BAB II
LANDASAN TEORI

Bab ini menjelaskan mengenai teori-teori dasar yang digunakan pada penelitian "IMPLEMENTASI *DEEP RESIDUAL NETWORK* (RESNET) DALAM IDENTIFIKASI PENYAKIT TUMOR OTAK PADA MANUSIA".

2.1. Deep Learning

Deep Learning (pembelajaran mendalam) merupakan suatu bidang yang ada pada *Machine Learning* yang diperkenalkan oleh Dechter pada tahun 1986. Setelah itu menjadi sangat populer dalam konteks *Deep Neural Network*, Pembelajar Jauh paling sukses, yang jauh lebih tua, sejak setengah abad (Schmidhuber, 2015). *Deep Learning* adalah penggunaan *layer* yang jumlahnya dapat mencapai ratusan sehingga dapat disebut "*deep*" (Mathworks, 2018). *Deep Learning* memungkinkan model komputasi yang terdiri dari beberapa lapisan pemrosesan untuk mempelajari representasi data dengan berbagai tingkatan abstrak. *Deep Learning* menemukan struktur rumit dalam kumpulan data besar dengan algoritma *backpropagation* untuk menunjukkan suatu mesin harus mengubah parameter yang digunakan untuk menghitung nilai di setiap lapisan dari nilai di lapisan sebelumnya atau proses nya berjalan mundur pada setiap lapisan nya (Lecun, Bengio, & Hinton, 2015).

Deep learning terbagi menjadi tiga kategori pendekatan yaitu *supervised learning*, *unsupervised learning*, dan *reinforcement learning*. Salah satunya potensi dari *deep learning* adalah mengganti fitur buatan tangan dengan algoritma yang efisien untuk pembelajaran *hierarkis unsupervised* (fitur tanpa pengawasan) atau *semi-supervised feature learning* (semi-diawasi) dan *hierarchical feature extraction* (ekstraksi fitur). Penerapan deep learning penerapan deep learning telah digunakan dalam beberapa bidang seperti klasifikasi gambar, klasifikasi video, *object detection, object recognition, text-to-speech, natural language processing, robotic, text classification,* dan *singing synthesis* (Schmidhuber, 2015).

2.1.1. Deep Supervised Learning

Supervised learning sebuah pendekatan menghasilkan fungsi yang memetakan vektor menjadi salah satu kelas dengan melihat data latih dalam input dan output nya (Nasteski, 2017). Data tersebut diberi label pada setiap kategorinya dan di setiap kategori memiliki parameter bobot yang dihasilkan yang sudah diberi label (Lecun et al., 2015). Adapun beberapa algoritma yang menggunakan *supervised learning* adalah *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), *Long Short Term Memory* (LSTM), DenseNet, dan lain-lain (Minar & Naher, 2018). Adapun kelebihan dari *supervised learning* yaitu:

- Menentukan kelas-kelas dalam data pelatihan secara tepat.
- Proses pembelajaran lebih sederhana.
- Mengetahui banyak kelas yang ada sebelum memberikan data untuk pelatihan.
- Untuk mendukung dalam masalah dari klasifikasi.
- Memprediksi nilai target numerik dari beberapa data dan label yang diberikan.

Selain itu ada kekurangan dari *supervised learning* yaitu:

- *Supervised learning* tidak dapat memberi informasi yang tidak diketahui dari data pelatihan.
- *Supervised learning* tidak dapat mengelompokkan atau mengklasifikasikan data dengan menemukan fitur nya seorang diri.
- Data bukan dari salah satu kelas dalam data pelatihan, hasilnya masuk kedalam label kelas yang salah.
- Memerlukan banyak waktu untuk komputasi.
- Memprediksi nilai target numerik dari beberapa data dan label yang diberikan.

Supervised learning tipe learning mempunyai variable *input* dan variable *output*, dan menggunakan satu algoritma atau lebih untuk mempelajari fungsi pemetaan dari *input* ke *output*. *Learning* berhenti ketika algoritma mencapai level performa yang diterima. Permasalahan *supervised learning* dapat dikelompokkan menjadi masalah regresi dan masalah klasifikasi.

2.1.2. Deep Unsupervised Learning

Unsupervised learning sebuah pendekatan yang tidak mempunyai data latih sehingga tidak memiliki label. Data latih tersebut akan diproses dengan ekstraksi ciri untuk pemberian label/kelas (Nasteski, 2017). Adapun beberapa algoritma yang menggunakan *unsupervised learning* adalah *Principal Components Analysis* (PCA) dan *Independent Components Analysis* (ICA) (Ghahramani, 2004). Adapun kelebihan dari *unsupervised learning* yaitu:

- Tidak diperlukan pengetahuan sebelumnya tentang area gambar.
- Menghasilkan kelas spektral yang unik.
- Peluang untuk tingkat kesalahan manusia sedikit.
- Relatif mudah.

Selain itu ada kekurangan dari *unsupervised learning* yaitu:

- Tidak dapat mengambil waktu untuk menafsirkan kelas spektral.
- Tidak mempertimbangkan hubungan spasial dalam data.
- Spektral tidak selalu mewakili fitur-fitur di lapangan.

Unsupervised learning tipe learning hanya mempunyai data masukan (input data) tetapi tidak ada output variable yang berhubungan. Permasalahan *unsupervised learning* dapat dikelompokkan menjadi *clustering problems* dan *association problems.*

2.1.3. Deep Reinforcement Learning

Reinforcement learning sebuah pendekatan dalam proses *feedback*, dan akan terus melakukan proses untuk mencapai tujuan nya tersebut. Dan tujuan tersebut tercapai akan mendapatkan *reward*. Proses ini akan terus berlangsung dengan tujuan untuk memaksimalkan *reward* yang didapat. Adapun beberapa algoritma yang menggunakan *reinforcement learning* adalah *Feedforward Neural Network* (FNN) dan *Recurrent Neural Network* (RNN) (Schmidhuber, 2015). Adapun kelebihan dari *Reinforcement learning* yaitu:

- Memaksimalkan Kinerja.
- Mempertahankan Perubahan untuk jangka waktu yang lama.

Selain itu ada kekurangan dari *Reinforcement learning* yaitu:

- Terlalu banyak penguatan yang dapat mengurangi hasil.

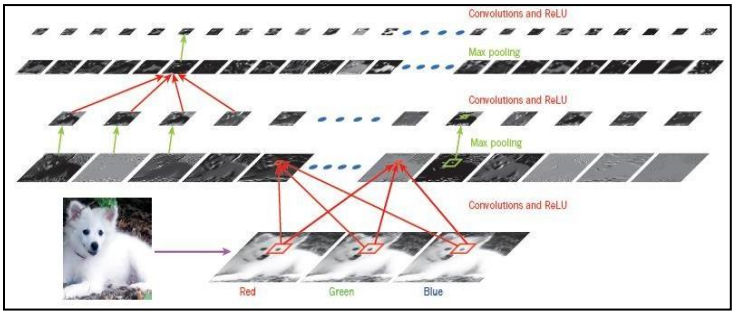
Reinforcement learning yaitu mesin dan *software* dapat menentukan tingkah laku ideal terhadap sebuah konteks yang spesifik secara otomatis, dengan tujuan untuk memaksimalkan performa. Umpan balik (*feedback*) dibutuhkan untuk mesin mempelajari tingkah lakunya, hal ini disebut *reinforcement* signal.

2.2. Image Classification

Image Classification merupakan proses yang dapat mengklasifikasikan gambar sesuai dengan kategori tertentu. Dalam *image classification* mungkin dirancang untuk mengetahui apakah suatu gambar mengandung sosok manusia atau tidak. Saat mendeteksi suatu objek sepele bagi manusia, klasifikasi gambar yang kuat masih merupakan tantangan dalam visi komputer aplikasi (Bow & Bow, 2015).

2.3. Convolutional Neural Network

Convolutional Neural Network (CNN) dirancang untuk memproses suatu data yang ada dalam bentuk banyak array, contohnya gambar warna yang terdiri dari 2D array yang mengandung piksel dalam tiga macam warna yaitu *Red*, *Green*, dan *Blue*. Ada berbagai macan bentuk CNN adalah 1D untuk sinyal dan urutan biasa nya digunakan untuk bahasa, 2D untuk gambar atau suara; dan 3D untuk video atau gambar volumetrik (Lecun et al., 2015). *Convolutional Neural Network* (CNN) merupakan pengembangan dari metode *Multilayer Perceptron* (MLP) yang termasuk kedalam *Deep Neural Network* disebabkan kedalaman lapisan yang berlipat-lipat dan banyak diaplikasikan pada data gambar, yang diberi nama NeoCognitron oleh peneliti Kunihiko Fukushima dari NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Jepang . Model CNN berhasil diterapkan dengan nama LeNet oleh LeChun dari AT&T Bell Laboratories di Holmdel, New Jersey, USA pada penelitian mengenai pengenalan angka dan huruf. CNN telah mengungguli metode *Machine Learning* lainnya seperti SVM, K-NN, dan lain-lain pada klasifikasi objek pada gambar (Eka Putra, 2016).



Gambar 2.1. Arsitektur CNN Secara Umum

(Sumber : Yann LeCun, Yoshua Bengio, Geoffrey Hinton, 2015)

Pada Gambar 2.1 terdapat citra yang diilustrasikan yang terdapat nilai *red*, *green*, dan *blue*. Nilai matriks berwarna *red* masuk ke tahap operasi *convolution*, nilai matriks di ekstraksi fitur dengan nilai kernel yang sudah ditetapkan dari operasi *convolution*. Hasil nilai matriks operasi convolution melakukan proses operasi ReLU *activation*, kemudian hasil dari operasi ReLu *activation* masuk kedalam proses operasi *max pooling*. Selanjutnya masuk kedalam proses operasi *convolution* dan operasi ReLU *activation*, nilai matriks yang sudah diproses kedalam *convolution* dan ReLU *activation* di proses ke operasi *max pooling* kembali sehingga nilai matriks nya semakin kecil dari citra masukan atau ukuran citra semula. Nilai citra tersebut semakin mengecil nilai matriks nya sampai nilai probabilitas antara 0 sampai 1. CNN terdapat 3layer utama, yaitu *convolutional layers, pooling layers,* dan *fully connected layers*. Pada *convolutional layers,* Proses konvolusi memanfaatkan pada gambar terhadap filter atau kernel untuk mengekstraksi fitur dari gambar. Untuk *pooling layer* adalah untuk mereduksi ukuran gambar yang telah didapat dari *convolutional layer*. Sedangkan pada *fully connected layer*, citra yang telah diperkecil pada *pooling layer* dirubah menjadi 1 dimensi agar data dapat diklasifikasikan secara linear. Karena sifat proses konvolusi, maka CNN hanya dapat digunakan pada data yang memiliki struktur dua dimensi seperti gambar dan suara.

2.3.1. Preprocessing

Pada tahap awal proses preprocessing yaitu dilakukan *resize* untuk mengubah ukuran citra dengan memperkecil ukuran citra pada arah horizontal dan/atau vertikal menjadi ukuran 224x224 piksel. Hal ini bertujuan untuk

menyeragamkan ukuran dari masing-masing citra yang digunakan selama proses pelatihan dan pengujian.

Normalized data yaitu normalisasi piksel dangan antara dari 0 sampai 1. *Preprocessing* ini digunakan pada arsitektur LeNet (Choi et al., 2005). Pada operasi ini ditujukan pada persamaan (2.1).

img /= 255

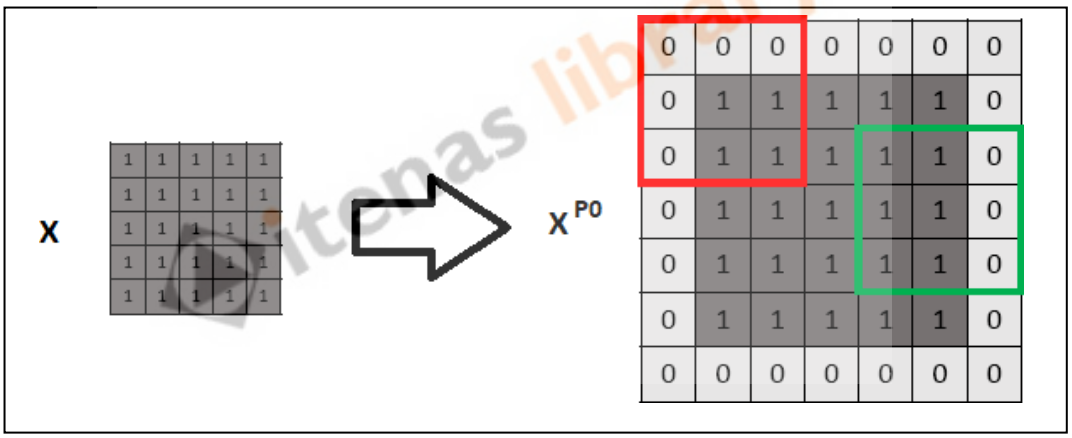
…….(2.1)

Dimana :

Img = nilai matriks dari citra

2.3.2. Padding

Zero padding merupakan operasi penambahan nilai 0 pada kolom dan baris matriks sehingga matriks citra tersebut menjadi lebih besar yang akan menjadikan citra tersebut semakin halus (G. Liu et al., 2018).



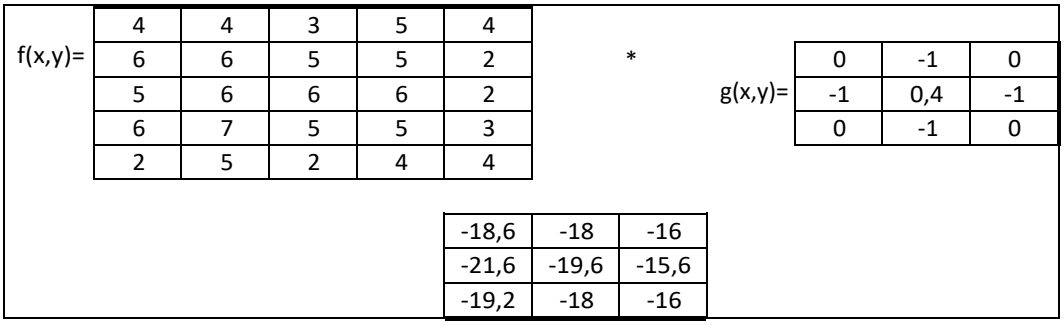
Gambar 2.2. Ilustrasi *Zero Padding*

(Sumber : G. Liu et al., 2018)

2.3.3. Convolution

Convolution layer adalah blok utama di dalam CNN yang terdiri dari beragam filter yang di inisial secara acak untuk melakukan operasi konvolusi yang berfungsi sebagai ekstraksi fitur untuk mempelajari representasi fitur dari suatu input gambar. Pada *convolution layer*, *neuron* tersusun menjadi *feature maps*. Setiap neuron pada feature map sebagai *receptive field*, terhubung pada neuron-neuron dari *convolution layer* sebelumnya melalui serangkaian bobot yang dilatih ,

biasa juga disebut dengan filter bank.(Lecun et al., 2015) Operasi *convolution* diilustrasikan pada Gambar 2.3.



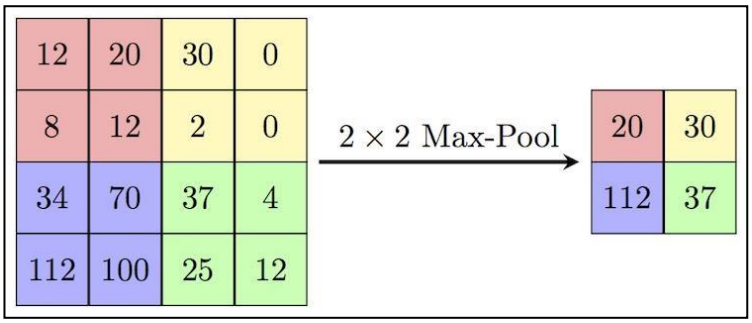
Gambar 2.3. Ilustrasi Operasi *Convolution*

Nilai matriks input f(x,y) dengan 5x5 dengan g(x,y) dengan filter 3x3, dimasukan filter 3x3 dengan nilai matriks input yang pertama [4, 4, 3, 6, 6, 5, 5, 6, 6] di convolution dengan filter [0, -1, 0, -1, 0.4, -1, 0, -1, 0] dengan hasil yang disimpan matriks yang baru. Selanjutnya nilai matriks digeser 1 langkah [4, 3, 5, 6, 5, 5, 6, 6, 6] dengan kernel [0, -1, 0, -1, 0.4, -1, 0, -1, 0] dengan hasil yang disimpan matriks yang baru. Proses terus dilakukan sampai nilai matriks terpenuhi dengan hasil matriks [-18.6, -18, -16, -21.6, -19.6, -15.6, -19.2, -18, -16]. Menurut (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015) , penggunaan *pooling layer* pada CNN hanya bertujuan untuk mereduksi ukuran citra sehingga dapat dengan mudah digantikan dengan sebuah *convolution layer* dengan *stride* yang sama dengan *pooling layer* yang bersangkutan.

2.3.4. Pooling

Pooling adalah untuk melakukan seleksi atau mengurangi resolusi *feature maps* yang tidak terpakai. Setiap peta fitur yang dikumpulkan sesuai dengan satu peta fitur dari lapisan sebelumnya. Dalam sebagian besar CNN, metode subsampling yang digunakan adalah *max pooling*. *Max pooling* membagi output dari *convolution layer* menjadi beberapa grid kecil lalu mengambil nilai maksimal dari setiap grid untuk menyusun matriks citra yang telah direduksi. Sedangkan *average pooling* membagi output menjadi beberapa grid kecil untuk mengambil nilai rata-rata dari setiap grid yang ada pada citra untuk menyusun matriks yang

sudah direduksi (Scherer, Müller, & Behnke, 2010). Operasi *max-pooling* diilustrasikan pada Gambar 2.4.



Gambar 2.4. Ilustrasi *Max Pooling*

Pada kotak sebelah kiri nilai matriks dengan 4x4 yang akan di lakukan operasi *max pooling* dengan filter 2x2 yang ada pada kotak sebelah kanan, pada kotak berwarna merah terdapat nilai [12, 20, 8, 12] lakukan dengan operasi *max pooling* dengan filter 2x2 ambil nilai matriks yang paling terbesar dari kotak matriks merah tersebut dan hasilnya mendapatkan nilai 20. Selanjutnya pada kotak berwarna kuning terdapat nilai [30, 0, 2, 0] lakukan dengan operasi *max pooling* dengan filter 2x2 ambil nilai matriks yang paling terbesar dari kotak matriks kuning tersebut dan hasilnya mendapatkan nilai 30. Selanjutnya pada kotak berwarna biru terdapat nilai [34, 70, 112, 100] lakukan dengan operasi *max pooling* dengan filter 2x2 ambil nilai matriks yang paling terbesar dari kotak matriks biru tersebut dan hasilnya mendapatkan nilai 112. Selanjutnya pada kotak berwarna hijau terdapat nilai [37, 4, 25, 12] lakukan dengan operasi *max pooling* dengan filter 2x2 ambil nilai matriks yang paling terbesar dari kotak matriks hijau tersebut dan hasilnya mendapatkan nilai 37. Hasil akhir dari *max pooling* dengan nilai matriks [20, 30, 112, 37]. *Max pooling* operasi yang mengambil nilai maksimal dari nilai keseluruhan matriks dan hasil dari *max pooling* tersebut dapat ditunjukkan pada Gambar 2.4.

2.3.5. Batch Normalization

Operasi *batch normalization* digunakan untuk mempercepat proses training dan meningkatkan learning rates pada saat pembuatan model. *Batch normalization* bekerja dengan menyamakan distribusi pada setiap nilai input yang selalu berubah dikarenakan perubahan parameter pada *layer* sebelumnya selama proses *training* (Ioffe and Szegedy n.d.,2015). Operasi *batch normalization* dirumuskan pada persamaan (2.2), (2.3), (2.4), dan (2.5).

- Hitung *mini bacth mean* dengan persamaan:

$$\mu B \leftarrow \frac{1}{m}\sum_{i=1}^{m} xi \qquad \text{.....(2.2)}$$

- Hitung *mini batch variance* dengan persamaan:

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(xi - \mu B)^2 \qquad \text{.....(2.3)}$$

- Hitung normalisasi dengan persamaan:

$$\hat{x}i \leftarrow \frac{xi - \mu B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \text{.....(2.4)}$$

- Hitung *scale and shift* dengan persamaan:

$$yi \leftarrow \gamma \hat{x}i + \beta = BN\gamma,\beta(xi) \qquad \text{.....(2.5)}$$

Keterangan:

μB = merupakan nilai rata-rata dari *batch*

m = jumlah dataset pelatihan

ϵ = merupakan nilai konstan yang meningkatkan stabilitas numerik ketika batch variance sangat kecil

γ dan β = parameter pada saat pelatihan γ = 1 dan β = 0

$\epsilon = 1e - 8$

2.3.6. ReLU *Activation*

Rectified Linear Units (ReLU) untuk fungsi aktivasi yang diperkenalkan oleh Geoffrey Hinton dan Vinod Nair dan digunakan pada *neural network*, digunakan untuk mengubah nilai x menjadi 0 jika nilai x tersebut bernilai negatif, sedangkan sebaliknya untuk nilai x tetap dipertahankan apabila nilai tidak kurang dari 0 (Agarap, 2018). Operasi ReLU *activation* bisa ditunjukkan pada persamaan (2.6) dan Gambar 2.5.

$$f(xi) = \max(0, xi) \begin{cases} xi, if\ xi \geq 0 \\ 0, if\ xi < 0 \end{cases} \quad \text{.....(2.6)}$$

Keterangan :

$f(xi)$ = nilai dari ReLU *activation*

Xi = nilai matriks dari citra

4,7	5,2	-2,3	→	4,7	5,2	0
-1,3	2,2	4		0	2,2	4
9	7	5		9	7	5

Gambar 2.5. ReLU *Activation*

2.3.7. Softmax *Activation*

Softmax diterapkan pada lapisan terakhir pada jaringan saraf. Softmax lebih dari itu umum digunakan daripada ReLU, sigmoid atau tanh (). Ini digunakan untuk menghitung probabilitas distribusi dari vektor bilangan real. Fungsi Softmax menghasilkan output yang merupakan kisaran nilai antara 0 dan 1, dengan jumlah probabilitas sama dengan 1 (Nwankpa, Ijomah, Gachagan, & Marshall, 2018). Persamaan softmax *activation* ditunjukan pada persamaan (2.7).

$$\hat{y} = \frac{\exp(xi)}{\sum_j exp^{(xi)}} \quad \text{.....(2.7)}$$

Keterangan :

$\hat{y}$ = hasil dari fungsi softmax

Xi = kelas ke i (i=1,2..dst). pada penelitian ini memakai 2

kelas J = nilai dari vektor

2.3.8. Cross Entropy

Cross entropy merupakan fungsi untuk kerugian dan gradien digunakan dalam *multi-classification.* Untuk mengukur entropi relatif antara dua distribusi probabilitas pada data yang sama. *Cross entropy* fungsi kerugian yang digunakan ketika melatih model *neural network*. Cara kerjanya mengurangi log negatif dari dataset (Borovcnik, Bentz, & Kapadia, 1991). Cross entropy terdapat persamaan (2.8).

$$L_{cross-entropy} = -\sum_{j} y_i \log(\hat{y}) \quad \text{.....(2.8)}$$

Keterangan:

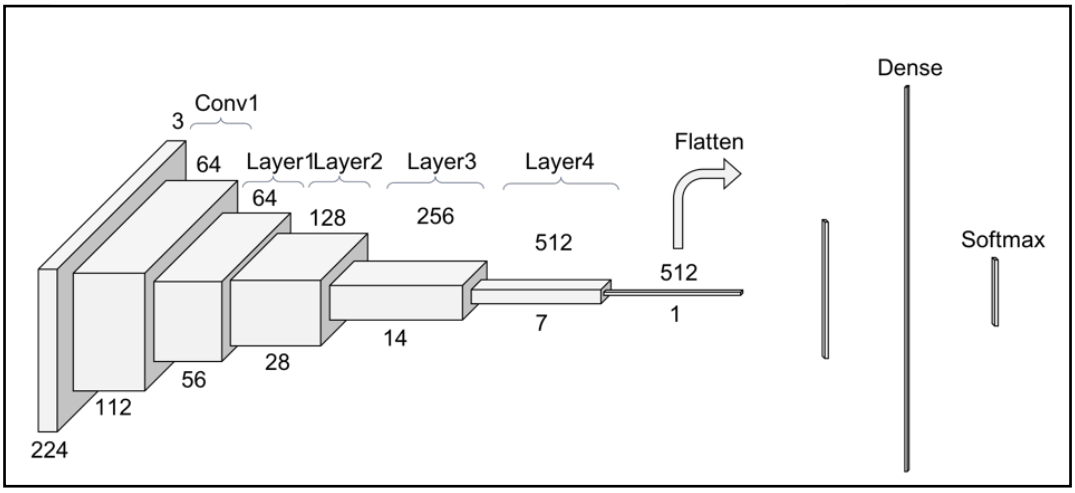
$\hat{y}$=hasil dari softmax activation

y_i = nilai dari kelas, bernilai 1 jika kelas yang benar dan bernilai 0 jika kelas yang salah

j= kelas ke-j(1,2)

2.4. Arsitektur ResNet

Residual Network (ResNet) memenangkan *ILSVRC* pada tahun 2015 dengan menggunakan skip connection dan features batch normalization. Jaringan ini memungkinkan pelatihan Neural Network dengan 152 layer dan mampu mengurangi kompleksitas dibandingkan dengan VGGNet. Tingkat kesalahan yang dicapai dari arsitektur ini adalah sebesar 3.57% pada Top-5, sehingga mampu mengalahkan kinerja human-level pada dataset yang diberikan.

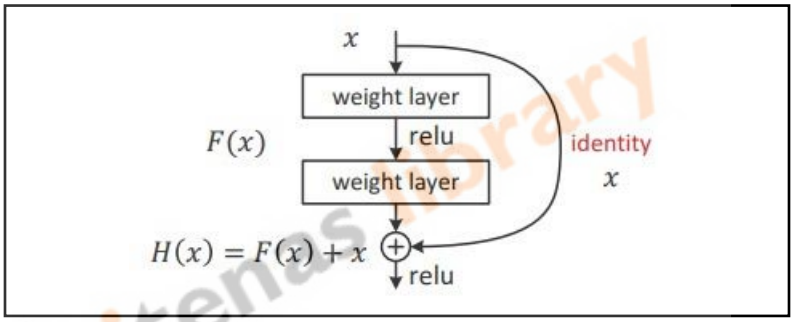


Gambar 2.6. Contoh Jaringan ResNet

(Sumber: medium.com)

Deep Residual Network atau yang biasa disebut sebagai *ResNet* merupakan salah satu arsitektur dari *CNN* yang diusulkan oleh *He*. pada tahun 2015. Arsitektur ini dibagun untuk mengatasi permasalahan pada pelatihan *Deep Learning*, karena pelatihan *Deep Learning* pada umumnya memakan cukup banyak waktu dan terbatas pada jumlah lapisan tertentu. Solusi permasalahan yang diusulkan oleh *ResNet* adalah dengan menerapkan *skip connection* atau *shortcut*. Kelebihan model *ResNet* dibandingkan dengan model arsitektur *CNN* yang lain adalah kinerja dari model ini tidak menurun walaupun arsitekturnya semakin dalam. Selain itu, perhitungan komputasi yang dilakukan lebih ringan dan kemampuan untuk melatih jaringan yang lebih baik.

Model *ResNet* diimplementasikan dengan melakukan *skip connection* pada dua sampai tiga layer yang mengandung *ReLU* dan *batch normalization* di antaraarsitekturnya *He* (2016).



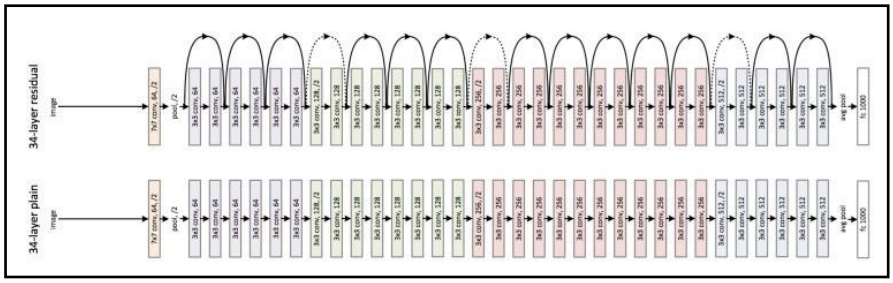
Gambar 2.7. Blok Deep Residual Network

(Sumber: Deep Residual Learning for Image Recognition, 2015)

He mengadopsi pembelajaran residual untuk diterapkan pada beberapa tumpukan layer. *Residual Block* pada *ResNet* didefinisikan sebagai berikut.

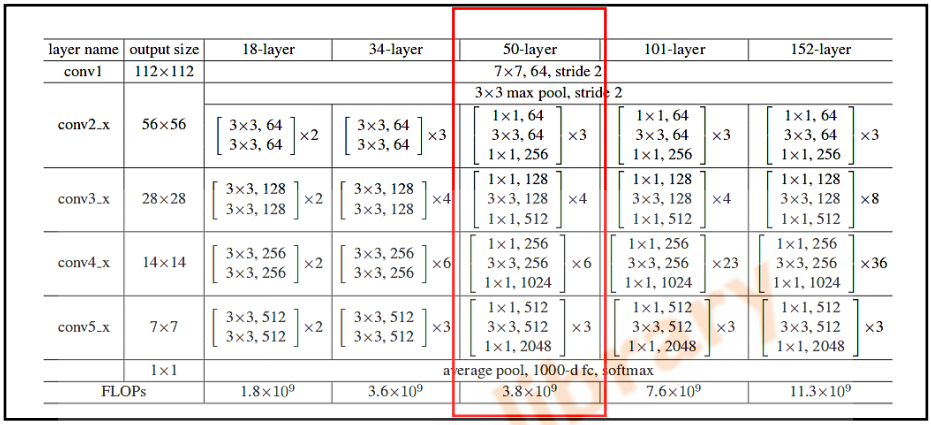
$$y = F(x, W_i + x)$$

Dimana x dan y merupakan vektor masukan dan keluaran dari layer. Kemudian fungsi F merepresentasikan oleh residual map untuk dipelajari (He, 2016).



Gambar 2.8. Perbandingan Jaringan Biasa dengan Jaringan ResNet

Residual Block pada *ResNet* dapat dilakukan apabila dimensi data masukan sama dengan dimensi data keluaran. Selain itu, Setiap blok *ResNet* terdiri dari 2 layer (untuk jaringan *ResNet 18* dan *ResNet 34*) atau 3 layer (untuk jaringan *ResNet 50, 101, 152*). Dua lapisan awal dari arsitektur *ResNet* menyerupai *GoogleNet* dengan melakukan convolution 7 × 7 dan *max pooling* berukuran 3 × 3 dengan jumlah stride 2. *ResNet* memiliki beberapa macam jenis arsitektur yang dibedakan berdasarkan jumlah layer yang digunakan, mulai dari 18 layer, 34 layer, 50 layer, 101 layer, sampai 152 layer (He, 2016).



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Gambar 2.9. Arsitektur ResNet

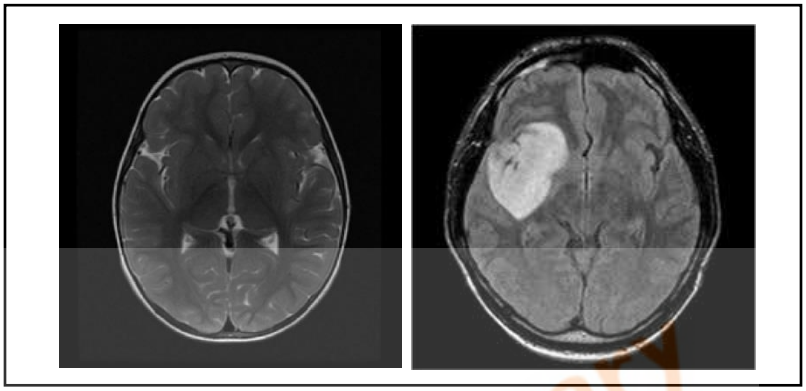
(Sumber : Deep Residual Learning for Image Recognition, 2015)

Untuk setiap komposisi lapisan menggunakan *Batch Normalization*,ReLU dan 3x3 *Convolution*. Pada setiap blok ada input berupa matriks sesuai dengan pixel citra kemudian masuk ke proses *Batch Normalization* untuk mengurangi adanya *overfitting* pada saat proses *training*, ReLU untuk hanya membuat pembatas pada bilangan nol, artinya apabila $x \leq 0$ maka $x = 0$ dan apabila $x > 0$ maka $x = x$, 3x3 *Convolution* proses dimana citra matrik yang sudah melewati proses ReLU akan dikalikan dengan matrik 3x3 *Convolution* dan output yang dihasilkan berupa matriks yang sudah di proses sebelumnya. Yang diilustrasikan pada Gambar 2.9.

Bottleneck: Bahwa *convolution* dengan filter 1×1 dapat disebut sebagai lapisan *bottleneck* sebelum setiap *convolution* dengan filter 3×3 untuk mengurangi jumlah peta fitur masukan, dengan meningkatkan efisiensi komputasi.

2.5. Tumor Otak

Tumor otak merupakan salah satu bagian dari tumor pada sistem saraf, di samping tumor *spinal* dan tumor saraf *perifer*. Tumor otak dapat berupa tumor yang sifatnya primer ataupun yang merupakan metastasis dari tumor pada organ lainnya. Menurut *International Agency for Research on Cancer*, lebih dari 126.000 orang di dunia setiap tahunnya mengidap penyakit tumor otak dan lebih dari 97.000 orang meninggal dunia.



Gambar 2.10. Tumor Otak dan Bukan Tumor Otak

Penyebab tumor otak dan tumor primer lain biasanya sama. Menurut *American Cancer Society*, peneliti menyebut penyebab tumor otak bisa terjadi karena perubahan DNA. DNA adalah bahan kimia yang membentuk gen dan mengontrol bagaimana setiap sel berfungsi. Umumnya, sel manusia bisa tumbuh dan berfungsi dengan normal mengikuti informasi dari DNA setiap sel. Gen dapat mengontrol kapan sel tumbuh, membelah menjadi sel baru, dan mati. Tumor ganas dapat muncul saat DNA mengalami mutasi dan mengaktifkan gen yang mengatur pertumbuhan sel (onkogen). Sel yang semula normal juga bisa jadi tak terkendali saat gen pengendali pertumbuhan sel bermutasi, dan mengubah fungsinya jadi tidak aktif. Dengan mutasi gen tersebut, sel jadi terus tumbuh, membelah dengan cepat, dan tak terkendali sampai membentuk massa sel *abnormal* yang disebut tumor. Perubahan gen ini terkadang terjadi karena faktor keturunan atau faktor eksternal seperti gaya hidup. (Afifah, 2020)

2.6. Pengujian Kinerja Sistem

Dalam penelitian ini untuk mengukur kinerja sistem dari klasifikasi citra sama halnya dengan sistem information retrieval, yaitu dengan mengukur *accuracy, precision, recall,* dan *f-measure*. *Precision* bagian dari citra yang diambil dengan tepat/relevan. Sedangkan *recall* bagian dari citra yang tepat/relevan yang diambil oleh sistem. Sedangkan *accuracy* merupakan tingkat kedekatan antara nilai prediksi dengan nilai aktual. *F-Measure* merupakan ukuran akurasi uji dari perhitungan *precision* dan *recall* (Pardede & Husada, 2016).

Tabel 2.1 Confusion Matrix

Kejadian	*Positive*	*Negative*
	True Positive(TP)	*True Negative*(TN)
	False Positive(FP)	*False Negative*(FN)

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad \text{.....(2.9)}$$

$$Precision = \frac{TP}{TP+FP} \quad \text{.....(2.10)}$$

$$Recall = \frac{TP}{TP+FN} \quad \text{.....(2.11)}$$

$$F\text{-}Measure = 2\ x\ \frac{(PrecisionxRecall)}{(Precision+Recall)} \quad \text{.....(2.12)}$$

Dimana:

TP = Memprediksi jumlah citra yang benar

TN = Memprediksi jumlah yang bukan citra yang tidak ada penyakit.

FP = Memprediksi jumlah citra yang salah.

FN = Memprediksi jumlah citra yang salah klasifikasi.

2.7. Studi Kasus

2.7.1.*Preprocessing*

Studi kasus dengan *preprocessing* citra dengan tahap pertama di *resize* menjadi 224x224 piksel. Kemudian di proses menggunakan *normalized* data yang memakai persamaan (2.1).

$$\text{Elemen matriks piksel citra img} = \begin{bmatrix} 145 & 67 & 122 \\ 123 & 125 & 170 \\ 97 & 178 & 102 \end{bmatrix} /255$$

Sehingga setelah di *normalized* data nilai citra tersebut berubah seperti:

$$\text{Elemen matriks piksel citra img} = \begin{bmatrix} 0.5686 & 0.2627 & 0.4784 \\ 0.4824 & 0.4901 & 0.6667 \\ 0.3803 & 0.6980 & 0.4 \end{bmatrix}$$

2.7.2. Convolution

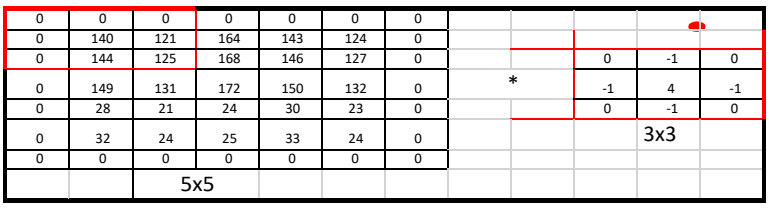
Studi kasus pada proses *convolution* yaitu menggunakan matriks hasil dari *preprocessing* dilakukan pada matriks yang sudah di *zero padding* terhadap suatu filter berukuran 3x3. Digunakan sebanyak 64 unit *neuron* dengan perkalian 3x3, dan *padding* = "*same*". Jumlah *neuron* pada *hidden layer* adalah 64 , itu artinya gambar yang dihasilkan mempunyai 64 *feature maps*. Namun dikarenakan piksel citra berukuran 224x224 piksel, maka diambil matriks 6x6 sehingga dapat memahami konsep dari *convolution* nya seperti di Gambar 2.11.

0	0	0	0	0	0	0						
0	140	121	164	143	124	0						
0	144	125	168	146	127	0			0	-1	0	
0	149	131	172	150	132	0		*	-1	4	-1	
0	28	21	24	30	23	0			0	-1	0	
0	32	24	25	33	24	0				3x3		
0	0	0	0	0	0	0						
		5x5										

Gambar 2.11. Ilustrasi Proses *Convolution*

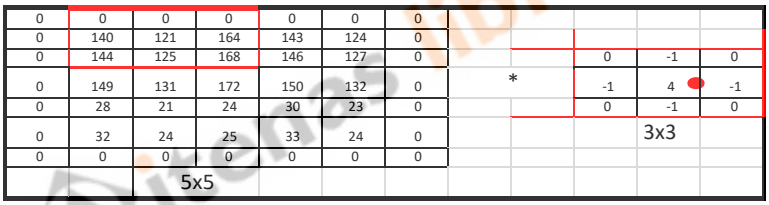
Untuk mengetahui hasil operasi *convolution* tersebut, berikut merupakan langkah-langkah perkalian proses *convolution* antara matriks 6x6 dengan filter 3x3:

1. Untuk mendapatkan nilai matriks dimulai pada baris 1, 2, dan 3 ke kolom 1 di kalikan dengan filter 3x3 pada Gambar 2.11, jumlahkan (0*0)+(0*-1)+(0*0)+(0*-1)+(140*4)+(121*-1)+(0*0)+(144*-1)+(125*0)=295.



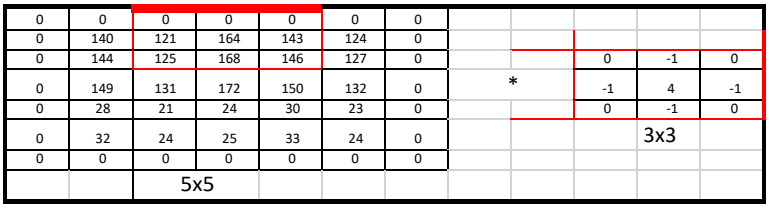
Gambar 2.12. Ilustrasi Proses *Convolution* Langkah 1

2. Untuk mendapatkan nilai matriks pada baris 1, 2, dan 3 ke kolom 2 di kalikan dengan filter 3x3 pada Gambar 2.12, jumlahkan (0*0)+(0*-1)+(0*0)+(140*-1)+(121*4)+(164*-1)+(144*0)+(125*1)+(168*0)=55, yang diilustrasikan pada Gambar 2.13.



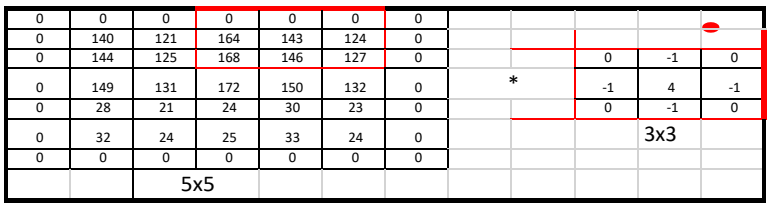
Gambar 2.13. Ilustrasi Proses *Convolution* Langkah 2

3. Untuk mendapatkan nilai matriks pada baris 1, 2, dan 3 ke kolom 3 di kalikan dengan filter 3x3 pada Gambar 2.13, jumlahkan (0*0)+(0*-1)+(0*0)+(121*-1)+(164*4)+(143*-1)+(125*0)+(168*-1)+(146*0)=224.



Gambar 2.14. Ilustrasi Proses Convolution Langkah 3

4. Untuk mendapatkan nilai matriks pada baris 1, 2, dan 3 ke kolom 3 di kalikan dengan filter 3x3 pada Gambar 2.14, jumlahkan (0*0)+(0*-1)+(0*0)+(164*-1)+(143*4)+(124*-1)+(168*0)+(146*-1)+(127*0)=138.



Gambar 2.15. Ilustrasi Proses *Convolution* Langkah 4

5. Untuk mendapatkan nilai matriks pada baris 2, 3, 4, dan 5 lakukan langkah seperti langkah 1, 2, 3, dan 4 yang diilustrasikan pada gambar 2.15. Pada Gambar 2.16. yang diilustrasikan hasil dari perkalian *convolution* dengan filter 3x3.

295	55	224	138	226
162	-64	65	-4	106
293	57	215	120	228
-90	-123	-152	-110	-94
76	18	19	53	40

Gambar 2.16 Hasil Matriks *Convolution* dengan Filter 3x3

2.7.3. Batch Normalization

Studi Kasus operasi *batch normalization* diimplementasikan dengan matriks 5x5 dari hasil *convolution* yang diilustrasikan pada Gambar 2.15. Berikut merupakan langkah-langkah dari *batch normalization*:

1. *Mini Batch Mean*

   Lakukan operasi ini dengan yang memakai persamaan (2.2) pada sub bab 2.3.5 jumlahkan baris matriks dan dibagi dengan baris kolomnya. Seperti nilai matriks [295, 55, 224, 138, 226] nilai tersebut didapat pada hasil *convolution* dengan filter 3x3. Nilai tersebut seperti yang sudah dijelaskan dibagi dengan jumlah nilai pada baris ke 1. Pada proses ini akan menghasilkan matriks 5x1 seperti Gambar 2.17.

- (295+55+224+138+226)/5= 187,6
- (162+(-64)+65+(-4)+106)/5= 53
- (293+57+215+120+228)/5= 182,6
- (-90+(-123)+(-152)+(-110)+(-94))/5= -113,8
- (76+18+19+53+40)/5= 41,2

187,6	53	182,6	-113,8	41,2

Gambar 2.17 Matriks *Mini Batch Mean*

2. Mini Batch Varian

Menghitung nilai matriks *Mini Batch Varian* dengan yang memakai persamaan (2.3).

a. Nilai dari hasil operasi *convolution* dikurangi dengan hasil operasi *batch normalization* sub operasi *mini batch mean*. Seperti nilai dari *convolution* pada baris ke-1 kolom-1 295 sedangkan nilai *mini batch mean* pada baris ke-1 kolom ke-1 187,6. Hasil dari pengurangan baris ke-1 kolom ke-1 107,4 dan hasilnya disimpan pada *feature maps*, Selanjutnya *convolution* baris ke-1 kolom ke-2 dengan nilai 55, baris ke-1 kolom ke-2 dengan nilai 53 yang hasil pengurangannya sebesar 2.

- 295-187,6= 107,4
- 55-53=2
- 224-182,6=41,4
- 138-(-113,8) =251,8
- 226-41,2=184,8

Lakukan operasi tersebut untuk baris ke-2 sampai dengan baris ke-5 pada hasil operasi *convolution*, dan ini hasil dari operasi *convolution* dengan pengurangan sub proses *mini batch varian* yang sudah dilakukan. Diilustrasikan pada Gambar 2.18.

107,4	2	41,4	251,8	184,8
-25,6	-117	-117,6	109,8	64,8
105,4	4	32,4	233,8	186,8
-277,6	-176	-334,6	3,8	-135,2
-111,6	-35	-163,6	166,8	-1,2

Gambar 2.18 Matriks hasil Pengurangan *Convolution* Dengan *Mini Batch Mean*

b. Dipangkatkan setiap nilai matriks hasil dari pengurangan antara *convolution* dengan sub proses *mini batch mean*. Pada Gambar 2.17 baris ke-1 kolom ke-1 dengan nilai 107.4 dipangkatkan nilai tersebut yang hasilnya mendapatkan nilai 11534.76, langkah tersebut dilakukan pada baris ke-1 sampai dengan baris ke-5, sehingga menghasilkan nilai matriks dipangkatkan dari langkah 2a seperti Gambar 2.19.

11534,76	4	1713,96	63403,24	34151,04
655,36	13689	13829,76	12056,04	4199,04
11109,16	16	1049,76	54662,44	34894,24
77061,76	30976	111957,2	14,44	18279,04
12454,56	1225	26764,96	27822,24	1,44

Gambar 2.19 Matriks Hasil Dipangkatkan

c. Jumlahkan baris ke-i matriks dan dibagi dengan panjang barisnya nya pada Gambar 2.18. Seperti baris ke-1 dengan nilai matriks [11534.76, 4, 1713.96, 63403.24, 34151.04] dan dibagi dengan jumlah panjang baris yang bernilai 5 dan hasilnya 22161.4. Lakukan langkah tersebut sampai dengan baris ke-5. Pada proses ini akan menghasilkan matriks 5x1 seperti yang diilustrasikan Gambar 2.20.
   - (11534,76+4+1731,96+63403,24+34151,04)/5= 22161,4
   - (655,36+13689+13829,76+12056,04+4199,04)/5=8885,84
   - (11109,16+16+1049,76+54662,44+34894,24)/5= 20346,32
   - (77061,76+30976+111957,2+14,44+18279,04)/5= 47657,68
   - (12454,56+1225 +26764,96+27822,24+1,44)/5= 13653,64

22161,4	8885,84	20346,32	47657,68	13653,64

Gambar 2.20 Matriks Hasil Operasi Langkah 2c

3. Hitung normalisasi

a. Hitung akar pangkat pada persamaan (2.4) dari hasil pengjumlahan setiap nilai matriks pada hasil *mini batch vairan* langkah 2c. Seperti pada baris ke-1 kolom ke-1 dengan nilai 22161.4 di akar pangkat nilai kemudian jumlahkan dengan nilai 10-8 dengan hasil 148.8671. Langkah tersebut dilakukan sampai dengan kolom ke-5, sehingga menghasilkan nilai matriks seperti pada Gambar 2.21.

148,8671	94,26473	142,6405	218,3064	116,8488

Gambar 2.21 Matriks Hasil Operasi Langkah 3a

b. Hitung setiap baris matriks hasil dari langkah 2a dibagi dengan hasil matriks langkah 3a, seperti nilai matriks pada baris ke-1 kolom ke-1 dengan nilai 107.4 dengan nilai matriks yang ada pada langkah 3a dengan nilai 148.8671 dengan membagi nilai dengan hasil 0.72144908. Membagi nilai langkah 2a dengan nilai 3a yang diilustrasikan seperti pada Gambar 2.22.

- 107,4/148,8671=0,721449082
- 2/94,26473=0,021216842
- 41,4/142,6405272=0,290240094
- 251,8/218,3063902=1,153425
- 184,8/116,8487912=1,581531

0,72144908	0,021217	0,29024	1,153425	1,581531
-0,1719655	-1,24119	-0,82445	0,502963	0,554563
0,70801428	0,042434	0,227144	1,070972	1,598647
-1,8647511	-1,86708	-2,34576	0,017407	-1,15705
-0,7496622	-0,37129	-1,14694	0,764064	-0,01027

Gambar 2.22 Matriks Hasil *Batch Normalization*

2.7.4. ReLU *Activation*

Pada proses dengan persamaan (2.6) pada sub bab 2.3.6 ini dilakukan pada setiap matriks pada gambar dengan mengubah nilai negatif menjadi sama dengan 0, dan mempertahankan yang nilainya lebih besar dari 0. Pada hasil *batch normalization* dengan sub proses normalisasi dengan baris ke-1 kolom ke-1 nilai matriks 0,721449 selanjutnya lakukan operasi ReLU *activation* dengan hasil nilai 0,721449 dikarenakan nilai nya dipertahankan. Lakukan langkah tersebut sampai baris ke-5 kolom ke-5. Seperti yang sudah dijelaskan sebelumnya operasi ReLU *activation* ini mengubah nilai matriks yang bernilai negatif (-) menjadi 0. Sehingga nilai matriks hasil ReLU *activation* yang diilustrasikan pada Gambar 2.23.

0,721449	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,708014	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.23 Matriks Hasil ReLU *activation*

2.7.5. Max Pooling

Studi kasus *max pooling* yang dilakukan yaitu menggunakan perkalian sebesar 3x3 dan *strides* = 2 terhadap matriks hasil ReLU *activation* pada Gambar 2.19.

1. Langkah pertama pada kotak merah dengan operasi *max pooling* dicari nilai terbesar dari nilai pada kotak merah yang hasil nilai matriks nya 0.721449 yang diilustrasikan pada Gambar 2.24.

0,721449	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,708014	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.24 Matriks Perhitungan dari *Max Pooling* Langkah 1

2. Langkah selanjutnya terlebih dahulu menggunakan pergeseran 2 strides seperti pada kotak biru dengan operasi *max pooling* dicari nilai terbesar dari nilai pada kotak biru yang hasil nilai matriks nya 1.598647 yang diilustrasikan pada Gambar 2.25.

0,721449	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,708014	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.25 Matriks Perhitungan dari *Max Pooling* Langkah 2

3. Langkah selanjutnya terlebih dahulu menggunakan pergeseran 2 strides seperti pada kotak hijau dengan operasi *max pooling* dicari nilai terbesar dari nilai pada kotak hijau yang hasil nilai matriks nya 0.708014 yang diilustrasikan pada Gambar 2.26.

0,721449	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,708014	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.26 Matriks Perhitungan dari *Max Pooling* Langkah 3

4. Langkah selanjutnya terlebih dahulu menggunakan pergeseran 2 strides seperti pada kotak ungu dengan operasi *max pooling* dicari nilai terbesar dari nilai pada kotak ungu yang hasil nilai matriks nya 1.598647 yang diilustrasikan pada Gambar 2.27.

0,721449	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,708014	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.27 Matriks Perhitungan dari *Max Pooling* Langkah 3

5. Hasil dari langkah 1 sampai langkah 4 dari *max pooling* yang diilustrasikan pada Gambar 2.28.

0,721449	1,598647
0,708014	1,598647

Gambar 2.28 Matriks Hasil dari *Max Pooling*

2.7.6. Global Average Pooling

Studi kasus *global average pooling* dengan filter 7x7 yang dapat dilihat pada Gambar 2.29.

0,72144908	0,021217	0,29024	1,153425	1,581531
0	0	0	0,502963	0,554563
0,70801428	0,042434	0,227144	1,070972	1,598647
0	0	0	0,017407	0
0	0	0	0,764064	0

Gambar 2.29 Matriks dari ReLU *Activation*

Operasi ini dijumlahkan semua elemen matriks yang ada dibagi dengan jumlah elemen matriks yang ada, seperti pada Gambar 2.30:

- (0,72144908+0,021217+0,29024+1,153425+1,581531+0+0+0+0,502963+0,554563+0,70801428+0,042434+0,227144+1,070972+1,598647+0+0+0+0,017407+0+0+0+0,764064+0)/25=0,370163

0,72144908	0,021217	0,29024	1,153425	1,581531	
0	0	0	0,502963	0,554563	
0,70801428	0,042434	0,227144	1,070972	1,598647	0,370163
0	0	0	0,017407	0	
0	0	0	0,764064	0	

Gambar 2.30 Matriks Hasil *Global Average Pooling*

2.7.7. Softmax *Activation*

Studi kasus dari softmax *activation* hasil dari operasi *global average pooling* dengan filter 7x7 dengan persamaan (2.7).

1. Hasil dari matriks *global average pooling* [0.370163, 0.629837]. keterangan untuk nilai 0.370163, 0.629837 hasil dari operasi *global average pooling* dimana nilai tersebut secara keseluruhan bernilai 1.
2. Jumlahkan hasil dari matriks *global average pooling* dengan eksponen.

$$\sum_j exp^{(xi)} = e^{0,370163} + e^{0,629837} = 1.4479706146 + 1.8773045537 = 3.3252751683$$

3. Lakukan pembagian antara eksponen nilai *global average pooling* dengan nilai penjumlahan eksponen nya.

$$\hat{y} = \frac{exp^{0,370163}}{3.3252751683}, \frac{exp^{0,629837}}{3.3252751683}$$

$$\hat{y} = [\,0.44, 0.56]$$

2.7.8. Cross Entropy

Cross entropy loss untuk melihat nilai *error* yang ada pada dataset. Pada penelitian ini menggunakan *categorical cross entropy* yang memiliki pada persamaan (2.8).

- Masukan nilai hasil dari softmax *activation.*

$$\hat{y} = \begin{bmatrix} 0.44 \\ 0.56 \end{bmatrix}, \mathrm{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$L_{cross-entropy} = -0\log(0.44) - 1\log(0.56)$$

$$L_{cross-entropy} = -0 - 1\log(0.56) \approx 0.2518$$