
# Applied Machine Learning Final Project

## Using CNN and DQN to play Sekiro

Haochun Sun, Hao Zhang,
Hau Lam Fong

UNIVERSITY OF COPENHAGEN

# Inspiration:
# Using Machine Learning to play Super Mario

- A simple interface
- Two main keys: up and down

# Why Sekiro

- Resurrection in situ
- Using the modifier to facilitate neural network training
- Using script to implement code automation operation

- CNN METHOD
- GOOGLENET
- DQN
- DISCUSSION

# CNN method

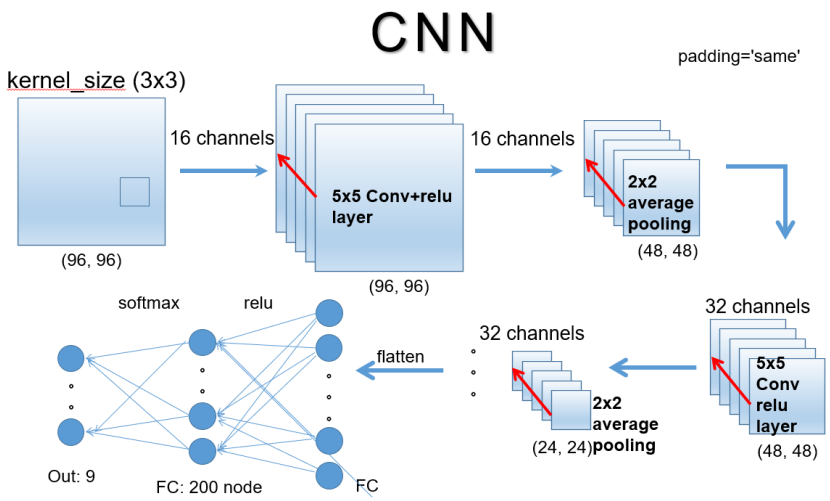
- Image recognition based on CNN network
- Input: the gray value matrix of each frame of the game screen
- Label: The key pressed when the corresponding screen appears
- A typical multi-classification problem:

# Why CNN?

- Feature recognition
- Low time cost
- Feature Preserve

- Only first word of slide title has initial capital letter (except proper nouns)
- Don’t reduce this standard text size
  - Second level bullet point
    - Third level bullet point
- No punctuation usually needed in bullet points

# CNN model



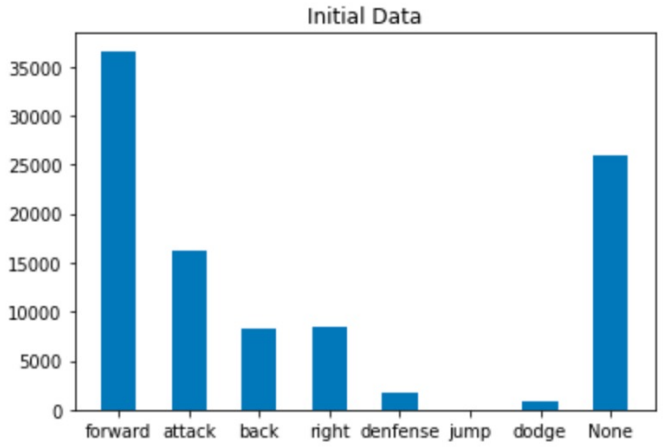

- Sequential model of tflearn.keras
- LeNet structure
- 5 conv2D layers, 3 fully connected layers
- BatchNormalization

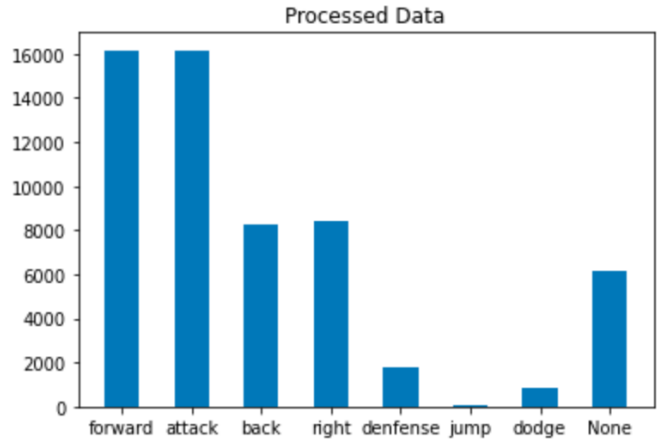
# Mark Dataset

**Defense**

**Attack**

# Data Reduction




- W: forward
- A: left
- S: right
- D: back
- H: attack
- J: defense
- K: jump
- L: dodge
- None: do nothing

# Sample

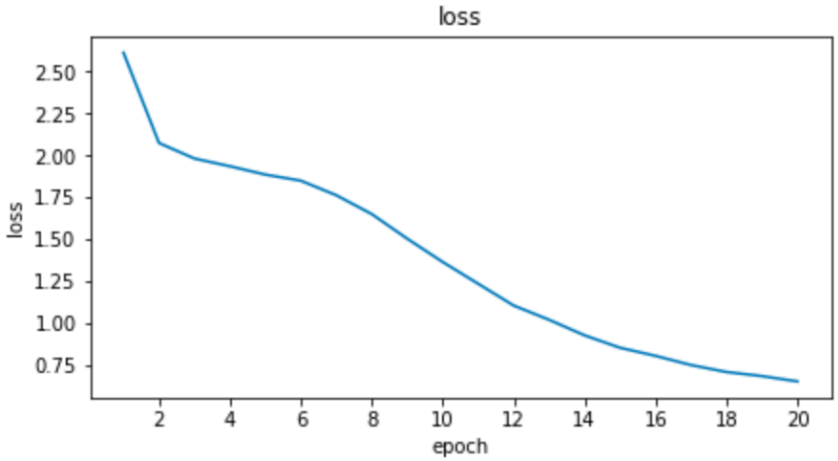
right

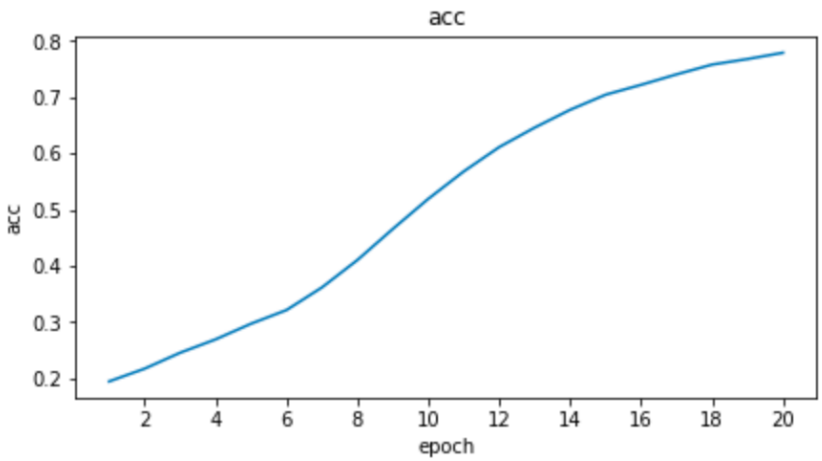
```
H
loop took 0.13663601875305176 second
Do nothing
loop took 0.09574270248413086 second
D
loop took 0.223402738571167 seconds
D
loop took 0.24888277053833008 second
D
loop took 0.2077794075012207 seconds
A
loop took 0.2103123664855957 seconds
```

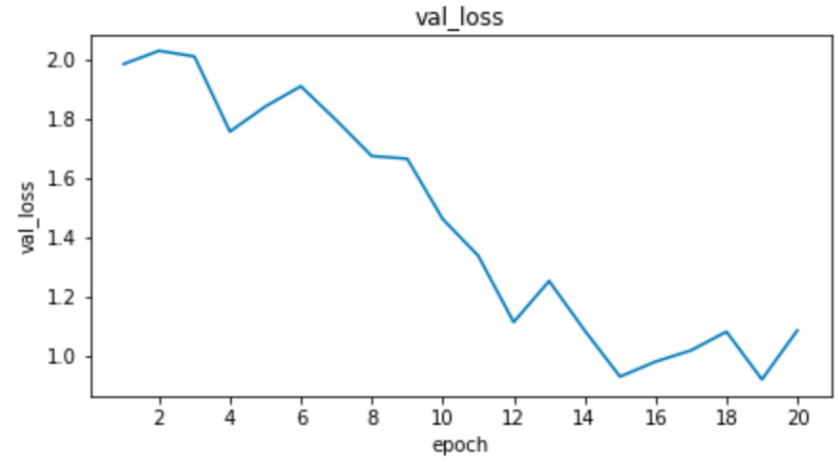

attack

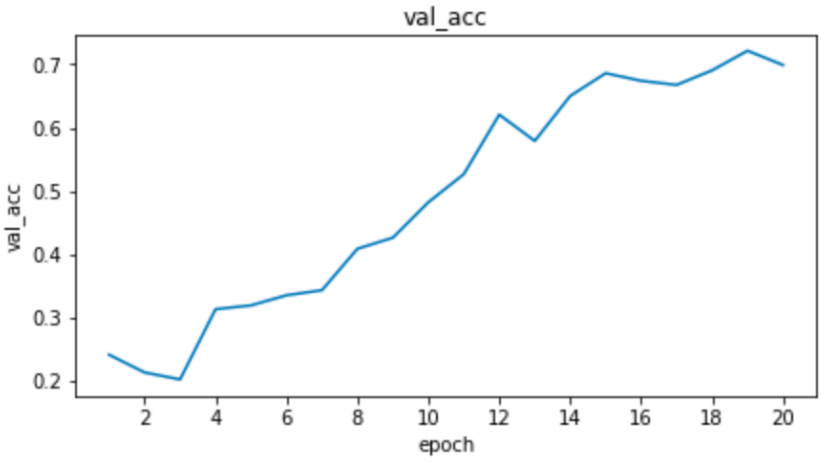
```
Do nothing
loop took 0.0498661994934082 secon
s
J
loop took 0.10682082176208496 seco
ds
J
loop took 0.1206820011138916 secon
s
J
loop took 0.1127016544342041 secon
s
```

# Performance of our model



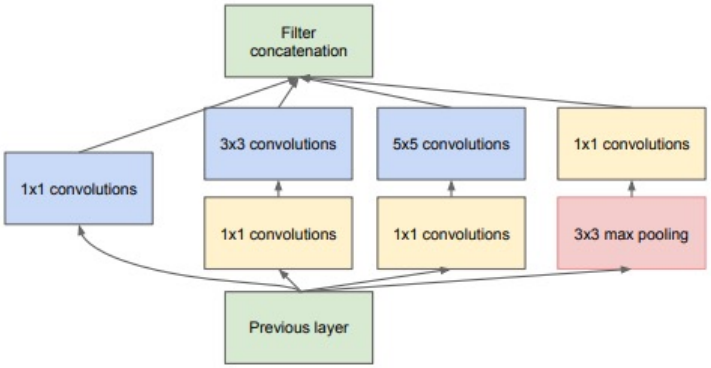




- The maximum val_accuracy is around 0.7


# GoogLeNet

# Inception Module




arXiv:1409.4842

- Combine 1x1, 3x3 and 5x5 convolution
- Using 1x1 convolution to reduce dimensions
- Alternative parallel pooling
- Concatenation

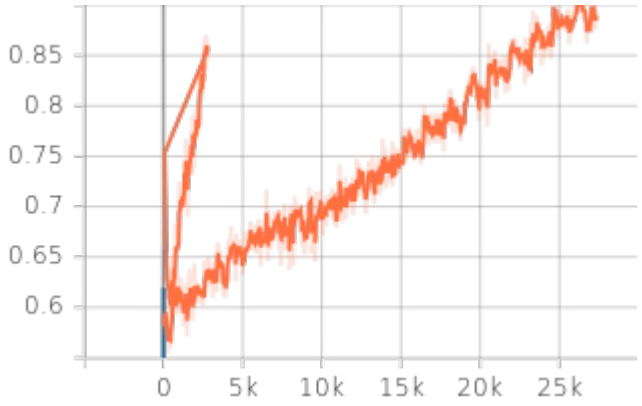
# Construction

| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

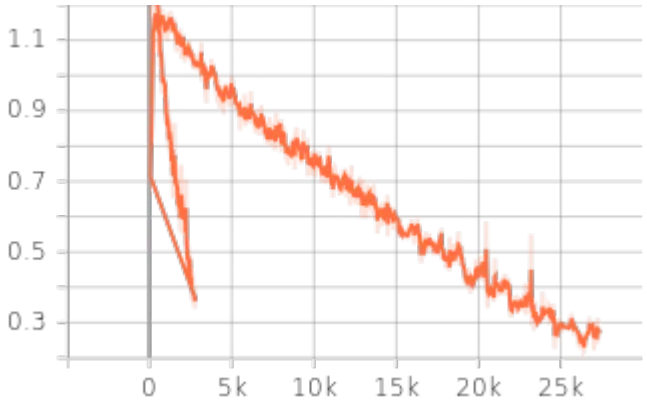
arXiv:1409.4842

- Inception
- Global average pooling
- ReLu activation functions
- Auxiliary Classifier

# Performance



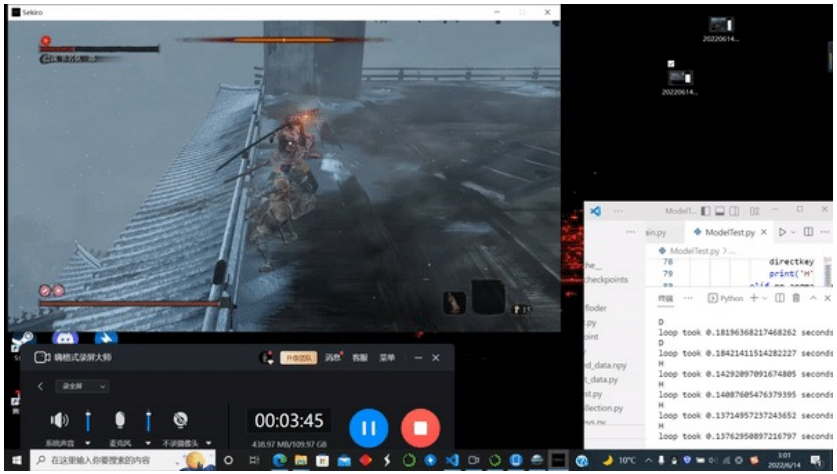

a) Accuracy with epoch=20




b) Entropy-loss with epoch=20

# FAST and LESS error!!!

# Simple CNN VS GoogLeNet



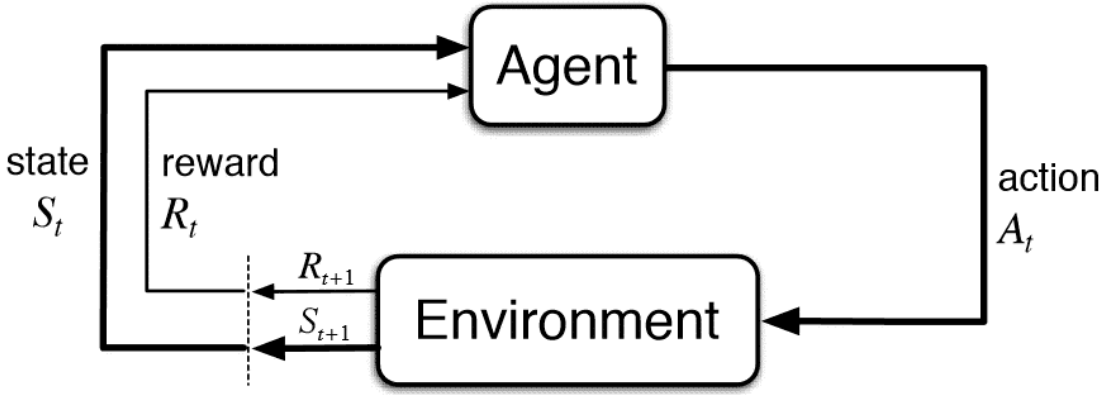


# Deep Q Network

UNIVERSITY OF COPENHAGEN

# Reinforcement learning

- Environment Observation value/status: State
- Action Select Policy: Policy
- Actions performed
- Reward
- The next state S′



# Station

Grayscale, 96*96

# Action

- Define initial_epsilon & final_epsilon

Jump

Dodge

Attack

Defense

# Q-learning

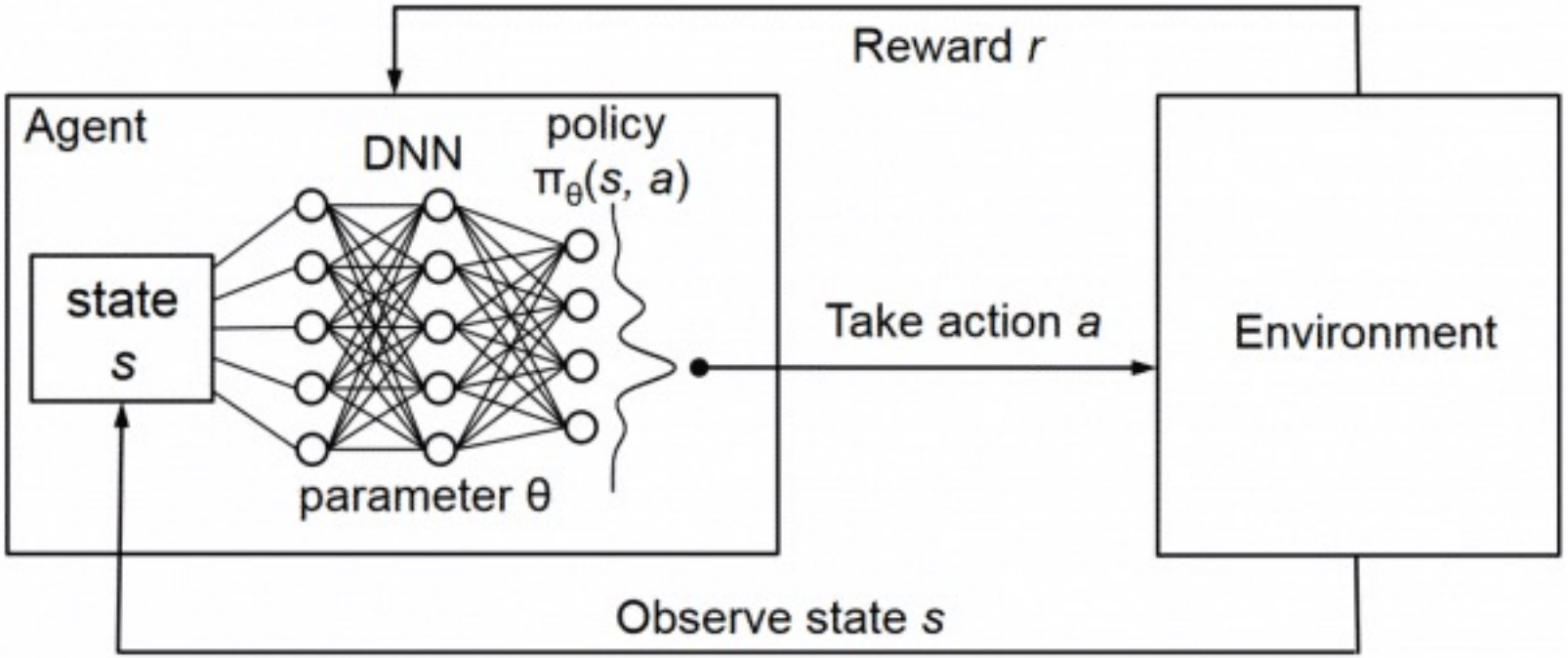
- Q(S, A)
- In state S, the sum of future rewards after taking action A.
- Q value update: S-A table

| | defense | attack | dodge | jump | nothing |
|---|---|---|---|---|---|
| s1 | 7 | -5 | 5 | 10 | 1 |
| s2 | -10 | 0 | 8 | -10 | -10 |
| s3 | 0 | 10 | 0 | 5 | 0 |
| …… | …… | …… | …… | …… | …… |

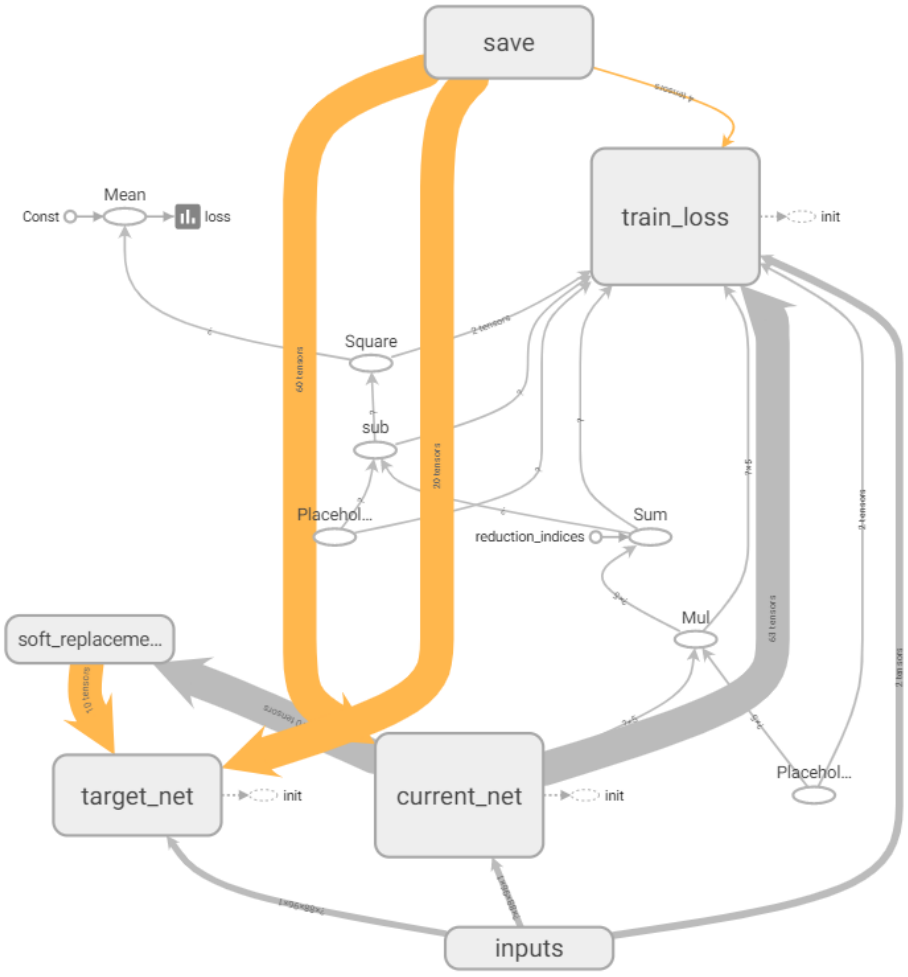
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r + \gamma \cdot max_a Q(s_{t+1}, a))$$

# Deep Q Network

- The combinations of states and actions are inexhaustible
- Deep neural network is used to approximate the optimal solution



# Graph


save
train_loss
init
Mean
Const
loss
Square
sub
Placehol...
Sum
reduction_indices
Mul
soft_replaceme...
target_net
init
current_net
init
Placehol...
inputs

# Game Interface

# Preprocessing

576*576

96*96

# Reward Mechanism

**Sekiro Hp Bar**


**Boss HP Bar**

**Boss Defense Bar**

# Episode Setting

- Define episode
- Boss die


# Reward Setting

```
def action_judge_shc(boss_blood, next_boss_blood, self_blood, next_self_blood, boss_bar,
next_boss_bar):
    if next_self_blood <= 0:
        done = 1 #next boss
        reward = -20 #sekiro die
    else:
        if next_boss_blood - boss_blood >30:
            reward = 200 #boss die
        else:
            if next_boss_blood == boss_blood :
                if next_self_blood == self_blood:
                    if next_boss_bar - boss_bar > 2:
                        reward = 25 #attack bar
                    else:
                        reward = 1 #nothing happen or dodge
                else:
                    reward = -10 #sekiro is attacked
            elif next_boss_blood - boss_blood > -6:
                reward = 7 #jump
            else:
                reward = 40 #attack boss
        done = 0
        if reward != 1:
            print(reward)
    return reward, done
```

# Calculation speed

- GPU: RTX 2070 SUPER

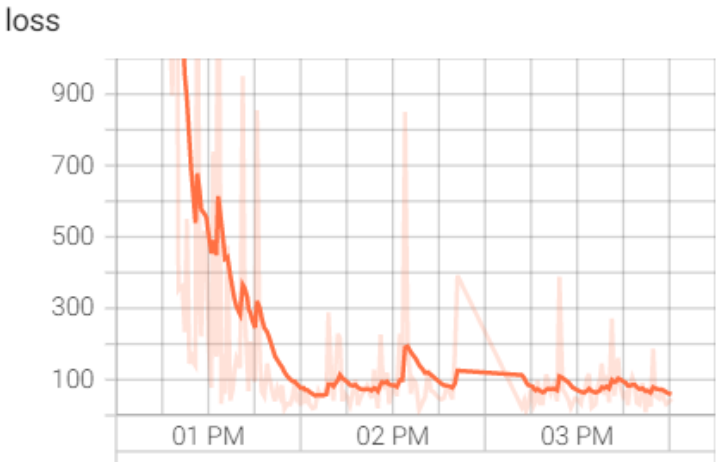
```
loop 5 took 0.10202240943908691 seconds
loop 6 took 0.10402512550354004 seconds
loop 7 took 0.09702205657958984 seconds
loop 8 took 0.0890195369720459 seconds
loop 9 took 0.09902238845825195 seconds
loop 10 took 0.09702205657958984 seconds
loop 11 took 0.09002065658569336 seconds
loop 12 took 0.09602212905883789 seconds
loop 13 took 0.04501008987426758 seconds
loop 14 took 0.10202407836914062 seconds
loop 15 took 0.11202478408813477 seconds
```

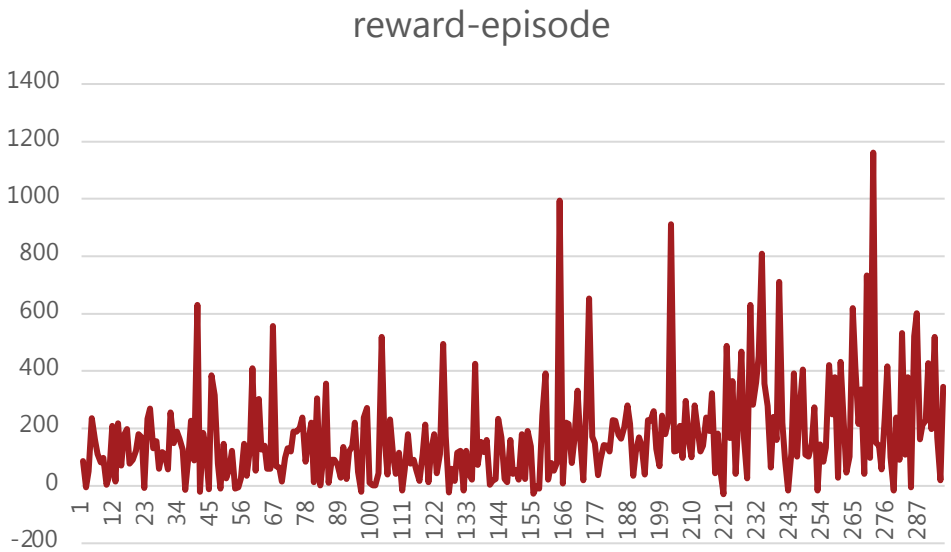
- CPU: i7-11800H

```
loop 36 takes 0.29449963569641113s
loop 37 takes 0.2924988269805908s
28.6
loop 38 takes 0.3020007610321045s
loop 39 takes 0.2762160301208496s
loop 40 takes 0.2312324047088623s
```

# Performance

# Performance

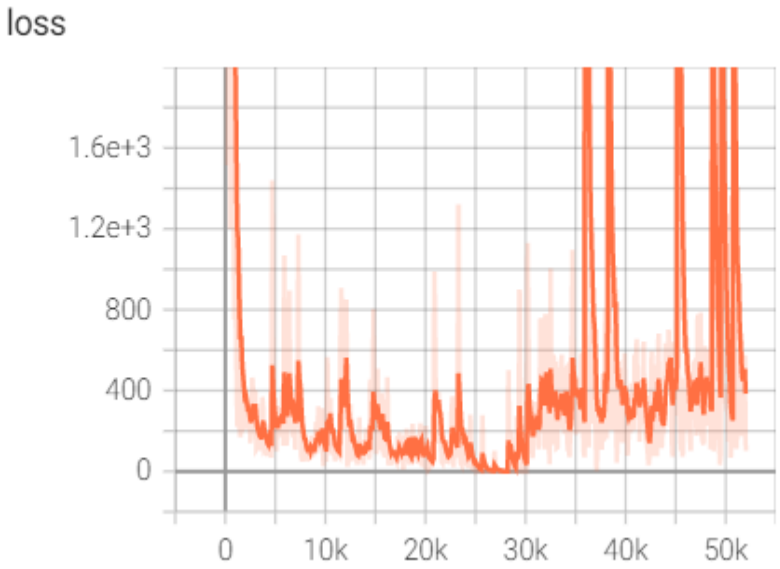


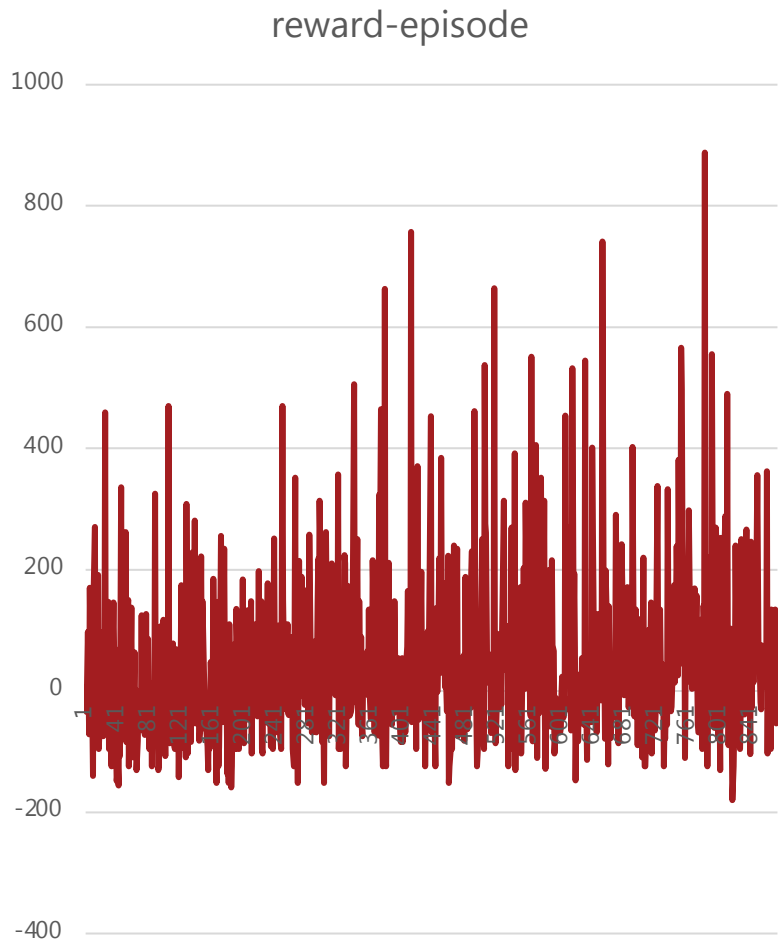



# Other Reward Setting

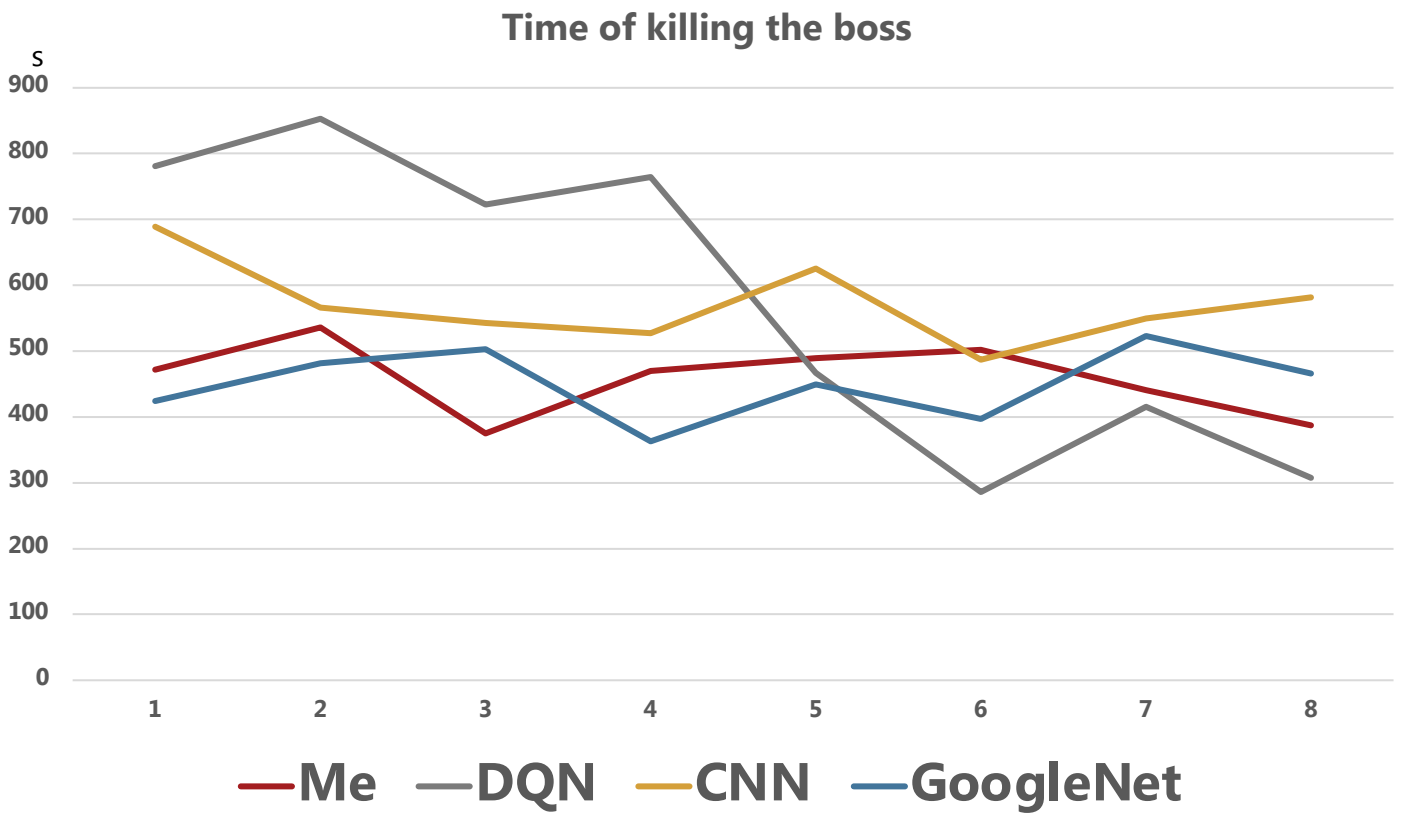
ATTACK reward is high

JUMP reward is high


loss
1.6e+3
1.2e+3
800
400
0
0
10k
20k
30k
40k
50k


reward-episode
1000
800
600
400
200
0
-200
-400

# Comparison


Time of killing the boss
S
900
800
700
600
500
400
300
200
100
0
1
2
3
4
5
6
7
8
Me
DQN
CNN
GoogleNet


# Discussion

# CNN Method

- BIG training data
- Expensive time
- BAD performance
- BUT you can play GAMES!

VS

# DQN Method

- SMALL training data
- Cheap time
- GOOD performance
- You CANNOT use your computer during training time!

# Improvement for CNN

- Excessive and efficient training data
- Use advanced CNN Model (GoogLeNet)
- Reduce NOISE

# Improvement for DQN

- ENOUGH training time
- GOOD reward setting
- Reduce NOISE

"

Prediction is very difficult, especially if it´s about the future.

Niels Bohr


# THANK YOU!

UNIVERSITY OF COPENHAGEN

# Appendix

- Check CNN Overfitting
- We noticed that the val_accuracy and val_loss of the CNN model fluctuated after epoch 15, and we were concerned that this was caused by overfitting. So we checked the performance of the model corresponding to epoch 15 and epoch 20 on 20,000 samples, and finally decided to use epoch 20.

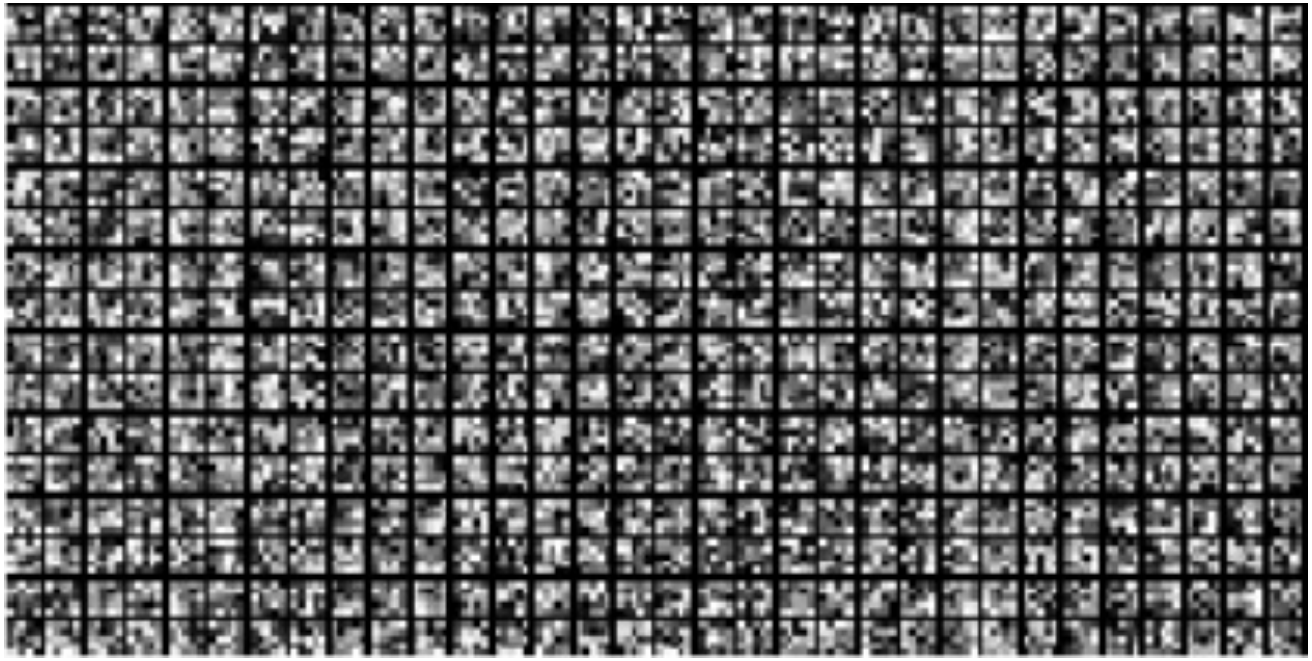
epoch 15

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| W | 0.90 | 0.81 | 0.85 | 4032 |
| A | 0.92 | 0.76 | 0.83 | 3956 |
| S | 0.81 | 0.88 | 0.84 | 1997 |
| D | 0.84 | 0.70 | 0.76 | 2116 |
| H | 0.50 | 0.70 | 0.59 | 3318 |
| J | 0.47 | 0.82 | 0.60 | 438 |
| K | 0.50 | 0.12 | 0.20 | 8 |
| L | 0.53 | 0.11 | 0.18 | 229 |
| None | 0.64 | 0.61 | 0.63 | 3906 |
| micro avg | 0.73 | 0.73 | 0.73 | 20000 |
| macro avg | 0.68 | 0.61 | 0.61 | 20000 |
| weighted avg | 0.76 | 0.73 | 0.74 | 20000 |
| samples avg | 0.73 | 0.73 | 0.73 | 20000 |

epoch 20

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| W | 0.87 | 0.85 | 0.86 | 4032 |
| A | 0.83 | 0.87 | 0.85 | 3956 |
| S | 0.78 | 0.92 | 0.84 | 1997 |
| D | 0.68 | 0.84 | 0.75 | 2116 |
| H | 0.62 | 0.69 | 0.66 | 3318 |
| J | 0.68 | 0.73 | 0.70 | 438 |
| K | 1.00 | 0.12 | 0.22 | 8 |
| L | 0.29 | 0.22 | 0.25 | 229 |
| None | 0.79 | 0.54 | 0.64 | 3906 |
| micro avg | 0.76 | 0.76 | 0.76 | 20000 |
| macro avg | 0.73 | 0.64 | 0.64 | 20000 |
| weighted avg | 0.77 | 0.76 | 0.76 | 20000 |
| samples avg | 0.76 | 0.76 | 0.76 | 20000 |

# Maxpooling



# Normalization

- After comparing the performance of BatchNormalization and LayerNormalization, we found that BatchNormalization takes less time and improves accuracy more.
- The ETA of one epoch when using BatchNormalization layer is around 3650 s.

```
1664/79314 [..............................] - ETA: 1:00:45 - loss: 11.8206 - acc: 0.1713
```

- The ETA of one epoch when using LayerNormalization layer is around 5100 s.

```
128/79314 [..............................] - ETA: 1:25:50 - loss: 3.2105 - acc: 0.0469
```

- At the same time, BatchNormalization gives a much batter accuracy.
- So we added BatchNormalization layer after each 2D convolutional layer.

# Reward setting

```
# def action_judge_shc(boss_blood, next_boss_blood, self_blood, next_self_blood, boss_bar, next_boss_bar, target):
#     if next_self_blood <= 0:
#         done = 1#next boss
#         reward = -10
#     else:
#         if next_boss_blood - boss_blood >30:
#             reward = 100
#         else:
#             if next_boss_blood == boss_blood :
#                 if next_self_blood == self_blood:
#                     if next_boss_bar - boss_bar > 2:
#                         reward = 10
#                     else:
#                         reward = 0#nothing happen or dodge
#                 else:
#                     reward = -10
#             elif next_boss_blood - boss_blood > -6:
#                 reward = 10
#             else:
#                 reward = 10
#         done = 0#boss alive
#         if reward != 0:
#             print(reward)
#     return reward, done
```

# Loop & Action Choice

```
agent.Store_Data(station, action, reward, next_station, done)
if len(agent.replay_buffer) > big_BATCH_SIZE:
    num_step += 1
    agent.Train_Network(big_BATCH_SIZE, num_step)
if target_step % UPDATE_STEP == 0:
    agent.Update_Target_Network()
```

```
def Choose_Action(self, state):
    Q_value = self.Q_value.eval(feed_dict={self.state_input: [state]})[0]
    if random.random() <= self.epsilon:
        self.epsilon -= (Initial_epsilon - Final_epsilon) / 2333
        return random.randint(0, self.action_dim - 1)
    else:
        self.epsilon -= (Initial_epsilon - Final_epsilon) / 2333
        return np.argmax(Q_value)
```