BAB II
TINJAUAN LITERATUR

2.1. Tinjauan Studi

Sebagai bahan referensi dalam penulisan laporan penelitian tugas akhir, dengan ini akan dipaparkan hasil dari penelitian-penelitian terdahulu yang memiliki hubungan dengan objek permasalahan dan solusi dari permasalahan. Berikut adalah Tabel 2.1 yang berisikan tentang tinjauan studi dari penelitian terdahulu :

Tabel 2.1 Rekapitulai Hasil Penelitian Terdahulu

No.	Nama Penulis	Masalah	Tahun	Metode	Hasil
1	Inseok oh [5]	Menjawab tantangan penarapan metode *Deep* RL pada *game* yang lebih kompleks	2019	*Actor-Critic* menggunakan kurikulum *self-play dan Deep Neural Network*	Pengembangan *Agent AI* pada *game* "Blade & Soul" yang telah melawan lima pemain profesional dan menghasilkan *winrate* 62%
2	Yu-Jhe Li [9]	Menginvestigasi peforma metode *Deep* RL pada 2.5D *fighting games*, dimana memiliki ambiguitas dalam penampilan visual seperti tinggi atau kedalaman karakter serta aksi yang dapat dieksekusi secara berurutan.	2018	*Asynchronous Advantage Actor-Critic* (A3C), Recurrent Info Network, dan CNN	Pengembangan *Agent* AI pada *game* "Litle Fighter 2" dengan menggunakan metode A3C yang telah dimodifikasi menggunakan LSTM. Mendapatkan *winrate* sebesar 95.5% pada mode *basic* dan 95.9% pada mode *advanced*.

No.	Nama Penulis	Masalah	Tahun	Metode	Hasil
3	Vlad Firoiu [11]	Menginvestigasi peforma metode *Deep* RL pada game *Super Smash Bros. Melee* (SSBM)	2017	*Actor-Critic, Deep Q-Learning, Deep Neural Network,* dan *Self-Play Curriculum*	Pengembangan *Agent* AI pada *game* "Super Smash Bros. Melee". Dari penelitian ini didapati kalau *Policy Gradient Method* lebih baik dari *Q-Learning Method* dalam kurikulum *self-play*.
4	Matheus R. F. Mendonc [13]	Meningkatnya permintaan pemain video *games* untuk menyediakan games yang lebih realistis baik itu dari sisi grafis dan dari sisi *agent* yang dapat berperilaku seperti manusia	2015	*Q-Learning Reinforcement* Learning, dan *Artificial Neural Network*	Pengembangan *Agent* AI pada *game* "Boxer".
5	Dae-Wook Kim [14]	Memverifikasi apakah algoritma RL dengan kurikulum *self-play* dapat mencapai kinerja yang maksimal dalam game 1-vs-1 seperti FightingICE	2020	*Proximal policy optimization* (PPO), *Monte Carle Tree Search* (MCTS), dan ANN	Pengembangan *Agent* AI pada kompetisi FightingICE dan menghasilkan *winrate* sebesar 94.4%

Pada penelitian ini [5] dilakukan pengembangan Agent AI dalam permainan "Blade & Soul" menggunakan metode Deep RL yang mengandung juga

kurikulum novel self-play dan teknik Data Skipping. Melalui kurikulum self-play, dibuat 3 Agent AI dengan gaya bermain yang berbeda-beda yaitu “Aggressive”, “Balanced”, dan “Defensive”. Dalam penelitian ini peneliti menggunakan 3 *reward* yaitu *time penalty, healt potion ratio,* dan *distance* penalty. Hasil dari penelitian ini adalah Agent dengan gaya bermain “Aggressive” mempunyai winrate sebesar 92%, Agent dengan gaya bermain “Balanced” mempunyai winrate sebesar 42%, dan Agent dengan gaya bermain “Defensive” mempunyai winrate sebesar 50%.

Pada penelitian yang dilakukan oleh Yu-Jhe Li pada tahun 2018 [9], ia dan kawan-kawannya mengembangkan sebuah Agent AI pada permainan “Litle Fighter 2”. Dimana pada penelitian ini mereka menggunakan metode RL *Asynchronous Advantage Actor-Critic* (A3C) serta untuk neural networknya mereka menggambungkan CNN dengan *Recurrent Info Network*. *Reward* yang digunakan dalam penelitian ini terbagi dalam dua kategori, yaitu positif dan negatif reward. Ketika Agent berhasil memukul lawan maka Agent akan mendapatkan *reward* positif, sedangkan ketika *Agent* terkena pukulan lawan maka akan menerima reward negatif. Hasil yang didapatkan pada penelitian ini adalah winrate sebesar 95.5% pada mode “basic” dan 95.9% pada mode “Advanced”.

Pada penelitian ini [11] dilakukan pengembangan Agent AI dalam permainan “Super Smash Bros. Melee (SSBM)” menggunakan metode Deep RL. Jenis

RL yang digunakan dalam penelitian ini ada Policy Gradient Reinforcement Learning dan Q-Learning *Reinforcement Learning*. Pada penelitian bertujuan untuk mencari tahu mana metode terbaik diantara kedua jenis RL untuk permainan SSBM. Hasil yang didapatkan adalah bahwa metode *Policy Gradient* RL lebih baik dibandingkan dengan *Q-Learning* RL dalam kurikulum *self-play* atau dalam berlatih dengan jaringan buatan lain yang sedang berlatih juga.

Pada penelitian ini [13] dilakukan pengembangan Agent AI menggunakan metode *Deep* RL pada game "Boxer". RL yang digunakan dalam penelitian ini adalah Q-Learning. Pada penelitian ini bertujuan untuk menciptakan *Agent* AI yang mempunyai kemampuan bermain seperti manusia pada umumnya. *Reward* yang digunakan dalam penelitian ini terbagi kedalam 3 jenis, yaitu "*Reward by Victory*" dimana *Agent* akan mendapatkan *reward* sebesar 1 jika berhasil memenangkan sebuah pertandingan dan akan mendapatkan *reward* sebesar -1 jika kalah dalam pertandingan. Lalu yang kedua adalah "*Reward by Hit*" yaitu *reward* berdasarkan jenis pukulan yang diluncurkan dan yang diterima oleh *Agent*, jika berhasil meluncurkan pukulan ke lawan maka akan mendapatkan *reward* positif, sedangkan jika menerima pukulan dari lawan akan mendapatkan *reward* negatif. Pada penelitian ini tidak diberitahu berapa *winrate* yang dihasilkan oleh *Agent* yang telah dikembangkan.

Pada penelitian ini [14] dilakukan pengembangan Agent AI pada kompetisi "FightingICE" menggunakan *Deep* RL (*Proximal policy optimization* (PPO)) dan *Monte Carle Tree Search* (MCTS). *Reward* yang digunakan dalam penelitian ini adalah yang pertama ketika Agent memukul lawan maka Agent akan mendapatkan *reward* positif, dan begitupun sebaliknya. Lalu yang kedua, jika Agent memenangkan pertandingan maka ia akan mendapatkan *reward* sebesar +10, jika seri akan mendapatkan *reward* sebesar 0, dan jika kalah akan mendapatkan *reward* sebesar -10. Hasil yang didapatkan pada penelitian ini adalah *winrate* sebesar 94.4%.

Dari seluruh penelitian terkait, yang dapat menjadi acuan dalam penelitan yang akan dilakukan adalah dengan menggunakan metode *Deep Q Learning* dan *Actor-Critic* sehingga dapat menghasilkan *agent* AI yang dapat menjadikan pengalaman bermain menjadi lebih baik.

2.2. Landasan Teori

2.2.1. *Fighting Games*

Fighting games adalah salah satu gim aksi dimana dua karakter di layar terlibat dalam suatu pertempuran satu lawan satu. Fighting game sering kali menampilkan pertarungan tidak bersenjata, seperti tinju dan martial arts, tetapi juga dapat mencakup pertarungan dengan senjata seperti pedang atau senapan. Pemain diberi opsi untuk mengontrol karakter di layar dan terlibat

dalam sebuah pertarungan jarak dekat dengan lawan. Karakter dalam fighting game sering kali memiliki gerakat seni bela diri dan kekuatan ekstrim yang dilebih-lebihkan. Struktur gameplay dari fighting games biasanya mencakup beberapa round per pertarungan. Agar efektif dalam bermain, pemain sering kali perlu menguasai beberapa teknik seperti kombo, blocking, dan serangan balik. Kombo berarti merangkai serangkaian serangan [10]. Pada saat ini banyak sekali *fighting games* yang beredar pada marketplace gim seperti Steam, Uplay, dan GOG. Salah satu *fighting games* yang tertua dan menjadi salah satu game yang mempopulerkan genre *fighting games* itu sendiri adalah "Street Fighter II".

2.2.1.1. OpenAI Gym Retro

OpenAI Gym Retro adalah sebuah platform yang dapat digunakan untuk riset reinforcement learning pada games. Dalam platform ini terdapat 70 Atari games dan 30 Sega games menjadi lebih dari 1,000 game di berbagai emulator. OpenAI Gym Retro biasanya digunakan untuk melakukan penelitian tentang algoritma RL. Dengan OpenAI Gym Retro, kita dapat mempelajari kemampuan untuk menggeneralisasi antara game dengan konsep yang mirip namun dengan tampilan yang berbeda [15]. OpenAI Gym Retro menginzinkan kita mengubah video games klasik menjadi sebuh *environment* untuk algoritma RL. Dalam OpenAI Gym Retro kita akan mendapatkan 1000 lebih games klasik dari berbagai konsol seperti Sega Genesis dan Sega Master

System, dan Nintendo NES, SNES, dan Game Boy, dan juga dukungan awal untuk Sega Game Gear, Nintendo Game Boy Color, Nintendo Game Boy Advance, dan NEC TurboGrafx.

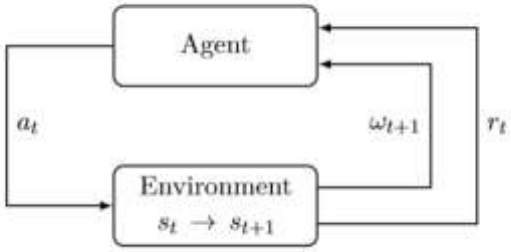
2.2.1.2. *Street Fighter II*

"*Street Fighter* II" adalah *game* yang dirilis pada 6 Februari tahun 1991 . Pada awal perilisan game ini telah laku terjual sebanyak 60.000 kabinet di seluruh dunia [16]. *Game* ini disebut-sebut sebagai *game* yang memulai kebangkitan *fighting game* pada *arcade* di tahun 1990-an. Pada *game* ini terdapat 8 karakter yang bisa dimainkan, karakter yang paling sering digunakan adalah Ryu, yang dimana gambarnya terletak pada *artwork* pada *Game* ini juga yang membuat pergeseran pada dunia *game* kompetitif pada saat itu, dimana sebelum *street fighter* II muncul *high score* digunakan sebagai penentu pemain yang terbaik, tetapi ketika *street fighter* II muncul *game* ini mengizinkan pemain untuk berhadapan satu lawan satu dalam sebuah kompetisi untuk menentukan siapa yang terbaik diantara mereka.

2.2.2. *Deep Reinforcement Learning*

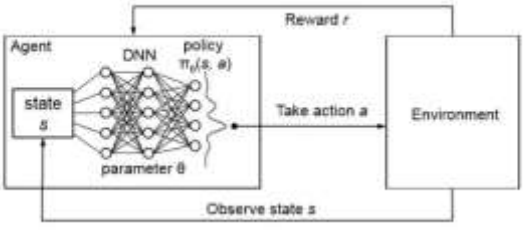
Reinforcement Learning (RL) adalah sebuah cabang dari *machine learning* dimana sebuah *agent* akan belajar dengan cara berinteraksi dengan *environtment*. Sebuah kerangka kerja RL akan mengizinkan *agent* untuk belajar dengan metode *trial* dan *error*. RL *agent* akan menerima *reward* dari aksi yang dilakukan dan tujuan dari proses pembelajaran dari RL adalah untuk

memilih aksi yang dapat memaksimalkan jumlah *reward* dalam setiap waktu. Dengan kata lain, *agent*, dengan mengamati hasil dari tindakan yang diambilnya terhadap *environment*, dan mencoba mempelajari urutan aksi yang optimal untuk dieksekusi dalam mencapai tujuannya [17]. Permasalahan dalam RL dapat kita formulasikan sebagai *discrete-time stochastic control process* dimana *agent* akan berinteraksi dengan lingkungannya dengan cara yang digambarkan pada Gambar 2.2.



Gambar 2.1 Interaksi *agent-environment* pada RL [17]

Pada Gambar 2.2 kita bisa lihat bahwa pertama *agent* akan mengirimkan aksi (a_t) kepada *environment* dalam sebuah state (s_t) dan state akan berpindah ke state selanjutnya (s_{t+1}), lalu *agent* akan mendapatkan *reward* (r_t) dari aksi tersebut, kemudian *agent* akan melakukan observasi (w_{t+1}).



Gambar 2.2. Arsitektur *Deep* RL [18].

Deep reinforcement learning adalah sebuah kombinasi dari *reinforcement learning* (RL) and *deep learning* [19]. Dimana setiap *state* akan menjadi sebuah masukkan dari suatu *Artificial Neural Network* dan kemudian akan menghasilkan keluaran berupa aksi-aksi yang bisa dilakukan pada sebuah state. Pada pengimplementasiannya terdapat beberapa algoritma *deep* RL diantaranya adalah *Deep Q-Learning* dan *Actor-Critic*. Sehingga jika digambarkan akan menjadi seperti Gambar 2.2.

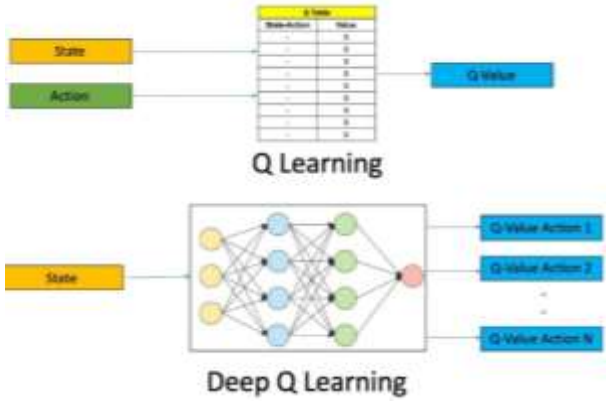
2.2.2.1. *Deep Q-Learning*

Algoritma *Q-Learning* bekerja dengan mempertimbangkan sekumpulan *state* (S) yang menggambarkan kemungkinan *states* dari *agent*, sekumpulan aksi (A) yang dapat dilakukan pada setiap *state* yang ada, dan *reward function* ($\pi(s, a)$) yang terkait dengan aksi ($a \in$ A) yang dipilih oleh *agent* ketika berada pada *state* (s $\in$ S). Setiap tuple (s, a) berhubungan dengan *Q-Value*, dimana *Q-Value* adalah hal yang dipakai untuk mengukur kualitas dari aksi a ketika dieksekusi pada *state* s. Nilai ini disimpan dalam tabel yang disebut denga *Q-Table* [13]. Kapanpun ketika sebuah aksi a dieksekusi dalam sebuah *state* s, maka *reward* akan dikalkulasikan melalui $\pi(s, a)$ dan *Q-Value* Q(s, a) akan diperbaharui berdasarkan nilai yang sesungguhnya. Fungsi pembaharuan dari *Q-Learning* dapat digambarkan seperti ini:

$$Q(s,a) = (1- \alpha)Q(s,a) + \alpha(r + \gamma\, max(Q(s', a'))) \quad (2.2.1)$$

Fungsi pembaharuan dari *Q-Learning* memperhitungkan dua variabel, yaitu *learning rate* yang dilambangkan dengan α, dan *discount rate*, yang dilambangkan dengan γ. *Learning rate* akan disetel dengan nilai dari 0 sampai 1. Menyetelnya dengan nilai 0 itu berarti *Q-Values* tidak pernah diperbaharui, sehingga tidak ada hal yang dipelajar. Jika menetapkannya dengan nilai tinggi seperti 0.9 maka berarti pembelajaran dapat terjadi dengan cepat. *Discount rate* menentukan nilai dari *reward* di masa depan. Semakin kecil nilai γ, maka agen akan semakin mementingkan *reward* pada waktu terdekat, bukan *reward* di masa depan.

Deep Q-Learning adalah sebuah algoritma penggabungan antara *Artificial Neural Network* (ANN) dan *Q-Learning* [18]. Pada *Deep Q-Learning* penentuan nilai *Q-Value* dilakukan dengan menggunakan ANN. Jika digambarkan perbedaan antara *Q-Learning* dan *Deep Q-Learning* maka bisa dilihat pada Gambar 2.3.



Gambar 2.3. Perbedaan *Q-Learning* dan *Deep Q-Learning* [20].

Pada algoritma *Q-Learning* seluruh kemungkinan *Q-Value* akan disimpan pada sebuah tabel yang dinamakan dengan *Q-Table*. Akan tetapi hal ini sangat tidak efektif jika kita berhadapan dengan jumlah kemungkinan aksi yang sangat besar, hal ini akan membuat ukuran dari *Q-Table* akan sangat besar sekali. Dalam algoritma *Deep Q-Learning* kondisi *state* saat ini akan menjadi input dari ANN dan kemudian akan menghasilkan sekumpulan *Q-Value* dari sekumpulan aksi, dimana nantinya *agent* akan memilih aksi dengan tiga cara, yaitu secara eksploitasi, eksplorasi, atau menggunakan metode *epsilon greedy exploration*. Eksplorasi, dimana membiarkan *agent* untuk melakukan eksplorasi terhadap tiap-tiap *action* dan diharapkan *agent* dapat menemukan informasi lebih detail tentang *action* yang akan ia ambil. Pendekatan ini memungkinkan *agent* untuk mengambil keputusan yang lebih tepat di masa depan. Pendekatan yang kedua adalah ekploitasi, dimana *agent* akan memilih secara *greedy*, dimana ia akan memilih *action* dengan *Q-Value* paling besar.

Lalu yang ketiga adalah dengan menggunakan *epsilon greedy exploration*, dimana metode ini akan menyeimbangkan antara ekplorasi dan eksploitasi.

$$\text{Action pada waktu (t)} = \begin{cases} \max Q_t\,(ekploitasi) & : probabilitas\ 1-\in \\ random\ action\ (eksplorasi) & : probabilitas\ \in \end{cases}$$

Metode ini menggunakan nilai epsilon, dimana pada awal training nilai epsilon (∈) akan diset menjadi 1 dan membuat *agent* akan cenderung memilih pendekatan ekplorasi pada awal-awal *training*. Ini berguna agar *agent* dapat mempelajari *environment* yang ada. Lalu seiring berjalannya waktu nilai epsilon (∈) akan diset turun, sehingga ketika *agent* telah lebih paham akan *environment* maka pemilihan action akan diubah ke pendekatan eksploitasi.

Setelah melakukan pemilihan *action* maka *agent* akan mengeksekusi *action* tersebut dan kemudian ia akan mendapatkan reward dan berpindah ke *state* selanjutnya. Setelah itu akan dilakukan perhitungan *Loss* menggunakan sisi kanan persamaan bellman.

$$Loss\ = \left(R_t\ +\ \gamma\ max\left(Q(s', a')\right) -\ Q(s, a)\right)^2 \qquad (2.2.2)$$

Dimana R_{t+1} adalah *reward* yang didapat oleh *agent* pada *state* s+1, γ adalah faktor diskonto dan $max\left(Q(s', a')\right)$ adalah *Q-Value* paling besar yang didapatkan dari memasukkan state s+1 kedalam *main network.* Perhitungan *loss* dilakukan dengan mengurangkan target *Q-Value* dengan *Q-Value* yang

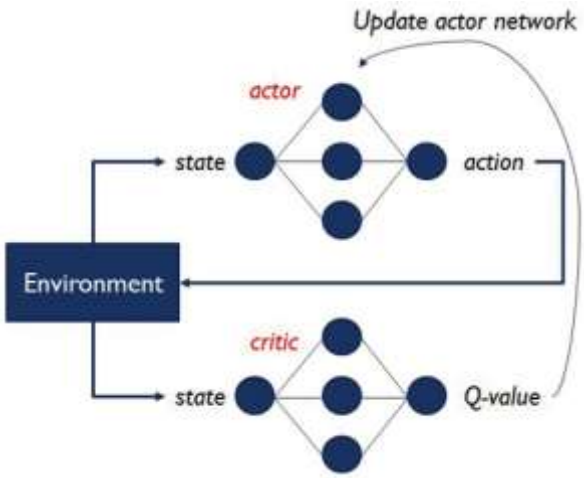
yang didapat dari *network* (Q(s,a)). *Loss* ini akan digunakan untuk mengupdate bobot dari *network* yang ada menggunakan *backpropagation* dan *stochastic gradient descent*, sehinga dapat diformulasikan bobot yang baru sebagai berikut :

$$W' = W - \alpha(\frac{\partial L}{\partial w}) \quad (2.2.3)$$

Dimana W'adalah bobot yang baru, W adalah bobot lama, α adalah learning rate yang digunakan, dan $\frac{\partial L}{\partial w}$ adalah *gradient* yang akan didapatkan pada proses *backpropagation* dengan mencari turunan parsial dari *loss* dan bobot pada *network*. Kemudian langkah ini diulang sampai beberapa episode.

2.2.2.2. *Actor-Critic*

Actor-critic merupakan jenis algoritma RL yang merupakan gabungan dari metode *policy based learning* dan *value based learning*. *Actor-critic* terdiri dari 2 komponen, yaitu *actor* dan *critic* [21]. Actor adalah komponen yang membentuk *policy*. sedangkan critic adalah komponen yang memberikan evaluasi terhadap *policy* yang dibuat oleh *actor*. Penggambaran jaringan *actor* dan *critic* bisa kita lihat pada Gambar 2.4.

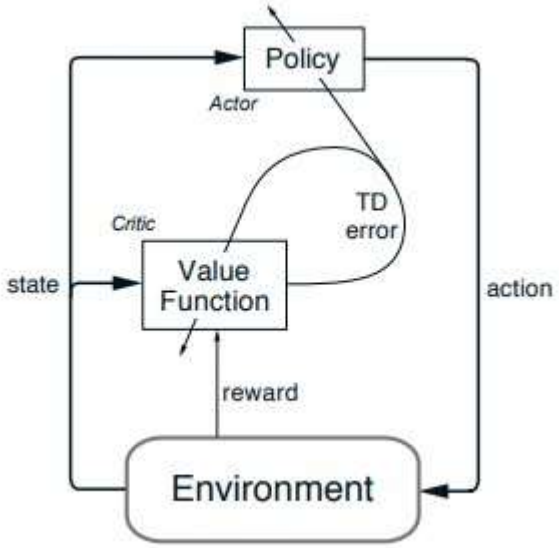


Gambar 2.4. Jaringan *Actor* dan *Critic*.

Metode *Actor–critic* adalah metode *Temporal difference* (TD) yang memiliki struktur memori yang terpisah yang bertujuan untuk menunjukkan secara ekspilisit bahwa *policy* yang tidak bergantung pada fungsi *value*. *Actor* dikenal sebagai elemen yang membentuk struktur dari *policy*, disebabkan *actor* bertugas untuk memilih aksi apa yang akan dilakukan oleh *agent* dalam *state* tertentu, dan sedangkan elemen yang bertugas untuk mengestimasi fungsi *value* adalah *critic*, disebabkan *critic* akan memberikan evaluasi terhadap aksi yang dibuat oleh *actor* [22]. Pembelajaran pada algoritma *actor-critic* akan selalu bergantung kepada *policy* yang dibuat oleh *actor*.

Proses yang terjadi pada algoritma *actor-critic* adalah, pertama *state* pada waktu t akan menjadi input untuk jaringan *actor* dan *critic*. Jaringan *actor* akan menghasilkan keluaran berupa *action* yang akan dieksekusi pada environtment. Lalu setelah itu *agent* akan mengeksekusi *action* tersebut.

Critic harus belajar dan mengevaluasi *policy* apapun yang saat ini diikuti oleh *actor*. *Critic* akan mengambil bentuk dari TD *error*. Nilai TD *error* ini adalah nilai skalar dari *critic* dan akan mendorong pembelajaran baik itu untuk *actor* maupun *critic,* seperti yang kita lihat pada Gambar 2.5.



Gambar 2.5. Arsitektur *Actor-Critic* [22].

Biasanya, *Critic* bisa dikatakan sebagai *state-value function*. Setelah setiap pemilihan aksi, *critic* akan mengevaluasi *state* selanjutnya untuk menentukan apakah segala sesuatunya berjalan lebih baik atau lebih buruk. Untuk menghitung *loss* yang didapat dari jaringan *critic* maka kita akan memakai formula TD *error,* sebagai berikut :

$$Loss = ((R_t + \gamma \mathrm{Vt}(S_{t+1})) - \mathrm{V}(\mathrm{St}))^2 \qquad (2.2.4)$$

Dimana Vt adalah *value funtion* yang diperoleh dari jaringan *critic* pada waktu *t*. Vt(S_{t+1}) adalah nilai *value function* maksimal pada waktu t+1, dan R_t adalah *reward* yang didapat oleh *agent* setelah melakukan *action* yang didapat dari jaringan *actor*. Setelah menghitung nilai *loss* maka akan dilakukan pengupdatean bobot pada jaringan *critic* menggunakan *stochastic gradient descent* dan *backpropagation*, sehinga dapat diformulasikan bobot yang baru dengan persamaan yang dapat dilihat pada persamaan 2.2.3

Setelah dilakukan pengupdatean bobot pada jaringan *critic*, maka jaringan *critic* sekali lagi akan memproses *state* saat waktu t untuk menghasilkan *value function* yang baru. Setelah mendapatkan *value function* yang baru, kita akan mengupdate bobot pada jaringan *actor* menggunakan *stochastic policy gradient* yang dapat dituliskan seperti ini :

$$\Delta\theta = \beta\ [\nabla\theta\ (log\ \pi w(s,a))Q\pi w\ (s,a)] \qquad (2.2.5)$$

Dimana $log\ \pi w(s,a)$ adalah nilai yang didapatkan dari jaringan *actor* pada *state* t untuk aksi a, dan $Q\pi w\ (s,a)$ adalah *value function* yang baru dan β adalah *learning rate* dari jaringan *actor*.

2.2.3. *Convolutional Neural Network*

Convolutional Neural Network (CNN) adalah pengembangan dari *Multilayer Perceptron* (MLP) yang didesain untuk mengolah data dua dimensi. CNN termasuk dalam jenis *Deep Neural Network* karena kedalaman jaringan yang tinggi dan banyak diaplikasikan pada data citra. Pada kasus klasifikasi citra, MLP kurang sesuai untuk digunakan karena tidak menyimpan informasi spasial dari data citra dan menganggap setiap piksel adalah fitur yang independen sehingga menghasilkan hasil yang kurang baik [23]. Sebuah CNN terdiri dari beberapa layer. Struktur CNN dapat dibagi menjadi 4 area [24] sebagai berikut:

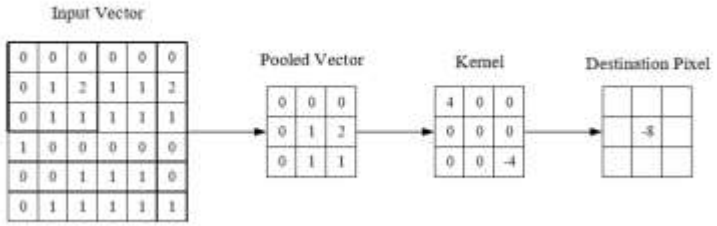
1. Input Layer

   Input layer adalah layer pertama dari arsitektur CNN dimana fungsi dari layer ini adalah menerima masukkan berupa pixel gambar.

2. *Convolutional* Layer

   Convolution Layer melakukan operasi konvolusi pada output dari layer sebelumnya. Konvolusi adalah suatu istilah matematis yang berati mengaplikasikan sebuah fungsi pada output fungsi lain secara berulang. Dalam pengolahan citra, konvolusi berarti mengaplikasikan sebuah *kernel* pada citra disemua offset yang memungkinkan seperti yang ditunjukkan pada Gambar 2.6. Input vector adalah citra yang akan dikonvolusi. Proses akan dilakukan secara bertahap dengan

memotong bagian-bagian input vektor menjadi vektor seukuran dengan kernel, potongan vektor ini disebut dengan vektor *pooled*.



Gambar 2.6. Proses Konvulasi [24].

Lalu Tujuan dilakukannya konvolusi pada data citra adalah untuk mengekstraksi fitur dari citra input. Konvolusi akan menghasilkan transformasi linear dari data input sesuai informasi spasial pada data. Bobot pada layer tersebut menspesifikasikan kernel konvolusi yang digunakan, sehingga kernel konvolusi dapat dilatih berdasarkan input pada CNN.

3. *Pooling* Layer

   Pooling layer bertujuan untuk secara bertahap mengurangi dimensi dari gambar, dan selanjutnya mengurangi jumlah parameter dan kompleksitas komputasi model.

4. *Fully-Connected* Layer

   Fully-Connected layer adalah layer yang akan melakukan proses komputasi untuk menghasilkan skor probabilitas dari tiap-tiap kelas yang akan diprediksi

2.2.4. Pengalaman Bermain

Gaming Experience atau Pengalaman Bermain dinilai berdasarkan dari emosi, pikiran, reaksi dan perilaku pemain karena dipengaruhi dari fungsionalitas, konten, layanan, *player affinity*, dan penilaian dari pemain [25]. Terdapat beberapa elemen yang dapat digunakan sebagai tolak ukur dalam mengukur pengalaman bermain, tolak ukur tersebut adalah :

1. *User Interface* (UI)

   Dalam UI terdapat 2 tolak ukur, yaitu *usability* dan konsisten. *Usability* apabila semua fitur yang ada pada *game* berfungsi dengan baik, dan konsisten apabila suatu *event* yang digunakan juga digunakan di tempat lain dengan model yang sama, sehingga mengurangi ingatan jangka pendek.

2. *User Experience* (UX)

   Dalam UX terdapat 3 tolak ukur, yaitu *useful, usable*, dan *desirable*. *Useful* apabila dapat memenuhi suatu kebutuhan dasar pemain atau dapat dikatakan *game* memiliki manfaat bagi pemain. *Usable* apabila game dapat dimainkan secara efisien dan mudah dipelajar. Lalu *desirable* apabila dapat berkontribusi terhadap kepuasaan pengguna.

3. *Gameplay Experience*

Gameplay Experience berarti kombinasi dari sesuatu yang diharapkan oleh seseorang terhadap suatu objek dimana hal tersebut menjadi sesuatu yang menyenangkan bagi pemain. Kemuian juga ditambahkan peninjauan ulang dimana konten yang terdapat di dalamnya layak untuk dipelajari.

4. *Game Balancing*

   Game Balancing berarti game dapat berlaku adil pada setiap tindakan yang dilakukan memberikan dampak yang sesuai di dalam *game*. Setiap keberhasilan dan kegagalan dapat dipahami secara rasional. Di dalam *game balancing* juga terdapat *different skill levels*¸ dimana ini dibutuhkan untuk menentukan kepuasan pemain, dimana terdapat tantangan yang memiliki tingkat kesulitan yang berbeda di setiap level.

2.2.4.1. *Win Rate*

Dalam istilah yang paling sederhana, tingkat kemenangan (*Win Rate*) adalah seberapa sering seorang pemain dapat berharap untuk menang dalam permainan. Tingkat kemenangan ini mungkin muncul sebagai persentase. Kita dapat menghitung Win Rate dengan rumus total kemenangan : total bermain x 100% [26].

Saat kedua player bertemu dalam suatu permainan, baik itu secara *online multiplayer* ataupun tradisional *multiplayer*, hal yang pertama kali diperhatikan oleh pemain bukanlah skill melainkan *win rate*, setelah itu baru pemain melihat skill. Tidak jarang juga yang sudah melihat skill tapi setelah melihat *win rate* mereka beranggapan kalau permainan itu hanya keberuntungan belaka saja [26].