Visual Prediction with Action Feedback

Pascal Pompey
Stanford, cs231n
papompey@stanford.edu

Sudeep Sudhir Jain
Stanford, cs231n
sudeepj@stanford.edu

Andrei Bajenov
Stanford, cs231n
abajenov@stanford.edu

Abstract

This work focuses on the problem of video prediction applied to the Atari game called Pac-Man. We present the results of different architectures applied to the video prediction problem along with a discussion of the advantages and performance of each. We also present the results of a hyper-parameter search that explain the poor performance of some models.

1. Introduction

Trying to predict what is going to happen is a central part of understanding one's environment. Humans are particularly good at doing this using visual cues: given a few images of the environment, humans can quite accurately predict how this environment is likely to modify and what the next visual input shall roughly be like. Examples include predicting the next positions of cars on a highway or predicting the next position of a ball being thrown. This project focuses on video prediction and aims at predicting the next image frame in a set of images of the same scene ordered in time. The relevance of this task is fairly obvious: being able to predict where a car or pedestrian is likely to be next, or where an object is flying toward are just few of many examples showing that visual anticipation is central to many AI tasks.

Solving the general problem is very hard. There are many variables at play, and making such a model is unfeasible given our resources. We simplify the problem by solving it for a much simpler environment: the Atari Pac-Man game. Pac-Man is a good candidate because it is: (1) relatively complex, (2) has some reduced elements of uncertainty (e.g. the direction taken by 'ghosts'), and (3) is a fairly simple game in that most of the next moves can be inferred from visual cues. For instance, the ghosts tend to look in the direction they move in and they also tend to continue in their current direction while possible.

To reduce randomness in the observed environment, the learner is given the action taken by the agent controlling the game as input. This means the next Pac-Man move (up, down, left or right) is given as input to our models.

The data-set used for this work is a suite of frames generated using the open AI Gym API [2] and the action taken by the agent for each of these frame. The algorithm is given a sequence of frames (as RGB images) and the associated actions (e.g. move up, down etc.) sequentially. For each input frame set, the algorithm is tasked with predicting the next frame from the same sequence as an RGB image.

2. Related Work

Analyzing the changes in a video has been a widely studied and is known in the literature as optical flow [1]. Optical flow aims at following a given pixel's movement throughout a suite of images in a video. However, until now, very little attention has been given to the problem of video prediction, which, given the previous images, aims at predicting the next image in a video sequence.

With the development of convolutional neural networks (CNNs) [12] video prediction is now the focus of more research. Three recent works are of particular interest. A variety of architectures to predict next frames in a number of Atari games from the open AI Gym repository are proposed in [14]. Having observed that using the L2 norm (or root mean squared error) to train a video prediction system often led to the generation of blurry images, researchers proposed a new loss criterion along with other training methods using Generative Aversarial Networks (GANs) for image prediction [13]. In [16], the authors use a deep neural network using convolutional encoder-decoder and a convolutional LSTM for the prediction of future frames in natural video sequences.

In general, Video prediction techniques rely heavily on Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Auto-encoders (AE), Generative Adversarial Networks (GANs), and Deep Residual Networks (ResNets); all of which come into play to build a video prediction system.

Convolutional Neural Networks (CNNs) were first introduced in [12] and have dominated the field of image recognition ever since their reintroduction in [11]. Their ability to capture localized patterns in images by applying multi-

ple layers of filters tailored to a particular application made them the current gold standard of image processing.

Auto-Encoders (AE) are closely related to video prediction as the aim of a video prediction system is to project a video into itself, and therefore, video prediction may be considered a type of auto-encoding. AE were first developed in [17]. AE demonstrated that it was possible to encode images in a latent space with a dimensionality potentially much smaller than the image itself while still being able to reconstruct (decode) the original image. A notable development is that of Variational Auto-Encoders (VAEs) [10]. Recent work [8] has shown that besides having very pleasing mathematical properties, VAEs are able to fully understand the physical laws underpinning the movement of simples objects, such as a pendulum, using solely visual inputs.

By substantially reducing the size of the original image, Auto Encoders make it possible to use Recurrent Neural Networks (RNNs) which otherwise would not fit into the memory of current hardware. RNNs are relevant to the field of video prediction because many video sequences follow the Markov property. In the case of video prediction, the Markov property means that, given a suite of images $[I_1, \cdots, I_t]$ it is possible to define a state s_t and a state update mechanism $s_{t+s} = f(s_t, I_t)$ so that s_{t+1} contains all the information required to predict the next video frame $I_{t+1} = g(s_{t+1})$. RNNs focus precisely on capturing that type of Markov recursion. Two notable RNN architectures are considered to have the best performance: the Long Short Term Memory networks (LSTM) [7] and the Gated Recurrent Units (GRU) [3].

Generative Adversarial Networks (GANs) have been used to generate realistic images [5]. GANs use a game theoretic approach to learning by letting a Generative network G and a Discriminative network D compete with one another. G tries to fool D into thinking the images it generates images are real, while D is trying to find ways to discriminate between real images and images generated by G. These networks are relevant for video prediction because, as demonstrated in [15], they do not suffer from the blurry spots that often result from using other methods with an L2 loss.

ResNets [6] are a recent improvement showing that the training of deep networks can be stabilized and improved substantially by leap-frogging information through some network layers. As will be seen in this work, this idea can be applied to video prediction with great success. Indeed, forcing a video prediction system to predict the next image as $I_{t+t} = Network\ Output(I_t) + I_t$ enables a network to focus on only predicting the difference between two frames instead of the complete image; a much simpler learning problem.

Finally, developing a video prediction system would be short of impossible without the use of state of the art deep-learning frameworks. Pytorch [4] was used to develop all the models presented in this report. Throughout the project, the Adam [9] optimizer was used to train the models.

3. Problem definition

3.1. Data simulation

The Open AI gym [2] is a library that allows simulation of a number of environments for reinforcement learning. Simulation has the advantage of trivializing the requirements for a labeled data-set. For the purpose of video prediction, it has the further advantage of a simpler and more controlled environment, where the intended action of some of the elements in the video are known (i.e. the input to the Atari console).

All the Atari environments implement the following interface: given a time-step t:

1. the environment provides an observation I_t (most often under the form of an RGB image) for the current time-step
2. a step function enables to have the agent take an action a_t in the environment and, following that action, the next observation is returned: $I_{t+1} = env.step(I_t, a_t)$

In this work, the Atari Pac-Man game was used to simulate a video prediction problem. This work does not focus on reinforcement learning and, therefore the agent behaved at random during the simulations. However, the action taken by the agent was recorded along with the image generated by the game, as that action will be used as input to for the networks.

As generating images is trivialized by the use of open AI Gym, generating validation and test sets is simply done by generating further game simulations after the training phase.

3.2. Accuracy criteria

In this work, the Root Mean Squared Error (RMSE) was used. The RMSE score was computed across all the pixels of all the channels of the predicted image, meaning if $\hat{I}_{t+1}$is the predicted image and I_{t+1} is the true image then:

$$RMSE\left(\hat{I}_t, I_t\right) = \frac{1}{K} \sum_{(c,i,j)} \left(\hat{I}_t(c,i,j) - I_t(c,i,j)\right)^2$$

where c is the index of the channel, (i, j) the position of the pixel in the image and K a normalization constant equal to the number of channels times the number of pixels.

The RMSE score is known to have a number of flaws; for image generation, it is known to generate blurry images. E.g. if a pixel is always either red or blue, the RMSE score

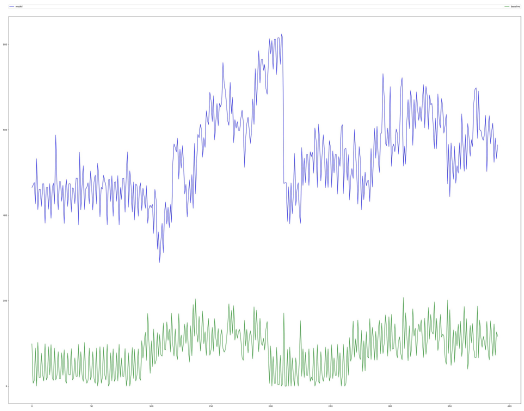

Figure 1. An Auto-encoder was tested on 400 frames of a Pac-Man game. The model is fairly good for the first 100 frames but prediction accuracy decreases sharply after that. Interestingly frame 100 coincides with the actual game starting; with the protagonists actually starting to move on the Pac-Man board. In that example, Pac-Man was caught by a ghost at frame 220; at that step the game resets to the usual starting environment, and this translates into the visible sharp decrease in prediction loss from the model. The model was therefore over-fitting to the starting game setup frames and not learning about the in-game behaviors.

will tend to average, constantly outputting a violet pixel. Such an average pixel will minimize the RMSE loss score but not generate a sharp image.

There exist a number of other criteria, that could be used, for instance GAN or Mean Average Percentage Error (MAPE). In the case of Pac-Man, as the color code is very limited, it is feasible to recast the problem to a per pixel classification problem, where the aim is to classify the color of the pixel. We chose RMSE loss because its simple and well understood.

3.3. Biased sampling and over-fitting Pac-Man

Despite the fact that simulation enables to generate an infinite number of training images, it is possible to over-fit Pac-Man if one is not careful about the simulation setting. The Pac-Man environment starts each game with a number of static frames before the game actually begins. As these first 100 frames are always nearly identical at the beginning of each game, a model can over-fit to these frames.

Figure 1 is an example of this phenomenon. This example indicates that it is easy to over-fit a specific image manifold instead of learning the game.

3.4. Generating consistent batches

Variants of Stochastic Gradient Descent (SGD) optimization methods [9] underpin almost all the deep-learning algorithms. SGD requires to train the model by batches, with each of the batches being independent draws from the input space.

In this work, we used batch sizes of 8. To generate batches for video prediction, 8 Atari games were played in parallel, each being responsible for generating one image of that batch. This was required to ensure that each batch element actually refers to a correct time-series of images from a batch. Originally, only one game environment was used to generate all images in a batch. That had the effect of confusing our RNN models, because, after each batch, the images in each time-series were jumping by batch-size steps instead of just one. In effect that was equivalent to making 8 step ahead prediction.

3.5. Pure Markov Property

Most of the prior work on video prediction [14, 13] works by taking a series of frames as input $[I_1, ..., I_t]$ in order to predict the next frame I_{t+1}. In this work, only the last image I_t will be used in our models.

3.6. Baseline

For the baseline, we use the current frame of a video sequence as a prediction for the next frame. This is generally quite a hard baseline to beat because in Pac-Man, changes between consecutive frames are very minimal. The prediction of the game environment is perfectly sharp, so the only observed error is in the position of the ghosts and the Pac-Man.

4. Models

A number of architectures were tested to solve the Pac-Man video-prediction problem. The following presents the architecture of the models that were tested.

4.1. Architectural Considerations

Filter size It is known that it is better to stack multiple convolution layers with a small filters (e.g. 3) than to have a single layer with a large filter size (e.g.12). Therefore we set the filter size of all the convolution layers in the models to 3.

Minimal receptive field Neurons in a convolutional network have a receptive field, which is the surface of the original image that is involved in the computation of the output score from that neuron. Intuitively, the receptive field represents what the neuron 'sees' of the original image. An immediate implication is that if a neuron doesn't have a sufficient receptive field, it won't be able to apprehend some elements of the image.

The objects in the Pac-Man game have a given size and evolve in a world of paths delimited by walls. This means

that, to be able to understand what is happening in the game, the final layers of a CNN need to have a minimum receptive field of at least the object and its surrounding walls. After some tests, it was concluded that the minimal receptive field to see Pac-Man or a ghost along with its two closest walls was 13 pixels. 13 pixels is therefore the lower bound for the receptive field of our final encoding CNN layers.

This consideration enabled us to compute the minimum number of layers required in our CNN to ensure that receptive field.

4.2. Three flavours of ResNets

Three key methods were used to generate the output of our models:

StandardNets For our first, naive models, the output of the neural network was used as the predicted RGB image. These network didn't perform well, starting with a loss around $40k$ and stagnating at a loss of around 1500 at convergence. 1500 is a few orders of magnitude higher than the baseline.

ResNets The second models used the ResNet idea of [6]. This is equivalent to the intuition that it might be much simpler to predict the difference between two frames. It is possible to force the model network to predict the difference by adding the previous image to the image output by the network, leading to:

$$\hat{I}_{t+1} = I_t + NN\left(I_t, a_t\right)$$

where $\hat{I}_{t+1}$is the image output by the complete model, I_t (resp a_t) is the image (resp. action) at the previous timestep and $NN\left(I_t, a_t\right)$ is the image generated by the neural network.

ResNets proved very efficient, converging to the baseline after just 2 batches of learning. However, once at the baseline (which means the output of the neural network model were only zeros), it remained there. In the validation graphs, the baseline error and the Resnet model error were exactly equal. This, obviously, was not satisfactory.

DiffNets The last models just changed one sign in the ResNet equation; leading to:

$$\hat{I}_{t+1} = I_t - NN\left(I_t, a_t\right)$$

In short, the neural network predicts what shall be subtracted from the image to morph it into the next one. While less accurate than both, the ResNets and the baseline, these were considered the best performing models because (1) they did reach accuracies in the order of magnitude of the baseline and (2) contrary to the ResNet, their results were not completely trivial (e.g. the network outputting the baseline image) but actually learned the Pac-Man layout and patterns (ghost and Pac-Man). All the models presented in the next section are of the DiffNets type.

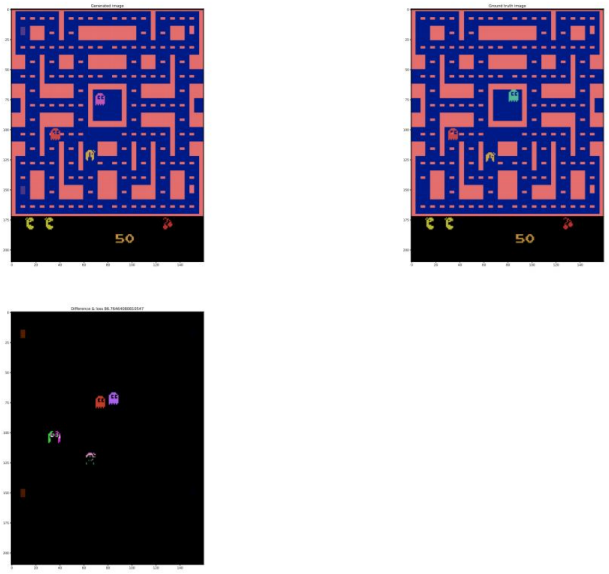

Figure 2. The output of a ResNet model: the top left image is the image generated by the model, the top right the ground truth, the bottom left image is the difference between the ground truth and the generated image i.e. $I_{t+1} - I_t$

4.3. Neural Network architectures

4.3.1 Architecture 1: Purely Forward CNN

Model Description The first model tested was a simple convolutional neural network that conserved the size of the original images and used

- 6 convolutional layers with 120 filters conserving the original image size (filter size 3, padding 1, stride 1) to reach the intended receptive field of 13.
- 6 convTranspose layers conserving the original image size (filter size 3, padding 1, stride 1) with the final layer generating a three layers RGB image.

Results As visible on Fig.3 the model converged quickly to loss values similar to the baseline.

This model was able to get results that were slightly (yet not significantly) better than the baseline (see Fig. 4).

An example of an image generated with the full forward CNN is given in Fig.5. Observing the generated images and their difference with the target revealed that:

- The model was able to capture the shapes of the Pac-Man and ghosts

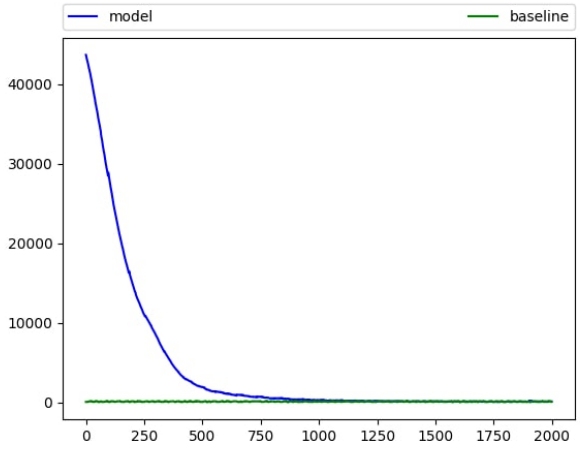

Figure 3. Full loss history over a training run for the purely forward CNN shows a healthy convergence.

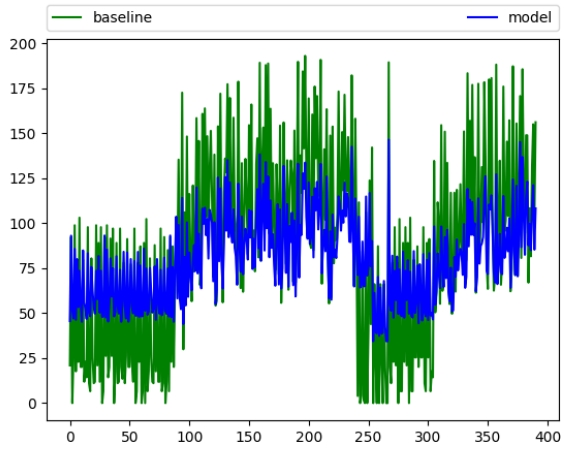

Figure 4. Loss comparison between the baseline model and the purely forward CNN.

- The RMSE criterion was pushing the model to mean values in areas of high uncertainty, resulting to more blurry images around the Pac-Man, ghosts or blinking objects.

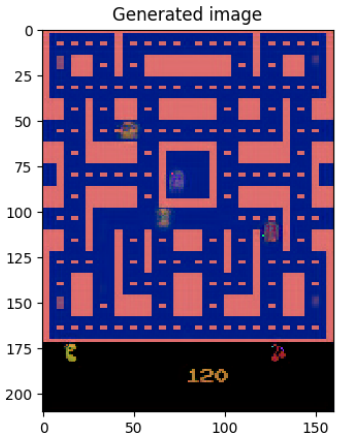

Figure 5. An image generated by the purely forward CNN. The network achieves lower loss by blurring the patterns of the Ghosts and Pac-Man

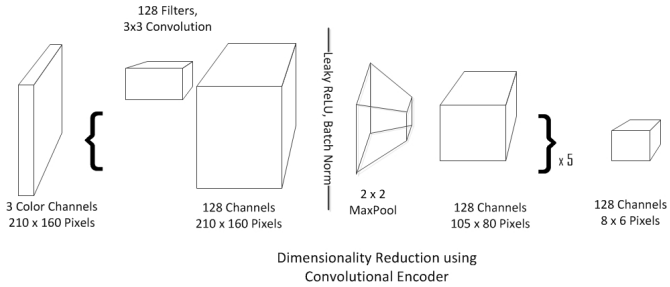

Figure 6. Reducing Convolutional Auto Encoder

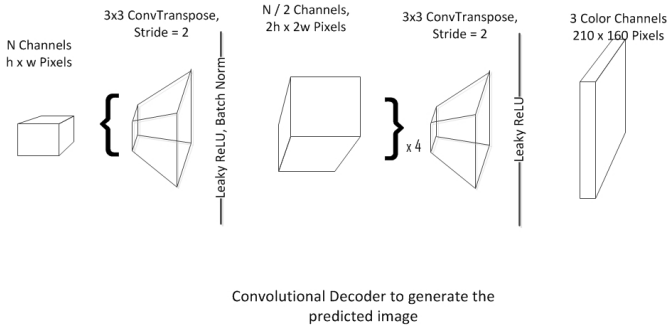

Figure 7. CNN architecture to predict the next image

4.3.2 Architecture 2: Reducing Auto-Encoder

Model 1 had the shortcoming of having a lot of parameters. This not only made it slow to train and score but also prevented the use of RNNs as the output at the end of the encoder would be of far too high dimensionality.

Model 2 aimed to address this short-coming.

Model Description A CNN encoder divides the size of the image by 2 in the height and width, until a receptive field of at least 13 is reached. To do this, two methods were tested: (1) using a stride of 2 and (2) using max-pooling layers. A ConvTranspose decoder scales up the generated fields until the size of the original image is recreated. A graphical representation of the model is shown in Figures 6 and 7

At each encoding layer, the number of channels was multiplied by 2 until it reached a cap value, from which point the number of channels was kept constant. Vice-versa, the

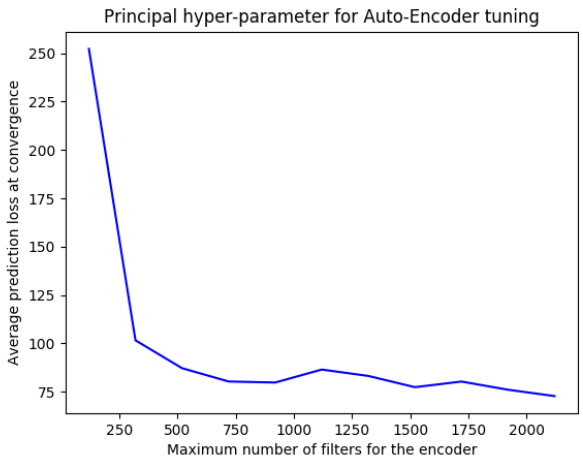

Figure 8. Hyper-parameter tuning: setting the maximal number of filters right halves the loss. More filters is better, although there is an inflection point around 400 filters from which the accuracy return for adding more filters (and hence parameters) strongly diminishes.

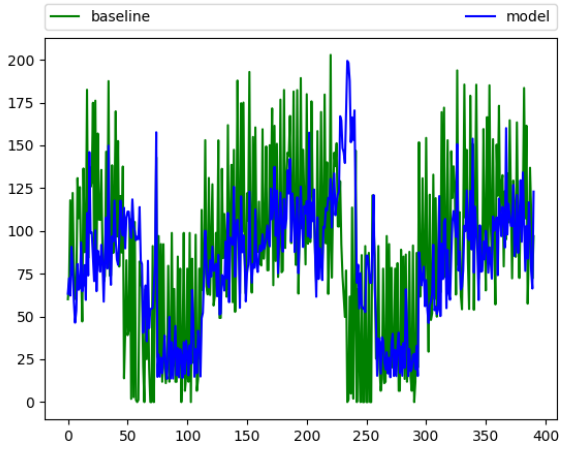

Figure 9. Loss comparison between the baseline model and the Reducing AE, here with a network having a cap at 1520 filters per layer.

number of channels in the decoder had hits number of channels divided by two until it reached a cap. As discussed below, it turns out that the cap value on the maximum number of channels for the encoder is a key hyper-parameter, central to the performance of the model.

Hyper-parameters tuning Getting the reducing auto-encoder to top performance required selecting the correct parameters for dimensioning the network. The parameters of particular interest were:

- the number of layers in the network: smaller networks with depth 6 for the encoder and decoder were found to perform better.
- whether max pooling or striding was better for dimensionality reduction: both performed similarly.
- the maximum number of channels allowed in the encoder: this was found to be the key hyper-parameter. Fig.8 illustrates the importance of that parameter.
- the learning routine (in our experience, Adam works best)

While many hyper-parameters were tried, only the plots w.r.t to the maximum number of filters in the encoder is shown in the interest of space. This parameter was found to have the greatest impact on loss.

Results The reducing AE achieved results similar to the purely forward CNN, but with far fewer parameters.

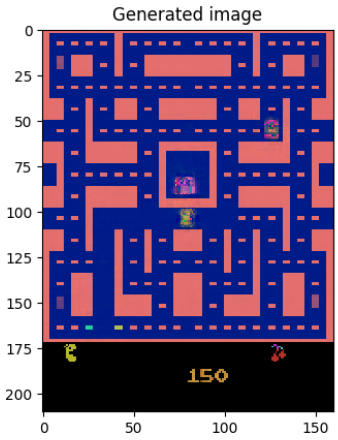

Figure 10. An image generated by a Reducing AE having a cap at 1920 filters. The blurring effect on the Ghosts and Pac-Man is even more pronounced

4.3.3 Architecture 3: Reducing Encoder encapsulating an LSTM

Model Description Pac-Man follows the Markov property. Based on a suitable state s_t and the current image I_t, it is possible to fully determine the next state s_{t+1} from which one can generate the next image I_{t+1}. It is therefore perfectly legitimate to try and use some RNNs as core components of the architecture.

RNNs contain fully connected layers and are therefore very memory intensive. This means that using them without triggering out of memory errors requires aggressively scaling down the original image of (3*210*160) to a much smaller parameters set.

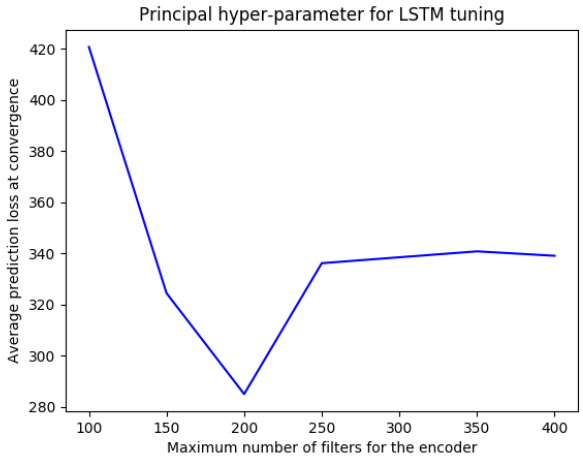

Figure 11. Hyper-parameter tuning: setting the maximal number of filters right halves the loss.

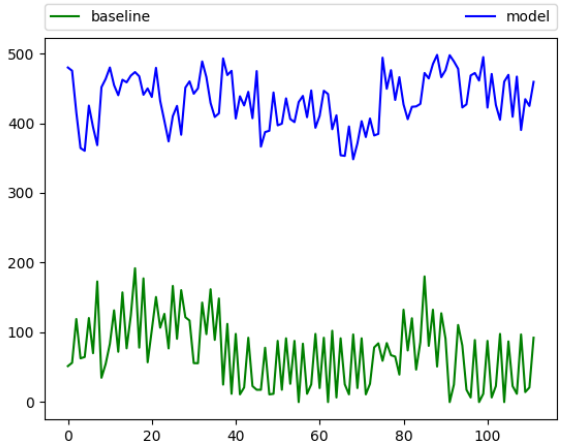

Figure 12. Loss of a Convolutional Auto Encoder in an LSTM RNN compared to the baseline.

The architecture was therefore to:

1. Use a convolutional encoder to reduce the original image to a reasonable size.
2. Apply the RNN on the flattened output of that image.
3. Use a convolutional decoder to reconstruct the image from the RNN updated state

Hyper parameters tuning Similar to the reducing auto-encoder, the LSTM model required tuning of its dimensioning hyper-parameters. Note, that through the flatten operation in this architecture, the LSTM hidden size and output size are direct linear combinations of the number of output filters in the encoder.

One new hyper-parameter was required in the RNN case: the number of steps the model would do back-propagation through time.

Similar to the reducing AE, the main hyper-parameter influencing the performance of the LSTM model was the maximum number of filters, or, equivalently, the size of the LSTM hidden state and cell. Figure 11 demonstrates the impact of this parameter. Note that to prevent out of memory exceptions, the size of the LSTM hidden state had to be kept small.

Results As shown in figure Fig.12 the LSTM model was not able to achieve a loss close to the baseline. A reason for this is apparent in the analysis of Fig. 8: fitting the image in memory required capping the number of filters used in the encoder to too low a number, resulting in a detrimental loss of information.

Figure 13 features an image generated with an LSTM model with hidden size 250. It shows that the LSTM model

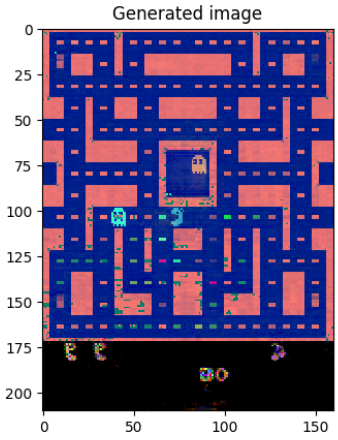

Figure 13. Image generated by a LSTM model with hidden size 250

is struggling to even regenerate the Pac-Man board. One encouraging note however is that, as visible on that image, the LSTM model seems to be able to decide whether the Pac-Man object will have an open mouth or not. This indicates that keeping a markovian state information indeed enabled the model to (1) disambiguate between the open and closed Pac-Man states and (2) start anticipating when these change roughly occur.

5. Visualizing and interpreting output

Being able to interpret what is leading to errors is key in designing a video prediction system. Fig.14 shows an example of the type of visualization that was developed for that purpose. Each model's accuracy was tested on generated games of 500 frames, each of the model's predictions

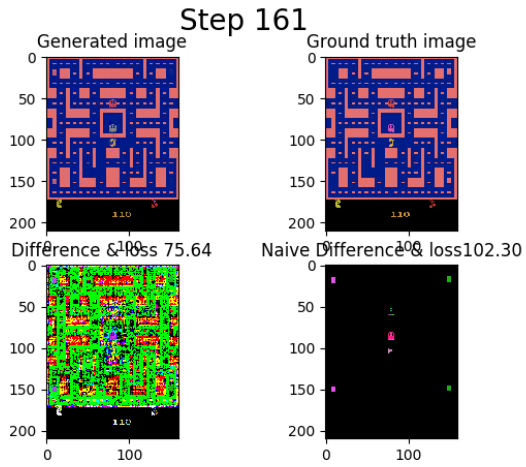

Figure 14. Example of the images used to get insight into a video prediction model. The top left image is the generated image, the top right the ground truth, the bottom left image is the difference between the generated image and the ground truth, the bottom right image is the difference between the ground truth and the previous frame

was saved in a report pdf that enabled us to follow frame by frame what the model was doing.

This is how we determined, that the effect of biased sampling during the data generation process could lead to overfitting (as presented in subsection 3.3). Examples of such pdf reports can be found in the zip attached with the report.

6. Conclusion

We set out to solve the problem of video prediction applied to the Atari game called Pac-Man. We designed a few models to try to evaluate ideas we learned from prior work.

We found that the performance of all our models was very close to that of the baseline with only a few models slightly outperforming it. It turned out to be a difficult task to get the models to output images as sharp as those given by the baseline.

We saw good performance with a purely forward CNN model. It was able to outperform the baseline model, however the generated images were blurry mainly due to the fact that we used an RMSE loss.

The Reducing Auto-Encoder model was also able to barely beat the baseline. The advantage it had over the purely forward CNN model is that it used far fewer parameters and generated sharper images.

The smaller parameter numbers of the Reducing Auto-Encoder model also allowed us to incorporate RNNs. Unfortunately, even with the reduced parameter numbers, we didn't have sufficient hardware resources to run with the number of filters we needed. As such, the performance of our RNN models was worse than that of the baseline. With better hardware we believe we should have been able to get the RNN models to beat our baseline.

7. Future Work

There are a few ideas we would like to try next.

- Try feeding a sequence of images directly into our CNN models, without resorting to RNNs. This takes less hardware resources and may give better results.
- Experiment with Gated Recurrent Units (GRUs). GRUs have been found to have similar performance to LSTM while having a much smaller parameter space. The fact that GRUs use less memory means it would be possible to increase the number of filters at the end of the encoder, which, following the analysis of figure Fig.8, is likely to improve generation.
- Train a separate model for each action taken by the player. This would result with 4x the number of parameters but would likely slightly improve the performance.
- Given more time and hardware, expand the hyperparameter tuning to more parameters. This would allow us to better fine-tune the models.
- Experiment with different non-linearity functions to try to get the ResNet models to learn instead of outputting 0s.

References

[1] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[3] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[4] B. J. Erickson, P. Korfiatis, Z. Akkus, T. Kline, and K. Philbrick. Toolkits and libraries for deep learning. *Journal of Digital Imaging*, pages 1–6, 2017.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[13] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[14] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[15] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[16] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. Decomposing motion and content for natural video sequence prediction. *ICLR*, 1(2):7, 2017.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.