Paper reproduction - Physics-as Inverse-Graphics: Unsupervised Physical Parameter Estimation From Video

This blogpost is the result of the reproducibility project conducted by students from TU Delft as part of the course CS4240 Deep Learning (Q3 2024).

Group members of this project:

- Elize Alwash (5016126) - E.Alwash@student.tudelft.nl (mailto:E.Alwash@student.tudelft.nl)
- Valerio Gori (5308739) - V.gori@student.tudelft.nl (mailto:V.gori@student.tudelft.nl)
- Veronika Tajgler (5838444) - V.Tajgler@student.tudelft.nl (mailto:V.Tajgler@student.tudelft.nl)
- Xin Yue Zhang (5117305) - X.Y.Zhang@student.tudelft.nl (mailto:X.Y.Zhang@student.tudelft.nl)

Introduction

In order to control or perform state estimation for physical modelling, we typically rely on physical parameter and system identification where dedicated sensing equiment and carefully constructed experiments are needed. Currently, there exists machine learning approaches that require training by supervised regression from video to object coordinates before being able to estimate explicit physical parameters. The paper "Physics-as-Inverse-Graphics: Unsupervised Physical Parameter Estimation from Video" introduces a model that estimates physical parameters directly from video data without the need for labeled states or objects. This approach is based on vision-as-inverse-graphics encoder-decoder system that allows to render and de-render images using a spatial transformer that allows latent representation of the states (e.g., velocity and position). By using a differentiable physics engine, the model is able to learn the parameters of the system (e.g., spring constant) governed by known differential equations. In other words, the paper provides a solution to unsupervised learning of physical parameters from video with no knowldege of the system states. By integrating the dynamics of the system expressed by the differential equation, the model significantly enhances the ability to perform long-term video predictions and visionbased model-predictive control. The paper is trained on 5 different scenes: two balls bouncing off the image edges, two balls connected by a spring, all on a black background three coloured balls with a gravitational pull and 2 MINST digits connected by a spring on a CIFAR background.

The goal of this blog post is to describe a partial reproduction of this paper and is structured as follows: first the reproduction goal are listed, then the arhictecture of the paper is discussed followed by an overview of implementation used for reproduction. At last a discussion on the results is given.

Reproduction goals

The goal of the project is to reproduce the results from the papers "Unsupervised Physical Parameter Estimation from Video" by Miguel Jaques, et al. The main focuse falls on reproducing the following objectives from the article:

- Movement prediction in Figure 2
- Decoder Masks in Figure 4
- Learned parameter (i.e., spring constant) from Table 1

The reproduction objectives are shown below. Firstly, Figure 1, corresponds to Figure 2 of the paper, and it compares future frame prediction (extrapolation) for the three balls with a gravitational pull. The Figure shows the accuracy of the Physics + InverseGraphics model compared to the true sequence and previoulsy existent architetures.

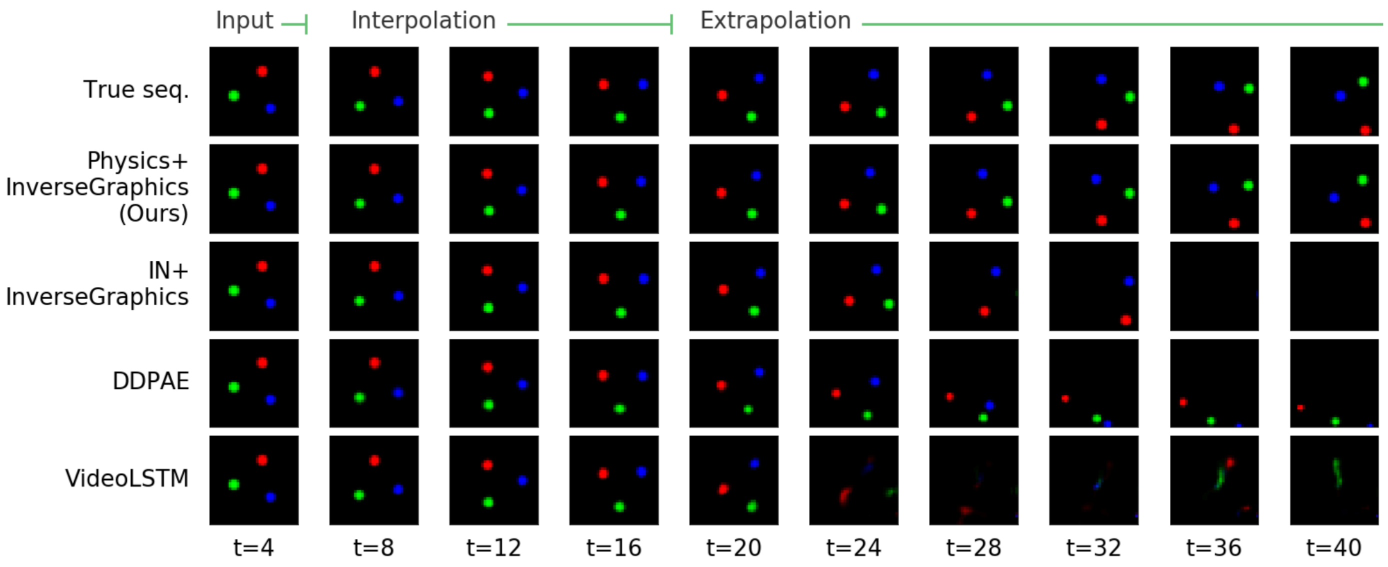


Figure 1: Future frame prediction for 3-ball gravitational system. Adopted from Figure 2 of [1]

Secondly, Figure 4 of [1] shows the content and the masks learned by the decoder. The masks and contents will be further discussed in later section. The image is shown in Figure 2.

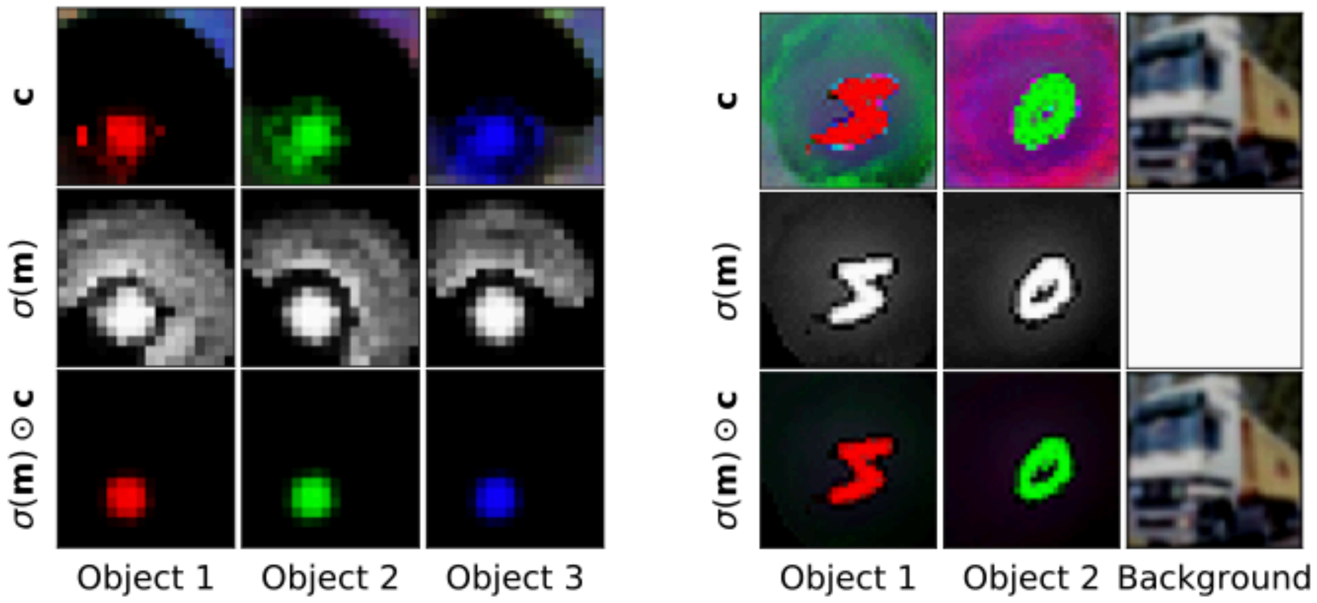
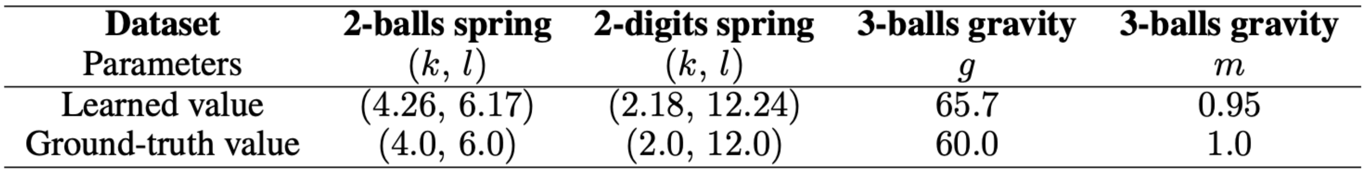


Figure 2: learned contents ©, masks (σ(m)) and their combination used for rendering (σ(m) ⊙c)

Lastly, Table 1 must be reproduced: this is shown in Figure 3. The table compares the real parameters against the learned ones for each dataset.



Dataset	**2-balls spring**	**2-digits spring**	**3-balls gravity**	**3-balls gravity**
Parameters	$(k,\, l)$	$(k,\, l)$	g	m
Learned value	(4.26, 6.17)	(2.18, 12.24)	65.7	0.95
Ground-truth value	(4.0, 6.0)	(2.0, 12.0)	60.0	1.0

Figure 3: physical parameters learned from video

Method

The authors provided the original code used to perform the experiments discussed in the paper. However, due to compatibility issues, it was not possible to use it. The original code was written in Tensorflow 1.x and other outdated packages. Hence, the decision was made to focus on reproduction of the architecture described in paper using pytorch and using the original code for guidance.

Data set creation

Datasets were provided by the authors of the paper. Moreover, the code for generating dataset for any problem investigated by the researchers was given, hence, it was possible to generate a new dataset. For instance, the dataset for the first problem - the three balls with a gravitational pull - is an .npz file containing a multi-dimensional array:

- train_x: This set is for training and contains 5,000 examples. Each example is a 20-frame sequence, with each frame being a 36x36 RGB image (3 channels). Hence, the input for the model is a tensor with size (B, 20, 36, 36, 3), where B is a batch size;
- valid_x: This is the validation set with 500 examples;
- test_x: This is the test set, also with 500 examples.

Training procedure

To train the model, 'train_x' dataset was converted to PyTorch tensor. Furthermore, by utilizing TensorDataset() and DataLoader() with batch size 32 and shuffle, the data was ready for training. The decision was made to use Adam optimizer with fixed learning rate and Mean Squared Error (MSE) loss.

The training loop comprizes of standard steps. For each batch, the model performs a forward pass, after which a loss function is used to compute the error of the prediction compared to the true values. The loss function is as follows:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{pred}} + \alpha\mathcal{L}_{\text{rec}} = \sum_{t=L+1}^{L+T_{\text{pred}}} \mathcal{L}(\hat{I}_t^{\text{pred}}, I_t) + \alpha \sum_{t=1}^{L+T_{\text{pred}}} \mathcal{L}(\tilde{I}_t^{\text{ae}}, I_t)$$

where α is a hyper-parameter. After, backpropagation and parameter update is done, and followed by the validation step.

Architecture

In order to give the reader a complete understanding on the model, this section gives an overview on the general architecture of the model.

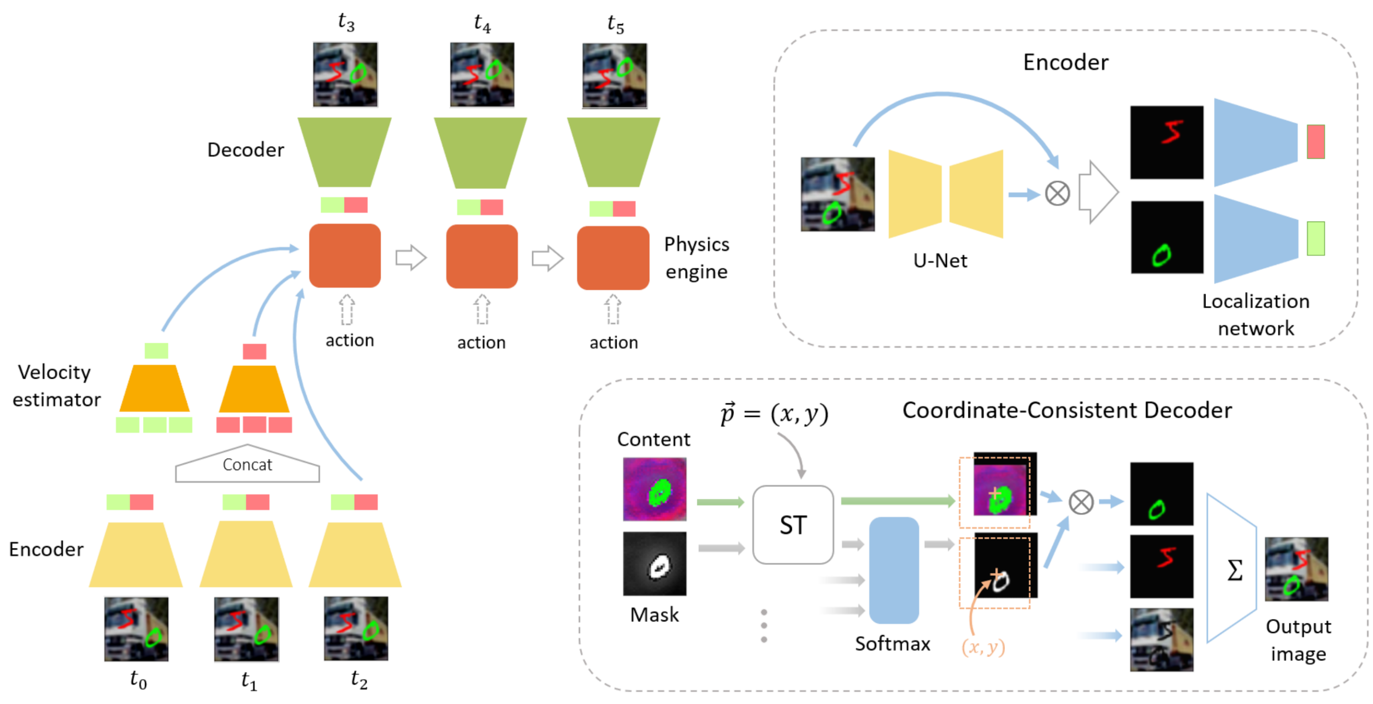


Figure 4: High level view of architecture

The video prediction architecture consists of 4 parts: encoder, velocity estimator, a differentiable phisycs engine, and a decoder. The encoder takes the input images and returns the coordinate in the x-y plane for each object in the input image, next the position is fed into the velocity estimator to compute the velocity at each input frame. The differentiable physics engine is then needed to compute the objects trajectories. Last the, the position data are fed into the decoder which outputs a predicted image. A general overview on the architecture with each specific module is shown in Figure 4.

Encoder

The Encoder net takes a single frame of size *[36 x 36]* pixels as input and outputs a vector containing positions of dimension *[N X D]*, the D-dimensional coordinates of each of N objects in the scene. The object's coordinates are extracted by passing the input frame through a U-Net. The U-Net architecture is used for semantic segmentation, which output these masks representing the object locations.

In our case, since our reproducibility project focused on reproducing the images with the three floating balls, where the image size was of *[36 x 36]*, we opted for using a shallow U-Net instead of the regular U-Net. The shallow U-Net is used when the image size is below *[40 x 40]* pixels. The (shallow) Net produces N unnormalized masks and a learnable background mask, which are then stacked and passed through a softmax to produce *N + 1* normalized masks. The input image is then multiplied by each mask, and a 2-layer location network produces coordinate of size *[batch size x (NxD)]*.

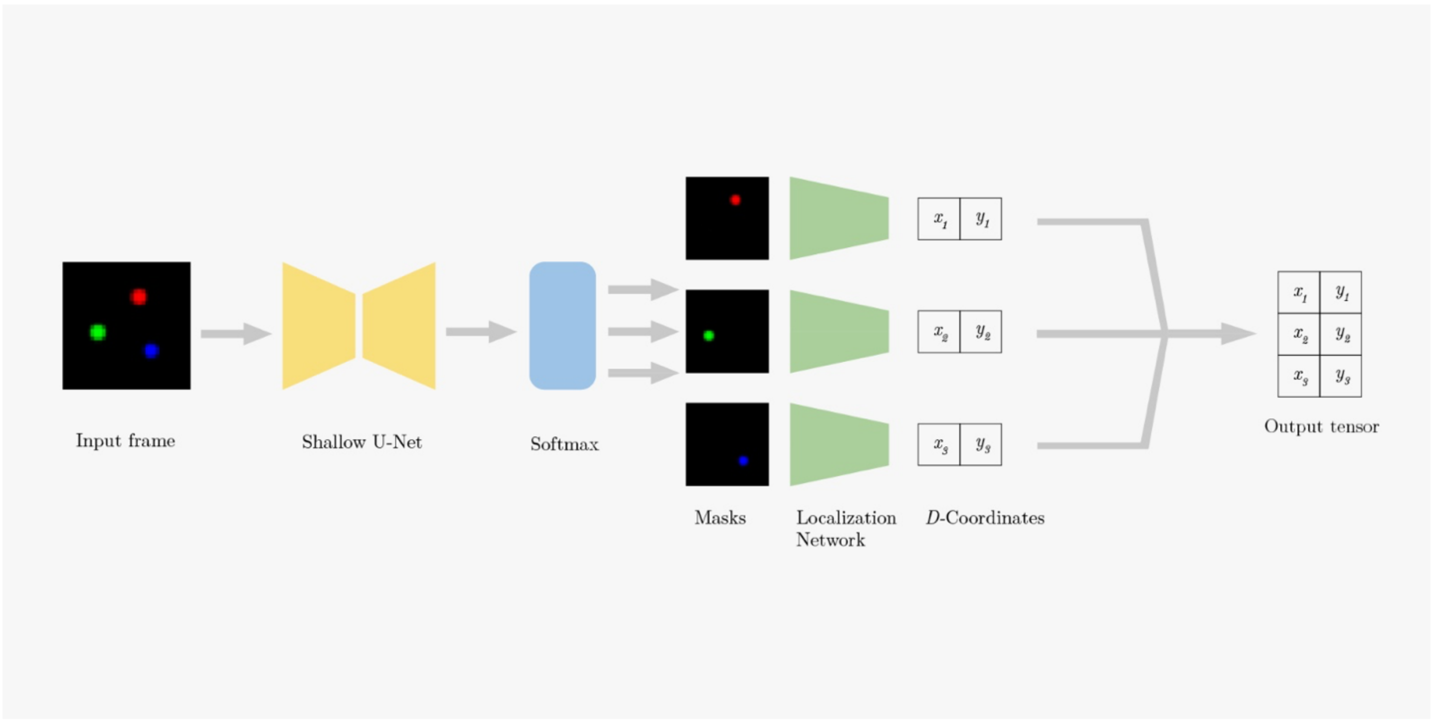


Figure 5: Our graphic depiction of the encoder based on our own PyTorch code

Decoder

The decoder takes positional coordinates from the encoder or physics model, with an input size of *[batch size x (NxD)]*. The first step in the decoder architecture is the initialization of the content and template tensors. This is done via the function "Variable from network" which uses an multi-layer-preceptron to produce tensors of a specified size. The contents and template tensors are initialized at sizes *[N x template size x template size x convolution channels]* and *[N x template size x template size x 1]* respectively. The template size is set at a smaller value than the image size to reudce computational time when fed through the spatial transformer network (STN). Next the tensors are concatenated and fed through to the STN. The decoder makes use of an STN as a separate module. The STN is implemented for spatial invariance, to impose a correct latent coordinate to pixel coordinate correspondence. The STN is implemented as a three stages module: a localisation net (to predict the parameters of the transformation), grid generator (i.e. generates a grid used to sample the input data for transformation), and sampler (extract the pixel values, by applying transformation to the input data based on the generated grid). It follows that the STN returns the transformed version of the input according to the learned parameters of the affine transformation. After the STN, the tensor is split back up into the masks and the contents. The masks created in the decoder are similar to the ones from the U-Net in the encoder. They are segmented maps that isolate a single object. Next the backgrounds are initialized. The background tensor for the contents has the same size as the original input image, *[batch size x N x image size x image size x channels]*, this is to make sure no information is lost. However the background mask is initialized at the same size as the mask itself; *[N x template size x template size x 3]*. Before the next step, all images with template size are padded to match the image size to ensure feasibility of the tensor multiplication. Finally the masks are multiplied with the matching content tensors (per object). The products yields the tensors on the left side of Figure 6 which highlight a single object e.g., a single ball. The same is done for the background tensor, which creates this tensor containing background information without the objects. Lastly all the tensors are summed to create one complete image of *[size batch x N x image size x image size x channels]*, representing the next frame in the sequence.

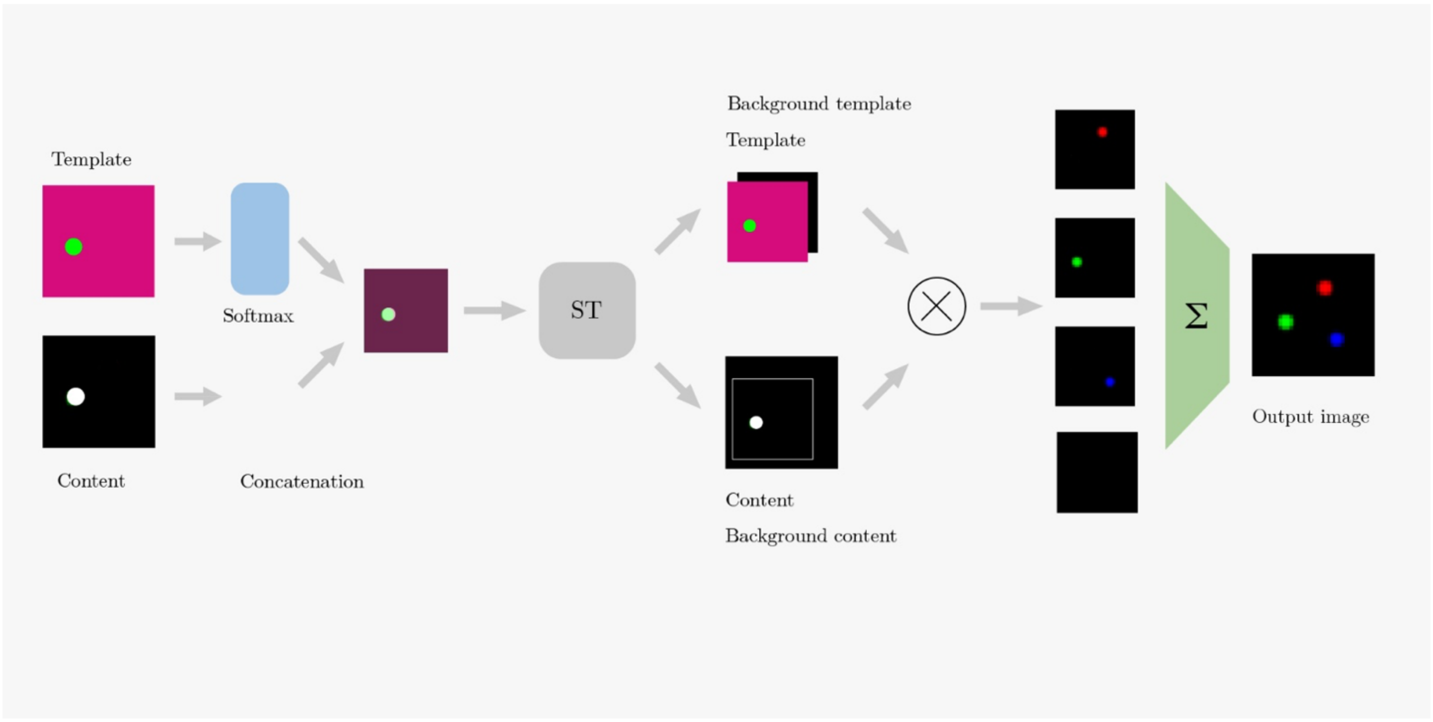
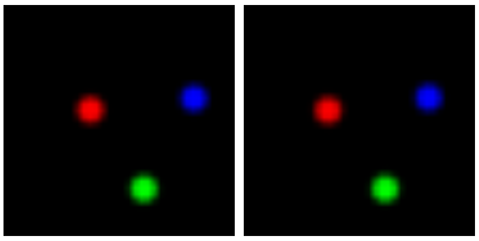


Figure 6: Our graphic depiction of the decoder based on our own PyTorch code

Results

Reimplementation of the submodels from Tensorflow 1.x into a code variant written using Pytorch. Discrepancies between the paper and the code of the authors were found, specifically between their high level view of their architecture. Hence, we produced new diagrams (Figures 5 and 6) of the Encoder and Decoder specifically, which are a proper representation of the architecture of the code.



Conclusion

During the reproduction process we gained a better insight into the working of multiple topics discussed in the field of deep learning. Some being convolutional encoders and decorders, transformers and semantic segmentation. Our main contribution is the reimplementation of the submodules in PyTorch instead of TensorFlow, which was the original library used by the authors of the paper. By re-writing the code from scratch we gained a deeper understanding of the working of each submodule. The encoder and decoder seem to correctly work together, outputting position coordinates and a reconstructed image respectively. The velocity estimator and physcis models have also been rewritten in PyTorch. However, we are unsure about their accuracy. Additionally, due to time constraints the submodules have not yet been trained together either.

Moreover, the paper is challenging to reproduce, not only due to the incompatible code, but also due to the general complexity of the architecture, and the misalignment between the code and the paper. In other words, the code often included functions that are not discussed in the paper, while other have a poor explanation in the paper.

As a recommendation for future reproduction, an understanding of both TensorFlow and PyTorch would be favourable. Additionally more time would be desirable to completely understand and reproduce the code correctly.

Work distribution

- Elize Alwash: Poster writing, Blog post writing, Debugging, Implementation of the Encoder and Decoder
- Valerio Gori: Poster writing, Blog post writing, Implementation of the Spatial Transformer and Decoder, Debugging
- Veronika Tajgler: Implementation of all of the submodules and combining them into a working auto-encoder, Debugging, Poster writing, Blog post writing
- Xin Yue Zhang: Blog post writing, Debugging, Implementation of the Shallow U-net and Decoder, Illustrating

References

[1] Jaques, M., Burke, M., & Hospedales, T. (2020). PHYSICS-AS-INVERSE-GRAPHICS: UNSUPERVISED PHYSICAL PARAMETER ESTIMATION FROM VIDEO. 8th International Conference on Learning Representations, ICLR 2020.