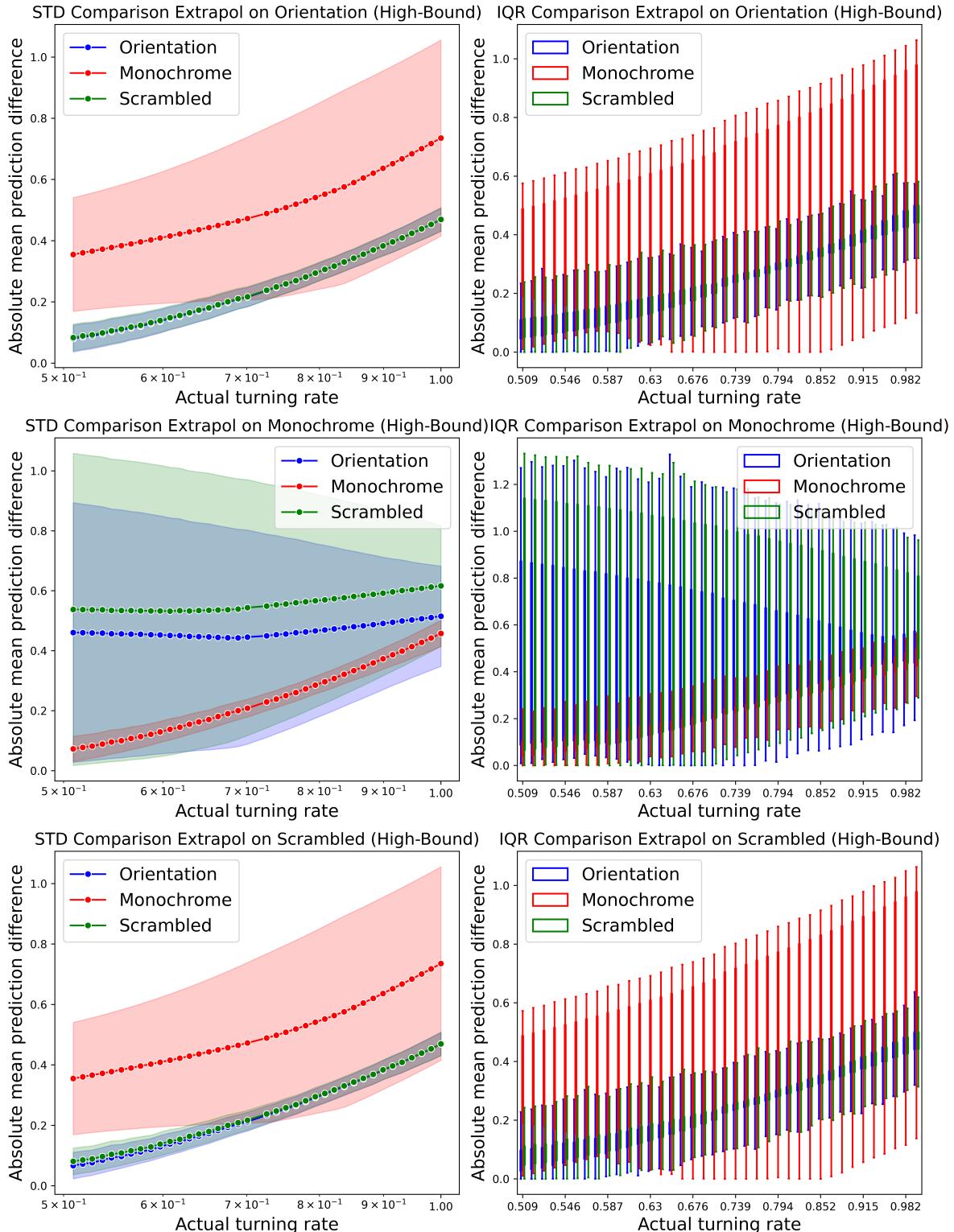


Figure 120.4: png

121 Motivational Report

122 0. Table of Contents

- 0. Table of Contents
- 1. Introduction
- 2. Original Motivational Report
- 3. Supervisor Feedback
- 4. Responding to Supervisor Feedback

123 1. Introduction

This is the landing spot for work on the Motivational Report. It was initially prompted in Week 2 as a way to get familiarised with the literature; it is expected to be a considerable part of the Interim Report, whose [deadline](#) is in Week 9.

124 2. Original Motivational Report (Week 2)

124.0.0.0.1 Pasted from “Motivational Report” chapter in Week 2

Active matter is, broadly, a subcategory of matter systems distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a force-free medium (for instance, the forces between particles and the fluid they move through cancel)[1]. The agents therefore have direct access to energy, and use it to autonomously propel and direct themselves (with different models and situations allowing for various degrees of freedom). The study of active matter is generally grounded in (but not limited to) observed behaviour of biological agents, as they are the primary (though not only) examples of active matter in nature.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the natural world. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2]. This is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3] (note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics under duress might deal holistically, rather than individually, with other human agents[4]). Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by medicine[5].

Outside of biology, active matter research serves to emulate, or otherwise learn from, naturally-occurring behaviours in order to analyse a potentially more general thermodynamic state. Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibria can be modified and generalised to non-equilibrium states, and how these generalisations hold as departure from equilibrium through various means is increased[6]. These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn,

would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[7].

The feature in active matter of converting stored and homogenously available energy, such as chemical potential, into mechanical work is also of great importance to the field: understanding how this can work and how to facilitate, among other things, long-term energy access across the active matter substance is a key pursuit of nanotechnology[8]. Statistical and computational models can lend insight into individual and collective dynamics, and in turn give way to new experimental designs of nano/micromechanical systems.

499 words.

124.0.0.1 References

1. Active matter: quantifying the departure from equilibrium. Flenner & Szamel
2. From Disorder to Order in Marching Locusts. Buhl et al. (2006)
3. Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts. Silverberg et al. (2013)
4. How simple rules determine pedestrian behavior and crowd disasters. Moussaid, Helbing & Theraulaz (2011)
5. Active matter at the interface between materials science and cell biology. Needleman & Zvonimir (2017)
6. Phase Separation and Multibody Effects in Three-Dimensional Active Brownian Particles. Turci & Wilding (2021)
7. Nano/Micromotors in Active Matte. Lv, Yank & Li (2022)
8. Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton et al. (2004)

125 3. Supervisor Feedback (Week 3)

125.0.0.0.0.1 Comments prefaced with FT and hyperlinked, commented initially on week2.md in main repository (pure markdown version of Week2)

Active matter is, broadly, a subcategory of matter systems FT “matter systems is unclear distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a force-free medium (for instance, the forces between particles and the fluid they move through cancel FT not exactly. Theres i no mechanical equilibrium. On the contrary, there is dissipation)[1]. The agents therefore have *direct* access to energy, and use it to autonomously propel and direct themselves (with different models and situations allowing for various degrees of freedom). The study of active matter is generally grounded in (but not limited to) observed behaviour of biological agents, as they are the primary (though not only) examples of active matter in nature. FT very good

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the natural world FT be more precise: it is the world of living organisms, which constantly dissipate energy to perform their biological functions. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2]. This is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. FT: You are onto something here. Physicist Andrea Cavagna likes to say that “*Physics gauges the surprise in biology*” Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3] (note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics under duress might deal holistically, rather than individually, with other human agents[4] FT not clear to me, please explain). Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by the medical sciences[5].

Outside of biology, active matter research serves to emulate, or otherwise learn from naturally-occurring behaviours in order to analyse a potentially more general thermodynamic state FT “state” is not a good word. Are you thinking about a more general thermodynamic

framework? . Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibrium can be modified and generalised to non-equilibrium states, and how these generalisations hold as departure from equilibrium through various means is increased[6] FT: not easy to read, but the idea is important: we can be just slight off equilibrium, and have a so-called linear-response regime, or we could be beyond linear response . These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[7].

The feature in active matter of converting stored and homogenously available energy, such as chemical potential, into mechanical work is also of great importance to the field: understanding how this can work and how to facilitate, among other things, long-term energy access across the active matter substance is a key pursuit of nanotechnology[8]. Statistical and computational models can lend insight into individual and collective dynamics, and in turn give way to new experimental designs of nano/micromechanical systems.

FT: You could get into more specifics, illustrating some examples of interesting behaviorus such as pattern formation or phase separation

502 words.

125.0.0.1 References

1. Active matter: quantifying the departure from equilibrium. Flenner & Szamel
2. From Disorder to Order in Marching Locusts. Buhl et al. (2006)
3. Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts. Silverberg et al. (2013)
4. How simple rules determine pedestrian behavior and crowd disasters. Moussaid, Helbing & Theraulaz (2011)
5. Active matter at the interface between materials science and cell biology. Needleman & Zvonimir (2017)
6. Phase Separation and Multibody Effects in Three-Dimensional Active Brownian Particles. Turci & Wilding (2021)
7. Nano/Micromotors in Active Matte. Lv, Yank & Li (2022)
8. Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton et al. (2004)

126 4. Responding to Supervisor Feedback

Week 4

My original context is presented, with the supervisor comments hyperlinked and prefaced by “FT”. Underneath I add my own comments in the form of bullet points. Only the commented parts of the report are shown.

Active matter is, broadly, a subcategory of matter systems FT “matter systems is unclear distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a force-free medium (for instance, the forces between particles and the fluid they move through cancel FT not exactly. Theres i no mechanical equilibrium. On the contrary, there is dissipation

- Here I was looking for a broad category to place active matter into; matter systems is indeed too vague. I would have been better off calling it a subcategory of soft matter systems.
- I don’t know exactly where I got the mechanical equilibrium confusion. I may have read some very specific thing that I generalised, but yes, dissipation ought to happen - one of the most important aspects of active matter is the requirement of supplying each autonomous agent with a steady energy supply which they steadily (or perhaps not so steadily in more complex models) use up.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the natural world FT be more precise: it is the world of living organisms, which constantly dissipate energy to perform their biological functions. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2]. This is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. FT: You are onto something here. Physicist Andrea Cavagna likes to say that “*Physics gauges the surprise in biology*” Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3] (note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics under duress might

deal holistically, rather than individually, with other human agents[4] FT not clear to me, please explain). Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by the medical sciences[5].

- I forgot that ‘natural world’ in English tends to refer more to general physical processes rather than specifically living organisms; I’ll try to be more specific regarding what active matter models help with understanding.
- From the brief look I managed to take at the literature, it seems that discussion of human behaviour in terms of physical systems is quite contentious. In hindsight, I should spend more than a sentence explaining this: the ‘cognitive heuristics’ argument for holism refers to the way humans deal with other humans in immediate crises. Many models will have an individual agent deal with other (in some way) adjacent agents individually; that is to say, it defines its relationship to each agent in turn, and then computes its behaviour. There are psychological arguments that this is not the case, and that instead humans might under duress conceptualise crowds (still) as a collective, and take actions in relation to the collective itself. At the time of writing this, it is unclear to me whether there are any active matter models that apply this ‘holistic’ method; the writers I cited, I believe, were criticising the models that do not attempt to do so. This is the case with the basic models I have engaged with so far (such as ABPs). It’s hard to imagine (though not impossible) how such a model can be implemented, but I don’t doubt that newer human-tracking physical models might work in this direction.
- Either way, I’ll look into Andrea Cavagna’s work. I’m interested in exploring this point more in detail.

Outside of biology, active matter research serves to emulate, or otherwise learn from naturally-occurring behaviours in order to analyse a potentially more general thermodynamic state FT “state” is not a good word. Are you thinking about a more general thermodynamic framework? . Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibrium can be modified and generalised to non-equilibrium states, and how these generalisations hold as departure from equilibrium through various means is increased[6] FT: not easy to read, but the idea is important: we can be just slight off equilibrium, and have a so-called linear-response regime, or we could be beyond linear response . These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[7].

- I take the point that ‘state’ is the wrong word; another loss in translation. I did mean a more general thermodynamic framework; thermodynamic ‘state’ implies thermal equilibrium, which is exactly what active matter does not have!
- I do get a bit long-winded here; I’ll try to rephrase this paragraph a bit and make sentences more readable

FT: You could get into more specifics, illustrating some examples of interesting behaviorus such as pattern formation or phase separation

- Yes, I’ll look into examples of pattern formation, as those tend to be quite demonstrative of what active matter study can do.

#5. Fleshing out Motivational Report Weeks 5-6

Active matter is, broadly, a subcategory of condensed matter systems distinguished primarily by energy input homogeneously distributed across all constituents (agents) of the system, which in turn set their own self-propelled motion across a medium. The agents therefore have *direct* access to energy, and use it to autonomously propel and direct themselves (with different models and situations allowing for various degrees of freedom). The study of active matter is generally grounded in (but not limited to) observed behaviour of biological agents, as they are the primary (though not only) examples of active matter in nature.

The evident motivation in studying active matter is that it helps understand biological behaviours, and therefore the world of living organisms, where energy is constantly dissipated in order to perform various biological functions. Macroscopically, the construction of theoretical models can help explain, and to a limited degree predict, the behaviour of animals (such as locusts) undergoing collectively-emergent swarming behaviours (where each animal can be treated as its own autonomous agent, sharing the same generally stable ‘rules’ of altering speed and orientation while interacting with each other and the environment)[2].

This biological emulation through physical models is not limited to what can be termed ‘simple’ behaviour; human behaviour can be partially mapped and understood within physically-indexed accounts of autonomous choices within (overtly or suggestively) constrained collective action. Interesting examples are swarming behaviours identified in traffic, crowd disasters and concerts[3]. Note however that physical models are sometimes challenged in literature due to potential oversimplifications, insofar as, for instance, cognitive heuristics (the autonomous individual behaviour of a human) under duress might deal holistically, rather than individually, with other human agents[4]. The issue is that most active matter systems only form individual relationships between agents, and do not account for the way an agent interacts with the group as a whole - the resulting individual behaviour is merely a summation of the agent’s response to each other agent around it. There are psychological arguments that this is not the case, and that instead humans might under duress conceptualise crowds as a collective, and take actions in relation to the collective itself. This objection rests on the assumption that this

holistic heuristic does not *emerge* from individual relations, of course (in which case mapping relationships strictly between individuals is unproblematic).

These insights lead to the exploration of various models. For flocks of birds, individual cognitive heuristics tend to suffice - self-propelled particles with adaptive movement patterns based on neighbours can accurately reproduce some migrational patterns [5]. Microscopically, active matter models offer insight into understanding how hierarchically-organised emergence happens within cell tissues, and how it may be leveraged by medicine[6]. Bacteria lends a great example for exploring the intertwining of phenomena to be emulated by active matter. Some strains (such as *Bacillus subtilis*) can be modelled using both direct physical interaction (between individuals) and long-distance biochemical signalling (within the collective), with complexity and clustering developing in response to harsh external conditions [7]. The latter interaction is called quorum sensing, the adaptation of the individual to local population density; this has developed into its own active matter branch of individual-to-collective behaviour [8]. Using such models, it is possible to recover the aforementioned human holistic cognitive heuristics [9].

Outside of biology, active matter research serves to emulate, or otherwise learn from, naturally-occurring behaviours in order to analyse a potentially more general thermodynamic framework. Due to the necessarily dissipative use of energy within self-organised agents, and their internally-induced behaviour, active matter is not described by the statistical mechanics of equilibrium states. The question then arises whether, through quantitative computation and qualitative modelling/theorising, the thermodynamic laws of equilibria can be modified and generalised to non-equilibrium states. Exploring how these generalisations would hold as departure from equilibrium through various means is increased is then paramount[10]. These generalisations would, ideally, collapse into the known statistical thermodynamics states within the equilibrium limit. These insights, in turn, would facilitate the creation of synthetic active matter, whose potential, although speculative, ranges from the biomedical application of nanomachine targeted drug delivery possibilities to the location-mapping application of nanoscopic/microscopic environmental sensing[11].

The feature in active matter of converting stored and homogenously available energy, such as chemical potential, into mechanical work is also of great importance to the field: understanding how this can work and how to facilitate, among other things, long-term energy access across the active matter substance is a key pursuit of nanotechnology[12]. Statistical and computational models can lend insight into individual and collective dynamics, and in turn give way to new experimental designs of nano/micromechanical systems.

756 words.

126.0.0.1 References

1. Active matter: quantifying the departure from equilibrium. Flenner & Szamel

2. From Disorder to Order in Marching Locusts. Buhl et al. (2006)
3. Collective Motion of Humans in Mosh and Circle Pits at Heavy Metal Concerts. Silverberg et al. (2013)
4. How simple rules determine pedestrian behavior and crowd disasters. Moussaid, Helbing & Theraulaz (2011)
5. Novel Type of Phase Transition in a System of Self-Driven Particles, Vicsek et al. (1995)
6. Active matter at the interface between materials science and cell biology. Needleman & Zvonimir (2017)
7. Formation of complex bacterial colonies via self-generated vortices, Czirok et al. (1996)
8. Self-organization of active particles by quorum sensing rules, Bäuerle et al. (2018)
9. Quorum sensing as a mechanism to harness the wisdom of the crowds, Moreno-Gámez et al. (2023)
10. Phase Separation and Multibody Effects in Three-Dimensional Active Brownian Particles. Turci & Wilding (2021)
11. Nano/Micromotors in Active Matte. Lv, Yank & Li (2022)
12. Catalytic Nanomotors: Autonomous Movement of Striped Nanorods, Paxton et al. (2004)

127 CNN Models

This page hosts all of the main model architectures utilised in our project, indexed by model number (MN). ### MN_0

127.0.1 MN_1

1. CONV (filters=3,kernel_size=(3,3),padding='same',strides=(3,3),activation="relu", input_shape=shape)
2. BN
3. CONV (filters=3,kernel_size=(3,3),padding='same')
4. BN
5. MAXPOOL (pool_size=(3,3))
6. CONV (filters=6,kernel_size=(3,3),padding='same')
7. BN
8. CONV (filters=6,kernel_size=(3,3),padding='same')
9. BN
10. MAXPOOL (pool_size=(3,3))
11. DENSE (units=128,activation='relu')
12. DO (0.2) (without layout optimiser)
13. DENSE (units=10,activation='softmax')
14. FLATTEN
15. DENSE (units=1,activation='linear')

127.0.2 MN_1.5

1. CONV (filters=3,kernel_size=(3,3),padding='same',strides=(3,3),activation='relu',input_shape=shape)
2. BN
3. CONV (filters=3,kernel_size=(3,3),padding='same')
4. BN
5. MAXPOOL (pool_size=(3,3))
6. CONV (filters=6,kernel_size=(3,3),padding='same')
7. BN
8. CONV (filters=6,kernel_size=(3,3),padding='same')
9. BN

10. MAXPOOL (pool_size=(3,3))
11. FC (units=128,activation='relu')
12. DO (0.2) (without layout optimiser)
13. FC (units=10,activation='softmax')
14. FLATTEN
15. FC (units=1,activation='linear')

127.0.3 MN_2

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=6,kernel_size=(5,5),padding='same')
6. MAXPOOL (pool_size=(2,2),padding='same')
7. ReLU
8. BN
9. AVGPOOL
10. DO (0.2) (without layout optimiser)
11. DENSE (units=64,activation='relu')
12. DO (0.2) (without layout optimiser)
13. DENSE (units=10,activation='softmax')
14. FLATTEN
15. DENSE (units=1,activation='linear')

127.0.4 MN_3

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)

127.0.5 MN_3*

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=(5,5),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')

128 Preliminary CNN Training and Analysis

This is a brief example of the methodology used throughout the CNN training and analysis part of this project.

129 Import packages

```
import numpy as np
import h5py
import glob
import re
import tensorflow as tf

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.stats import pearsonr

np.set_printoptions(precision=3, suppress=True)
```

```
2024-02-14 14:08:18.312814: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342]
2024-02-14 14:08:18.312841: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] W
2024-02-14 14:08:18.312870: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518]
2024-02-14 14:08:18.319367: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-02-14 14:08:19.589028: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W
```

130 Setup GPU

First, follow instructions [here](#), or alternatively run:

```
for a in /sys/bus/pci/devices/*; do echo 0 | sudo tee -a $a/numa_node; done
```

We do this as a workaround for [this error](#):

```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)
print(tf.config.list_physical_devices('GPU'), tf.test.gpu_device_name())
```

[]

```
2024-02-14 14:08:25.970865: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:2211] Cannot determine if GPU 0 is supported
2024-02-14 14:08:26.349400: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot determine if GPU 0 is supported
Skipping registering GPU devices...
2024-02-14 14:08:26.353220: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:2211] Cannot determine if GPU 0 is supported
2024-02-14 14:08:26.353403: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot determine if GPU 0 is supported
Skipping registering GPU devices...
```

131 Import and prepare data

```
def extract_floats(string):
    return re.findall(r"[-+]?(\d*\.\d+|\d+)", string)

def data_load():
    density = 0.15
    files = glob.glob(f"../data/dataset_tumble_*_0.15.h5") #imports all tumbling rates for
    inputs,outputs = [],[]
    for f in files:
        tumble = float(extract_floats(f)[0])
        with h5py.File(f, "r") as fin:
            count = 0
            for key in fin.keys():
                img = fin[key] [:]
                img = img.reshape((img.shape[0], img.shape[1],1))
                shape = img.shape
                inputs.append(img)
                outputs.append(tumble)
                count+=1

    # Scramble the dataset
    order = np.arange(len(outputs)).astype(int)
    order = np.random.permutation(order)
    return np.array(inputs)[order],np.array(outputs)[order],shape

x,y,shape = data_load()

print("Number of unique alpha: ", len(np.unique(y)))
print("Shape of x: ", np.shape(x))
print("Shape of y: ", np.shape(y))
```

Number of unique alpha: 5
Shape of x: (70000, 128, 128, 1)

```
Shape of y: (70000,)
```

We have $10000 * \text{number of unique alpha snapshots}$ total, we split them into a training set and a validation set:

```
last = 20000
x_train, y_train = x[:-last], y[:-last]
x_val,y_val = x[-last:],y[-last:]

print("Size of training data: ", len(x_train))
print("Size of validation data: ", len(x_val))
```

```
Size of training data: 50000
Size of validation data: 20000
```

132 Setup and train our model

```
from tensorflow import keras
from keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,Dropout,MaxPooling2D,BatchNormali
import contextlib

@contextlib.contextmanager
def options(options):
    old_opts = tf.config.optimizer.get_experimental_options()
    tf.config.optimizer.set_experimental_options(options)
    try:
        yield
    finally:
        tf.config.optimizer.set_experimental_options(old_opts)
```

Run this after analysis to reset model and release RAM before changing the architecture

```
import gc

K.clear_session()
del prediction
del model
del history

print("Collected: ", gc.collect())
```

132.1 Setting up the model's architecture

```
model = Sequential()

model.add(Conv2D(filters=3, kernel_size=(3,3), padding='same', strides=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=3, kernel_size=(3,3), padding='same', input_shape=shape))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(3, 3)))

#model.add(AveragePooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters=6, kernel_size=(3,3), padding='same'))
model.add(BatchNormalization())

model.add(Conv2D(filters=6, kernel_size=(3,3), padding='same'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Dense(units=128, activation='relu'))

with options({"layout_optimizer": False}):
    model.add(Dropout(0.2))
model.add(Dense(units=10, activation='softmax'))

model.add(Flatten())
model.add(Dense(units=1, activation='linear'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 43, 43, 3)	30
batch_normalization (Batch Normalization)	(None, 43, 43, 3)	12

conv2d_1 (Conv2D)	(None, 43, 43, 3)	84
batch_normalization_1 (BatchNormalization)	(None, 43, 43, 3)	12
max_pooling2d (MaxPooling2D)	(None, 14, 14, 3)	0
conv2d_2 (Conv2D)	(None, 14, 14, 6)	168
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 6)	24
conv2d_3 (Conv2D)	(None, 14, 14, 6)	330
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 6)	24
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 6)	0
dense (Dense)	(None, 4, 4, 128)	896
dropout (Dropout)	(None, 4, 4, 128)	0
dense_1 (Dense)	(None, 4, 4, 10)	1290
flatten (Flatten)	(None, 160)	0
dense_2 (Dense)	(None, 1)	161

Total params: 3031 (11.84 KB)
 Trainable params: 2995 (11.70 KB)
 Non-trainable params: 36 (144.00 Byte)

132.2 Optimizer

```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='mean_absolute_error', optimizer=optimizer, metrics=['accuracy'])
```

132.3 Training and evaluation

Before training, these are the “predictions”:

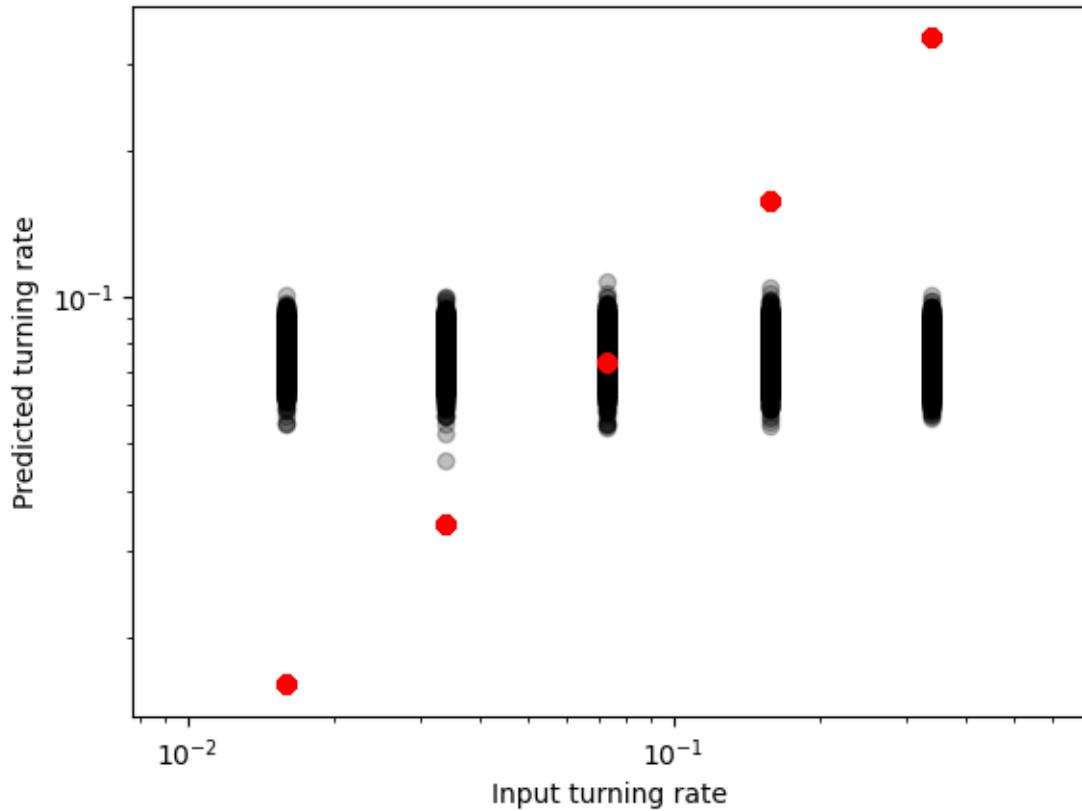
```
prediction = model.predict(x_val, batch_size=64)
print("Shape of prediction : ", np.shape(prediction))

plt.plot(y_val, prediction.T[0], 'o', c='k', alpha=0.25)
plt.plot(y_val, y_val, 'o', color='r')

print("Pearson's correlation coeff: ", pearsonr(y_val, prediction.T[0]).statistic)
plt.xlabel("Input turning rate")
plt.ylabel("Predicted turning rate")
plt.axis("equal")
plt.xscale("log")
plt.yscale("log")
```

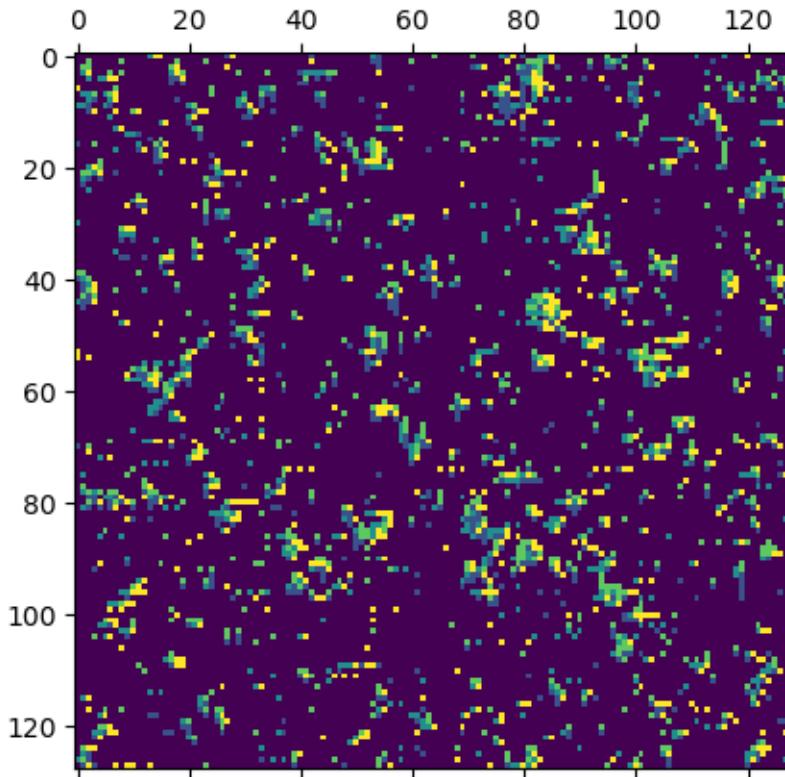
2024-02-14 14:12:29.587687: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation

```
313/313 [=====] - 8s 25ms/step
Shape of prediction : (20000, 1)
Pearson's correlation coeff: -0.04497726280558032
```



```
demo_idx = 100
plt.matshow(x_val[demo_idx])
print("Actual: ", y_val[demo_idx])
print("Predicted: ", prediction.T[0][demo_idx])
```

Actual: 0.034
Predicted: 0.07371252



We can play with the architecture and see how the untrained predictions can change too.

132.4 Run the training

```
history = model.fit(  
    x_train,  
    y_train,  
    epochs=10,  
    verbose=True,  
    batch_size=64,  
    validation_data=(x_val, y_val)  
)
```

Epoch 1/10

2024-02-14 14:13:58.918844: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o

```
781/782 [=====>.] - ETA: 0s - loss: 0.0456 - accuracy: 0.0000e+00

2024-02-14 14:14:55.119286: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o

782/782 [=====] - 54s 66ms/step - loss: 0.0456 - accuracy: 0.0000e+00
Epoch 2/10
782/782 [=====] - 50s 64ms/step - loss: 0.0269 - accuracy: 0.0000e+00
Epoch 3/10
782/782 [=====] - 50s 64ms/step - loss: 0.0260 - accuracy: 0.0000e+00
Epoch 4/10
782/782 [=====] - 37s 48ms/step - loss: 0.0236 - accuracy: 0.0000e+00
Epoch 5/10
782/782 [=====] - 43s 55ms/step - loss: 0.0221 - accuracy: 0.0000e+00
Epoch 6/10
782/782 [=====] - 43s 55ms/step - loss: 0.0212 - accuracy: 0.0000e+00
Epoch 7/10
782/782 [=====] - 43s 55ms/step - loss: 0.0208 - accuracy: 0.0000e+00
Epoch 8/10
782/782 [=====] - 43s 55ms/step - loss: 0.0204 - accuracy: 0.0000e+00
Epoch 9/10
782/782 [=====] - 49s 63ms/step - loss: 0.0199 - accuracy: 0.0000e+00
Epoch 10/10
782/782 [=====] - 69s 88ms/step - loss: 0.0196 - accuracy: 0.0000e+00
```

```
    print("Evaluate on test data:")
    results = model.evaluate(x_val, y_val, batch_size=64, verbose=0)
    print("Test loss:", results[0])
    print("Test accuracy:", results[1])
```

```
Evaluate on test data:
```

```
2024-02-14 14:22:51.483660: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o

Test loss: 0.01768036000430584
Test accuracy: 0.0
```

133 Analyse training results

```
prediction = model.predict(x_val, batch_size=64)
print("Shape of prediction : ", np.shape(prediction))
```

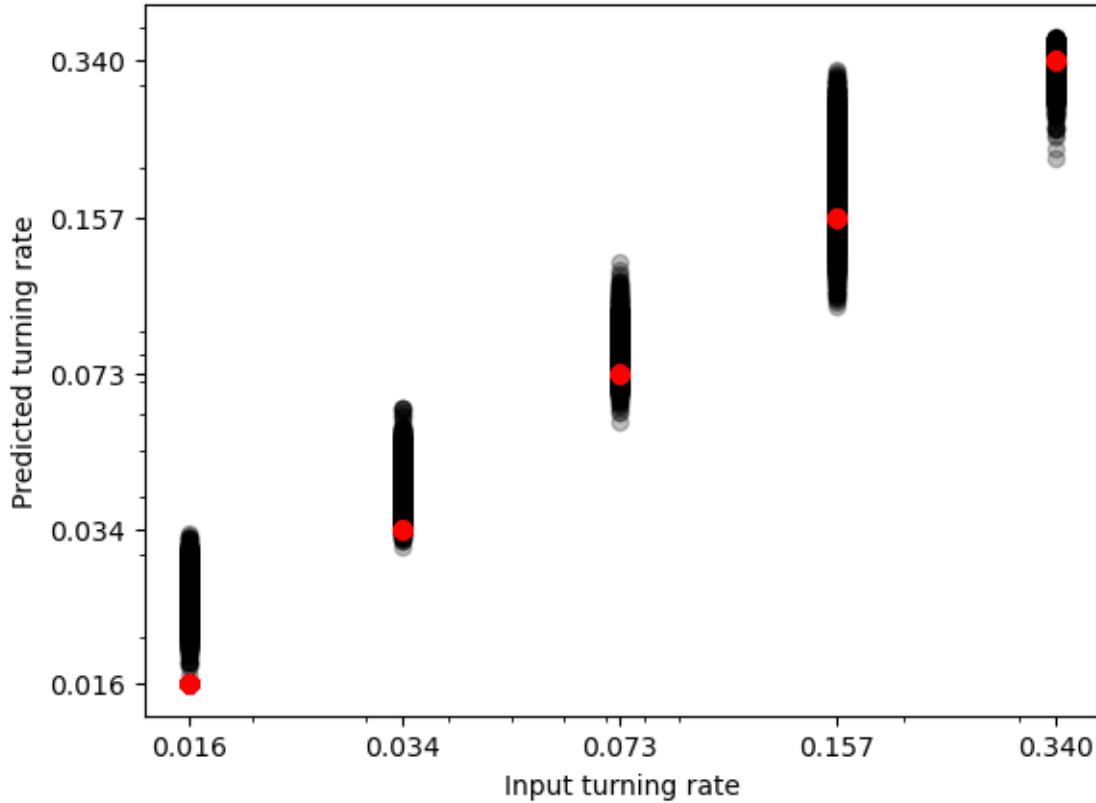
```
2024-02-14 14:23:08.376226: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation 0x55e000000000: 313/313 [=====] - 10s 32ms/step
Shape of prediction : (20000, 1)
```

```
fig, ax = plt.subplots()

ax.plot(y_val, prediction.T[0], 'o', c='k', alpha=0.25)
ax.plot(y_val, y_val, 'o', color='r')

print("Pearson's correlation coeff: ", pearsonr(y_val, prediction.T[0]).statistic)
ax.set_xlabel("Input turning rate")
ax.set_ylabel("Predicted turning rate")
#ax.set_aspect("equal")
ax.set_xscale("log")
ax.set_yscale("log")
ax.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
ax.get_yaxis().set_major_formatter(ticker.ScalarFormatter())
ax.set_xticks(np.unique(y))
ax.set_yticks(np.unique(y))
plt.show()
print(np.unique(y))
```

```
Pearson's correlation coeff: 0.9788006506188127
```



[0.016 0.034 0.073 0.157 0.34]

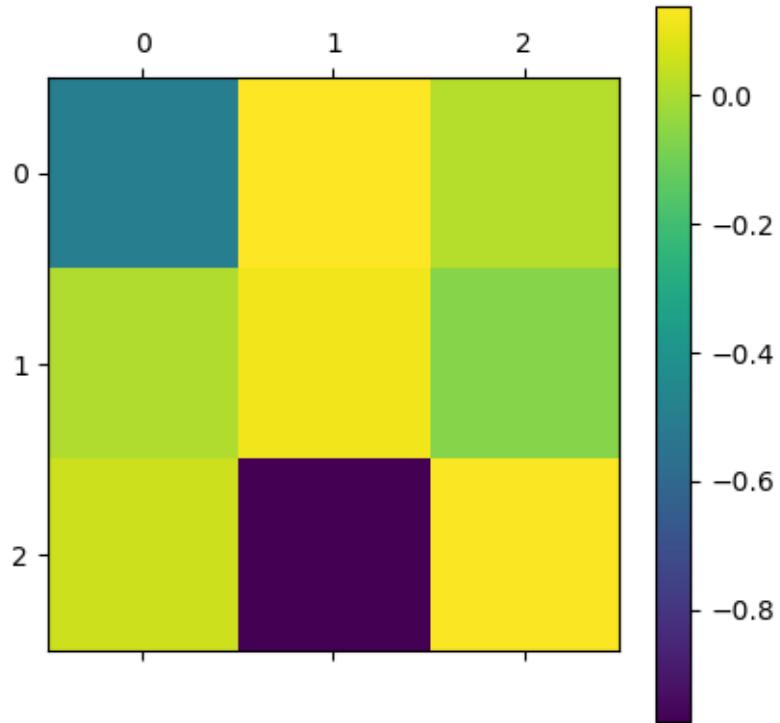
133.0.1 Kernel analysis

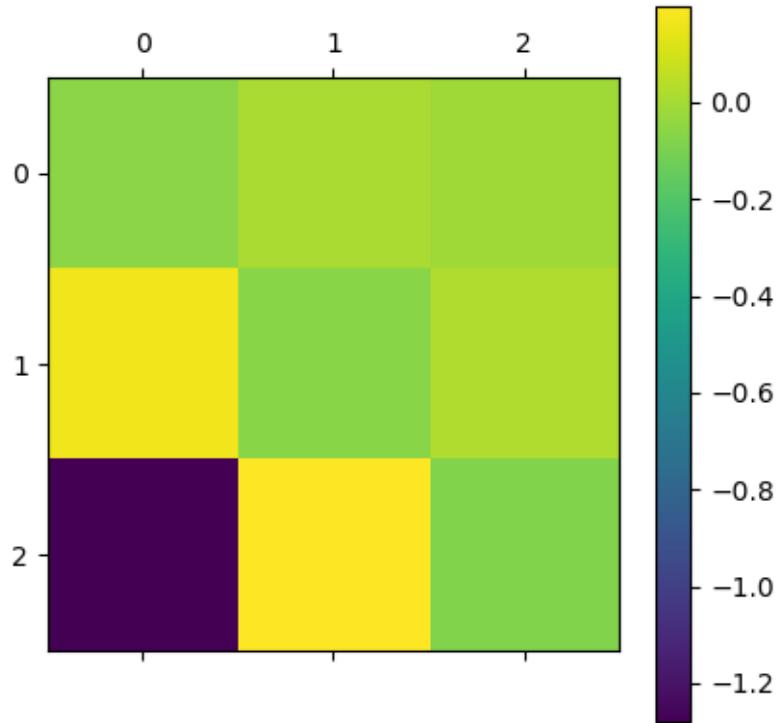
```

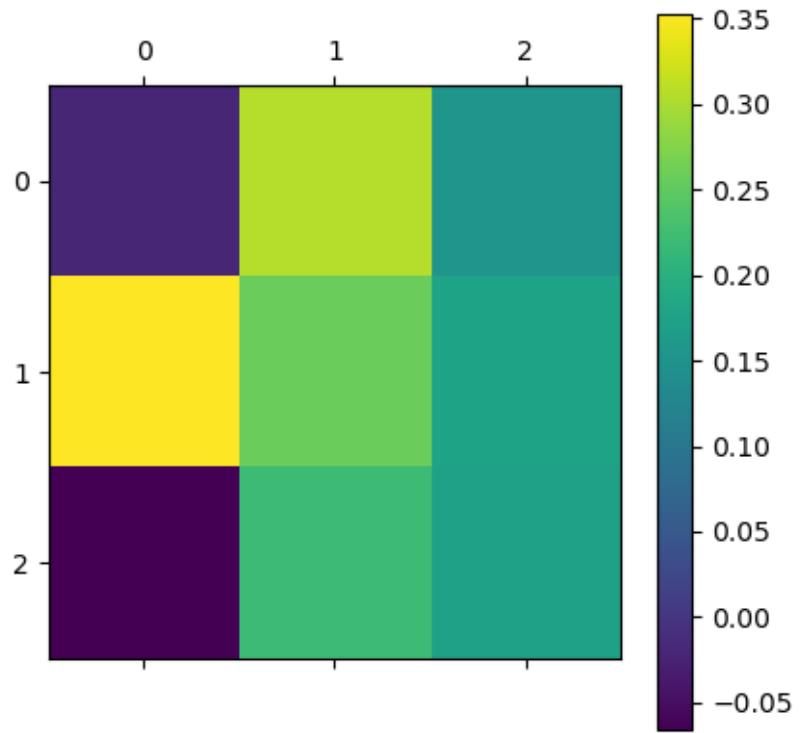
filters, biases = model.layers[0].get_weights()
print (filters.shape[-1])
for k in range(filters.shape[-1]):
    f = filters[:, :, :, k]
    plt.matshow(f.squeeze())
    plt.colorbar()

```

3







134 Save model (if needed)

```
model.save("../models/cp_14feb_prelim.keras")
```