BAB II
LANDASAN TEORI

Berikut ini merupakan beberapa teori yang digunakan dalam peneltian yang dilakukan.

2.1 *Machine Learning*

Machine learning merupakan bidang studi yang didasari oleh gagasan bahwa mesin dapat belajar sendiri tanpa diprogram secara eksplisit. Data yang digunakan sistem untuk belajar disebut *dataset,* setiap contoh pelatihan disebut *trainingset* atau sampel. Semakin banyak data, semakin baik pembelajarannya (Hahn, 2019). Secara umum tipe belajar dari *machine learning* terbagi menjadi tiga metode yaitu *Supervised Laerning, Unsupervised Learning,* dan *Reinforcemen Learning*.

1. *Supervised Learning*

Supervised Learning adalah pembelajaran ML untuk menyelesaikan masalah rumit melalui *training* di dalam sistem saraf buatan menggunakan data yang sudah diberikan label. Label adalah *tag* dari data yang digunakan dalam model *machine learning*. *Supervised* lebih banyak digunakan dalam tugas klasifikasi dan regresi (Mitchell, 1997).

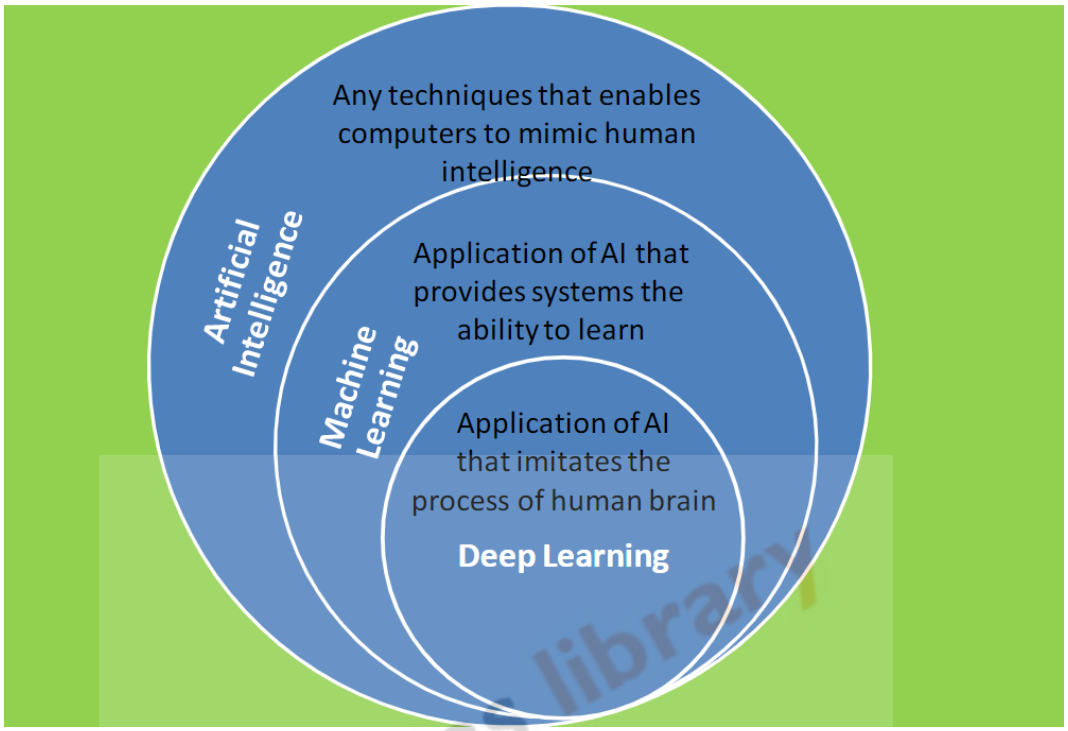
2. *Unsupervised Learning*

Unsupervised Learning adalah pembelajaran ML dengan menggunakan pola data yang sebelumnya belum terlihat atau belum terpikirkan. Teknik ini digunakan untuk mengidentifikasi pola-pola tersembunyi dalam data input yang tidak berlabel dan mengolah informasi melalui karakteristik yang terdapat pada pola tersebut dan biasa digunakan dalam tugas *clustering* (Mitchell, 1997).

3. *Reinforcemen Learning*

Reinforcemen Learning adalah pembelajaran ML yang mengumpulkan informasi dengan berinteraksi pada lingkungan dan dapat belajar dari kesalahan untuk mendapatkan balasan atau tanggapan balik positif maupun negatif untuk

berkembang, biasanya digunakan pada *logic* game dan mobil otomatis. Pada Gambar 2.1 dijabarkan cabang dari studi *machine learning*.



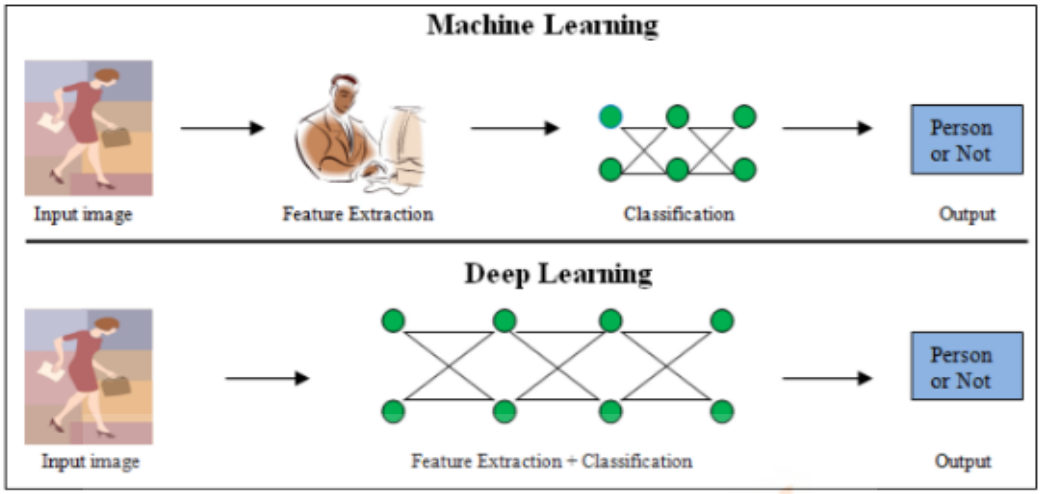
Gambar 2.1 Hubungan *AI*, *machine learning*, dan *deep learning*
Sumber : (Hahn, 2019)

2.2 *Deep Learning*

Deep Learning merupakan sub-bidang dari *machine learning* yang di perkenalkan pada tahun 1986, lalu diterapkan pada tahun 2000 pada jaringan saraf tiruan atau *Artificial Neural Network* (ANN) (Schmidhuber, 2015). *Deep learning* dapat mengajarkan komputer untuk melakukan pekerjaan seperti manusia. *Deep learning* memiliki jaringan yang berjumlah ratusan lapisan atau *layer* bahkan lebih banyak lagi, itu sebabnya disebut *deep learning.*

Salah satu potensi dari *deep learning* adalah mengganti fitur buatan tangan dengan algoritma yang efisien untuk pembelajaran *hirarkis unsupervised* (tanpa pengawasan) atau *semi-supervised feature learning (*semi-diawasi) dan *hierarchical feature extraction* (esktraksi fitur) (Schmidhuber, 2015). Penerapan *deep learning* telah digunakan dalam beberapa bidang seperti klasifikasi gambar, klasifikasi video, deteksi objek, pengenalan pola, *text-to-speech, natural language*

processing, robotik, dan klasifikasi teks. Gambar 2.2 adalah ilustrasi perbedaan *Machine Learning* dan *Deep Learning*.



Gambar 2.2 Perbedaan *mahine learning* dan *deep learning*
Sumber : (Krishna & Kalluri, 2019)

2.3 *Image Classification*

Image classification merupakan proses yang dirancang untuk mengklasifikasi gambar sesuai dengan kategori tertentu. Dalam tugas klasifikasi gambar, pemetaan fitur dari gambar ke label kelas diwakili oleh vektor fitur atau piksel gambar.

Image classification memainkan peran penting dalam *computer vision* dan aplikasinya, contohnya seperti mengkategorikan gerakan dan *image retrieval* (Fang et al., 2019). Metode berbasis jaringan *convolutional neural network* telah menunjukan kinerja yang baik dalam klasifikasi gambar, yang bertujuan untuk mencari tau fitur-fitur dari gambar pelatihan.

2.4 *Pre-processing*

Pre-processing data merupakan standar operasi untuk melakukan pelatihan data pada *neural network* untuk menyeragamkan ukuran citra. Dilakukan *resize* ke ukuran 224x224 untuk memperkecil ukuran citra pada arah horizontal dan vertikal. Kemudian melakukan normalisasi data dengan operasi pembagian setiap piksel nilai gambar dengan 255 untuk mengubah nilai piksel kedalam rentang [0,1] agar

mengurangi biaya komputasi (Tammina, 2019). Operasi ini ditujukan pada Persamaan (2.1).

$$image_{x,y} = \frac{image_{x,y}}{255} \quad \text{.....(2.1)}$$

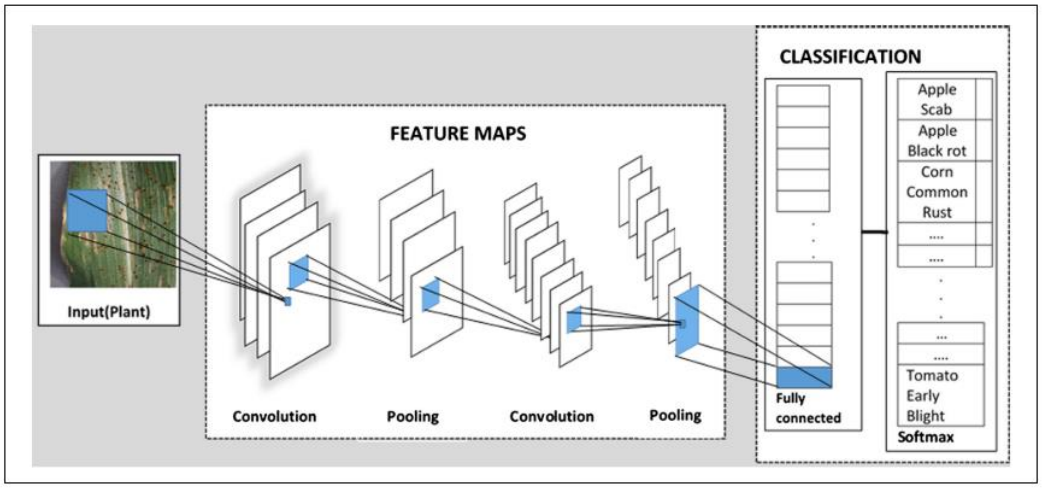
Sumber : (Tammina, 2019)

2.5 *Convolutional Neural Network*

Convolutional Nueral Network (CNN) merupakan pengembangan dari metode MLP (*Multilayer Perceptron*) yang awalnya diberinama *NeoCognitron* oleh peneliti bernama Kunihiko Fukushima yang berasal dari NHK *Broadcasting Science Research Laboratories*, Tokyo, Jepang. Kemudian YanLecun dan teman teman mematangkan MLP menjadi arsitektur LeNet dengan penelitiannya tentang pengenalan tulisan tangan (LeCun, Bottou, Bengio, & Haffner, 1998).

Penerapan CNN terus berkembang dan Alex Krizhevsky berhasil menerapkan CNN dengan menjuarai kompetisi *ImageNet Large Scale Visual Recognition Challenge* pada tahun 2012 dengan model yang bernama AlexNet. Pada perkembangannya, terdapat banyak arsitektur CNN yang sering digunakan pada penelitian seperti AlexNet, VGGNet, GoogLeNet, ResNet dll.

Convolutional Nueral Network termasuk dalam jenis *deep learning* karena kedalaman jaringannya. CNN merupakan operasi konvolusi yang menggabungkan beberapa lapisan pemrosesan, menggunakan beberapa elemen yang beroprasi secara pararel yang diadaptasi dari sistem saraf biologis. Pada CNN setiap neuron direpresentasikan dalam bentuk 2 dimensi, sehingga metode ini cocok untuk pemrosesan dengan input berupa citra (Too et al., 2019). Metode ini sangat efektif dan biasa digunakan pada aplikasi *computer vision*. Arsitektur CNN di di ilustrasikan pada Gambar 2.3.



Gambar 2.3 Arsitektur *convolutional neural network*

Sumber : (Too et al., 2019)

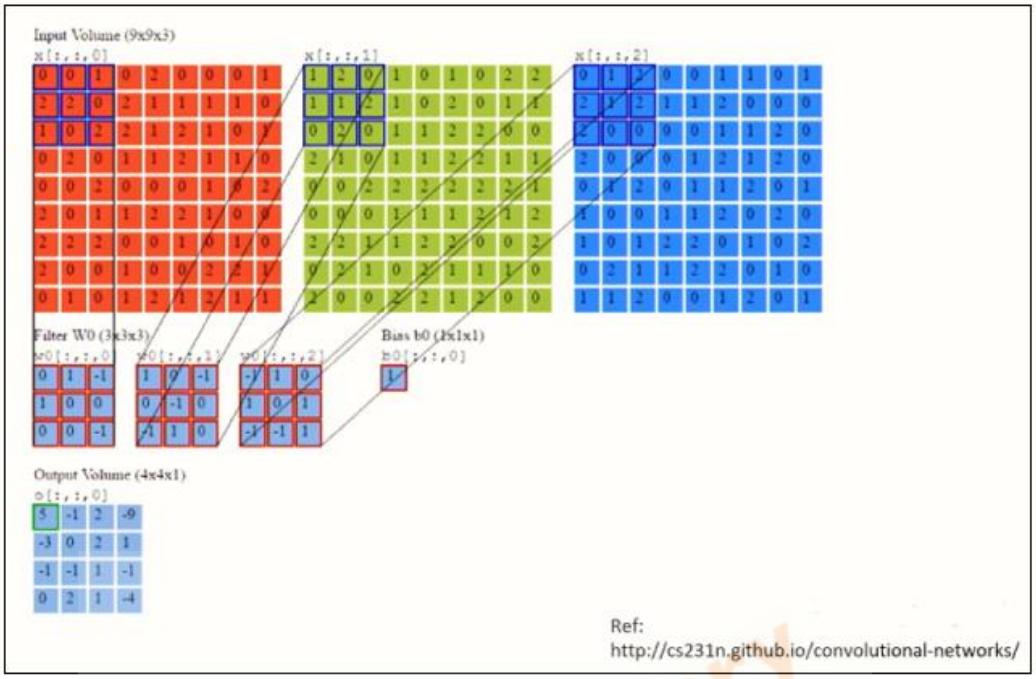
Convolutional neural network dibentuk dengan bantuan berbagai lapisan untuk melakukan klasifikasi gambar. Arsitektur CNN berisi berbagai lapisan seperti:

2.5.1 Lapisan Input

Lapisan input ini menerima gambar mentah dan di teruskan ke lapisan selanjutnya untuk ekstraksi fitur.

2.5.2 *Convolution*

Setelah lapisan input masuk ke proses konvolusi. Dalam lapisan ini diterapkan berbagai filter pada gambar untuk menemukan fitur gambar. Fitur-fitur ini digunakan untuk menghitung kecocokan pada setiap proses pengujian. Cara menghitung proses konvolusi diilustraksikan pada Gambar 2.4 dimana terdapat perkalian piksel gambar dengan filternya untuk mendapatkan output *feature maps*. Pada operasi konvolusi, terdiri dari operasi *padding*, *stride*, dan filter sebagai parameternya. Untuk menghitung ukuran peta fitur atau *feature maps* dilakukan perhitungan menggunakan Persamaan (2.2)



Gambar 2.4 Ilustrasi operasi konvolusi

Sumber : (http://cs231n.github.io/convolutional-networks)

$$Output = \frac{n+2p-f}{s} + 1 \quad \text{.....(2.2)}$$

Keterangan: n = Panjang / Tinggi Input

F = Panjang / Tinggi Kernel Filter

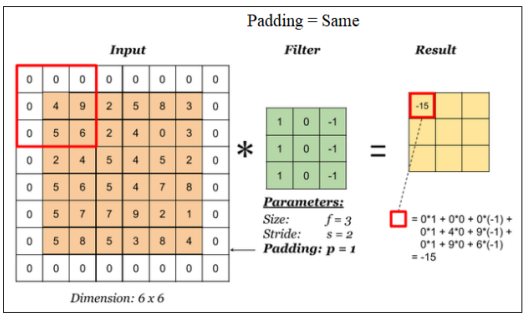
P = Padding

S = *stride*

Sumber : (https://labs.bawi.io/deep-learning-convolutional-neural-networks/)

Padding terdiri dari 2 macam yaitu *padding same* dan *padding valid*.

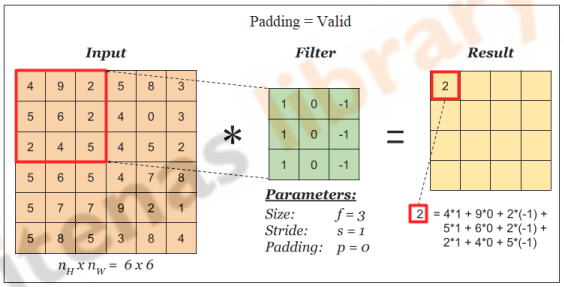
- *Padding same*, disebut juga dengan *zero padding.* Cara kerjanya menambahkan piksel tambahan di sekelilling matriks dengan nilai 0 untuk mengatasi kehilangan piksel citra saat pergeseran kernel pada proses konvolusi. Cara perhitungannya yaitu menggunakan Persamaan(2.2) dimana nilai p = $\frac{f}{2}$, dengan hasil dibulatkan kebawah atau *math.floor()*. Di ilustrasikan seperti Gambar 2.5.



Gambar 2. 5 Ilustrasi *padding same*

Sumber : (https://indoml.com/)

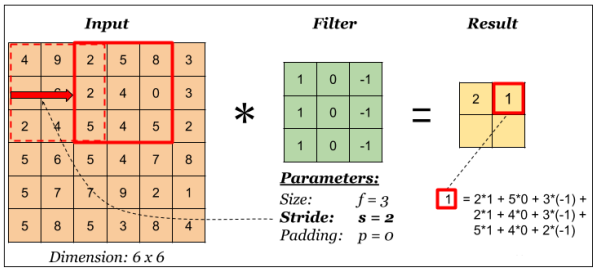
- *Padding valid*, cara kerja *padding valid* yaitu mengambil titik tengah pada operasi konvolusi dalam menentukan nilai matriks di *feature maps*. Cara perthituungannya menggunakan Persamaan(2.2) dimana nilai p = 0**.** Operasi *padding valid* di ilustrasikan pada Gambar 2.6.



Gambar 2. 6 Ilustrasi *padding valid*

Sumber : (https://indoml.com/)

- *Strides*, merupakan parameter yang menentukan jumlah pergeseran filter. Jika nilai *strides* 2, maka filter akan bergeser sebanyak 2 piksel secara horisontal dan vertikal. Semakin kecil pergeseran *stride*, maka fitur yang akan didapat akan semakin detail. *Stride*s ditunjukan pada Gambar 2.7



Gambar 2. 7 Ilustrasi *stride*

Sumber : (https://indoml.com/)

2.5.3 *Batch Normalization*

Batchnormalization digunakan untuk mengurangi pergeseran kovarian atau menyamakan distribusi setiap nilai *input* yang selalau berubah karena perubahan pada *layer* sebelumnya selama proses *training* (Ioffe & Szegedy, 2015). Operasi *batchnormalization* dilakukan sebelum fungsi aktivasi setiap lapisan input. Proses *batch normalization* terdiri dari proses *zero-center input* dengan menghitung *mini batch mean* dan *minibatch varian* lalu dinormalisasi kemudian menghitung *scale & shift*. Adapun penjelasan tiap proses yang dilakukan yaitu sebagai berikut:

1. *Zero-center input*

Proses ini menghitung rata-rata input dan standar deviasi (*input mini bacth*), maka dari itu disebut *batchnormalization* karna menormarlkan data pada *mini batch* untuk mempercepat proses pelatihan dan meningkatkan *learning rates* pada saat pembuatan model (Fadilah, Pardede, & Amelia, 2019). Terdapat empat proses yatu menghiung *mini batch mean* dan *mini batch varian*, normalisasi input, dan *scale and shift*.

Persamaan (2.3) *mini batch mean*

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

Persamaan (2.4) *mini batch varian* :

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(xi - \mu B)^2 \qquad \text{.....(2.4)}$$

Keterangan :

m = jumlah dataset pelatihan

μB = nilai rata-rata dari batch

σB = standar deviasi dari *minibatch*

2. Persamaan (2.5) Normalisasi Input

$$\hat{x}i \leftarrow \frac{xi - \mu B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{.....(2.5)}$$

Keterangan :

$\hat{x}$ = zero-centered dan normalisasi input

$\epsilon =$ *varible* tambahan, biasanya berinlai 10^{-8} untuk menghindari pembagian dengan 0 jika $\sigma = 0$ dalam beberapa estiamsi.

3. Persamaan (2.6) *scale and shift*

$$yi \leftarrow \gamma\hat{x}i + \beta = BN\gamma,\beta(xi) \quad \text{.....(2.6)}$$

Keterangan :

y_i = output scaled dan shifted dari operasi *batchnormalization*

y = *scale* dengan nilai = 1

β = *shift* dengn nilai = *0*

Sumber : (Fadilah et al., 2019)

2.5.4 *Rectified Linear Unit* (ReLu)

Lapisan setelah konvolusi adalah operasi *Rectified Linear Unit.* Lapisan ini menggantikan angka negatif dari lapisan konvolusi dengan 0 (nol), yang membantu untuk *training* yang lebih cepat dan efektif (Nwankpa, Ijomah, Gachagan, & Marshall, 2018).

Persamaan (2.7) ReLu

$$f(xi) = \max(0, xi) \begin{cases} xi, if\ xi \geq 0 \\ 0, if\ xi < 0 \end{cases} \quad \text{.....(2.7)}$$

Keterangan:

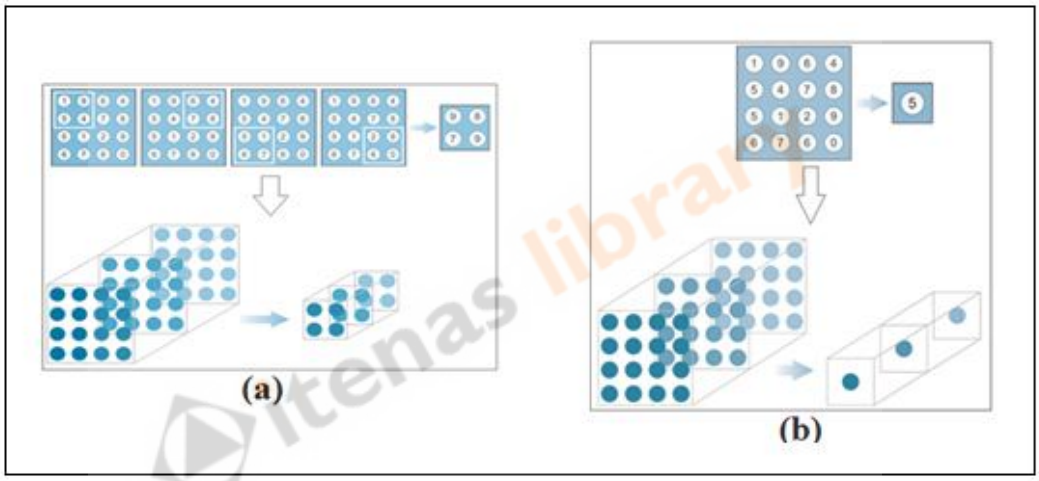
$f(xi)$ = nilai dari ReLU *activation*

Xi = nilai matriks dari citra

Sumber : (Nwankpa et al., 2018)

2.5.5 *Pooling*

Fitur yang diekstraksi dikirim ke lapisan *pooling*. Lapisan ini berfugnsi untuk mereduksinya dimensi gambar yang besar dan mereduksi parameter untuk menyimpan informasi penting. Tujuan nya untuk menjaga nilai maksimum dari setiap lapisan. Lapisan ini tidak memiliki bobot. *Pooling* terbagi menjadi dua yaitu *Maxpooling*, dimana cara kerjanya mengambil nilai terbesar dari hasil konvolusi dan *Global Average Pooling*, mengambil nilai rata rata dari hasil konvolusi. *Maxpooling* dan *Global Average Pooling* diilustrasikan pada Gambar 2.8. Untuk menghitung *feature maps pooling* menggunakan Persamaan (2.8) .



Gambar 2.8 Ilustrasi operasi *pooling*
(a) *Maxpooling* (b) *Global average pooling*

Sumber : (Elgendy, 2019)

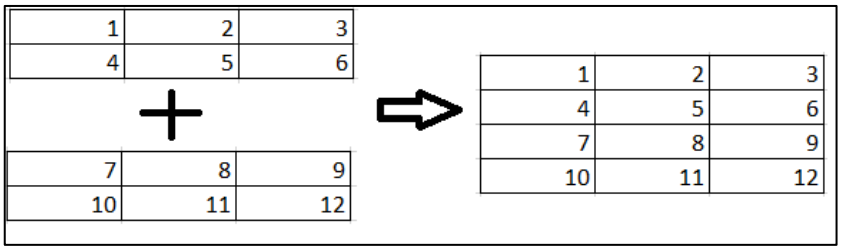
$$Output_{maxpool} = \frac{n-f}{S} + 1 \quad \text{.....(2.8)}$$

Sumber : (Elgendy, 2019)

2.5.6 *Concatenate*

Concatenate adalah proses untuk menggabungkan dua buah matriks menjadi satu buah matriks (Ronneberger, Fischer, & Brox, 2015). Proses ini dilakukan untuk

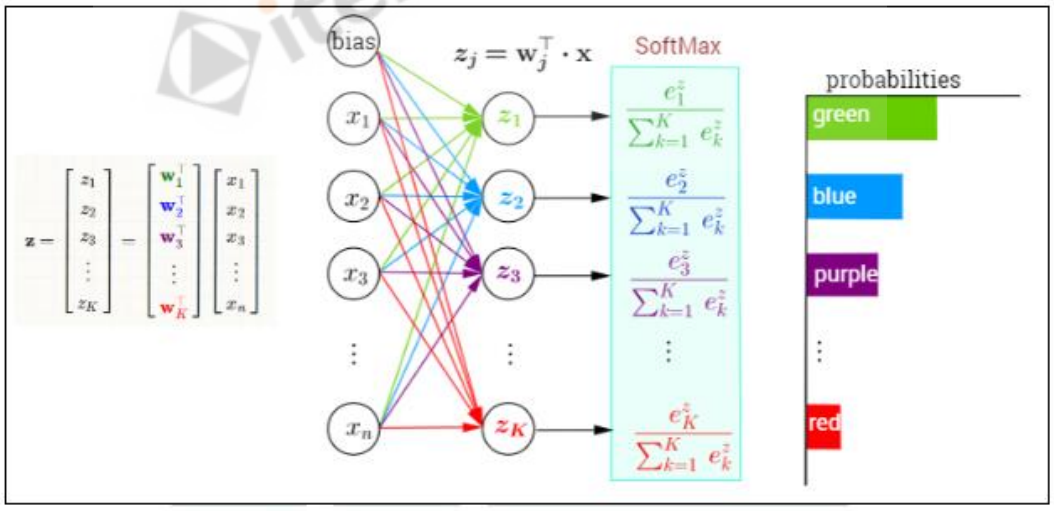
menghasilkan dimensi fitur baru. Ilustrasi concatenate diilustrasikan pada Gambar 2.9



Gambar 2. 9 Ilustrasi *concatenate*

2.5.7 *Fully Connected Layer*

Lapisan terakhir adalah lapisan *fully connected* atau lapisan yang sepenuhnya terhubung, cara kerja nya mengambil gambar yang sudah dilakukan filter tingkat tinggi dan menerjemahkannya ke label dengan kategori. Lapisan ini mendapatkan input dari proses sebelumnya untuk menentukan fitur mana yang paling berkorelasi dengan kelas tertentu (Basha, Dubey, Pulabaigari, & Mukherjee, 2019). Fungsi dari lapisan ini adalah untuk menyatukan semua node menjadi satu dimensi seperti yang diilustrasikan pada Gambar 2.10.



Gambar 2.10 Ilustrasi *fully connected layer*

Sumber : (Wulandari et al., 2019)

Berdasarkan Gambar 2.10, terdapat perhitungan di dialamnya yang di definisikan sebegai berikut:

$$z_j = \sum_{j=1}^{c} w_{i,j}^{T} x_i + b_j \quad \text{.....(2.9)}$$

Keterangan :

z_j = nilai keluaran dari jaringan

x_i= nilai masukan hasil ekstraksi fitur

$w_{i,j}$= bobot dari jaringan berukuran *i x j*

i = jumlah fitur masukan

j = jumlah target kelas

b_j = bias dari jaringan

Sumber : (Wulandari et al., 2019)

2.5.8 *Softmax*

Softmax digunakan untuk mendapatkan hasil klasifikasi. Nilai *softmax* berada pada interval [0 – 1] dan memiliki jumlah 1 jika seluruh elemennya dijumlahkan. Fungsi ini biasanya digunakan diujung lapisan dari *fully connected* untuk klasifikasi lebih dari dua kelas yang digunakan pada CNN untuk mendapat nilai probabilitas suatu objek terhadap kelas yang ada (Wulandari et al., 2019). Persamaan (2.10) *softmax* dan Persamaan (2.11) turunan *softmax*

$$S_j = \frac{e^{z_j}}{\sum_{d=1}^{C} e^{z_d}} \quad \text{.....(2.10)}$$

$$\frac{\partial S_j}{\partial z_i} = \begin{cases} \frac{e^{z_i}\sum_C - e^{z_i}e^{z_i}}{(\sum_C)^2} = \frac{e^{z_i}}{\sum_C}\frac{\sum_C - e^{z_i}}{\sum_C} = s_i(1 - s_i), & \text{jika } j = i \\ \frac{-e^{z_i}e^{z_i}}{(\sum_C)^2} = -\frac{e^{z_i}}{\sum_C}\frac{e^{z_j}}{\sum_C} = -s_i s_j, & \text{jika } j \neq i \end{cases} \quad \text{.....(2.11)}$$

Keterangan :

z_j = nilai keluaran dari jaringan

S_j = probabilitas dari $z_j$untuk setiap C kelas yang berbeda

$\sum_C = \sum_{d=1}^{C} e^{z_d}$

e = eksponen, dimana memiliki ketetapan yaitu 2.7183

i = jumlah fitur masukan

j = jumlah target kelas

Sumber : (Wulandari et al., 2019)

2.5.9 *Cross Entropy Loss Function*

Loss function merupakan fungsi yang menggambarkan kerugian terkait dengan semua kemungkinan yang dihasilkan oleh model. Ketika suatu model memiliki kelas yang banyak, perlu adanya cara untuk mengukur perbedaan antara probabilitas hasil hipotesis dan probabilitas kebenaran yang asli, dan selama pelatihan, banyak algoritma yang dapat menyesuaikan parameter sehingga perbedaan ini diminimalkan (Wulandari et al., 2019). Cara kerja *cross entropy* adalah meminimalkan log negatif dari dataset.

Persamaan (2.12) *cross entropy loss function*

$$\mathrm{D}\,(s,p) = -\sum_{j} p_j \log s_j \quad \text{.....(2.12)}$$

Keterangan:

s_j= Nilai hasil prediksi

p_j = nilai sesungguhnya

j = jumlah target kelas

Sumber : (Wulandari et al., 2019)

2.5.10 *Stochastic Gradient Descent*

Stochastic Gradient Descent adalah algoritma yang digunakan secara umum pada *machine leanring*, *neural network*, dan *deep learning* untuk melakukan optimasi. Algoritma ini sering disebut SGD, cara kerjanya yaitu melakukan optimasi pada parameter bobot atau *weight* dan bias dengan cara mengurangi nilai

weight awal dengan *gradient* yang di dapat selama proses *backward pass* atau propagasi mundur, tujuannya untuk meminimalkan nilai *loss function* (Ruder, 2017). Perhitungan SGD dihitung menggunakan persamaan (2.13).

$$w_{i,j} = w_{i,j} + \Delta w_{i,j}, \text{dimana } \Delta w_{i,j} = -\eta \frac{\partial D}{\partial w_{i,j}} \quad \text{.....(2.13)}$$

Keterangan :

$w_{i,j}$= bobot dari jaringan berukuran *i x j*

η= tingkat pembelajaran atau *learning rate*

i = jumlah fitur masukan

j = jumlah target kelas

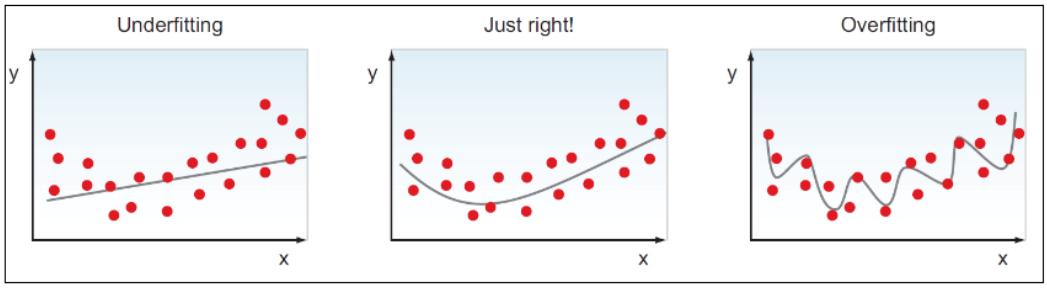
Sumber : (Wulandari et al., 2019)

2.6 *Overfittng, Underfitting, and Regularization*

Pada pembuatan model *mahcine learning* dan *deep learning* hal ini merupakan konsep yang penting, dimana suatu model harus mampu memprediksi data yang belum pernah dilihat sebelumnya dengan tepat atau disebut juga generalisasi (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014).

- *Undefitting* adalah keadaan ketika model memiliki kinerja buruk baik pada data pelatihan dan data uji atau *unseen examples*.
- *Overfitting* adalah keadaan ketika model memiliki kinerja baik hanya untuk data pelatihan atau *seen examples* tetapi tidak memiliki kinerja yang baik untuk data uji atau *unseen examples*. Untuk mengatasi *overfitting* salah satunya dengan menggunakan teknik *regularization*.

Overfitting dan *undefitting* diilutstrasikan pada Gambar 2.11.



Gambar 2.11 *Underfitting* dan *overfitting*

Sumber : (Elgendy, 2019)

2.6.1 *Reguralization*

Regularization adalah teknik yang bertujuan untuk mengurangi kompleksitas jaringan. Cara umum teknik regularization terbagi menjadi tiga yaitu *L2 Regularization*, *Dropout*, dan Data augmentasi.

2.6.2 *L2 Regularization*

L2 regularization adalah teknik regularisasi untuk menyederhanakan model dengan melakukan pengurangan nilai bobot pada *hidden layer* yang membuat nilai bobot kecil mendekati 0, tapi tidak membuat bobot sama dengan 0 (Elgendy, 2019). Operasi ini dihitung menggunakan Persamaan 2.14.

$$eror\ function_{new} = eror\ function_{old} + regularization\ term$$

$$L2\ regularization\ term = \frac{\lambda}{2m} * \sum ||w||^2 \quad \text{.....(2.14)}$$

Keterangan :

λ : regularization parameter

m : number of instance

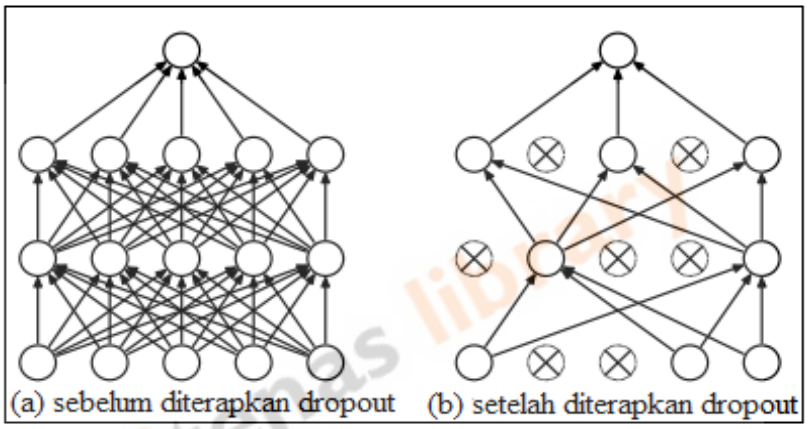
w : weight

Sumber : (Elgendy, 2019)

2.6.3 *Dropout*

Dropout merupakan teknik *regularization* yang sangat efektif untuk menyederhanakan kompleksitas jaringan saraf untuk menghindari *overfitting*. Cara kerja *dropout* yaitu memutuskan beberapa neuron penghubung, oleh karena itu, neuron sebelumnya harus mencari neuron lain untuk dapat melanjutkan ke lapisan

terakhir, neuron yang akan dihilangkan akan dipilih secara acak oleh sistem dan bobotnya tidak diperbarui dalam proses pelatihan (Srivastava et al., 2014). Ini bertujuan untuk mengurangi pembelajaran yang saling bergantung di setiap neuron. Dalam jurnal asli yang mengusulkan lapisan *dropout* oleh (Srivastava et al., 2014), *dropout* di implementasikan pada masing-masing lapisan *fully connected* yang merupakan konfigurasi paling umum. Namun penelitian terbaru, lapisan *dropout* bisa di implementasikan pada lapisan konvolusi setelah operasi ReLu (Liu, Tian, & B, 2017). *Dropout layer* diilustrasikan pada Gambar 2.12.



Gambar 2. 12 Ilustrasi *dropout*

Sumber : (Srivastava et al., 2014)

2.6.4 Data Augmentasi

Data augmentasi merupakan salah satu teknik mencegah *overfitting* dengan menghasilkan lebih banyak data pelatihan. Tujuannya agar model tidak akan melihat gambar yang sama persis dua kali (Chollet, 2018). Operasi ini dilakukan pada data pelatihan saja dan tidak dilakukan untuk data validasi. Biasanya terdiri dari operasi transformasi diantaranya:

- *rotation_range :* menerapkan rotasi gambar dengan skala 0-180 derajat.
- *Width_shift and height_shift* : menerapkan pergeseran gambar secara *horizontal* dan *vertical*.
- *Shear_range* : menerapkan transformasi geser secara acak
- *Zoom_range* :menerapkan transformasi memperbesar gambar
- *Horizontal_flip* : membalik secara horizontal

Augmentasi data diilustrasikan pada Gambar 2.13.

Gambar 2.13 Ilustrasi Augmentasi Data

Sumber : (Chollet, 2018)

2.7 *Transfer Learning*

Transfer learning adalah metode menggunakan jaringan saraf yang sudah dilatih sebelumnya lalu mengurangi jumlah parameter dengan cara mengambil beberapa bagian dari model yang sudah dilatih untuk digunakan dalam mengenali model baru (Abas, Ismail, Yassin, & Taib, 2018). Didasari oleh fakta bahwa manusia dapat menerapkan pengetahuan yang dipelajari sebelumnya untuk memecahkan masalah baru dengan lebih cepat dan dengan solusi yang lebih baik.

Jaringan saraf sangat bergantung pada jumlah data untuk mencapai kinerja yang tinggi. Berikut adalah alasan mengapa *transfer learning* digunakan :

1. Masalah data, *deep learning* membutuhkan banyak data untuk bisa mendapatkan hasil yang baik. Membutuhkan banyak waktu untuk mendapatkan data berlabel jika dilakukan oleh manusia dalam mengambil gambar dan memberi label satu-per-satu.
2. Masalah komputasi, bahkan jika sudah mempunyai puluhan ribu data gambar untuk menyelesaikan masalah yang dimiliki, secara komputasi untuk melatih jaringan saraf yang dalam menggunakan puluhan ribu gambar tersebut akan sangat mahal membutuhkan waktu berhari-hari menggunakan GPU dan perlu dilakukan proses berulang untuk mendapatkan hasil yang memuaskan.

2.7.1 Pendekatan *Transfer Learning*

Terdapat tiga pendekatan utama *transfer learning* menurut (Elgendy, 2019) sebagai berikut:

1. *Pretrained as a classifier,* pada pendekatan ini domain sumber dengan domain target sangat mirip. *pre-trained model* digunakan langsung untuk mengklasifikasi target. Pada pendekatan ini, *pre-trained* model hanya digunakan untuk memprediksi gambar tanpa ada pelatihan tambahan.
2. *Pretrained as a feature extractor*, pada pendekatan ini data domain sumber dengan domain target mirip. Model dilatih menggunakan dataset besar *ImageNet* kemudian bobot dan arsitekturnya di latih ulang dengan cara membekukan bagian esktraksi fitur, menghapus bagian lapisan klasifikasi / *classification layer* dan menambahkan lapisan klasifikasi bari untuk gambar target.
3. *Fine-tuning*, pada pendekatan data domain sumber dan domain target sangat berbeda. Diperlukan ekstraksi peta fitur yang tepat dari domain sumber lalu menyempurnakannya agar sesuai dengan domain target.

2.7.2 Memilih Pendekatan *Transfer Learning*

Pada dasarnya lapisan konvolusional pada lapisan bawah/awal merupakan proses ekstraksi fitur-fitur umum (Elgendy, 2019). Semakin dalam jaringan, maka fitur ekstraksi akan semakin detail. Dua faktor penting yang harus diperhatikan dalam penerapan *transfer learning* diantaranya:

1. Ukuran dataset target (sedikit atau banyak).
2. Kesamaan domain dataset antara sumber dan target.

Dari faktor yang dijelaskan pada poin 1 dan poin 2, disimpulkan pada Tabel 2.1 dan diilusrasikan pada Gambar 2.14 untuk aturan umum pendekatan *transfer learning*.

Tabel 2.1
Memilih Pendekatan *Transfer Learning*

Skenario	Ukuran data targat	Perbedaan domain antara dataset asli dan baru	Pendekatan
1.	Kecil	Serupa	*Pre-trained as a feature extractor*
2.	Besar	Serupa	*Fine tune through the full network*
3.	Kecil	Sangat berbeda	*Fine tune from activations earlier in the network*
4.	Besar	Sangat berbeda	*Fine tune through the entire network*

Sumber : (Elgendy, 2019)

Keterangan:

1. Skenario 1: Dataset target kecil dan sangat mirip dengan dataset sumber

   Karena dataset sangat mirip dengan dataset target, pada kasus ini baiknya menerapkan *transfer learning* dengan hanya melatih lapisan klasifikasi atau *classifier* saja. Kurang baik jika melakukan pelatihan pada lapisan ekstraksi fitur, karena dataset yang sedikit tidak mampu memberikan informasi fitur yang cukup untuk menggeneralisasi data, akibatnya akan *overvitting*.

2. Skenario 2: Dataset target besar dan mirip dengan dataset sumber

   Seperti pada skenario poin ke-1, namun, karena memiliki dataset yang banyak, latih ulang lapisan fitur yang merepresentasikan fitur spesifik, yaitu pada lapisan atas atau *top-layer*. Bekukan sekitar 60-80% dari jaringan *pre-trained* dan lapisan sisanya gunakan untuk melatih dataset baru.

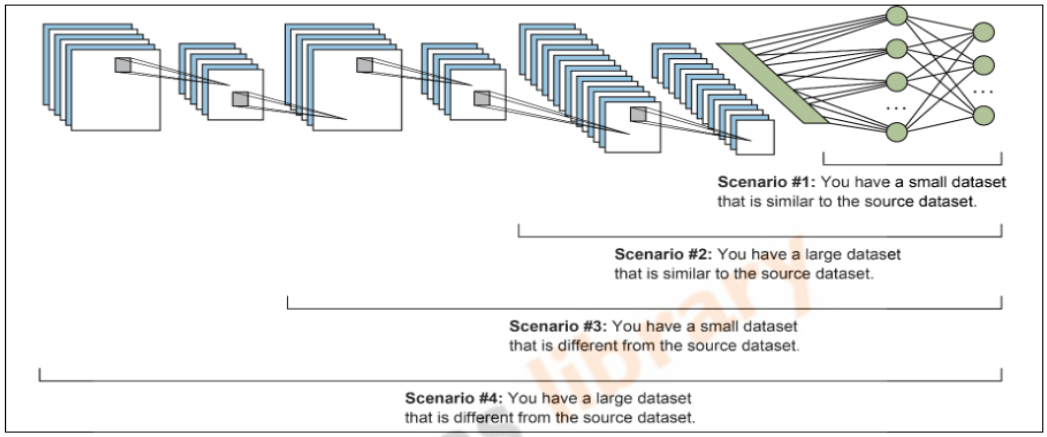
3. Skenario 3: Dataset target kecil dan berbeda dari dataset sumber

   Solusi pada masalah ini yaitu membekukan sepertiga pertama atau setengah dari jaringan *pre-trained*. Tidak baik untuk membekukan fitur tingkat tinggi dari jaringan *pre-trained*. Lapisan awal merepresentasikan peta fitur yang sangat umum dan berguna untuk dataset target yang kecil.

4. Skenario 4: Dataset target besar dan berbeda dari dataset sumber

Solusi skenario ini yaitu melatih seluruh jaringan dari awal dan tidak menggunakan *transfer learning*. Namun, keunggulan *transfer learning* bila digunakan berguna untuk membuat model lebih cepat konvergen.

Adapun kesimpulan dari ke-empat skenario di atas diilustrasikan seperti Gambar 2.14.



Gambar 2. 14 Tingkat Penyetelan *Transfer Learning* berdasarkan empat skenario

2.8 *ImageNet*

ImageNet adalah kumpulan data yang berisi lebih dari 15 juta gambar beresolusi tinggi berlabel yang terdiri dari 22.000 kategori. Gambar ini dikumpulkan dari web dan diberi label menggunakan alat *amazon mechanical turk crowd-sourcing tool* oleh *labeler* manusia (Krizhevsky & Hinton, 2012). Dalam *computer vision*, dataset ini digunakan sebagai *subset* pada kompetisi tahunan *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) yang merupakan kompetisi penting dalam pengenalan dan klasifikasi gambar. Kompetisi ini merupakan bagian dari *Pascal Visual Object Challange* yang dilaksanakan pada tahun 2010. Secara keseluruhan ILSVRC menggunakan *subset ImageNet* sebanyak 1,4 juta dan 1000 kelas objek. Hasil kompetisi ini, biasanya melaporkan hasil dengan dua tingkat kesalahan, yaitu *top-1* dan *top-5*. Pada Tabel 2.2 dijelaskan model arsitektur pemenang ILSVRC dan jumlah parameternya, sampai saat ini dijadikan sebagai *pre-trained* model.

Tabel 2.2
Eror rates Arsitektur Pemenang Kompetisi ILSVRC

Arsitektur CNN	*Top-5 error rates (%)*	Tahun	Oleh	Jumlah parameter
LeNet	28.2	1998	YannLeCun et al.	600.000
AlexNet	16.4	2012	Alex K et al.	62.3 juta
VGG	7.30	2014	Simonyan, Zisserman	138 juta
GoogleNet	6.70	2014	Google Inc	4 juta
ResNet	3.57	2015	Kaiming He	25 juta

Sumber : (Krishna & Kalluri, 2019)

ImageNet banyak digunakan peneliti sebagai *subset pre-training* untuk mengadaptasi fitur-fiturnya ke tugas baru dan sudah menjadi standar *de facto* untuk menyelesaikan berbagai variasi masalah pada *computer vision* (Efros, 2016). Diperoleh hasil yang baik menggunakan *transfer learning* pada penelitian klasifikasi gambar spesifik (Suh, IJsselmuiden, Hofstee, & van Henten, 2018) mendapatkan akurasi 98,7% dengan dataset hanya sebanyak 1100 menggunakan arsitektur AlexNet pada bidang agrikultur untuk klasifikasi *sugar beets* dan *volunter potato*. Lalu penelitian dengan gambar spesifik lainnya dibidang medis (Shu, 2019) mendapatkan akruasi sebesar 94,82% menggunakan arsitektur VGG19 pada klasifikasi tumor otak menggunakan dataset berjumlah 3064.

Disisi lain, dataset *imagenet* berisi sejumlah gambar hampir untuk semua kelas termasuk yang langka sekalipun. Dataset ini menjadi patokan untuk dalam *computer vision* seperti Celtech101 / Celtech256 dan PASCAL yang berperan penting dalam pengenalan objek dan penelitian tentang penegenalan. Pembuatan *ImageNet* mengikuti hirarki *wordnet* dimana setiap kata / frasa yang berarti di dalam *wordnet* disebut sebagai "set sinonim" atau "*synset"*. Dijelaskan pada Tabel 2.3 objek gambar yang terdapat pada *ImageNet*.

Tabel 2. 3
Statistik Data Tingkat Tinggi Pada Kategori *ImageNet*

High level category	*# synset (subcategories)*	*Avg # images per synset*	*Total # images*
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

Sumber: (www.image-net.org)

2.9 Arsitektur *Visual Geometry Group (VGG Network)*

VGGNet dikembangkan oleh *Visual Geometry Group* di Oxford *University*. Model ini menjuarai kontes ILSVRC sebagai *1st Runner-up Image Classification*

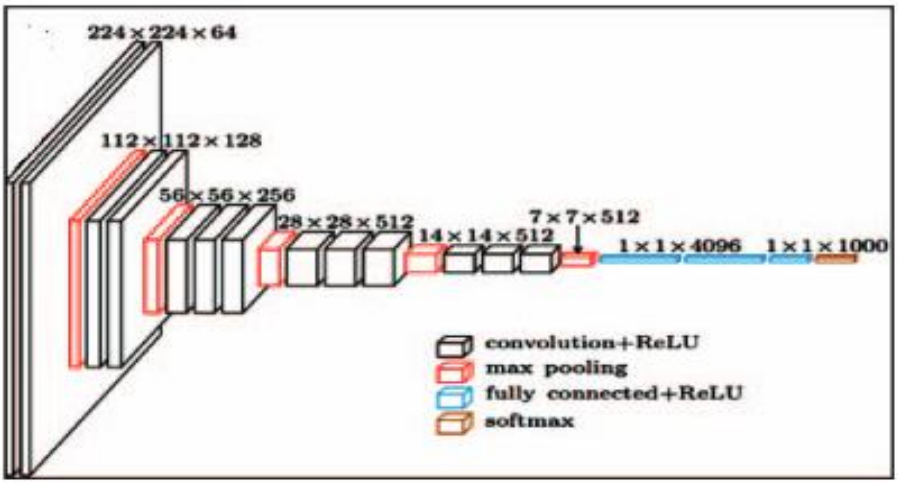
dan pemenang Image *Localization* pada tahun 2014 . Model VGGNet dirancang mengurangi kernel ukuran besar pada AlexNet 1x1 dan 5x5 diganti dengan beberapa kernel 3x3 dengan 1 *stride* yang berguna untuk mengekstrak fitur kompleks (Simonyan & Zisserman, 2015). Kemudian menggunakan *max pooling* 3x3 dengan 2 *strides*. Model ini sudah terlatih pada dataset besar gambar ImageNet dengan 1000 kelas yang berbeda Konfigurasi arsitektur VGG dijelaskan pada Tabel 2.4.

Tabel 2.4
Konfigurasi Arsitektur VGGNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Sumber : (Simonyan & Zisserman, 2015)

Pada penelitian ini digunakan arsitektur VGG16 yang di dalamnya terdapat proses 5 blok konvolusi yang terdiri dari operasi konvolsi 3x3 menggunakan 1 *stride* dengan jenis *padding same / zero padding* lalu di aktivasi menggunakan relu. Kemudian setiap selesai opearsi blok konvolusional di reduksi menggunakan operasi maxpooling 2x2 dengan 2 *stride* dan diakhiri dengan 2 dense layer sebanyak 4096 node/neuron. Model VGG16 diilustrasikan pada Gambar 2.15.



Gambar 2. 15 Ilustrasi Arsitektur VGG16

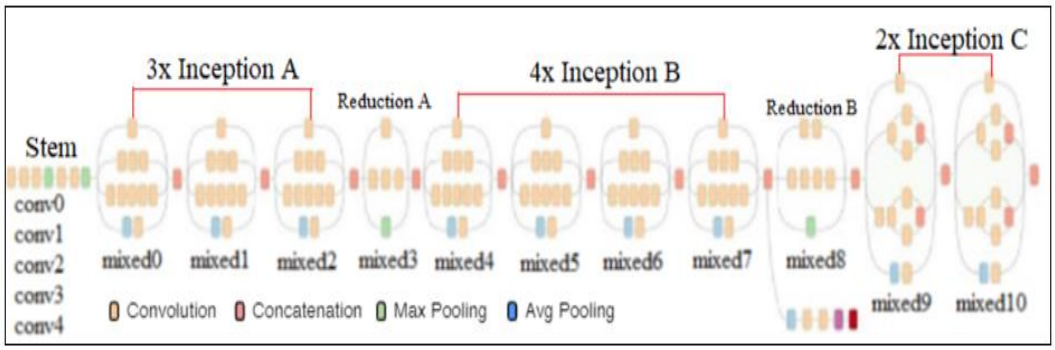
Sumber : (Kaur & Gandhi, 2019)

2.10 Arsitektur Google *Network* (Inceptionv3)

Model Inception-v3 merupakan pengembangan model GoogleNet atau Inception-v1 yang dikembangkan pada penelitian (Szegedy, Liu, et al., 2015a) yang menjuarai kontes *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) pada kategori *image classification* di tahun 2014. Kemudian disempurnakan dengan menambahkan normalisasi batch (BN) yang dinamakan Inception-v2 (Szegedy, Liu, et al., 2015b). Kemudian dikembangkan kembali menjadi InceptionV3 dengan pengembangan faktorisasi tambahan pada operasi konvolusi, menjadi *1st runner-up image classification* pada kontes ImageNet Large Scale Visual Recognition Challenge (ILSVRC) di tahun 2015 dalam mengenali 1000 objek kelas ImageNet.

Inception-v3 terdiri dari 5 lapisan konvolusional dasar (stem) dengan tipe *valid padding* yang terdiri dari conv2d_0 hingga conv2d_4 dimana setiap operasi konvolusi diikuti oleh aktivasi ReLu dan BatchNormalization. Kemudian diikuti oleh 11 modul inception yang terdiri dari modul mixed0 hingga modul mixed10 dengan tipe same padding disetiap operasi konvolusi nya. 11 blok modul Inceptionv3 dirancang dengan faktorisasi konvolusi 1x1, 3x3, 1x3, 3x1, 5x5, 1x7, dan 7x7. (Szegedy, Vanhoucke, et al., 2015). Ketiga, bagian *classification* di Model ini sudah terlatih pada dataset besar gambar ImageNet dengan 1000 kelas yang

berbeda. Model Inception-v3 diilustrasikan pada Gambar 2.16 dan konfigurasi model ditujukan pada Tabel 2.5**.**

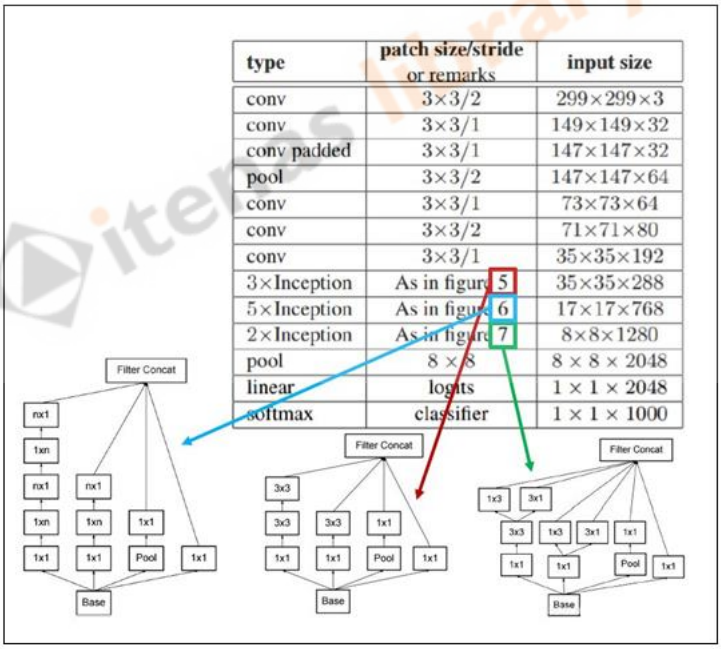


Gambar 2. 16 Ilustrasi Arsitektur Inceptionv3

Sumber: (Habibzadeh Motlagh, Jannesari, Rezaei, Totonchi, & Baharvand, 2018)

Tabel 2..5
Konfigurasi Arsitektur Inceptionv3

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000



Sumber telah diolah kembali : (Szegedy, Vanhoucke, et al., 2015)

2.11 Pelatihan *Neural Network*

Untuk membangun dan menggunakan *neural network* terdirid dari 3 tahap yaitu pelatihan atau *training* dan evaluasi atau *evalutation*. (Elgendy, 2019). Pada tahap pelatihan, bobot dan bias tiap node atau neuron akan di *update* secara tarus menerus

sesuai dengan parameter yang diatur. Pelatihan terdiri dari 2 (dua) tahap yaitu *forward pass* dan *backwar pass*.

2.11.1 *Forward Pass*

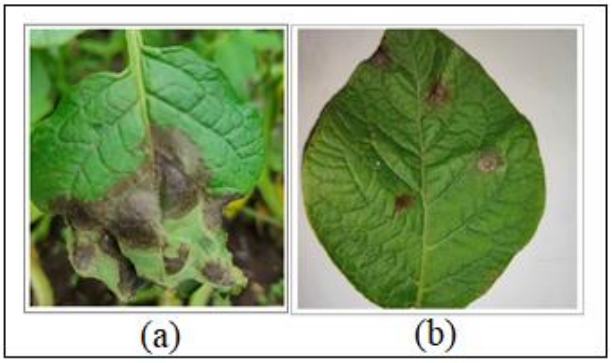
Forward pass atau sering disebut propagasi maju yaitu proses yang akan membawa data mengalir pada neuron. Tahapan ini data akan dibawa dari mulai input melewati tiap neuron yang ada pada lapisan tersembunyi (*hidden layer*) seperti pada lapisan konvolusi di sub-bab 2.5.2 pada Gambar 2.3 sampai kepada lapisan keluaran (*output layer*). Setelah mencapai lapisan keluaran, akan dihitung *cost* atau error yang terjadi selama proses dilakukan (Elgendy, 2019).

2.11.2 *Backward Pass*

Pada proses *backward pass* atau sering disebut poropagasi mundur yaitu proses dimana kesalahan yang didapat pada proses sebelumnya yakni *forward pass* akan digunakan untuk memperbarui setiap bobot dan bias dengan *learning rate* tertentu. Kedua proses baik *forward pass* dan *backward pass* akan dilakukan secara berulang sampai didapatkan nilai bobot dan bias yang dapat memberikan nilai kesalahan sekecil mungkin pada lapisan keluaran (Elgendy, 2019).

2.12 Penyakit Daun Kentang

Penyakit daun kentang pada penelitian ini dibagi menjadi dua kelas, diantaranya busuk daun fitoftotra (*phitopthora late blight)* dan bercak kering (*early blight)* yang timbul pada saat musim kemarau. Gejala dalam kasus *late blight* adalah munculnya bercak gelap seperti Gambar 2.17(a) biasanya timbul pada saat musim hujan, pada ujung daun dan batang tanaman. Jamur putih muncul dibawah daun dalam kondisi lembab dan pada akhirnya seluruh tanaman dapat dengan cepat mati dan menyebabkan gagal panen. Sedangkan gejala dalam kasus *early bilght* pada tahap awal bintik cokelat gelap (seperti cincin) muncul dibagian tengah daun seperti Gambar 2.17(b), jika dibiarkan daun akan berubah menjadi kuning dan mengering yang akan menyebabkan kerusakan defoliasi prematur tanaman kentang kemudian menyerang bauah. Menurut (Rakhmawati et al., 2018).



Gambar 2.17 Daun kentang yang terkena penyakit
(a) Busuk daun fitoftora, (b) Bercak kering

Sumber: Balai Penelitian Tanaman dan Sayuran

2.13 Tinjauan Pustaka

Dalam penelitian ini terdapat beberapa pustaka yang berkaitan dengan kegiatan penelitian yang dilakukan, antara lain:

(Efros, 2016) melakukan penelitian yang berjudul "*What makes ImageNet Good For Transfer Larning?*". Penelitian dilakukan menggunakan metode *transfer learning* dengan model AlexNet. Tujuan dari penelitian ini adalah mensurvey skenario *transfer learning* dengan berbagai dataset terkenal seperti SUN, PASCAL, dan ImageNet. Hasil penelitian disimpulkan bahwa model *pre-trained* yang sudah dilatih pada dataset ImageNet menjadi standar *de-facto* untuk metode *transfer learning* dalam hal klasifikasi gambar karena dataset ImageNet merupakan data yang diguanakan dalam melahirkan algoritma baru yang canggih yang mampu mengatasi masalah pengenalan objek. Kontribusi penelitian ini menjadi dasar penggunaan ImageNet sebagai *pre-trained weight* untuk penelitian yang akan dilakukan. Pembeda dengan penelitian yang dilakukan adalah arsitektur yang digunakan, pada penelitian ini menggunakan arsitektur VGG16 dan Inceptionv3.

(Mohanty, Hughes, & Salathé, 2016), melakukan penelitian yang berjudul "*Using Deep Learning For Image-Based Plant Disease Detection*". Penelitian dilakukan mengguanakan metode *Deep Learning* dengan model AlexNet dan GoogleNet yang di *training* dengan sekenario *from scarth* dan *transfer learning*. Data yang latih digunakan adalah Plantvillage dengan jumlah 54.306 dengan 14

kelas. Model diuji dengan *unseen* 121 data dari Bing Image Search dan 119 data dari IPM. Hasil penelitian didapat akurasi *testing* terbaik menggunakan *transfer learning* GoogleNet dengan akurasi *testing* mencapai 99.35%. Kontribusi penelitian ini adalah menguji reliabilitas dataset PlantVillage yang diuji terhadap data gambar *unseen* yang *real-life*. Pembeda dengan penelitian yang dilakukan adalah penelitian yang dilakukan menggunakan metode *transfer learning* VGG16 dan InceptionV3.

(Rakhmawati et al., 2018) melakukan penelitian dengan judul "Klasifikasi Penyakit Daun Kentang Berdasarkan Fitur Tekstur Dan Fitur Warna Menggunakan Support Vector Machine". Penelitian dilakukan menggunakan metode GLCM dan Color moment sebagai ekstrator fitur tekstur dan warna kemudian menggunaakn SVM sebagai *classifier*. Data yang digunakan berjumlah 300 citra pelatihan dan 90 citra uji yang terdiri dari 3 kelas daun kentang dari Badan Pengkajian Teknologi Pertanian. Tujuan penelitian adalah mengklasifikasi penyakit daun kentang. Hasil penelitian mendapatkan akurasi mencapaui 80%. Selain itu juga, pada penelitian disimpulkan bahwa pola dari daun berpenyakit sangat mempengaruhi identifikasi. Oleh karena, itu pemilihan daun non-penyakit atau sehat harus lebih selektif yang tanpa bercak sama sekali. Kontirbusi penelitian ini adalah menggunakan metode SVM dalam mengklasifikasi penyakit daun kentang. Pembeda dengan penelitian yang dilakukan adalah penelitian yang dilakukan menggunakan metode *transfer learning* dengan model VGG16 dan InceptionV3.

(Goncharov, Ososkov, Nechaevskiy, Uzhinskiy, & Nestsiarenia, 2019) melakukan penelitian dengan judul "*Disease Detection on the Plant Leaves by Deep Learning*". Penelitian dilakukan menggunakan metode *Siamese Network* untuk klasifikasi penyakit daun anggur dengan 4 kelas. Data latih yang digunakan adalah data daun anggur PlantVillage yang diuji oleh gambar dari internet yang berjumlah 71 data. Hasil penelitian mendapat akurasi *testing* sebesar 90%. Kontribusi penelitian ini adalah mengembangkan *deep neural network* yang bernama *Siamase Network* atau jaringan kembar siam (*twin*) untuk memecahkan masalah klasifikasi penyakit daun anggur. Pembeda dari penelitian yang dilakukan adalah penelitian

yang dilakukan menggunakan metode *transfer learning* VGG16 dan InceptionV3 dengan dataset penyakit daun kentang.

(Bernico, Li, & Zhang, 2019), melakukan penelitian dengan judul "*Investigating the Impact of Data Volume and Domain Similarity on Transfer Learning Applications*". Tujuan peneltian adalah membahas skenario pengaruh kemiripan domain sumber, domain target, dan volume dataset. Kontribusi penelitian ini memberikan skenario untuk *transfer learning* pada model VGG16, VGG-Face, dan InceptionV3 berdasarkan situasi volume data serta dari sisi tingkat kesamaan domain target dan domain sumber. Hasil penelitian disimpulkan jika dataset besar (banyak) dan domain sama, lakukan *feature as a calssifier* atau mengganti lapisan *classifier* nya saja. Namun data sedikit dan domain berbeda, lakukan *fine-tuning*. Pembeda penelitian yang dilakukan adalah penelitian menggunakan pendekatan *fine-tuning* untuk dataset penyakit daun kentang.

(Too et al., 2019) melakukan penelitian dengan judul "*A comparative Study Fine-Tuning Deep Learning Models for Plant Disease Identification*". Penelitian membahas teknik *fine-tuning* pada model VGG16, InceptionV4, ResNet-50, ResNet-101, ResNet-152, DenseNet-121, dan DenseNet untuk klasifikasi dataset *PlantVillage*. Hasil penelitian didapatkan InceptionV4 98.08%, VGG16 81.83%, ResNet-50 99.59%, ResNet-101 99.66%, ResNet-152 99.59%, dan DensNet 99.75%. DenseNet-121 mendapat akurasi testing terbaik 99.75% dengan 30 epoch. Kontribusi penelitian ini mempelajari teknik *fine-tuning* yang diimplementasikan pada berbagai *pret-rained* model. Pembeda penelitian yang dilakukan adalah penelitian yang dilakukan hanya membandingkan Inceptionv3 dan VGG16 dengan menggunakan dataset *PlantVillage* penyakit daun kentang saja.

(Basha et al., 2019) melakukan penelitian dengan judul "*Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification*". Tujuan penelitian ini menganalisa pengaruh lapisan FC untuk klasifikasi gambar. Hasil penelitian disimpulkan bahwa pemilihan lapisan FC yang tepat bukan hanya memberikan kinerja yang baik pada model, tapi juga mengurangi waktu yang diperlukan untuk melatih model. Didapatkan pada hasil penelitian CNN

yang dangkal memerlukan lebih banyak node dalam lapisan FC. Disisi lain, CNN yang lebih dalam memerlukan sedikit jumlah neuron di lapisan FC. Kontribusi penelitian ini adalah mempelajari pengaruh Size of FC pada lapisan *classifier* CNN. Pembeda penelitian yang dilakukan adalah penelitian yang dilakukan menggunakan 3 FC layer dengan parameter 4096 neuron dan 2048 neuron dengan *dropout rate none* sampai 0.5.

(Arya & Singh, 2019) melakukan penelitian dengan judul "*A Comparative Study of CNN and AlexNet for Detection of Disease in Potato and Mango Leaf*". Tujuan penelitian yaitu membandungkan CNN dan arsitektur AlexNet dengan *Transfer learning* Kemudian penelitian lain terkait penyakit tanaman dibahas oleh Arya. Dalam penelitian ini menggunakan 2 arsitektur yaitu CNN dengan 3 *layer feature learning* dan 2 *layer classification* serta model AlexNet dengan t*ransfer learning*. Dataset yang digunakan pada penelitian ini dari website PlantVillage dan diambil secara real-time dari perkebunan GBPUAT. Total data berjumlah 4004 yang terdiri dari 4 kelas, 2 kelas tanaman mangga dan 2 kelas tanaman kentang. Hasil penelitian mendapatkan akurasi 90.85% untuk CNN dan 98.33% untuk AlexNet. Kontribusi penelitian ini adalah klasifikasi penyakit pada daun mangga dan daun kentang serta membandingkan kinerja pada kedua model menggunakan metode CNN dan *transfer learning* AlexNet. Pembeda dengan penelitian yang dilakukan adalah penelitian menggunakan model VGG16 dan InceptionV3.

(Gangwar et al., 2020) melakukan penelitian yang berjudul "*Grape Leaf Diseases Classification using Transfer Learning*". Tujuan penelitian yaitu mengklasifikasi penyakit pada tanaman daun anggur menggunakan konsep metode *transfer learning* menggunakan *pre-trained* model InceptionV3, VGG16, dan VGG19 dengan *classifier* SVM, regresi logistik, dan *neural network*. Penelitian ini menggunakan dataset PlantVillage berjumlah 4062 yang terdiri dari 4 kelas dimana data pelatihan sebanyak 3209 dan data uji sebanyak 853. Didapatkan hasil klasifikasi terbaik InceptionV3 dengan classifier regresi logistik mencapai akurasi 99.4%. Kontribusi penelitian ini adalah klasifikasi penyakit pada daun anggur menggunakan metode *transfer learning* dengan memodifikasi lapisan atas model

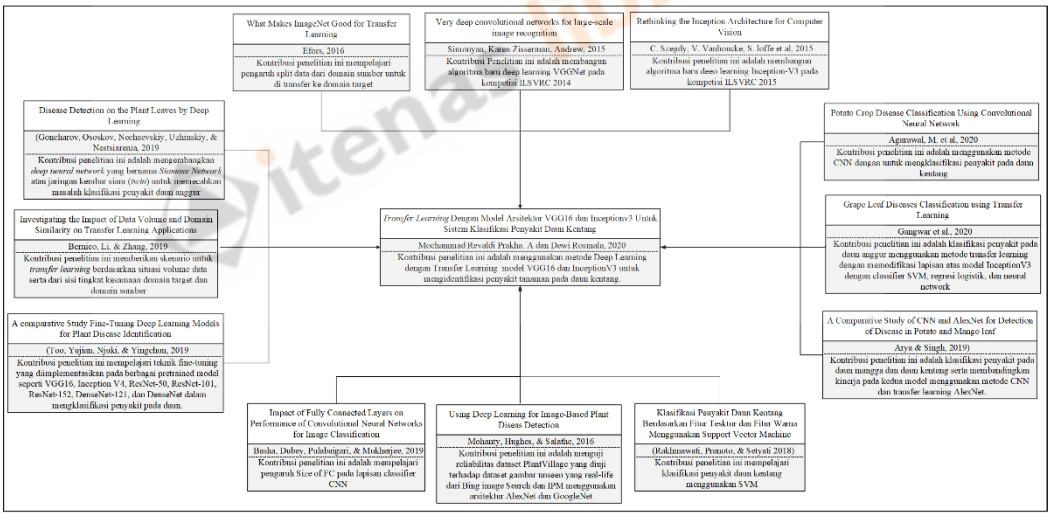
InceptionV3 dengan classifier SVM, regresi logistik, dan neural network. Pembeda penelitian yang dilakukan adalah penelitian menggunakan model VGG16 dan InceptionV3 dengan lapisan atas yang sama dengan malakukan eksperimen *Size of FC 4096x2* dan 2048x2 dengan *dropout rate* kemudian menggunakan objek penelitian daun kentang dengan 3 kelas.

(Agarawal, M. et al, 2020) melakukan penelitian dengan judul "*Potato Crop Disease Classification Using Convolutional Neural Network*". Penelitian dilakukan menggunakan CNN menggunakan empat lapisan dengan 32, 16, dan 8 filter di masing-masing lapisan. Tujuan dari penelitian ini adalah mengklasifikasi penyakit daun kentang. Dataset yang digunakan yaitu penyakit bercak kering, busuk daun, dan sehat dari website PlantVillage yang berjumlah 3000 data pelatihan. Didapatkan akurasi training sebesar 99.47% dan akurasi testing sebesar 99.8% menggunaakn 150 data pengujian. Kontribusi penelitian ini adalah menggunakan metode CNN dengan untuk mengklasifikasi penyakit pada daun kentang. Pembeda dengan penelitian yang dilakukan adalah metode yang digunakan *transfer learning* menggunakan *pre-trained* model VGG16 dan InceptionV3.

Keterhubungan tinjauan pustaka diatas memiliki keterhubungan pada penelitian ini yaitu (Rakhmawati et al., 2018) melakukan penelitian dengan objek yang sama yaitu penyakit daun kentang dengan 3 kelas (bercak kering, busuk daun, dan daun sehat) namun menggunakan metode SVM. Penelitian (Efros, 2016) menjelaskan secara detail bahwa *pre-trained weight ImageNet* menjadi standar *de-facto* untuk dilakukan metode *transfer learning*. Penelitian (Goncharov et al., 2019) melakukan penelitian klasifikasi penyakit daun anggur 4 kelas dengan metode bernama *Siamase Network* untuk mengklasifikasi penyakit daun anggur 4 kelas. Penelitian (Bernico et al., 2019) melakukan penelitian pengaruh kesamaan domain antara domain sumber dan domain target untuk menggunakan metode *transfer learning*. Penelitian (Too et al., 2019) memliki keterhubungan dengan peneltiain ini yaitu implementasi dan membandingkan metode *transfer learning* menggunakan dataset *PlantVillage* namun digunakan berbagai *pre-trained model* (VGG16,

InceptionV4, ResNet-50, ResNet-101, ResNet-152, DenseNet-121, dan DenseNet). Penelitian (Basha et al., 2019) melakukan penelitian tentang pengaruh jumlah neuron pada lapisan *Fully Connected*. Penelitian (Mohanty et al., 2016) melakukan penelitian uji reliabilitas dataset PlantVillage terhadap data uji lain (121 data IPM dan 119 Bing image Search) menggunakan arsitektur GoogleNet dan AlexNet. Kemudian penelitian terkait dengan panyakit daun dilakukan oleh (Arya & Singh, 2019) klasifikasi penyakit daun kentang dan mangga dengan CNN dan *transfer learning* AlexNet, (Gangwar et al., 2020) klasifikasi daun anggur menggukan metode *transfer learning* InceptionV3 dnegan *classifier* regresi logistik, dan (Agarawal, M. et al, 2020) klasifikasi penyakit daun kentang menggunakan metode CNN empat lapisan dengan pendekatan *caviar* dalam melatih modelnya.

Berdasarkan tinjauan pustaka tersebut, maka pada Gambar 2.18 berikut ditampilan pemetaan tinjauan pustaka yang menunjang penelitian ini.



Gambar 2. 18 Peta Tinjauan Pustaka

2.14 Pengujian Kinerja Sistem

Untuk mengevaluasi kinerja model klasifikasi dan pengenalan objek pada machine learning dan deep learning digunakan evaluasi *confusion matrix* berdasarkan precision, recall, f1-score, serta akurasi sebagai pengujian performansi model dari proses pengujian dataset test (Abas et al., 2018). Terdapat empat istilah untuk merepresentasikan hasil proses klasifikasi tersebut diantaranya :

- TP (*True Positif)* adalah data positif yang terbukti kebenarannya.
- FP (*False Positif*) adalah data negatif yang terbukti kebenarannya.
- TN (*True Negatif*) adalah data positif yang tidak terbukti kebenarannya.
- FN (*False Negatif*) adalah data negatif yang tidak terbukti kebenarannya.

Tabel 2 6

Metrics Predict

Metrics		Predict Class		
		Class 1	Class 2	Class 3
Actual Class	Class 1	True Positive		
	Class 2		True Positive	
	Class 3			True Positive

Berikut adalah penjelasan dan persamaan yang digunakan dalam menentukan kinerja model :

1. *Accuracy,* merupakan rasio prediksi benar dengan keseluruhan data. Untuk menghitung akurasi digunakan

   Persamaan 2.15 *Classification Accuracy*

$$Classification\ accuracy = \frac{correctly\ predicted\ samples}{total\ samples\ in\ the\ set} \quad (2.15)$$

2. *Precision,* merupakan bagian dari objek yang diprediksi benar. Dihitung menggunakan Persamaan (2.16) *Precision*

$$Precision = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l}(TP_i + FP_i)} \quad \text{.....(2.16)}$$

3. *Recall / Sensitivity,* digunakan untuk mengetahui seberapa akurat kinerja model untuk mengklasifikasi benar, atau dengan kata lain berapa kali model keliru mengklasifikasi *false negative.*

   Persamaan 2.17 *Recall*

$$Recall = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l}(TP_i + FN_i)} \quad \text{.....(2.17)}$$

4. F_1Score, merupakan perbandingan rata-rata *precission* dan *recall*. Dengan kata lain meringkas kinerja pengklasifikasian dengan satu metrik yang mewakili *precission* dan *recall*. Dihitung menggunakan Persamaan 2.18.

$$F_1score = \frac{2*precission*recall}{precission+recall} \quad \text{.....(2.18)}$$

Sumber : (Abas et al., 2018)

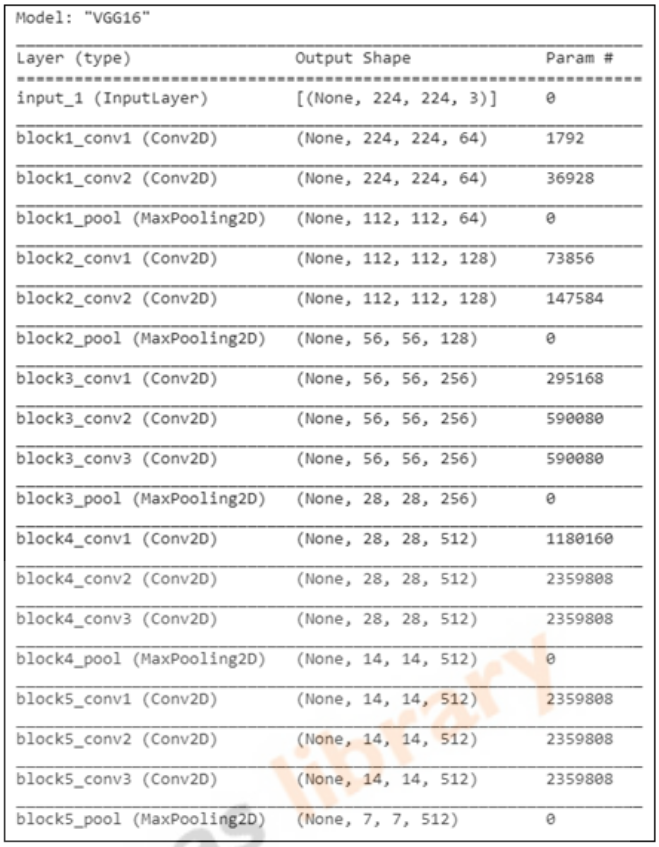
2.15 Studi Kasus

Pada subbab ini dijelaskan mengenai studi kasus dari *transfer learning* dan *fine tuning* model VGG16 dan model Inceptionv3 serta proses *preprocessing data*, konvolusi, *Batchnormalization*, *MaxPooling*, aktivasi ReLu, aktivasi *softmax* dan fungsi *loss cross entropy*.

2.15.1 *Transfer Learning* dan *Fine-Tuning* VGG16 dan Inceptionv3

Transfer learning dan *fine-tune* VGG16 pada penelitian ini dilakukan secara mundur. Pada VGG16 *tuning* dari block5 secara mundur sampai tuning seluruh jaringan. Pada studi kasus ini dilakukan tuning pada block ke-3 atau lapisan konvolusi ke-6 dengan menambah lapisan atas dengan parameter *Size of FC* 4096x2 dan *dropout* 0.4. Berikut adalah algoritma dari *transfer learning* menggunakan *pre-trained* model VGG16 :

1. Unduh bobot *Pre-trained* VGG16 tanpa lapisan atas. Didapatkan total parameter yang akan di transfer sebanyak 14.714.688, dimana pada lapisan *pre-trained* ini terdapat 14.454.528 *trainable params*, 260.160 non-*trainable params*. Gambar 2.19 meurpakan parameter *pre-trained* VGG16.

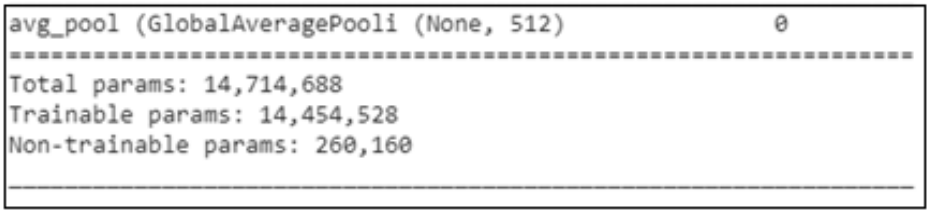


Model: "VGG16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Gambar 2. 19 *Pre-trained* VGG16

2. Tambahkan lapisan *Global Average Pooling* untuk mengubah matriks menjadi vektor 1D dari output shape block5 7x7x512 menjadi 1x1x512 seperti pada Gambar 2.20.



```
avg_pool (GlobalAveragePooli (None, 512)                0
=================================================================
Total params: 14,714,688
Trainable params: 14,454,528
Non-trainable params: 260,160
_________________________________________________________________
```

Gambar 2. 20 Hasil *add* GAP model VGG16

3. Lakukan *tuning* pada model *pre-trained*, pada studi kasus ini dilakukan *tuning* pada model *pre-trained* VGG16 lapisan ke-7 atau konvolusi blok3. *Tuning* yaitu melakukan pemilihan layer untuk di *training* ulang atau di update bobotnya. *Tranable = True* jika lapisan di *training* ulang, *Trainable False* jika lapisan tidak di *training* ulang (freeze) atau tidak di update bobotnya. Hasil *tuning* VGG16 diilustrasikan pada Gambar 2.21.

	Layer Type	Layer Name	Layer Trainable
0	<tensorflow.python.keras.engine.input_layer.In...	input_1	False
1	<tensorflow.python.keras.layers.convolutional....	block1_conv1	False
2	<tensorflow.python.keras.layers.convolutional....	block1_conv2	False
3	<tensorflow.python.keras.layers.pooling.MaxPoo...	block1_pool	False
4	<tensorflow.python.keras.layers.convolutional....	block2_conv1	False
5	<tensorflow.python.keras.layers.convolutional....	block2_conv2	False
6	<tensorflow.python.keras.layers.pooling.MaxPoo...	block2_pool	False
7	<tensorflow.python.keras.layers.convolutional....	block3_conv1	True
8	<tensorflow.python.keras.layers.convolutional....	block3_conv2	True
9	<tensorflow.python.keras.layers.convolutional....	block3_conv3	True
10	<tensorflow.python.keras.layers.pooling.MaxPoo...	block3_pool	True
11	<tensorflow.python.keras.layers.convolutional....	block4_conv1	True
12	<tensorflow.python.keras.layers.convolutional....	block4_conv2	True
13	<tensorflow.python.keras.layers.convolutional....	block4_conv3	True
14	<tensorflow.python.keras.layers.pooling.MaxPoo...	block4_pool	True
15	<tensorflow.python.keras.layers.convolutional....	block5_conv1	True
16	<tensorflow.python.keras.layers.convolutional....	block5_conv2	True
17	<tensorflow.python.keras.layers.convolutional....	block5_conv3	True
18	<tensorflow.python.keras.layers.pooling.MaxPoo...	block5_pool	True
19	<tensorflow.python.keras.layers.pooling.Global...	avg_pool	True

Gambar 2. 21 Hasil *fine-tuning pre-trained* VGG16 pada lapisan blok ke-3

4. Modifikasi lapisan atas dengan menambahkan 3 *fully connected layer* pada lapisan atas, 2 *FC layer* menggunakan 4096 nueron dengan *dropout* 0.4, dan *FC layer* terakhir menggunakan 3 neuron dengan aktivasi softmax. Hasil modifikasi lapisan atas pada model VGG16 diilustrasikan pada Gambar 2.22.

20	<tensorflow.python.keras.layers.core.Dense obj...	Dense_1	True
21	<tensorflow.python.keras.layers.core.Dropout o...	dropout	True
22	<tensorflow.python.keras.layers.core.Dense obj...	Dense_2	True
23	<tensorflow.python.keras.layers.core.Dropout o...	dropout_1	True
24	<tensorflow.python.keras.layers.core.Dense obj...	softmax	True

Gambar 2. 22 Modifikasi lapisan atas VGG16

5. Dari hasil *tuning*, didapatkan total parameter sebanyak 33.609.539, dimana 33.349.379 *trainable* dan 260.160 *non-trainable* parameter. Hasil *tuning* diilustrasikan pada Gambar 2.23.

	Layer Type	Layer Name	Layer Trainable
0	<tensorflow.python.keras.engine.functional.Fun...	VGG16	True
1	<tensorflow.python.keras.layers.core.Dense obj...	Dense_1	True
2	<tensorflow.python.keras.layers.core.Dropout o...	dropout	True
3	<tensorflow.python.keras.layers.core.Dense obj...	Dense_2	True
4	<tensorflow.python.keras.layers.core.Dropout o...	dropout_1	True
5	<tensorflow.python.keras.layers.core.Dense obj...	softmax	True

```
=================================================================
Total params: 33,609,539
Trainable params: 33,349,379
Non-trainable params: 260,160
_________________________________________________________________
```

Gambar 2. 23 Hasil *transfer learning* dan *fine-tuning* VGG16

Sedangkan *Transfer learning* dan *fine-tune* InceptionV3 pada penelitian ini mulai dari *tuning* dari modul Inception mixed10 secara mundur sampai tuning seluruh jaringan. Pada studi kasus ini dilakukan tuning pada modul Inception *mixed6* atau lapisan ke-165 dengan menambah lapisan atas menggunakan parameter *Size of FC* 4096x2 dan *dropout* 0.4. Berikut adalah algoritma dari transfer learning menggunakan *pre-trained* model InceptionV3 :

1. Unduh bobot *Pre-trained* InceptionV3 tanpa lapisan atas. Didapatkan total parameter sebanyak 21.802.784, dimana pada lapisan pre-pretrained ini terdapat 16.641.216 *trainble params*, dan 5.161.568 *non-trainable params*. *pre-trained* model InceptionV3 ditunjukan pada Gambar 2.24.

```
Model: "Inceptionv3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 111, 111, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 111, 111, 32) 96          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 111, 111, 32) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 5, 5, 192)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 5, 5, 2048)   0           activation_85[0][0]
                                                                 mixed9_1[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 16,641,216
Non-trainable params: 5,161,568
__________________________________________________________________________________________________
```

Gambar 2. 24 *Pre-trained* InceptionV3

2. Tambahkan lapisan *Global Average Pooling* untuk mengubah matriks menjadi vektor 1D dari output shape 5x5x2048 pada mixed10, setelah menambah lapisan GlobalAveragePooling menjadi 1x1x2048 seperti pada Gambar 2.25.

```
avg_pool (GlobalAveragePooling2 (None, 2048)         0           mixed10[0][0]
==================================================================================================
Total params: 21,802,784
Trainable params: 16,641,216
Non-trainable params: 5,161,568
```

Gambar 2. 25 Hasil penambahan lapisan GAP InceptionV3

3. Lakukan *transfer learning* dan *tuning* pada model *pre-trained*, pada studi kasus ini dilakukan *tuning* model InceptionV3 pada lapisan ke-165 atau modul Inception mixed6. Tuning yaitu melakukan pemilihan layer untuk di training ulang. *Trainable True* jika lapisan di *training* ulang atau di update bobotnya, *Trainable False* jika lapisan tidak di *training* ulang (freeze) atau tidak di update bobotnya. Hasil tuning InceptionV3 diilustrasikan pada Gambar 2.26.

	Layer Type	Layer Name	Layer Trainable
0	<tensorflow.python.keras.engine.input_layer.In...	input_1	False
1	<tensorflow.python.keras.layers.convolutional....	conv2d	False
2	<tensorflow.python.keras.layers.normalization_...	batch_normalization	False
3	<tensorflow.python.keras.layers.core.Activatio...	activation	False
4	<tensorflow.python.keras.layers.convolutional....	conv2d_1	False
...	...	...	...
307	<tensorflow.python.keras.layers.merge.Concaten...	mixed9_1	True
308	<tensorflow.python.keras.layers.merge.Concaten...	concatenate_1	True
309	<tensorflow.python.keras.layers.core.Activatio...	activation_93	True
310	<tensorflow.python.keras.layers.merge.Concaten...	mixed10	True
311	<tensorflow.python.keras.layers.pooling.Global...	avg_pool	True

312 rows × 3 columns

```
layer 160: activation_40, trainable: False
layer 161: activation_43, trainable: False
layer 162: activation_48, trainable: False
layer 163: activation_49, trainable: False
layer 164: mixed5, trainable: False
layer 165: conv2d_54, trainable: True
layer 166: batch_normalization_54, trainable: True
layer 167: activation_54, trainable: True
layer 168: conv2d_55, trainable: True
layer 169: batch_normalization_55, trainable: True
layer 170: activation_55, trainable: True
```

Gambar 2. 26 Hasil *tuning* InceptionV3

4. Modifikasi lapisan atas dengan menambahkan 3 *fully connected layer* pada lapisan atas, 2 FC layer menggunakan 4096 nueron dengan *dropout* 0.4, dan *FC layer* terakhir menggunakan 3 neuron dengan aktivasi *softmax*. Hasil modifikasi lapisan atas pada model InceptionV3 diilustrasikan pada Gambar 2.27.

312	<tensorflow.python.keras.layers.core.Dense obj...	Dense_1	True
313	<tensorflow.python.keras.layers.core.Dropout o...	dropout	True
314	<tensorflow.python.keras.layers.core.Dense obj...	Dense_2	True
315	<tensorflow.python.keras.layers.core.Dropout o...	dropout_1	True
316	<tensorflow.python.keras.layers.core.Dense obj...	softmax	True

Gambar 2. 27 Modifikasi lapisan atas InceptionV3

5. Dari hasil *tuning*, didapatkan total parameter sebanyak 46.989.091, 41.827.523 trainable dan 5.161.568 non-trainable parameter Hasil *tuning* diilustrasikan pada Gambar 2.28.

	Layer Type	Layer Name	Layer Trainable
0	<tensorflow.python.keras.engine.functional.Fun...	Inceptionv3	True
1	<tensorflow.python.keras.layers.core.Dense obj...	Dense_1	True
2	<tensorflow.python.keras.layers.core.Dropout o...	dropout	True
3	<tensorflow.python.keras.layers.core.Dense obj...	Dense_2	True
4	<tensorflow.python.keras.layers.core.Dropout o...	dropout_1	True
5	<tensorflow.python.keras.layers.core.Dense obj...	softmax	True

```
Total params: 46,989,091
Trainable params: 41,827,523
Non-trainable params: 5,161,568
```

Gambar 2. 28 Hasil *transfer learning* dan *fine-tuning* InceptionV3

2.15.2 *Pre-processing*

Hal yang dilakukan pada proses *pre-processing* yaitu melakukan *resize* ke ukuran 224x224. Kemudian menormalisasi data data ke interval [0,1] dengan membagi nilai piksel citra dengan 255 menggunakan persamaan (2.1). Namun, pada studi kasus ini, supaya memahami cara kerja dari proses konvolusi, nilai matriks yang digunakan hanya mengambil sebagian sampel yaitu matriks 5x5 pada *channel R*. diilustrasikasi operasi *preprocessing* sebagai berikut:

$$\text{img} = \begin{bmatrix} 137 & 133 & 164 & 163 & 143 \\ 139 & 142 & 164 & 157 & 165 \\ 142 & 142 & 142 & 162 & 157 \\ 163 & 143 & 145 & 142 & 162 \\ 162 & 143 & 145 & 145 & 167 \end{bmatrix} /255$$

Hasil setelah *pre-processing*

$$\text{img} = \begin{bmatrix} 0.5372549 & 0.6431373 & 0.643137 & 0.647059 & 0.560784 \\ 0.54509807 & 0.5568628 & 0.643137 & 0.615686 & 0.647059 \\ 0.5568628 & 0.5568628 & 0.556863 & 0.635294 & 0.615686 \\ 0.6392157 & 0.56078434 & 0.568628 & 0.556863 & 0.635294 \\ 0.63252941 & 0.56078434 & 0.568628 & 0.568628 & 0.654902 \end{bmatrix}$$

2.15.3 *Convolution*

Studi kasus yang digunakan untuk proses *convolution* yaitu menggunakan matriks hasil *preprocessing* sub-bab 2.16.2 untuk mendapatkan *feature maps*. Digunakan kernel 3x3, 1 *strides*, dan *padding = "same"*, penjelasan *padding* secara lengkap dijelasakan pada sub-bab 2.5.2. Proses konvolusi ditunjukan pada Gambar 2.29.

Matriks Hasil Preprocessing

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

Filter

-1	0	1
-2	0	2
-1	0	1

Gambar 2.29 Hasil operasi konvolusi tipe *same padding*.

Gambar 2.29 diatas menunjukan operasi konvolusi dari sebuah piksel citra 5x5 dengan filter 3x3. Proses ini menggunakan jenis *same padding*, dimana ditambahkan nilai 0 disekeliling matriks atau disebut dengan *zero padding*. Hasil *feature maps* operasi konvolusi 3x3 dari matriks input 5x5 ditampilkan pada Gambar 2.30. Adapun perhitungan output *feature maps* menggunakan *padding = "same"* sebagai berikut:

$$\text{Output} = \frac{W-F+2P}{s} + 1 \rightarrow \frac{5 - 3 + 2(\frac{3}{2})}{1} + 1 \rightarrow = 2 + 2\,(1) + 1 = 5$$

- *Catatan :* pembagian *filter* dan *strides* dibulatkan kebawah atau fungsi *math.floor()*. Maka *feature maps* yang dihasilkan ialah matriks 5x5.

Proses secara detail untuk mendapatkan hasil proses konvolusi pada Gambar 2.29 matriks 5x5 dengan filter 3x3, *strides* = (1,1) dan *padding = "same"* sebagai berikut:

1. Untuk mendapatkan nilai pada baris 1 kolom 1, pada Gambar 2.31 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.30 Konvolusui baris 1 kolom 1 dengan filter

Position (1,1) = (0*-1) + (0*0) + (0*1) + (0*-2) + (0.5372549*0) + (0.6431373*2) + (0*-1) + (0.54509807*0) + (0.5568628*1) = 1.843137

2. Untuk mendapatkan nilai pada baris 1 kolom 2, pada Gambar 2.31 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.31 Konvolusi baris 1 kolom 2 dengan filter

Position (1,2) = (0*-1) + (0*0) + (0*1) + (0.5372549*-2) + (0.6431373*0) + (0.643137*2) + (0.54509807*-1) + (0.5568628*0) + (0.6431373*1) = 0.3098

3. Untuk mendapatkan nilai pada baris 1 kolom 3, pada Gambar 2.32 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.32 Konvolusi baris 1 kolom 3 dengan filter

Position (1,3) = (0*-1) + (0*0) + (0*1) + (0.6431373*-2) + (0.643137*0) + (0.647059*2) + (0.5568628*-1) + (0.643137*0) + (0.615686*1) = 0.066667

4. Untuk mendapatkan nilai pada baris 1 kolom 4, pada Gambar 2.34 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.33 Konvolusi Baris 1 Kolom 4 dengan Filter

Position (1,4) = (0*-1) + (0*0) + (0*1) + (0.6431373*-2) + (0.647059*0) + (0.560784*2) + (0.643137*-1) + (0.615686*0) + (0.647059*1) = -0.160784

5. Untuk mendapatkan nilai pada baris 1 kolom 5, pada Gambar 2.35 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.34 Konvolusi Baris 1 Kolom 5 dengan Filter

Position (1,5) = (0*-1) + (0*0) + (0*1) + (0.64705884*-2) + (0.56078434*0) + (0*2) + (0.6156863*-1) + (0.64705884*0) + (0*1) = -1.9098

6. Untuk mendapatkan nilai pada baris 2 kolom 1, pada Gambar 2.35 dijelaskan perhitungan konvolusi dapat dilihat sebagai berikut:

0	0	0	0	0	0	0
0	0.5372549	0.6431373	0.643137	0.647059	0.560784	0
0	0.54509807	0.5568628	0.643137	0.615686	0.647059	0
0	0.5568628	0.5568628	0.556863	0.635294	0.615686	0
0	0.6392157	0.56078434	0.568628	0.556863	0.635294	0
0	0.6352941	0.56078434	0.568628	0.568628	0.654902	0
0	0	0	0	0	0	0

X

-1	0	1
-2	0	2
-1	0	1

Gambar 2.35 Konvolusi Baris 2 kolom 1 dengan Filter

Position (2,1) = (0*-1) + (0.5372549 *0) + (0.6431373x1) + (0*0.54509807) + (0.5568628*0) + (0*0.6431373) + (0*-1) + (0.5568628*0) + (0.5568628*1) = 2.313726

7. Untuk menghitung position baris kolom (2,2), (2,3), (2,4), (2,5) lakukan cara yang sama pada poin ke-2. Lalu untuk menghitung (3,1), (4,1), (5,1) lakukan cara yang sama pada poin ke-6 sehingga didapatkan hasil konvolusi matriks masukan dengan filter ditunjukan pada Gambar 2.37.

1.843137	0.309803	0.066667	-0.16078	-1.9098
2.313726	0.30196	0.199999	-0.01569	-2.51373
2.231373	0.027452	0.211764	0.188234	-2.44314
2.239216	-0.20784	0.078432	0.278429	-2.31765
1.682353	-0.20392	0.011766	0.239214	-1.69412

Gambar 2. 36 Hasil konvolusi *channel R* dengan filter

2.15.4 *Batch Normalization*

Sebagai studi kasus, akan di implementasikan pada matriks 5x5 hasil konvolusi *channel* R yang ditampilan pada Gambar 2.31. Berikut ini merupakan langkah-langkah untuk melakukan *batch normalization*:

1. Menghitung *mini batch mean*

   Lakukan operasi ini menggunakan Persamaan (2.3) dengan cara menjumlahkan setiap kolom matriks hasil konvolusi dan bagi dengan panjang kolomnya. Pada proses ini, output akan menghasilkan matriks 1x5 seperti Gambar 2.37.

   - $\frac{(1.8431374 + 2.313726 + 2.231374 + 2.239216 + 1.682353)}{5} = 2.061961$
   - $\frac{(0.309803 + 0.30196 + 0.027452 + -0.20784 + -0.20392)}{5} = 0.045491$
   - $\frac{(0.066667 + 0.199999 + 0.211764 + 0.078432 + 0.011766)}{5} = 0.1137256$
   - $\frac{(-0.16078 + -0.01569 + 0.188234 + 0.278429 + 0.239214)}{5} = 0.1058814$

- $\frac{(-1.9098 + -2.51373 + -2.44314 + -2.31765 + -1.69412)}{5}$ = -2.17588

2.061961	0.045491	0.113726	0.105882	-2.17588

Gambar 2.37 Matirks hasil penjumlahan kolom *mini batch mean*

2. Menghitung *mini batch varian* menggunakan Persamaan (2.4). Berikut merupakah langkah-langkah untuk melakukan operasi *mini batch varian*:
   a. Kurangi nilai matriks hasil konvolusi pada Gambar 2.36 dengan matriks hasil *mini batch mean* yang ditujukan pada Gambar 2.37

1.843137	0.309803	0.066667	-0.16078	-1.9098
2.313726	0.30196	0.199999	-0.01569	-2.51373
2.231373	0.027452	0.211764	0.188234	-2.44314
2.239216	-0.20784	0.078432	0.278429	-2.31765
1.682353	-0.20392	0.011766	0.239214	-1.69412

—

2.061961	0.045491	0.113726	0.105882	-2.17588

- 1.8431374 – 2.061961 = -0.218824
- 0.30980403 - 0.0454912 = 0.264312
- 0.066667 – 0.1137256 = -0.0470586
- -0.16078 – 0.1058824 = -0.2666614
- -1.9098 – (-2.175688) = 0.265888

Lakukan operasi tersebut untuk baris ke-2 sampai ke-5 sehingga menghasilkan matriks seperti Gambar 2.38.

-0.21882	0.264312	-0.04706	-0.26666	0.265888
0.251765	0.256469	0.086273	-0.12157	-0.338042
0.169412	-0.01804	0.098038	0.082353	-0.267452
0.177255	-0.25333	-0.03529	0.172548	-0.141962
-0.37961	-0.24941	-0.10196	0.133333	0.481568

Gambar 2.38 Matriks *mini batch variance*

b. Kuadratkan setiap elemen pada matriks *mini batch variance* pada Gambar 2.38 sehingga menghasilkan matriks seperti Gambar 2.39. Seperti contoh pada baris 1 kolom 1 (1,1) memiliki nilai $(-0.21882)^2$ = 0.047884

0.047884	0.069861	0.002215	0.071108	0.070696429
0.063386	0.065776	0.007443	0.01478	0.114272394
0.0287	0.000325	0.009612	0.006782	0.071530572
0.031419	0.064177	0.001246	0.029773	0.020153209
0.144102	0.062206	0.010396	0.017778	0.231907739

Gambar 2.39 Hasil kuadrat dari matriks *mini batch variance*

c. Jumlahkan setiap kolom hasil kudrat *mini batch vairance* yang terdapat pada matriks Gambar 2.39 kemudian bagi dengan panjang kolomnya sehingga menghasilkan matriks 1x5 seperti pada Gambar 2.41. Berikut ini adalah cara perhitungannya:

- $\frac{(0.047884 + 0.063386 + 0.0287 + 0.031419 + 0.144102)}{5}$ = 0.063098
- $\frac{(0.069861 + 0.065777 + 0.000325 + 0.06417 + 0.062206)}{5}$ = 0.052469
- $\frac{(0.002215 + 0.007443 + 0.009612 + 0.001246 + 0.010396)}{5}$ = 0.006182
- $\frac{(0.071109 + 0.01478 + 0.006782 + 0.029773 + 0.017778)}{5}$ = 0.028044
- $\frac{(0.070696429 + 0.114272394 + 0.071530572 + 0.020153209 + 0.231907739)}{5}$

  = 0.101712069

0.063098	0.052469	0.006182	0.028044	0.101712069

Gambar 2.40 Matriks Hasil operasi langkah 2c

3. Hitung normalisasi menggunakan persamaan (2.5) pada matriks hasil *mini batch varian* langkah 2c yang ada di Gambar 2.40.

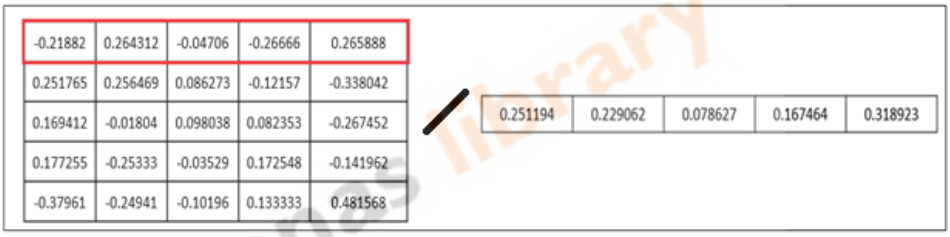
a. Hitung akar pangkat dua dari hasil penjumlahan setiap element matriks pada matriks hasil *mini batch vairan* langkah 2c pada Gambar 2.41 kemudian dijumlahkan dengan 10^{-8} sehingga menghasilkan matriks seperti Gambar 2.41.

$$\sqrt[2]{0.063098 + 10^{-8}} = 0.251194$$

0.251194	0.229062	0.078627	0.167464	0.318923

Gambar 2.41 Matriks hasil operasi langkah 3a

b. Hitung pembagain setiap baris pada matriks hasil 2a *mini batch variance* di Gambar 2.38 dengan hasil matriks normalisasi langkah 3a di Gambar 2.41. Berikut merupakan hasil perhitungannya pada baris pertama 2a/3a:



- -0.21882/0.251194= -0.87114
- 0.264313/0.229062= 1.153894
- -0.04706/0.078627 = -0.5985
- -0.26666/0.167464 = -1.59235
- 0.265888 /0.318923 = 0.833705

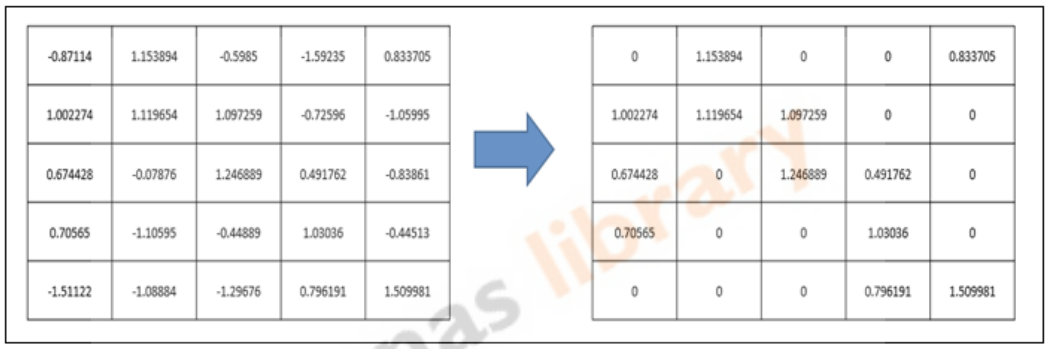
Lakukan opesi pembagian pada baris kedua, ketiga, dan keempat di matriks 2a Gambar 2.38 sehingga didapat matriks pada Gambar 2.42.

-0.87114	1.153893	-0.59851	-1.59236	0.833705
1.002274	1.119653	1.097258	-0.72596	-1.05995
0.674428	-0.07875	1.246889	0.491764	-0.83861
0.70565	-1.10595	-0.44888	1.030359	-0.44513
-1.51122	-1.08884	-1.29676	0.796189	1.509981

Gambar 2.42 Matriks hasil *Batchnormalization*

2.15.5 Aktivasi ReLu

Pada proses ini dilakukan menggunakan persamaan (2.7) pada sub-bab 2.5.4. ReLu dilakukan pada setiap hasil konvolusi dan atau setelah *batch normalization*, studi kasus ini menggunakan matriks hasil *batchnormalization* Gambar 2.42 dengan mengubah nilai negatif menjadi sama dengan 0, dan mempertahankan yang nilainya lebih besar dari 0. Contohnya pada elemen matriks baris 1 kolom 1 (1,1) memiliki nilai -0.87114 setelah dilakukan operasi ReLu akan menjadi 0. Sehingga matriks hasil operasi ReLu terhadap matriks Gambar 2.42 menghasilkan matriks seperti pada Gambar 2.43.



Gambar 2.43 Hasil Aktivasi ReLu

2.15.6 *Pooling*

Operasi *pooling* terbagi menjadi 3 jenis, yaitu *max pooling*, *average pooling*, dan *global average pooling*. Adapun studi kasus *pooling* dijelaskan sebagai berikut.

2.15.6.1 *Max pooling dan Average pooling*

Operasi *maxpooling* yang dilakukan pada VGG16 yaitu menggunakan perkalian dengan filter 3x3 dan 2 *strides* terhadap matriks hasil aktivasi ReLu. Berikut adalah hasil operasi *max pooling* yang ditampilkan pada Gambar 2.45 dan *Average pooling* pada Gambar 2.46.

- Sebelum menghitung isi dari dalam matriksnya, perlu di ketahui telebih dahulu peta fitur yang akan didapat, untuk kasus *max pooling* dan *average pooling* digunakan persamaan (2.8)

Parameter studi kasus *Max pooling* dan *average pooling* Matriks input : 5x5 Ukuran kernel konvolusi : 3x3 Ukuran *stride* : 2

$$Output_{maxpool} = \frac{n-f}{S} + 1 = \frac{5-3}{2} + 1 = 2$$

Maka peta fitur yang didapat yaitu matriks 2x2.

0	1.153894	0	0	0.833705
1.002274	1.119654	1.097259	0	0
0.674428	0	1.246889	0.491762	0
0.70565	0	0	1.03036	0
0	0	0	0.796191	1.509981

Gambar 2.44 Operasi *max pooling* 3x3, 2 *strides*

1.246889	1.246889
1.246889	1.509981

Gambar 2.45 Hasil *max pooling 3x3, 2 strides*

Berikut adalah perhitungan hasil *Average pooling*:

- $\frac{(0 + 1.153894 + 0 + 1.002274 + 1.119654 + 1.097259 + 0.674428 + 0 + 1.246889)}{9}$

  = 0.6994
- $\frac{(0{,}089 + 0{,}035 + 0 + 0 + 0 + 0 + 0{,}063 + 0 + 0)}{9}$ = 0.4068
- $\frac{(0{,}467 + 0 + 0.063 + 0 + 0 + 0 + 0{,}141 + 0{,}182 + 0{,}249)}{9}$ = 0.2919

- $\frac{(0{,}063 + 0 + 0 + 0 + 0{,}125 + 0 + 0{,}249 + 1{,}690 + 0)}{9} = 0.5639$

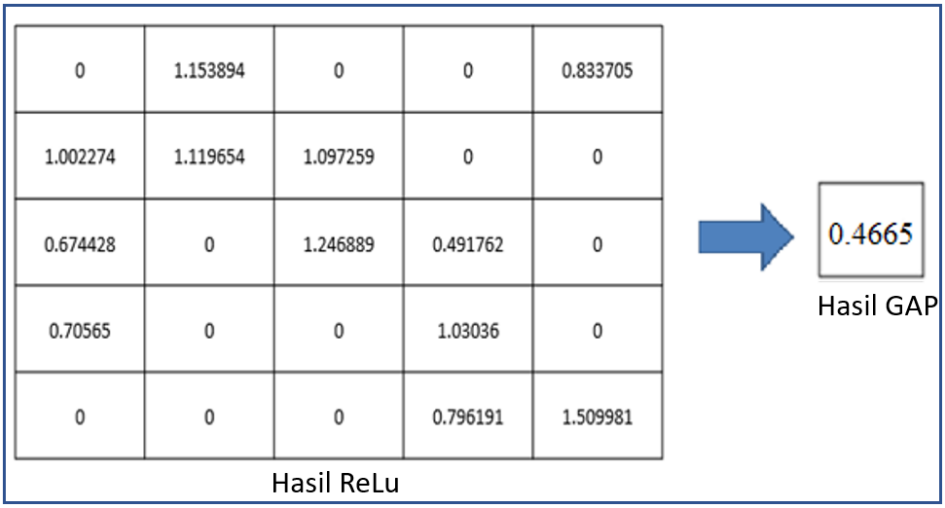
0.6994	0.4068
0.2919	0.5639

Gambar 2.46 Hasil *Average pooling*

2.15.6.2 *Global Average Pooling*

Setelah diterapkan beberapa *convolution layer* dengan model VGG16 dan InceptionV3 untuk pembelajaran fitur, tahap selanjutnya dilanjutkan pada lapisan klasifikasi yaitu GAP. Pada penelitain ini kedua model menggunakan GAP untuk mengubah *feature maps* menjadi fitur vektor 1D. Pada studi kasus ini, digunakan matriks 5x5 sebagai contoh dengan jumlah filter 1. Di dapatkan operasi GAP terhadap matriks hasil aktivasi ReLu ditunjukan pada Gambar 2.47. Untuk lebih jelasnya dijabarkan sebagai berikut:

- (0 + 1.153894 + 0 + 0 + 0.833705 + 1.002274 + 1.119654 + 1.0977259 + 0+0+0.674428+0+1.246889+0.491762+0+0.70565+0+0+1.03036+0+0+0 +0+0+0.796191+1.509981)/25 = 0.4665

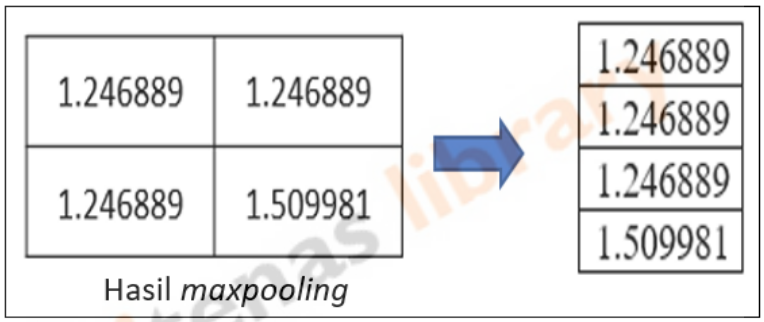


Gambar 2.47 Hasil operasi *global average pooling*

Nilai GAP pada Gambar 2.47 masih menghitung pada 1 *channel R* dengan jumlah filter 1, banyaknya hasil GAP tergantung jumlah filter yang ada pada masing-masing operasi konvolusi di setiap model. Setelah GAP dilanjutkan dengan lapisan *fully connected layer*.

2.15.7 *Fully Connected Layer*

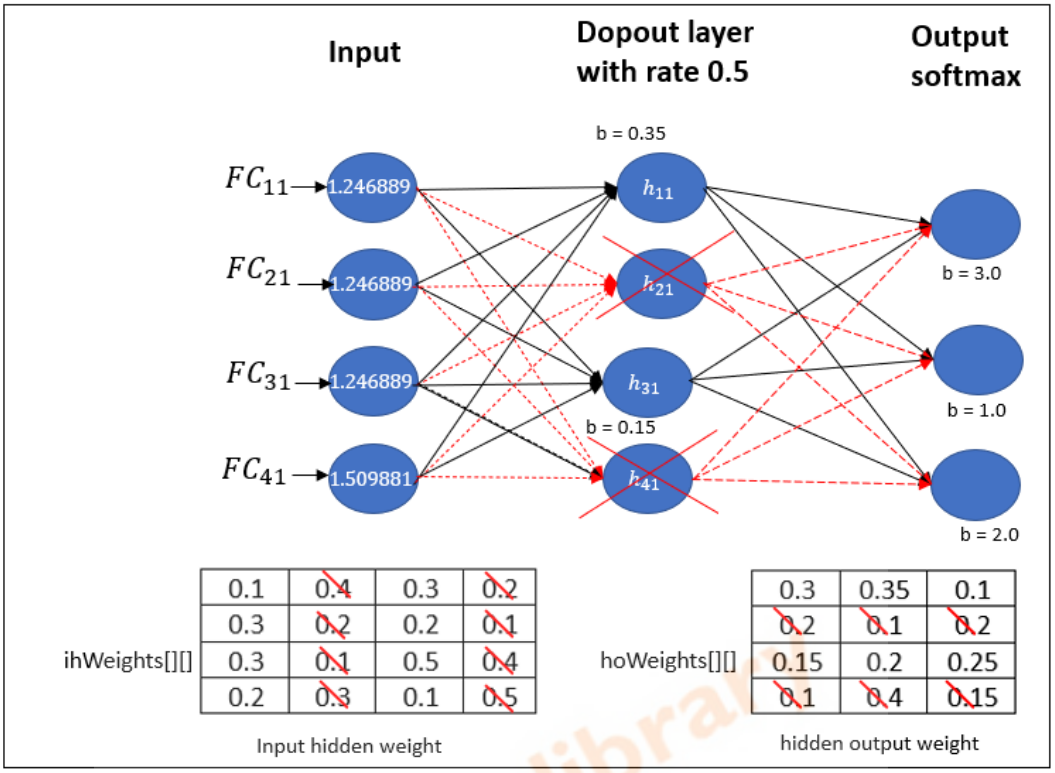
Fully Connected adalah lapisan terakhir dimana data berupa matriks ditumpuk dengan bentuk 1D (vektor), sehingga bisa dilakukan klasifkasi dengan lebih mudah karena bersifat linear. Untuk contoh studi kasus operasi *fully connected* diterapkan pada matriks hasil *maxpooling* yang terdapat pada Gambar 2.45 kemudian ditampilkan hasilnya pada Gambar 2.48.



Gambar 2. 48 Hasil *fully connected layer*

2.15.8 *Dropout*

Proses *dropout* merupakan proses menghilangkan neuron yang terdapat pada *hidden layer* di dalam jaringan. Neuron yang akan dihilangkan dipilih secara acak. Hal ini dapat dibilang bahwa kontribusi neuron yang dihilangkan diberhentikan. *Dropout* di implementasikan pada masing-masing lapisan *fully connected* yang merupakan konfigurasi paling umum yang diusulkan oleh (Srivastava et al., 2014) Namun ada juga pada lapisan konvolusi setelah operasi ReLu (Liu, Tian, & B, 2017). Pada studi kasus ini di implementasikan setelah lapisan *fully connected* yaitu yang ditunjukan pada Gambar 2.14, diberikan contoh menggunakan *dropout rate* 0.5 atau dihilangkan sementara sebesar 50%. Nilai bobot neuron yang dihilangkan akan sama dengan 0.



Gambar 2. 49 ilustrasi proses *dropout*

Adapaun penjelasan dari perhitungan di dalam *hidden* layer untuk proses *dropout* adalah sebagai berikut :

$h_{11} = ((\mathrm{FC}_{11} * 0.1) + (F\mathrm{FC}_{21} * 0.3) + (\mathrm{FC}_{31} * 0.3) + (\mathrm{FC}_{41} * 0.2) + b)$

$h_{11} = ((1{,}246889 * 0.1) + (1{,}246889 * 0.3) + (1{,}246889 * 0.3) + (1{,}54098810 * 0{,}2) + 0{,}35)$

$h_{11} = (0{,}1247 + 0{,}3741 + 0{,}3741 + 0{,}1082 + 0{,}35) = 1{,}331$

$h_{31} = ((\mathrm{FC}_{11} * 0{,}3) + (\mathrm{FC}_{21} * 0{,}2) + (\mathrm{FC}_{31} * 0{,}5) + (\mathrm{FC}_{31} * 0{,}1) + b)$

$h_{31} = ((1{,}246889 * 0{,}3) + (1{,}246889 * 0{,}2) + (1{,}246889 * 0{,}5) + (1{,}54098810 * 0.1) + 0{,}15)$

$h_{31} = (0{,}3741 + 0{,}2494 + 0{,}6234 + 0{,}1541 + 0{,}15) = 1{,}551$

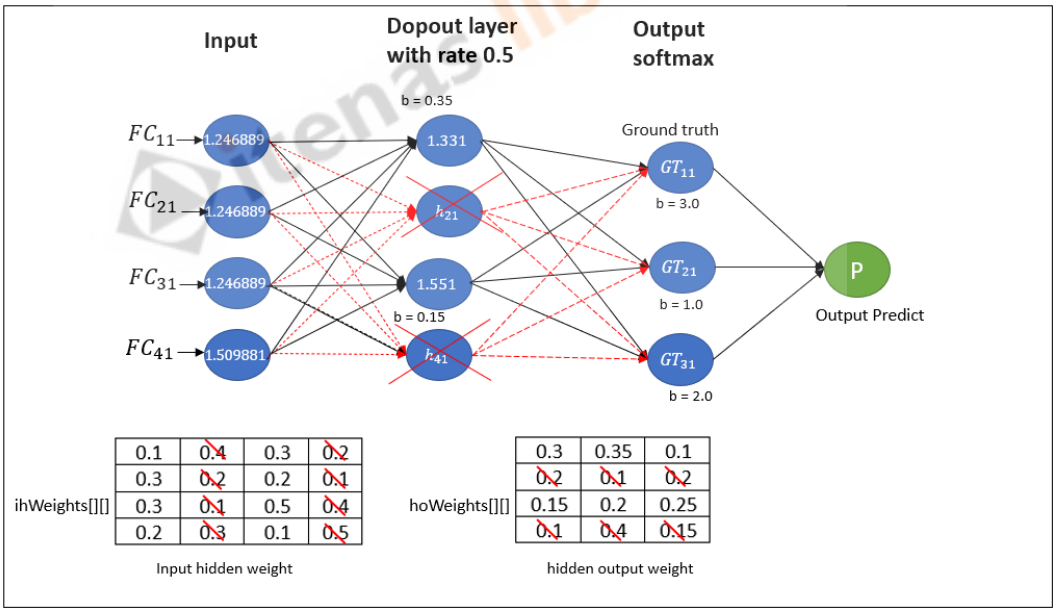
Setelah diterapkan dropout pada lapisan *fully connected*, selanjutnya dilakukan perhtiungan probabilitas masing-masing kelas menggunakan *softmax.*

2.15.9 *Softmax*

Fungsi *softmax* diterapkan pada vektor 1D *fully connected layer* dengan *dropout* yang dijelaskan pada Gambar 2.49, *softmax* dugunakan untuk mendapat nilai probabilitas suatu obyek terhadap kelas tertentu. Perhitungan hasil aktivasi *softmax* dijelaskan sebagai berikut dimana aktivasi *softmax* dihitung menggunakan Persamaan (2.10) dimana eksponen e = 2.7183:

$$S_j = \frac{e^{z_j}}{\sum_{d=1}^{C} e^{z_d}}$$

1. Misalkan untuk studi kasus, penelitian ini terdapat 3 fitur vektor karena terdiri dari 3 kelas, maka dari itu di hitung 3 *ground truth* dari hasil vektor *fully connected layer*. Karena proses ini terjadi di *hidden layer*, maka dari itu tidak dapat dilihat nilai *weight* dan bias nya. Oleh karena itu nilai *weight* dan bias di inisialisasi secara *random* yang diilustrasikan di Gambar 2.50.



Gambar 2. 50 Proses perhitungan *fully connected* dengan *dropout* menggunakan aktivasi *softmax*

2. Perhitungan propagasi maju menggunakan aktivasi *softmax*

$$GT_{11} = ((h_{11} * 0{,}3) + (h_{31} * 0{,}15) + b)$$

$$GT_{11} = ((1{,}331 * 0{,}3) + (1{,}551 * 0{,}15) + 3.0)$$

$$GT_{11} = 3.4819$$

$$GT_{21} = GT_{11} = ((h_{11} * 0{,}35) + (h_{31} * 0{,}2) + b)$$
$$GT_{21} = ((1{,}331 * 0.35) + (1{,}551 * 0{,}2) + 1.0)$$
$$GT_{21} = 1{,}7761$$

$$GT_{31} = GT_{11} = ((h_{11} * 0.1) + (h_{31} * 0{,}25) + b)$$
$$GT_{31} = ((1{,}331 * 0.1) + (1{,}551 * 0{,}25) + 2{,}0)$$
$$GT_{31} = 2{,}5208$$

$$\text{Softmax}(GT_{11}) = \frac{e^{(GT_{11})}}{(e^{(GT_{11})} + e^{(GT_{21})} + e^{(GT_{31})})}$$

$$\text{Softmax}(GT_{11}) = \frac{2{,}7183^{3{,}4819}}{(2{,}7183^{3{,}4819} + 2{,}7183^{1.7761} + 2{,}7183^{2{,}5208})}$$

$$\text{Softmax}(GT_{11}) = \frac{32{,}5215}{32{,}5215 + 3{,}2466 + 9{,}4953)}$$

$$\text{Softmax}(GT_{11}) = 0{,}7185$$

$$\text{Softmax}(GT_{21}) = \frac{e^{(GT_{21})}}{(e^{(GT_{11})} + e^{(GT_{21})} + e^{(GT_{31})})}$$

$$\text{Softmax}(GT_{21}) = \frac{2.7183^{1{,}7761}}{(2{,}7183^{3{,}4819} + 2{,}7183^{1.7761} + 2{,}7183^{2{,}5208})}$$

$$\text{Softmax}(GT_{21}) = \frac{3{,}2466}{32{,}5215 + 3{,}2466 + 9{,}4953)}$$

$$\text{Softmax}(GT_{21}) = 0{,}0717$$

$$\text{Softmax}(GT_{31}) = \frac{e^{(GT_{31})}}{(e^{(GT_{11})} + e^{(GT_{21})} + e^{(GT_{31})})}$$

$$\text{Softmax}(GT_{31}) = \frac{2{,}7183^{2{,}5208}}{(2{,}7183^{3{,}4819} + 2{,}7183^{1.7761} + 2{,}7183^{2{,}5208})}$$

$$\text{Softmax}(GT_{31}) = \frac{9{,}4953}{32{,}5215 + 3{,}2466 + 9{,}4953)}$$

Softmax(GT_{31}) = 0.2098

Dari hasil perhitungan untuk menghitung probablitias *ground truth* dari 3 kelas menghasilkan nilai *softmax* sebagai berikut, dimana hasil prediksi mendapatkan *pribabilitas* nilai p tertinggi pada kelas pertama.

$s = [0.7185, 0.0717, 0.2098]$

maka jika di konversi ke dalam nilai sesungguhnya menjadi

$p = [1,0,0]$

❖ Catatan : probabilitas hasil *softmax* harus sama dengan 1 jika di jumlahkan.

Setelah di dapat hasil preiksi, selanjtunya dihitung eror nya menggunakan *cross entropy*.

2.15.10 *Categoircal Corss Entropy*

Fungsi *loss* diterapkan untuk menghitung eror yang berguna mengupdate bobot se-minimum mungkin, pada penelitian ini menggunakan *categorical cross entropy*. Adapun penjelasan perhitungan studi kasusnya sebagai berikut:

a. Fungsi *loss* menggunakan Persamaan (2.10) :

$$D(s,p) = -\sum_{j} p_j \log s_j$$

Dimana:

S = Nilai hasil prediksi

$p =$ nilai sesungguhnya

$j =$ jumlah target kelas

Asumsi probabilitas kelas tertinggi yang di didapatkan dari *softmax* adalah kelas ke-satu, sehingga indeks ke-satu menjadi nilai p = 1, adapun perhitungan nya sebagai berikut:

s = $\begin{bmatrix} 0.7185 \\ 0.0717 \\ 0.2098 \end{bmatrix}$, p = $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$D(s,p) = -(p_1 log s_1 + p_2 log s_2 + p_3 log s_3)$

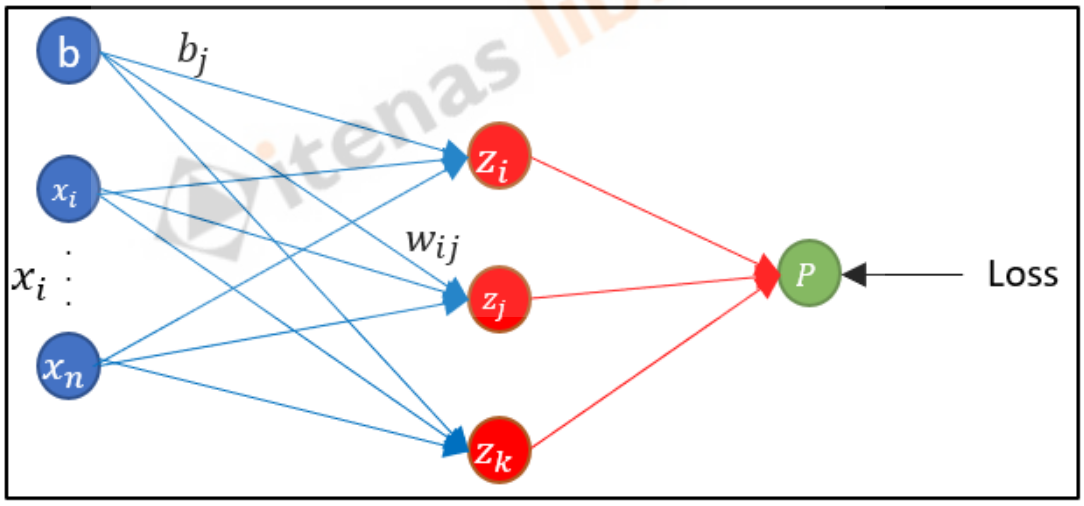
$D(s,p) = -1 * \log(0.7185) - 0 * log(0.0717) - 0 * \log(0.2098)$

$= -\log(0.7185) \approx 0.1436$

Dari perhitungan *cost* atau eror menggunakan *cross entropy*, diperoleh nilai *loss function* sebesar 0.1436. Nilai tersebut masih cukup besar untuk sistem klasifikasi, karena yang semakin baik adalah yang sangat mendekati 0. Sehingga diperlukan proses propagasi mundur atau *backward pass* untuk memperbaiki parameter-parameter yang mempengaruhi model.

2.15.11 *Backward Pass*

Proses *backward pass* atau propagasi mundur digunakan untuk meminimalkan nilai *loss*, perlu dilakukan *update* pada parameter-parameter yang mempengaruhi pembelajaran model seperti nilai bobot dan bias. Pada peneleitian ini menggunakan metode *update* parameter menggunakan *Stochastic Gradient Descent.* Nilai dari parameter-parameter yang baru di dapat melalui pengurangan nilai parameter lama dengan nilai *gradient* yang diperoleh. Pada Gambar 2.51 adalah ilustrasi bagaimana *backward pass* berjalan.



Gambar 2. 51 Ilustrasi *fully connected* dengan aktivasi *softmax*

Pada tahap pertama *backward pass*, dilakukan perhitungan $\frac{\partial D}{\partial z_i}$ yang merupakan gradien dari *cross entropy* untuk fungsi *softmax* menggunakan *chain rule* sebagai berikut.

$$\frac{\partial D}{\partial z_i} = -\sum_{j=1}^{c} \frac{\partial \log s_j}{\partial z_i}$$

$$= -\sum_{j=1}^{c} p_j \frac{\partial\ log s_j}{\partial z_i}, \text{dimana } \partial\ log s_j = \frac{\partial s_j}{s_j} \log e$$

$$= -\sum_{j=1}^{c} p_j \frac{\partial\ s_j}{S_j} \frac{\log e}{\partial z_i}$$

$$= -\sum_{j=1}^{c} p_j \frac{\partial\ s_j}{S_j} \frac{1}{\partial z_i}$$

$$= -\sum_{j=1}^{c} \frac{p_j}{S_j} \frac{\partial\ s_j}{\partial z_i}$$

$$= -\frac{p_i}{s_i} \frac{\partial s_i}{\partial z_i} - \sum_{j \neq 1}^{c} \frac{p_j}{S_j} \frac{\partial\ s_j}{\partial z_i}, \text{untuk } \frac{\partial s_i}{\partial z_i} = s_i(1 - s_i) \text{ dan } \frac{\partial s_j}{\partial z_i} = -s_i s_j$$

$$= -\frac{p_i}{s_i} s_i(1 - s_i) - \sum_{j \neq 1}^{c} \frac{p_j}{s_j} (-s_i s_j)$$

$$= -p_i + p_i s_i + \sum_{j \neq 1}^{c} p_j s_i$$

$$= -p_i + \sum_{j=1}^{c} p_j s_i$$

$$= -p_i + s_i \sum_{j=1}^{c} p_j$$

$$= s_i - p_i$$

Keterangan :

z_j = nilai keluaran dari jaringan

s_i= menyatakan probabiltas dari $z_j$untuk setiap C kelas yang berbeda

p_j = nilai sesungguhnya

j = jumlah target kelas

Sumber : (Wulandari et al., 2019)

Dari persamaan diatas, akan digunakan untuk memperoleh nilai bobot $\frac{\partial D}{\partial w_{ji}}$ dan nilai bias $\frac{\partial D}{\partial b_j}$. Sehingga, $\frac{\partial D}{\partial w_{ji}}$ diperoleh persamaan (2.19) dan untuk $\frac{\partial D}{\partial b_j}$ didapatkan persamaan (2.20).

$$\frac{\partial D}{\partial w_{ji}} = \frac{\partial D}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = (s_j - p_j)\frac{\partial(\sum_{j=1}^{c} w_{ji}x_i + b_j)}{\partial w_{ji}} = (s_j - p_j)x_i \quad .....(2.19)$$

$$\frac{\partial D}{b_j} = \frac{\partial D}{\partial z_j}\frac{\partial z_j}{\partial b_j} = (s_j - p_j)\frac{\partial(\sum_{j=1}^{c} w_{ji}x_i + b_j)}{\partial w_{ji}} = (s_j - p_j) \quad .....(2.20)$$

Keterangan :

b_j = bias dari jaringan

j = jumlah target kelas

w_{ji} = bobot dari jaringan berukuran *j x i*

Sumber : (Wulandari et al., 2019)

Kemudian persamaan diatas digunakan untuk *update* nilai bobot dan bias. Pada penelitian ini menggunakan *learning rate* 0.0001. Namun, pada studi kasus ini digunakan nilai *learning rate* η sebesar 0.001 agar nilai *output* tidak terlalu kecil sehingga proses perhitungan dapat mudah dipahami. Untuk nilai α secara umum yaitu menggunakan α = 0.0 karena pada penelitian menggunakan *framework* keras yang secara *deefault* tidak diterapkan momentum.

$$w_{jk}(t+1) = w_{jk}(t) - \eta\frac{\partial D}{\partial w_{ji}} + \alpha\Delta\, w_{jk}(t-1)$$

$$= w_{jk}(t) - 0.001\,(s_j - p_j)(\mathrm{t})x_i + 0.0\Delta\, w_{jk}(t-1)$$

$$b_j(t+1) = b_j(t) - \eta\frac{\partial D}{\partial b_j} + \alpha\Delta\, b_j(t-1)$$

$$= b_j(t) - 0.001\,(s_j - p_j)(\mathrm{t}) + 0.0\Delta\, b_{jk}(t-1)$$

Berikut adalah merupakan contoh *update* bobot dimana nilai awal Δw_{jk}(t - 1) dan $\Delta\, b_{jk}$(t - 1) belum bisa didapat karena merupakan nilai perubahan bobot dan bias pada saat terakhir kali di *update*, maka dari itu pada contoh ini nilai awal Δw_{jk}(t - 1) = 0 dan $\Delta\, b_{jk}$(t - 1) = 0 karena masih pembaruan bobot pertama.

Pada studi kasus ini menggunakan nilai *hidden output weight* pada Gambar 2.50.

Misalkan menghitung *update weight* ke-1, maka

$$w_{jk}\,(\mathrm{t}+1) = w_{jk}(\mathrm{t}) - 0.001\,(s_j - p_j)(\mathrm{t})x_i + 0.0\Delta w_{jk}(t-1)$$

$$w_{j1}(2) = w_{jk}(1) - 0.001(s_j - p_j)(1)\,x_i$$

$$= \begin{bmatrix} 0.3 & 0 & 0.15 & 0 \\ 0.35 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0.25 & 0 \end{bmatrix} -$$

$$0.001 \begin{bmatrix} 0.7185 \\ 0.0717 \\ 0.2098 \end{bmatrix} \begin{bmatrix} 1.246889 & 1.246889 & 1.246889 & 1.509981 \end{bmatrix}$$

$$= \begin{bmatrix} 0.3 & 0 & 0.15 & 0 \\ 0.35 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0.25 & 0 \end{bmatrix} - 0.001 \begin{bmatrix} 0.3549 & 0.3549 & 0.3549 & 0.4298 \\ 0.4238 & 0.4238 & 0.4238 & 0.5132 \\ 0.4669 & 0.4669 & 0.4238 & 0.5654 \end{bmatrix}$$

$$= \begin{bmatrix} 0.3 & 0 & 0.15 & 0 \\ 0.35 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0.25 & 0 \end{bmatrix} - \begin{bmatrix} 0.0003549 & 0.0003549 & 0.0003549 & 0.00042988 \\ 0.0004238 & 0.0004238 & 0.0004238 & 0.0005132 \\ 0.0004669 & 0.0004669 & 0.0004238 & 0.0005654 \end{bmatrix}$$

$$= \begin{bmatrix} 2.99967 & -0.0003549 & 0.14965 & -0.00042988 \\ 0.34957 & -0.0004238 & 0.19958 & -0.00042988 \\ 0.09953 & -0.0004669 & 0.24957 & -0.0005654 \end{bmatrix}$$

$$b_j\,(\mathrm{t}+1) = b_j(\mathrm{t}) - 0.001\,(s_j - p_j)(\mathrm{t}) + 0.0\Delta b_{jk}(t-1)$$

Misalkan menghitung bias ke-1, maka

$$b_1(2) = b_j(1) - 0.001(s_j - p_j)(1)$$

$$= \begin{bmatrix} 1.246889 \\ 1.246889 \\ 1.246889 \\ 1.509981 \end{bmatrix} - 0.001 \begin{bmatrix} 0.7185 \\ 0.0717 \\ 0.2098 \end{bmatrix}$$

$$= \begin{bmatrix} 1.246889 \\ 1.246889 \\ 1.246889 \\ 1.509981 \end{bmatrix} - \begin{bmatrix} 0.0007185 \\ 0.0000717 \\ 0.0002098 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1.2462705 \\ 1.2468173 \\ 1.2466792 \\ 1.509981 \end{bmatrix}$$

Sehingga didiapat parameter baru setelah di *update* adalah sebagai berikut

$$w_{j1} = \begin{bmatrix} 2.99967 & -0.0003549 & 0.14965 & -0.00042988 \\ 0.34957 & -0.0004238 & 0.19958 & -0.00042988 \\ 0.09953 & -0.0004669 & 0.24957 & -0.0005654 \end{bmatrix}$$

$$b_1 \begin{bmatrix} 1.2462705 \\ 1.2468173 \\ 1.2466792 \\ 1.509981 \end{bmatrix}$$

Setelah dilakukan *update* bobot, proses kembali lagi ke *forward pass* atau propagasi maju kemudian proses *backward pass* ini dilakukan kembali sampai kita mendapatkan nilai *loss* paling kecil.