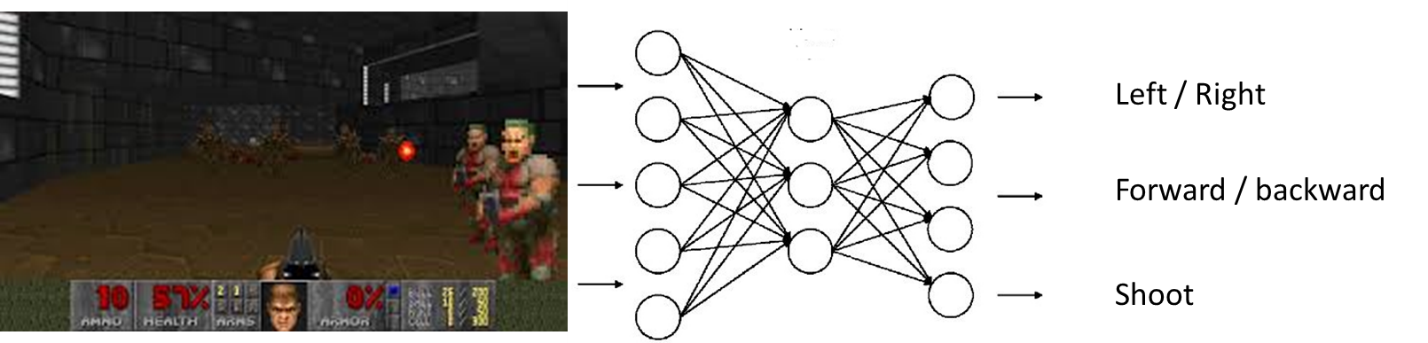
**Naïve approach**

The core idea of the approach taken in the project is to apply evolutionary algorithms to create and fine-tune deep networks. As we discussed, a naïve approach was to directly use the NEAT algorithm (Neuro Evolution of Augmented Topology) to discover a multi-layered network that maps pixel information to actions in Doom.



As suspected, this approach did not show very good results, mainly because of the high dimensionality of the input array (64x48 pixels image with 3 channels [RGB], which results in a total of 9216 units). The NEAT algorithm seems to have a hard time mapping those to reasonable actions.

**Two-steps approach**

With the assumption that it was the high dimensionality what stopped NEAT from performing here, I decided to see if there are ways of reducing this complexity using evolutionary algorithms. As it turns out, a couple of authors[[1]](#footnote-2) have suggested that one way of doing so in the visual domain is by evolving *feature detectors*. The intuition behind it is to instead of using all of the raw pixel data on the neural network, one can reduce this information to the most important features present on images. Note that this is not a compression technique, such as the ones used with autoencoders; this is an approach that actively throws away some information and only keeps the fundamental features; nor it is an active feature detection technique -there is no preconceived knowledge on which features are worth keeping.

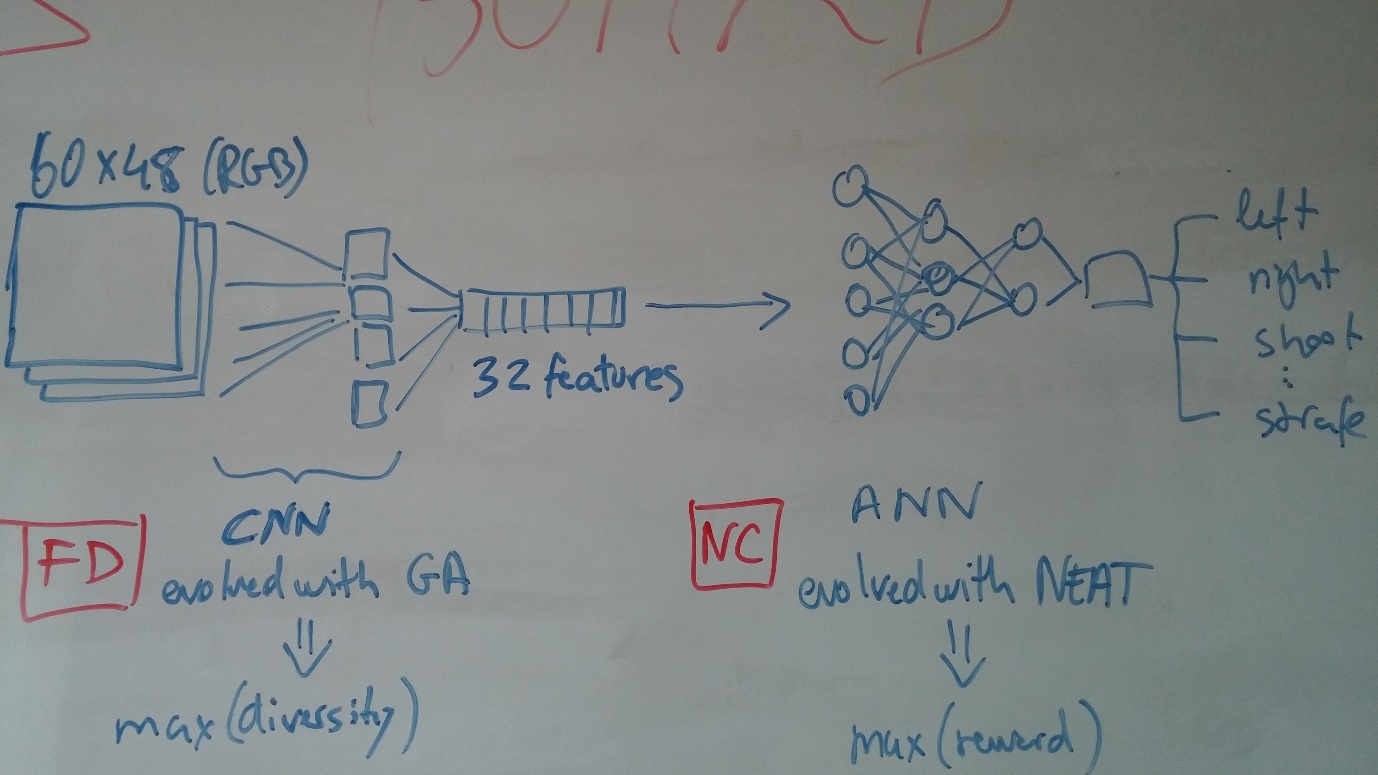
Using a genetic algorithm (with mutation and crossover as source of variation), one can evolve a neural network that learns those important features (whatever they might be). In essence, the network is reducing the dimensionality from 64x48x3 to a small feature vector (32, 64 or 128, for instance). In order to do that, the key is to select a fitness function that forces features to be as spread out as possible.

As the network evolves, feature vectors that are spread out (based on the images given) are favoured, which is equivalent to saying images will have as different outputs as possible from each other. The assumption here is that this will be enough to then build reasonable behaviour from.

In my approach, I am using a custom Genetic Algorithm to evolve the weights of a Convolutional Network. Part of the project will test which architecture is more appropriate, but at the moment I am using 4 convoluted layers each followed by a downsampling layer (max pool), ending with a fully connected layer of 32 units (which are the 32 features output).

But of course building a good feature detector is just half of the process. Once an appropriate feature detector is evolved, we are in a position to use the NEAT algorithm to evolve a network that maps the 32 features to a set of Doom actions (depending on the scenario, but typically left/right, forward/backward and shooting). The fitness function to evolve the controller is scenario-dependent, but normally it considers the number of enemies killed, the health packs and ammo picked, and the time to complete the level.

This is a high level diagram on my white board that presents the two steps approach



As you can see in the videos shared, the performance is far from perfect, but sufficiently good to suggest the goodness of the approach.

1. *Evolving Deep Unsupervised Convolutional Networks for Vision-Based Reinforcement Learning*, from Koutnik et al.; and *HyperNEAT and Novelty Search for Image Recognition*, from Kocmanek. [↑](#footnote-ref-2)