We trained 2 types of models:

- Models that directly receive the game state as input (namely the distance on the x and y coordinates between the bird and the pipe)
- Models that receive the game image as input

Models that receive the game image as input

1) Learning algorithm

   We divide our algorithm into “iterations”. During each iteration we play one game (based on the current q-network) and append all the decisions that we made to the replay buffer in the (state, action_taken, reward, nxt_state) format. Next, we randomly choose **batch_size** such moments, compute the update for these moments based on the QLearning formula and run one gradient descent step on these new values.

2) Image preprocessing

   To make the model converge faster we applied the following operations on the image:
   a) We removed the background
   b) We applied grayscaling to compress the 3 channels (R, G, B) into 1
   c) We applied a threshold filter such that only the pipes, the ground and the bird are made of 1s while everything else is set to 0
   d) We cut the image such that only a little is visible to the left of the bird, you can only see the next pipe to the right and the ground is no longer visible
   e) We finally downsized the image
   f) The final size is (14, 31, 1)

3) Architecture

```
input_dim = (14, 31, 1)
output_dim = 2

max_score_all = 0

def create_q_network(input_dim, output_dim):
    inputs = Input(shape=input_dim)

    x = Conv2D(8, kernel_size=3, activation='relu', strides=2)(inputs)
    x = Conv2D(8, kernel_size=3, activation='relu', strides=2)(x)

    x = Flatten()(x)

    x = Dense(32, activation='relu')(x)  # Smaller dense layer
    x = Dense(32, activation='relu')(x)

    outputs = Dense(output_dim, activation='linear')(x)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer="adam", loss='mse')

    return model
```

We used 2 convolutional layers to extract features from the input, followed by 2 dense layers. We don't need too big of a model because the input is already quite small and preprocessed.

4) Hyperparameters

gamma = 0.9 -> the factor by which the reward is multiplied when it is propagated to earlier states, taken from literature

epsilon = 0.8 -> The percentage of choices that are chosen randomly (initially 80%)
epsilon_min = 0.1 -> The minimum epsilon, chosen such that the model keeps exploring even in the later iterations
epsilon_decay = 0.995 -> The factor by which epsilon is multiplied during each iteration

batch_size = 128 -> Number of moments which we train at each iteration
max_memory = 20000 -> Maximum number of moments that are kept in the replay buffer at any time.

5) Experimentation attempts

- We tried training an autoencoder model that encoded each image into a latent space. In the end it proved hard to create a big enough database to train such a model while maintaining quality (for example not over representing the starting frames)
- We tried training a model that from a frame computed the distances from the bird to the next pipe gap. This proved hard for similar reasons as above.
- Changing the architecture
    - Adding convolutional layers
    - Increasing\decreasing the size of the dense layers

6) Results

The best score we achieved with this approach was **5**.

These are the scores from 10 consecutive runs:

0 3 4 0 1 0 3 0 2 2

Models that directly receive the game state as input

The learning algorithm and the hyperparameters are the same as above.

Architecture:

```
def create_q_network(input_dim, output_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        #Dense(128, activation='relu'),
        Dense(32, activation='relu'),
        Dense(32, activation='relu'),
        Dense(output_dim, activation='linear')
    ])
    model.compile(optimizer="adam", loss='mse')
    return model
```

Because the input is very small we only need 2 dense layers with 32 nodes each.

Experimentation attempts

We tried passing the last 4 frames as the state instead of just the current one, but this did not improve the performance.

Results:

The best score we achieved with this approach was **44**.

These are the scores from 10 consecutive runs:

1 4 1 7 1 11 6 8 32 9