UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Exploring the Emergence of Temperature Concept within Deep Neural Networks through Next-Frame Image Prediction

Author:
Dario GALLEGO

Supervisors:
Dr. Oriol PUJOL
Dr. Jordi VITRIÀ

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2023

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Exploring the Emergence of Temperature Concept within Deep Neural Networks through Next-Frame Image Prediction

by Dario GALLEGO

Deep learning has revolutionized numerous domains by enabling the creation of powerful models capable of learning complex patterns from vast amounts of data. However, the intrinsic opacity of deep neural networks has raised concerns about their decision-making processes, limiting their application in critical domains such as healthcare, finance and justice.

The study of explainability in deep learning models aims to shed light on the inner workings of these models, enabling us to understand how they arrive at their predictions and uncovering emergent concepts that are not explicitly present in the training data.

A study of the emergence of the concept of temperature was performed by forcing a reconstruction and prediction of the next frame in images of physical systems of balls with movement. The goal was to determine whether the neural network could acquire an understanding of temperature.

Experimental observations were performed with architectures such as Convolutional Autoencoders and U-Net, but the neural networks did not accomplish the task of learning the temperature concept due to many needed concepts that were not considered initially. Despite that, a prediction of the temperature value was performed by using the optical flow of the system.

Acknowledgements

I would like to express my heartfelt gratitude to my two supervisors for their invaluable support and warm guidance throughout this journey. Their wealth of experience and profound insights played a pivotal role in shaping this thesis project, making it both engaging and captivating.

Additionally, I extend my deepest appreciation to those who have nourished and uplifted me during the development and writing stages. Your unwavering encouragement and nourishing words have been instrumental in my progress.

Contents

Chapter 1

Introduction

1.1 Motivation

Deep learning has revolutionized numerous domains by enabling the creation of powerful models capable of learning complex patterns from vast amounts of data. However, the intrinsic opacity of deep neural networks has raised concerns about their decision-making processes, limiting their application in critical domains such as healthcare, finance and justice.

The study of explainability in deep learning models aims to shed light on the inner workings of these models, enabling us to understand how they arrive at their predictions and uncovering emergent concepts that are not explicitly present in the training data. These emergent concepts can hold invaluable insights, enabling novel discoveries and advancing our understanding of complex phenomena. By delving into the inner layers of deep neural networks, we can try to uncover these latent concepts, providing new avenues for research and problem-solving. Furthermore, the study of emergent concepts facilitates the development of more robust and transferable models, allowing them to generalize better to new tasks and domains.

For instance, McGrath et al. (2021) present a groundbreaking study that showcases the discovery of emergent concepts within these models. By focusing on AlphaZero, an AI system developed by DeepMind and Silver et al. (2018), the paper provides insights into how the model acquires chess knowledge and uncovers latent concepts that were not explicitly present in the training data.

One of the remarkable aspects highlighted in the paper is that AlphaZero learns various chess concepts, such as opening strategies, material evaluation and mobility, without any prior human knowledge or explicit instruction. Through the process of self-play and reinforcement learning, the model autonomously discovers and refines these emergent concepts, thereby demonstrating its ability to acquire knowledge beyond what was explicitly provided.

This finding is significant as it showcases the power of deep learning models to extract meaningful and strategic insights from complex domains like chess. The emergence of these concepts reveals the model's capacity to capture and represent high-level abstractions within its neural network, enabling it to make informed decisions based on its understanding of the game and highlighting the potential of deep learning models to uncover hidden patterns and strategies that were previously unknown to human players.

We would like to emphasize the significance of explainability in deep learning and the exploration of emergent concepts, as they enable us to comprehend the reasoning behind a model's decisions and uncover novel insights that advance our understanding and utilization of AI systems. The journey towards explainable AI is not only essential for unlocking the full potential of deep learning but also for ensuring the ethical and responsible deployment of these powerful technologies.

1.2 Objectives

In this study, our primary objective was to investigate the emergence of latent concepts within neural networks. Specifically, we aimed to train a neural network to learn the concept of temperature by employing an image reconstruction and prediction task where we try to compress the relevant information into a low-dimension meaningful space. As we could think when we want to explain something to anybody, it is essential for the neural network to possess the capability to encapsulate the concept within a concise and meaningful space that facilitates comprehension. Consequently, to see temperature concept emergence we must first compress the information and subsequently try to extract the concept from there.

The task involved providing the network with two images of a physical system consisting of moving balls. The network's challenge was to generate a reconstructed image that accurately represented the system at a subsequent time point.

By designing the experiment in this manner, we sought to determine whether the neural network could acquire an understanding of velocity, which in turn would allow it to infer the concept of temperature. We were particularly interested in observing the emergence of temperature as a concept, as it was not explicitly present in the provided data. The model had to deduce velocities solely by analyzing the pixel values in the images, which served as the network's input.

1.3 Findings

Following the completion of our experimentation, it became evident that the task at hand was more challenging than initially anticipated, with the concept of temperature not emerging as clearly as expected. On the next chapters we will see what happened and we will go into the factors contributing to the increased difficulty of the task. To provide a brief preview, the underlying issue stemmed from our assumption that the neural network would inherently acquire a comprehensive understanding of various foundational concepts required to grasp the temperature concept. Some of these concepts were particle or body concept, movement, quantity, and space.

Consequently, we ended up trying to predict the temperature of the system by using the optical flow of the frames, bypassing numerous intermediate tasks necessary for bridging the gap between the frames and the temperature concept. In doing so, we encapsulated some of these concepts hidden within the optical flow, which also needs them to emerge.

1.4 Layout

The structure of this report is as follows: Firstly, it begins with an introduction where the motivation behind this work, the objectives, and a brief summary of the findings are explained. Then, it provides explanations about the background of the project, including the concepts of emergence and emergent concepts, as well as the methods employed to uncover them within black-box models.

Afterwards, a chapter is dedicated to explaining the methodologies used and the experimentation process, with a particular focus on the path that was followed as a result of the conclusions and findings obtained. Following this, there is a summary of the conclusions drawn from the research. Lastly, the bibliography is included, listing the sources referenced in this report.

Chapter 2

Background

2.1 Emergence

Emergent concepts play a fundamental role in our understanding of the world around us. These are concepts that emerge as a result of our observations and experiences, allowing us to explain and make sense of various phenomena. Temperature is a prime example of such an emergent concept.

Temperature is defined as a measure of the average kinetic energy of the particles in a substance or system (Bormashenko, 2020). It is a concept that has been recognized and utilized by humans for centuries. However, it is not a concept we invented, it was already present on the world and required careful observation, experimentation and definition to develop a comprehensive understanding of temperature and its associated concepts. (Sándor Biró, 2011)

Moreover, as our understanding of temperature deepened, we discovered the relationship between temperature and other physical properties. For instance, temperature is closely linked to the concept of velocity, especially in the context of gases. By definition, temperature measures the average kinetic energy and velocity of gas molecules. As temperature increases, so does the average velocity of the molecules.

The concept of temperature has proven to be essential in explaining a wide range of physical phenomena. It helps us understand the transfer of thermal energy, the behavior of gases, the expansion and contraction of materials, and many other aspects of physics and chemistry. Temperature is also crucial in various practical applications, such as weather forecasting, thermodynamics, and the design of heating and cooling systems.

Another pretty interesting emergent concept is emergent intelligence. Emergent intelligence refers to the phenomenon where complex and intelligent behavior arises from the interactions of simpler, individual components. It is the collective behavior or intelligence that emerges from the interactions and organization of individual elements or agents in a system.

One of the most fascinating examples of emergent intelligence can be found in social insect colonies, such as ant colonies and beehives. Individually, ants or bees may exhibit relatively simple behavior, but when they interact and work together, they exhibit highly organized and intelligent behavior at the colony level. Tasks such as foraging, building intricate nests, and defending the colony emerge from the combined actions of many individuals following simple rules. (Wheeler, 1911; Gordon, 2000; Schmid-Hempel, 2002; McCreery, 2017)

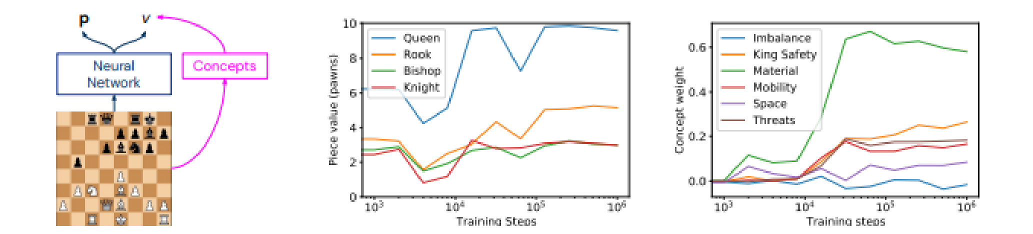
Another notable example of emergent intelligence is found in the field of artificial intelligence and machine learning. Neural networks, inspired by the structure and function of the human brain, are composed of interconnected nodes that work together to process information. Individually, these nodes may only perform basic computations, but when they are connected and well-trained on large datasets, the network can exhibit complex behaviors such as image recognition, natural language processing, and even learning concepts that were not explicit on the dataset.

Emergent intelligence can also be observed in human societies. Human beings, as individuals, possess a range of cognitive abilities, but it is through social interactions and collaboration that our collective intelligence emerges (Luo et al., 2008). The exchange of ideas, knowledge, and perspectives leads to the generation of new insights, innovations, and problem-solving strategies that surpass the capabilities of any individual. By studying emergent intelligence, researchers seek to uncover the underlying mechanisms that lead to the emergence of intelligent behavior, with the potential to inspire new approaches in many fields (Woolley et al., 2010).

2.2 Interrogating black-box models

Interrogating black-box models, particularly deep learning models, is a valuable approach to uncover latent emergent concepts and gain insights into their internal workings. This process involves examining the inner mechanisms and representations of the model to better understand how it operates and acquires knowledge.

McGrath et al. (2021) employs various techniques to interrogate the neural network and understand its acquired knowledge. They delve into the neural network's representations and examine its response to specific chess positions and moves.



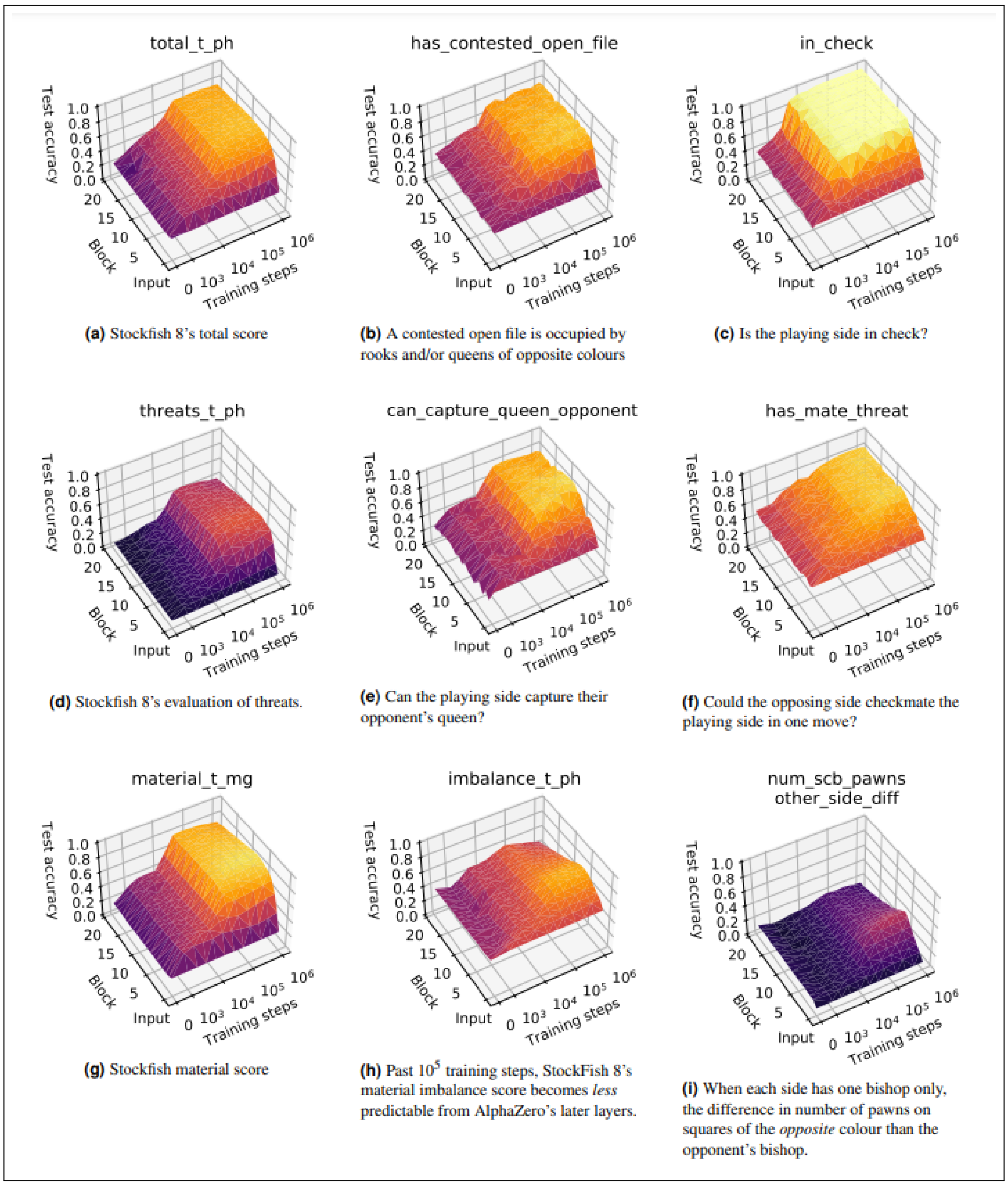
(a) Value regression methodology: we train a generalized linear model on concepts to predict AlphaZero's value head for each neural network checkpoint.

(b) Piece value weights converge to values close to those predicted by conventional theory.

(c) Material predicts value early in training, with more subtle concepts such as mobility and king safety emerging later.

FIGURE 2.1: Value regression from human-defined concepts over time. (with copyright permission from the journal)

One technique used in interrogating the model was training a sparse regression model that learnt from the activations across different layers and training steps and tried to predict both continuous and binary concepts that humans use when playing chess (see figure 2.1). By scoring concept regression over the layers and temporal axes as AlphaZero was trained, they were able to visualise the acquisition of conceptual knowledge. This means that the accuracy of these models could be used as a proxy to measure the presence or absence of (linearly-decodable) information. By comparing accuracy scores across layers and checkpoints they aimed to understand what concepts were encoded, when in training they emerged, and where in the network they were strongly encoded.



(a) Stockfish 8's total score

(b) A contested open file is occupied by rooks and/or queens of opposite colours

(c) Is the playing side in check?

(d) Stockfish 8's evaluation of threats.

(e) Can the playing side capture their opponent's queen?

(f) Could the opposing side checkmate the playing side in one move?

(g) Stockfish material score

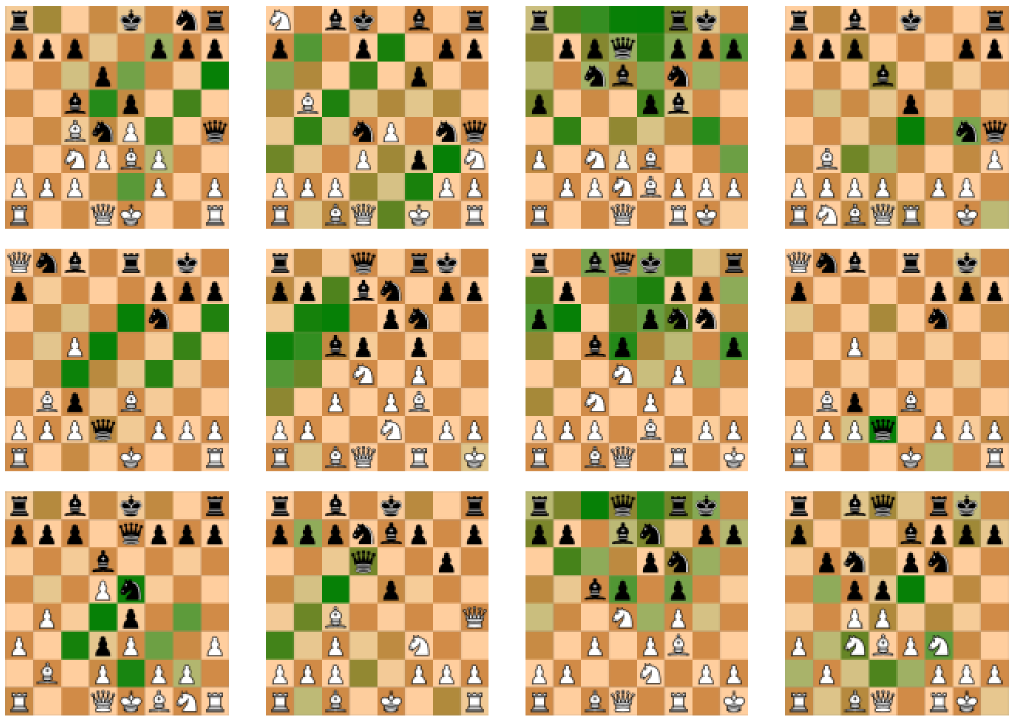
(h) Past 10^5 training steps, StockFish 8's material imbalance score becomes *less* predictable from AlphaZero's later layers.

(i) When each side has one bishop only, the difference in number of pawns on squares of the *opposite* colour than the opponent's bishop.

FIGURE 2.2: What-when-where plots for a selection of Stockfish 8 and custom concepts (with copyright permission from the journal)

They also tried to examine where and when were these concepts learned (see figure 2.2) and their relationship over time with the network's ability to estimate the value of chess positions. The value function in AlphaZero represents the predicted outcome or advantage of a particular chess position. The authors investigated whether those concepts acquired by the network contributed to its ability to estimate the value function effectively. It emphasized how the acquired knowledge, represented by those concepts, influenced the network's estimation of the value of chess positions, ultimately impacting its gameplay performance.

Additionally, an expert complemented their experiments with a qualitative assessment. This assessment was an attempt to identify themes and differences in style of play between AlphaZero checkpoints in the early phase of training. The expert concluded that the neural network was learning a list of concepts and improving its performance across training time.



(a) Development of diagonal moves for player (block 1, factor 26 of 36). **(b)** Fully developed diagonal moves for opponent (block 3, factor 22 of 36). **(c)** Count of opponent's potential piece moves (block 3, factor 11 of 36). **(d)** Potential good squares to move to? (block 18, factor 22 of 36).

FIGURE 2.3: Visualisation of Non-negative Matrix Factorisation factors in a fully-trained AlphaZero network, showing development of threats and anticipation of possible moves by the opponent, as well as a factor that may be involved in move selection (with copyright permission from the journal).

Finally, they also took an alternative approach, using simple unsupervised approaches involving non-negative matrix factorisation and correlation analysis. From there they tried to uncover what factors were involved in move selection (see figure 2.3). We will not focus on this approach for our experiments.

In our experiments we will try to compress the temperature concept into the low-dimensional bottleneck layers and we will try to interrogate the model by predicting the temperature concept with a neural network that uses as input the activation layers of the main model.

Chapter 3

Methods and Experimentation

3.1 Experimentation protocol

First and foremost, the code used for the experiments discussed in this section can be accessed at the following GitHub repository.

3.1.1 Hypothesis

Let us consider a physical system of balls with movement. We assume that the task of predicting how is the system going to look in a time instant by looking at two previous instants needs the concept of temperature to be solved when compressing. The human approach would be to look at the different balls, identify the correspondence between balls of the first instant and second instants, compute their positions and movement vectors and apply this vector by the time step to the positions on the second instant. For this reason we would be learning the velocity concept, and therefore the temperature concept should emerge. To simplify this task we will assume that the time step between instants is always the same.

We set as an hyphotesis that the temperature concept should emerge from the training of the neural network on this task, while not being present on the input, and we should be able to find this concept in the intermediate layers. To be able to do this we will try to compress the information that goes through the encoder into a low-dimensional bottleneck layer to force the neural network to learn relevant data.

3.1.2 Defining the experimental setting

The proposed experimental setting can be described as:

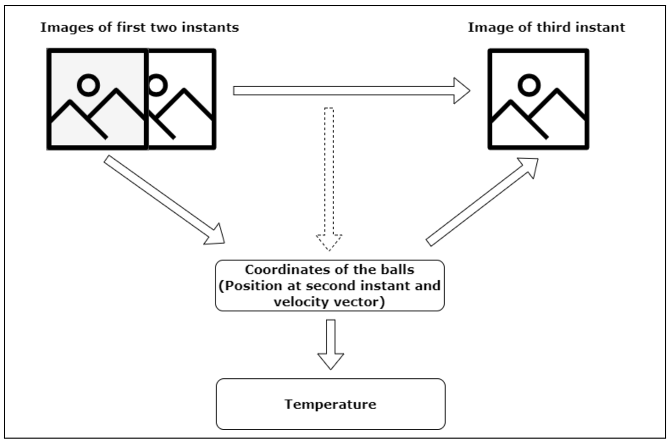

Images of first two instants
Image of third instant
Coordinates of the balls (Position at second instant and velocity vector)
Temperature

FIGURE 3.1: Experiment diagram

We used images to represent the physical systems at each instant. Our intention was to train a neural network to solve these tasks without having explicitly defined the velocity or the temperature in the data.

We could solve this task following the two paths in figure 3.1:

1. In order to enforce the concept to emerge due to the compression of the information we need a bottleneck strategy (e.g. an autoencoder-like architecture)

2. However, in order to verify whether this is even possible, we could force the intermediate representation of both positions and velocities.

This gives rise to two different experiments answering "is it possible to regress positions and velocities from the images?" and "is it possible to infer and reconstruct the image from the intermediate representations?".

In the second approach we tried to regress both the coordinates learnt on the first task of the other approach and the temperature of the system, defined as the mean of the velocity norms of all balls.

Therefore, we set up different experiments to solve the different parts and paths of the task. One experiment would be to use the first two images as input to predict the third time instant, and then try to regress the temperature value of the system from the intermediate layers of the neural network.

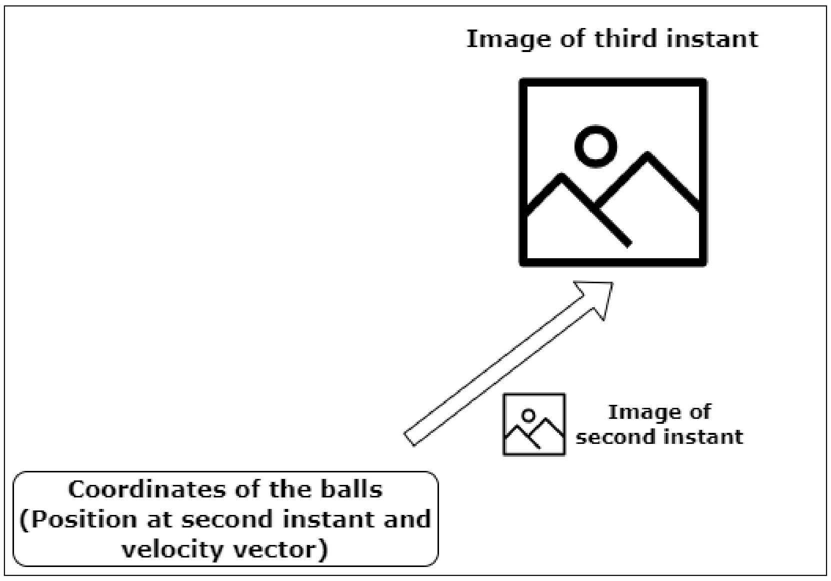


Image of third instant
Image of second instant
Coordinates of the balls (Position at second instant and velocity vector)

FIGURE 3.2: Diagram of the second task in the indirect path

The other two experiments would be to try to infer the coordinates of each one of the balls and their velocity vectors using the previous two images as input, and to predict the third instant image (as in figure 3.2) using as input the coordinates of the balls, and also the image of the second instant, to have context about the type of reconstructed image we want (to know the size of the balls, their color, have a reference axis or axis origin, etc.)

In all experiments we tried different combinations of features in the images, such as using one or multiple balls (but same amount for all images), and coloring the balls or using gray-scale.

To be able to generate the images representing the physical systems we used a simulations library called PyBullet.

3.1.3 PyBullet

PyBullet (Coumans and Bai, 2016) is a physics engine and a Python interface to the Bullet Physics library. It serves as a powerful tool for simulating and analyzing physical dynamics in virtual environments. In the context of creating our dataset, PyBullet was utilized to generate the simplified physical systems consisting of balls with movement.

To create the dataset, PyBullet provided the necessary capabilities for physics simulation and rendering. It allowed us to define the physical properties of objects, such as size, shape, position, velocity and friction. We used PyBullet's simulation capabilities to model the behavior of the balls, specifying their initial positions and velocities using a uniform distribution to encompass a wide range of movements, including different directions and magnitudes. We also had to disable some properties that were automatically initialized (gravity, friction, damping, etc.) to keep the same velocity vectors across all instants.

PyBullet's physics simulation then evolved the system over time, updating the positions and velocities of the balls based on their physical properties. At each time instant, we stored the coordinates vector that served later to generate the images.

In summary, PyBullet is an incredibly comprehensive and extensive library that offers a wide range of functionalities for physics simulation and robotic control, and while our task didn't require us to delve into the full depth of PyBullet's capabilities, the library proved to be a valuable asset in achieving our objectives efficiently, serving as the foundation for generating our dataset.

3.2 Experiment 1: The direct approach. Regressing temperature from the bottleneck layer

Our first experiment consisted on two parts:

First we wanted to predict the third instant by inputting the previous two instants and we tackled this task by training an autoencoder-like neural network to predict and reconstruct this third image by looking at the images of the physical system on the first and second instants. Autoencoders were introduced by Rumelhart, Hinton, and Williams (1985) and are very useful neural networks to compress meaningul information into low-dimension latent spaces (Hinton and Salakhutdinov, 2006).

Secondly, once we had a good enough model, we wanted to infer on the hidden layers to regress the temperature. We expected to see the emergence of the temperature concept.

3.2.1 Dataset

The intended dataset was a collection of 2D images of 3 different instants in simplified physical systems consisting on a few circles that represented balls with movement. All of them had the same magnitude in their movements (same norm in the velocity vector) but different directions. Both the position of the balls and the velocity of the vector were chosen by using a uniform distribution to ensure that all possible movements (more movement, less movement, different directions and positioning across all the space) were considered. In the case of the velocity the uniform distribution was closed in an interval with big enough values to ensure that there was always some movement. Another thing we took into account was to get a square grid where all balls were contained and we set the grid space to the limits of this square to ensure that all balls were displayed and had the same axis, avoiding that we had similar samples with the same directions of movement but at different velocities.

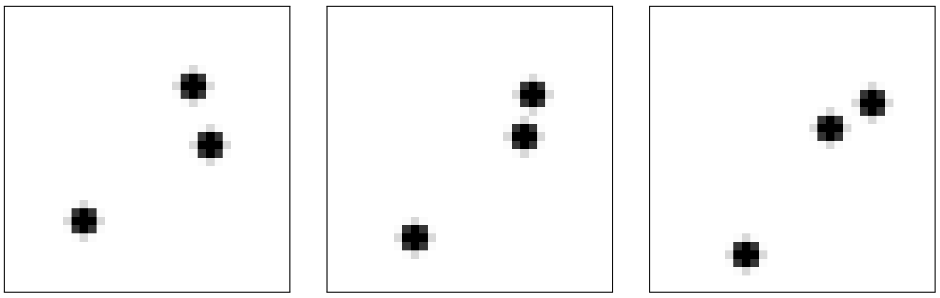


FIGURE 3.3: Images of the 3 instants (ordered from left to right) of a sample

The dataset consisted on 10000 samples (like the one in figure 3.3) with 3 frames each, with 3 balls on each image, of size 32x32x1 (we used gray-scale for simplicity).

We also collected the velocity vectors (v_x, v_y) of the 3 balls. The velocity vectors remained constant across time, so we only stored them once. From these velocity vectors we computed the temperature of the system, which was the mean of the norms of these vectors.

Initially, we stored the vectors relative to the simulation, but due to the pixeling limitations of the image (pixels are a discrete value, and because of that we can have notably different velocity vectors with the same visible movement in the collected images) we ended up plotting each ball separately, in the same way we plotted to save the images, just to collect the coordinates and velocities relative to the compressed image, to then plot again with all the balls and store this image. By compressed image coordinates we refer to the pixel indexes of the center of the ball. To compute it we got all the pixels where there was a part of the ball (which was always 5 in both axis) and we got the mean. Then, we used directly these values to compute the velocity vector.

3.2.2 Setting 1: Convolutional Autoencoder

Convolutional autoencoders (Zhang, 2018) are a type of neural network architecture that combines convolutional neural networks (CNNs) with autoencoders. They efficiently learn and extract meaningful features from high-dimensional data, with a specific emphasis on image data. They are primarily used for unsupervised learning tasks, such as dimensionality reduction, feature extraction, and image denoising.

To understand a convolutional autoencoder, let's first discuss its two main components: the encoder and the decoder.

- Encoder: The encoder part of a convolutional autoencoder takes an input image or data and gradually reduces its dimensionality through a series of convolutional layers. These layers are designed to extract meaningful features from the input by encoding it into a lower-dimensional latent space. Each convolutional layer applies a set of learnable filters to the input, which convolves across the image and produces feature maps. The filters capture different patterns, such as edges, textures, or shapes, at various spatial scales. Pooling layers are often inserted after convolutional layers to further downsample the feature maps and retain the most salient information.

- Decoder: The decoder component of a convolutional autoencoder takes the low-dimensional representation produced by the encoder and reconstructs the original input image or data. It uses transposed convolutional layers (also known as deconvolutional layers or upsampling layers) to gradually upsample the feature maps back to their original dimensions. Similar to the encoder, the decoder consists of a series of learnable filters that convolve across the feature maps and generate reconstructed outputs. The final output of the decoder should ideally be as close as possible to the original input.

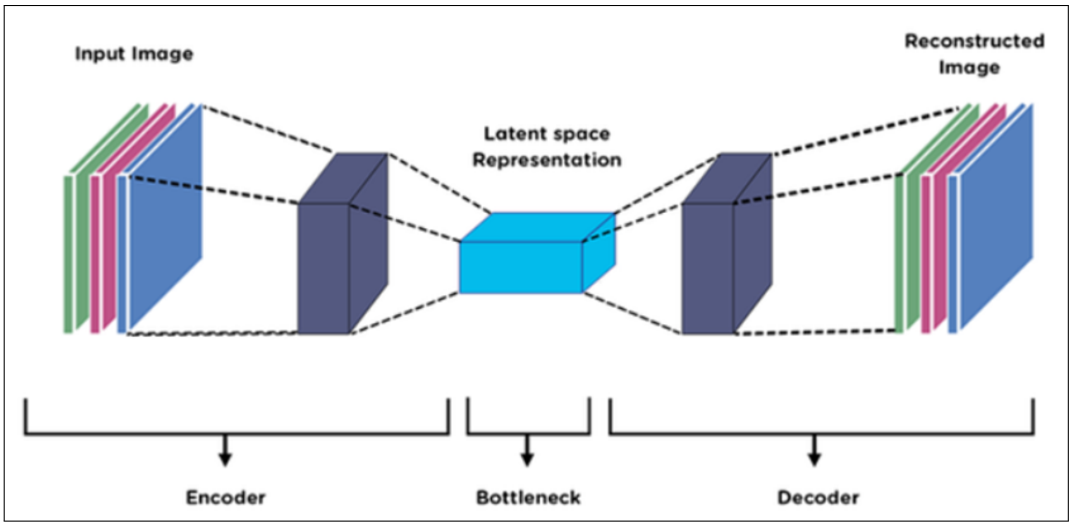


FIGURE 3.4: Convolutional Autoencoder

Therefore, a convolutional autoencoder (figure 3.4) consists on an encoder, that compresses the latent concepts into a low-dimensional latent space that represents the images, followed by a decoder, that reconstructs the images from the encoded representation.

The training process of a convolutional autoencoder involves minimizing a loss function that quantifies the difference between the input and the reconstructed output. The most commonly used loss function is the mean squared error (MSE), which measures the pixel-wise difference between the two images. By minimizing this loss, the autoencoder learns to encode and decode the input data efficiently, capturing the most important features while discarding unnecessary information.

Convolutional autoencoders have been widely used in computer vision tasks, such as image reconstruction, denoising, inpainting, and anomaly detection. They have also found applications in various domains, including medical imaging, autonomous vehicles, and natural language processing.

Experimentation

On this setting we built our own convolutional autoencoder.

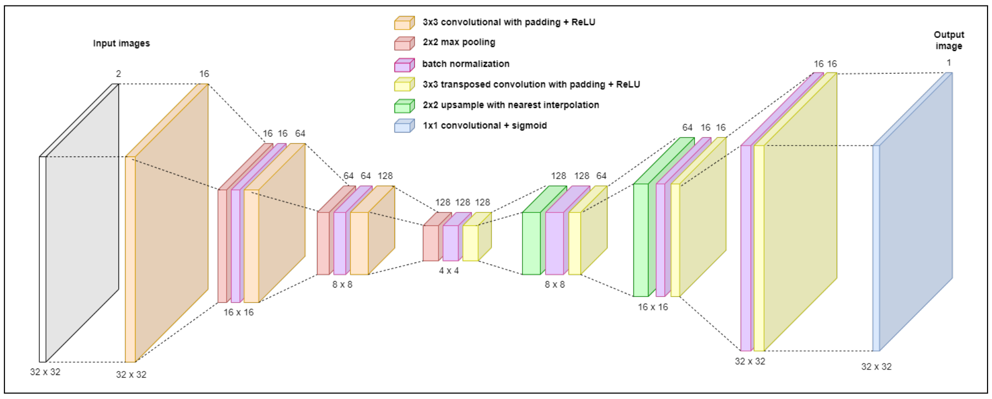


FIGURE 3.5: Initial convolutional autoencoder. The number of channels is denoted on top of the layer. The x-y-size is provided at the bottom of the layer.

As we can see in figure 3.5, we started with an encoder consisting on 3 downsample blocks, 1 bottleneck block and 3 upsample blocks. Each enconder block was composed by a 3x3 convolution with ReLU activation and same padding followed by a 2x2 MaxPooling and batch normalization. On the bottleneck layer we tried with and without adding extra fully-connected layers to try to reduce the dimensionality of the intermediate layer. Lastly, the decoder block were composed by 3x3 transposed convolutions with ReLU and same padding followed by a 2x2 upsampling using nearest interpolation and batch normalization. We ended with a 3x3 transposed convolution with Sigmoid activation and just one filter to summarise all the information into the output image.

The dense layers purpose was to further decrease the dimensionality of the bottleneck and to break the spatial correlations associated to the image data structure. However, they were worsening the predictions of the model since it was losing the spacial information and was scattering the colored region (corresponding to the balls). Consequently, we proceeded with an architecture that did not incorporate fully-connected layers.

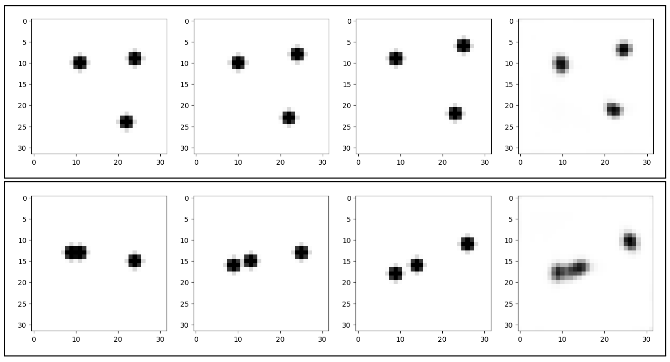


FIGURE 3.6: Two examples where we can see (from left to right) the three frames and the prediction of the autoencoder. As we can see the positions predicted are good but not perfect and there is some scattering when multiple balls are near.

After training the model and checking that the predictions (figure 3.6) were good enough to consider the prediction task to be learnt, we tried to build some temperature regressors feeding them with the flattened output of the bottleneck layer (the previous layer to the first transposed convolution) of size 4x4x128.

We tried by building a neural network consisting on stacking dense layers with no activation (using ReLU or other activations worsened the results), and by using multiple built-in Scikit-learn regressors such as LinearRegression, MLPRegressor, GradientBoostingRegressor, RandomForestRegressor and KNeighborsRegressor, and XGBRegressor from the XGBoost library.

The predicted temperatures were not accurate enough (the best mean squared error we got was 0.0085, but the models were too far from the actual values in many samples; in some cases the regressors were just predicting the same value for all samples) to consider that the temperature concept was inside the bottleneck layer and the regressor was able to learnt it from there. For that reason and acknowledging the subpar quality of the reconstructed images, we tried with a U-Net, a more complex architecture that we expected to yield better reconstructions and hopefully leading to the emergence of the desired concept.

3.2.3 Setting 2: U-Net

As our second setting we built a model using a U-Net architecture. Due to the incorporation of skip connections we expected a better performance against the one we got with the convolutional autoencoder.

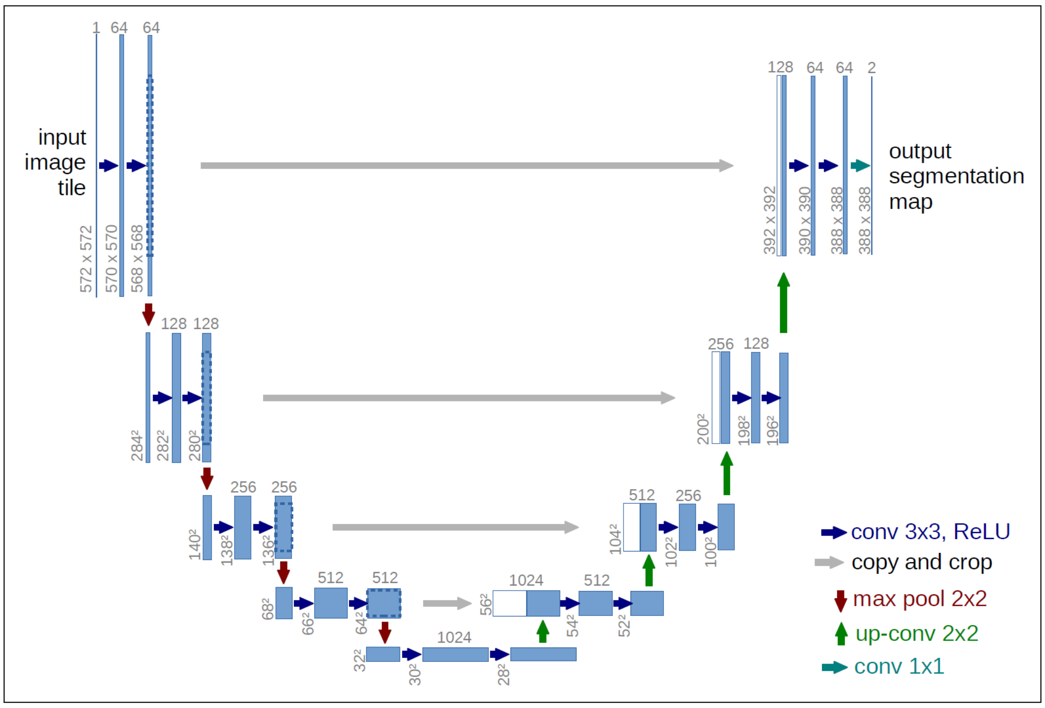


FIGURE 3.7: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. (with copyright permission from the journal)

The U-Net architecture is a convolutional neural network (CNN) architecture that is widely used for image segmentation tasks, particularly in biomedical image analysis. It was proposed by Ronneberger, Fischer, and Brox (2015) and has since become a popular choice due to its effectiveness in capturing both local and global image features.

The U-Net architecture derives its name from its U-shaped network design (figure 3.7). It consists of an encoder path and a decoder path, with skip connections between corresponding encoder and decoder layers. This design allows the network to combine low-level and high-level features from different scales of the input image, enabling precise localization and segmentation.

The encoder path of the U-Net architecture resembles a typical CNN, where convolutional and pooling layers are used to progressively downsample the input image. These layers extract hierarchical features, capturing both local and global context. Each downsampling step in the encoder halves the spatial resolution while increasing the number of feature channels.

The decoder path of the U-Net is responsible for upsampling and reconstructing the segmented output. It consists of transposed convolutional layers, also known as deconvolutional layers, to gradually upsample the feature maps. These layers perform the reverse operation of convolution, expanding the spatial dimensions while reducing the number of feature channels.

The skip connections in the U-Net architecture are a key feature that enables the fusion of multi-scale features. These connections directly connect corresponding layers between the encoder and decoder paths. By merging features at different scales, the U-Net can capture fine-grained details while maintaining a holistic understanding of the input image.

The final layer of the U-Net typically uses a 1x1 convolution to map the feature maps to the desired number of output channels, representing the segmented output. The activation function used in the final layer depends on the specific segmentation task, such as sigmoid for binary segmentation or softmax for multi-class segmentation.

The U-Net architecture has shown excellent performance in various image segmentation tasks, including cell segmentation, medical image segmentation, and semantic segmentation. Its ability to capture multi-scale features and leverage skip connections makes it particularly effective in scenarios where precise localization and accurate segmentation boundaries are crucial.

In summary, the U-Net architecture is a convolutional neural network that combines an encoder path for feature extraction with a decoder path for upsampling and reconstruction. It utilizes skip connections to merge multi-scale features, enabling precise image segmentation.

Experimentation

With the advantages of the U-Net in mind, we aimed to improve the predictions and reduce some struggles we saw with the convolutional autoencoder, such as the scattering in the images, as it focused primarily on capturing global image features.

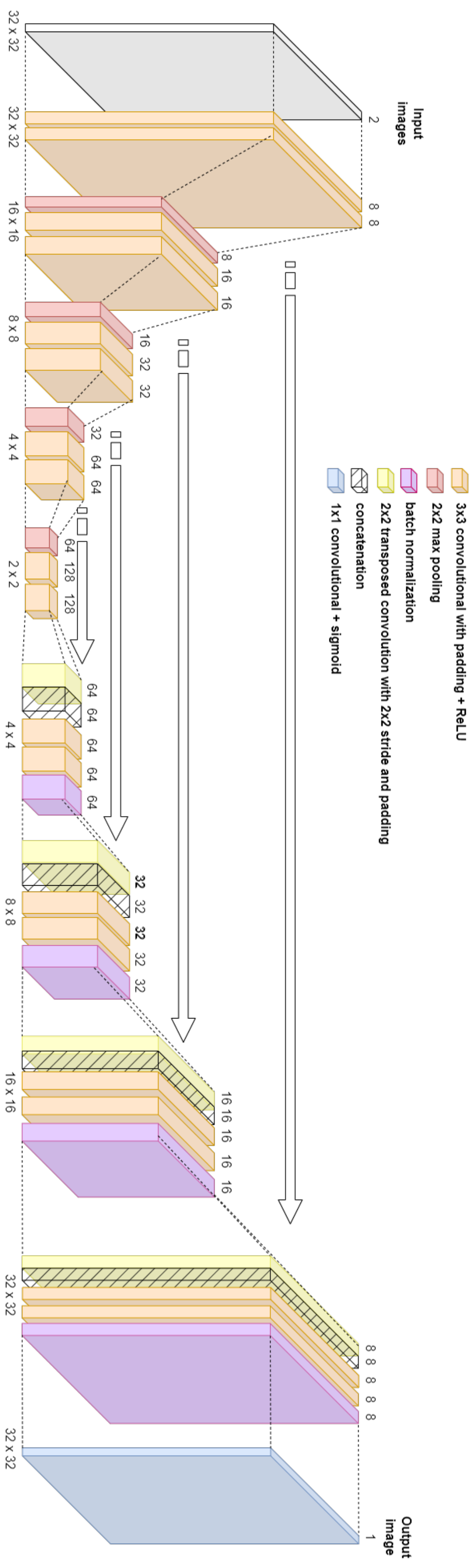


FIGURE 3.8: Our version of the U-Net architecture. The number of channels is denoted on top of the layer. The x-y-size is provided at the bottom of the layer. The arrows represent the skip connections.

We used a U-Net architecture maintaining a similar structure of layers and size of them to the one used with the convolutional autoencoder. Our U-Net (figure 3.8) consisted on 4 downsampling blocks (2 3x3 convolutions with same padding and ReLU followed by a 2x2 MaxPooling) 1 bottleneck block (composed by 2 3x3 convolutions with same padding and ReLU) and 4 upsampling block (consisting on a 2x2 transposed convolution with same padding and 2x2 strides, concatenated with the MaxPooling output of the respective downsampling block and ended with 2 3x3 convolutions with ReLU and padding and a Batch Normalization to close it).

In contrast with the convolutional autoencoder, we added dropout in between the 2 convolutions in some blocks to help generalize and only used batch normalization on the upsampling blocks.

We reduced (increased) the dimensionality of the layers by half (by double) on the downsampling (upsampling) blocks. Similarly we doubled (starting at 8) the number of filters applied on each downsampling and reduced them by half (also ending at 8) on each upsampling block, getting a bottleneck layer with dimension 2x2x128 (4 times smaller than the one we got with the convolutional autoencoder).

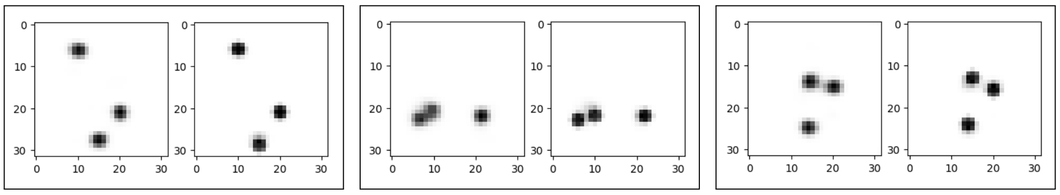


FIGURE 3.9: Predicted image and image to predict over the prediction of 3 samples.

As we can see on figure 3.9, the U-Net captured the positions of the balls more accurately and had less scatter. This time, to regress the temperature we used the output from the last concatenation because we would be connecting the all the paths that send processed information (all the skip connections and the processed information through the downsampling). The used regressors consisted on a linear regression, a completely fully-connected neural network and a convolution followed by a fully-connected network. None of this regressors was able to predict the temperature from the system, with the best model getting around a 0.008 mean squared error (MSE). Consequently, we revised the intermediate layers by looking at the activation layers and trying to understand what was happening.

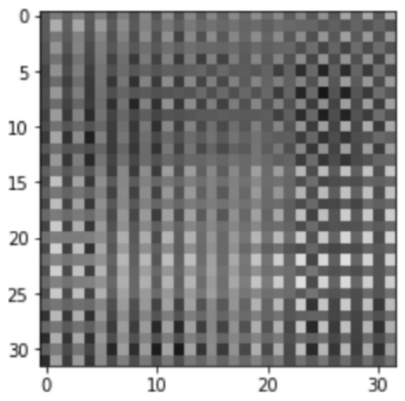


FIGURE 3.10: Activation layer with checkerboard artifacts

First, we saw that checkerboard artifacts were appearing on the activation layers (figure 3.10). This was due to the MaxPooling. Thus, we changed the MaxPooling layers by 3x3 convolutions with stride 2x2 to avoid this problem but still reduce the dimensionality (we also added a 1x1 padding to reduce the dimensionality by half). For this same reason we changed the transposed convolutions by just the upsampling step interpolating with the nearest pixels.

Another modification was to remove the dropout layers and every batch normalization from the neural network. This also led us to realize that we could simplify a lot the neural network while not losing much precision (we would get more scatter but the positions of the balls would still be well predicted). By doing this we reduced the computation time and the dimensionality of the intermediate layer, being now 32x32x2 (because we were using the last concatenation layer) where it was 32x32x16 before.

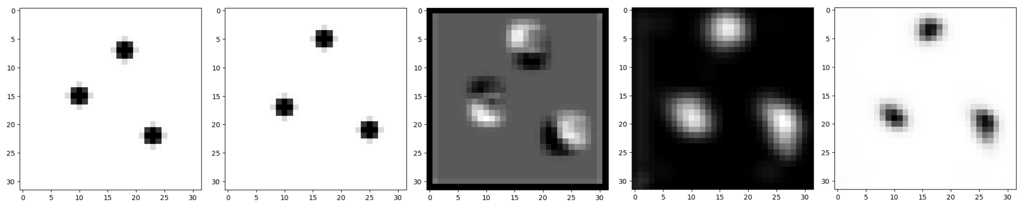


FIGURE 3.11: From left to right: input images (first and second instant), bottleneck layer (the one coming from the skip connection and the one coming from the upsampling) and the output image.

After applying all this modifications and looking again at the activation layers (figure 3.11) we could see that the neural network was learning the positions of the balls at the first and second images in a way that looker very similar to the computation of the derivative of the position: the velocity. Nonetheless, we were still not able to regress the temperature. At best we got a MSE around 0.0085, but we still had some regressors that were predicting the mean of the values to predict.

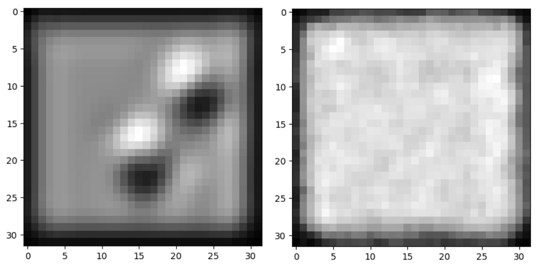


FIGURE 3.12: Activation layers that were concatenated on the bottleneck layer. On the left the activation coming from the upsampling and on the right the one coming from the dense layer of the skip connection.

Another thing we tried was adding a two fully-connected layers on the bottleneck to force the neural network to compress the information but this had no success. Even though, this showed us that, while the skip connection improved the predictions, it was not really needed to get decent predictions. It was sending noise (figure 3.12), that helped generalize, but the real learning was happening on the other path.

Trying with colored images

We generated a similar dataset with the only difference that now the images were colored, going from 32x32x1 to 32x32x3 in the images dimensions. We thought this could be useful to identify the correspondence between the values of the array and the balls. The temperature value and velocity vectors were computed the same way as before.

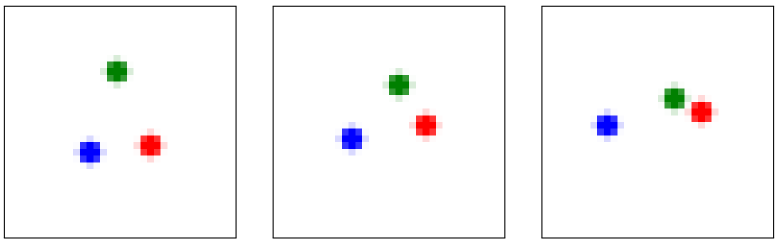


FIGURE 3.13: Sample of the colored dataset.

By using one different color for each ball, as it can be seen in figure 3.13, we thought that it would be easier for the neural network to detect the correspondence between balls in different instants.

We used the same architecture as with gray-scaled images (adapting both the input and output layers) and after running the experiments we saw that this approach did not yield any significative improvement over the previous.

3.2.4 Setting 3: Restricted Convolutional Autoencoder

Having seen that the skip connections were not really helping to regress the temperature (the important information was slipping in through the residual path when we added the fully-connected layer on the skip connections), we went back to the convolutional autoencoder.

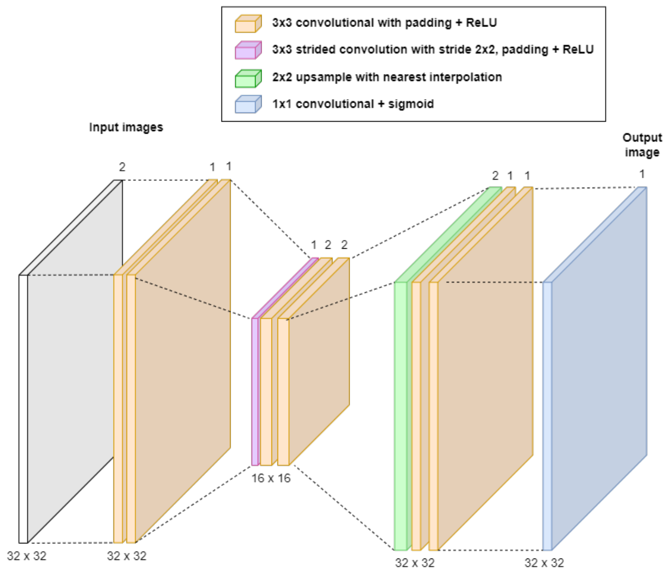

3x3 convolutional with padding + ReLU
3x3 strided convolution with stride 2x2, padding + ReLU
2x2 upsample with nearest interpolation
1x1 convolutional + sigmoid
Input images
Output image
2
1 1
1 2 2
2 1 1
1
16 x 16
32 x 32
32 x 32
32 x 32
32 x 32

FIGURE 3.14: Simpler convolutional autoencoder. The number of channels is denoted on top of the layer. The x-y-size is provided at the bottom of the layer.

Nevertheless we want back to a simpler convolutional autoencoder (figure 3.14) with one downsample block (2 3x3 convolutions with padding and ReLU and a strided convolution with 3x3 filter, 2x2 stride, padding and ReLU activation), one bottleneck block (2 convolutional layers with same parameters as before) and an umsampling block (one 2x2 upsampling layer and 2 convolutional layers).

The biggest simplifications on the dimensionality of this architecture were that we only used one downsampling and one upsampling blocks, and the number of channels used on each layer (having at most 2 channels per layer).

This model was able to reconstruct and predict correctly because the balls were not moving more than 4 pixels, from one instant to the other, on any axis. Therefore, the 2 3x3 convolutions were able to always capture this movement. If we considered bigger images or balls moving faster we should increase the number of layers, as we had in the convolutional autoencoder from the initial setting.

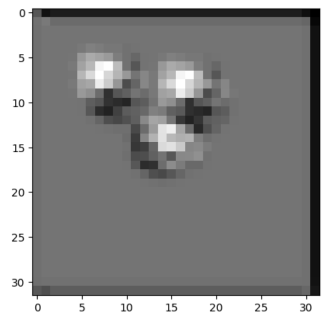


FIGURE 3.15: Activation layer from the simpler convolutional autoencoder

As we can see on figure 3.15, the position and its derivative still seemed to be there, although the activation layer did not look as smooth as before.

With this simpler model we believed that by reducing the complexity we removed redundant information while keeping the relevant. Although we were still not able to infer the temperature from its activation layers. We even went back; increasing the MSE to 0.024.

Trying with just one ball

We used a simpler approach (considering just images with one ball) to see if the actual problem was having more than one ball and this was confusing the neural network, because it might not be able to identify which ball on the second image corresponded to another ball in the first one.

Unluckily, the neural network was just minimizing the error by giving the same value to all the pixels. Having just one ball on the image meant having a lot of white pixels in the image, and this impacted into that bad solution.

3.3 Experiment 2: From images to concepts. Regressing coordinates from the bottleneck layer

After seeing that we were not able to extract the temperature concept from the weights of the neural network, we tried to infer the coordinates of the ball and its velocity vectors instead.

On this experiment we took the already built model on the previous experiment and try to infer from its activation layers.

3.3.1 Dataset

For this experiment we reused the images generated for the previous experiment while adding the data respective to the coordinates of the balls on each image. To get this data we took the same steps as we did when we got the velocity data: we plotted each ball separately and collected the pixel corresponding to the center of the ball as its position. These were the values we used to compute the velocity vectors, but this time we stored them.

We had to keep an array of size $8n$ for each one of the samples, where n is the number of balls in the image. This size comes out of having to store the 2-dimensional position of each ball for the 3 instants ($6n$), and the velocity vectors of all balls ($2n$). In fact, we actually only used the coordinates of the balls on the second image and the velocity vectors, reducing by half the needed array. We regressed these values because those would be the ones used (with the second image) to predict the third instant, on the task represented in figure 3.2.

This experiment was run with the 2 image sets with multiple balls we had generated before (3 black balls and 3 balls with different colors).

3.3.2 Setting and results

We used the model we ended the previous experiment with. Thus, there were no variations on the architecture and model aspects. One thing we did modify was the way to extract the information from the intermediate layers. We saw that one problem that might be arising was that we were flattening the activation images (which were outputs of convolutional layers) and loosing the spatial context. Hence, we adopted a new way procedure with the regressor: we started with a convolutional layer with the filter size being the same as the input size, just to get one value, and applying many filters. Those values (with as many as filters applied) were then flattened, to reduce the dimensions to one, and used in a fully-connected network.

After using this approach, we saw that in both datasets the coordinates and velocity vectors were not accurately predicted. Against our initial belief, assigning one color to each ball to distinguish them in the following images did not help the neural network to predict the positions and coordinates of the ball. For instance, the array contained the positions of the balls in an specific order, which was also present in the coloring of the balls. Across all samples the first ball had always the same color, and equivalently for the other balls.

While this could still be a problem the gray-scaled images with multiple balls had that did not seem to be the only problem, because with colored balls we were still not able to predict the coordinates.

Owing to that there were no good results (best MSE was around 0.08; the increase in MSE was also due to predicting $8n$ instead of just one) on this task we aimed to see if that was also happening on a dedicated task where the neural network was using the initial images as input instead of an activation layer output.

3.4 Experiment 3: Regressing coordinates from initial frames

On this third experiment we tried to regress the coordinates of the balls: their position (on the second instant) and their velocity vectors with a neural network that used as input the images of the first two instants.

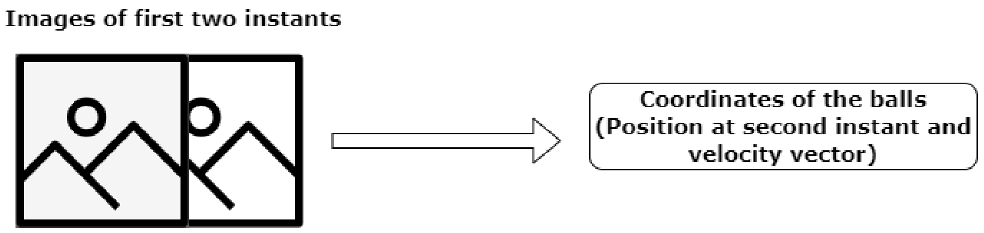

Images of first two instants
Coordinates of the balls (Position at second instant and velocity vector)

FIGURE 3.16: Diagram of the 3rd experiment.

With the approach in figure 3.16 we were actually asking the neural network to solve this problem, while before we were solving another task completely different (to predict and reconstruct the next instant) and then using its activation layers to solve this second task.

What could have happened was that, due to not trying to solve this task from the beggining, some of its relevant information was lost in the process. Therefore, we expected the neural network to easily solve this task with this dedicated approach.

The dataset used for this experiment was the same as in the previous experiment (it was the same task with a different approach) but this time we did not need the 3rd image, since we were not training a neural network to reconstruct and predict the 3rd instant. Additionaly, this time we also used the imageset of physical systems with one ball.

3.4.1 Settings and results

For this experiment we used the same architecture we had on the previous experiment but adding the encoder from the convolutional autoencoder. By doing this we ensured that there were no architecture deviations and the only difference was on the training of the model, training all the layers on this experiment and using a pre-trained block on the previous one.

The architecture utilized consisted on two 3x3 convolutional layers (of 1 filter) with ReLU activation and no padding, followed by a 28x28 convolutional layer (of 6 filters) also with ReLU and no padding, and two dense layers with 64 and $2n$ nodes respectively, where n was the number of balls on each image. Another thing to note is that from now on we decided to consider the mean absolute error (MAE) instead of the MSE when training the regressors. This decision was taken because the values we were trying to predict were small and we wanted to avoid minimizing the loss function by predicting the mean value for all samples.

	3 black balls dataset	**3 colored balls dataset**	**1 black ball dataset**
MAE	0.20	0.20	0.15

TABLE 3.1: Table that shows the Mean Absolute Error when predicting the coordinates of the system by using the two initial images as input in the 3 datasets.

The obtained results in 3.1 were the same as with the indirect approach (no good results on both datasets with 3 balls per image). Besides that, we got good predictions when we had just one ball, but only on the position coordinates.

3.5 Experiment 4: Introducing the optical flow

On the initial design of the experiments we thought that the task of predicting the next frame and the emergence of the temperature concept would be much more simpler. Nonetheless, after all the experimentation we encountered with a list of concepts we had assumed but the neural network would need to acquire an understanding of to be able to learn the temperature concept, such as physical bodies, counting, movement, space, axis, and central point, where some of them are local concepts and some others are global concepts.

Therefore, we saw that this task was much harder than we thought and we wanted to try to easen this task for the neural network by adding a new concept: the optical flow.

By using the optical flow we aimed to shorten the path from the images to the temperature value by skipping the step from images to optical flow and regressing the temperature value using the synthetic optical flow we computed by hand.

3.5.1 Optical Flow

Optical flow refers to the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between the observer and the scene (Horn and Schunck, 1981). It is a fundamental concept in computer vision that aims to understand and analyze the motion information present in video sequences or image sequences.

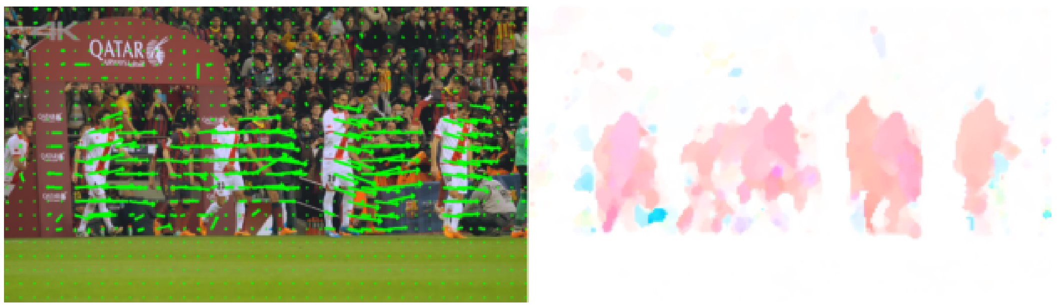


FIGURE 3.17: Example of optical flows (in the left we have the displacement vectors for each pixel, in the right the bounding boxes of the movement of each body.

Optical flow algorithms estimate the displacement vector for each pixel or feature point between consecutive frames in a sequence (see figure 3.17). This displacement vector represents the motion of that pixel or feature point from one frame to another. The optical flow field provides information about the direction and magnitude of motion for each pixel in the image sequence.

Optical flow has numerous applications in computer vision, such as motion tracking, video stabilization, object detection and tracking, action recognition, and 3D scene reconstruction. It is particularly useful in scenarios where understanding and analyzing the motion dynamics of objects or scenes are important.

3.5.2 Dataset

For this experiment we used the first two frames of each sample (already stored), the temperature value for all samples (previously used) and the optical flow (new data), computed from the first two images by hand.

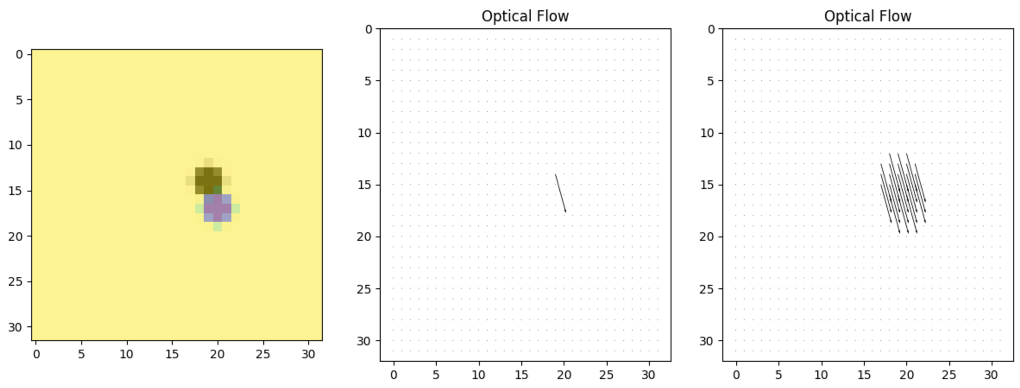


FIGURE 3.18: On the left image we can see the ball on the first frame (black) and on the second instant (purple). The central image is the optical flow only considering the central point and the right image is considering all points in the ball region.

When computing the optical flow we considered two different versions (see figure 3.18): one where we considered movement for all the pixels in the ball region and another version where we considered movement only on the central pixel of a ball.

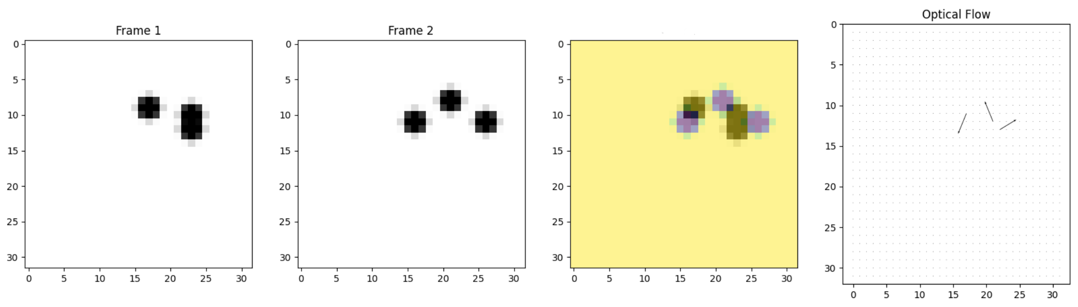


FIGURE 3.19: Sample where occlusion happened between balls. As we can see we still have three vectors. On the third image we have the balls on the first frame (black) and on the second frame (purple).

The first version was only used for images with just one ball (we did this to avoid the problem of assigning two velocity vectors to the same pixel where one ball occluded another, as it can be seen in figure 3.19).

3.5.3 RAFT

To predict the optical flow from the images there are many deep learning models. Nowadays, the state-of-the-art model is RAFT, which stands for "Recurrent, All-Pairs Field Transforms" and was introduced by Teed and Deng, 2020. The architecture of the RAFT model is designed to capture long-range dependencies between pixels in an image sequence while maintaining spatial consistency.

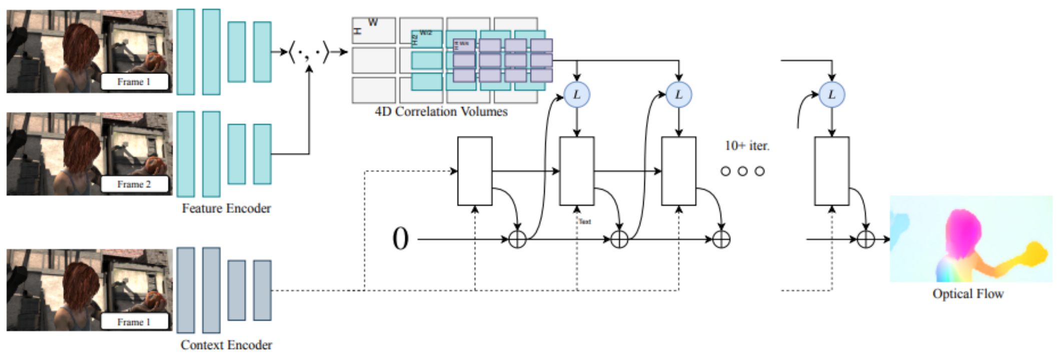


FIGURE 3.20: RAFT architecture (image taken with copyright permission from the journal).

RAFT consists of three main components (see figure 3.20): (1) a feature encoder that extracts per-pixel features from both input images, along with a context encoder that extracts features from only the first image; (2) a correlation layer that produces a 4D correlation volume for all pairs of pixels, with subsequent pooling to produce lower resolution volumes; (3) a recurrent GRU-based (Gate Recurrent Unit-based) update operator that retrieves values from the correlation volumes and iteratively updates a flow field initialized at zero.

- Feature Extraction Network: The first step in the RAFT architecture is to extract feature representations from input images. A convolutional neural network is used as the feature extraction network to encode the appearance and contextual information from the images.
- Correlation Layer: The correlation layer is a key component of the RAFT model, which computes a cost volume by correlating the features from two consecutive frames. This cost volume represents the similarity between pixels in the two frames and helps to estimate the displacement of pixels. The correlation layer is designed to be differentiable, allowing the model to learn the correlation process during training.
- Recurrent Update Module: The recurrent update module is responsible for refining the initial optical flow estimation iteratively. It takes the correlation volume as input and generates displacement vectors for each pixel. The update module is recurrent, meaning it iteratively refines the flow estimation by passing information between iterations. This recurrent design allows the model to capture long-range dependencies and improve the overall accuracy of optical flow estimation.

During training, the RAFT model is optimized to minimize the photometric error between the predicted flow and ground truth flow, since they are outputting the bounding boxes of movement (as in figure 3.17). The model is trained on large-scale datasets with ground truth flow annotations to learn to generalize well to various scenes and motion patterns.

The RAFT model has achieved state-of-the-art performance on several benchmark datasets for optical flow estimation, surpassing previous methods in terms of accuracy and robustness. Its architecture, which combines feature extraction, correlation, and recurrent updates, allows it to capture fine-grained motion details and handle complex motion patterns in image sequences.

3.5.4 Settings and results

On this task we built a very simple fully-connected neural network with 2 hidden layers of size 64 and $2n$, where n was the number of balls per image. The first hidden layer also had ReLU activation.

	1 ball imageset version 1	**1 ball imageset version 1**	**1 ball imageset version 2**
MAE	0.007	0.025	0.16

TABLE 3.2: Table that shows the Mean Absolute Error when regressing the temperature from the optical flow for both 1-ball imagesets (version 1 is considering all points in the ball region and version 2 is with only the central point) and the 3-balls imageset.

In the case of the 1-ball imageset, both versions (all points and only central point) produced great results, with a smaller mean absolute error for the version with optical flow in all points (table 3.2). For the model with 3-balls imagesets we got worse results but still decent.

Consequently, we was able to predict the temperature of the system by using its optical flow on the first two frames. This means that this final task was simple enough, and by simple enough we mean that it did not need too many inductive biases nor concepts to be learned, to be able to extract the temperature concept from the optical flow information.

This result highlights that the initial task was too hard for the model, showing that the potential problem, or one of them, was the amount of concepts it needed to learn to see the temperature concept emerge.

Chapter 4

Conclusions

Despite the efforts, we were not able to regress the temperature from the weights in any neural network we trained to reconstruct the third frame by using the two initial frames. Therefore, we ended up rejecting the initial hypothesis and showing that further research is needed on this area. We have seen that the emergence of concepts, in particular the temperature concept, within neural networks is a complex process that requires numerous inductive biases. These biases are crucial for enabling neural networks to possess foundational knowledge and gain a needed head start in acquiring the desired concept.

By looking at the activations of the bottleneck layer (see figures 3.11 and 3.15) we could think that the velocity was learnt by the model. We could say that it was computing the derivative of position, with the black region being the initial position and the white region the position on the next frame. Therefore, it would seem that it was learning the displacement of the balls in terms of the velocity and its direction (more separate regions would mean higher velocities).

Nonetheless we could not state this due to not being able to regress the temperature nor the coordinates (see sections 3.2 and 3.3). What it may be doing remains unknown and it shows the task was not as simple as we thought, or at least not solved in the way we expect the model to do so.

This has also happened in other studies. For instance, this occured to Lee and Reddish (1981). They explored the behavior of gannets and their relationship with ecological optics. One of the key findings of the study challenged the initial assumption that gannets relied on computing velocity to time the closing of their wings during their dives.

Contrary to this assumption, the research revealed that gannets actually utilized a simpler mechanism than computing the acceleration, distance and velocity to know the exact time to close their wings. This mechanism, that we have already talked about, is the optical flow. Gannets relied on the optical flow information to time the closing of their wings accurately during their dives.

Moreover, although we were able to predict the temperature of the system by using the synthetic optical flow, a similar situation could be happening with the temperature regression starting from the optical flow. One might assume that the transition from optical flow to temperature is as straightforward as counting the number of objects within the system, calculating their velocities, and predicting the average velocity as the temperature. However, it is possible that this is not the case, and the neural network may employ alternative mechanisms to accomplish this task. For instance, numerous models addressing the optical flow problem exhibit intricate patterns and do not solely rely on computing pixel correlations.

Now, let us outline the many concepts we took for granted and explain why they were needed by the neural network to be able to learn the temperature concept.

- Body and particle concepts: The model would need to be able to recognize the different balls as separate bodies that possess distinct identities and interact with each other and the surrounding environment. By recognizing the individuality of each ball, the model could learn their positions, velocities, and trajectories throughout the system, which are individual properties of each ball. This understanding would enable the model to discern the properties that define each ball, such as mass, size, and shape (in our imageset they are the same for all balls).

  By the definition of temperature we can see that the particle concept is essential, as it is actually present on its definition.

- Quantity: Counting the number of balls in the system allows the model to quantify the objects present, which has implications for temperature and movement. The model needs this concept to know how many balls are in the system. If we have more balls in the system, it generally indicates a larger number of particles engaged in motion. As a result, the overall movement and kinetic energy within the system are likely to be higher. Conversely, if the number of balls is reduced, there are fewer particles contributing to the overall motion and energy of the system. In such cases, the average kinetic energy and temperature tend to be lower.

  In summary, counting the number of balls in the system allows you to understand the quantity of particles involved in the motion. More balls generally lead to increased temperature due to a higher level of movement.

- Space / Size of the balls: The size of the space in which the balls are present plays a significant role in understanding temperature emergence and the behavior of the system. The spatial dimensions of the system provide boundaries within which the balls can move and interact. We can interchange space and size of the balls by thinking on the same system but looked from closer (or farther) for bigger (or smaller) balls.

  If the system of balls is contained within a relatively small and confined space the interactions among the balls may be more frequent and intense due to the limited available space. This increased interaction can result in a higher temperature within the system.

  On the other hand, if the system of balls is situated in a larger space, the individual balls may have more freedom to move without encountering other balls as frequently. In this scenario, the overall motion among the balls may be less intense, potentially leading to a lower temperature within the system.

  We could think of a crowded street with tons of people crossing in different directions and now take this same amount of people to a larger space. Now, we would have a smaller temperature because the people (bodies) are more separated across the space.

- Correspondence between balls: The model would need to know what ball corresponds to another in two distinct frames. Without this knowledge we would have too many combinations of possible velocity and coordinates vectors and the model would not be able to determine the correct velocities, and consequently not being able to infer the temperature value. On the colored imageset we used coloring to try to easen the learning of this concept.

- Occlusion: A crucial concept because it addresses the problem of detecting fewer balls when some of them are hidden or partially obstructed from view. Occlusion occurs when one ball obscures another ball, making it difficult to visually perceive and accurately count the entire set of balls in the system, resulting in an incomplete understanding of the overall behavior and properties of the system.

  Occlusion poses challenges to tracking the particles. When a body is occluded, its trajectory and movement cannot be directly observed. In some cases, occluded balls may still have a portion visible, but still poses a difficulty. Detecting and recognizing these partially visible balls requires the model to adapt to varying levels of visibility and develop robust object recognition capabilities.

  Addressing occlusion is crucial for obtaining an accurate count and understanding the complete set of balls within the system. Techniques like extrapolation, prediction, and contextual inference can assist in estimating the presence of occluded balls and adjusting the count and the balls position and velocities accordingly.

- Velocity: Velocity is a vector quantity that describes the speed and direction of an object's motion. The balls have velocities as they move in different directions within the system. In our dataset the speed was the same for all balls in the same sample but the directions were different.

  The velocity is needed to be able to determine the kinetic energy of the balls, and consequently the temperature of the system. The kinetic energy of an object is determined by its mass and the square of its velocity. Therefore, higher velocities correspond to greater kinetic energy, indicating a more vigorous and energetic motion.

- Distance: Distance and velocity are indeed interconnected concepts that can influence each other in systems of balls. Velocity is a measure of the rate at which an object changes its position with respect to time. By multiplying the velocity of a ball by the time elapsed, you can determine the distance traveled by the ball. Thus, the distance covered by a ball is directly related to its velocity and the time interval over which it is measured.

- Smoothness: Smoothness is a concept that relates to the absence of abrupt changes or irregularities in the motion of the balls within a system. Smoothness refers to the regularity and continuity of the paths or trajectories followed by the balls. When the motion of the balls is smooth, it indicates that they move in a predictable and continuous manner without sudden stops, starts, or jerky movements.

  In a smooth system, where energy is conserved, there is less or no energy dissipation, and the average kinetic energy of the balls is more likely to remain constant. This stability in kinetic energy contributes to a more stable temperature within the system. In addition, smoothness allows for easier observation and analysis of the system's behavior. When the motion of the balls is smooth, it becomes simpler to track their trajectories, measure velocities, and identify patterns or regularities in their movement. This facilitates a better understanding of the system's dynamics and temperature characteristics.

It is clear that the emergence of the temperature concept was a undeniably more complex than as we initially thought. From all the experiment we carried out, we was only able to predict the temperature by using the optical flow, but never with the weights of the intermediate layers of the model.

This study shows that deep learning models are not end-to-end models that we can input data to and expect them to solve any problem we present to them. For many challenges, and this specific one, we need to teach them many inductive biases to be able to get the correct output.

Despite the existence of this question for a long time, the research surrounding this topic remains an active area with significant work yet to be undertaken. It has sparked a renewed enthusiasm for investigating its intricacies and uncovering new insights, and some articles have appeared recently (Räz, 2023). Hence, it is evident that the question holds significant interest and continues to captivate researchers in the present context. There is a recognized need to delve deeper into this subject, exploring unexplored avenues and addressing unresolved aspects. Consequently, it is clear that there is ample scope for further investigation and advancements in our understanding of this topic.

Bibliography

1. Bormashenko, Edward (2020). “What Is Temperature? Modern Outlook on the Concept of Temperature”. In: *Entropy* 22.12, p. 1366. DOI: 10.3390/e22121366. URL: https://doi.org/10.3390%2Fe22121366.

2. Coumans, Erwin and Yunfei Bai (2016). “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: *GitHub repository*.

3. Gordon, Deborah (Jan. 2000). *Ants at Work: How an Insect Society Is Organized*. ISBN: 978-0-393-32132-6.

4. Hinton, G. E. and R. R. Salakhutdinov (2006). “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786, pp. 504–507. DOI: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647. URL: https://www.science.org/doi/abs/10.1126/science.1127647.

5. Horn, Berthold K.P. and Brian G. Schunck (1981). “Determining optical flow”. In: *Artificial Intelligence* 17.1, pp. 185–203. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. URL: https://www.sciencedirect.com/science/article/pii/0004370281900242.

6. Lee, David and Paul Reddish (Sept. 1981). “Plummeting Gannets: A Paradigm of Ecological Optics”. In: *Nature* 293. DOI: 10.1038/293293a0.

7. Luo, Shuangling et al. (Dec. 2008). “Toward collective intelligence of online communities: A primitive conceptual model”. In: *Journal of Systems Science and Systems Engineering* 18, pp. 203–221. DOI: 10.1007/s11518-009-5095-0.

8. McCreery, Helen Felicity (2017). “Cooperative Transport in Ants: Emergent Coordination and Collective Problem Solving”. PhD thesis. University of Colorado at Boulder.

9. McGrath, Thomas et al. (2021). “Acquisition of Chess Knowledge in AlphaZero”. In: *CoRR* abs/2111.09259. arXiv: 2111.09259. URL: https://arxiv.org/abs/2111.09259.

10. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597. arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

11. Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

12. Räz, Tim (2023). “Methods for identifying emergent concepts in deep neural networks”. In: *Patterns* 4.6, p. 100761. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2023.100761. URL: https://www.sciencedirect.com/science/article/pii/S266638992300106X.

13. Schmid-Hempel, Paul (Dec. 2002). "B. Hölldobler, E. O. Wilson (1990): "The Ants" Springer, Berlin, 732 pp. DM 198.—". In: *Journal of Evolutionary Biology* 5, pp. 169 –171. DOI: 10.1046/j.1420-9101.1992.5010169.x.

14. Silver, David et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144. DOI: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. URL: https://www.science.org/doi/abs/10.1126/science.aar6404.

15. Sándor Biró, Tamás. (2011). *Is there a temperature? : conceptual challenges at high energy, acceleration and complexity*. eng. Fundamental theories of physics ; 1014. New York: Springer. ISBN: 9781441980403.

16. Teed, Zachary and Jia Deng (2020). "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow". In: *CoRR* abs/2003.12039. arXiv: 2003.12039. URL: https://arxiv.org/abs/2003.12039.

17. Wheeler, William Morton (1911). "The ant-colony as an organism". In: *Journal of Morphology* 22.2, pp. 307–325. DOI: https://doi.org/10.1002/jmor.1050220206. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jmor.1050220206. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jmor.1050220206.

18. Woolley, Anita Williams et al. (2010). "Evidence for a Collective Intelligence Factor in the Performance of Human Groups". In: *Science* 330.6004, pp. 686–688. DOI: 10.1126/science.1193147. eprint: https://www.science.org/doi/pdf/10.1126/science.1193147. URL: https://www.science.org/doi/abs/10.1126/science.1193147.

19. Zhang, Yifei (2018). *A Better Autoencoder for Image: Convolutional Autoencoder*.