Self-Supervised Learning of Task-Specific Physical Representations from Dynamic Interactions

Namrata Deka
Department of Computer Science
University of British Columbia

Abstract—**Human beings interact with and manipulate objects everyday in their lives, and most of these interactions are carried out to achieve some pre-determined goal. These goals dictate the dynamic of the interactions as well as the knowledge of physical properties of the objects that are required to carry out the interactions. In this work we explore the possibility of learning task-specific physical representations of objects that are better suited for a robotic agent to complete the task. We propose TaskPhysNet, a self-supervised deep-learning framework that can work with reward-based objectives to learn the required physical representations. We specifically look at the task of navigating through clutter by pushing objects aside to reach a fixed goal. Our experiments show that learning task-specific physical representations is beneficial for efficient completion of the task at hand compared to traditional return maximization without considering the physical properties of object interactions.**

I. Introduction

Understanding physics is important for manipulating objects in the real world. Knowledge of physical properties such as mass, coefficient of friction, center of mass, and various material properties is crucial for efficient and accurate control of physically-interactive robotic agents. However, not all of these properties need be known all the time to function in the world. For example, mass and coefficient of friction are essential properties to know when the task is to push an object. However, when stacking objects vertically, knowledge of center-of-mass becomes more important than mass and knowledge of coefficient of friction may not even be required. In this work we explore the possibility of enabling an artificial agent to learn such task-specific physical properties for better planning and manipulation strategies.

Specifically, we concentrate on the task of navigation through clutter where:

- The goal of an agent is to reach a fixed target destination.
- If the agent is surrounded by obstacles, it must push them aside to clear a path to the target.
- When deciding which obstacle to push, it must learn to choose the ones with less weight.

The framework we propose is also generalizable to other tasks that may require knowledge of a different set of physical properties. Our framework TaskPhysNet **combines deep reinforcement learning with a self-supervised auxiliary objective** that incentivizes the learning of dense representations that capture the physical properties that are required for the task. The auxiliary objective along with the objective to maximise a reward function enables the learning of only those physical properties that are relevant to the task and can yield high returns.

TaskPhysNet takes as input the current state and visual observation of the system, and predicts the action distribution from which to sample an action to perform as well as an optical flow image depicting how the observed visual input will change on applying the sampled action. Since, optical flow can be computed automatically given the images before and after the action was applied, training of TaskPhysNet can take place without needing any manually obtained annotations. Figure 1 shows an overview of our proposed deep architecture.

The main contributions of this work are:

- A self-supervised deep learning based framework to learn task-specific object physical representations.
- A novel environment to train an agent to push through clutter, called Push-Nav.

We also propose three different metrics to evaluate different aspects of TaskPhysNet. Section IV, where we show that TaskPhysNet is able to outperform a baseline that does not learn dense representations to encode physical properties.

We present a comprehensive detail of our method, environment and training procedure in section III. Python implementation for this work is available here.

II. Related Work

Learning object physics is not a novel problem and researchers in the past have explored various methods to model object physics. Early approaches to modelling physical properties were to use explicit physics models that were parameterised by mass and force. Atkeson et al. [1] estimated the mass and moment of inertia of grasped objects using a robotic wrist's force-torque sensors. Similarly, instead of grasping, Yu et al. [17] used fingertips equipped with force-torque sensors to estimate inertial parameters by pushing. More recently, Wu et al. in [13, 14] used deep features to estimate physical properties of objects from their appearance and motion using physical simulation engines. A number of approaches based on supervised end-to-end deep learning also exist [5, 6, 12, 16]. All of these approaches build upon explicitly defined physical models which limits their practical usage. Also, supervised end-to-end deep learning approaches are limited by the amount of human-annotated data required for training.

Alternatively, self-supervised learning has become popular to learn latent representations without explicit supervision

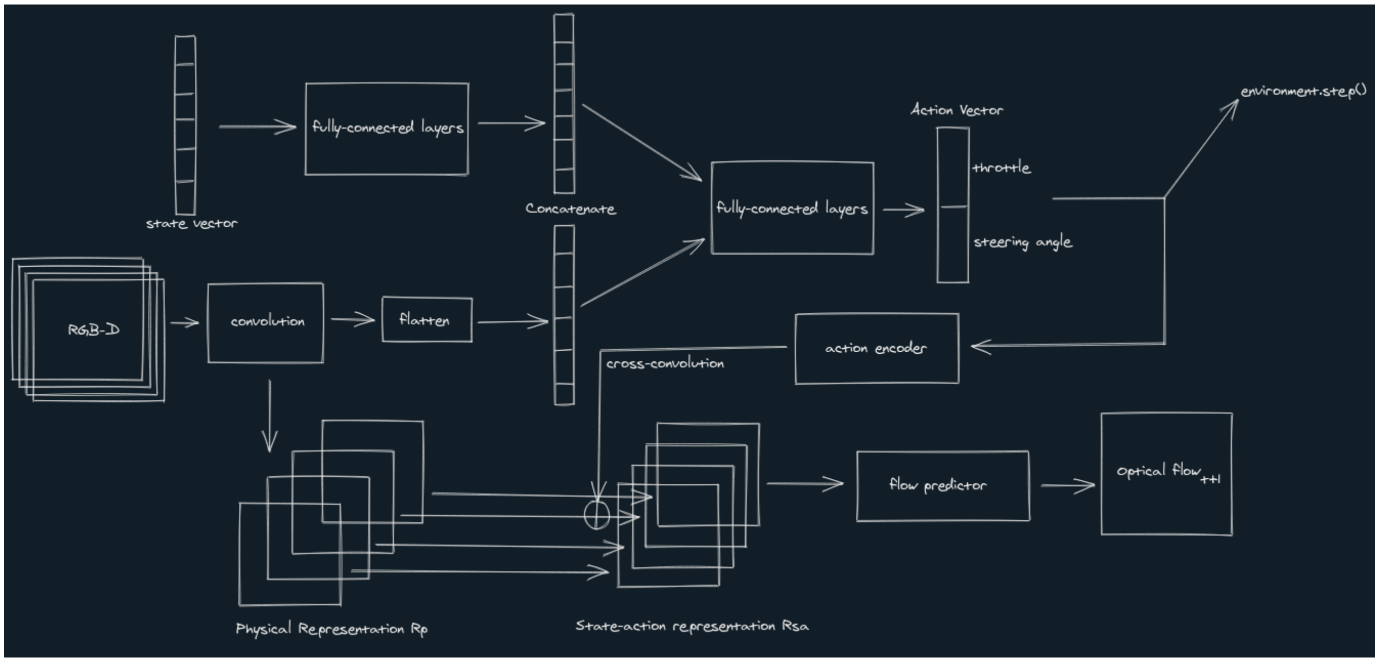

Fig. 1. **Architecture overview of TaskPhysNet**. The observed RGB-D image is fed through convolution layers to compute the physical representations R_p, which are flattened to create the physical feature vector. Simulataneously, the state feature vector is obtained from the input state vector. Both feature vectors are concatenated and further processed to get the action a_t. An action encoder computes the action kernels from a_t which are cross-convolved with R_p to output dense state-action representations. These are the inputs to a flow predictor network which estimated the optical flow between the current image and the next image that is observed.

and use of physics models. Byravan and and Fox [2] used deep networks to approximate rigid body motions. Pinto et al. [10] proposed a curiosity-driven framework to interact with objects and learn visual representations. Denil et al. [4] used reinforcement learning to select the best set of experiments to perform to build object models, and Zheng at al [18] used learnt object representations to decode exact physical property values. But all of these approaches learn representations that capture visual properties of objects which are not as advantageous as knowing the physical properties needed for interactions.

Push-net proposed by Li et al. [8] learns physical object properties required for planar pushing. It is a supervised learning method that learns to push objects to a target location. Push-net learns a single latent vector for the entire input image and therefore does not have the ability to learn object-centric properties and generalize to multi-object images. Xu et al. addressed this by introducing DensePhysNet [15]. By sliding and pushing objects they learn a pixel-wise dense representation of each object that encodes their mass and coefficient of friction. However, their strategy to explore the interaction space is random and not tailored to the physical properties to be learned. They also do not utilize the color information of the observed scene and miss out on material properties which can be decoded from color, lighting and shading information. Our method, TaskPhysNet addresses both of these issues by using RGB-D signals to learn task-specific properties while simultaneously learning action policies that are beneficial for the task at hand. Another approach proposed by Kannabiran et al. [7] that is close to our setting learns optimal policies to explore the interaction space to visit the most informative states with respect to the physical property to learn. However, their method requires manually annotated values of mass for each object the agent interacts with which is not always feasible. Similar to DensePhysNet, our approach can be trained in a self-supervised manner making it more practical and feasible to use.

III. Method

The goal of TaskPhysNet is to learn task-specific physical representations of objects through self-supervision while learning a policy to perform the said task. To achieve this we train a deep predictive model on RGB-D images to estimate an optical flow image and action simultaneously. The optical flow represents the estimated motion per pixel of the image when the predicted action is performed. Similar to DensePhysNet [15], the idea is that to accurately estimate the action-conditioned optical flow, the deep network must acquire an understanding of the physical properties that govern the motion of objects. TaskPhysNet also takes as input the current state of the agent represented as a 17-dimensional vector. Information from both the state and observation are fused to predict the next action. In the following subsections we will provide details of all the different aspects of our method.

A. The Environment

Figure 2 shows a snapshot of our training environment for the task of pushing through clutter to reach a target goal. We

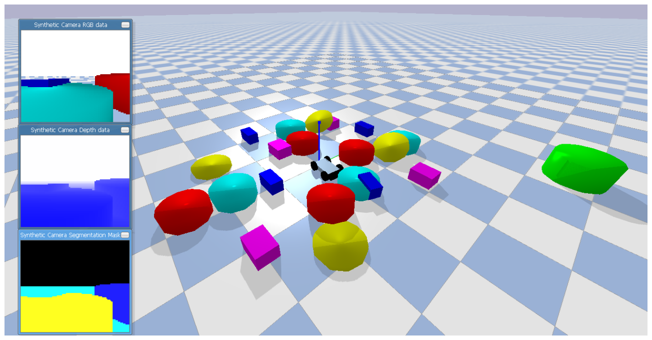
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask

Fig. 2. A snapshot of our Push-Nav environment. Our agent is a gray vehicle with a black head-board that senses head-on collisions. Obstacles are color-coded objects of varying shapes, sizes and masses. Target location to reach is marked by a fixed green cylinder outside the clutter of obstacles.

are calling this environment Push-Nav. The agent is a vehicle equipped with a force-torque sensor to measure the impact of head-on collisions. It can also sense when any part of its base body comes in contact with an obstacle. The obstacles are objects of varying shape, size, color and mass. The dimensions and mass of each object is color-coded which means that all objects with the same color will have the same mass and same size. On initializing the environment, obstacles are randomly placed in a circle around the vehicle so as to create the clutter. The target destination is marked by a green cylinder of fixed dimensions and is always placed in a random position outside the circle of clutter.

1) State Space: The state s_t of the agent at time t is represented as a 17-dimensional vector. It consists of the locations $x, x_g \in \mathbb{R}^2$ of the vehicle and the target on the X-Y plane, orientation θ as a two-dimensional vector $[\sin\theta, \cos\theta]$, velocity $\dot{x} \in \mathbb{R}^2$ of the vehicle, total collision force vector $F_{base} \in \mathbb{R}^3$ detected by the base link of the vehicle, and head-on collision force and torque $F_{head}, \tau_{head} \in \mathbb{R}^3$ sensed by the force-torque sensor. A depth sensor mounted on the vehicle captures RGB-D images I_t at each time step with a resolution of 84x84 pixels.

2) Action Space: The action a_t at time t is a two-dimensional vector containing the throttle and steering angle values to control the motion of the vehicle.

B. The Reward

For the goals of Push-Nav we design a reward function that encourages navigation towards the target and discourages collision forces of large magnitudes that are required to push heavy objects. To that end, the total reward r_t at time step t is a linear combination of a distance reward r_t^d and a push penalty r_t^f which are defined as follows:

$$r_t^d = \max(0, \delta_{t-1} - \delta_t) \tag{1}$$

where, δ_t is distance to the goal at time step t.

$$r_t^f = -\|F_{head}\|_2 \tag{2}$$

The total reward at time t is:

$$r_t = w_d r_t^d + w_f r_t^f \tag{3}$$

C. TaskPhysNet

TaskPhysNet, as shown in Figure 1, consists of five different modules: a state encoder, a physics encoder, an action predictor, an action encoder and a flow predictor. At each time step t, the state vector s_t is fed into the state encoder to obtain state features f_s. At the same time, the observed image I_t is fed into the physics encoder which outputs physical representations R_p. R_p is convolved using 2D convolution operations and flattened to create physical features f_p. The state and physical features f_s and f_p are concatenated and passed through the action predictor to get the action a_t, which results in the next state and observation s_{t+1} and I_{t+1}. The transaction (I_t, a_t, I_{t+1}) is stored for the self-supervised optimization process later on.

Once the action a_t is estimated, it is fed to the action encoder and reshaped to give us a set of action kernels a_t^k. Like in DensePhysNet [15], these kernels are applied to the physical representations R_p. This outputs state-action representations R_{sa}. This way, the network is incentivized to learn physical representations R_p that can encode the required object physics to estimate the effect of the predicted action on them. Finally, the state-action representation R_{sa} is fed to the flow predictor module to predict the dense optical flow image $O_{t,t+1}$ between the current image I_t and next image I_{t+1}.

D. Training

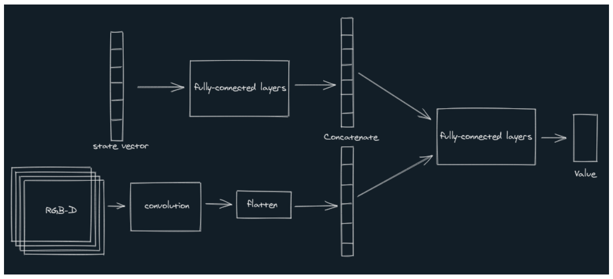

Fig. 3. Critic network architecture.

We set up the learning task as a reinforcement learning problem using the proximal policy optimization(PP0) algorithm [11]. The network defined in section III-C serves as the actor, and the critic is a simpler network as shown in Figure 3. At each iteration we collect a dataset of $(s_t, I_t, a_t, r_t, s_{t+1}, I_{t+1})$ transactions using the current policy. We then compute the ground truth optical flow image $O^*_{t,t+1}$ using images (I_t, I_{t+1}) from the data collected. The estimated optical flow $O_{t,t+1}$ is computed using I_t and a_t as defined in III-C. The self-supervised mean squared-error between $O^*_{t,t+1}$ and $O_{t,t+1}$ is added to the PPO actor loss. This combined objective to both maximise the reward as well as minimize the optical flow error encourages the network to learn physical representations R_p that capture the object physics beneficial for the task of Push-Nav. This learning framework can be generalised to other tasks by simply changing the environment and modifying the action output in the actor network.

E. *Architecture Details*

Here we provide more granular details about each module in our proposed network. The state encoder takes as input a 17-dimensional state vector and passes it through two fully connected layers with 256 and 512 hidden neurons respectively with ReLU activation functions.

The physical encoder takes as input a 4-channel 84x84 RGB-D image and feeds it to two convolutional layers with 16 and 32 feature maps using one 5x5 kernel and one 3x3 kernel. Each convolution is followed by a batch normalization and ReLU activation. The physical representation obtained is convolved further using four 2D convolutions layers containing 64, 128, 256 and 512 feature maps with ReLU layers and flattened to form the physical feature vector f_p.

The action predictor is a fully connected network with one hidden layer of 512 units with a ReLU activation. It expects a concatenated input of 1024 dimensions.

The action encoder consists of seven fully connected layers with 64, 128, 256, 256, 256, 512 and 800 hidden neurons. The output is reshaped to create 32 kernels, each of size 5x5.

Finally, the flow predictor takes as input the 32-channel state-action representation and passes it through four 2D convolution layers with 32, 32, 32 and 2 channels respectively. There are batch normalization and ReLU activations between every layer.

IV. Experiments

A. *Implementation Details*

We develop and train our framework using Python and PyTorch [9]. The simulation is done using PyBullet which is a Python wrapper over the Bullet [3] physics engine. Both the actor and critic networks are updated using Adam optimizers with learning rates of 2.5e-6 and 1e-4 respectively. In each iteration the policy and value functions are updated for four epochs using minibatches of size 128. Each action is applied for four time steps before querying the policy for the next action to increase the stability of the artificial agent's motion. To encourage exploration of the action space we also add the entropy of the estimated action distribution to the actor loss.

B. *Evaluation*

There are several aspects to TaskPhysNet that can be evaluated, namely:

- How does the resulting policy compare to a baseline which does not learn physical properties of objects?
- Is the learned policy able to perform the given task correctly?
- Do the learned physical representations encode properties specific to the given task?

1) *Comparison with a Baseline:* We train a baseline actor-critic model which does not use self-supervision to learn any physical property of objects. The baseline architecture differs from TaskPhysNet only with respect to the actor network where convolutional features of the input RGB-D image are directly flattened and sent to the action predictor along with the state vector features. This is similar to the architecture in Figure 3 with the exception of the last layer which outputs a 2-dimensional action vector. The baseline is trained using standard PPO objectives without any self-supervised losses.

We evaluate the total rewards collected by both the baseline and TaskPhysNet policies over one hundred random episodes. The baseline policy is able to collect an average of 39.06 units per episode, and TaskPhysNet was able to collect an average of 47.05 units. i.e. almost 10 units higher than the baseline.

2) *Correctness of the Policy:* The desired behavior for the agent when pushing obstacles is that it should choose to push lighter objects over heavier ones. To evaluate this we create a simple environment where the path to the target is a straight line blocked by the different obstacles placed in a line perpendicular to the shortest path. Figure 4 shows the first-person view of the trajectory of the vehicle in such a case. The yellow cylinder is the lightest obstacle followed by red, blue, sky, and purple. The baseline policy decides to turn right towards the sky object to reach the goal which is incorrect as per the weight ordering. The policy resulting from TaskPhysNet correctly moves towards the yellow object as desired.

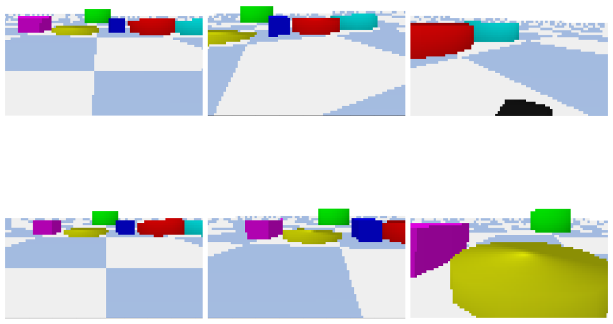

Fig. 4. These images are observed by the agent and they show the trajectory it takes to reach the target location. **top row:** baseline. **bottom row:** TaskPhysNet.

3) *Decoding Task-Specific Object Physics:* To evaluate if TaskPhysNet learned representations for the relevant physical properties we propose to explicitly decoder various property values from the learnt physical representations R_p. Due to time and resource constraints, this metric was not implemented but we would like to outline the plan for it as follows:

- Extract the 32-dimensional physical property encoding from the learnt dense physical representation R_p.
- Train separate linear classifiers using these features for various physical properties like mass, coefficient of friction, center of mass, rigidity, etc.
- Examine the resulting classification metrics, and check whether the accuracies for the properties relevant to Push-Nav (like mass and coefficient of friction) are higher than the accuracies for other physical properties.

The intuition behind this criterion being that task-specific physical representations should be unable to help infer non-relevant physical properties.

V. Conclusion

In conclusion, in this work we presented a self-supervised framework to learn physical object properties that are relevant to a given task. We also presented a novel environment built in PyBullet with the objective to push through clutter to reach a target location. We argued that knowledge of goal-oriented physical properties of objects is important to learn effective state-action policies to perform the task. Finally, we tested our proposed framework and hypothesis using two metrics, and proposed a third metric to be implemented in the future. Also, the evaluation comparing the collected rewards with the baseline could be strengthened further by performing statistical significance tests to prove or disprove the hypothesis that learning task-specific physical representations lead to better policies.

References

[1] Christopher G Atkeson, Chae H An, and John M Hollerbach. Estimation of inertial parameters of manipulator loads and links. *The International Journal of Robotics Research*, 5(3):101–119, 1986.

[2] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE, 2017.

[3] Erwin Coumanns. Bullet physics library, 2012. *URL: http://bulletphysics. org*, 2(8).

[4] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando De Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*, 2016.

[5] Sebastien Ehrhardt, Aron Monszpart, Niloy J Mitra, and Andrea Vedaldi. Taking visual motion prediction to new heightfields. *Computer Vision and Image Understanding*, 181:14–25, 2019.

[6] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[7] Niranjan Kumar Kannabiran, Irfan Essa, and C Karen Liu. Estimating mass distribution of articulated objects through physical interaction. *arXiv preprint arXiv:1907.03964*, 2019.

[8] Jue Kun Li, Wee Sun Lee, and David Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems*, volume 14, pages 1–9, 2018.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-li pdf.

[10] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

[13] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in neural information processing systems*, 28:127–135, 2015.

[14] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

[15] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019.

[16] Tian Ye, Xiaolong Wang, James Davidson, and Abhinav Gupta. Interpretable intuitive physics model. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018.

[17] Yong Yu, Tetsu Arima, and Showzow Tsujio. Estimation of object inertia parameters on robot pushing operation. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1657–1662. IEEE, 2005.

[18] David Zheng, Vinson Luo, Jiajun Wu, and Joshua B Tenenbaum. Unsupervised learning of latent physical properties using perception-prediction networks. *arXiv preprint arXiv:1807.09244*, 2018.