BAB II

LANDASAN TEORI

Berikut merupakan beberapa teori dan pustaka yang digunakan dalam penelitian yang dilakukan

2.1 Tinjauan Pustaka

(Iizuka, Simo-Serra, and Ishikawa 2016) melakukan penelitian dengan judul "*Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification*". Penelitian ini bertujuan untuk mengetahui tingkat akurasi pada proses pewarnaan citra dengan menggunakan metode *CNN* yang mengesktraksi *global* dan *local feature,* kontribusi penelitian ini adalah penggunaan *CNN* sebagai dasar akrsitektur pada proses pewarnaan citra dan minimal data *training* yang dibutuhkan. pembeda dari penelitian ini adalah penambahan arsitektur *CNN* untuk proses pengecekan dan penggunaan *pre-processing CLAHE* untuk meningkatkan akurasi pewarnaan.

(Zhang et al. 2017) melakukan penelitian dengan judul "*Real-Time User-Guided Image Colorization with Learned Deep Priors*". Penelitian ini bertujuan untuk mengetahui tingkat akurasi pewarnaan dengan menggunakan metode *U-net* dengan *global hints* dan *local hints* sebgai fitur kunci pada proses pewarnaan. Kontribusi untuk penelitian ini adalah penggunaan arsitektur *U-net* pada proses pewarnaan. Perbedaan dengan penelitian ini yaitu penambahan *Critic* sebagai validator pewarna dan *pre-processing CLAHE*

(Richard Zhang, Phillip Isola 2016) melakukan penelitian dengan judul "*Colorful Image Colorization"*. Penelitian ini bertujuan untuk mengetahui tingkat akurasi pada proses pewarnaan citra dengan menggunakan metode *CNN* , kontribusi penelitian ini adalah penggunaan *CNN* sebagai dasar akrsitektur dan cara pengujian pewarnaan citra. pembeda dari penelitian ini adalah penambahan arsitektur *CNN* yaitu *critic* untuk proses pengecekan dan penggunaan *pre-processing CLAHE* untuk meningkatkan akurasi pewarnaan.

(Qin et al. 2017) melakukan penelitian dengan judul "*Research on Image Colorization Algorith Based on Resiidual Neural network"*. Tujuan penelitian ini adalah untuk mengetahui tingkat akurasi pewarnaan citra menggunakan residual *network* yang terdiri dari *CNN*. Kontribusi penelitian ini adalah analisis dan penggunaan residual *network* pada pewarnaan citra, yang telah dimodifikasi untuk digunakan pada arsitektur *Unet*. Perbedaan dengan penelitian yang dilakukan adalah penggunaan residual *network* ada di dalam arsitektur *Unet*, penambahan *Critic* dan penggunaan *CLAHE* sebagai *pre-processing*

(Cheng 2016) melakukan penelitian dengan judul "*Deep Colorization"*. Memiliki tujuan untuk mengimplementasikan *deep learning* pada proses pewarnaan. Kontribusi untuk penelitian ini adalah analisis akhirnya yaitu proses pewarnaan citra menggunakan *deep learning* membutuhkan citra latih yang sangat banyak, perbedaan dengan penelitian yang dilakukan adalah penggunaan *No-GAN* dan *CLAHE* sebagai metode untuk pewarnaan citranya.

(Leibe et al. 2016) melakukan penelitian dengan judul "*Learning Representations for Automatic Colorization"*. Memiliki tujuan untuk mengimplementasikan *VGG-16* dan, kontribusi penelitian ini adalah teknik menghitung akurasi pewarnaan citra dan analisis tentang kebutuhan data latih yang banyak. Pembeda dengan penelitian ini ialah implementasi *No-GAN* dan *CLAHE* sebagai metode pewarnaan citra.

(Guadarrama et al. 2019) melakukan penelitian dengan judul "*PixColor:Pixel Recursive Colorization"*. Memiliki tujuan untuk melakukan pewarnaan citra dengan metode *PixelCNN* yang memproses *low colorization* dan *feedfoward CNN* untuk *high resolution refinement.* Kontribusi penelitian ini adalah analisis tentang penggunaan *pixelCNN* yang arsitekturnya hampir sama dengan *GAN*. Perbedaan dengan penelitian yang dilakukan adalah penggunaan *No-GAN* sebgai arsitektur utama pada pewarnaan citra.

(Gupta et al. 2017) melakukan penelitian dengan judul "*A learning-based approach for automatic image and video colorization"*. Tujuannya mengimplementasi metode *superpixel extraction*, *color quantization* dan *random forest learning* untuk proses pewarnaan citra. kontribusi penelitian ini adalah

analisis penelitiannya, yaitu dengan menggunakan *spatial feature extraction* terdapat masalah *bleeding* ketika objek nya terlalu kecil pada citra. Perbedaan dengan penelitian yang dilakukan adalah penggunaan metode yang berbeda, dimana di penelitian yang telah dilakukan digunakan metode *No-GAN* dan *CLAHE*

(Varga and Szirányi 2017) melakukan penelitian dengan judul "*Convolutional Neural network for automatic image colorization"*(Varga and Szirányi 2017). Tujuan dari penelitian ini adalah penggunaan *VGG-16* pada proses pewarnaan, kontribusi penelitian ini adalah analisis bahwa penggunaan *VGG-16* masih kurang maksimal pada proses pewarnaan citra karena bisa memberikan warna *blur* pada citra hasil pewarnaan. Perbedaan dengan penelitian yang telah dilakukan adalah penggunaan arsitektur *U-net* untuk pewarnaan, bukan *VGG-16*.

(Cao et al. 2017) melakukan penelitian dengan judul "*Unsupervised Diverse Colorization via Generative Adversarial Network"*. Bertujuan untuk mengimplementasi *Generative Adversarial Network* pada pewarnaan citra. kontribusi untuk penelitian ini adalah analisis tentang optimalisasi pada proses *learning*, dimana *GAN* sendiri memiliki kekurangan pada *loss function.* Perbedaan dengan penelitian yang telah dilakukan adalah penggunaan metode *No-GAN* sebagai proses pelatihannya dan penambahan *CLAHE* sebagai *pre-processing*.

(Koo 2016) melakukan penelitian dengan judul "*Automatic Colorization with Deep Convolutional Generative Adversarial network"*. Bertujuan untuk mengimplementasikan metode *Generative adversarial network* sebagai arstektur utamanya dan *YUV color space* pada proses pewarnaan. Kontribusi penelitiannya adalah analisis tentang terjadinya *Vanishing gradient* dimana *generator* gagal mengimbangi *critic* sehingga gagal mempelajari pola citra. perbedaan dengan penelitian ini adalah melakukan pelatihan sesuai dengan metode *No-GAN* dimana melatih parsial antara *generator* dan *critic*, selain itu penambahan *CLAHE* sebagai *pre-processing*

(Deshpande, Rock, and Forsyth 2015) melakukan penelitian dengan judul "*Learning Large-Scale Automatic Image Colorization"*. Bertujuan untuk mengimplementasikan metode *Learch* pada proses pewarnaan citra. kontribusi penelitiannya adalah analisis pada kebutuhan *dataset* yang besar, dimana

penggunaan data latih yang kurang mencakup semua objek dan kelas pada citra uji menghasilkan citra yang blur. Perbedaan pada penelitian ini adalah penggunaan metode *No-GAN* dan *CLAHE* pada proses pewarnaannya

Terdapat berbagai metode untuk melakukan pewarnaan citra sebagai contoh penggunaan metode learch **(Deshpande et al. 2015)**, namun metode ini terpaku akan data latih, sehingga jika objek data uji tidak ada pada data latih, maka sistem gagal memberikan warna pada citra. ada pula metode *random forest* yang dikombinasikan dengan *superpixel* **(Gupta et al. 2017)**, metode ini memiliki kekurangan jika objek yang ada pada citra memiliki ukuran yang kecil. Disisi lain penggunaan *CNN* pada proses pewarnaan citra memiliki akurasi yang tinggi berdasarkan beberapa penelitian yang telah dilakukan, seperti pada penelitian **(Iizuka et al. 2016)** yang berhasil melakukan pewarnaan citra menggunakan *CNN* dan ada pula penelitian **(Richard Zhang, Phillip Isola 2016)**. Terdapat pula *mod*ifikasi *CNN* yang lebih kompleks dan mendapatkan akurasi yang lebih baik, seperti pada penelitian **(Qin et al. 2017)** yang menggunakan metode *residual network*, atau penelitian **(Leibe et al. 2016)** & **(Varga and Szirányi 2017)** yang menggunakan metode *VGG*-16, arsitektur yang komplek ini mendorong pewarnaan citra pada tingkatan yang lebih baik, namun metode yang rumit ini masih memiliki kekurangan yaitu menghasilkan citra yang blur. Terdapat pula penelitian yang lebih kompleks dengan menggabungkan beberapa arsitektur seperti pada penelitian yang dilakukan oleh **(Guadarrama et al. 2019)** yang menggunakan metode *Pixcolor* yang mengkombinasikan *CNN* dengan *residual network*, ada pula penelitian lain yang menggunakan metode *U-NET* **(Zhang et al. 2017)**, *Unet* mendapatkan akurasi yang semakin baik, namun masih memiliki kekurangan dimana hasil citra yang diwarnai masih ada yang *blur* atau masih *grayscale*. maka muncul penelitian lain yang mengembangkan penelitian dasar *Unet* ini yaitu penelitian **(Cao et al. 2017)** yang menggunakan *Generative adversarial network*, karena menggunakan *GAN* yang masih *default*, terdapat kekurangan pada *loss function* dan hasilnya kurang maksimal. Terdapat pula metode kombinasi antara *GAN* dan *YUV color space* yang dilakukan oleh **(Koo 2016),** memiliki kekurangan pada proses pelatihannya, dimana *generator* gagal mengikuti *critic***.**

2.2 Image colorization

Image colorization adalah proses menambahkan warna pada gambar hitam putih atau *grayscale* (Gupta et al. 2017). Proses ini merubah *mod*el warna dari *grayscale* atau hitam putih yang memiliki 1 tingkat kedalaman warna menjadi *mod*el warna yang memiliki 3 tingkat kedalaman warna seperti *LAB*, *CMYK* atau *RGB.*

Proses pewarnaan citra dilakukan menggunakan 2 metode yaitu secara manual dan otomatis (Richard Zhang, Phillip Isola 2016). Pewarnaan secara manual dilakukan oleh seseorang yang memiliki keahlian dibidang multimedia, karena pekerjaan ini berubungan dengan seni dan teknologi implementasi seni pada *computer*, seperti *photoshop*. Dengan pengalaman sebagai ahli multimedia, mereka bisa menentukan warna yang paling natural dan cocok sesuai dengan mata manusia kepada sebuah citra, selain menentukan warna mereka juga mewarnai citra tersebut sesuai dengan warna yang sudah di tentukan (Richard Zhang, Phillip Isola 2016).

Proses pewarnaan yang kedua adalah menggunakan bantuan *computer* atau secara otomatis, banyak metode yang digunakan pada proses pewarnaan citra secara otomatis ini mulai dari penggunaan *conversi* secara langsung dan *probabilistic distribution* (Royer, Kolesnikov, and Lampert 2019) mendapatkan hasil yang didapat tidak sama dengan citra asli dikarenakan tidak adanya pelatihan, namun pada percobaan itu *system* berhasil memunculkan warna citra. Selanjutnya ada yang menggunakan *sparse representation* (B. Li et al. 2017), dimana citra berhasil diwarnai berdasarkan sebuah pendoman warna yang di berikan oleh *user*. Ada juga yang menggunakan *residual neural network* (Qin et al. 2017) mendapatkan hasil yang menyerupai aslinya, namun memiliki kekurangan karena hasil pewarnaan sangat bergantung pada warna yang ada di *database*

Gambar 2.1 Contoh Image Colorization

(Sumber :(Cheng 2016))

2.3 Deep Learning

Deep learning adalah metode pembelajaran yang dilakukan oleh mesin dengan cara meniru bagaimana otak manusia bekerja. *Deep learning* terdiri dari berbagai tingkatan yang di representasikan sebagai *layer*, dimana tiap-tiap *layer* saling bergantung satu sama lain (Deng and Yu 2013). Pada tiap *layer* terdiri dari beberapa operasi dan transformasi yang berguna untuk mengambil data penting yang ada pada tiap data masukan, hasil data yang didapat dari sebuah *layer* akan dijadikan sebagai *input* untuk *layer* selanjutnya. Dengan semakin banyaknya jumlah *layer* yang ada, sistem bisa menangkap relasi antar data, yang nantinya akan digabungkan untuk merepresentasikan sebuah objek (Bengio and Courville, Ian Goodfellow 2017) . Sebagai contoh diberikan *input* berupa citra dalam bentuk *array* pada *layer input*, setelah mengalami beberapa proses pada *layer* tersebut, data akan berubah menjadi data yang merepresentasikan sebuah garis *horizontal* atau *vertical*. Data garis tersebut akan dimasukan ke *layer* selanjutnya yang nantinya akan menghasilkan sebuah representasi dari objek.(Ponti et al. 2017). *Deep learning* memberikan terobosan baru dibidang *speech recognition, visual object recognition* dan *object detection* dengan proses *backpropagation*-nya yang bisa memperbarui *parameter* tiap *layer* yang ada pada *mod*el.(Lecun, Bengio, and Hinton 2015)

Menurut (Yan 2016) terdapat 2 proses besar pada Deep learning yaitu, *feedforward* dan *backpropagation*. Sama seperti machine learning, *feed forward* berfungsi untuk mengambil fitur-fitur yang ada pada citra, perbedaan pada deep

learning adalah dalamnya *layer* yang membentuk arsitektur tersebut (Yan 2016). Selain *feed forward* terdapat pula proses kedua yaitu *backpropagation* yang digunakan untuk proses pelatihan atau pembelajaran sistem. *input backpropagation* adalah nilai *loss* atau jarak perbandingan antara data *input* dengan data asli, nilai *loss* ini digunakan untuk memperbaiki susunan dan kombinasi bobot pada arsitektur (Lecun et al. 2015). Terdapat 3 pendekatan pada *deep learning* yaitu *deep supervised learning, deep unsupervised learning* dan *deep reinforcement learning*

2.3.1 Deep Supervised Learning

Deep Supervised learning adalah proses pembelajaran *deep learning* dimana dimana *dataset* yang digunakan untuk proses pembelajaran sudah diberi label, selanjutnya dilakukan prediksi data uji terhadap label yang ada pada data latih yang tersedia (Bengio and Courville, Ian Goodfellow 2017). *Deep Supervised learning* biasa digunakan pada proses klasifikasi atau regresi yang diimplementasikan pada arsitektur *Convolutional Neural Network* (*CNN*), *Recurrent Neural Network* (*RNN*), *Long Short Term Memory* (*LSTM*) (Minar and Naher 2018)

2.3.2 Deep Unsupervised Learning

Deep Unsupervised learning adalah proses pembelajaran *deep learning* tanpa dibantu oleh label, sehingga *mod*el akan melakukan ekstraksi fitur dari data *input* dan memberikan label pada hasil ekstraksi fitur tersebut secara otomatis . (Bengio and Courville, Ian Goodfellow 2017). *Deep Unsupervised learning* digunakan pada proses klastering atau *generating* data. Contoh metodenya adalah *Principal Component Analysis* (*PCA*) dan *Singular Value Decomposition* (*SVD*) (Minar and Naher 2018).

2.3.3 Deep Reinforcement Learning

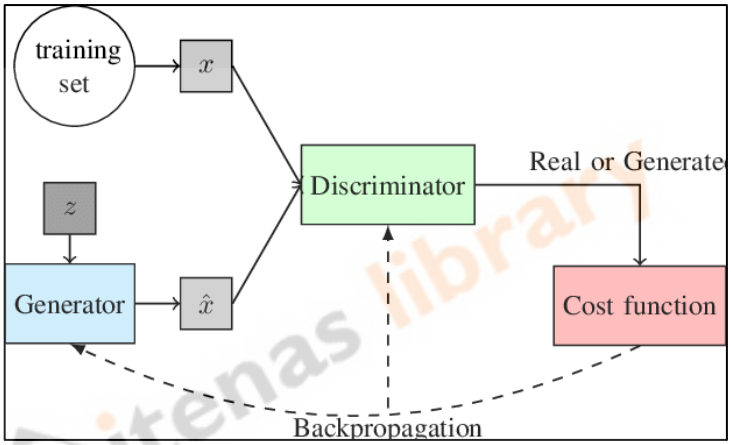
Deep Reinforcement Learning adalah kombinasi antara *deep learning* dan *reinforcement learning*, pendekatan ini melakukan pembelajaran melalui *trial* and

error. Dengan mencoba berbagai kombinasi yang ada, sistem akan mengambil cara terbaik untuk menyelesaikan suatu masalah (Mousavi, Schukat, and Howley 2018). Pendekatan ini biasa digunakan pada *environtment* yang belum diketahui, seperti pada mobil pintar (François-lavet et al. 2018)

2.4 Generative Adversarial Network

Generative Adversial Network terbagi menjadi 2 bagian *neural network* yaitu *generator* dan *descriptor* atau *critic*

2.4.1 Teori Generative Adversarial Network



Gambar 2.2 Generative Adversarial Network

(Sumber :(Ponti et al. 2017))

Generative Adversarial Network (GAN) adalah arsitektur jaringan saraf tiruan atau *neural network* yang bertujuan untuk membentuk atau membuat suatu data yang benar-benar baru berdasarkan data latih atau data yang sudah dilihat sebelumnya, biasa di implementasikan pada data citra yang berupa *pixel*. *GAN* memiliki kelebihan di bidang *image-to-image translation* dan *image generator* termasuk pewarnaan citra (C. Li et al. 2017). *GAN* terdiri dari 2 buah *neural network* yang saling berkompetisi yaitu *generator* yang terbuat dari *convolutional neural network*, memiliki tugas untuk menciptakan citra dan *descriptor* atau *critic* yang terdiri dari *binary network* yang bertugas untuk melakukan *checking* keaslian hasil citra sudah sesuai dengan data latih (C. Li et al. 2017). Dengan melatih

generator dan *critic* atau *descriptor*, *GAN* bisa menciptakan gambar baru yang sesuai dengan persyaratan atau data latih yang ada.

2.4.2 Normalization Image

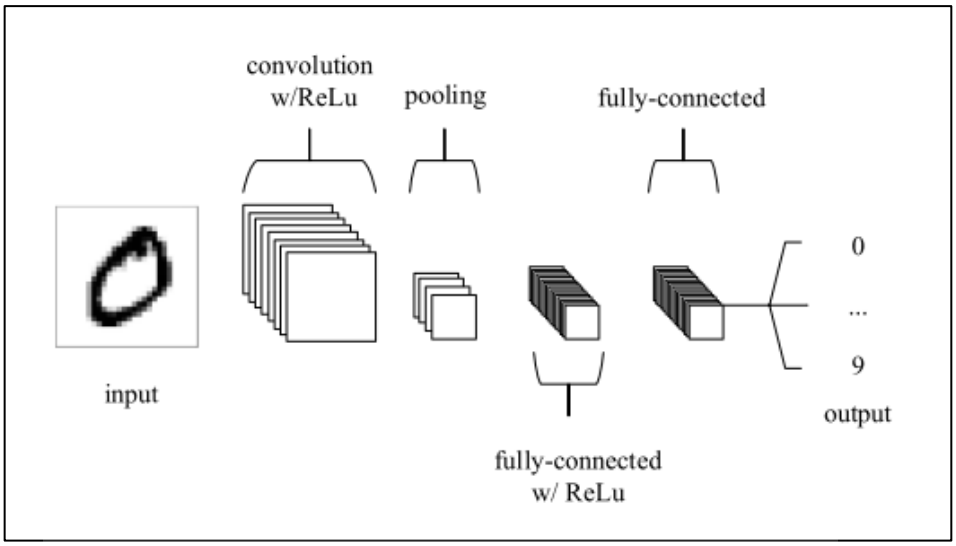
Normalisasi dilakukan sebagai salah satu *pre-processing* pada citra yang akan dijadikan *input* pada sebuah proses *neural network*, normalisasi digunakan untuk mempersingkat proses perhitungan dan melakukan standarisasi pada tiap nilai *pixel* (Manju et al. 2019). Standarisasi adalah proses transformasi atau perubahan nilai pixel yang ada pada citra sehingga nilai citra tersebut ada pada range tertentu, sebagai contoh normalisasi *image* di penelitian ini digunakan *range* 0-1 agar mempermudah perhitungan konvolusi sehingga bisa mempercepat proses sistem berkerja. Normalisasi dilakukan dengan membagi tiap *pixel* dengan nilai *pixel* maksimalnya sesuai dengan rumus (1).

$$image_{x,y} = \frac{image_{x,y}}{255} \quad \text{.....(1)}$$

2.4.3 *Convolution* Neural Network

Convolution neural network adalah jaringan syaraf tiruan yang terdiri dari 3 *layer* utama, yaitu *convolution layer* atau layer konvolusi yang digunakan untuk proses konvolusi, *pooling layer* atau layer *pooling* yang digunakan untuk memperkecil dimensi citra dan *fully-connected layer* atau layer koneksi penuh yang digunakan untuk menghitung hasil akhir (O'Shea and Nash 2015). *CNN* di utamakan untuk *image processing* karena memiliki kemampuan untuk mengambil fitur penting atau *pattern* pada citra dengan perhitungan yang lebih ringan dibanding dengan *Artificial Neural Network* (Varga and Szirányi 2017). Pada *convolution layer* akan dilakukan proses konvolusi antara citra dengan *kernel* atau *weight* pada *layer* tersebut, berguna untuk pengambilan fitur pada citra. Setelah itu dilakukan proses *pooling* pada *pooling layer* yang berfungsi untuk memadatkan atau mengeliminasi fitur citra yang telah di dapat, berguna untuk memperkecil dimensi dan mempersingkat proses perhitungan selanjutnya. Proses terakhir adalah *flatten* oleh *fully-connected* layer, pada *layer* ini hasil fitur citra yang telah

dipadatkan akan dijadikan *array* 1 dimensi untuk nantinya digunakan sebagai proses klasifikasi (O'Shea and Nash 2015).



Gambar 2.3 Arsitektur Dasar *CNN*

(O'Shea and Nash 2015)

2.4.4 *Convolution*

Convolusi adalah sebuah proses matematis yang menggabungkan dua buah matrik menjadi matriks baru dengan cara memasukan fungsi matematis ke matriks *input* dan *filter* atau *kernel* sesuai dengan rumus (2) (O'Shea and Nash 2015). Proses ini bertujuan untuk mengambil fitur-fitur penting pada matriks *input* atau citra *input*, itur-fitur penting pada citra terbagi menjadi beberapa kedalaman, pada kedalaman awal fitur yang dianggap penting adalah garis *horizontal* dan *vertical* yang ada pada citra, untuk kedalaman selanjutnya fitur yang dianggap penting lebih kompleks dengan menggabungkan fitur-fitur sebelumnya yang sudah di dapat, seperti fitur hidung, mata dan mulut. Hingga akhirnya kedalaman terakhir bisa mengambil kombinasi *pixel* yang merepresentasikan muka dengan menggabungkan fitur sebelumnya, yaitu mulut, mata dan hidung. Proses konvolusi terjadi dari kiri atas bergerak ke kanan lalu kebawah jika sudah mencapai ujung kanan, Proses pergeseran ini dinamakan *stride*. Ilustrasi konvolusi ada pada Gambar 2.4
konvolusi terjadi dengan rumus :

$$h(x) = F(x) * g(x) = \sum_{-\infty}^{\infty} F(a)g(x-a) \quad \text{..... (2)}$$

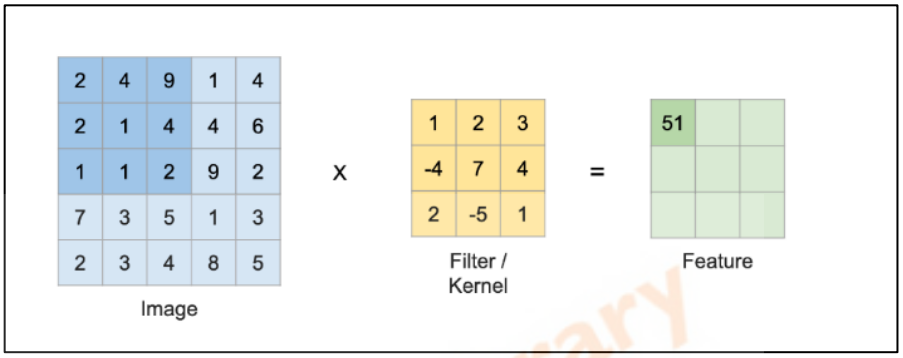
Dimana

F(x) = citra asli

g(x) = *kernel* konvolusi

x = nilai *pixel*

a = nilai *pixel* pada koordinat citra

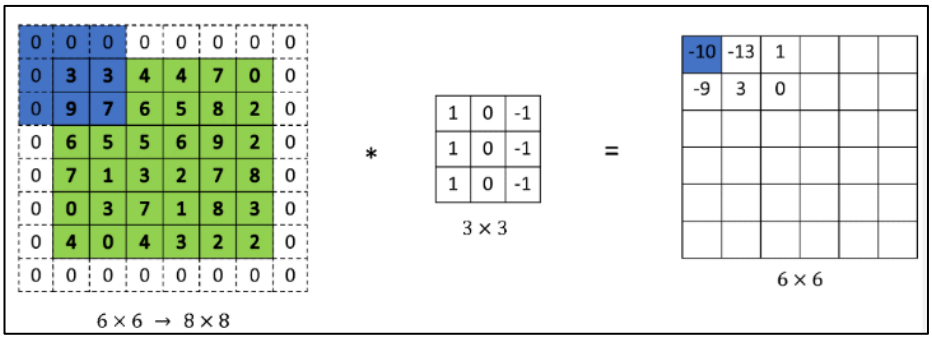


Gambar 2.4 ilustrasi konvolusi

(Sumber : https://miro.medium.com/)

2.4.5 Zero *Padding*

padding dilakukan untuk mencegah *feature loss* (Varga and Szirányi 2017). ketika dilakukan konvolusi pada citra 5x5 dengan *kernel* 3x3 maka didapat hasil citra dengan ukuran 2x2, ukuran yang kecil ini menghilangkan banyak informasi untuk proses selanjutnya maka dari itu ditambahkan *pixel* 0 di setiap koordinat terluar sehingga ukuran *pixel* menjadi 3x3. Dengan menambahkan 0 pada setiap sisi ini semua citra input bisa dilakukan konvolusi tanpa merubah hasil konvolusinya, maka dari itu proses tidak ada *feature* yang terbuang atau yang tidak digunakan pada proses konvolusi nanti Ilustrasi *padding* ada pada Gambar 2.5

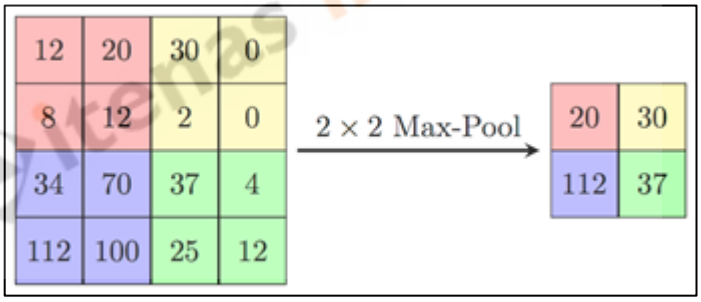


Gambar 2.5 ilustrasi *padding*

(Sumber : http://datahacker.rs/what-is-*padding*-cnn/)

2.4.6 Pooling

Pooling layer bertujuan untuk mengurangi dimensi citra, sehingga mengurangi kompleksitas komputasi *mod*el (O'Shea and Nash 2015). Salah satu varian *pooling* adalah *max pooling* dimana diambil nilai terbesar sesuai dengan ukuran *filter*, proses ini bertujuan untuk mengurangi dimensi tetapi tetap menjaga nilai maksimum dari setiap lapisan. Ilustrasi pooling ada pada Gambar 2.6



Gambar 2.6 ilustrasi pooling

(Sumber : embarc.org)

2.4.7 ReLU Aktivasi

Relu activation berguna sebagai aktivasi *node* selanjutnya dengan mengeliminasi nilai *node* jika nilainya kurang dari 0, berfungsi untuk standarisasi *input* atau ada pada range 0-1 saja pada *node* atau *layer* selanjutnya, sehingga *input* dari tiap *layer* tetap bersifat positif dan mengurangi waktu komputasi (O'Shea and Nash 2015). Memiliki rumus sesuai dengan rumus (3)

$$g(z) = \max\{0, z\} \quad \text{..... (3)}$$

Dimana

g(z) = nilai setelah aktivasi

0 = nilai batas ambang

z = nilai hasil konvolusi

2.4.8 Upsampling Layer

setelah mendapat nilai *feature* pada *CNN* dilakukan *up-sampling* untuk mengembalikan ukuran citra menjadi semula. Pengembalian ukuran citra ini caranya adalah dengan meningkatkan ukuran semula citra dan melakukan duplikasi *pixel* sesuai dengan lokasinya. Proses ini akan digunakan di proses selanjutnya atau proses penggabungan untuk membuat citra berwarna (Ronneberger, Fischer, and Brox 2015). Salah satu jenis pada proses *up-sampling* adalah *nearest neighbor*, dimana sebuah nilai matriks akan diperbanyak pada lokasi yang berdekatan dengan lokasi matriks tersebut. Ilustrasi *up-sampling* ada pada Gambar 2.7.

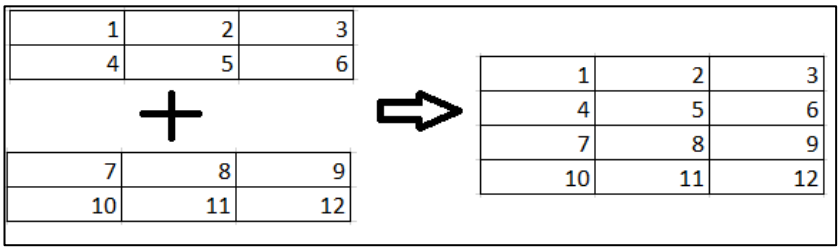
Gambar 2.7 ilustrasi Upsampling

(Sumber : Yi-Fan Liou)

2.4.9 Concatenate

Concatenate adalah proses untuk menggabungkan dua buah matriks menjadi satu buah matriks (Ronneberger et al. 2015). Proses ini dilakukan untuk menghasilkan dimensi baru atau kanal warna baru, selain itu juga digunakan agar proses penggabungan tidak kekurangan fitur citra lama, dengan menggabungkan

citra sebelum proses pooling dan citra setelah proses ekstraksi ciri maka fitur yang ada pada citra tidak akan hilang ketika di rekonstruksi ulang.



Gambar 2.8 ilustrasi Concatenate

2.5 No-Generative Adversarial Network

No-GAN adalah sebuah metode *mod*ifikasi dari *GAN* yang diciptakan oleh Jason Antic. metode ini sama seperti *GAN*, perbedaannya ada pada proses pembelajaran, proses pembelajaran dari *No-GAN* dilakukan perbagian, *generator* melakukan pelatihan sendiri dan *descriptor* melakukan pelatihan sendiri, selanjutnya dilakukan pelatihan gabungan menggunakan *GAN* namun tidak secara keseluruhan, hanya beberapa persen dari total data latih (Antic, 2020). Metode *No-GAN* mengambil *loss function* berupa *Wessertain loss*. Ilustrasi pelatihan ada pada Gambar 2.9

Wessertain loss mengambil nilai dari rentang -1 hingga 1, fungsi dari nilai ini adalah untuk patokan perbaikan *generator* dan *critic*. *Wessertain loss function* memiliki rumus(4)(5) :

$$Critic\ Loss : D(x) - D\big(G(z)\big) \dots (4)$$

$$Generator\ Loss : D\big(G(z)\big) \dots (5)$$

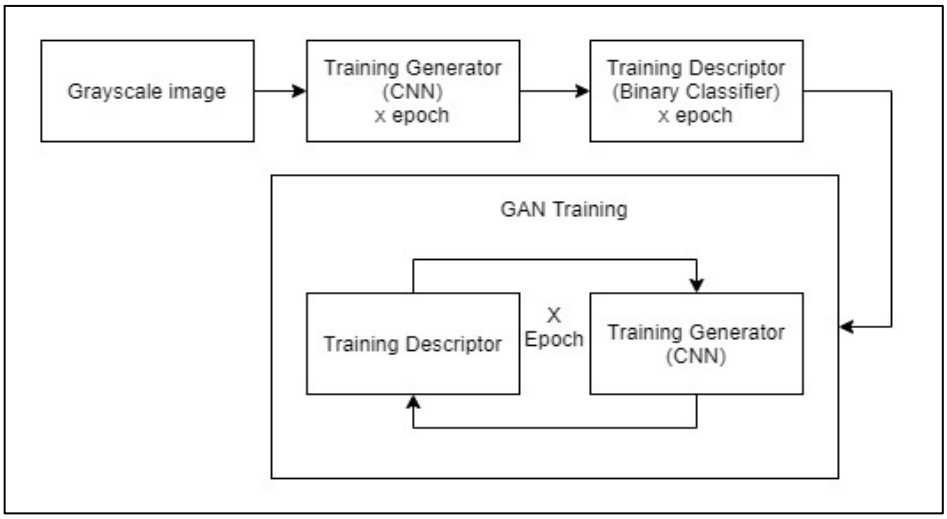
Dimana :

D(x) : *descriptor* atau *critic output* untuk gambar yang asli.

G(z) : *output* dari *generator*

D(G(z)) : *output* dari *descriptor* atau *critic* untuk gambar yang palsu

dengan metode ini pewarnaan citra bisa dilakukan dengan menghasilkan citra yang realistis.



Gambar 2.9 No-*GAN* Training

2.6 Grayscale

Gambar 2.10 Citra Grayscale

(Sumber : geeksforgeeks.org)

Grayscale (Gambar 2.10) adalah proses merubah citra dari berwarna menjadi abu-abu. Proses ini dilakukan untuk memudahkan proses komputasi pada pengolahan citra karna nilai abu-abu akan merepresentasikan 3 kanal warna *RGB* (Saravanan 2010). Proses *grayscale* memiliki rumus(6)

$$grayscale = 0.333R + 0.333G + 0.333B \quad \text{.....(6)}$$

dimana :

R = nilai *pixel Red* pada citra

G = nilai *pixel Green* pada citra

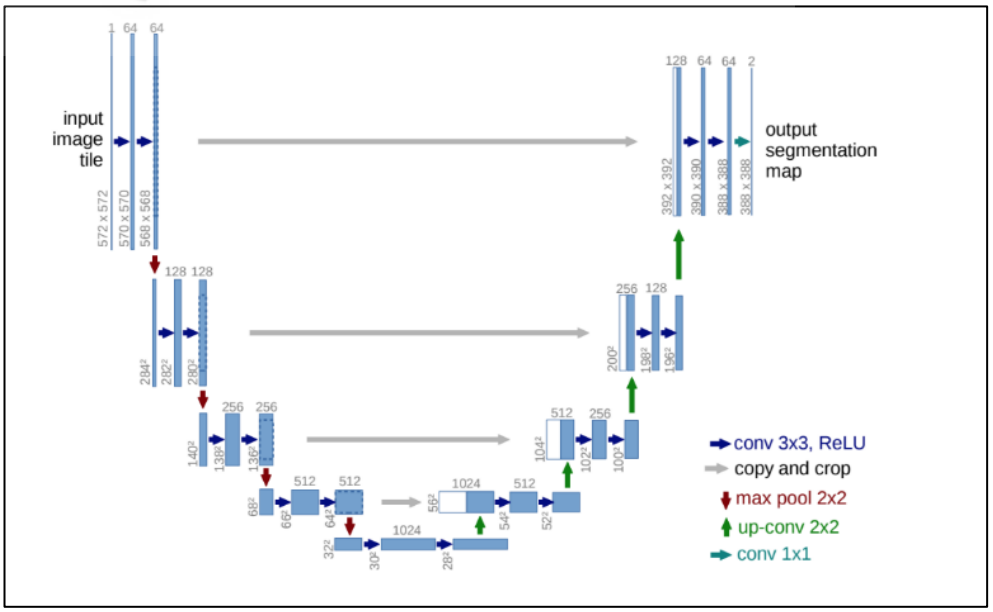
B = nilai *pixel Blue* pada citra

2.7 U-Net

U-net adalah salah satu arsitektur *CNN* yang digunakan untuk meningkatkan resolusi citra atau pewarnaan citra, karena memiliki kelebihan untuk membongkar sebuah citra dengan mengambil fitur-fitur penting pada citra dan melakukan pemasangan lagi namun dengan penambahan yang diinginkan pada citra tersebut (Ronneberger et al. 2015). *U-net* terdiri dari 2 bagian yaitu, *contracting path* dan *expanding path*.

Contracting path merupakan bagian kiri dari arsitektur yang terdiri dari konvolusi 3x3, aktivasi *Relu* dan *max pooling* dengan *stride* atau pergeseran *CNN* nya 2. Sedangkan pada bagian *expanding path* atau bagian kanan arsitektur terdiri dari *up convolution* 2x2, penggabungan 2 matrik, *convolusi* 3x3 dan aktivasi *Relu* (Ronneberger et al. 2015).

Bagian kiri berfungsi untuk melakukan segmentasi citra dengan mengambil fitur-fitur penting, sedangkan bagian kanan digunakan untuk menyatukan kembali citra dan membuat sebuah *layer* warna pada *output*. Ilustrasi *Unet* ada pada Gambar 2.11



Gambar 2.11 Unet Architecture

(Sumber : (Ronneberger et al. 2015))

2.8 *CLAHE*

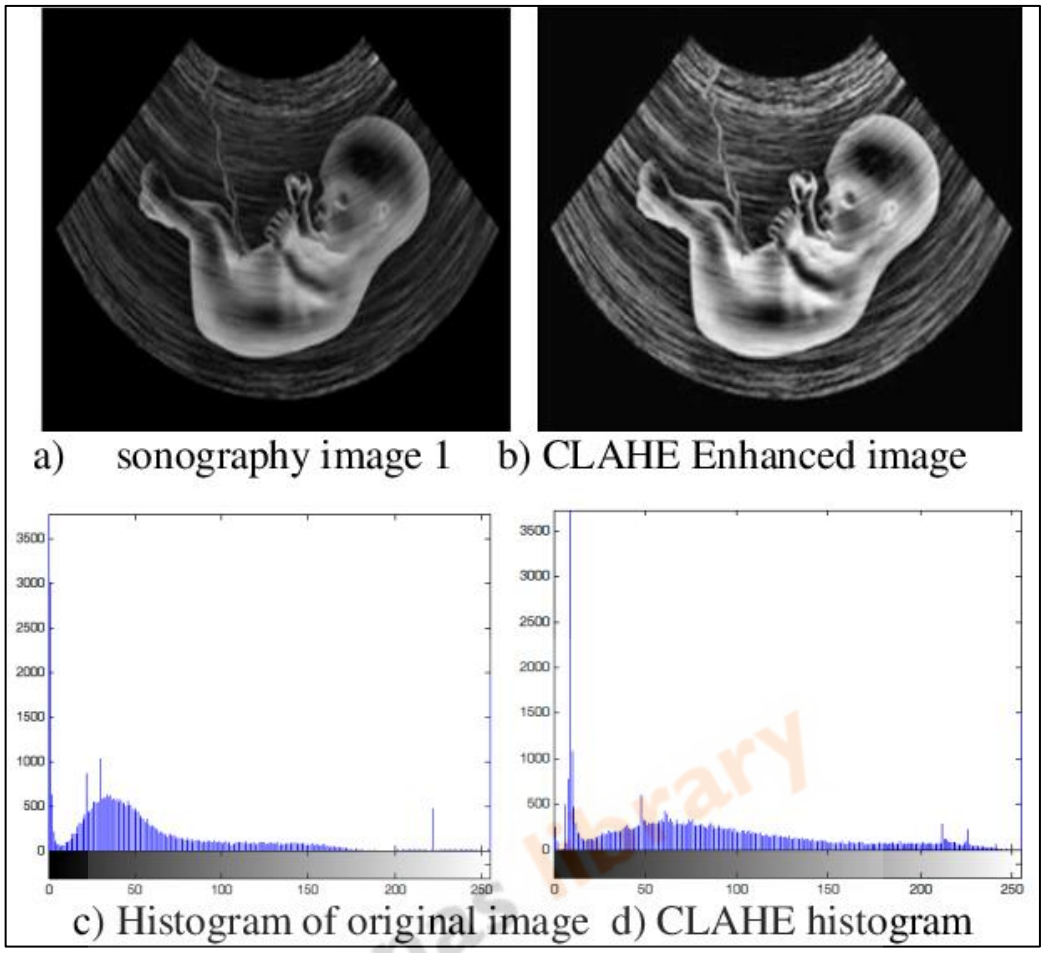
CLAHE (*Contrast Limited Adaptive Histogram Equalization*) adalah salah satu *image pre-processing* yang berfungsi untuk *image enhancement* yang meningkatkan nilai *contrast* bergantung dengan nilai *pixel* tetangganya sehingga tidak akan terjadi pencerahan *abnormal*(Manju et al. 2019).

CLAHE memiliki kelebihan dengan meningkatkan *contrast* suatu citra tanpa meningkatkan *noise*-nya, *image enhancement* melalui proses *histogram equalization* akan meningkatkan kecerahan citra namun metode ini juga akan meningkatkan kecerahan *noise* yang ada, sehingga penggunaan *CLAHE* lebih di rekomendasikan untuk *image enhancement*(Sonali et al. 2019)

Input proses *CLAHE* adalah citra *grayscale*, selanjutnya citra *grayscale* akan dibagi menjadi beberapa blok, pada tiap blok akan dicari nilai maksimal dari tiap *pixel* yang ada di blok tersebut dan selanjutnya semua nilai *pixel* yang ada di blok tersebut akan di normalisasi dengan landasan nilai maksimal yang sudah di dapat tersebut. Normalisasi terjadi dengan mencari distribusi komulatif sesuai dengan rumus pada (7), selanjutnya hasil distribusi komulatif akan di kalikan dengan *clip limit* (8) untuk mendistribusi ulang tiap *pixel*, proses ini terjadi pada tiap blok. Hasil dari proses *CLAHE* ada pada Gambar 2.12 & Gambar 2.13

Gambar 2.12 Contoh *CLAHE image* enhancement

(Sumber :(Manju et al. 2019))



Gambar 2.13 Contoh *CLAHE image* enhancement

(Sumber :(Bhan and Patel 2017))

$$f_{i,j}(n) = \frac{(N-1)}{M} . \sum_{k=0}^{n} h_{i,j}(K) \quad \text{.....(7)}$$

Dimana :

f = distribusi komulatif

N = nilai maksimum *pixel*

M = ukuran citra

K = frekuensi kemunculan nilai *pixel*

$$\beta = \frac{M}{N}\left(1 + \frac{\alpha}{100}(S_{max} - 1)\right) \quad \text{.....(8)}$$

Dimana :

M = ukuran region citra

N = Nilai maksimum *pixel* (256)

$\alpha =$ nilai *clip factor* (0-100)

$S_{max} =$ nilai maksimal *pixel* yang ada di dalam region

2.9 Pengujian Kinerja Sistem

Proses pengujian pada sistem pewarnaan citra menggunakan perhitungan utama berupa *SSIM* (*Structural Similiarity Index*), *SSIM* merupakan salah satu *mod*el perhitungan untuk menghitung secara kualitatif penurunan kualitas citra yang dihasilkan karena kompresi data atau transformasi data (Varga and Szirányi 2017). *Input* dari *SSIM* adalah 2 buah citra yang ukurannya sama, karena *SSIM* sendiri mengambil penilaian dari struktur yang terlihat pada kedua buah citra (Varga and Szirányi 2017). Selain menggunakan *SSIM* sebagai perhitungan *mod*el, digunakan pula *PSNR* (*Peak Signal-to-Noise Ratio*) dan *NRMSE* (*Normalized Root-Mean-Square Error*)

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad \ldots\ldots (9)$$

Dimana

μ_x = rata-rata x

μ_y = rata-rata y

σ_x = *varians* x

σ_y = *varians* y

σ_{xy} = kovarian xy

$c_1 = (k_1 L)^2, c_2 = (k_2 L)^2$ = dua *variable* untuk stabilisasi

L = nilai maksimum *pixel*

$k_1 = 0.01$ *dan* $k_2 = 0.03$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - X_i)^2 \quad \text{.....(10)}$$

$$PSNR = 20.log_{10}(MAX) - 10.log_{10}(MSE) \quad \text{.....(11)}$$

$$RMSE = \frac{\sqrt{MSE}}{Ymax - Ymin} \quad \text{.....(12)}$$

Dimana

N = banyaknya data

Y=nilai *pixel* pada citra 1

X=nilai *pixel* pada citra 2

MAX = nilai maksimal

Ymax = nilai maksimal pada suatu citra

Ymin = nilai minimal pada suatu citra

2.10 Studi Kasus

Pada subbab ini dijelaskan mengenai studi kasus dari *proses Normalize, CLAHE, padding, Convolution, ReLU Activasion dan Max pooling* yang dilakukan terhadap citra 5x5 yang ditampilkan pada Gambar 2.14.

Gambar 2.14 Studi Kasus

2.10.1 *CLAHE*

Pre-processing pertama yang dilakukan adalah proses *histogram equalization* untuk meratakan kecerahan dan meningkatkan kualitas gambar. Proses pertama adalah mengambil nilai *pixel* yang ada pada Gambar 2.14, Nilai *pixel* Gambar 2.14 ada pada gambar Gambar 2.15 , proses pengambilan nilai *pixel* dilakukan menggunakan *python* dengan cara melakukan *print* pada Gambar 2.14

181	179	180	176	175	169
196	155	150	172	155	120
52	196	185	171	149	132
108	150	100	71	103	29
81	90	92	109	54	54
104	70	56	123	58	20

Gambar 2.15 Nilai *pixel* citra studi kasus

Proses selanjutnya dilakukan pembagian *region*, pada contoh dilakukan pembagian dengan ukuran 3x3 sehingga citra akan seperti pada Gambar 2.16

181	179	180	176	175	169
196	155	150	172	155	120
52	196	185	171	149	132
108	150	100	71	103	29
81	90	92	109	54	54
104	70	56	123	58	20

Gambar 2.16 pembagian region

setelah proses pembagian selanjutnya dilakukan normalisasi histogram dengan cara mencari *CDF* pada tiap region. *CDF* adalah distribusi komulatif, dimana nilai ini merepresentasikan kemungkinan sebuah sampel acak yang diambil pada data akan kurang dari atau sama dengan sebuah nilai tertentu. Untuk mencari distribusi komulatif pada nilai *pixel* 150, hal pertama yang dilakukan adalah

mencari distribusi probabilitas, didapat dengan cara membagi frekuensi kemunculan dengan ukuran region, sehingga 1/9 = 0,11. Untuk distribusi komulatif di dapat dengan menambahkan distribusi probabilitas *pixel* tersebut dengan probabilitas nilai *pixel* sebelumnya, sehingga distribusi komulatif *pixel* 150 adalah 0,11 + 0,11 = 0,22. Proses ini dilakukan berulang pada tiap nilai *pixel* yang ada pada region1. Dengan melakukan perhitungan terhadap seluruh data yang ada pada region 1 maka didapatkan *CDF* seperti pada Tabel 2.1.

Setelah mendapat nilai distribusi komulatif, selanjutnya mencari nilai *clip limit*, terdapat sebuah variable independent yang ada pada rumus *clip limit* yaitu *clip factor*, *clip factor* sendiri adalah sebuah batas yang di*input* oleh *user* untuk mengontrol seberapa besar tingkat pencerahan terjadi (Joseph et al. 2017). Rentang *clip factor* adalah dari 0 – 100, dengan asumsi bahwa *clip factor* adalah 10 atau batas peningkatan kecerahan yang diinginkan *user* adalah 0,1. maka perhitungan *clip limit* pada region kiri atas adalah

9/256(1+10/100(196-1)) = 0.7

Perhitungan *clip limit* dilakukan dengan menggunakan rumus (8) dimana nilai 9 adalah besar regionnya, 256 adalah maksimal nilai *pixel* 8-bit, 196 adalah maksimal nilai *pixel* yang ada pada region. Didapat hasil 0.7

Tabel 2.1 Hasil *CDF*

Nilai *pixel*	Frekuensi kemunculan	Distribusi probabilitas	Distribusi komulatif
52	1	0.11	0.11
150	1	0.11	0.11 + 0.11 = 0.22
155	1	0.11	0.11 + 0.22 = 0.33
179	1	0.11	0.11 + 0.33 = 0.44
180	1	0.11	0.11 + 0.44 = 0.55
181	1	0.11	0.11 + 0.55 = 0.66
185	1	0.11	0.11 + 0.66 = 0.77
196	2	0.22	0.22 + 0.77 = 0.99

Proses selanjutnya adalah melakukan normalisasi histogram, caranya dengan mengkalikan distribusi komulatif tiap *pixel* dengan nilai maksimal dari nilai *pixel* yang ada di region tersebut, hasil dari perhitungan ini ada pada Tabel 2.2.

Tabel 2.2 hasil normalisasi

Nilai *pixel*	Distribusi komulatif	Hasil
52	0.11	0.11 * 196 = 21.56
150	0.22	0.22* 196 =43.12
155	0.33	0.33* 196 =64.68
179	0.44	0.44* 196 =86.24
180	0.55	0.55* 196 =107.8
181	0.66	0.66* 196 = 129.36
185	0.77	0.77* 196 =150.92
196	0.99	0.99* 196 =194.04

Setelah dinormalisasi maka proses selanjutnya melakukan *clipping* dengan menambahkan tiap hasil *pixel* hasil perkalian antara nilai normalisasi dengan *clip-limit*, memiliki aturan jika nilainya diatas nilai maksimal *pixel* maka nilai tidak akan diganti. Perhitungan proses *clipping* ada pada Tabel 2.3.

Tabel 2.3 Hasil perhitungan Clipping

Nilai *pixel*	Nilai normalisasi	Hasil	pembulatan
52	21.56	21.56+(21.56 * 0.7) = 36.625	37
150	43.12	43.12+(43.12 * 0.7) =73.304	73
155	64.68	64.68+(64.68 * 0.7) =109.956	110
179	86.24	86.24+(86.24 * 0.7) =146.608	147
180	107.8	107.8+(107.8 * 0.7) =183.26	183
181	129.36	129.36+(129.36 * 0.7) =219.912	220
185	150.92	150.92+(150.92 * 0.7) =256.564	255
196	194.04	194.04+(194.04 * 0.7) =329.84	255

Dengan hasil perhitungan normalisasi, nilai *pixel* di region kiri atas akan berubah menjadi

Tabel 2.4 Hasil *CLAHE* Region kiri atas

220	147	183
255	110	73
37	255	255

Proses tersebut dilakukan berulang pada tiap region. Sehingga mendapatkan hasil seperti pada gambar 2.17

Tabel 2.5 Hasil Akhir *CLAHE*

220	147	183	180	192	191
255	110	73	130	96	64
37	255	255	128	112	128
128	220	96	64	128	64
96	128	112	160	64	64
255	64	32	255	160	128

Gambar 2.17 hasil citra CLAHE

2.10.2 Normalisasi

Pada tahap *pre-processing* selanjutnya dilakukan normalisasi untuk standarisasi nilai dengan cara merubah semua *pixel* menjadi nilai antara 0-1. Proses dilakukan dengan membagi semua nilai *pixel* dengan nilai maksimal *pixel* yaitu 255. Berikut (Gambar 2.18 & Gambar 2.19) adalah contoh perhitungan dan hasil dari prosses normalisasi

220	147	183	180	192	191	/255
255	110	73	130	96	64	
37	255	255	128	112	128	
128	220	96	64	128	64	
96	128	112	160	64	64	
255	64	32	255	160	128	

Gambar 2.18 ilustrasi *pre-processing* dengan pembagian 255

0.862745	0.576471	0.717647	0.705882	0.752941	0.74902
1	0.431373	0.286275	0.509804	0.376471	0.25098
0.145098	1	1	0.501961	0.439216	0.501961
0.501961	0.862745	0.376471	0.25098	0.501961	0.25098
0.376471	0.501961	0.439216	0.627451	0.25098	0.25098
1	0.25098	0.12549	1	0.627451	0.501961

Gambar 2.19 hasil *pre-processing* dengan pembagian 255

2.10.3 *Padding*

Proses *padding* yang dilakukan pada matriks hasil normalisasi dengan menambahkan nilai 0 pada setiap sisi matriks. Nilai selain 0 adalah nilai pixel yang akan di proses dan nilai 0 adalah nilai pixel tambahan yang digunakan agar proses konvolusi tidak merubah dimensi pixel. Ilustrasi hasil *padding* ada pada Gambar 2.20

0	0	0	0	0	0	0	0
0	0.862745	0.576471	0.717647	0.705882	0.752941	0.74902	0
0	1	0.431373	0.286275	0.509804	0.376471	0.25098	0
0	0.145098	1	1	0.501961	0.439216	0.501961	0
0	0.501961	0.862745	0.376471	0.25098	0.501961	0.25098	0
0	0.376471	0.501961	0.439216	0.627451	0.25098	0.25098	0
0	1	0.25098	0.12549	1	0.627451	0.501961	0
0	0	0	0	0	0	0	0

Gambar 2.20 *padding*

2.10.4 konvolusi

Selanjutnya dilakukan proses konvolusi dengan menggunakan *kernel* 3x3 pada citra hasil *padding*. Digunakan kernel dengan dimensi 3x3 karena untuk pengambilan local feature secara optimal adalah penggunaan kernel berdimensi ganjil, sedangkan 1x1 tidak bisa mengambil feature pada citra, maka dari itu digunakan kernel 3x3. Range isi dari kernel berubah-ubah sesuai dengan proses pelatihan namun untuk inisialisasi digunakan nilai random antara -1 hingga 1. Gambar 2.21 merupakan *filter* 3x3

0	0	0
0	1	0
0	0	-1

Gambar 2.21 filter 3x3

Hasil dari proses *convolution* matriks 8x8 terhadap *filter* 3x3 1 *stride* akan menghasilkan *output* matriks berukuran 7x7 seperti yang diilustrasikan pada Gambar 2.22.

0.431373	0.290196	0.207843	0.329412	0.501961	0.74902
0	-0.56863	-0.21569	0.070588	-0.12549	0.25098
-0.71765	0.623529	0.74902	0	0.188235	0.501961
0	0.423529	-0.25098	0	0.25098	0.25098
0.12549	0.376471	-0.56078	0	-0.25098	0.25098
1	0.25098	0.12549	1	0.627451	0.501961

Gambar 2.22 Hasil konvolusi

Proses konvolusi dilakukan dengan melakukan perkalian antara *kernel* dengan citra *input*, berikut adalah contoh salah satu perhitungan yang dilakukan pada pojok kiri atas

1. Untuk mendapatkan nilai pada baris 1 kolom 1 pada Gambar 2.22 maka dilakukan proses perhitungan =

((0*0) + (0*0) + (0*0) +

(0*0) + (1*0.862745) + (0.576471*0) +

(0*0)+(0*1) + (0.431373*-1)) = 0.431373

			0	0	0	0	0	0	0	0
			0	0.862745	0.576471	0.717647	0.705882	0.752941	0.74902	0
			0	1	0.431373	0.286275	0.509804	0.376471	0.25098	0
0	0	0	0	0.145098	1	1	0.501961	0.439216	0.501961	0
0	1	0	0	0.501961	0.862745	0.376471	0.25098	0.501961	0.25098	0
0	0	-1	0	0.376471	0.501961	0.439216	0.627451	0.25098	0.25098	0
			0	1	0.25098	0.12549	1	0.627451	0.501961	0
			0	0	0	0	0	0	0	0

Gambar 2.23 Ilustrasi Proses Konvolusi

2. Untuk mendapatkan nilai pada baris 1 kolom 2 pada Gambar 2.22. Jumlahkan, ((0*0) + (0*0) + (0*0) +

(0.862745*0) + (0.576471 * 1) + (0.717647*0) +

(0*1) + (0*0.431373) + (-1*0.286275) = 0.290196

Proses perhitungan ini diilustrasikan pada Gambar 2.24. Kotak merah pada Gambar 2.24 dikonvolusikan dengan matriks 3x3 sebelahnya. Terdapat pergeseran 1 posisi ke kanan karena nilai *stride* = 1

			0	0	0	0	0	0	0	0
			0	0.862745	0.576471	0.717647	0.705882	0.752941	0.74902	0
			0	1	0.431373	0.286275	0.509804	0.376471	0.25098	0
0	0	0	0	0.145098	1	1	0.501961	0.439216	0.501961	0
0	1	0	0	0.501961	0.862745	0.376471	0.25098	0.501961	0.25098	0
0	0	-1	0	0.376471	0.501961	0.439216	0.627451	0.25098	0.25098	0
			0	1	0.25098	0.12549	1	0.627451	0.501961	0
			0	0	0	0	0	0	0	0

Gambar 2.24 Ilustrasi Proses Konvolusi dilatasi Langkah 2

3. Untuk mendapatkan nilai pada baris 1 kolom 3, 4,5 dan 6 lakukan hal yang sama dengan langkah 2 dengan pergeseran 1 stride.

4. Untuk mendapatkan nilai pada baris 2 kolom 1 pada Gambar 2.25 Jumlahkan, ((0*0) + (0*862745) + (0*576471)
+ (0*0) * (1*1) + (0*0.431373) +
(0*0) + (0*0.145098) + (-1*1) = 0

			0	0	0	0	0	0	0	0
			0	0.862745	0.576471	0.717647	0.705882	0.752941	0.74902	0
			0	1	0.431373	0.286275	0.509804	0.376471	0.25098	0
0	0	0	0	0.145098	1	1	0.501961	0.439216	0.501961	0
0	1	0	0	0.501961	0.862745	0.376471	0.25098	0.501961	0.25098	0
0	0	-1	0	0.376471	0.501961	0.439216	0.627451	0.25098	0.25098	0
			0	1	0.25098	0.12549	1	0.627451	0.501961	0
			0	0	0	0	0	0	0	0

Gambar 2.25 Ilustrasi Proses Konvolusi dilatasi Langkah 3

5. Untuk mendapatkan nilai pada baris 2 kolom 2, 3, 4, 5 dan 6 lakukan pergeseran 1 *stride* pada posisi terakhir kotak merah pada Gambar 2.25, seperti yang dilakukan pada langkah 2.

6. Untuk mendapatkan nilai pada baris 3 ,4,5 dan 6 maka lakukan seperti pada langkah 4 dan 5.

2.10.5 ReLU Aktivasi

Proses Aktivasi *ReLU* dilakukan pada setiap elemen matriks Gambar 2.22 dengan merubah nilai negatif menjadi 0, dan tetap mempertahankan nilainya jika lebih dari sama dengan 0 sehingga menghasilkan matriks pada Gambar 2.26

0.431373	0.290196	0.207843	0.329412	0.501961	0.74902

0	0	0	0.070588	0	0.25098
0	0.623529	0.74902	0	0.188235	0.501961
0	0.423529	0	0	0.25098	0.25098
0.12549	0.376471	0	0	0	0.25098
1	0.25098	0.12549	1	0.627451	0.501961

Gambar 2.26 Matriks Hasil Operasi ReLU

2.10.6 Pooling

Operasi *pooling* yang dilakukan adalah *max pooling* dimana pada setiap *size* 2x2 pada citra hasil *Relu* diambil nilai terbesarnya. Berikut adalah hasil operasi *max pooling* ditampilkan pada Gambar 2.27.

0.431373	0.329412	0.74902
0.623529	0.74902	0.501961
1	1	0.627451

Gambar 2.27 Hasil Ilustrasi Max Pooling