Now we're trying to learn the autoencoder and the policy at the same time. This changes things. We begin with an architecture we already know works. While not the best performing one, it's ok. Namely we take the autoencoder with layers as in the original Deepmind DQN paper, and a 2 FC layer policy. Because this worked when all layers were used as the $\mathrm{q}_n etwork, andwhentheconvlayerswerepre-$ $trainedthentheq-networkwastrainedintheencoder'slatentspace, weexpectthistoworkhereaswell.$

1 Experiment

1.1 Idea

First we try to replace the 2-step training process. Thus we start we a blank autoencoder and we train it in parallel with the q-network. The hypothesis is that we'll get the same final result as in the 2-step training procedure, but that the training process will be more unstable and that it will take longer. However, given the speed with which the autoencoder is trained, this shouldn't make the training more than say 25% slower.

1.2 Results

It completely failed. To see why, we need to unpack the results. In particular this means a few things. First, what did the autoencoder actually learn. Second, how much does the never-ending autoencoder training shift the latent space representations (we expect close to nothing of course, but this needs to be checked).

It failed because now it takes in 4 frames and you're taking loss over those 4 frames again. In the same network. And that certainly does not work. So either take the (4, 84, 84) observation and predict the first frame in the next observation or whatever, i.e. do forward prediction, or legit take 4, compress each one, and give all 4 to the policy (stacked ofc) and do a loss on each compression individually. Also because of this, the first layer of the policy network got 4 times smaller.