Using Foundational Segmentation Models for Learning Atari Game Physics

Colin De Vlieghere, Ben Edidin, Jasper Gerigk

Abstract

Data efficient Reinforcement Learning (RL) aims to achieve the superhuman performance that RL is known for while using significantly less data [1]. While humans usually view scenes as a combination of objects, RL approaches rely on a CNN backbone to extract the key information from the scene. Explicitly modeling objects could significantly speed up the learning behavior of RL agents as it removes the requirement for the CNN to learn to identify objects [2]. We investigate whether foundational segmentation models [3] [4] can be used to extract the relevant objects' information from a game, and if agents can benefit from this information. Rather than training an entire RL agent with this additional information, which is infeasible given our constraints, we focus on predicting the future rollout of the game by learning a transition model capable of long-term predictions of the scene. This task is central for an agent to develop effective plans and play the game well as seen in prior work [1]. The prediction model is composed of a feature extractor and recurrent predictor, which predicts the locations of the objects given by the masks in future frames. We demonstrate how modeling object interactions via a transformer is vital for performance, when such interactions occur in the environment. Atari games have been a key benchmark for the progress in RL and are thus the focus of our work. The code is available at https://github.com/Cubevoid/atari-obj-pred.

Background & Related Work

Reinforcement learning approaches can be categorized into model-free and model-based ones [5]. Model-free approaches do not explicitly learn a model of the environment and attempt to learn the optimal policy directly, while model-based approaches learn an explicit model of the environment which is used to help learn the optimal policy. Simulated Policy Learning (SimPLe) was an early model-based RL approach for Atari, which separated the learning process into three distinct steps, sampling from the game using the current policy, updating the world-model using self-supervised learning, and improving the policy using the current model [5]. The world-model is trained by predicting the next frame and the reward. A drawback of such an approach is that learning to recreate the entire frame requires learning irrelevant information such as background features. While model-free approaches do not explicitly learn a model, "Data-Efficient Reinforcement Learning with Self-Predictive Representations" (SPR) augments the learning process by predicting the agent's future latent space [1]. Building on the standard Deep Q Network (DQN) model, SPR takes as input the current state, encodes it into a latent space, and in addition to determining the action, uses a CNN transition model to predict the next states in the latent space to encourage the model to learn how the environment behaves.

Segment Anything is a multi-modal segmentation model created by Meta capable of predicting likely masks with or without a user's prompt [3]. It consists of transformer encoders for the image and prompt, which allows it to support prompts in the form of text, points, bounding boxes or masks, followed by a transformer decoder to create the masks. The model was trained on 11M images with over 1B corresponding masks. However the model was only trained on real-world images, and as we demonstrate it struggles with segmenting images from 2D games. Fast Segment Anything (FastSAM) is an alternative segmentation-model trained using a smaller subsample of the Segment Anything dataset [4]. Rather than using a Vision Transformer, it is based on the YOLOv8-seg object detector which uses a CNN, and can be up to 50x faster on inference.

Methodology

Our work is split into two parts, the first investigating the suitability of foundational segmentation models for extracting relevant objects from Atari and using them to collect a dataset for the second task, which is training a model to predict the objects long-term movements.

Atari

We used the OCAtari interface for the Atari games [2] to get the ground-truth object locations of the objects in the game alongside the normal observation, consisting of the frame. OCAtari also provided pre-trained DQN agents, which we were able to run to collect data from a strong policy. This means we got a good distribution of different game states compared to using a random agent. OCAtari provides a variety of algorithms to extract the objects [2]. We used the "revised" algorithm, which uses information from the RAM to determine the location. In practice, this worked relatively well, especially when the game was already underway for some timesteps. Sometimes, objects were returned which were not actually present with a stated position in the top left corner, so we filtered these objects out. Further bounding boxes were not always accurate as they were hard coded in the OCAtari code and for example changing width of numbers was not accounted for. Nevertheless, the locations were quite accurate overall.

After getting the segmentation masks from a foundational model, we did greedy matching to match the found segmentation masks to the ground truth object masks based on the detected size and positions. This was required since the segmentation models would often detect background objects as actual objects, as they did not understand the context.

SAM vs FastSAM

We evaluated the SAM and FastSAM foundational segmentation models on the Atari data to collect the masks for the current objects. Overall, we found that SAM and FastSAM have similar performance across a variety of games but FastSAM was significantly quicker. We compared the SAM and FastSAM models by comparing the detected objects for a set of sample frames. To compare the quality, we evaluated both the number of objects detected and the IoU between the two models' bounding boxes (averaged over the objects). We observed that SAM produced slightly visually better masks but was too slow to run for the entire dataset of 100k steps, so we wanted to see if we could use FastSAM instead.

In Berzerk, as seen in Figure 1, the number of objects per frame varies more because many characters appear and disappear at random. However, both models qualitatively do well on detecting the actual characters in the game, and the IoU metric supports this observation. An example of good results is seen in Figure 3. Both models also had issues with some walls being misidentified as the agents.

The models performed better for Pong and Freeway, with high IoUs as seen in Figure 2. The ground-truth number of objects for Pong is five but both models sometimes miss the ball, such as when it touches a paddle and the model detects both objects as one.

We also tried other games like Assault, DemonAttack, Seaquest (Figure 3), and Skiing, but both models reliably failed at either detecting all important objects in the scene and only detected the background, suggesting that some future work must be done on improving generalization performance of these models on Atari games. This also highlights the fact that the inductive biases which exist in real-world data do not map to Atari games, which is not something that hinders human performance. While we are able to get somewhat good masks on Berzerk, they are not consistent over time enough to allow good prediction. Thus, we selected Pong and Freeway to develop our prediction models for.

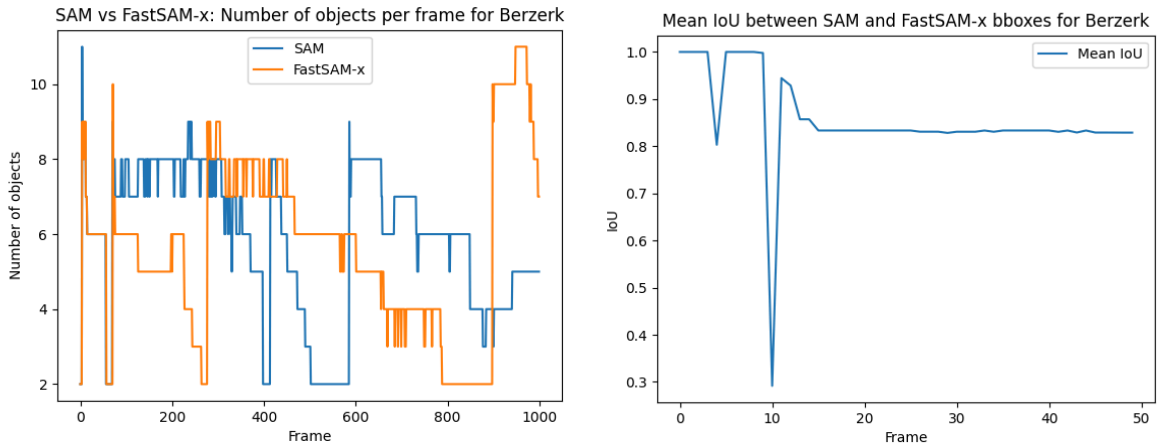

Figure 1: Number of objects per frame and IoU between SAM and FastSAM for Berzerk

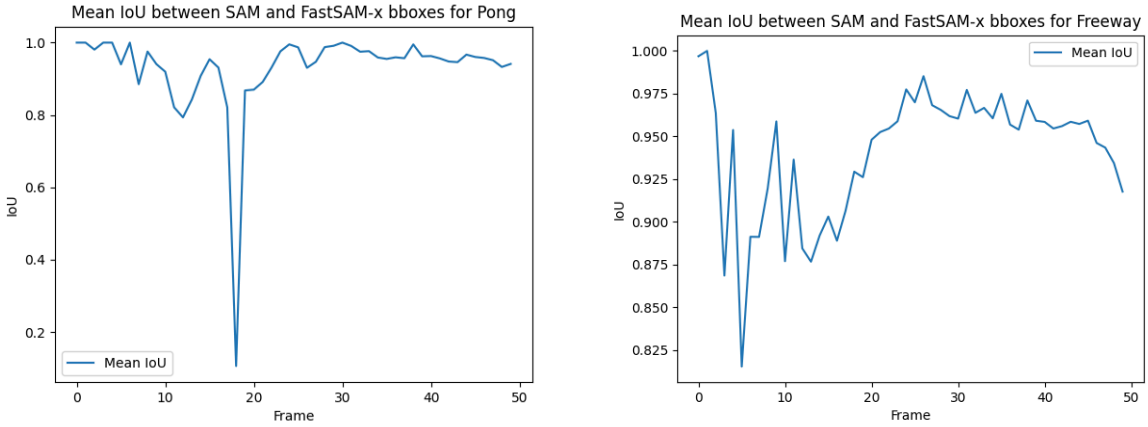

Figure 2: IoU between SAM and FastSAM for Pong and Freeway

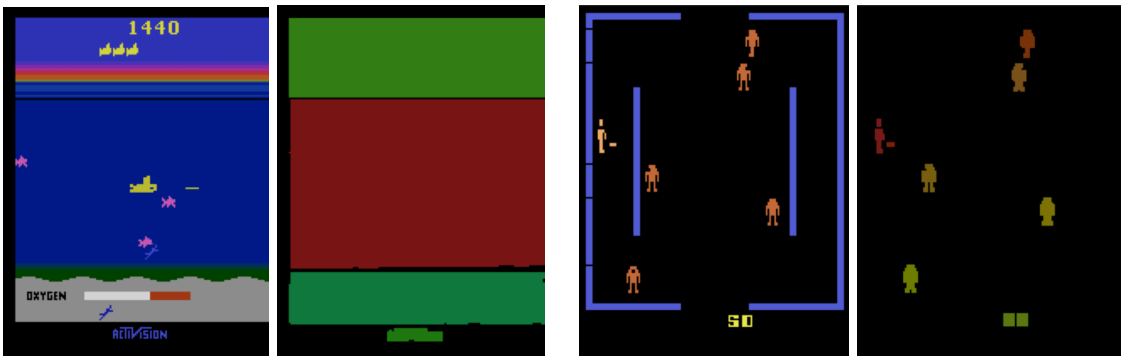

Figure 3: Game Frame vs. SAM Masks. Left: Seaquest. Right: Berzerk

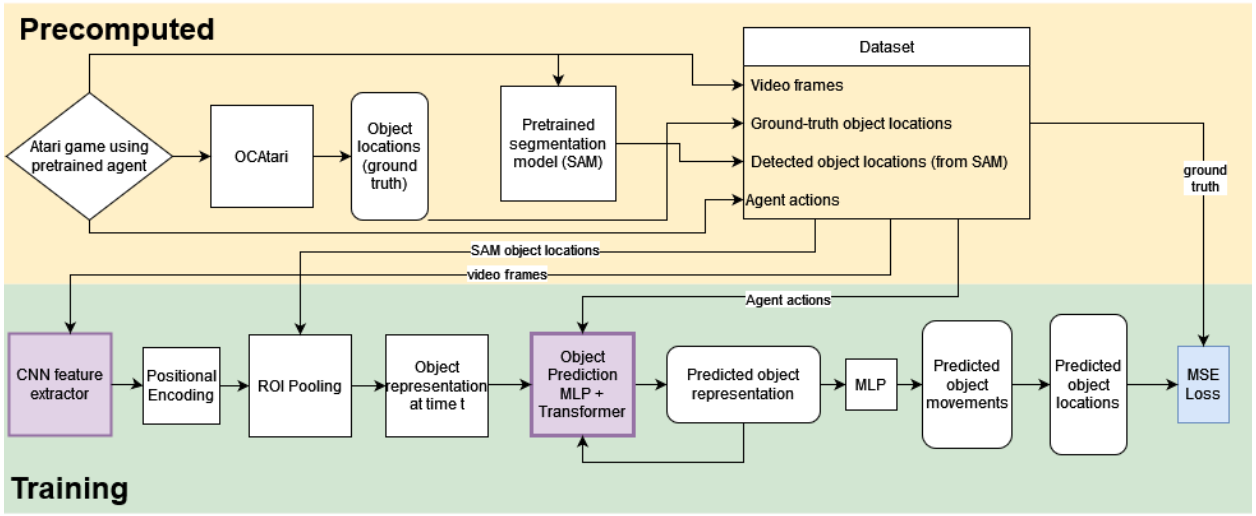

Figure 4: Architecture diagram of our model

Models

Our prediction model, "Transformer for Future Atari Time Steps" (TransFATS) consists of two main components: a feature extraction component, which maps the input images and object bounding boxes to feature representations of each object, and a prediction component, which takes these object features and predicts where the objects will be at future time steps.

The feature extractor receives images of the screen taken at the previous four timesteps, and a CNN is used to obtain a feature map. Learned positional encodings are added to enhance this feature map. Next, the features are max-pooled over the pre-computed object masks provided by the segmentation model to get the feature vector for each object. Once the features for each object have been determined, the features are put through a small MLP to encode them for the transformer.

This embedded representation then undergoes recurrent transformations through the transformer encoder conditioned on the current user action at each time step, predicting the next latent state. We use a transformer because it should be able to model the interactions between different objects, which is important in cases like when the ball bounces off a paddle in Pong. Finally, this output is decoded by another MLP to make residual predictions representing how much each object moved since the previous time step. The predicted positions can then be calculated given the current ground truth position. The loss used to train the model is the MSE loss with the ground truth positions. For more details on the model, see the Appendix.

Our baseline model uses the same feature extractor as the main model, but contains a much simpler predictor. The features are still encoded using a small MLP, but the recurrent portion of the model is just an MLP rather than a transformer. This means it is unable to model interactions between objects. The output is once again decoded by an MLP to create predictions for the movement at each time step, which is used to create the positional predictions.

Evaluation

Although MSE loss is used to train the model, it is not suitable for evaluating model performance for a variety of reasons, including its low human readability and high dependence on outliers. Empirically, we saw that the max errors in each batch were determined by mask mismatches, which are unavoidable given the source of masks and resulted in L1 errors of 0.6-0.9 for certain objects. Instead, the three

metrics that we chose to evaluate our models on were mean, median, and 90th percentile value, all with respect to L1 loss on the twentieth time step predicted, and only including objects which moved.

The reason for L1 loss is that it is easily interpreted as the percentage of the screen the predictions are off by, and it is less affected by outliers. The reason for only looking at objects that move was that including stationary objects skews loss metrics downwards, since our models perform incredibly well at making the trivial predictions for stationary objects, so including these objects gave the appearance that our model was performing better than it really was. We only looked at the 20th time step predictions since this time step is far into the future and reflects the model's actual abilities to understand the game's physics better compared to earlier time steps.

Mean, median, and 90th percentile value were chosen because together they show how well the model is doing overall, how well the model does on a typical object, and how well the model does on an object that is difficult to predict. Together, these metrics give a comprehensive and intuitive picture of model performance. The fact that the relative performance between two models on all three metrics agree on all experiments indicates that they are consistent indicators of the models performance. We nevertheless included all three since they give different insights.

Results

The TransFATS and baseline models were each trained on Pong and Freeway using FastSAM masks. The overall dataset had a size of 100,000 steps, of which we used the first 70,000 for training and the last 30,000 for testing. Individual Atari episodes usually take less than 3,000 steps, so the two datasets are primarily from different episodes. We trained the models for 10,000 steps with a batch size of 32. While this is less than an epoch, we did not see significant improvement when trained longer, so given our limited compute budget we did not train longer. We use the Adam optimizer with a learning rate of 0.001 [6]. We predicted the future position for 20 time steps. 20 time steps were chosen as the target as it is already difficult for humans, but even with stochasticity in the games, it is possible to still make accurate predictions. We used a history length of 4 on Pong, but this was not required for Freeway where a history length of 2 was sufficient.

Model	**L1 Mean**	**L1 Median**	**L1 90th percentile**
TransFATS	**0.1333**	**0.1022**	**0.2953**
Baseline	0.1767	0.1435	0.3836

Figure 5: TransFATS and Baseline Performance on Pong

Our model performed significantly better (>25%) than the baseline on Pong in all key metrics. This indicates that the addition of the transformer is useful as it allows the model to more accurately predict cases where objects have to interact. This can be seen in the first image of Figure 6 below, where the model correctly predicts how the ball will first bounce against the paddle and then wall. The second image shows that the model is able to make consistent long-term predictions. Note the horizontal shift in the left paddle on the second image, which shows that the model can still be improved further.

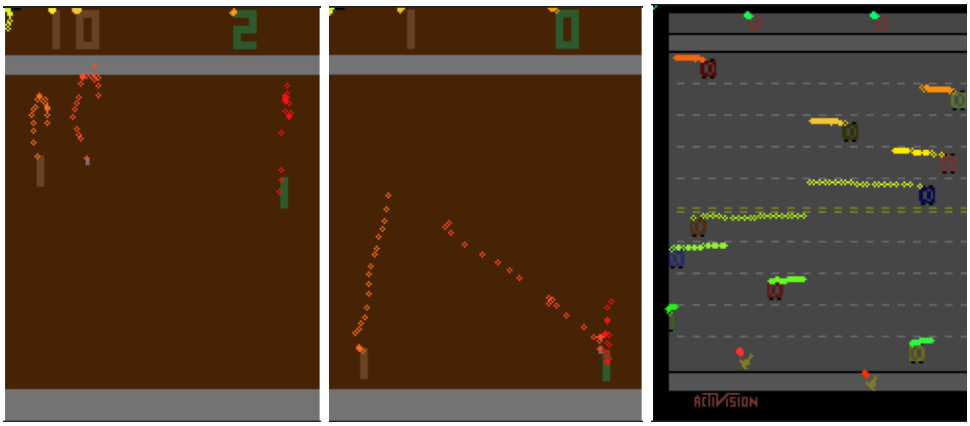

Figure 6: Sample model predictions on Pong and Freeway, using the better model for each game

On Freeway however, the transformer performs significantly worse than the baseline, as seen in the table below. This is most likely due to the fact that there is almost no interaction between the different objects, as the vehicles drive in their individual lanes. Therefore, adding a transformer only makes training more difficult and attention must learn to ignore all other objects.

Model	**L1 Mean**	**L1 Median**	**L1 90th percentile**
TransFATS	0.0800	0.0512	0.1425
Baseline	**0.0637**	**0.0421**	**0.1060**

Figure 7: TransFATS and Baseline Performance on Freeway

As Freeway is too simple an environment (there are no interactions between objects), we focus on Pong in the following ablations to better understand the performance of the model.

Ablations

Training Length

To determine how long to train the models for, we trained TransFATS for 1000, 5000, 10000 and 20000 iterations. As seen in Figure 8, the model performs much worse when trained for only 1000 or 5000 iterations, and training for 20000 iterations has no significant impact on model performance, so 10000 iterations was chosen.

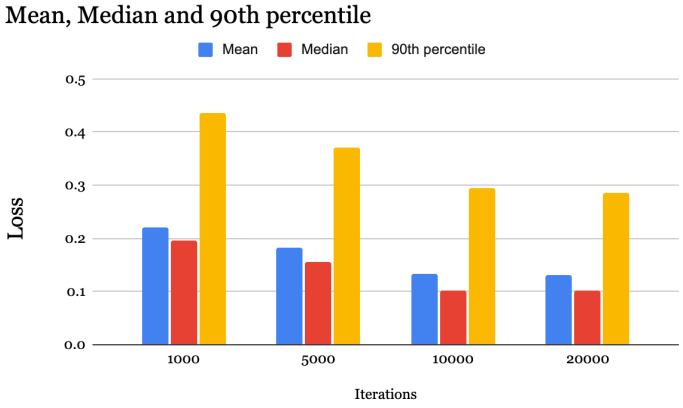

Figure 8: Loss vs. Number of Iterations

Residual Prediction

A simpler approach than predicting the movement between the previous and current frame would be to directly predict the position in the current frame. When simplifying the predictor to do this, and map the transformer outputs directly to positions, performance drops massively, which indicates that having a residual predictor is key to our performance.

Model	**L1 Mean**	**L1 Median**	**L1 90th percentile**
TransFATS	**0.1333**	**0.1022**	**0.2953**
Position Prediction	0.3658	0.3214	0.7116

Figure 9: TransFATS and Position Prediction on Pong

Feature Extractor

The Feature Extractor relies upon a CNN to extract the objects' past locations and other identifying information. To test how well the Feature Extractor was able to extract the past location, we added the ground truth past location as a flattened vector projected into the same dimension and concatenated to the CNN-based information before being projected down to the same dimension as before. As seen in the figure below, this only yields a slight improvement beyond our current model, which means that the performance bottleneck is not the Feature Extractor.

Model	**L1 Mean**	**L1 Median**	**L1 90th percentile**
TransFATS	0.1333	0.1022	0.2953
TransFATS with Ground Truth Past Position	**0.1261**	**0.0993**	**0.2688**

Figure 10: TransFATS and TransFat with Ground Truth Past Position Information on Pong

Time steps

When the model is trained for 10 time steps, it performs significantly worse, and when trained for 30 time steps, it performs somewhat worse compared to training on 20 time steps, so 20 time steps was chosen.

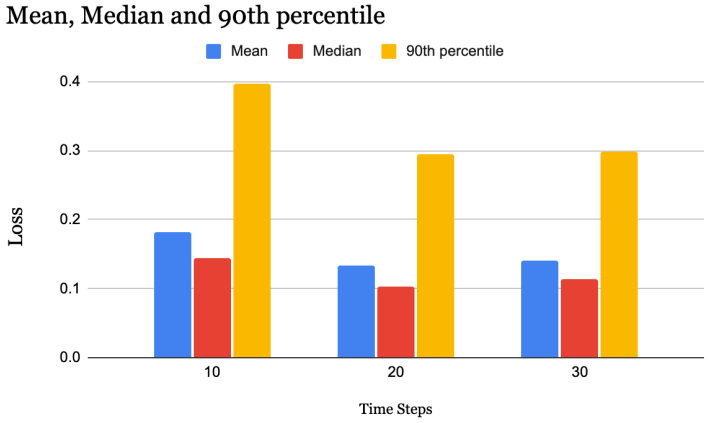

Figure 11: Loss vs. Number of Predicted Timesteps

Ethical Considerations

Since our work is focused on predicting objects in Atari games, there are no direct ethical issues arising from our project; however, using the same ideas and extending them to the real world could create ethical concerns. For example, a military entity could use real-life SAM masks of people or vehicles to predict their future locations in order to more accurately fire weapons at them. Or, this object tracking idea could be used to enhance surveillance systems.

Conclusion

Despite being called foundational segmentation models, we have found that SAM and FastSAM are not ready for usage with Atari games. Of the seven games evaluated, only on Pong and Atari was the object detection good enough for prediction usage. This indicates that the features found in natural images are not found in these significantly simpler Atari ones. For these two games, we proposed a novel architecture to predict the future behavior of objects. We found that a transformer benefited the performance on Pong, as it allowed the calculation of interactions between objects, but was detrimental to training on Freeway, where the objects moved independent of each other.

Future work

While our predictions are highly accurate for shorter time frames, they falter in performance for longer-term predictions. This suggests that our model struggles to simulate the more complex game physics as well as stochasticity in the game, although this is something that humans would struggle with as well. One path of future work could be integrating stochasticity into our model, which could give us confidence intervals for the predictions.

We also found that the SAM models fail at detecting masks in certain games like Seaquest or Skiing, instead only detecting the background features. Future work could be done on fine-tuning SAM or FastSAM specifically for 2D games, which would hopefully dramatically increase the generalization of our work on predicting object locations to all Atari games, as well as other 2D games.

References

[1] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. arXiv preprint arXiv:2007.05929, 2020.

[2] Quentin Delfosse, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting. Ocatari: Object-centric atari 2600 reinforcement learning environments. arXiv preprint arXiv:2306.08649, 2023.

[3] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan Yen Lo, et al. Segment anything. arXiv preprint arXiv:2304.02643, 2023.

[4] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. arXiv preprint arXiv:2306.12156, 2023.

[5] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374, 2019.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Appendix

Model Hyperparameters

The feature extractor input consists of four (3, 128, 128) images and ROIs of each object of size (128, 128) in the form of boolean masks. We have a convolutional layer that creates 64 channels using a kernel size of 7 and a stride of 2. Then there is a ReLU activation function and max pooling with kernel size 2, followed by another convolutional layer outputting 128 channels using a kernel size of 3 and a stride of 1, followed by another ReLU activation. Then the learned positional encodings are added to the output of the CNN to create the feature map for the images of size (128, 64, 64). We then max pool for each ROI over the feature map by interpolating the ROIs from (128, 128) to (64, 64). This creates a feature map of size 128 for each object.

Next these features are passed through a two layer MLP, with hidden size 32 and output size 120, using ReLU activation functions. The model then takes in the actions of the user, which are embedded to size (T, 8), which is then copied to create features for each object. Then at each time step, this is concatenated to the previous features to create a 128 length feature vector for each object, which is embedded back to a 120 length vector using a fully connected layer and a ReLU activation function.

This is fed through a transformer encoder with a single transformer encoder layer containing one head, an input size of 120, and an output size of 120. The output is again concatenated with the action embedding, and the process is repeated for each time step to create predictions of size (T, O, 120). These are mapped using a two layer MLP with hidden size 120 and ReLU activation function to create the predictions for movement at each time step, of size (T, O, 2), which is then used to make the final positional predictions.

In the baseline model, the transformer encoder is replaced by a MLP with two layers, both of which have input and output size 120, with ReLU activation functions.

Contributions

We were all happy with each other's individual contributions.

- **Colin**: Initial OCAtari setup, SAM vs. FastSAM analysis, data collection code including enabling SAM and FastSAM models, data loader code/testing, initial feature extractor and predictor, testing ablations, and writing parts of the paper including abstract, SAM vs. FastSAM, architecture diagram, ethics, and future work.
- **Jasper**: Data pipeline (collection and loader), collected data, contributed to predictor, ran most experiments, wrote parts of introduction, background, methodology, results, conclusion, future work.
- **Ben**: Worked on model architecture and model evaluation metrics. Wrote about the model architecture, performance, and evaluation metrics, and wrote the appendix.