Report On Summer Project

Joe Aplin
School of Physics, University of Bristol.
(Dated: August 4, 2024)

This summary will go over how to use the files to recreate some of the following results. We looked into the machine learning model designed by Chris Paunica and Nhat Pham, attempting to understand the ways the neural network interprets data given to it from the persistent exclusion process (PEP). We compared extrapolation of model sizes comparing accuracy, looked into the ways that data was augmented to give more training data and if it helps. We focused partly on interpreting the different layers of the model and their outputs to understand what the model was 'seeing' and compared the kernels generated to known filters for image processing to see if it was learning these methods by itself.

INITIALISATION

Within the src folder contains the file that is sued to create the simulation from the PEP design. In the file itself there is a line to select the number of iterations to output, defaulted to 1000, as well as a loop later on to generate the tumbling copies of data for outputting. This value can be changed and the loop commented out to change the type of data you want to generate. Additionally you can change the tumbling rates chosen. These are the only changes made to this file, otherwise the data can be generated in command line using "./sampler.py –density 0.1 –odd" where 0.1 is the density of pixels that are particles and –odd is removed to generate the evenly indexed tumbling rates.

Once you have data you can train a model using modelTrainer.ipynb. This will output a model in the models folder trained on the data. This is the basis of all models and so the rest is analysis of the model and the data.

An easy first thing to analyse is the learning curve of the models trained. To do this, uncomment the cell after Run Training to output the validation losses. To plot these, use learningCurves.ipynb and an example result may be as shown in Fig 1.

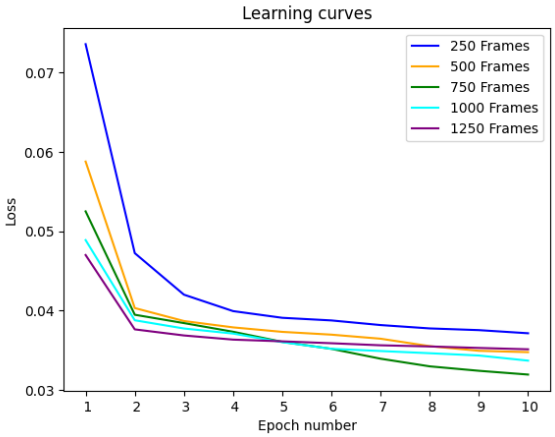


FIG. 1: This is the test learning curves of data of different sizes from 250 to 1250 output frames of the same tumbling rates and density. The larger the dataset, typically the slower the learning.

INITIAL ANALYSIS

With a model created we can started analysis of what the network is doing to understand the information it is given. Each layer in the network generates 3x3 kernels that are applied to the pixels in the input to perform some function in order to produce an output. Although this is self taught, we can compare them to known image filters used for image processing and see if the network is mirroring any of these filters. This is shown in kernelComparison.ipynb. The results of doing this were less obvious than it seemed as some filters would give large differences because they were close to an anti-mirror, but generally the shape of filters that the models learnt was similar to sobel or farid, so a cross as shown below in Fig 2.

Once you have this result, it is useful to see how these filters are used within the model to understand more of what it is doing. You can do this like in the modelBreakdown.ipynb file by creating a model from the input and a chosen layer output from another model, so it saves the weights and biases from an already trained model but can output an image from an intermediate layer. Fig 3 shows how an example input image is processed by the model. The first kernel seems to keep the grouping of clusters very similar, appearing to 'shrink' the image.

TRAINING DATA

Extrapolation

So far all of the analysis is available from just one dataset. We want to see how the model can change from different training sets and see if they can inform information about what the network sees. As shown by the previous two who worked on this model, percolation occurs more often at lower tumbling rates and plays an important role in the network's understanding of what the rumbling rate is. This can be exemplified as in Fig 4 where the only model that doesn't extrapolate well to

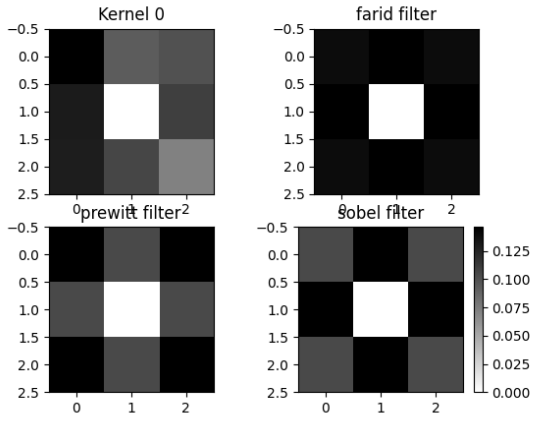
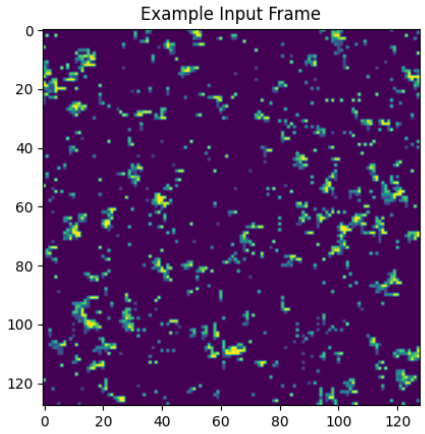


FIG. 2: This is an example kernel from the first layer of a model and the three known filters that are closest to the kernel. There is a clear lack of importance on the centre pixel with equal importance in each direction.

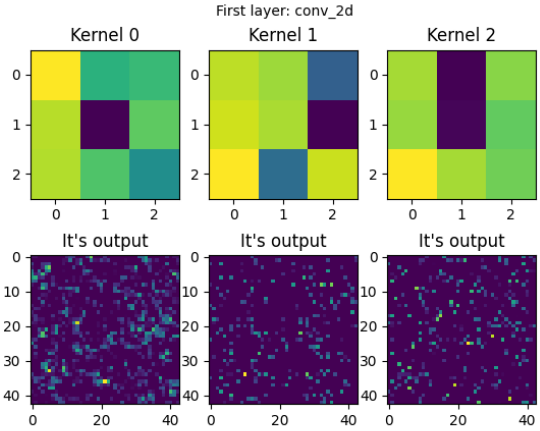
lower tumbling rates is the one that is given the most data to be trained on, suggesting it has overfitted to the larger tumbling rates and may have learnt another way to determine tumbling rates to other models. Also within this can be seen the performance of these differently sized datasets, where the expected result would be that for the model trained on the most data the accuracy would be assumed to be highest, but is not the case. This was explored by comparing the same inputs multiple times and seeing that the random nature of the simulation accounts for relatively large fluctuations in accuracy, but not all of the fluctuation.

Rolling

In the code for generating data is a loop to generate tumbling in the data. This means for every output frame, 13 more are generated that are the same image but slightly shifted. The reason for this was to teach the model to learn that the bounds of the frame are continuous so what happens on one bound is affected by the other side. Additionally this is a cheap way of generating more training data. This next section is the only part where rolling was not used, and that was to compare models trained on the rolling data compared to no rolling to see if the model was learning differently. Firstly, when it came to kernel analysis, it appeared the models trained on no rolling data learnt similar filters: sobel, farid and scahrr, all of which are either the same or have a similar shape of a cross to the rolling data model. A model trained on no rolling data was compared to a model trained on rolling data for their accuracies, and is shown in Fig 5. Here it's clear the rolling data training set helped the model to predict higher tumbling rates significantly but not for lower tumbling rates. This suggests the model could benefit from extra training data for higher



(a) Example frame of input tumbling rate 0.016, shows some light percolation.



(b) The output of the first layer is generated by 3 kernels, each shown here with their respective outputs.

FIG. 3: How the first 2d convolution layer changes the input image for input into the second layer.

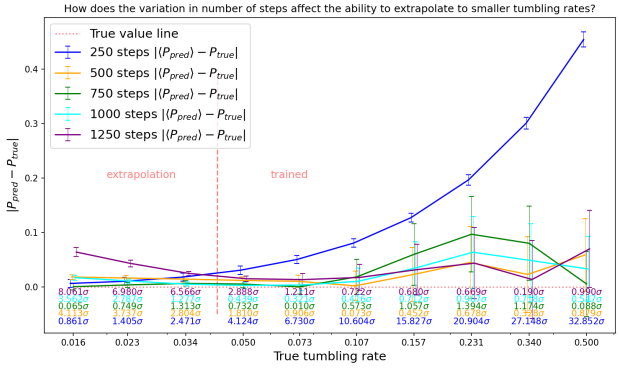


FIG. 4: The models shown here have all been trained on the upper 7 tumbling rates, but of varying data sizes. The model trained on the most model is the only one that doesn't extrapolate very well to lower tumbling rates.

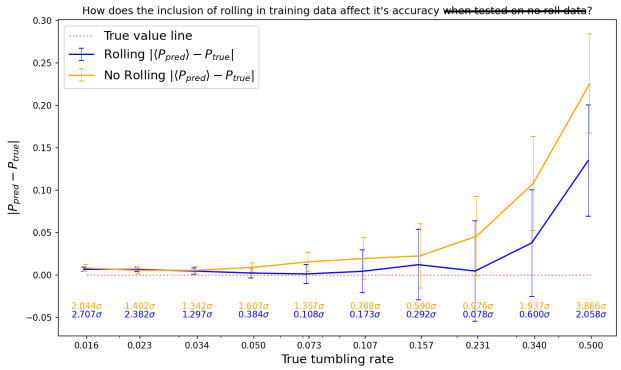


FIG. 5: A comparison of a model trained on rolling data to a model not trained on rolling data. The rolled data trained model has a higher accuracy mainly for higher tumbling rates.

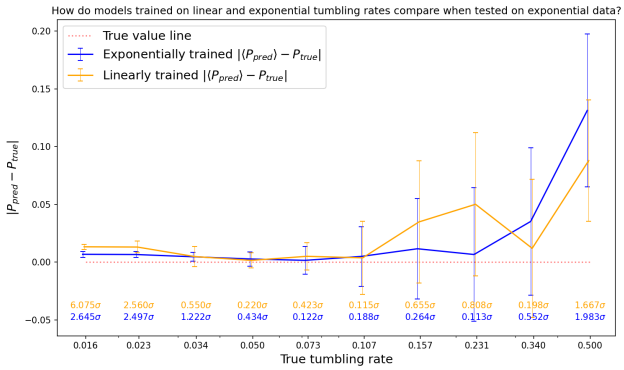


FIG. 6: A comparison of a model trained on linearly separated tumbling rates to one trained on normal exponentially separated tumbling rates. The exponential trained model dominates the middle tumbling rates and linear performs better at high tumbling rates.

tumbling rates.

Tumbling Rates

This leads into an issue raised by the previous two members who worked on this project who suggested looking into how the tumbling rates that are used to train the data affect the model. They used a logarithmic scale as this is the scale that changes within the structure of the model were mathematically predicted to change, however using a linear scale would mean the higher tumbling rates are less spread out and the model was not trained particularly for low tumbling rates, possibly generalising better to higher tumbling rates. The results of a comparison are shown in Fig 6, where the linearly trained set does seem to make up for the lack of accuracy at high tumbling rates for the compromise of lower accuracies in mid tumbling rates, yet still maintaining relative accuracy at lower tumbling rates. This does give more evidence that the model is able to predict much better at lower tumbling rates even when given an equal amount of these rates in the training data, suggesting the percolation is a strong part of the model's technique for determining the tumbling rate.

FUTURE EXTENSIONS

Possible areas for even further extension still exist in many places. Almost all models unless necessary (for example to stop the kernel crashing) were trained with the same batch size, learning rate and number of epochs. Using the learning curves shown here can help with making these decisions. Additionally we wanted to look at loss landscapes to see how complex the loss minimisation was for the model to see how far from an optimal solution the models were. More work could also be done on breaking down the model's layers to interpret what each kernel is doing rather than just seeing if they match already known filters which may unveil new ways to process these images to get key information from them.