بسم الله الرحمن الرحیم

Deep Learning

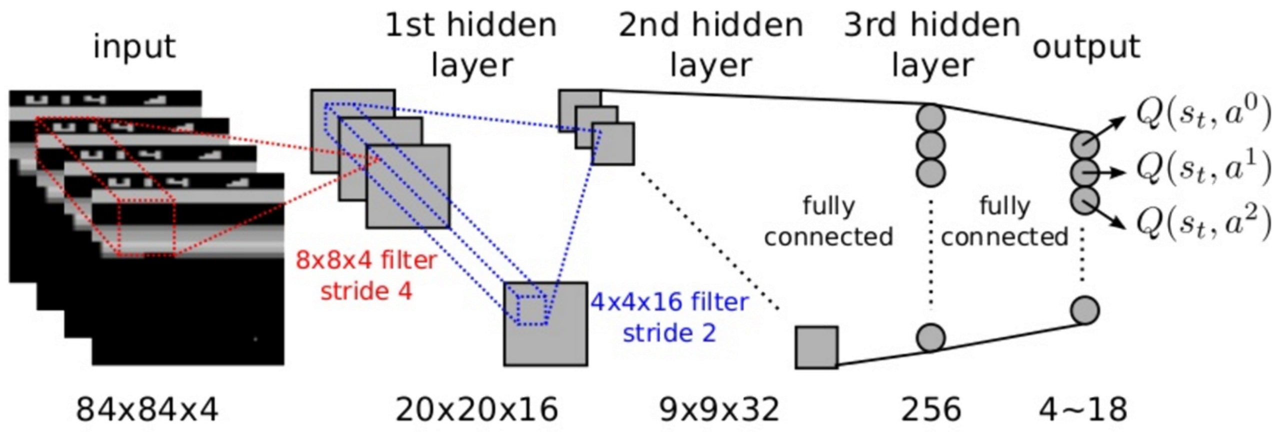
Mohammad Reza Mohammadi

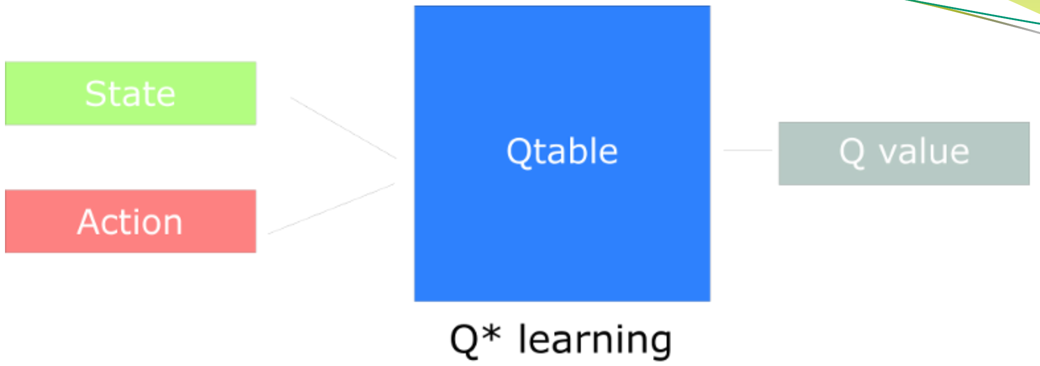
2021

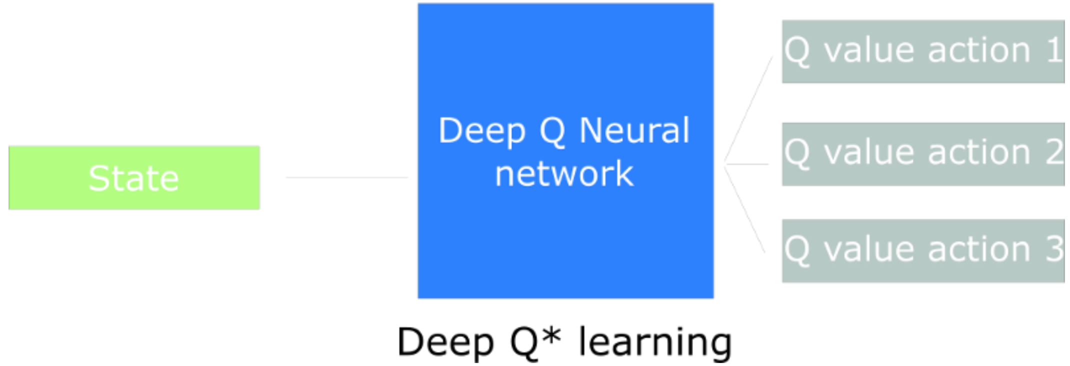
Deep Q-Learning

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(terminal\text{-}state, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

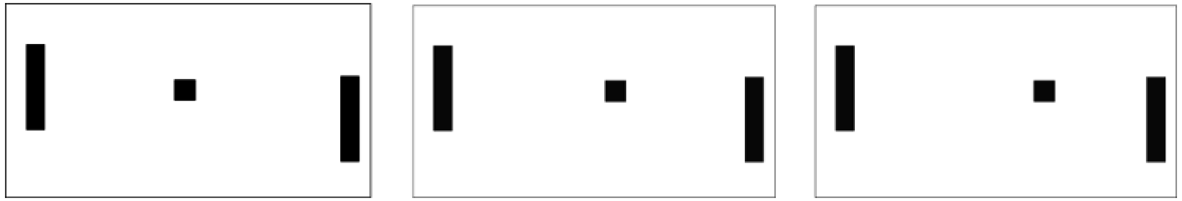


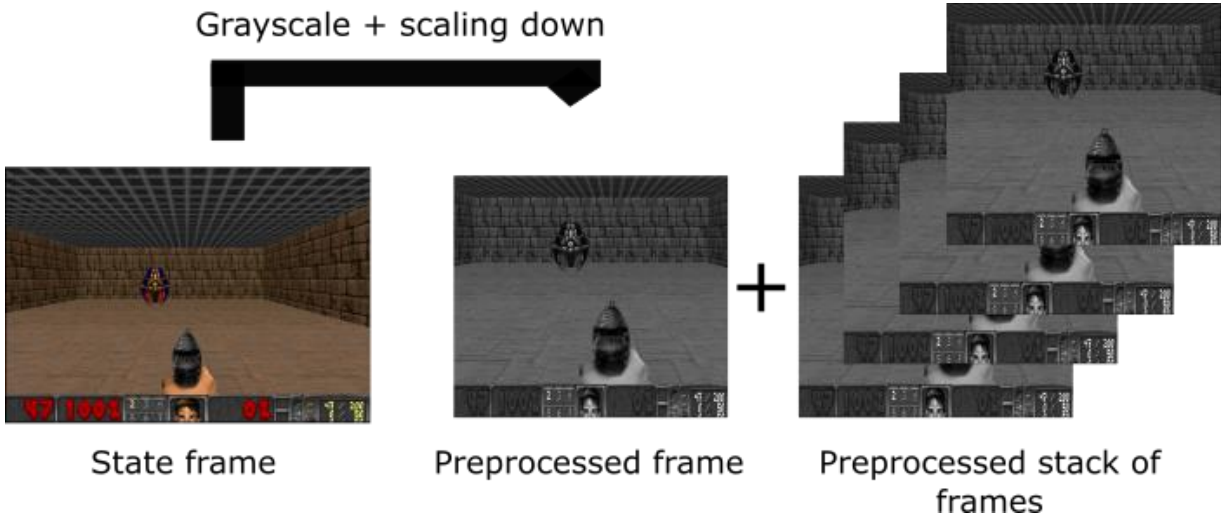




Preprocessing part

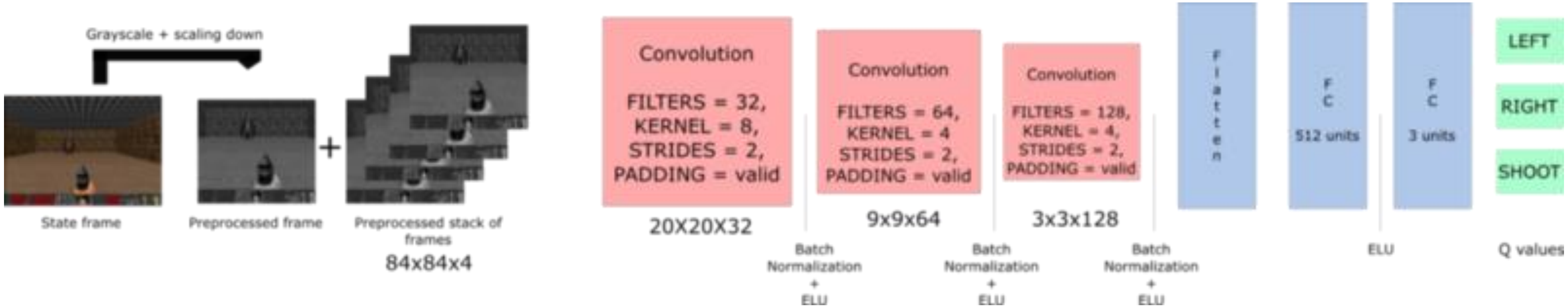
- We want to reduce the complexity of our states to reduce the computation time needed for training
  - First, we can grayscale each of our states
  - Then, we crop the frame
  - Then, we reduce the size of the frame
  - And stack four sub-frames together





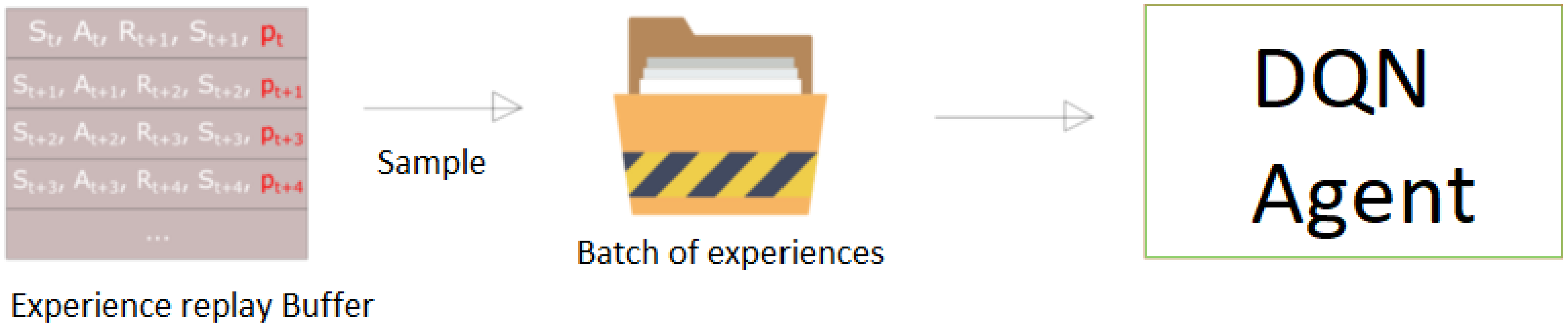
Convolutional network

- The frames are processed by three convolution layers
  - These layers allow you to exploit spatial relationships in images
  - But also, because frames are stacked together, you can exploit some spatial properties across those frames
- We use one fully connected layer and one output layer that produces the Q-value estimation for each action



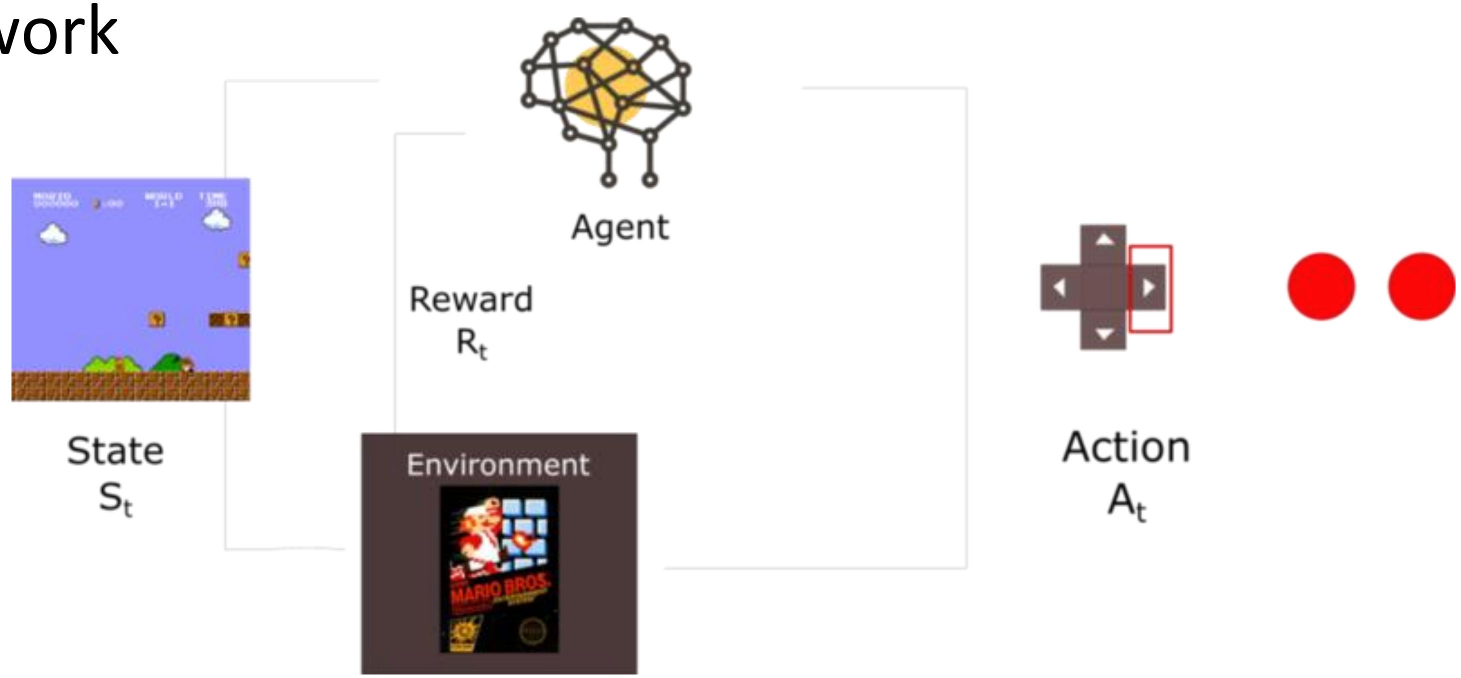
Experience Replay

- Making more efficient use of observed experience
- Experience replay will help us to handle two things:
  - Avoid forgetting previous experiences
  - Reduce correlations between experiences



Avoid forgetting previous experiences

- At each time step, we receive a tuple (state, action, reward, new_state)
  - We learn from it, and then throw this experience
- Our problem is that we give sequential samples from interactions with the environment to our neural network

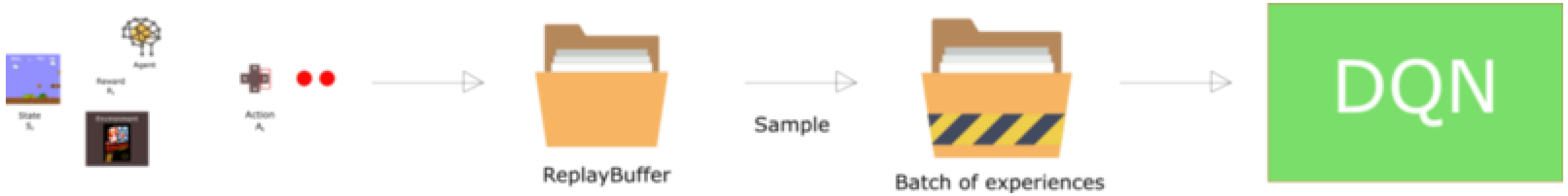


Avoid forgetting previous experiences

- Our problem is that we give sequential samples from interactions with the environment to our neural network
  - It tends to forget the previous experiences as it overwrites with new experiences
  - For instance, if we are in the first level and then the second (which is totally different), our agent can forget how to behave in the first level

Avoid forgetting previous experiences

- As a consequence, it can be more efficient to make use of previous experience, by learning with it multiple times
- Create a “replay buffer”
  - This stores experience tuples while interacting with the environment, and then we sample a small batch of tuple to feed our neural network

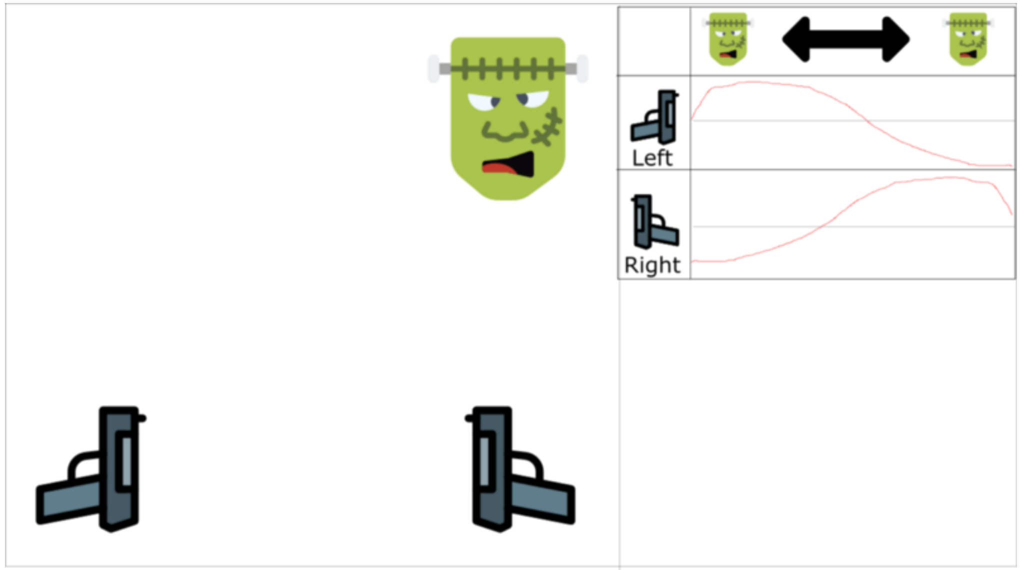


Reducing correlation between experiences

- We know that every action affects the next state
  - This outputs a sequence of experience tuples which can be highly correlated
- If we train the network in sequential order, we risk our agent being influenced by the effect of this correlation
- By sampling from the replay buffer at random, we can break this correlation
- This prevents action values from oscillating or diverging catastrophically

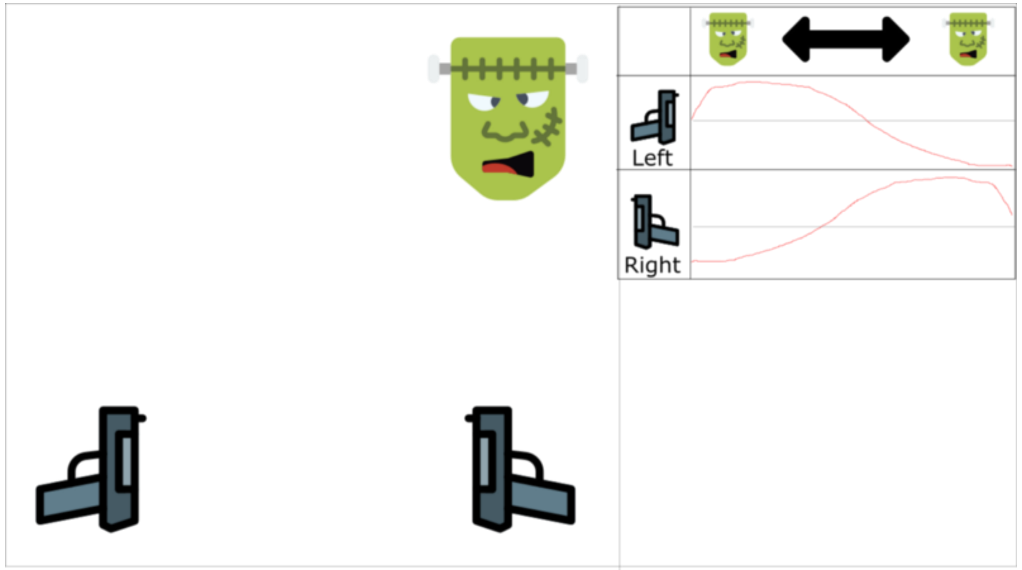
Reducing correlation between experiences

- Play a first-person shooter:
  - A monster can appear on the left or on the right
  - The goal of our agent is to shoot the monster
  - It has two guns and two actions
    - shoot left or shoot right



Reducing correlation between experiences

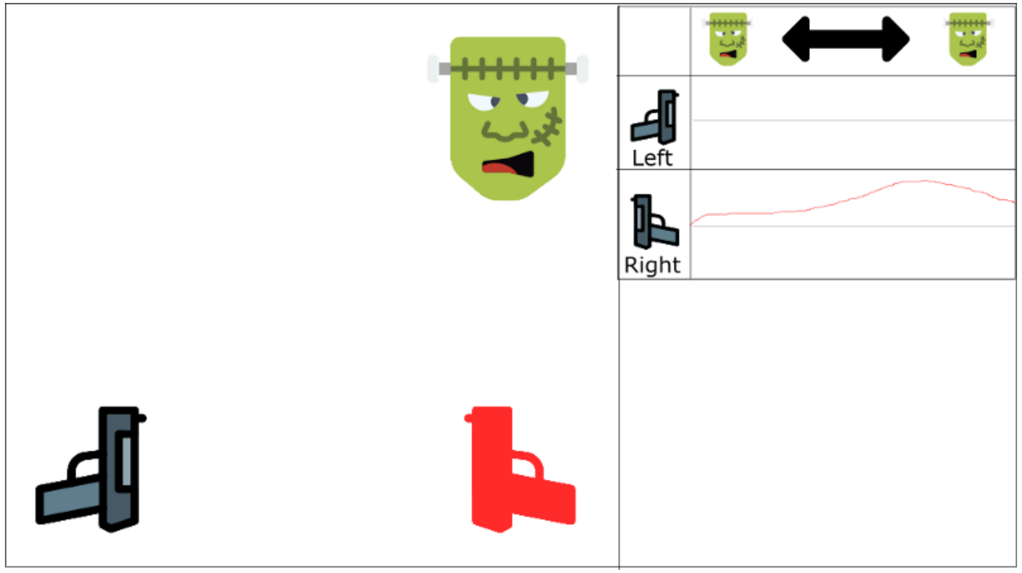
- We learn with ordered experience
- Say we know that if we shoot a monster, the probability that the next monster comes from the same direction is 70%
  - This is the correlation between our experiences tuples



Reducing correlation between experiences

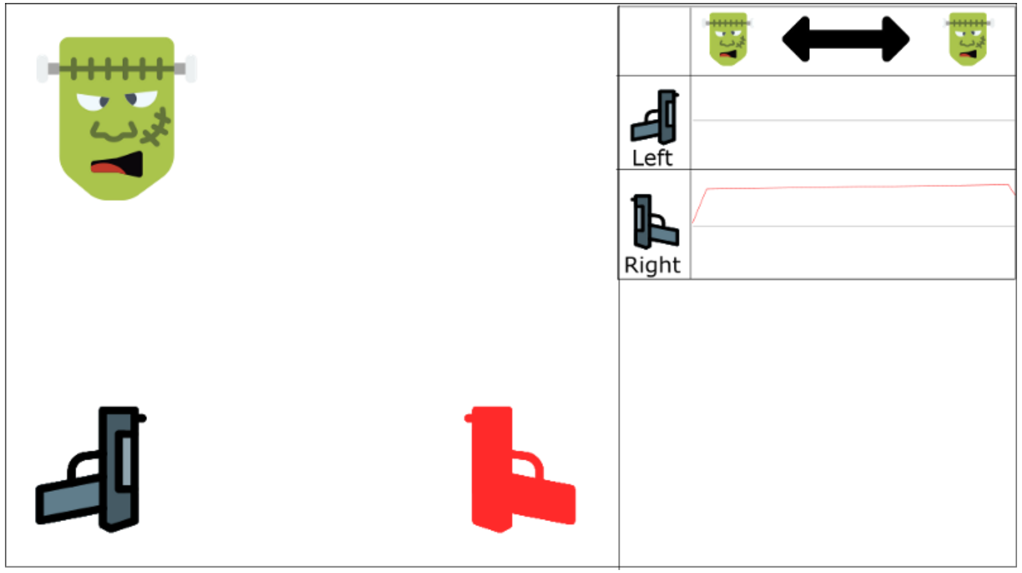
- Let's begin the training
  - Our agent sees the monster on the right, and shoots it using the right gun
  - Then the next monster also comes from the right (with 70% probability), and the agent will shoot with the right gun

- The problem is, this approach increases the value of using the right gun through the entire state space



Reducing correlation between experiences

- If our agent doesn't see a lot of left examples (since only 30% will probably come from the left), our agent will only finish by choosing right regardless of where the monster comes from



Reducing correlation between experiences

- To handle this problem:
  - First, we must stop learning while interacting with the environment
    - We should try different things and play a little randomly to explore the state space
    - We can save these experiences in the replay buffer
  - Then, we can recall these experiences and learn from them
    - After that, go back to play with updated value function

Deep Q-Learning

- We want to update our neural nets weights to reduce the error

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

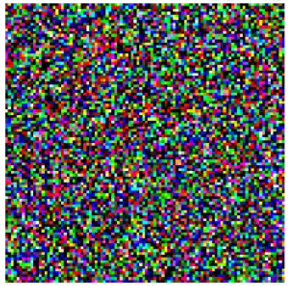
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

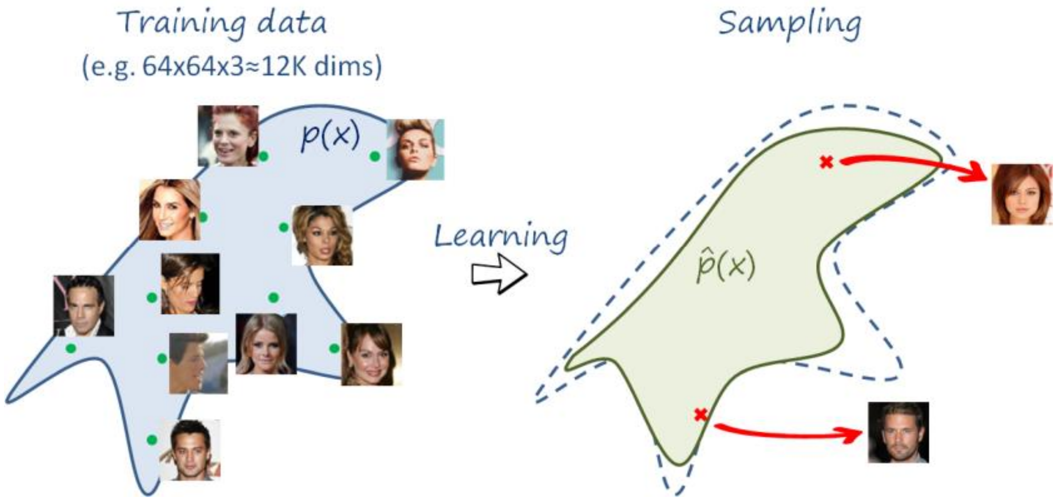
New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation
TD Target
TD Error

Generative Models

Generative models

- Given training data, generate new samples from same distribution

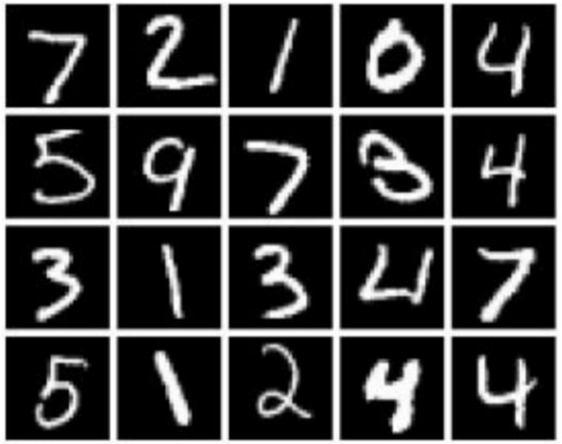




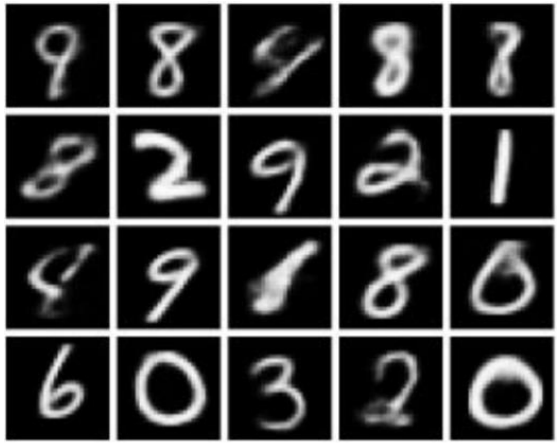
- Training data $\sim p_{data}(x)$
- Want to learn $p_{model}(x)$ similar to $p_{data}(x)$
- Generated samples $\sim p_{model}(x)$

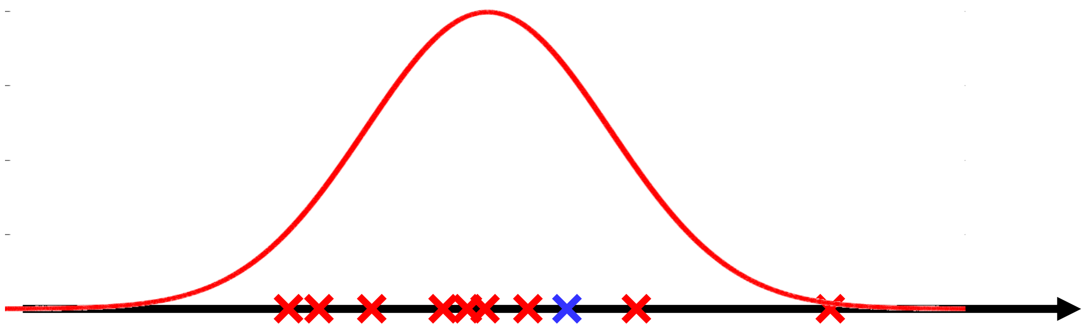
Generative models

- For example, suppose we know that the distribution of a random variable is Gaussian and we observe several instances of it
- We can estimate the parameters (mean and variance) by these observations
- Using $\sigma \times randn(\,) + m$, we can generate new samples



$\Rightarrow p_{\text{model}}(x) \Rightarrow$

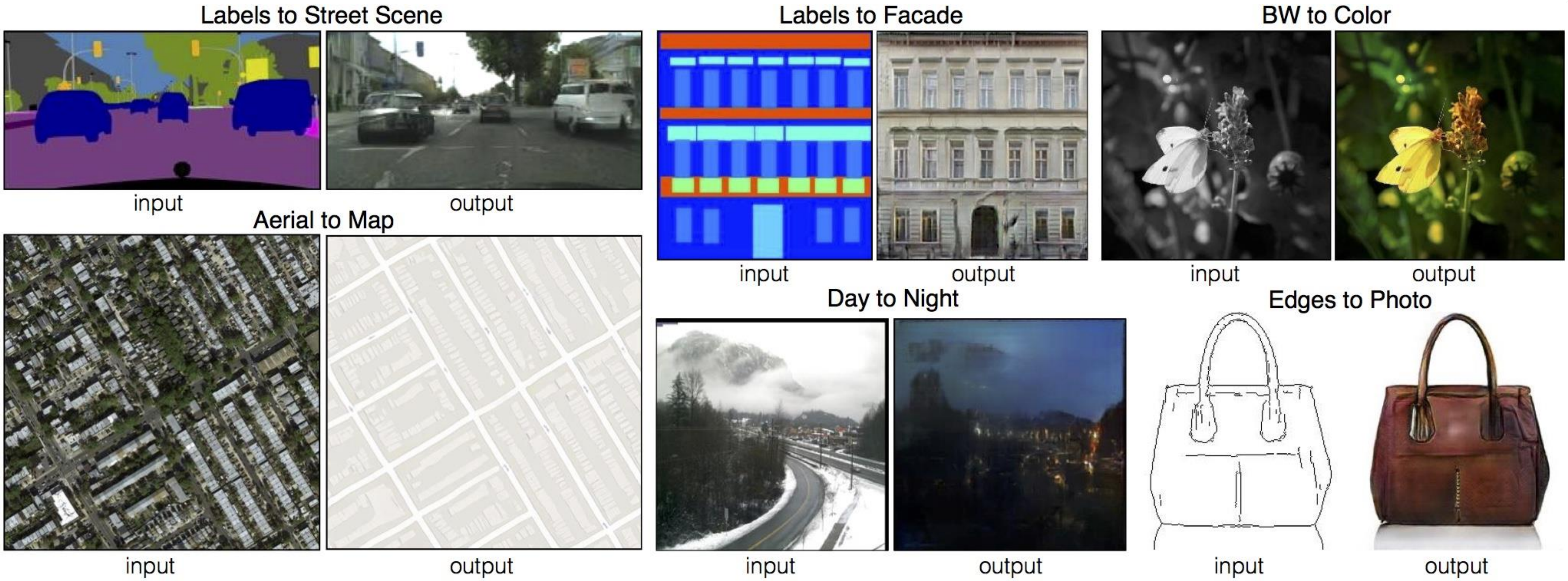




training data $\sim p_{\text{data}}(x)$

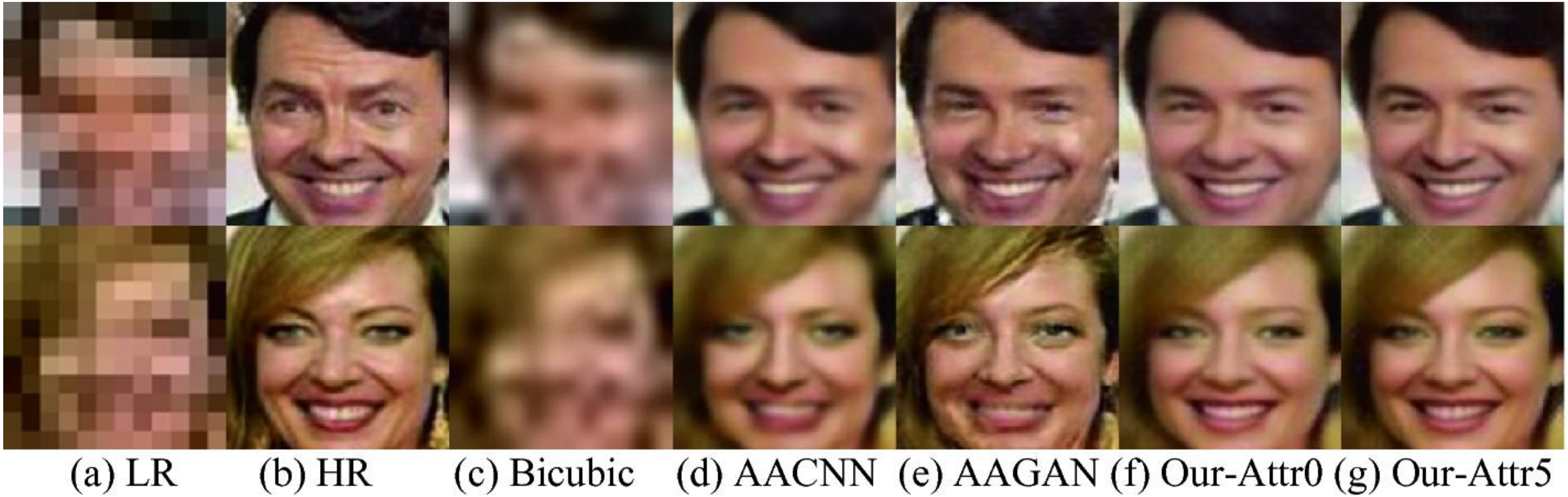
generated samples $\sim p_{\text{model}}(x)$

Generative models - applications



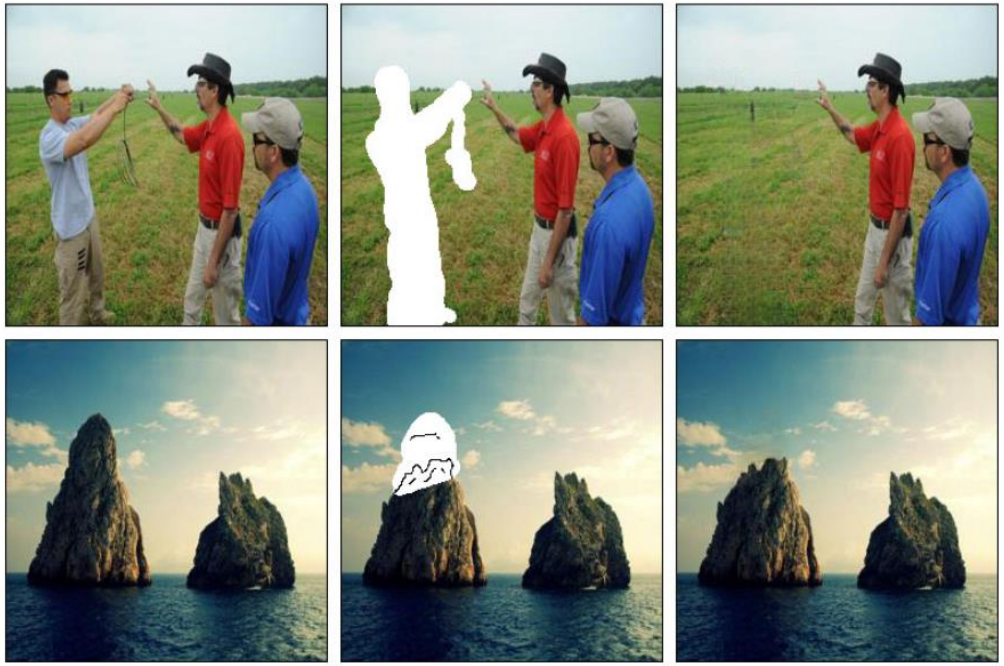
Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." *IEEE conference on computer vision and pattern recognition*. (2017).

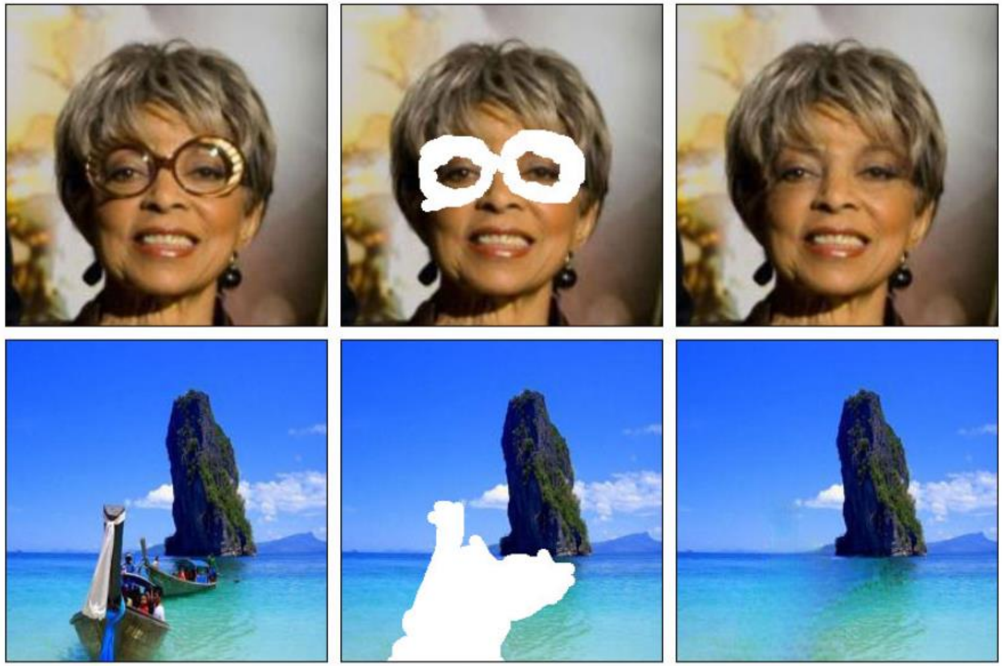
Generative models - applications



Dong, Ning, et al. "Face Super-Resolution via Discriminative-Attributes." *Chinese Conference on Pattern Recognition and Computer Vision*, (2019).

Generative models - applications





Nazeri, Kamyar, et al. "Edgeconnect: Generative image inpainting with adversarial edge learning." *arXiv preprint arXiv:1901.00212* (2019).