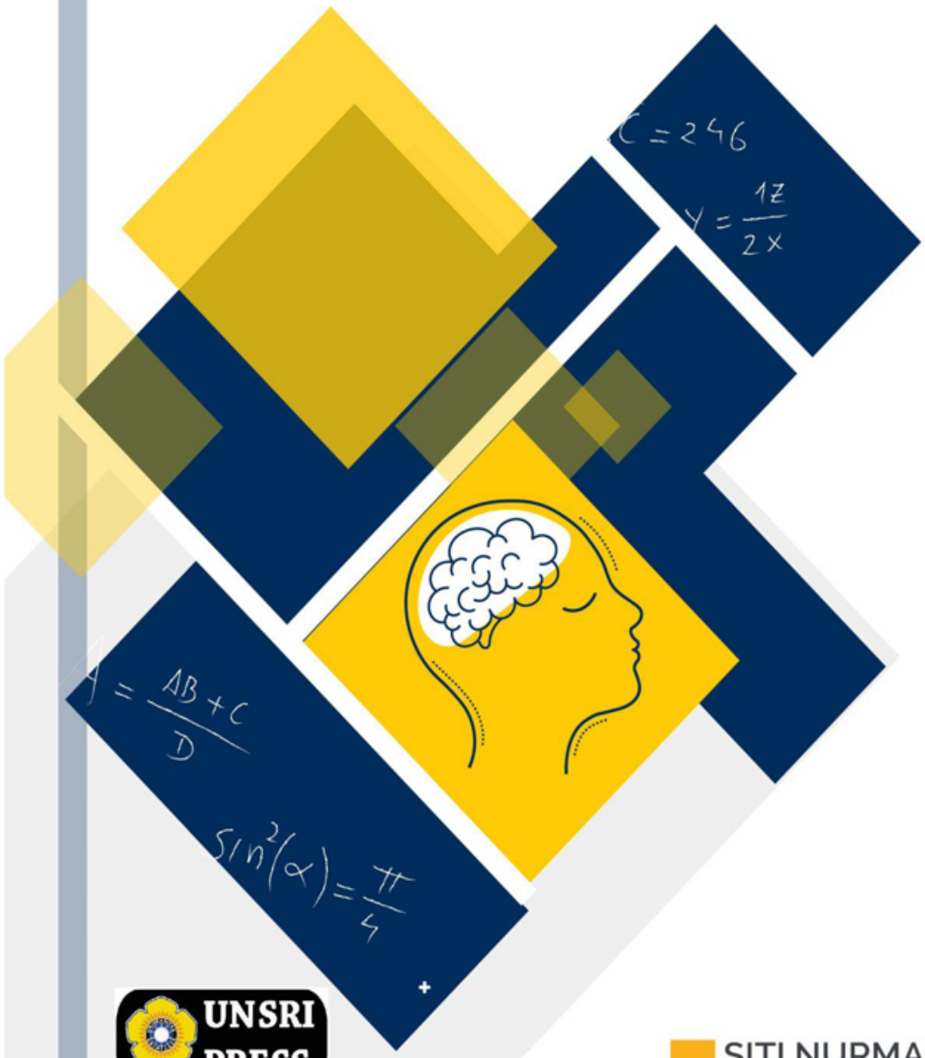
ISBN 978-979-587-995-4

PENGENALAN DEEP LEARNING DAN IMPLEMENTASINYA

Siti Nurmaini, dkk

PENGENALAN DEEP LEARNING DAN IMPLEMENTASINYA



UNSRI PRESS

SITI NURMAINI, dkk.
UNIVERSITAS SRIWIJAYA

PENGENALAN DEEP LEARNING DAN IMPLEMENTASINYA

Sanksi pelanggaran Pasal 72
Undang-undang Nomor 19 Tahun 2002
Tentang Perubahan atas Undang-undang Nomor 12 Tahun 1997
Pasal 44 Tentang Hak Cipta

1. Barang siapa dengan sengaja dan tanpa hak melakukan perbuatan sebagaimana dimaksud dalam Pasal 2 ayat (1) atau pasal 49 ayat (1) dan ayat (2) dipidana penjara masing-masing paling singkat 1 (satu) bulan dan/atau denda paling sedikit Rp. 1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp. 5.000.000.000,00 (lima miliar rupiah)

2. Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran hak cipta atau hak terkait, sebagaimana dimaksud ayat (1) dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau dendan paling banyak Rp.500.000.000,00 (lima ratus juta rupiah)

PENGENALAN DEEP LEARNING DAN IMPLEMENTASINYA

Siti Nurmaini
Annisa Darmawahyuni
Ade Iriani Sapitri
Muhammad Naufal Rachmatullah
Firdaus
Bambang Tutuko

UNSRI PRESS

PENGENALAN DEEP LEARNING DAN IMPLEMENTASINYA

Siti Nurmaini
Annisa Darmawahyuni
Ade Iriani Sapitri
Muhammad Naufal Rachmatullah
Firdaus
Bambang Tutuko

Tim Editor dan Tata Letak : Annisa Darmawahyuni

UPT. Penerbit dan Percetakan
Universitas Sriwijaya 2021
Kampus Unsri Palembang
Jalan Srijaya Negara, Bukit Besar Palembang 30139
Telp. 0711-360969
email : unsri.press@yahoo.com, penerbitunsri@gmail.com
website : www.unsri.unsripress.ac.id

Anggota APPTI No. 005.140.1.6.2021
Anggota IKAPI No. 001/SMS/96

137 halaman : 16 x 24 cm

ISBN : 978-979-587–995-4

PENGANTAR

Assalamu'alaikum Wr. Wb

Bismillahirahmannirrohiim,

Puji syukur dipanjatkan kehadirat Allah SWT atas limpahan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan buku dengan judul "Pengenalan *Deep Learning* dan Implementasinya". Buku ini menjelaskan mengenai dasar ilmu *Deep Learning* dan implementasinya dalam beberapa bidang antara lain citra dan sinyal medis, serta penamaan *author* dalam publikasi ilmiah.

Penulis berharap buku ini dapat bermanfaat bagi orang banyak, meski masih banyak kekurangan dan jauh dari kesempurnaan. Dalam penyusunan buku ini, penulis banyak mendapat dukungan dan bantuan dari berbagai pihak, baik moril maupun materil, sehingga buku ini dapat diselesaikan.

Akhir kata, dengan segala kerendahan hati dan keterbatasan, penulis berharap buku ini menghasilkan sesuatu yang bermanfaat, khususnya bagi Fakultas Ilmu Komputer Universitas Sriwijaya, secara langsung ataupun tidak langsung sebagai sumbangan pikiran dalam peningkatan mutu pembelajaran dan penelitian.

Waalaikumussalam Wr. Wb.

Palembang, Universitas Sriwijaya
Agustus, 2021
Tim Penulis

DAFTAR ISI

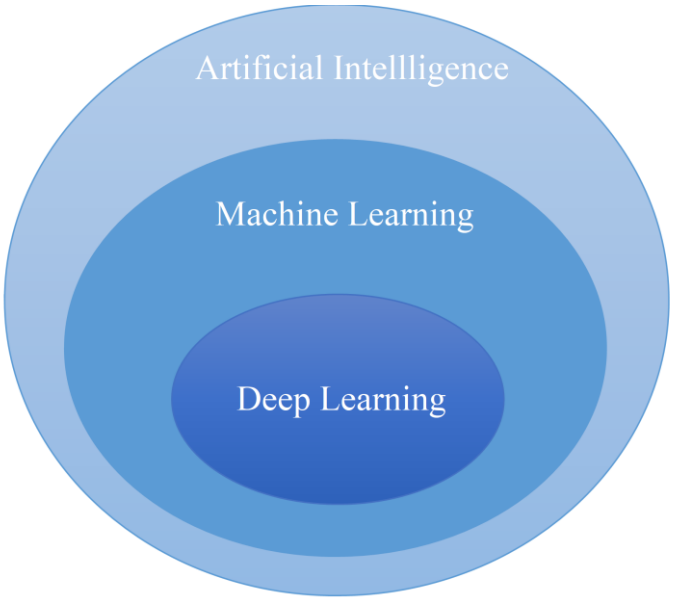
DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Umum

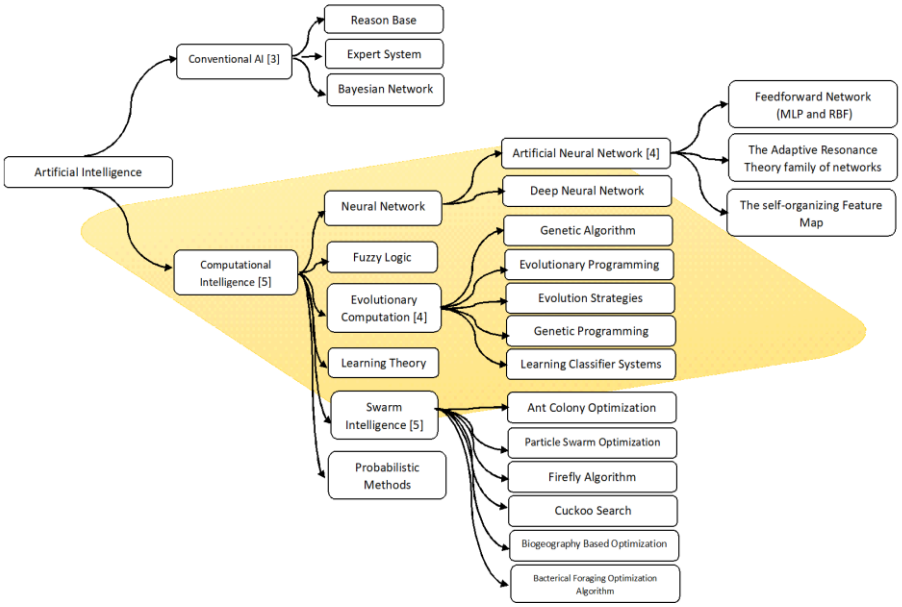
Bagian ini mendefinisikan dengan jelas apa yang terlintas ketika menyebutkan kecerdasan buatan (*Artificial Intelligence*, AI), pembelajaran mesin (*Machine Learning*, ML), atau pembelajaran mendalam (*Deep Learning*, DL). Apa itu kecerdasan buatan, pembelajaran mesin, dan pembelajaran mendalam? Bagaimana ketiga ilmu tersebut saling berhubungan? Gambar 1.1 berikut menjelaskan secara umum keterkaitan hubungan ketiganya.



Gambar 1.1. Hubungan *Artificial Intelligence*, *Machine Learning* dan *Deep Learning*

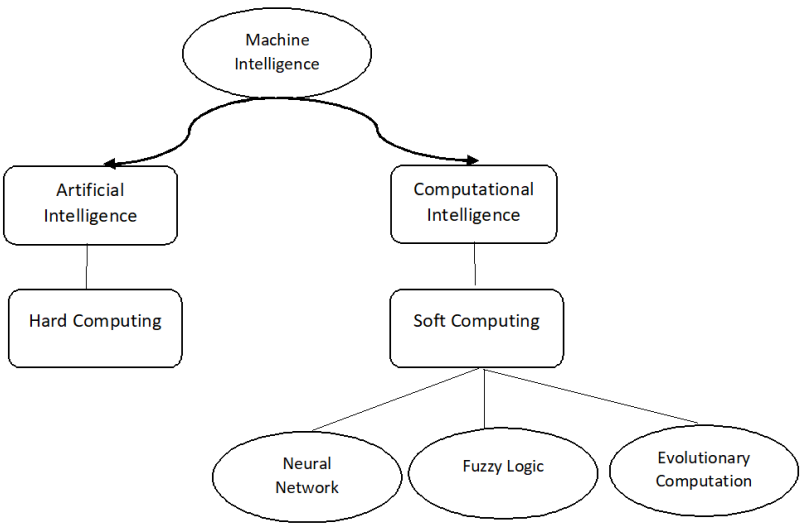
1.2 *Artificial Intelligence*

Kecerdasan buatan (*Artificial Intelligence*, AI) adalah sebuah disiplin ilmu untuk membuat mesin melakukan banyak hal yang memerlukan kecerdasan jika dilakukan oleh manusia (Minsky, 1961). Untuk mengetahui definisi sebenarnya dari AI, kita harus paham dulu: "Kapan program atau mesin dikatakan cerdas?" Usaha untuk menjawab pertanyaan tersebut telah dilakukan oleh seorang matematikawan dari Inggris, Alan Turing. Dia adalah inventor pengujian turing "Turing Test", yang dapat memutuskan mesin/program tersebut cerdas atau tidak (Rutkowski, 2008). Proses uji ini melibatkan seorang manusia A, yang menggunakan *keyboard* dan layar komputer, yang menanyakan pertanyaan yang sama pada komputer dan manusia B. Jika manusia A tidak bisa membedakan jawaban yang diberikan oleh komputer dari jawaban yang diberikan manusia B, maka bisa dikatakan bahwa komputer (program/mesin) itu cerdas (Rutkowski, 2008). Beberapa turunan algoritma AI bisa dilihat pada Gambar 1.2.



Gambar 1.2. Konsep Paham Pemikiran AI (Jin, 2016)(Jain, Tan, & Lim, 2008)(Siddique & Adeli, 2013)

Di sisi lain (Siddique & Adeli, 2013), Zadeh mengusulkan suatu pandangan berbeda dari kecerdasan mesin, dimana Ia membedakan teknik *hard computing* (HC) berdasarkan AI, sedangkan teknik *soft computing* (SC) berdasarkan CI (*Computational Intelligence*). Zadeh mendefiniskan SC sebagai penggabungan metode yang menyediakan perancangan sistem cerdas. Gambar 1.3 menunjukkan hubungan tersebut.



Gambar 1.3. Hubungan AI-HC dan CI-SC (Siddique & Adeli, 2013)

Teknologi yang berhubungan dengan *Computation Intelligence* yang dipakai dalam beberapa penelitian antara lain: *Pattern Recognition Image, Image Processing, Machine Learning, Computer Vision, Soft Computing, Evolutionary CompuNeural, Network Knowledge, Natural Language Processing, Image Retrieval, Decision Support System, Bioinformatics, Robotic Graph,* dan lain-lain.

AI lahir pada 1950-an, ketika beberapa perintis dari bidang baru ilmu komputer mulai bertanya apakah komputer dapat dibuat untuk "berpikir" —sebuah pertanyaan yang akibatnya masih kita jelajahi hingga saat ini. Definisi ringkas dari bidang itu adalah sebagai berikut: upaya untuk mengotomatisasi tugas-tugas intelektual yang biasanya dilakukan oleh manusia. Karena itu, AI adalah bidang umum yang mencakup pembelajaran mesin (*machine learning)* dan pembelajaran mendalam

(*deep learning*), tetapi itu juga mencakup banyak pendekatan lain yang tidak melibatkan pembelajaran apa pun. Pada awalnya sebuah program permainan catur hanya melibatkan aturan '*hard-coded*' yang dibuat oleh programmer, dan hal itu tidak memenuhi syarat sebagai *machine learning*.

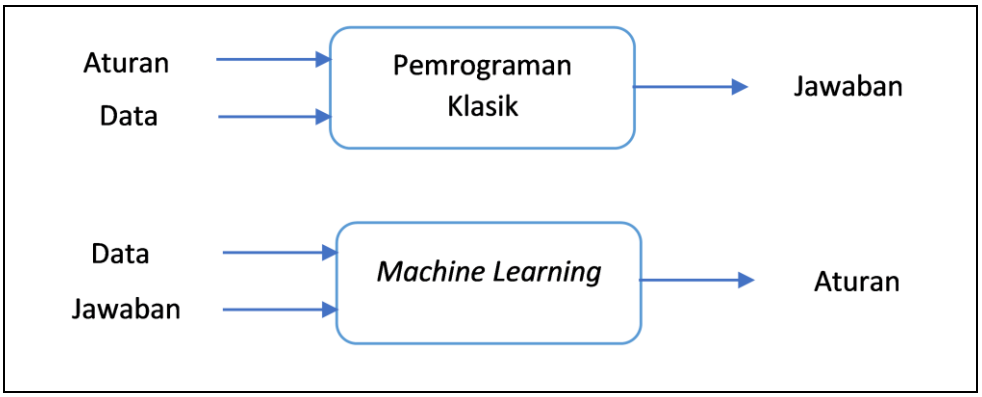
Untuk waktu yang cukup lama, banyak ahli percaya bahwa AI tingkat manusia dapat dicapai dengan membuat seperangkat aturan eksplisit yang cukup besar untuk memanipulasi pengetahuan oleh programer. Pendekatan ini dikenal sebagai AI simbolis, dan itu adalah paradigma dominan dalam bidang AI dari tahun 1950 hingga akhir 1980-an. Popularitasnya mencapai puncak selama sistem pakar *booming* tahun 1980-an. Meskipun pendekatan AI simbolis terbukti cocok untuk menyelesaikan masalah logis yang terdefinisi dengan baik, seperti bermain catur, ternyata sulit untuk menemukan aturan eksplisit untuk memecahkan masalah yang lebih kompleks dan tidak jelas, seperti klasifikasi gambar, pengenalan suara, dan penterjemah bahasa. Pendekatan baru muncul untuk mengambil tempat pendekatan AI simbolis yaitu *machine learning.*

1.3 *Machine Learning*

Pembelajaran mesin muncul dari pertanyaan: "Dapatkah komputer melampaui apa yang kita ketahui? Bagaimana cara memerintahkannya untuk melakukan suatu hal? Bagaimana komputer belajar sendiri untuk melakukan tugas yang ditentukan? Jika manusia membuat aturan

pemrosesan data secara manual, dapatkah komputer secara otomatis mempelajari aturan ini dengan hanya melihat data?".

Pertanyaan-pertanyaan ini membuka pintu ke paradigma pemrograman baru. Dalam pemrograman klasik, paradigma AI simbolis dimana aturan manusia dimasukkan aturan program komputer dan data yang akan diproses sesuai dengan aturan ini, sehingga mengeluarkan jawaban (lihat Gambar 1.4). Dengan *machine learning*, manusia memasukkan data serta jawaban, sehingga mengeluarkan aturan. Aturan-aturan ini kemudian dapat diterapkan pada data baru untuk menghasilkan jawaban yang sebenarnya.



Gambar 1.4. Paradigma *Machine Learning*

Dasarnya, sistem *machine learning* "dilatih", bukan diprogram secara eksplisit. Hal itu disajikan dengan banyak contoh yang relevan dengan tugas-tugas. Proses itu untuk menemukan struktur statistik dalam sampel yang pada akhirnya memungkinkan sistem untuk membuat aturan untuk mengotomatisasi tugas-tugas tersebut. Misalnya, jika ingin mengotomatisasi tugas untuk menandai suatu gambar, maka dapat

menyajikan sistem *machine learning* dengan banyak contoh gambar yang telah ditandai oleh manusia, dan sistem akan mempelajari aturan statistik untuk mengaitkan gambar dengan label tertentu.

Meskipun pendekatan *machine learning* baru mulai berkembang pada 1990-an, namun pendekatan ini dengan cepat menjadi sub bidang AI yang paling populer dan sukses. Tren juga seiring didorong oleh ketersediaan perangkat keras yang lebih cepat dan kumpulan data yang lebih besar. *Machine learning* terkait erat dengan statistik matematika, tetapi berbeda dari statistik dalam beberapa hal penting. Tidak seperti statistik, *machine learning* cenderung berurusan dengan dataset yang besar dan kompleks (seperti dataser jutaan gambar, dimana masing-masing terdiri dari puluhan ribu *pixel*), yang analisis statistik klasiknya berbeda dengan analisis Bayesian yang tidak praktis. Akibatnya, *machine learning*, dan terutama pembelajaran mendalam (*deep learning)*, menunjukkan teori matematika yang relatif sedikit — mungkin terlalu sedikit — dan berorientasi pada rekayasa. Gagasan disiplin ilmu ini langsung dibuktikan secara empiris lebih sering daripada secara teoritis.

1.4 *Deep Learning*

Untuk mendefinisikan pembelajaran dalam dan memahami perbedaan antara pembelajaran dalam (*deep learning*) dan pendekatan pembelajaran mesin lainnya, pertama-tama kita perlu beberapa gagasan tentang apa yang dilakukan algoritma pembelajaran mesin. Pernyataan sebelumnya menyatakan bahwa pembelajaran mesin menemukan aturan untuk menjalankan tugas dalam pemrosesan data, memberikan contoh apa yang

diharapkan. Jadi, untuk melakukan pembelajaran mesin, kita membutuhkan tiga hal:

1. Masukkan data — Misalnya, jika tugasnya adalah pengenalan suara, masukkan data ini bisa berupa file suara orang yang berbicara. Jika tugasnya adalah penandaan suatu gambar, maka masukkan data bisa berupa gambar.

2. Contoh keluaran yang diharapkan — Dalam tugas pengenalan suara, ini bisa berupa transkrip file suara buatan manusia. Dalam kasus gambar, *output* yang diharapkan bisa berupa "anjing," "kucing," dan sebagainya.

3. Cara untuk mengukur apakah algoritma melakukan pekerjaan dengan baik — Ini diperlukan untuk menentukan jarak antara keluaran algoritma saat ini dan keluaran yang diharapkan. Pengukuran digunakan sebagai sinyal umpan balik untuk menyesuaikan cara kerja algoritma. Langkah penyesuaian ini adalah apa yang kita sebut belajar.

Berdasarkan tiga hal di atas, maka sebuah model pembelajaran mesin mengubah data inputnya menjadi keluaran yang bermakna, proses yang “dipelajari” dari paparan ke contoh masukkan dan keluaran yang diketahui. Oleh karena itu, masalah utama dalam pembelajaran mesin dan pembelajaran mendalam adalah mengubah data secara bermakna; dengan kata lain, untuk mempelajari representasi berguna dari masukkan data yang ada — representasi yang membuat kita lebih dekat dengan keluaran yang diharapkan.

Deep learning adalah subbidang khusus dari pembelajaran mesin; pandangan baru tentang representasi pembelajaran dari data yang menekankan pada pembelajaran banyak lapisan dari representasi yang semakin bermakna. *Deep learning* merupakan representasi banyak layer yang berkontribusi pada model data yang disebut sebagai kedalaman model. Nama lain yang sesuai untuk bidang ini bisa saja merupakan representasi pembelajaran berlapis dan pembelajaran representasi hirarkis. *Deep learning* modern sering melibatkan puluhan atau bahkan ratusan lapisan representasi — dan mereka semua belajar secara otomatis dari paparan data pelatihan. Sementara itu, pendekatan lain untuk pembelajaran mesin cenderung fokus pada pembelajaran hanya satu atau dua lapisan representasi data; karenanya, mereka kadang-kadang disebut pembelajaran dangkal (*shallow*).

BAB 2
MACHINE LEARNING

Pembelajaran mesin, atau *machine learning* menjawab pertanyaan tentang bagaimana membangun komputer secara otomatis melalui pengalaman. Ini adalah salah satu bidang teknis yang paling cepat berkembang saat ini, terletak di persimpangan ilmu komputer dan statistik, dan pada inti kecerdasan buatan dan ilmu data. Kemajuan terbaru dalam pembelajaran mesin telah didorong baik oleh pengembangan algoritma dan teori pembelajaran baru berkelanjutan dalam ketersediaan data *online* dengan perhitungan biaya rendah. Adopsi metode pembelajaran mesin dapat ditemukan di seluruh ilmu sains, teknologi, dan perdagangan yang mengarah pada pengambilan keputusan berbasis bukti di banyak kalangan, termasuk perawatan kesehatan, manufaktur, pendidikan, pemodelan keuangan, kepolisian, dan pemasaran.

2.1 Definisi Umum

Pembelajaran mesin adalah disiplin yang difokuskan pada dua pertanyaan yang saling terkait: (i) Bagaimana seseorang dapat membangun sistem komputer yang secara otomatis melalui pengalaman?, (ii) Apa hukum teori dasar statistik informasi yang mendasar itu mengatur semua sistem pembelajaran, termasuk komputer, manusia, dan

organisasi?" (Jordan & Mitchell, 2015). Studi tentang pembelajaran mesin adalah penting baik untuk menjawab pertanyaan-pertanyaan dasar ilmiah dan membuat perangkat lunak komputer praktis yang mampu diproduksi dan diimplementasikan ke banyak aplikasi.

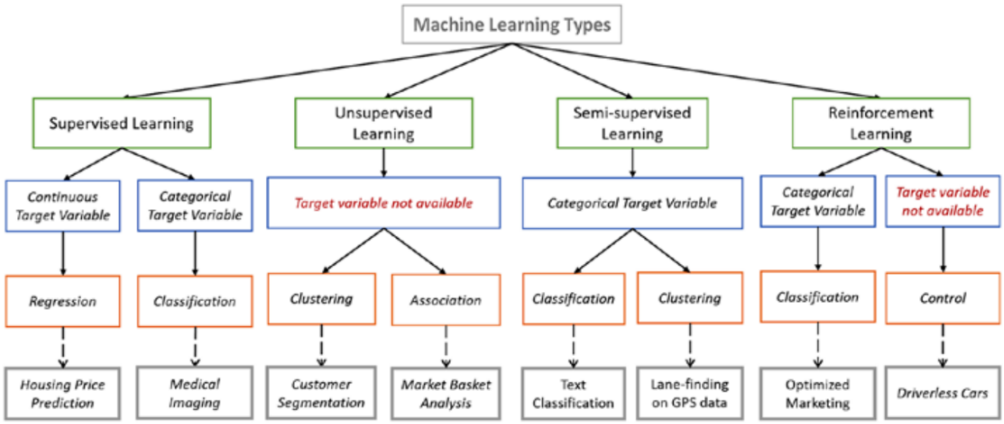
Machine Learning (pembelajaran mesin) sebagai satu set metode yang dapat secara otomatis mendeteksi pola dalam data, dan kemudian menemukan pola untuk memprediksi data masa depan, atau untuk melakukan jenis pengambilan keputusan lainnya di bawah ketidakpastian (Robert, 2014).

2.2 Tipe Pembelajaran *Machine Learning*

Berbagai algoritma pembelajaran mesin telah dikembangkan untuk mencakup beragam data dan jenis masalah yang dipamerkan di berbagai masalah pembelajaran mesin (Hastie, Tibshirani, Friedman, & Franklin, 2005)(John Lu, 2010). Secara konseptual, algoritma pembelajaran mesin dapat dipandu oleh pengalaman pelatihan, untuk menemukan program yang mengoptimalkan metrik kinerja. Ada beberapa variasi cara untuk menentukan jenis algoritma *Machine Learning*, tetapi umumnya mereka dapat dibagi ke dalam kategori sesuai dengan tujuannya dan kategori utama adalah sebagai berikut (Gambar 2.1) (Goldstein, Navar, & Carter, 2016)(Li, Rajagopalan, & Clifford, 2014):

1. *Supervised learning*
2. *Unsupervised Learning*
3. *Semi-supervised Learning*

4. *Reinforcement Learning*



Gambar 2.1. Tipe-tipe Pembelajaran dalam *Machine Learning* (Goldstein et al., 2016)(Li et al., 2014)

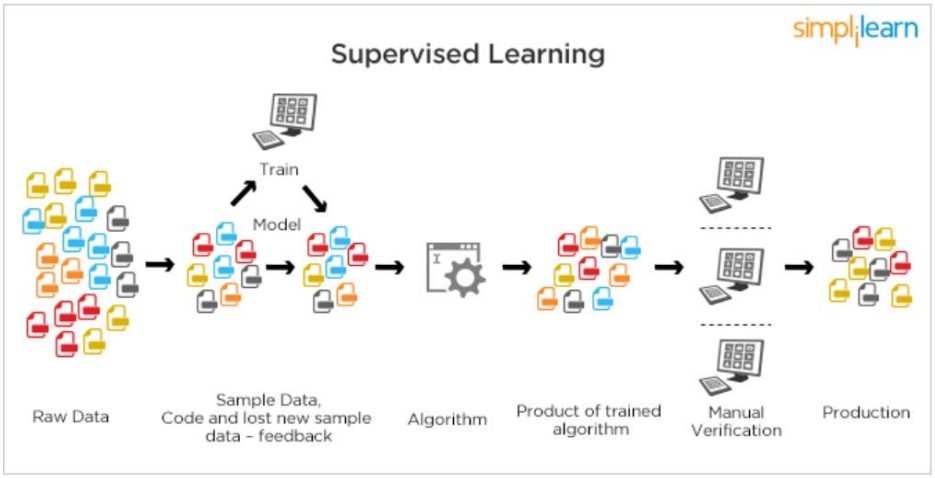
2.2.1 *Supervised Learning* (Pembelajaran Terarah)

Supervised learning merupakan metode yang paling populer dalam implementasi algoritma untuk *Machine Learning*. Dimana mesin dilatih untuk memberikan keluaran yang sudah ditetapkan atau diharapkan sebelumnya. Algoritma yang diterapkan pada mesin tersebut ditujukan untuk membuat suatu masukkan menjadi keluaran yang diharapkan. *Supervised learning* mencoba untuk memodelkan hubungan dan ketergantungan antara keluaran prediksi target dan fitur masukkan, sehingga kita dapat memprediksi nilai keluaran untuk data baru berdasarkan hubungan yang dipelajari dari kumpulan data sebelumnya (Goldstein et al., 2016).

Pada umunya, algoritma *Supervised learning* digunakan untuk (Gambar 2.2):

a. Algoritma klasifikasi (*classification)*: algoritma ini membuat model prediktif dari data pelatihan yang memiliki fitur dan label kelas. Model prediksi ini secara bergiliran menggunakan fitur yang dipelajari dari data pelatihan pada data baru, yang sebelumnya tidak terlihat untuk memprediksi label kelas mereka. Kelas *output* bersifat diskrit. Jenis-jenis algoritma klasifikasi termasuk *decision trees, random forests, support vector machines,* dan banyak lagi.

b. Algoritma regresi (*regression)*: algoritma ini digunakan untuk memprediksi nilai *output* berdasarkan beberapa fitur masukan yang diperoleh dari data. Untuk melakukan ini, algoritma membangun model berdasarkan fitur dan nilai *output* dari data pelatihan dan model ini digunakan untuk memprediksi nilai-nilai untuk data baru. Nilai *output* dalam hal ini bersifat kontinyu dan tidak diskrit. Jenis algoritma regresi termasuk *linear regression, multivariate regression, regression trees, and lasso regression,* dan banyak lainnya.



Gambar 2.2. Model Sistematis *Supervised Learning* (Hansen, Henriksen, Bach, & Matthiesen, 2017). Suatu model disiapkan melalui suatu proses pelatihan yang diperlukan untuk membuat prediksi dan dikoreksi ketika prediksi tersebut salah. Proses pelatihan berlanjut sampai model mencapai tingkat akurasi yang diinginkan pada data pelatihan.

Implementasi dari *Supervised learning* digunakan untuk *speech recognition* (pengenalan suara), *credit scoring* (penilaian kredit), *medical imaging* (pencitraan medis), dan *search engines* (mesin pencarian).

2.2.2 *Unsupervised Learning* (Pembelajaran Tak Terarah)

Unsupervised learning adalah keluarga algoritma pembelajaran mesin yang digunakan dalam deteksi pola dan pemodelan deskriptif. Algoritma ini mencoba menggunakan teknik pada data *input* untuk menambang aturan, mendeteksi pola, meringkas dan mengelompokkan titik-titik data yang membantu dalam menurunkan wawasan yang bermakna dan

mendeskripsikan data lebih baik kepada pengguna (Goldstein et al., 2016).

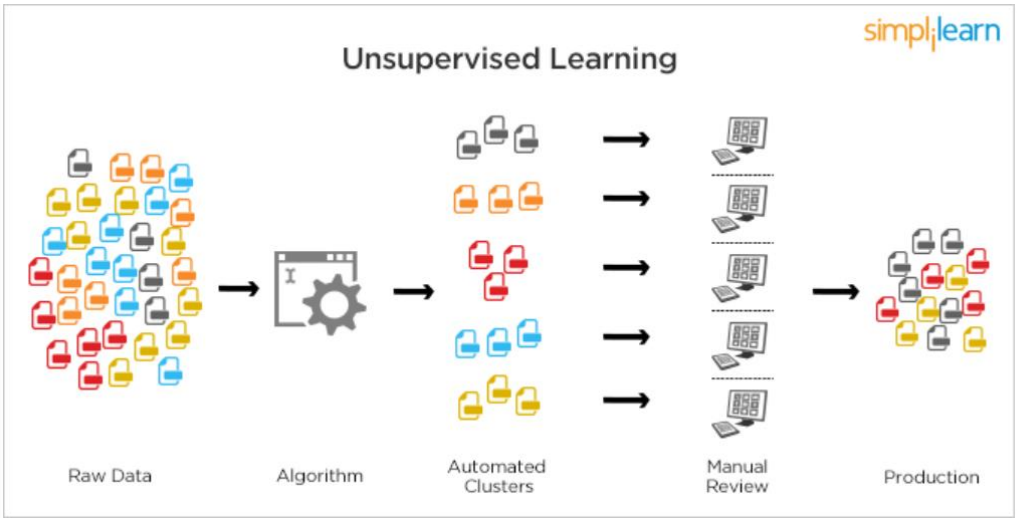
Di *unsupervised learning*, kita tidak mengajari mesin untuk menghasilkan suatu *output* tertentu. Kita hanya mengajarinya seperti apa *input* yang benar, dan *output* nya mereka sendiri yang akan menentukan, kita tidak memiliki ide sama sekali *output* seperti apa yang dihasilkan.

Pada umunya, algoritma *unsupervised learning* digunakan untuk (Gambar 2.3) (Li et al., 2014):

a. Algoritma pengelompokkan (*clustering)*: Tujuan utama dari algoritma ini adalah mengelompokkan atau memasukkan titik data masukan ke dalam kelas atau kategori yang berbeda hanya dengan menggunakan fitur yang berasal dari data masukan saja dan tidak ada informasi eksternal lainnya. Tidak seperti klasifikasi, label *output* tidak dikenal sebelumnya dalam pengelompokan. Beberapa algoritma pengelompokan populer termasuk *k-means, k-medoids,* dan *hierarchical clustering*.

b. Algoritma pembelajaran aturan asosiasi (*association rule learning*): Algoritma ini digunakan untuk menggali dan mengekstrak aturan dan pola dari kumpulan data. Aturan-aturan ini menjelaskan hubungan antara variabel dan atribut yang berbeda, dan juga menggambarkan set dan pola item yang sering terjadi dalam data. Algoritma populer termasuk *Apriori* dan *FP Growth.*

Implementasi dari *unsupervised learning* digunakan untuk segmentasi pelanggan dalam pemasaran, analisis jejaring sosial, segmentasi citra, klimatologi, dan banyak lagi.

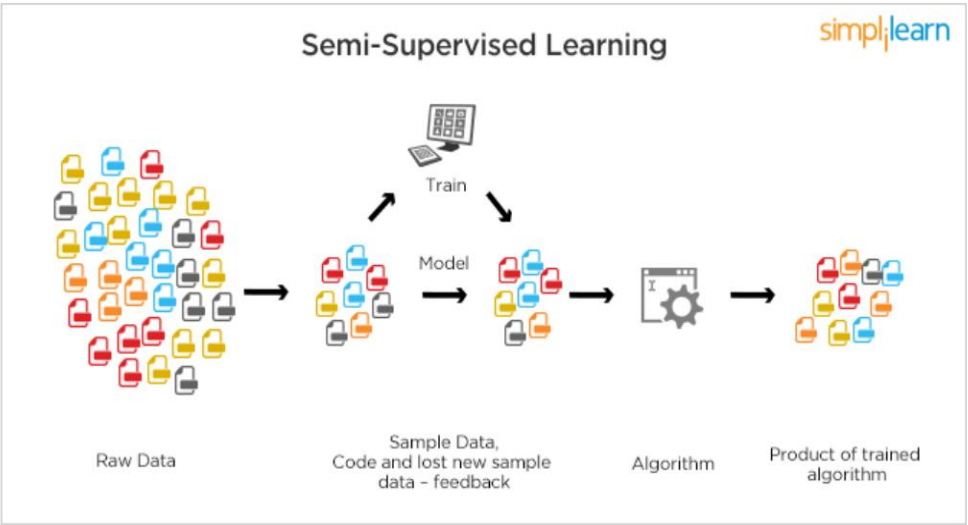


Gambar 2.3. Model Sistematis *Unsupervised Learning*
(Hansen et al., 2017)

2.2.3 *Semi-supervised Learning* (Pembelajaran Semi-Terarah)

Dalam dua tipe sebelumnya, tidak ada label untuk semua observasi dalam dataset atau label yang ada untuk semua pengamatan. *Semi-supervised learning* berada di antara keduanya. Dalam banyak situasi praktis, biaya untuk label cukup tinggi, karena membutuhkan ahli yang terampil untuk melakukan itu. Jadi, dengan tidak adanya label di sebagian besar pengamatan tetapi hadir di beberapa, algoritma *semi-supervised learning* adalah kandidat terbaik untuk membangun model. Metode-metode ini mengeksploitasi gagasan bahwa meskipun keanggotaan grup dari data tidak berlabel tidak diketahui, data ini membawa informasi penting tentang parameter grup (Goldstein et al., 2016).

Jadi, *semi-supervised learning* adalah bagian dari *supervised* dan *unsupervised learning* yang menggunakan data yang berlabel dan tidak berlabel untuk pelatihan. Dalam skenario yang khas, algoritma ini akan menggunakan sejumlah kecil data berlabel dengan sejumlah besar data tanpa label (Gambar 2.4). Implementasi dari *semi-supervised learning* digunakan untuk pengenalan wajah dan suara.



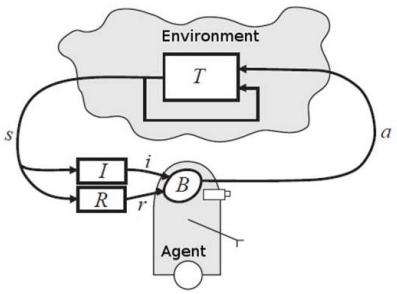
Gambar 2.4. Model Sistematis *Semi-Supervised Learning* (Hansen et al., 2017)

2.2.4 *Reinforcement Learning*

Reinforcement Learning adalah sejenis *Machine Learning*, yang memungkinkan mesin dan agen perangkat lunak untuk secara otomatis menentukan perilaku ideal dalam konteks tertentu, untuk memaksimalkan kinerjanya. “Bagaimana membuat suatu mesin itu dapat

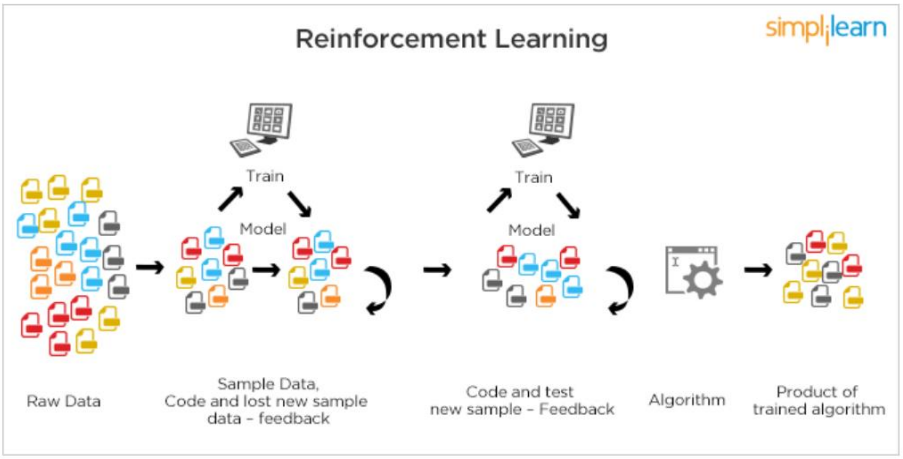
menjadi pintar setelah berinteraksi dengan lingkungan?". Pertanyaan itulah yang menjadi dasar dikembangkannya *Reinforcement Learning*.

Tiga komponen utama dapat diidentifikasi dalam fungsi *Reinforcement Learning*: agen, lingkungan, dan tindakan. Agen adalah pelajar atau pembuat keputusan, lingkungan mencakup segala sesuatu yang berinteraksi dengan agen, dan tindakannya adalah apa yang dapat dilakukan oleh agen (Gambar 2.5 dan 2.6).



Gambar 2.5. Interpretasi Permodelan *Reinforcement Learning* (Dey, 2016)(Littman, Moore, & others, 1996)

Algoritma yang sering dipakai pada metode *Q-Learning, Temporal Difference (TD),* dan *Deep Adversarial Networks.* Implementasi penggunaan *Reinforcement Learning pada* contoh kasus permainan papan yang dimainkan komputer (*Chess, Go*), tangan robotik, dan mobil *self-driving* (Goldstein et al., 2016).



Gambar 2.6. Model Sistematis *Reinforcement Learning* (Hansen et al., 2017)

BAB 3
DEEP LEARNING

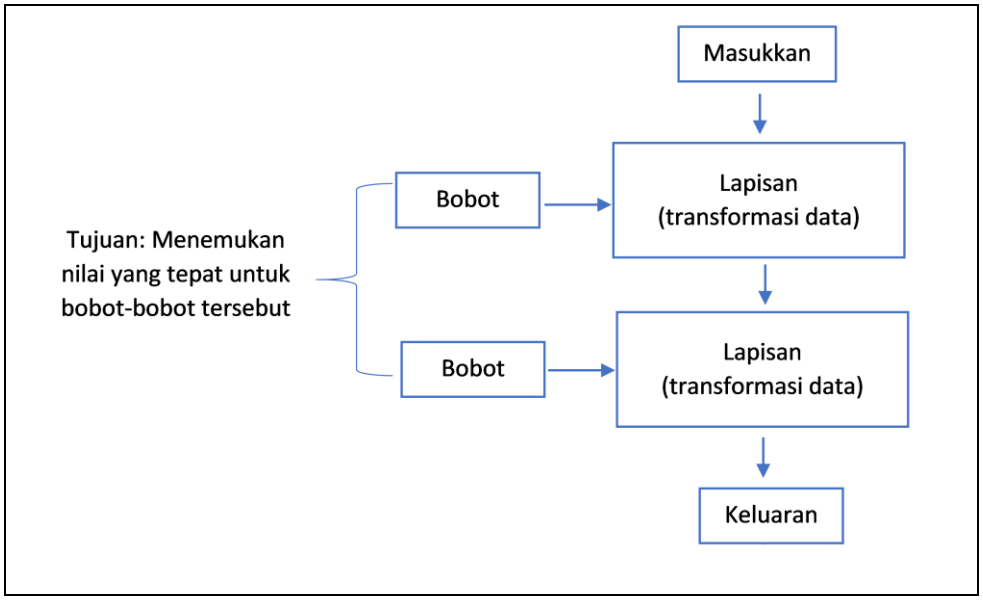
Bagian ini akan diperlihatkan bahwa pembelajaran mesin adalah tentang memetakan masukkan data (seperti gambar kucing) ke target (dengan label "cat" atau “kucing”), yang dilakukan dengan mengamati banyak contoh masukkan dan target. Selain itu akan diperlihatkan bahwa pembelajaran dalam (*deep learning*) melakukan pemetaan masukkan ke target melalui serangkaian transformasi (lapisan) data sederhana dimana transformasi data ini dipelajari melalui paparan contoh. Sekarang mari kita lihat bagaimana pembelajaran ini terjadi, secara konkret.

3.1 Bagaimana *Deep Learning* Berkerja?

Deep learning merupakan bagian dari metode *machine learning* yang berbasis jaringan syaraf tiruan. Metode ini digunakan untuk menangani kumpulan data besar dengan menggunakan algoritma *backpropagation* untuk menunjukkan bagaimana mesin harus mengubah parameter internal yang digunakan untuk menghitung representasi di setiap lapisan dari representasi di lapisan sebelumnya (LeCun, Bengio, & Hinton, 2015). *Deep learning* digunakan sebagai pendekatan pembelajaran mesin untuk memecahkan deteksi objek, klasifikasi gambar dan segmentasi semantic (Attia, Hossny, Nahavandi, & Yazdabadi, 2017).

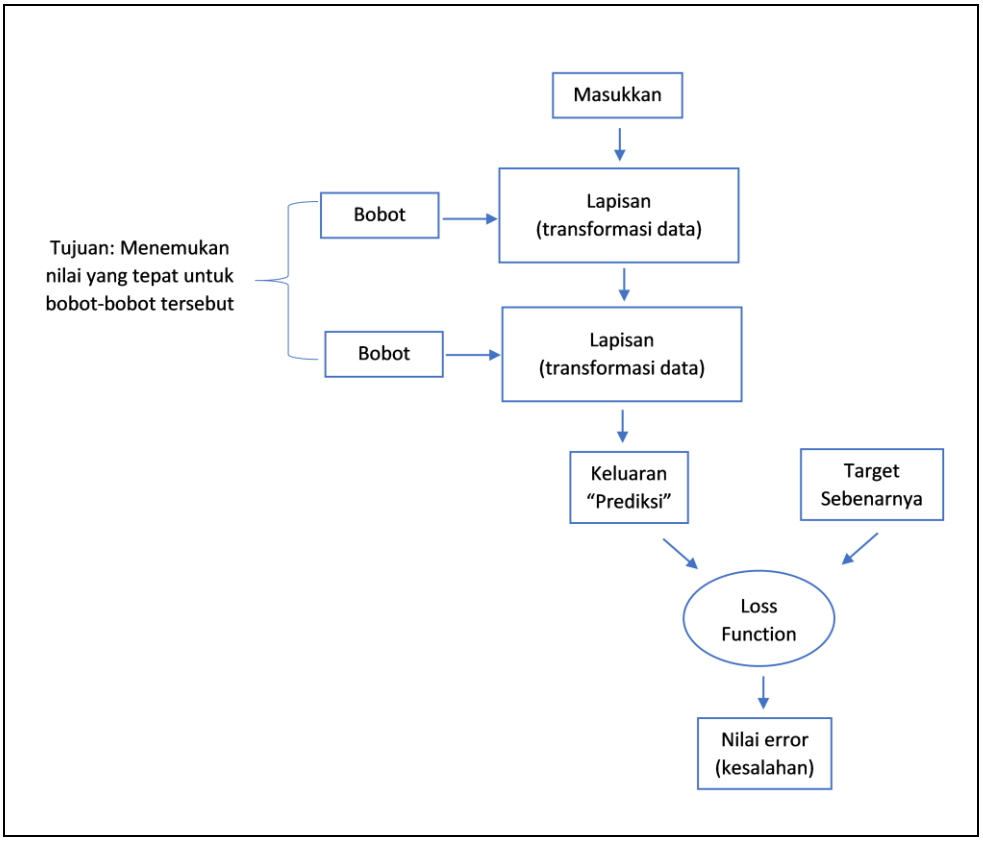
Deep learning memungkinkan model komputasi yang terdiri dari beberapa lapisan pemrosesan untuk mempelajari representasi data dengan berbagai tingkat abstraksi. Metode-metode ini telah secara dramatis meningkatkan *state-of-the-art* dalam pengenalan suara, pengenalan objek visual, deteksi objek dan banyak domain lain seperti penemuan obat dan genomik. *Deep learning* menemukan struktur rumit dalam kumpulan data besar dengan menggunakan algoritma *backpropagation* untuk menunjukkan bagaimana mesin harus mengubah parameter internal yang digunakan untuk menghitung representasi di setiap lapisan dari representasi di lapisan sebelumnya. Jaringan konvolusional yang dalam telah menghasilkan terobosan dalam memproses gambar, video, ucapan dan audio, sedangkan jaringan berulang telah menyoroti data sekuensial seperti teks dan ucapan.

Spesifikasi apa yang dilakukan lapisan (*layer*) terhadap data inputnya disimpan dalam bobot *layer*, yang intinya adalah sekelompok angka. Dalam istilah teknis, dikatakan bahwa transformasi yang diterapkan oleh *layer* adalah parameter berdasarkan bobotnya (Gambar 3.1). Bobot kadang-kadang juga disebut sebagai parameter lapisan. Dalam konteks ini, pembelajaran berarti menemukan seperangkat nilai untuk bobot semua lapisan dalam jaringan, sehingga jaringan akan memetakan *input* contoh dengan benar ke target terkait. Tapi ini masalahnya: jaringan saraf yang dalam dapat berisi puluhan juta parameter. Menemukan nilai yang benar untuk semuanya mungkin tampak seperti tugas yang menakutkan, terutama mengingat bahwa memodifikasi nilai dari satu parameter akan memengaruhi perilaku semua parameter lainnya.



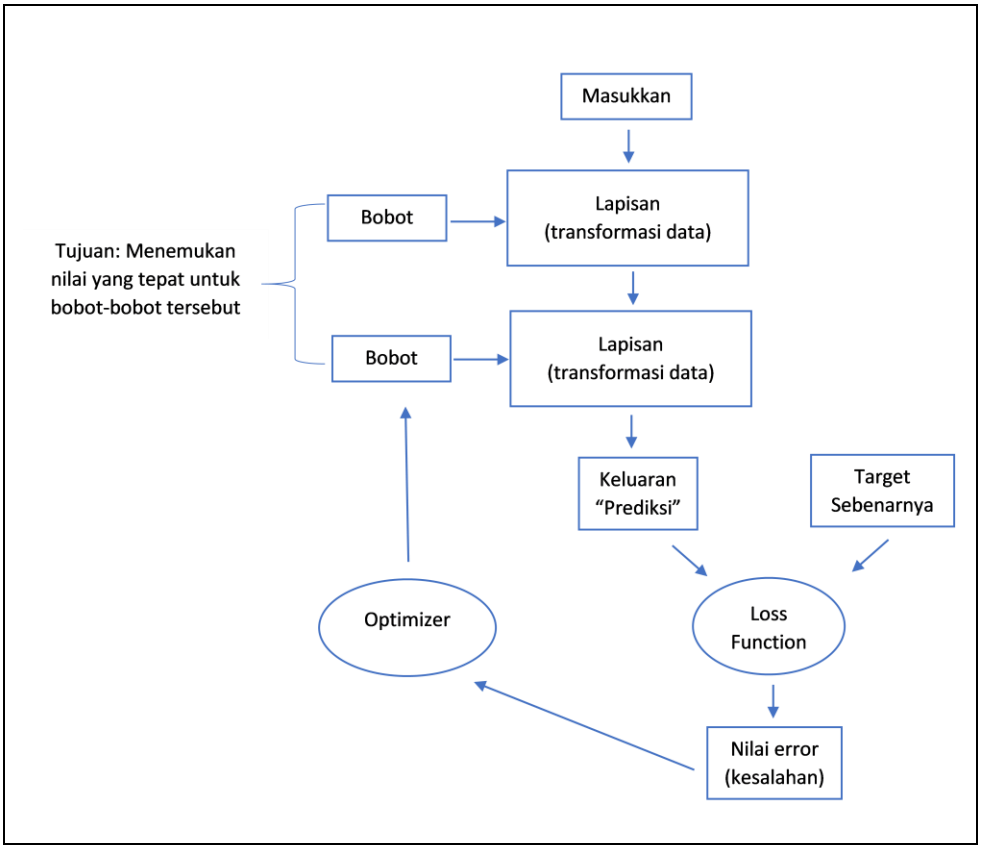
Gambar 3.1. Jaringan Saraf dengan bobot sebagai parameter

Untuk mengontrol keluaran dari jaringan saraf, anda harus dapat mengukur seberapa jauh keluaran dihasilkan dari apa yang diharapkan. Ini adalah pekerjaan dari *loss function*, dimana membandingkan hasil prediksi jaringan dengan target sebenarnya dan menghitung perbedaannya. Hal ini digunakan untuk menentukan seberapa baik jaringan telah dilakukan pada contoh khusus ini (Gambar 3.2).



Gambar 3.2. *Loss function* yang digunakan untuk mengukur kinerja jaringan

Trik mendasar dalam *deep learning* adalah menggunakan nilai *error* (kesalahan) sebagai sinyal umpan balik untuk menyesuaikan nilai bobot, ke arah yang akan menurunkan *error* (Gambar 3.3). Penyesuaian ini adalah tugas pengoptimal (*optimizer*), yang mengimplementasikan apa yang disebut algoritma *Backpropagation*: algoritma sentral dalam *deep learning*. Bab selanjutnya menjelaskan secara lebih rinci cara kerja *backpropagation.*



Gambar 3.3. *Loss function* digunakan sebagai sinyal umpan balik untuk menyesuaikan bobot

Awalnya, bobot jaringan diberi nilai acak, sehingga jaringan hanya mengimplementasikan serangkaian transformasi acak. Tentu saja, keluaran nya jauh dari yang seharusnya, dan nilai *error* nya sangat tinggi. Tetapi dengan setiap contoh proses jaringan, bobot disesuaikan sedikit ke arah yang benar, dan nilai *error* menurun. Ini adalah pengulangan pelatihan (*looping)* yang diulangi dalam jumlah yang cukup banyak (biasanya puluhan iterasi lebih dari ribuan contoh), menghasilkan nilai bobot yang meminimalkan *loss function*. Jaringan dengan nilai *error*

yang minimal adalah jaringan yang keluarannya sedekat mungkin dengan target dimana jaringan terlatih.

3.2 Mengapa *Deep Learning*?

Dua ide kunci *deep learning* untuk komputer vision — *Convolutional Neural Network* dan jaringan *backpropagation* — telah dipahami dengan baik pada tahun 1989. Algoritma memori jangka pendek-panjang dengan menggunakan *Long Short-Term Memory* (LSTM), yang merupakan metode *deep learning* yang berhubungan dengan anotasi waktu bahkan dikembangkan pada tahun 1997, dan konsep ilmu nya hampir tidak berubah sejak itu. Jadi mengapa *deep learning* baru berjalan setelah tahun 2012? Apa yang berubah dalam dua dekade ini?

Secara umum, tiga kekuatan teknis mendorong kemajuan dalam *machine learning*; (i) perangkat keras, (ii) kumpulan data (dataset); dan (iii) algoritma yang maju (*advance*). Karena bidang ini dipandu oleh temuan eksperimental daripada bersifat teori, kemajuan algoritma hanya menjadi mungkin ketika data dan perangkat keras yang sesuai tersedia untuk mencoba ide-ide baru (atau meningkatkan ide-ide lama, seperti yang sering terjadi). Hambatan nyata sepanjang 1990-2000, adalah data dan perangkat keras. Tapi saat ini hal tersebut tidak lagi menjadi masalah karena telah berkembangnya internet cepat, *chip* grafis berkinerja tinggi dikembangkan untuk kebutuhan pasar *game*.

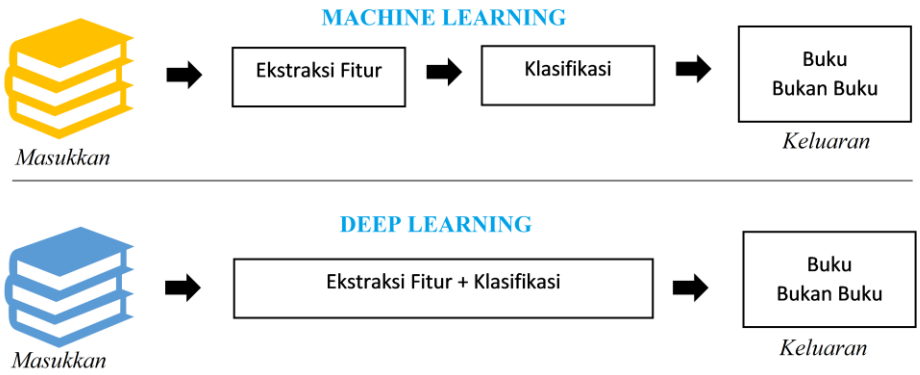
Deep learning memiliki beberapa sifat yang membenarkan statusnya sebagai revolusi AI. Kita mungkin tidak menggunakan jaringan saraf buatan dua dekade dari sekarang, tetapi apa pun yang kita gunakan akan secara langsung mewarisi sifat dari *deep learning* yang modern beserta konsep intinya. Properti penting ini dapat diurutkan menjadi tiga kategori:

- Kesederhanaan — *Deep learning* menghilangkan kebutuhan akan rekayasa fitur, mengganti jaringan yang rumit, rapuh, dan teknik berat dengan model sederhana.

- Skalabilitas — *Deep learning* sangat memungkinkan untuk paralelisasi pada *Graphics Processing Unit* (GPU) atau *Tensor Processing Unit* (TPU). Selain itu, model *deep learning* dilatih oleh iterasi lebih dari sejumlah kecil data, yang memungkinkan mereka untuk dilatih pada kumpulan data yang besar.

- Fleksibilitas dan Dapat Digunakan Kembali — tidak seperti banyak pendekatan *machine learning* sebelumnya, model *deep learning* dapat dilatih tentang data tambahan tanpa memulai kembali dari awal, menjadikannya layak untuk pembelajaran online berkelanjutan — properti penting untuk model produksi yang sangat besar. Selain itu, model *deep learning* yang terlatih dapat digunakan ulang dan karenanya dapat digunakan kembali; misalnya, dimungkinkan untuk mengambil model *deep learning* yang dilatih untuk klasifikasi gambar.

Deep Learning membutuhkan mesin-mesin canggih yang berbeda dengan algoritma *Machine Learning* tradisional. GPU telah menjadi bagian yang tidak terpisahkan sekarang untuk menjalankan setiap algoritma *Deep Learning*. Dalam teknik *Machine Learning* tradisional, sebagian besar fitur yang diterapkan perlu diidentifikasi oleh pakar domain untuk mengurangi kompleksitas data dan membuat pola lebih terlihat agar algoritma pembelajaran berfungsi. Keuntungan terbesar dari algoritma *Deep Learning* adalah bahwa mereka mencoba mempelajari fitur tingkat tinggi dari data secara bertahap (Gambar 3.4).



Gambar 3.4. *Deep Learning* dan *Machine Learning*

Kapan sebaiknya menggunakan *Deep Learning*? *Deep Learning* mengungguli teknik lain jika ukuran datanya besar. Tetapi dengan ukuran data yang kecil, algoritma *Machine Learning* tradisional lebih disukai. Teknik *Deep Learning* perlu memiliki infrastruktur canggih untuk dilatih dalam waktu yang wajar. Ketika ada kurangnya pemahaman domain untuk introspeksi fitur, teknik *Deep Learning* mengalahkan metode lain

karena tidak perlu terlalu khawatir tentang rekayasa fitur. *Deep Learning* benar-benar bisa diandalkan untuk memecahkan masalah-masalah kompleks seperti klasifikasi gambar, pemrosesan bahasa alami, dan pengenalan suara.

BAB 4
METODE DEEP LEARNING

Dalam beberapa tahun terakhir telah banyak dilakukan penelitian-penelitian menggunakan metode *deep learning*. Metode ini memiliki arsitektur pembelajaran mesin yang digunakan untuk menangani kumpulan data besar. Beberapa metode yang menerapkan *deep learning* antara lain *Autoencoder-Deep Neural Network (AE-DNN), Convolutional Neural Networks* (CNN), dan *Recurrent Neural Network* (RNN) (LeCun et al., 2015).

4.1 *Autoencoder-Deep Neural Network*

4.1.1 *Autoencoder*

Autoencoder merupakan salah satu variasi dari model *neural network* yang belajar dengan cara *unsupervised* (tidak terarah) dengan cara propagasi maju dan propagasi balik yang sama seperti *neural network* umumnya. Konsep autoencoder pertama kali diperkenalkan oleh Rumelhart et al. pada tahun 1986, dan dipopulerkan oleh Hinton et al. (Rumelhart, Hinton, & Williams, 1985), Bengio et al. (Bengio & Lecun, 2007), Erhan et al. (Erhan, Courville, & Vincent, 2010), dan Baldi (Baldi, 2012). Menurut Hinton (Hinton & Salakhutdinov, 2006) *autoencoder* adalah sebuah gabungan 2 *multilayer network encoder* dan

decoder yang mengubah data dimensi tinggi menjadi sebuah kode dimensi rendah dan mengembalikan data dari kode tersebut. *Autoencoder* bertujuan mengurangi dimensi dari sebuah data dengan mengidentifikasi serta menyeleksi fitur fitur yang relevan di dalam dataset kemudian direkonstruksi kembali (Abid, Fatih, & James, 2019). Representasi dengan dimensi yang lebih kecil dapat mengurangi konsumsi memori dan mempercepat proses pelatihan untuk tahap pembelajaran mesin selanjutnya seperti klasifikasi atau klasterisasi. Autoencoder dinilai mampu merekontruksi data yang jauh lebih baik sebagai algoritma reduksi dimensi dari *Principal Component Analysis* (PCA) dan *local linear embedding* (Hinton & Salakhutdinov, 2006).

Pada bentuk yang paling sederhana dari autoencoder terdiri dari *layer* masukan, 1 lapisan tersembunyi yang menjadi bagian *encoder* dan *layer* keluaran yang menjadi bagian *decoder*. Dengan jumlah *neuron* yang ada pada *layer* keluaran sama dengan jumlah *neuron* pada *layer* masukan dengan tujuan untuk merekonstruksi data masukan bukan seperti neural network pada umumnya yang memprediksi nilai target Y untuk masukkan. *Autoencoder* dilatih seperti model *neural network* pada umumnya dengan melakukan propagasi maju, hitung nilai *error* antara data rekonstruksi dengan data aslinya, kemudian dioptimisasi menggunakan gradient descent dan dilakukan penyesuaian bobot.

Pada *autoencoder* sederhana, bagian *encoder* mengambil data inputan x dan dipetakan menjadi data z dengan persamaan matematisnya (Courville, Bengio, & Goodfellow, 2016):

$$z = \sigma(Wx + b) \tag{4.1}$$

Dimana z merupakan representasi keluaran *encoder*, dengan x merupakan data masukan, σ merupakan fungsi aktivasi bagian *encoder*, W merupakan vektor bobot bagian *encoder*, dan b merupakan vektor bias bagian *encoder*. Kemudian pada bagian *decoder* memetakan data z untuk direkonstruksi kembali menjadi x' dengan dimensi yang sama seperti x yang ditunjukkan pada persamaan matematis (Courville et al., 2016):

$$x' = \sigma'(W'z + b') \tag{4.2}$$

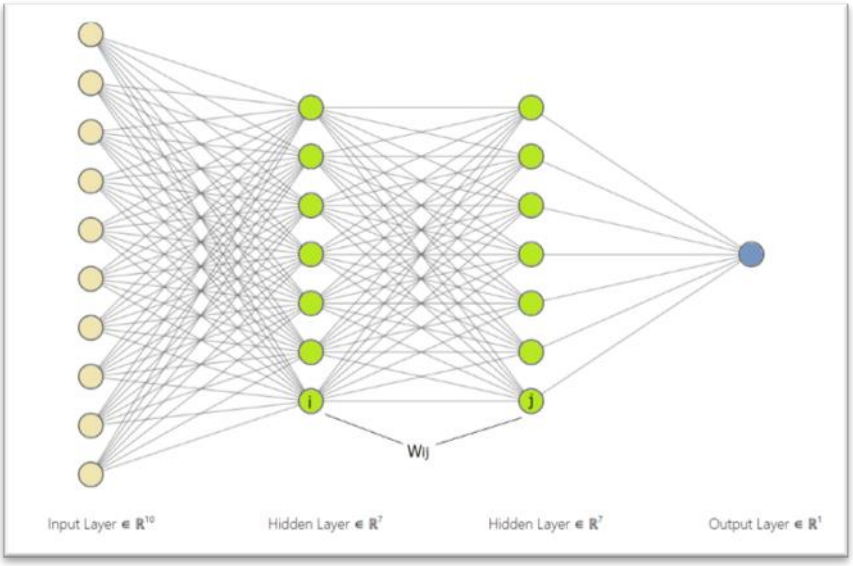
Dimana x' merupakan hasil rekonstruksi yang memiliki dimensi yang sama dengan x, σ' merupakan aktivasi fungsi di bagian *decoder*, W' merupakan vektor bobot *decoder*, dan b' merupakan vektor bias *decoder*.

Kemudian *autoencoder* akan dilatih dan dilakukan perhitungan *error* rekonstruksi menggunakan *loss function*, dengan contoh menggunakan *mean squared error* (Courville et al., 2016) :

$$L(x,x') = \|x - x'\|^2 = \left\|x - \sigma'(W'\big(\sigma(Wx + b)\big) + b'\right\|^2 \tag{4.3}$$

4.1.2 *Deep Neural Network*

Deep Neural Network (DNN) adalah arsitektur *neural network* yang memiliki layer tersembunyi yang lebih dalam atau banyak dari *Multilayer Perceptron* (MLP) dengan kedalaman lebih dari 2 lapisan tersembunyi (Giovanna Sannino & De Pietro, 2018), yang ditunjukkan pada Gambar 4.1. DNN memiliki kemampuan pemetaan kompleksitas data serta ekstraksi fitur data yang lebih baik dikarenakan memiliki arsitektur yang lebih mendalam (Bengio & Delalleau, 2011). Dengan kemampuan tersebut, DNN meraih kesuksesan di bidang pengenalan pola beberapa tahun terakhir (G. Sannino & De Pietro, 2018).



Gambar 4.1. Arsitektur *Deep Neural Network* dengan satu lapisan *input*, dua lapisan tersembunyi dan satu hasil *output* (Amato et al., 2013)

Dengan persamaan matematis DNN sebagai berikut (Patterson & Gibson, 2017):

1. Pertama, melakukan propagasi maju dari layer masukan i ke layer tersembunyi j

$$input\_sum_i = X_i \cdot W_{ij} + b$$

$$a_j = f(input_{sum_i}) \quad (4.4)$$

Dimana X_i merupakan vektor data masukan, W_{ij} merupakan vektor bobot penghubung antara *layer* masukan i ke *layer* tersembunyi j, b merupakan bias yang terdapat pada *layer* tersembunyi, $input\_sum_i$ merupakan vektor nilai terbobot, f merupakan aktivasi fungsi, dan a_j merupakan vektor nilai yang teraktivasi pada *neuron layer* tersembunyi j.

2. Melakukan propagasi maju dari *layer* tersembunyi j ke *layer* tersembunyi selanjutnya k.

$$input\_sum_j = a_j \cdot W_{jk} + b$$

$$a_k = f(input\_sum_j) \quad (4.5)$$

Dimana a_j merupakan vektor data masukan dari *layer* tersembunyi sebelumnya, W_{jk} merupakan vektor bobot penghubung antara *layer* tersembunyi j ke *layer* tersembunyi k, b merupakan bias yang terdapat pada *layer* tersembunyi, $input\_sum_j$ merupakan vektor nilai terbobot, f

merupakan aktivasi fungsi, dan a_k merupakan vektor nilai yang teraktivasi pada *neuron layer* tersembunyi k.

3. Kemudian mempropagasi maju dari *layer* tersembunyi k ke *layer* keluaran.

$$input\_sum_k = (a_k \cdot W_{kl} + b)$$

$$Y_l = f(input\_sum_k) \quad (4.6)$$

dimana Y_l merupakan vektor nilai keluaran yang telah teraktivasi, f merupakan fungsi aktivasi pada keluaran, $input\_sum_k$ merupakan vektor nilai terbobot pada *layer* keluaran, W_{kl} merupakan vektor bobot penghubung *layer* tersembunyi dan *layer* keluaran.

4. Menghitung nilai *error* pada *layer* keluaran dengan fungsi *loss cross-entropy*.

$$Error = -\sum_{i=1}^{N}\sum_{j=1}^{M} target_{l_{ij}} \times \log Y_{l_{ij}} \quad (4.7)$$

Dengan kalkulasi *error* pada turunan fungsi aktivasi adalah:

$$\Delta_k = Error \times f'(input\_sum_k) \quad (4.8)$$

dimana Δ_k merupakan kalkulasi kesalahan pada turunan aktivasi di *layer* keluaran k, dan *error* merupakan nilai kesalahan berdasarkan *loss function*.

5. Kemudian melakukan pembaruan bobot antara *layer* keluaran l dan *layer* tersembunyi k.

$$W_{kl} \leftarrow W_{kl} + \alpha \times a_k \times \Delta_k \quad (4.9)$$

dimana W_{kl} merupakan bobot penghubung antara *layer* keluaran l dan *layer* tersembunyi, α merupakan (*learning rate)*

6. Hitung kalkulasi *error* pada bobot *layer* tersembunyi j dan *layer* tersembunyi k.

$$\Delta_j = f'(input\_sum_j) + \sum_k W_{jk}\, \Delta_k \quad (4.10)$$

dimana Δ_j merupakan kalkulasi *error* pada bobot *layer* tersembunyi j dan *layer* tersembunyi k.

7. Kemudian melakukan pembaruan bobot antara *layer* tersembunyi j dan *layer* tersembunyi k

$$W_{jk} \leftarrow W_{jk} + \alpha \times a_j \times \Delta_j \quad (4.11)$$

dimana W_{jk} merupakan bobot penghubung antara *layer* tersembunyi j dan *layer* tersembunyi, α merupakan (*learning rate*).

8. Hitung kalkulasi *error* pada bobot *layer* masukan i dan *layer* tersembunyi j.

$$\Delta_i = f'(input\_sum_i) + \sum_j W_{ij}\, \Delta_j \qquad (4.12)$$

dimana Δ_i merupakan kalkulasi *error* pada bobot *layer* masukan o dan *layer* tersembunyi j.

9. Kemudian melakukan pembaruan bobot antara *layer* tersembunyi j dan *layer* tersembunyi k

$$W_{ij} \leftarrow W_{ij} + \alpha \times X_i \times \Delta_i \qquad (4.13)$$

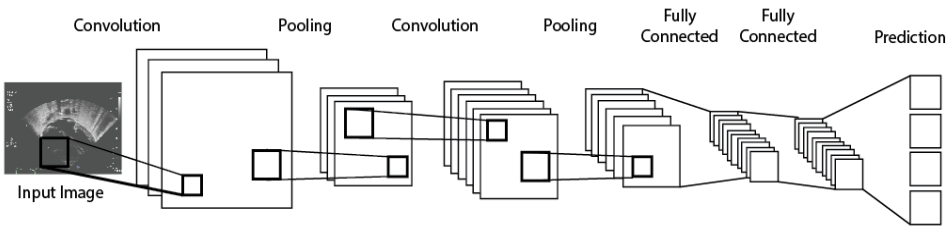
dimana W_{ij} merupakan bobot penghubung antara *layer* masukkan i dan *layer* tersembunyi, α merupakan (*learning rate*).

4.2 *Convolutional Neural Network*

Convolutional Neural Network (CNN) dikenal sebagai salah satu arsitektur deep learning yang khas dan variasi dari *Multilayer Perceptron* (MLP) yang terinspirasi oleh jaringan saraf manusia. CNN terdiri dari beberapa lapisan komputasi *neuron* dengan pemrosesan yang dilakukan minimal selangkah demi selangkah, yang mencapai peningkatan signifikan di bidang penelitian visi komputer.

Struktur jaringan saraf pada CNN terdiri dari node yang terhubung satu sama lain berdasarkan bobot. CNN biasanya tersusun dari lapisan konvolusional, lapisan *pooling*, dan lapisan yang terhubung sepenuhnya (*fully connected*) (Zhao & Kumar, 2017). Lapisan konvolusional bertujuan untuk mempelajari representasi fitur dari *input*, dan lapisan *pooling* bertujuan untuk mengurangi resolusi peta fitur (Q. Zhang et al., 2019).

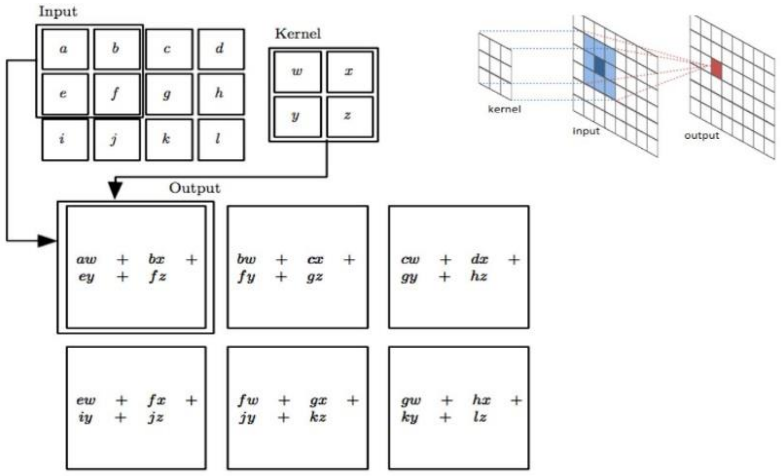
Pada tahun 1990-an, LeCun et. al., menerapkan algoritma pembelajaran berbasis gradient untuk CNN dan memperoleh hasil yang baik untuk masalah klasifikasi (LeCun, Bottou, Bengio, Haffner, & others, 1998). Dalam penelitian lebih lanjut dapat melakukan proses meningkatkan CNN, sehingga CNN memiliki beberapa keunggulan dibandingkan DNN. Selain memiliki beberapa keunggulan dibandingkan DNN, CNN juga berhasil dilatih dapat menyusus hierarki informasi seperti klasifikasi gambar, deteksi objek, informasi selama pra-pemrosesan dan segmentasi gambar. Komponen atau arsitektur utama dari proses CNN itu sendiri terdiri dari proses *convolutional, pooling layer* dan *fully connected layers* (Gambar 4.2).



Gambar 4.2. Arsitektur utama CNN

Convolutional layer

Convolutional layer merupakan blok bangunan inti CNN, di mana sebagian besar komputasi dilakukan pada lapisan ini. Pada lapisan ini akan menghasilkan sebuah filter dengan panjang dan tinggi. Misalkan, dilakukan proses *convolutional layer* dengan ukuran 5 x 5 x 5 dimana *pixel* 5 menyatakan tinggi *pixel* 5 lagi menyatakan lebar dan *pixel* 3 menyatakan tebal sesuai dengan channel dari image tersebut. Hasil dari ketiga filter tersebut akan melakukan pergeseran keseluruh bagian gambar. Setiap pergeseran akan dilakukan opersai "dot" antara *input* dan nilai dari filter tersebut untuk menghasilkan *output* yaitu *activation map*. Berikut adalah contoh proses operasi konvolusi yang terlihat pada Gambar 4.3:



Gambar 4.3. Operasi Konvolusi

Pada proses melakukan *convolutional layer*, ada atribut yang mungkin dilakukan pada proses tersebut yaitu *stride* dan *padding*.

Stride

Stride merupakan parameter yang digunakan untuk melakukan jumlah pegeseran filter pada proses convolutional layer. Jika nilai stride 1, maka *convolutional* filter akan bergeser sebanyak 1 *pixel* secara horizontal dan vertikal. Semakin kecil *stride* maka semakin detail informasi yang didapatkan dari sebuah masukkan gambar, namun membutuhkan proses komputasi lebih lama jika dibandingkan dengan *stride* yang besar.

Padding

Padding atau *zero padding* bertujuan untuk memanipulasi dimensi *output* dari *convolutional layer* (*feature map*). *Padding* merupakan parameter yang menentukan jumlah *pixel* yang berisi 0 yang ditambahkan disetiap sisi dari *input*. Untuk menghitung dimensi dari *feature map* dapat menggunakan persamaan:

$$Output = \frac{W-N+2P}{S} + 1 \qquad (4.14)$$

dimana, W adalah panjang/tinggi *input*, N adalah panjang/tinggi filter, P = *zero padding*, dan S adalah *stride*.

Pooling layer

Setelah dilakukan proses *convolution layer*, tahap selanjutnya adalah proses pooling atau sub-sampling layer (Lawrence, Giles, Tsoi, & Back, 1997). Keuntungan utama menggunakan teknik pooling adalah untuk mengurangi jumlah parameter yang bisa dilatih dan merepersentasikan data menjadi lebih kecil, mudah dikelola dan mudah mengontrol *overfitting*. Ada beberapa teknik pooling yang dapat digunakan yaitu *max pooling* dan *average pooling*. Max pooling adalah mengambil nilai maksimum dan *average pooling* mengambil nilai rata-rata. Berikut ilustrasi gambar tentang penggunaan *pooling layer*. Berikut contoh pengaplikasian proses *max pooling* (Gambar 4.4) dan *average pooling* (Gambar 4.5);

Max Pooling

Gambar 4.4. *Max pooling*

Average Pooling

Gambar 4.5. *Average pooling*

Fully Connected Layer (FC Layer)

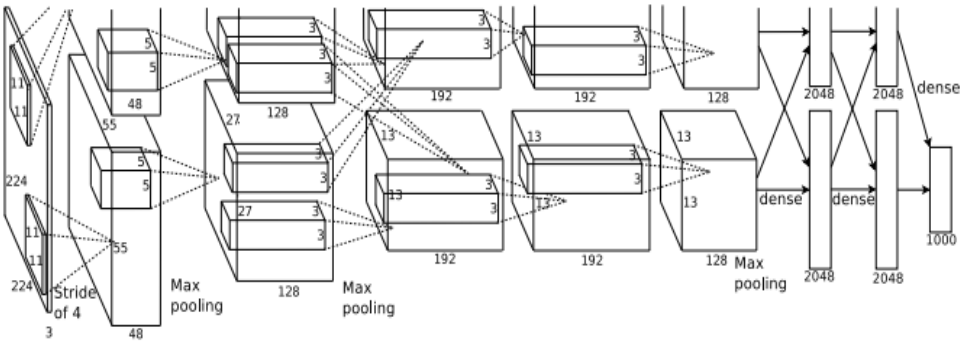
Feature map yang dihasilkan dari *feature extraction layer* masih berbentuk multidimensional array, sehingga harus melakukan "*flatten*" atau *reshape feature map* menjadi sebuah vector agar bisa digunakan sebagai *input* dari *fully-connected layer*. *FC layer* memiliki beberapa *hidden layer, activation function, output layer* dan *loss function.*

4.2.1 Arsitektur CNN

4.2.1.1 Arsitektur AlexNet

Arsitektur AlexNet yang diusulkan oleh Krizhevsky et al., memenangkan *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) (Krizhevsky, Sutskever, & Hinton, 2012). Arsitektur ini terdiri dari 5 lapisan konvolusional, beberapa di antaranya diikuti oleh lapisan max-pooling dan 3 lapisan yang terhubung sepenuhnya (*fully-connected layer*) dengan fungsi aktivasi *softmax* pada bagian akhir. Lapisan konvolusional pertama menyaring gambar *input* 224 ×224×3 dengan 96 kernel ukuran 11×11×3 dengan stride 4 *pixel*. Lapisan konvolusional kedua mengambil *output* dari lapisan konvolusional pertama dan menyaringnya dengan 256 kernel ukuran 5×5×48. Lapisan konvolusional ketiga memiliki 384 kernel ukuran 3×3×256 yang terhubung ke *output pooling*/normalisasi dari lapisan konvolusional kedua. Lapisan konvolusional keempat memiliki 384 kernel ukuran 3×3×192, dan lapisan konvolusional kelima memiliki 256 kernel ukuran 3×3×192. Lapisan konvolusional ketiga,

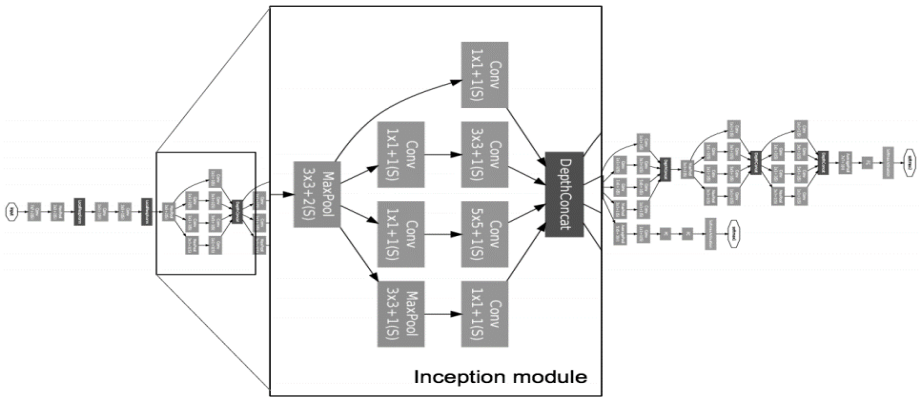
keempat, dan kelima terhubung satu sama lain tanpa lapisan *pooling*/normalisasi. Lapisan yang terhubung sepenuhnya (*fully-connected layer*) memiliki 4.096 *neuron*. Arsitektur AlexNet dapat dilihat pada Gambar 4.6.



Gambar 4.6. Arsitektur AlexNet (Krizhevsky et al., 2012)

4.2.1.2 Arsitektur GoogLeNet

Arsitektur GoogLeNet yang diusulkan oleh Szegedy et al., terdiri dari 22 lapisan konvolusional termasuk 9 modul Inception (Szegedy et al., 2015). Modul Inception memiliki 3 ukuran berbeda untuk lapisan konvolusi dengan filter kernel 5×5, 3×3, dan 1×1 dan lapisan *pooling* dengan filter 3×3. Ukuran bidang reseptif dalam jaringan ini adalah 224×224×3 menggunakan ruang warna RGB dengan parameter yang diberikan. Arsitektur GoogLeNet dapat dilihat pada Gambar 4.7.



Gambar 4.7. Arsitektur GoogLeNet (Szegedy et al., 2015)

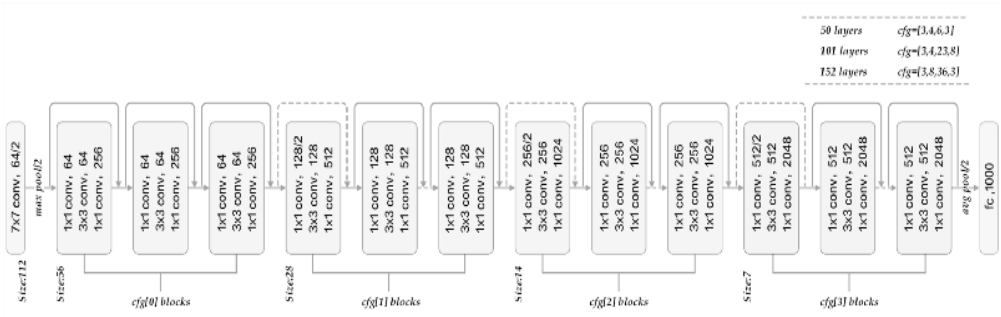
4.2.1.3 Arsitektur VGGNET

VGGNet merupakan arsitektur yang dibangun oleh Karen Simonyan dan AndrewZisserman. Arsitektur ini menyatakan bahwa kedalam sebuah jaringan merupakan komponen penting untuk menghasilkan performansi tinggi. Semakin dalam jaringan CNN maka semakin tinggi akurasinya (Simonyan & Zisserman, 2014). VGGNet memiliki banyak parameter sehingga membutuhkan banyak memori. Sebagian besar parameter ini ada pada *fully connected layer* pada bagian awal, karena jika itu dihilangkan ternyata tidak menurunkan performansi secara signifikan. Simonyan et al., mengusulkan *deep convolutional network* yang memiliki kedalaman antara 16-19 lapisan dan terdiri dari filter konvolusi yang sangat kecil (Simonyan & Zisserman, 2014). Arsitektur VGGNet dapat dilihat pada Gambar 4.8.

Gambar 4.8. Arsitektur VGGNET (Simonyan & Zisserman, 2014).

4.2.1.4 Arsitektur ResNet

He *et al*., memformulasikan ulang layer sebagai fungsi pembelajaran residual dengan mengacu pada *input* layer daripada mempelajari fungsi yang tidak direferensikan dan mengusulkan ResNet yang memiliki kedalaman maksimal 152 layer (He, Zhang, Ren, & Sun, 2016). Ini berarti bahwa ResNet delapan kali lebih dalam dari VGGNet, tetapi masih memiliki kompleksitas yang lebih rendah. Pada 2015, ResNet memenangkan *challenge* klasifikasi ILSVRC. Arsitektur ResNet dapat dilihat pada Gambar 4.9.



Gambar 4.9. Arsitekur ResNet (He et al., 2016)

4.2.2 Algoritma CNN

4.2.2.1 *Forward Pass*

Lapisan *input* adalah gambar yang dimensinya (lebar x tinggi) untuk *grayscale* dan (lebar x tinggi x 3) untuk gambar RGB. Operasi konvolusi dengan *input* gambar 2 dimensi secara umum dapat ditulis dengan persamaan sebagai berikut:

$$(K * I)_{(i,j)} = \sum_m \sum_n I_{(i-m,j-n)} K_{(m,n)} + b \tag{4.15}$$

dimana *i* dan *j* adalah *pixel* dari gambar, *K* sebagai kernel, *I* sebagai *input* dan *b* adalah bias. Operasi konvolusi dapat dilihat sebagai perkalian matriks antara *input* gambar dan filter kernel dimana outputnya dapat dihitung dengan *dot product*. Filter akan menggeser seluruh gambar mengulangi operasi *dot product* yang sama. Dua hal yang harus diperhatikan:

a. Filter harus memiliki jumlah *channel* yang sama dengan gambar *input*.
b. Secara umum diketahui bahwa semakin dalam jaringan semakin banyak filter digunakan, intuisi di baliknya adalah semakin banyak filter yang kita miliki, semakin banyak deteksi tepi dan fitur yang akan didapatkan.

Untuk melakukan operasi konvolusi, kernel diputar 180^o (secara horizontal dan vertikal). Persamaan konvolusi dari *input* pada *layer l* diberikan oleh:

$$x_{i,j}^{l} = rot_{180°}\{w_{m,n}^{l}\} * o_{i,j}^{l-1} + b_{i,j}^{l} \quad (4.16)$$

$$x_{i,j}^{l} = \sum_m \sum_n w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b_{i,j}^{l} \quad (4.17)$$

$$o_{i,j}^{l} = f(x_{i,j}^{l}) \quad (4.18)$$

Untuk membantu dalam *forward* dan *backpropagation*, digunakan notasi sebagai berikut:

1. *l* adalah lapisan ke-*l* di mana *l* = 1 adalah lapisan pertama dan *l* = *L* adalah lapisan terakhir.
2. *Input* x adalah dimensi H × W dan memiliki i oleh j sebagai iterator
3. Filter atau kernel w adalah dimensi k1 × k2 memiliki m oleh n sebagai iterator
4. $w^{l}{}_{m,n}$ adalah matriks bobot yang menghubungkan *neuron* lapisan *l* dengan *neuron* lapisan *l-1*.
5. b^{l} adalah unit bias pada *layer l*.
6. $x^{l}{}_{i,j}$ adalah vektor *input* yang bergeser pada *layer l* ditambah bias yang ditunjukkan:
   $$x_{i,j}^{l} = \sum_m \sum_n w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b^{l}$$
7. $o^{l}{}_{i,j}$ adalah vektor *output* pada *layer l* yang diberikan oleh:
   $o_{i,j}^{l} = f(x_{i,j}^{l})$
8. f (·) adalah fungsi aktivasi. Penerapan lapisan aktivasi ke vektor *input* pada *layer l* diberikan oleh $f(x_{i,j}^{l})$

Selain itu, penentuan dimensi *output* juga dapat ditentukan dari masing-masing lapisan dengan *hyperparameters.*

$$\frac{W-F+2P}{S}+1 \tag{4.19}$$

Berdasarkan persamaan (4.19), dapat dihitung ukuran spasial dari dimensi *output* dimana *hyperparameter* yang dipakai adalah ukuran lebar gambar *input* (W), filter/kernel (F), *Stride* yang diterapkan (S) dan jumlah *zero padding* yang digunakan (P). *Stride* merupakan nilai yang digunakan untuk menggeser filter melalui *input* gambar dan *Zero Padding* adalah nilai untuk mendapatkan angka nol di sekitar tepi gambar. Ilustrasi proses konvolusi pada *forward propagation* ditunjukkan pada Gambar 4.10.

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

$W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$	$W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$
$W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$	$W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$

Gambar 4.10. Operasi *Forward Convolution*

Untuk *nonlinearity layer*, lapisan ini mengambil *output* lapisan konvolusi dan menerapkan fungsi aktivasi ReLU seperti pada persamaan (4.20). Fungsi ini melakukan *thresholding* dengan nilai nol terhadap nilai *pixel*

pada *input* gambar. Aktivasi ini membuat seluruh nilai *pixel* yang bernilai kurang dari nol pada suatu gambar akan dijadikan nol.

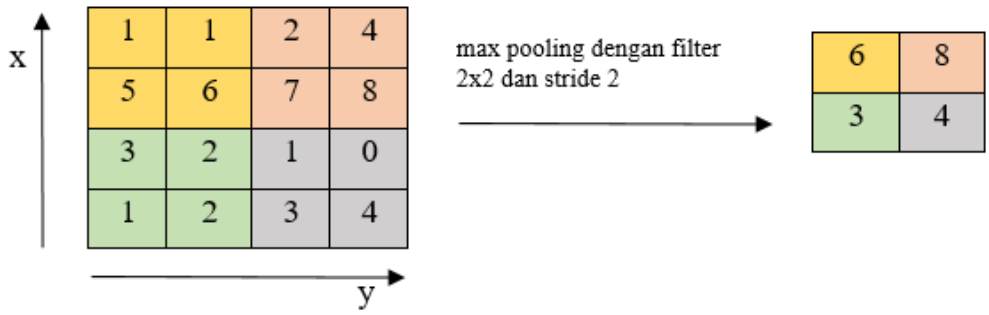
$$y_{i,j}^{l} = \sigma(x_{i,j}^{l}) \tag{4.20}$$

$$\sigma(\mathrm{x}) = \max(w_{m,n}^{l} o_{i,j}^{l-1} + b_{i,j}^{l}, 0) \tag{4.21}$$

Lalu, untuk *pooling layer* adalah lapisan yang mengurangi dimensi dari *feature map* atau lebih dikenal dengan langkah untuk *downsampling*, sehingga mempercepat komputasi karena parameter yang harus di*update* semakin sedikit dan mengatasi *overfitting*. Lapisan ini ditempatkan di antara dua lapisan konvolusional. Setiap peta fitur dari pooling layer dihubungkan ke peta fitur yang sesuai dari lapisan convolutional sebelumnya. Pooling yang digunakan adalah *Max Pooling*. *Max Pooling* akan menentukan nilai maksimum tiap pergeseran filter, fungsi *max pooling* ditunjukkan pada persamaan 4.22

$$f_{max}(x) = max_i\, x_i \tag{4.22}$$

dimana vektor x berisi nilai-nilai aktivasi dari daerah lokal *pooling* N *pixel* (dimensi *pooling* yang umum adalah 2x2 atau 3x3) dalam sebuah gambar atau *channel*. Ilustrasi proses pooling pada *forward propagation* ditunjukkan pada Gambar 4.11.



Gambar 4.11. Ilustrasi Proses *Max Pooling*

Hasil dari proses konvolusi menjadi *input* pada *fully-connected layer*. *Fully connected layer* adalah lapisan dimana semua *neuron* aktivasi dari lapisan sebelumnya terhubung semua dengan *neuron* di lapisan selanjutnya seperti halnya jaringan saraf tiruan biasa. Setiap aktivasi dari lapisan sebelumnya perlu diubah menjadi data satu dimensi sebelum dapat dihubungkan ke semua *neuron* di *fully connected layer*. Lapisan ini biasanya digunakan pada metode *Multi-Layer Perceptron* (MLP) dan bertujuan untuk mengolah data sehingga bisa diklasifikasikan. Persamaan umum untuk *fully connection layer* adalah sebagai berikut:

$$\hat{y} = \sigma\,(W \times f + b) \tag{4.23}$$

Untuk total prediksi P, *output* jaringan yang diprediksi y_p dan nilai targetnya yang sesuai t_p , *error* kuadrat rata-rata diberikan oleh:

$$E = \frac{1}{2}\sum_P (t_p - y_p)^2 \tag{4.24}$$

Proses belajar akan dicapai dengan menyesuaikan bobot sehingga y_p sedekat mungkin atau sama dengan t_p yang sesuai.

4.2.2.2 *Backward Pass*

Pada *backpropagation* ada dua *update* yang dilakukan, yaitu untuk bobot dan delta.

1. *UPDATE* BOBOT

Untuk menghitung $\frac{\partial E}{\partial w^l_{m',n'}}$ yang dapat diartikan sebagai pengukuran bagaimana perubahan dalam satu *pixel* $w_{m',n'}$ dalam kernel bobot mempengaruhi fungsi loss E. Konvolusi antara peta fitur masukan dimensi H × W dan kernel bobot dimensi $k_1 \times k_2$ menghasilkan peta fitur keluaran ukuran (H − k1 + 1) oleh (W − k2 + 1). Komponen gradien untuk bobot individu dapat diperoleh dengan menerapkan *chain rule* dengan cara berikut:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \quad (4.25)$$

Persamaan (4.25), $x^l_{i,j}$ sama dengan $\sum_m \sum_n w^l_{m,n}\, o^{l-1}_{i+m,j+n} + b^l$, sehingga didapat:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}} \left(\sum_m \sum_n w^l_{m,n}\, o^{l-1}_{i+m,j+n} + b^l \right) \quad (4.26)$$

Selanjutnya memperluas penjumlahan dalam persamaan (4.26) dan mengambil turunan parsial untuk semua komponen menghasilkan nilai nol untuk semua kecuali komponen di mana m = m′ dan n = n′ dalam $w^l_{m,n} o^{l-1}_{i+m,j+n}$ sebagai berikut:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{0,0} o^{l-1}_{i+0,j+0} + \cdots + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b^l \right) = \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{m',n'} o^{l-1}_{i+m',j+n'} \right) = o^{l-1}_{i+m',j+n'} \tag{4.27}$$

Mengganti persamaan (4.27) dalam persamaan (4.25) menghasilkan persamaan sebagai berikut:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \, o^{l-1}_{i+m',j+n'} = rot_{180°} \{\delta^l_{i,j}\} * o^{l-1}_{m',n'} \tag{4.28}$$

Penjumlahan ganda dalam persamaan (4.28) adalah hasil dari pembagian bobot dalam jaringan (kernel dengan bobot yang sama dimasukkan ke semua peta fitur *input* selama konvolusi). Penjumlahan mewakili kumpulan semua gradien $\delta^l_{i,j}$ yang berasal dari semua *output* di lapisan *l*. Selanjutnya diperoleh gradien dengan peta *filter* dan dilakukan korelasi silang yang ditransformasikan menjadi konvolusi dengan "membalik" matriks $\delta^l_{i,j}$ (secara horizontal dan vertikal) dengan cara yang sama seperti membalik *filter* selama *forward propagation.*

2. *UPDATE* DELTA

Untuk menghitung delta digunakan persamaan sebagai berikut:

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}} \tag{4.29}$$

Dengan menggunakan *chain rule* memberi persamaan berikut:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{i,j \in Q} \frac{\partial E}{\partial x_{Q}^{l+1}} \frac{\partial x_{Q}^{l+1}}{\partial x_{i',j'}^{l}} = \sum_{i,j \in Q} \delta_{Q}^{l+1} \frac{\partial x_{Q}^{l+1}}{\partial x_{i',j'}^{l}} \tag{4.30}$$

Di wilayah Q, tinggi berkisar dari $i' - 0$ hingga $i' - (k_1 - 1)$ dan lebar $j' - 0$ hingga $j' - (k_2 - 1)$. Keduanya hanya dapat diwakili oleh $i' - m$ dan $j' - n$ dalam penjumlahan karena iterator m dan n ada dalam rentang yang sama dengan $0 \leq m \leq k_1 - 1$ dan $0 \leq n \leq k_2 - 1$.

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-2} \frac{\partial E}{\partial x_{i'-m,j'-n}^{l+1}} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}}$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-2} \delta_{i'-m,j'-n}^{l+1} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} \tag{4.31}$$

Persamaan (4.31), $x_{i'-m,j'-n}^{l+1}$ setara dengan $\Sigma_{m'} \Sigma_{n'} \; w_{m'n'}^{l+1} \; o_{i'-m+m',j'-n+n'}^{l} + \; b^{l+1}$ dan sehingga didapat:

$$\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} = \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m'n'}^{l+1}\, o_{i'-m+m',j'-n+n'}^{l} + b^{l+1} \right)$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(\sum_{m'} \sum_{n'} w_{m'n'}^{l+1}\, f(x_{i'-m+m',j'-n+n'}^{l}) + b^{l+1} \right) \quad (4.32)$$

Lebih lanjut memperluas penjumlahan dalam persamaan (4.32) dan mengambil turunan parsial untuk semua komponen menghasilkan nilai nol untuk semua kecuali komponen di mana m ′ = m dan n ′ = n sehingga $f\,(x_{i'-m+m',j'-n+n'}^{l})$ menjadi $f\,(x_{i',j'}^{l})$ dan $w_{m'n'}^{l+1}$ menjadi $w_{m,n}^{l+1}$ pada bagian yang relevan dari penjumlahan yang diperluas sebagai berikut:

$$\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} = \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m'n'}^{l+1}\, f\left(x_{0-m+m',0-n+n'}^{l}\right) + \cdots + w_{m,n}^{l+1}\, f\left(x_{i',j'}^{l}\right) + \cdots + b^{l+1} \right)$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}} \left(w_{m,n}^{l+1}\, f\left(x_{i',j'}^{l}\right) \right) = w_{m,n}^{l+1}\, \frac{\partial}{\partial x_{i',j'}^{l}} \left(f\left(x_{i',j'}^{l}\right) \right) = w_{m,n}^{l+1}\, f'\left(x_{i',j'}^{l}\right)$$

(4.33)

Mengganti persamaan (4.33) dalam persamaan (4.31) memberikan hasil sebagai berikut:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-2} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1}\, f'\left(x_{i',j'}^{l}\right) \quad (4.34)$$

Dalam *backpropagation*, digunakan *flipped kernel* dan sebagai hasilnya didapat konvolusi yang dinyatakan sebagai korelasi silang dengan *flipped kernel*:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-2} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right) = rot_{180°}\left\{\sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-2} \delta_{i'+m,j'+n}^{l+1} w_{m,n}^{l+1}\right\} f'\left(x_{i',j'}^{l}\right) \quad (4.35)$$

$$= \delta_{i',j'}^{l+1} * rot_{180°}\left\{w_{m,n}^{l+1}\right\} f'\left(x_{i',j'}^{l}\right) \quad (4.36)$$

4.3 *Recurrent Neural Network*

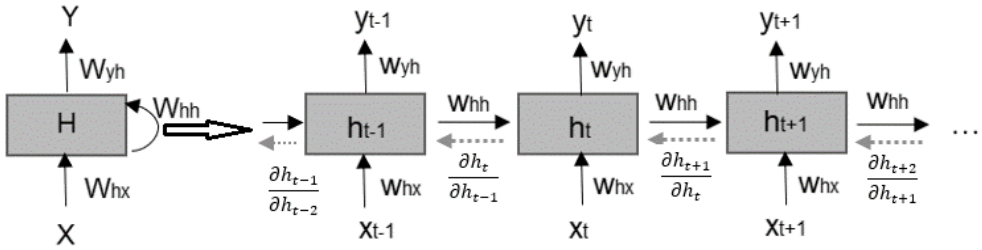
Recurrent Neural Network (RNN) adalah model pertama dari jaringan saraf tiruan berulang yang diperkenalkan. RNN merupakan jenis arsitektur jaringan saraf tiruan yang pemrosesannya dipanggil berulang-ulang untuk memproses data masukkan (Lui & Chow, 2018). Aktivasi simpul berulang terdiri dari umpan balik untuk dirinya sendiri dari satu langkah waktu ke langkah berikutnya. RNN masuk dalam kategori *deep learning* karena data diproses secara otomatis dan tanpa pendefinisian fitur (Wiatowski & Bölcskei, 2018).

RNN telah digunakan di berbagai bidang dan aplikasi menarik yang berhubungan dengan memori asosiatif, optimisasi dan generalisasi (Singh, Pandey, Pawar, & Janghel, 2018). RNN mengalami kemajuan yang pesat dan telah merevolusi bidang-bidang seperti pemrosesan data sinyal EKG (Lui & Chow, 2018) (Strodthoff & Strodthoff, 2018)(Goto et al., 2019), pemrosesan bahasa alami (Kumar et al., 2016)(Goldberg, 2016), pengenalan suara (Zen & Sak, 2015), sintesa musik (Choi, Fazekas, Sandler, & Cho, 2017), pemrosesan data finansial seri waktu

(Rout, Dash, Dash, & Bisoi, 2017), analisa deret DNA (Quang & Xie, 2016), dan sebagainya.

4.3.1 Arsitektur RNN

RNN memiliki proses *forward* dan *backward pass* yang sama dengan *artificial neural network* lainnya, hanya pada proses *backpropagation*, istilahnya menjadi *backpropagation through time* (BPTT)(Bullinaria, 2015). Proses *forward* (garis panah berwarna hitam) dan *backward pass* (garis putus-putus berwarna jingga) bisa digambarkan pada Gambar 4.12 berikut:



Gambar 4.12. Struktur RNN

Model RNN yang ditunjukkan Gambar 27 mengacu pada tiga bobot matriks, yaitu bobot matriks antara input dan *hidden layer* ($w_{hx} \in R^{h*x}$), bobot matriks antara dua *hidden layers* ($w_{hh} \in R^{h*h}$), dan bobot matriks antara hidden dan *output layer* ($w_{yh} \in R^{y*h}$). Selain itu bias juga bisa ditambahkan dalam model, yaitu bias vektor yang ditambah ke hidden

layer ($b_h \in R^h$) dan bias vektor yang ditambah ke *output layer* ($b_y \in R^y$). Model RNN tersebut bisa direpresentasikan pada persamaan:

$$h_t = f_w(h_{t-1}, x_t) \tag{4.37}$$

$$h_t = f(w_{hx}x_t + w_{hh}x_{t-1} + b_h) \tag{4.38}$$

$$\hat{y}_t = f(w_{yh}h_t + b_y) \tag{4.39}$$

Fungsi f pada persamaan (4.37), (4.38) dan (4.39) adalah fungsi aktivasi (contoh*: tanh, sigmoid, softmax*, dan sebagainya). RNN dilatih secara sekuensial dengan pembelajaran terawasi (*supervised learning*). Untuk langkah waktu t, *error* dihasilkan dari perbedaan antara prediksi dan aktual $(\hat{y}_t - y_t)$. Secara keseluruhan, *error* atau *loss* $\mathcal{L}$ adalah penjumlahan dari *loss* pada langkah waktu dari t ke T, $[t, T]$:

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T} \mathcal{L}(\hat{y}_t, y_t) \tag{4.40}$$

4.3.2 Algoritma RNN

1. *Forward pass*; Perhitungan *input* ke *hidden unit* dapat dihitung dengan mulai dari $t = 1$ dan secara rekursif menerapkan tiga persamaan, dimana terus menambah t pada setiap langkah:

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \quad (4.41)$$

$$b_h^t = \theta_h(a_h^t) \quad (4.42)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t \quad (4.43)$$

2. *Backward pass*; Perhitungan *derivative* dihitung dengan mulai dari $t = T$ dan secara rekursif menerapkan fungsi-fungsi di bawah ini, mengurangi t pada setiap langkah:

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right) \quad (4.44)$$

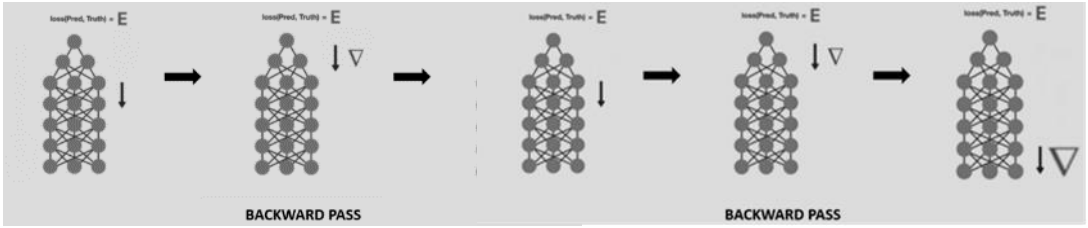
$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t} \quad (4.45)$$

3. Akhirnya, dengan mengingat bahwa bobot ke dan dari setiap unit di lapisan tersembunyi adalah sama pada setiap langkah waktu, maka menjumlahkan seluruh urutan untuk mendapatkan turunan yang berkaitan dengan masing-masing bobot jaringan:

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t \qquad (4.46)$$

4.3.3 *Vanishing/Exploding Gradient*

Vanishing atau *exploding gradient* terjadi saat *backward pass*, karena dalam *backpropagation* ada proses *derivative* dimana semakin sering derivasi dihitung, semakin kecil angkanya dan bahkan mendekati nilai 0 atau dikatakan menghilang ketika sampai pada lapisan-lapisan awal (Gambar 4.13).



(a) (b)

Gambar 4.13. (a) *Vanishing Gradient*, (b) *Exploding Gradient*

Misalkan suatu data memiliki total pada seluruh langkah waktu (*time step) T*:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \qquad (4.47)$$

Dengan aturan rantai (*chain rule*), persamaan (4.47) di atas dapat dijabarkan menjadi:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{4.48}$$

Anotasi $\frac{\partial h_t}{\partial h_k}$ pada persamaan (4.48) adalah turunan dari *hidden state* yang menyimpan memori pada waktu t yang sehubungan dengan *hidden state* pada waktu sebelumnya k. Fase ini melibatkan perkalian Jacobians $\frac{\partial h_i}{\partial h_{i-1}}$ untuk seluruh waktu t dan satu waktu k:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots \frac{\partial h_{k+1}}{\partial h_k} \tag{4.49}$$

$$= \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \tag{4.50}$$

Perkalian Jacobians pada persamaan (4.50) adalah turunan dari h_t yang ditulis h_{t-1} (misal: $\frac{\partial h_t}{\partial h_{t-1}}$), dimana ketika dievaluasi $W^T[f'(h_{t-1})]$:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[f'(h_{t-1})] \tag{4.51}$$

dimana $diag$ adalah proses konversi dari vektor ke matriks diagonal.

Matriks Jacobian $\frac{\partial h_t}{\partial h_{t-1}}$ menampilkan eigendecomposition yang diberikan $W^T diag[f'(h_{t-1})]$, maka dihasilkan *eigenvalues* $\lambda_1, \lambda_2, \ldots, \lambda_n$ dimana $|\lambda_1| > |\lambda_2| \ldots |\lambda_n|$ dan yang sesuai dengan *eigenvektor* $v_1, v_2, \ldots, v_n$. Tiap perubahan pada hidden state $\triangle h_t$ pada vektor v_i memiliki pengaruh perkalian antara *eigenvalue* yang berkaitan dengan eigenvektor $(\lambda_i \triangle h_t)$.

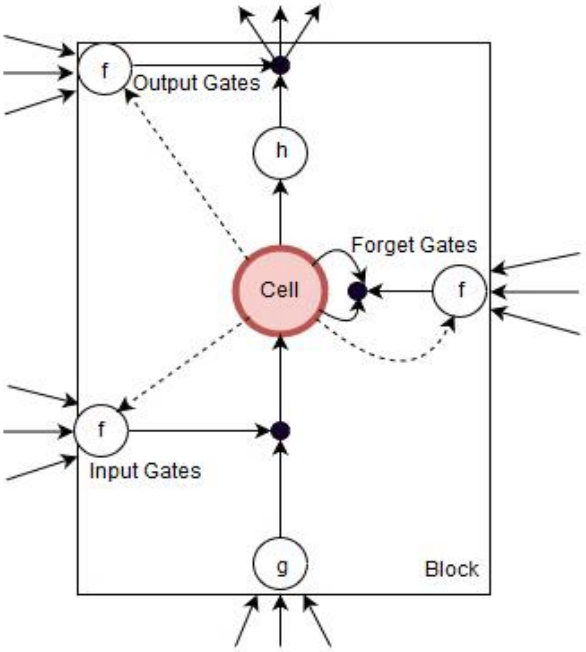
Perkalian Jacobian yang terlihat pada persamaan (4.51) mengimplikasikan langkah-langkah waktu berikutnya, yang menghasilkan penskalaan perubahan faktor yang setara dengan λ_i. λ_i merepresentasikan *eigenvalue* i^{th}. Urutan $\lambda_i^1 \triangle h_1, \lambda_i^2 \triangle h_2, \ldots, \lambda_i^n \triangle h_n$, dimana mudah untuk melihat faktor λ_i^t akan didominasi $\triangle h_t$ karena fase ini akan maju menjadi eksponensial secepat $t \to \infty$. Artinya jika dihasilkan *eigenvalue* terbesar $\lambda_i < 1$ maka akan terjadi *vanishing gradient*, sebaliknya jika nilai $\lambda_i > 1$, maka akan terjadi *exploding gradient*. Jadi nilai $\lambda_i = 1$ untuk menghindari *vanishing* atau *exploding gradient.*

Untuk mengatasi permasalahan *vanishing* dan *exploding gradient* pada RNN standar, maka dapat digunakan unit *Long Short-Term Memory*

(LSTM) dan *Gated Recurrent Unit* (GRU) (Pascanu, Mikolov, & Bengio, 2013).

4.3.4 *Long Short-Term Memory*

Long Short-Term Memory (LSTM) adalah jenis modul pemrosesan lain untuk RNN. LSTM diciptakan oleh Hochreiter & Schmidhuber (1997). Struktur RNN yang dimodelkan dengan arsitektur LSTM dijelaskan pada Gambar 4.14. Gambar 4.14 menjelaskan arsitektur LSTM yang terdiri dari tiga *gate* (f), yaitu *input, forget* dan *ouput gates*. Tidak ada fungsi aktivasi yang diterapkan di dalam sel memori. Fungsi aktivasi hanya ada di *input gates* (g) yaitu *tanh,* dan *output gates* (h) yaitu *sigmoid*. Fungsi aktivasi *sigmoid* yang merupakan aktivasi untuk gate tertutup (inisialisasi 0) dan *gate* terbuka (inisialisasi 1). Nilai 0 (nol) artinya informasinya diblok total yang menandakan untuk menyingkirkan informasi pembelajaran, sedangkan 1 (satu) artinya mengikutkan keseluruhan informasi yang menandakan untuk menjaga informasi pembelajaran. Koneksi bobot dari sel ke *gate* ditampilkan dengan garis putus-putus. Semua koneksi lain di dalam blok setara, yang memiliki bobot tetap bernilai 1.



Gambar 4.14. Arsitektur LSTM

Forward pass di RNN standar dihitung *input* x yang berututan dengan panjang T dengan mulai $t = 1$ dan secara rekursif dengan menerapkan pembaharuan persamaan sambal menambah t, dan *backward pass* BPTT dihitung mulai dari $t = T$, dan secara rekursif menghitung unit *derivative* di masing-masing langkah waktu (*time step*). Seperti RNN standar, semua status dan aktivasi diinisialisasi ke nol pada $t = 0$, dan semua $\delta = 0$ pada $t = T + 1$.

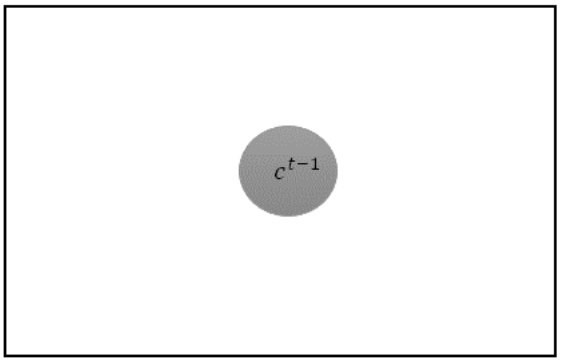
Persamaan LSTM hanya diberikan untuk satu blok memori saja. Untuk beberapa blok perhitungan hanya diulang untuk setiap blok, dalam urutan

apa pun. Subskrip i, f dan o merujuk masing-masing ke *input gates*, *forget gates*, dan *output gates* dari blok. Subskrip c mengacu pada salah satu dari sel memori C. Bobot dari sel c ke *input*, *forget*, dan *output gates* di anotasikan masing-masing yaitu w_i, w_f, w_o. Urutan perhitungan persamaan selama proses *forward* dan *backward pass* itu penting, dan harus dilanjutkan seperti yang ditentukan di bawah ini.

A. Forward Pass

Proses *forward pass* pada LSTM dengan satu sel memori adalah sebagai berikut:

(1) Inisialisasi *state* (sel memori): pada awal waktu t, sel-sel memori LSTM berisi nilai-nilai dari proses iterasi waktu sebelumnya $(t-1)$, dijelaskan pada Gambar 4.15.



Gambar 4.15. Inisialisasi *state* LSTM

(2) Komputasi *input* dan *gate*: pada waktu t, LSTM menerima inputan baru berupa vektor x^t (termasuk bias), serta *output* vektornya pada langkah waktu sebelumnya, h^{t-1}, yang dijelaskan pada Gambar 4.16.

$$a^t = tanh(W_c x^t + U_c h^{t-1}) \tag{4.52}$$

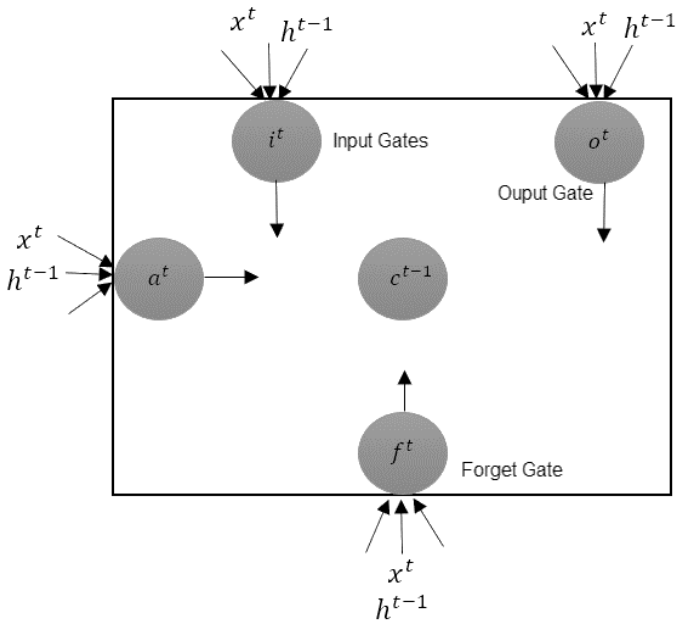
$$i^t = \sigma(W_i x^t + U_i h^{t-1}) = \sigma(\hat{i}^t) \tag{4.53}$$

$$f^t = \sigma\left(W_f x^t + U_f h^{t-1}\right) = \sigma(\hat{f}^t) \tag{4.54}$$

$$o^t = \sigma(W_o x^t + U_o h^{t-1}) = \sigma(\hat{o}^t) \tag{4.55}$$

Dengan mengabaikan fungsi aktivasi non-linearitas:

$$z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ f^t \\ o^t \end{bmatrix} = \begin{bmatrix} W^c & U^c \\ W^i & U^i \\ W^f & U^f \\ W^o & U^o \end{bmatrix} \cdot \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} = W.I^t \tag{4.56}$$



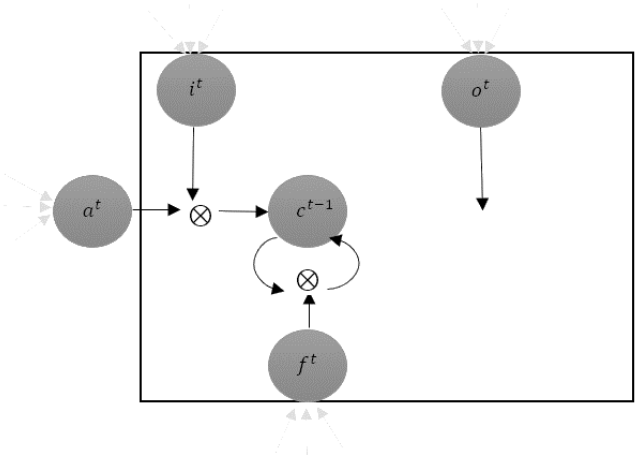
Gambar 4.16. Struktur *input, forget,* dan *output gates* LSTM dengan masukkan berupa vektor (x^t dan h^{t-1})

(3) Perbaharui (*update*) sel memori: pada tahapan ini nilai-nilai sel memori diperbaharui dengan mengkombinasikan a^t dan isi sel sebelumnya c^{t-1}. Kombinasi didasarkan pada besarnya *input gate* i^t dan *forget gate* f^t (dijelaskan pada Gambar 4.17)

$$c^t = i^t \odot a^t + f^t \odot c^{t-1} \tag{4.57}$$

dimana $\odot$ menunjukkan perkalian Hadamard. Isi dari sel-sel memori diperbaharui sampai nilai terakhir.

$$c^{t-1} \rightarrow c^t \tag{4.58}$$

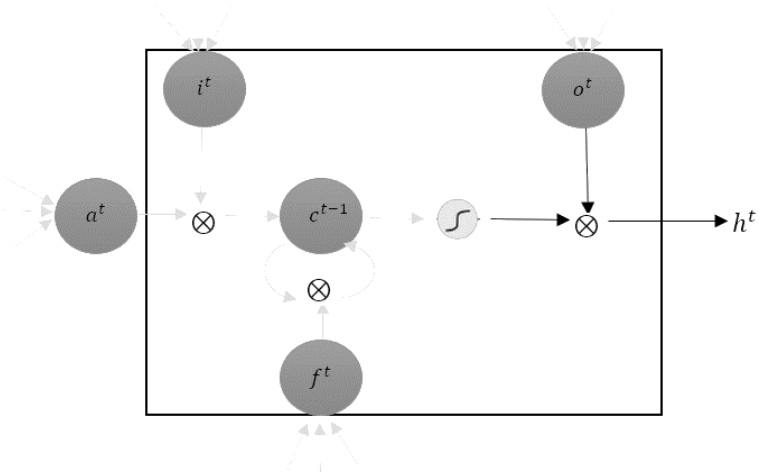


Gambar 4.17. Proses perbaharuan (*update*) sel memori

(4) *Output* (keluaran): Di akhir, sel LSTM menghitung nilai *output* dengan melewatkan nilai sel yang diperbaharui (*update*) melalui non-linearitas. *Output gate* menentukan berapa banyak *output* yang dihitung benar-benar dikeluarkan dari sel sebagai hasil akhir h^t (dijelaskan pada Gambar 4.18)

$$h^t = o^t \odot f(c^t) \tag{4.59}$$

Fungsi aktivasi f di fase ini adalah *tanh*.



Gambar 4.18. Proses LSTM hingga menghasilkan *output*

B. Backward pass

Backpropagation through time (BPTT) dikenal proses *backward pass* pada LSTM. Berikut penjelasan tiap-tiap proses yang dijelaskan dengan:

(1) Hasil keluaran dari proses *forward pass*, dimana $h^t = o^t \odot f(c^t)$ pada persamaan (4.60), dengan $\delta h^t = \frac{\partial E}{\partial h^t}$, maka dengan aturan rantai (*chain rule*):

$$\frac{\partial E}{\partial o_i^t} = \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial o_i^t} = \delta h_i^t \cdot f(c_i^t) \tag{4.60}$$

menjadi:

$$\delta o^t = \delta h^t \odot f(c^t) \tag{4.61}$$

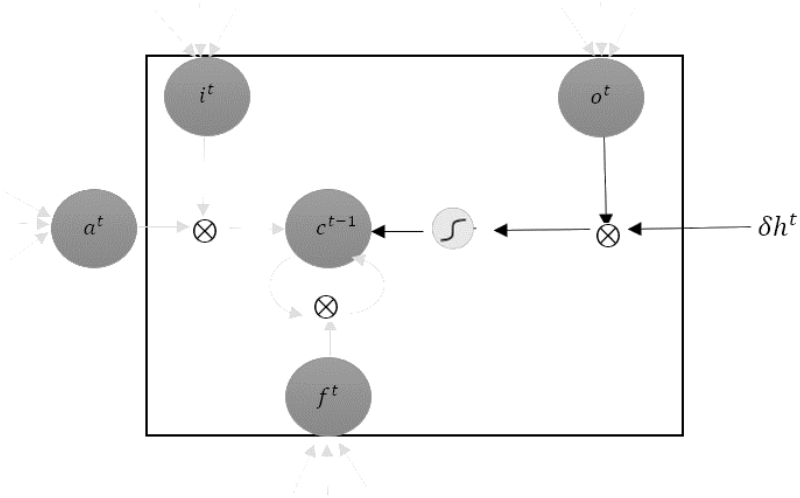
kemudian:

$$\frac{\partial E}{\partial c_i^t} = \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial c_i^t} = \delta h_i^t \cdot o_i^t (1 - f^2(c_i^t)) \quad (4.62)$$

menjadi:

$$\delta c^t += \delta h^t \odot o^t \odot (1 - f^2(c^t)) \quad (4.63)$$

Fungsi aktivasi f di fase ini adalah *tanh.* Perhatikan untuk tanda $+=$ adalah agar *gradient* ditambahkan ke *gradient* dari langkah waktu $t + 1$. Penjelasan pada tahap pertama di *backward pass* digambarkan pada Gambar 4.19.



Gambar 4.19. Proses *backward pass* dari hasil keluaran *forward pass*

(2) Perbaharui sel memori LSTM pada proses *backward pass*, dimana saat proses *forward pass* $c^t = i^t \odot a^t + f^t \odot c^{t-1}$, dengan $\delta c^t = \frac{\partial E}{\partial c^t}$ yang dijelaskan pada Gambar 4.20, maka dari:

$$\frac{\partial E}{\partial i_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial i_i^t} = \delta c_i^t \cdot a_i^t \tag{4.64}$$

menjadi:

$$\delta i^t = \delta c^t \odot a^t \tag{4.65}$$

untuk:

$$\frac{\partial E}{\partial f_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial f_i^t} = \delta c_i^t \cdot c_i^{t-1} \tag{4.66}$$

menjadi:

$$\delta f^t = \delta c^t \odot c^{t-1} \tag{4.67}$$

untuk:

$$\frac{\partial E}{\partial a_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial a_i^t} = \delta c_i^t \cdot i_i^t \tag{4.68}$$

menjadi:

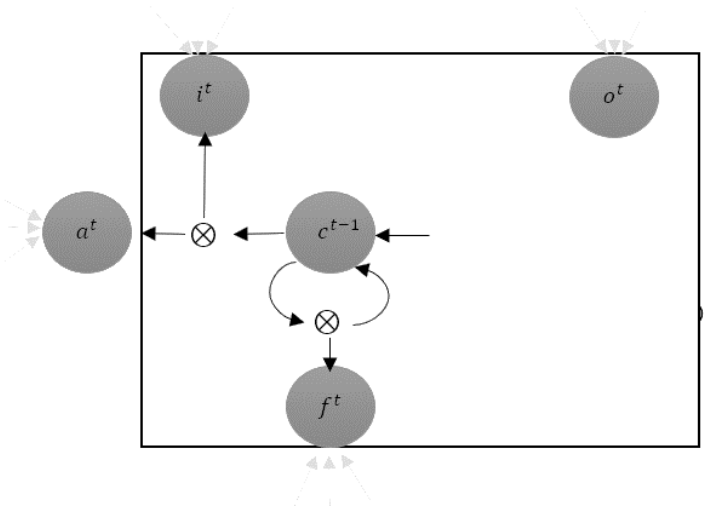
$$\delta a^t = \delta c^t \odot i^t \tag{4.69}$$

untuk:

$$\frac{\partial E}{\partial c_i^{t-1}} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial c_i^{t-1}} = \delta c_i^t \cdot f_i^t \tag{4.70}$$

menjadi:

$$\delta c^{t-1} = \delta c^t \odot f^t \tag{4.71}$$



Gambar 4.20. Proses sel memori yang di*update* pada proses *backward pass*

(3) Komputasi *input* dan *gate* pada fase *backward pass*, dimana ketika *forward pass*, maka untuk mendapatkan δz^t diperlukan parameter $\delta \hat{a}^t, \delta \hat{i}^t, \delta \hat{f}^t,$ dan $\delta \hat{o}^t$ (dijelaskan pada Gambar 4.21):

$$\delta \hat{a}^t = \delta a^t \odot (1 - tanh^2(\hat{a}^t)) \tag{4.72}$$

$$\delta \hat{i}^t = \delta i^t \odot i^t \odot (1 - i^t) \tag{4.73}$$

$$\delta \hat{f}^t = \delta f^t \odot f^t \odot (1 - f^t) \tag{4.74}$$

$$\delta\hat{o}^t = \delta o^t \odot o^t \odot (1 - o^t) \tag{4.75}$$

sehingga menghasilkan:

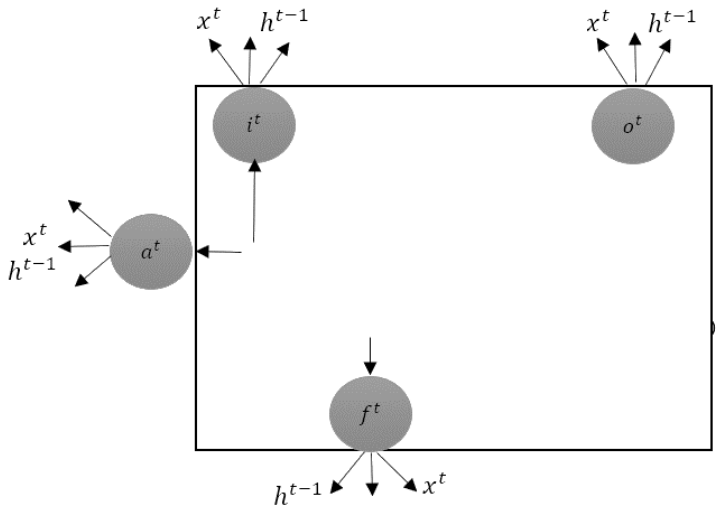
$$\delta z^t = [\delta\hat{a}^t, \delta\hat{\imath}^t, \delta\hat{f}^t, \delta\hat{o}^t]^T \tag{4.76}$$

Kemudian saat perhitungan pada *forward pass* dimana $z^t = W.I^t$, dengan menggunakan parameter δz^t, maka $\delta I^t = W^t \cdot \delta z^t$ dengan:

$$I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \tag{4.77}$$

dimana δh^{t-1} dapat diambil dari δI^t dan menghasilkan:

$$\delta W^t = \delta z^t \cdot (I^t)^T \tag{4.78}$$



Gambar 4.21. Perubahan komputasi *input* dan *gate* pada *backward pass*

Contoh Perhitungan Manual LSTM:

Misal, diketahui inisialisasi bobot:

$$W_a = \begin{bmatrix} 0.45 \\ 0.25 \end{bmatrix}, U_a = [0.15],\ b_a = [0.2]$$

$$W_i = \begin{bmatrix} 0.95 \\ 0.8 \end{bmatrix}, U_i = [0.8],\ \ b_i = [0.65]$$

$$W_f = \begin{bmatrix} 0.7 \\ 0.45 \end{bmatrix}, U_f = [0.1],\ \ b_f = [0.15]$$

$$W_o = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}, U_o = [0.25],\ \ b_o = [0.1]$$

Dengan 2 masukkan data $x_o = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ dengan label 0.5, dan $x_1 = \begin{bmatrix} 0.5 \\ 3 \end{bmatrix}$ dengan label 1.25

Penyelesaian:

Forward pass, dimana $t = 0$, dengan menggunakan persamaan (4.52), (4.53), (4.54), dan (4.55), maka:

$$a_o = \tanh\left([0.45\ 0.25]\begin{bmatrix}1\\2\end{bmatrix} + [0.5][0] + [0.2]\right) = 0.81775$$

$$i_o = \text{sigmoid}\left([0.95\ 0.8]\begin{bmatrix}1\\2\end{bmatrix} + [0.8][0] + [0.65]\right) = 0.96083$$

$$f_o = \text{sigmoid}\left([0.7\ 0.45]\begin{bmatrix}1\\2\end{bmatrix} + [0.1][0] + [0.15]\right) = 0.85195$$

$$o_o = \text{sigmoid}\left([0.6\ 0.4]\begin{bmatrix}1\\2\end{bmatrix} + [0.25][0] + [0.1]\right) = 0.8175$$

Lalu,

$$c_o = (0.81775 * 0.96083) + (0.85195 * 0) = 0.78572$$

Maka dihasilkan *output*,

$$h_o = tanh(0.78572) * 0.8175 = 0.53631$$

Lalu, perlu menghitung nilai kesalahan (*error*) antara ouput prediksi dan aktual. Perhitungan nilai *error* bisa menggunakan fungsi *loss*, missa dengan persamaan:

$E(x, \hat{x}) = \frac{(x,\hat{x})^2}{2}$, dimana turunannya adalah $\partial_x E(x, \hat{x}) = (x - \hat{x})$, sehingga:

$\Delta_0 = 0.53631 - 0.5 = 0.03631$. Jadi *error* nya adalah 0.03631

Backward pass, dimana $t = T$, maka untuk menghitung *Backpropagation Through Time* (BPTT) dimana jika diketahui $T = -0.01828$, $f_T = 0.87030$, dan $\delta c_T = -0.07111$, jadi:

$$\delta o = \Delta_0 + \Delta_T = 0.03631 - 0.01828 = 0.01803$$

$$\delta c_0 = 0.01803 * 0.8175 * \left(1 - tanh^2(0.78572)\right) - 0.07111 * 0.87030 = -0.05349$$

maka dihasilkan $\delta \hat{a}_0 = -0.01703$ $\delta \hat{\imath}_0 = -0.00165$, $\delta \hat{f}_0 = 0$, dan $\delta \hat{o}_0 = 0.00176$.

Hitung dengan persamaan $\delta I^t = W^t \cdot \delta z^t$, maka

$$\delta I^t = \begin{bmatrix} 0.45 & 0.95 & 0.7 & 0.6 \\ 0.25 & 0.8 & 0.45 & 0.4 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = \begin{bmatrix} -0.00817 \\ -0.00487 \end{bmatrix}$$

maka hasil akhir perubahan bobot,

$$\delta W^t = \begin{bmatrix} 0.15 & 0.8 & 0.1 & 0.25 \end{bmatrix} \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = [-0.00343]$$

4.3.5 *Gated Recurrent Unit*

Gated Recurrent Unit (GRU) dimunculkan dalam makalah oleh Cho dkk (Cho et al., 2014) dan Chung dkk (Chung, Gulcehre, Cho, & Bengio, 2014). Keutamaan GRU adalah komputasinya lebih sederhana dari LSTM, dan juga dapat dianggap sebagai variasi pada LSTM karena keduanya dirancang serupa dan dalam beberapa kasus, menghasilkan hasil yang sama baiknya. Gerbang sebelumnya menentukan cara

menggabungkan *input* baru dengan memori sebelumnya dan yang berikutnya menentukan berapa banyak memori itu harus dijaga.

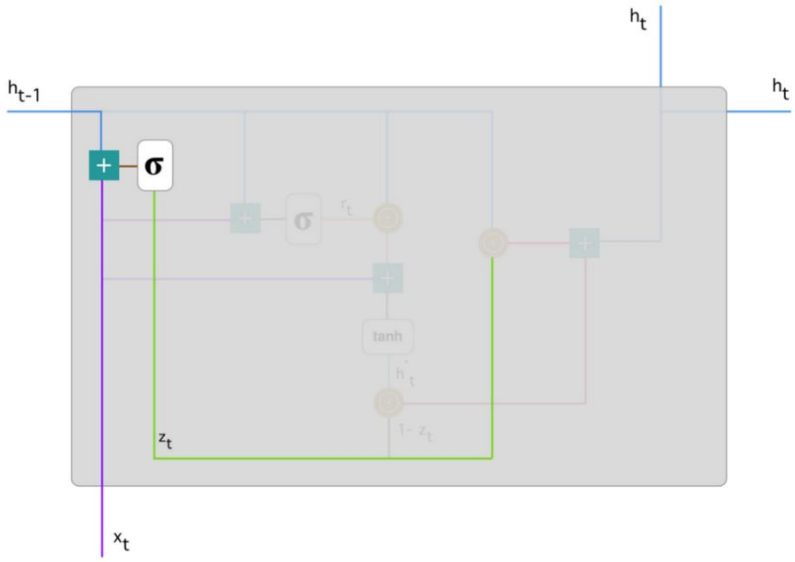
Model GRU juga secara signifikan mengungguli model pembelajaran mesin tradisional yang mengandalkan fitur agregat dan dilatih dengan vektor one-hot atau vektor kode yang dikelompokkan. GRU memiliki gerbang unit untuk memodulasi aliran informasi di dalam unit. GRU berbeda dengan LSTM karena tidak memiliki mekanisme untuk mengontrol sejauh mana mekanisme tersebut menelusuri *state* itu sendiri dan seluruh *state* setiap saat (Stojanovski, Strezoski, Madjarov, & Dimitrovski, 2016).

Berbeda dengan arsitektur LSTM, arsitektur GRU terdiri dari dua gerbang; gerbang *reset*, dan gerbang *update* (Singh et al., 2018). Turunan model matematis GRU adalah sebagai berikut:

1. Menghitung *update gate* z_t untuk langkah waktu *t* menggunakan persamaan:

$$z_t = \sigma(W^z x_t + U^z h_{t-1}) \quad (4.79)$$

x_t merupakan *input* pada jaringan yang dikalikan dengan bobotnya sendiri W^z. Hal yang sama berlaku untuk h_{t-1} yang menyimpan informasi untuk unit $t-1$ sebelumnya dan dikalikan dengan bobotnya sendiri U^z. Kedua hasil ditambahkan bersama-sama dan fungsi aktivasi *sigmoid* diterapkan untuk menekan hasil antara 0 dan 1. Mengikuti skema di atas, maka bisa digambarkan pada Gambar 4.22:



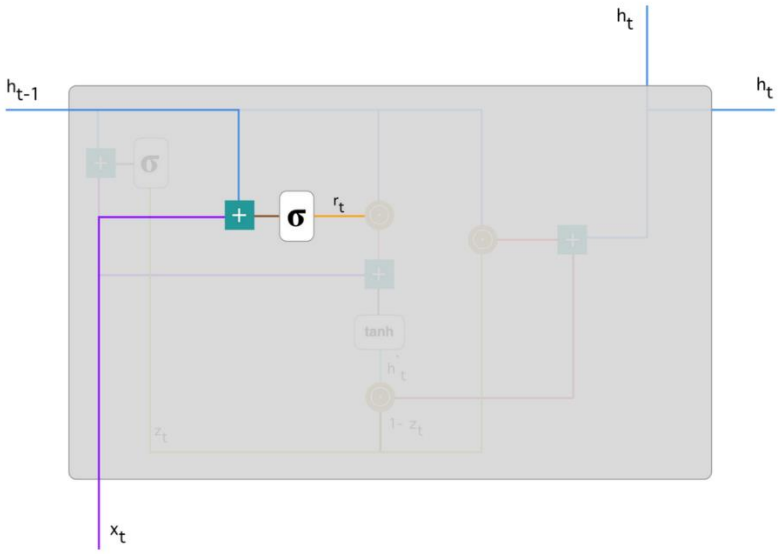
Gambar 4.22. Proses *update gate*

Update gate membantu model untuk menentukan berapa banyak informasi masa lalu (dari langkah waktu sebelumnya) yang harus diteruskan ke masa depan. Itu sangat kuat karena model dapat memutuskan untuk menyalin semua informasi dari masa lalu dan menghilangkan risiko masalah *vanishing gradient.*

2. Menghitung *reset gate*; digunakan untuk memutuskan berapa banyak informasi masa lalu untuk dilupakan dengan persamaan berikut:

$$r_t = \sigma(W^r x_t + U^r h_{t-1}) \tag{4.80}$$

Persamaan ini sama dengan persamaan untuk *update gate*. Perbedaannya terletak pada bobot dan penggunaan gerbang, yang akan terlihat sedikit. Skema proses ini ditunjukkan pada Gambar 4.23 berikut:



Gambar 4.23. Proses *reset gate*

Seperti proses sebelumnya pada *update gate*, kita pasang h_{t-1} pada garis biru dan x_t pada garis ungu, kalikan dengan bobot yang sesuai, jumlah hasilnya dan terapkan fungsi *sigmoid*.

3. Menghitung konten memori atau *state* baru yang akan menggunakan *reset gate* untuk untuk menyimpan informasi yang relevan dari masa lalu di waktu sebelumnya. Persamaannya sebagai berikut:

$$h_t = tanh(Wx_t + r_t \odot Uh_{t-1}) \quad (4.81)$$

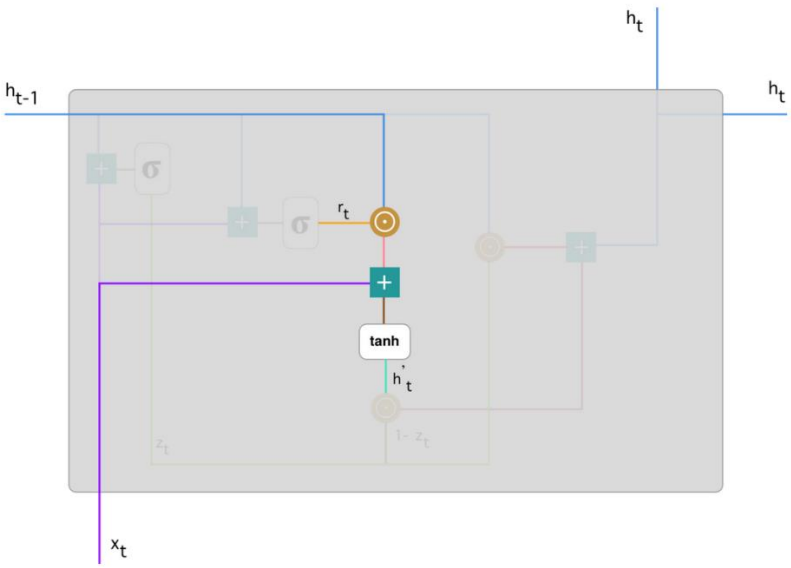
(a) Kalikan *input* x_t dengan bobot W dan h_{t-1} dengan bobot U.

(b) Hitung perkalian Hadamard $\odot$ (berdasarkan elemen) antara *reset gate* r_t dan Uh_{t-1}. Proses ini akan menentukan apa yang harus dihapus dari langkah waktu sebelumnya.

(c) Jumlah hasil dari perhitungan (a) dan (b)

(d) Gunakan fungsi non-linear *tanh*

Skema proses ini ditunjukkan pada Gambar 4.24 berikut:



Gambar 4.24. Proses menghitung konten memori atau *state*

4. Langkah terakhir, jaringan perlu menghitung vektor h_t yang menyimpan informasi untuk unit saat ini dan meneruskannya ke jaringan. Untuk melakukan itu dibutuhkan *update gate*. Proses ini menentukan apa yang dikumpulkan dari konten memori saat ini - h'_t dan apa dari langkah sebelumnya h_{t-1}:

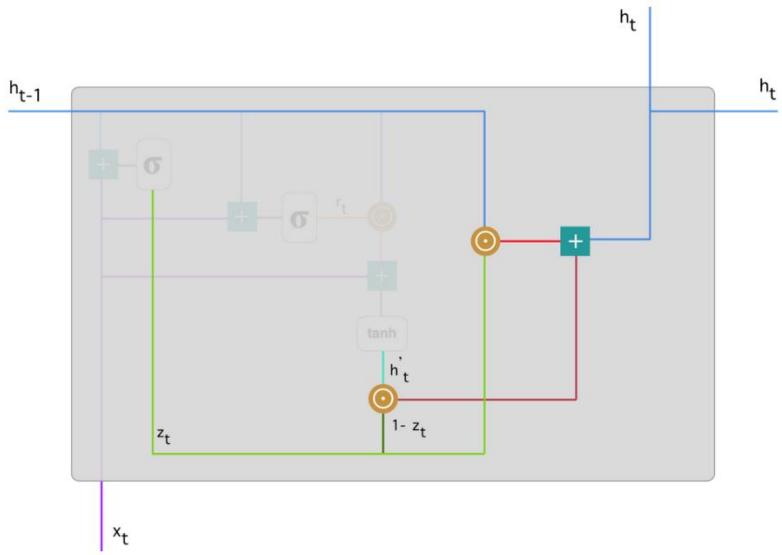
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \qquad (4.82)$$

(a) Terapkan perkalian *elementwise update gate* z_t dan h_{t-1}

(b) Terapkan perkalian *elementwise* ke $(1 - z_t)$ dan h'_t

(c) Jumlahkan hasil dari langkah (a) dan (b)

Skema proses ini ditunjukkan pada Gambar 4.25 berikut:



Gambar 4.25. Proses menghitung memori di *current time step*

Gambar 4.25 melihat bagaimana z_t pada garis hijau digunakan untuk menghitung $(1 - z_t)$ yang dikombinasikan dengan h'_t pada garis hijau terang, menghasilkan hasil di garis merah gelap. z_t juga digunakan dengan h_{t-1} pada garis biru dalam perkalian *elementwise.* Akhirnya, h_t pada garis biru adalah hasil dari penjumlahan dari *output* yang sesuai dengan garis merah terang dan merah gelap.

BAB 5

IMPLEMENTASI DEEP LEARNING MENGGUNAKAN KERAS

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. *Python* mendukung multi paradigma pemrograman pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. *Python* bisa di gunakan pada Keras. Keras merupakan *interface library* yang dibangun untuk mensederhanakan implementasi algoritma-algoritma Deep Learning di atas *TensorFlow*. *TensorFlow* sendiri merupakan platform *High Performance* computing berbasis alur graph.

5.1 *Deep Neural Network*

Pertama kali yang dilakukan adalah *import Library* yang dibutuhkan seperti *Keras* dan *Numpy* (Gambar 5.1).

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
```

Gambar 5.1. *Import Library*

Misalkan, untuk penjelasan ini menggunakan *Pima Indians Diabetes Database* (https://www.kaggle.com/uciml/pima-indians-diabetes-database), untuk mengklasifikasi pasien sehat dan pasien positif Diabetes (Gambar 5.2 dan 5.3).

```
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.data.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
```

Gambar 5.2. *Load Data*

Define Model

```
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Compile Model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit Model

```
# Fit the model
history=model.fit(X, Y, validation_split=0.33, epochs=200, batch_size=10)
```

Gambar 5.3. Arsitektur model neural network standar

5.2 *Convolutional Neural Network*

Proses membangun CNN selalu melibatkan empat langkah utama yaitu:

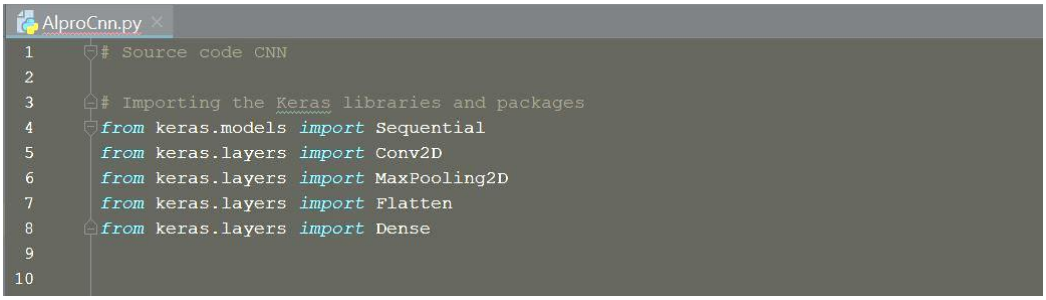
Tahap – 1 : *Convolution*

Tahap – 2 : *Pooling*

Tahap – 3 : *Flattening*

Tahap – 4 : *Fully Connection Layer*

Berikut adalah contoh implementasi *convolutional neural network* menggunakan *python*. Tahap pertama yang dilakukan adalah *import* semua *packages* yang diperlukan pada CNN. *Packages* yang digunakan pada implementasi ini adalah dengan menggunakan *tensorflow backend* (Gambar 5.4).



Gambar 5.4. *Import packages* yang akan digunakan

Berikut penjelasan dari fungsi *packages* yang akan digunakan dalam membangun CNN. Dari *source code* yang telah dibangun.

Baris ke-4. *Import Sequential* untuk menginisialisasi model *neural network*. Ada dua cara dasar untuk melakukan inisialisasi *neural network* yaitu dengan urutan lapisan atau grafik.

Baris ke-5. *Import* Conv2D untuk melakukan operasi konvolusi pada langkah pertama CNN pada gambar. Karena pada kasus ini hal yang dilakukan adalah gambar sehingga mengguakan Conv2D jika hal yang dilakukan bersifat video mungkin harus menggunakan Conv3D.

Baris ke-6. *Import MaxPooling* 2D yang digunakan untuk melakukan operasi pooling yang dilakukan pada tahap ke 2 proses membangun CNN. Ada berbagai jenis operasi *pooling*, tapi pada kasus ini kami menggunakan *MaxPooling* untuk membangun CNN.

Baris ke-7. *Import Flatten* yang digunakan untuk proses *flatten* untuk mengubah semua *array* 2 dimensi yang dihasilkan menjadi vektor linear kontinu.

Baris ke-8. *Import Dense* yang digunakan untuk melakukan *full connection* dari *neural network* yang merupakan tahap ke 4 dalam proses membangun CNN.

Sekarang membuat *objek sequential class* (Gambar 5.5):

```
11    classifier = Sequential()
```

Gambar 5.5. *Objek sequential class*

Selanjutnya adalah melakukan proses *convolution* berikut *source code* yang telah dibangun. Diketahui pada contoh kasus diatas *input layer* yang digunakan adalah 10 x 10 dan filter (kernel) yang digunakan adalah 3 x 3 (Gambar 5.6).

```
13    # Tahap 1 - Convolution
14    classifier.add(Conv2D(8, (3, 3), input_shape = (10, 10, 3), activation = 'relu'))
```

Gambar 5.6. Tahap konvolusi

Diketahui pada tahap sebelumnya adalah membuat objek *neural network* menjadi *sequential* kemudian melakukan proses konvolusi dengan menggunakan *packages* Conv2D. Dimana diketahui pada kasus diatas gambar *input* yang digunakan adalah 10 x 10 dan "3" merupakan proses RGB dengan jumlah 32 dengan ukuran 3 x 3. Setelah itu melakukan proses fungsi aktivasi yang digunakan pada implementasi ini adalah ReLU.

Tahap selanjutnya adalah melakukan proses operasi *pooling* pada *feature map* yang telah dihasilkan setelah operasi konvolusi. Seperti yang telah dijelaskan pada teori diatas bahwa tujuan utama proses *pooling* adalah untuk mengurangi ukuran gambar sebanyak mungkin. Berikut *source code* yang telah dimplementasikan (Gambar 5.7).

```
17  # Tahap 2 - Pooling
18  classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Gambar 5.7. Tahap *pooling*

Pada tahap ini, mengambil nilai *MaxPooling* pada matriks 2 x 2.

Berikut *source code* proses melakukan *convolution* kembali (Gambar 5.8).

```
21  # Tahap 2 - Convolutional Layer
22  classifier.add(Conv2D(8, (3, 3), activation = 'relu'))
23  classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Gambar 5.8. *Convolutional layer code*

Tahap selanjutnya adalah melakukan proses konversi semua gambar yang dikumpulkan menjadi vektor dengan menggunakan *flatten*. Pada proses *flatten* ini yang dilakukan adalah mengambil *array* dari 2D menggabungkan *pixel* gambar dan mengonversinya menjadi vektor tunggal. Berikut *source code flatten* yang dibangun (Gambar 5.9).

```
26  # Tahap 3 - Flattening
27  classifier.add(Flatten())
```

Gambar 5.9. *Flattening code*

Tahap selanjutnya adalah *fully connected layer* yang akan mengubungkan *set node* yang kita dapatkan setelah melakukan proses *flatten. Set node* ini digunakan sebagai *input layer*. Karena pada proses ini akan ada proses *input layer* dan *output layer*. Berikut *source code* yang dibangun pada proses *fully connected layer* (Gambar 5.10).

```
30  # Tahap 4 - Fully Connection Layer
31  classifier.add(Dense(units = 128, activation = 'relu'))
32  classifier.add(Dense(units = 1, activation = 'sigmoid'))
33  # Proses CNN
34  classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Gambar 5.10. *Fully connected layer code*

Saat menginisialisasi *output layer* yang seharusnya hanya berisi satu node, karena ini adalah proses klasifikasi biner. Node ini diharapkan memberi keluaran berupa nilai biner dari gambar. Fungsi aktivasi *sigmoid* untuk final *layer*. Proses membangun CNN pun telah selesai.

5.3 *Recurrent Neural Network*

Berikut struktur RNN, LSTM dan GRU dengan menggunakan bahasa pemrograman *phyton* pada *Keras* (Gambar 5.11 – 5.17):

Standard RNN:

```
# create model
model = Sequential()
model.add(SimpleRNN(..., input_shape=(...)))
model.add(Dense(...))
```

Gambar 5.11. Model *Standard* RNN

***Long Short-Term Memory* (LSTM):**

1. *Standard* LSTM

```
# create model
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
model.add(Dense(...))
```

Gambar 5.12. Model *standard* LSTM

2. *Stacked*-LSTM

```
# create model
model = Sequential()
model.add(LSTM(...,return_sequences=True, input_shape=(...)))
model.add(LSTM(...))
model.add(Dense(...))
```

Gambar 5.13. Model *stacked* LSTM

3. *Convolutional* LSTM

```
# create model
model = Sequential()
model.add(TimeDistributed(Conv2D(...))
model.add(TimeDistributed(MaxPooling2D(...)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(...))
model.add(Dense(...))
```

Gambar 5.14. Model *convolutional* LSTM

4. *Encoder – Decoder* LSTM

```
# create model
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
model.add(RepeatVector(...))
model.add(LSTM(..., return_sequences=True))
model.add(TimeDistributed(Dense(...)))
```

Gambar 5.15. Model *encoder-decoder* LSTM

5. *Bidirectional* LSTM

```
# create model
model = Sequential()
model.add(Bidirectional(LSTM(...), input_shape=(...)))
```

Gambar 5.16. Model *encoder-decoder* LSTM

***Gated Recurrent Unit* (GRU):**

```
# create model
model = Sequential()
model.add(GRU(..., input_shape=(...)))
model.add(Dense(...))
```

Gambar 5.17. Model *encoder-decoder* LSTM

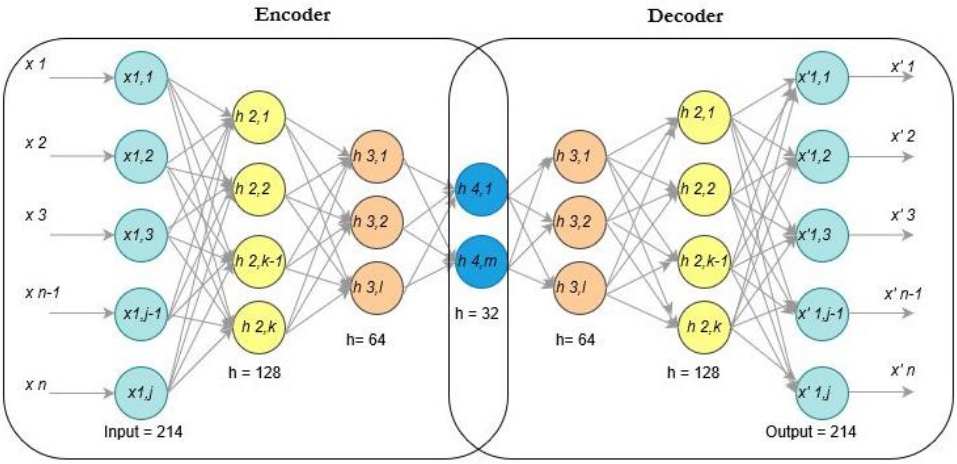
5.4 Studi Kasus *Deep Learning*

5.4.1 Implementasi *Autoencoder-Deep Neural Network*

Dataset yang digunakan merupakan dataset *malware ransomware* dan *benign* dari *drebin project* Arp dkk. (2014). Dataset *malware* yang digunakan dalam penelitian ini berjumlah 94.520 data *malware ransomware*. Sedangkan dataset *benign* yang digunakan berjumlah 161.092 data *benign*. Data awal akan dibagi yaitu data untuk pelatihan dan validasi data. Pembagian data dengan skenario *split data* 70% data untuk pelatihan dan 30% untuk pengujian. Data pelatihan berjumlah 171260 sedangkan data validasi berjumlah 84352.

***Autoencoder*:**

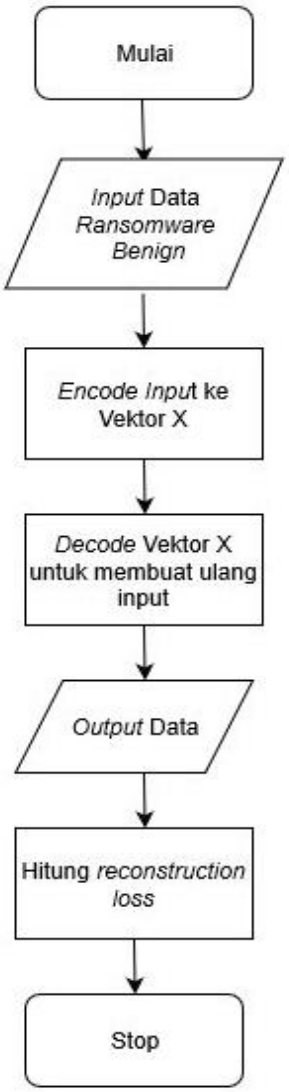
Berikut arsitektur *autoencoder* yang diusulkan pada Gambar 5.18 di bawah ini yang terdiri dari 214 atribut pada *input*, 3 *hidden layer* dan *output*. Dimana akan dibagi menjadi proses *encoder* dan *decoder*.



Gambar 5.18. Arsitektur *autoencoder* yang diusulkan

Gambar 5.18 menunjukkan arsitektur *autoencoder* yang diusulkan yaitu *input* yang terdiri dari 214 atribut. Dan 3 *hidden layer* dimana *hidden layer* pertama terdiri dari 128 *neuron*, *hidden layer* kedua 64 *neuron* dan *hidden layer* ketiga 32 *neuron*.

Berikut proses yang dilakukan dalam *autoencoder* yang digambarkan dalam bentuk *flowchart* pada Gambar 5.19.



Gambar 5.19. Flowchart *Autoencoder*

Proses *autoencoder* diawali dengan *input* data *ransomware* dan *benign.* Lalu *input* di *encode* ke vektor x. Setelah di *encode* maka masuk ke

proses *decode* untuk memuat ulang *input* yang merupakan *output* data yang sudah direduksi dimensinya. Setelah itu hitung *reconstruction loss*.

Klasifikasi *Deep Neural Network*:

Arsitektur DNN yang dibuat memiliki layer *input*, *layer hidden* dan *layer output*. *Layer input* terdiri dari 214 atribut. Ada 5 *layers hidden* dengan menggunakan fungsi aktivasi ReLu. Untuk *layer hidden* pertama terdiri dari 150 *neuron*. *Layer hidden* kedua terdiri dari 120 *neuron*. *Layer hidden* ketiga 100 *neuron*. *Layer hidden* keempat 50 *neuron*. *Layer hidden* kelima 20 *neuron* dan *layer output* satu *neuron* dikarenakan fungsi aktivasi pada *layer output* adalah *sigmoid* dan algoritma yang digunakan adalah *backpropagation*. Saat proses pelatihan data, jumlah data akan dibagi yaitu jumlah data untuk proses pelatihan (*training*) adalah 70% dan 30% untuk pengujian (*testing*) (Tabel 5.1).

Tabel 5.1

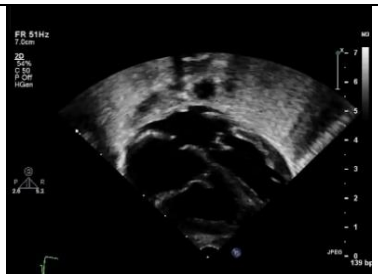
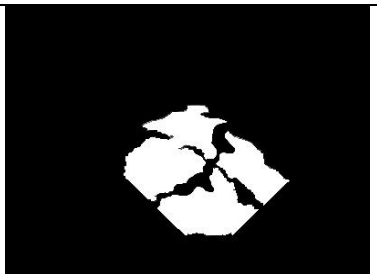
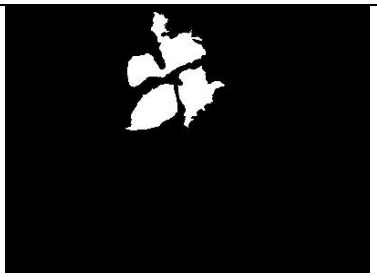
Hasil Performa Validasi Menggunakan DNN-*Autoencoder*

Autoencoder	Pelatihan	Pengujian (%)				
	Akurasi	Akurasi	Presisi	Sensitivitas	Spesifisitas	F1-Score
70 : 30	99,93	99,96	99,91	99,98	99,53	98,25

5.4.2 Implementasi Segmentasi Citra Ultrasonografi (USG) Berbasis *Convolutional Neural Network*

Dalam melakukan proses klasifikasi penyakit jantung bayi berbasis citra terdapat salah satu proses yang cukup penting yaitu segmentasi 4 chamber jantung. Proses segmentasi 4 *chamber* adalah proses cukup penting karena dapat membantu mengurangi kompleksitas dari proses klasifikasi dengan mengurangi ruang lingkup pencarian objek. Segmentasi 4 *chamber* termasuk ke dalam proses segmentasi semantik karena model yang dibangun mencoba untuk melakukan proses pelabelan terhadap setiap *pixel* dalam gambar dengan kelas yang sesuai (*pixel background* atau *foreground*). Penelitian ini menggunakan dataset primer untuk melakukan proses segmentasi semantik.

Dataset yang digunakan dalam penelitian ini menggunakan video yang diambil dari sebuah situs web berbagi video (Youtube) dengan berfokus kepada 3 jenis penyakit yaitu ASD, VSD, dan Normal. Selain itu jenis pengambilan gambar yang digunakan adalah 4 *chamber view*. Lebih lanjut untuk melakukan proses segmentasi diperlukan citra ground truth yang digunakan sebagai *baseline* informasi untuk segmentasi. Gambar 5.20 menunjukkan contoh dari dataset yang digunakan (orisinal dan ground truth).

Kelas	Gambar Orisinal	Gambar *Ground Truth*
Normal		
ASD		
VSD		

Gambar 5.20. Tiga kondisi kelainan jantung pada janin

Metode Segmentasi (Arsitektur U-NET):

Pada penelitian ini metode segmentasi yang digunakan berbasis *deep learning* menggunakan *Convolutional Neural Network* (CNN). Arsitektur CNN yang digunakan dalam melakukan segmentasi semantik adalah U-

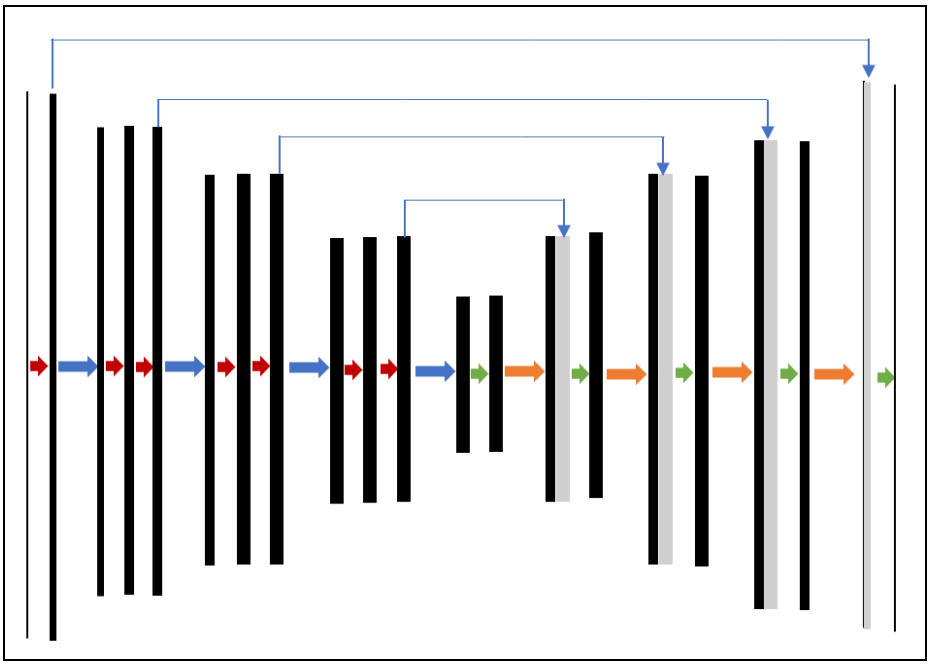
Net. Arsitektur ini telah terbukti dapat melakukan proses segmentasi semantik pada dataset medis. Beberapa kasus yang berhasil diselesaikan menggunakan arsitektur U-Net seperti segmentasi tumor otak, segmentasi sel, serta segmentasi citra satelit. Bentuk arsitektur U-Net ditunjukkan pada Gambar 5.21.

Arsitektur U-NET merupakan arsitektur berjenis *end-to-end fully convolutional network* (FCN), yaitu hanya berisi lapisan konvolusi tanpa mengandung fully-connected (*dense*) sehingga arsitektur ini dapat menerima gambar dengan ukuran yang berbeda. Dalam arsitektur yang akan digunakan terdapat dua bagian utama, bagian enkode dan dekoder.

Bagian pertama adalah *contraction path* (disebut juga enkode) berfungsi untuk menangkap konteks yang terdapat di dalam gambar. Pada bagian pertama, citra yang berukuran 128 x 128 x 1 akan di *downsample* sehingga dapat menghasilkan konteks dari citra yang diteliti. Di dalam arsitektur enkode terdiri dari beberapa layer konvolusi dan *max pooling*. Ukuran *kernel* konvolusi yang digunakan pada bagian ini sebesar 3 x 3. Kemudian setiap operasi konvolusi yang dilakukan akan selalu diikuti oleh operasi non-linearity ReLU. Selain itu, pada bagian ini juga terdapat operasi *pooling* yang berukuran 2 x 2 dengan pergeseran sebanyak 2 kali. Ukuran terakhir dari tahapan dekoder adalah 8 x 8 x 256.

Bagian kedua adalah *symmetric expanding path* (disebut juga dekoder) yang digunakan untuk melokalisasi objek menggunakan transformasi konvolusi. Tahapan dekoder berisi operasi *upsampling* dari hasil dekoder. Pada operasi dekoder, ukuran gambar akan meningkat secara bertahap. Selain itu kedalaman dari citra akan berkurang secara bertahap

mulai dari 8x8x256 hingga 128x128x1. Secara intuitif, proses dekoder memulihkan informasi yang dihasilkan dari proses enkode dengan cara melakukan tahapan up-sampling secara perlahan. Kemudian, untuk mendapatkan hasil segmentasi objek yang lebih baik, pada setiap layer dekode, dilakukan proses *skip-connection*. Proses ini dilakukan dengan menyatukan *output* dari lapisan konvolusi pada bagian dekoder, yang kemudian ditransformasikan dengan *feature-map* dari tahapan enkode pada level yang sama.

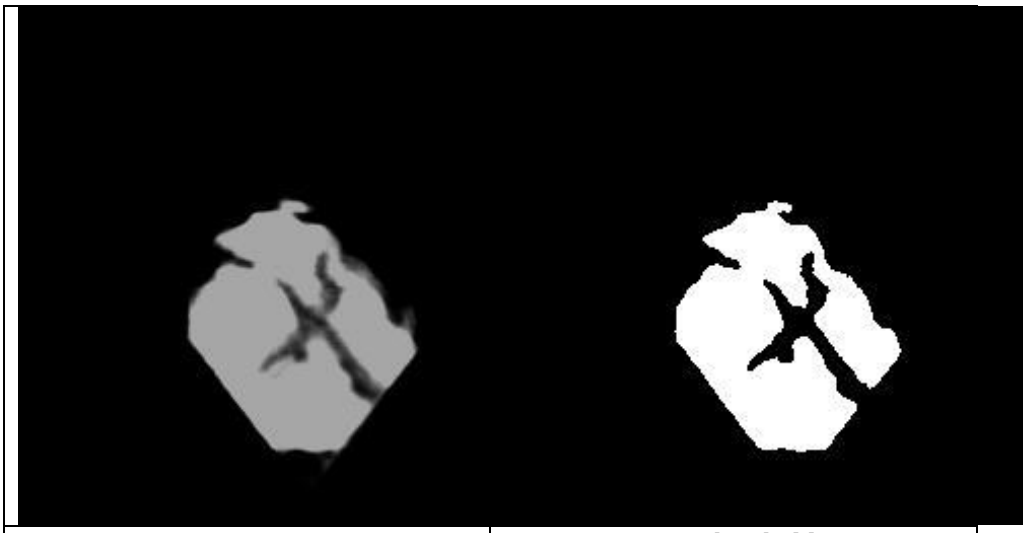


: 3 x 3 Konvolusi 2D + ReLU (*Pre-Training*)
: 2 x 2 *Max Polling*
: 3 x 3 Konvolusi 2D + ReLU
: 3 x 3 Transpose Konvolusi 2D (Pergeseran 2x) + ReLU

Gambar 5.21. Arsitektur U-NET

Hasil Percobaan pada studi kasus ini bisa dilihat pada Gambar 5.22 berikut;

Perbandingan hasil post-processing jika menggunakan hanya satu kelas ASD:



Tanpa Post-Processing	Fix Threshold
Otsu Threshold	Gaussian Threshold

Gambar 5.22. Hasil percobaan 1 kelas ASD

5.4.3 Implementasi Klasifikasi Sinyal Elektrokardiogram Berbasis *Long Short-Term Memory*

Studi kasus ini menggunakan dataset PhysioNet: *The PTB Diagnostic* dikarenakan basis data ini berisi sinyal EKG yang merepresentasikan kondisi jantung normal dan 9 kelainan jantung; yaitu *Myocardial Infarction, Cardiomyopathy*, *Heart Failure*, *Dysrhythmia*, *Bundle branch block, Myocardial hypertrophy*, *Valvular heart disease*, *Myocarditis,* dan *Miscellaneous*. Untuk studi kasus ini hanya menggunakan kelas diagnostik jantung normal (*healthy controls*), infark miokard, kardiomiopati, blokade cabang berkas, dan disritmia.

Jumlah data sinyal EKG dibagi 90% untuk set data pelatihan (*training*) dan 10% untuk pengujian (*testing*) dari total 13.610 data sinyal yang telah disegmentasi per *window sized* selama 4 detik. Set data pelatihan yang dipakai tidak digunakan untuk pengujian jaringan LSTM, dan sebaliknya. Dari 90% data *training*, 10% digunakan untuk proses validasi data. Jumlah 13.610 data dipisah secara acak, dengan *split data* otomatis (*shuffled sampling).*

Proses pelatihan LSTM dilatih dengan menggunakan *hyperparameter* fungsi aktivasi *tanh* dan *sigmoid* di *hidden gates, softmax* pada *output* akhir, *optimizer* Adam dengan *learning rate* 0,001, dengan jumlah 100 *epochs*. Hasil proses pelatihan dengan menggunakan data pelatihan ditunjukkan pada Tabel 5.2 dan 5.3. Pengujian model LSTM dengan data uji juga dilakukan untuk mengetahui hasil kinerja evaluasi pada proses klasifikasi multikelas antara kondisi jantung normal, infark miokard, kardiomiopati, blokade cabang berkas, dan disritmia yang juga

ditunjukkan pada Tabel 5.3. Parameter evaluasi yang digunakan adalah akurasi, sensitivitas, spesifisitas, presisi dan F1-score.

Tabel 5.2

Hasil Kinerja Evaluasi Pelatihan Model LSTM

Parameter evaluasi (%)	Kondisi Jantung					Rata-rata
	Normal	Infark Miokard	Kardio-miopati	Blokade cabang berkas	Disritmia	
Akurasi	97,53	96,95	99,72	99,46	99,42	98,61
Sensitivitas	93,87	97,25	97,49	96,66	91,74	95,40
Spesifisitas	98,31	96,09	99,89	99,55	99,57	98,68
Presisi	92,21	98,63	94,09	86,65	80,97	90,51
F1-Score	93,03	97,94	95,76	91,38	86,02	92,82

Tabel 5.3

Hasil Kinerja Evaluasi Pengujian Model LSTM

Parameter evaluasi (%)	Kondisi Jantung					Rata-rata
	Normal	Infark Miokard	Kardio-miopati	Blokade cabang berkas	Disritmia	
Akurasi	96,66	96,13	99,69	99,69	99,38	98,31
Sensitivitas	92,98	96,08	99,80	98,80	94,05	96,34
Spesifisitas	97,34	96,26	99,68	99,73	99,59	98,52
Presisi	86,65	98,58	92,93	93,18	89,77	92,22
F1-Score	89,70	97,32	96,34	95,91	91,86	94,22

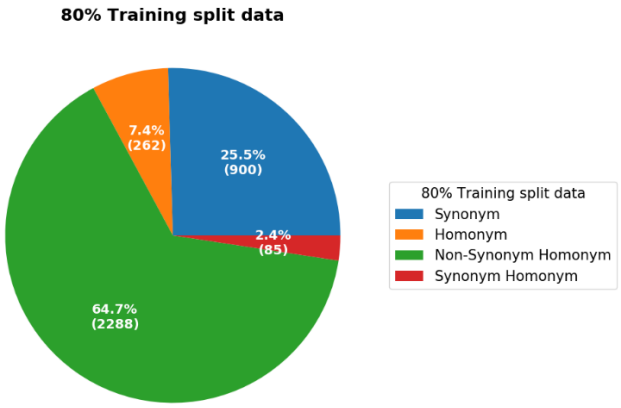
5.4.4 Implementasi *Deep Neural Network* pada Kasus *Author Name Disambiguation* (AND)

Sampai saat ini, sebagian besar perpustakaan digital ilmiah memiliki sejumlah besar data publikasi. Perpustakaan-perpustakaan ini menyediakan fitur dan layanan untuk memudahkan akademisi mencari

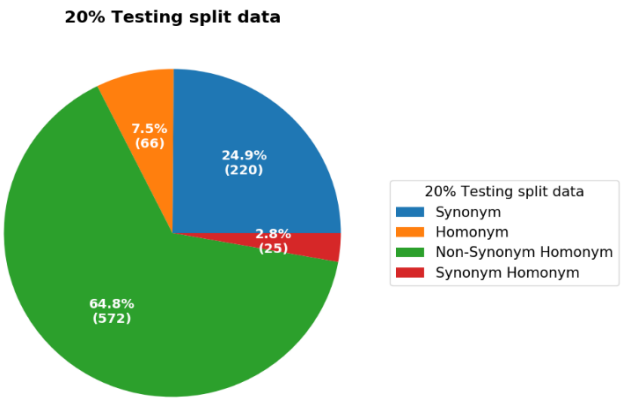
literatur. Data publikasi dapat digunakan untuk mengukur kualitas dan dampak publikasi dari kelompok masyarakat atau individu. Informasi ini membantu lembaga pendanaan untuk memberikan keputusan hibah bagi individu.

Mengenali data publikasi yang dimiliki oleh seorang individu itu menantang. Banyak penulis menulis nama mereka dalam publikasi dengan presentasi yang berbeda (sinonim). Masalah sinonim disebabkan oleh berbagai hal seperti kesalahan ejaan dan perubahan nama. Selain itu, banyak penulis memiliki nama yang sama (homonim). Masalah sinonim dan homonim adalah tantangan utama untuk mengenali kepengarangan publikasi.

Dataset dibagi menjadi 80% data pelatihan dan 20% data pengujian (Gambar 5.23). Dengan mempertimbangkan jumlah referensi penulis dan distribusi data pelatihan dan pengujian, jumlah kelas yang digunakan dalam klasifikasi ditentukan. Penulis yang memiliki referensi lebih dari lima referensi ditetapkan sebagai satu kelas, dan yang lainnya dihapus. Jumlah kelas ditetapkan sebagai 266 kelas penulis.



(a)



(b)

Gambar 5.23. Pembagian data sinonim dan homonim penulis (a) 80% data pelatihan, dan (b) 20% data pengujian.

Hasil percobaan studi kasus untuk kelas homonim, sinonim, homonim-sinonim, dan bukan homonim-sinonim pada publikasi ilmiah bisa dilihat pada Tabel 5.4 berikut;

Tabel 5.4

Hasil kinerja *Deep Neural Networks* Untuk Implementasi *Author Name Disambiguation*

Kelas	Akurasi	Presisi	Sensitivitas	F1-Score
Semua Data	0.9999	0.9799	0.9821	0.9791
Homonim	0.9918	0.7854	0.7969	0.7852
Sinonim	0.9986	0.9072	0.9232	0.9094
Homonim-Sinonim	0.9683	0.6250	0.6429	0.6327
Bukan Homonim-Sinonim	1.0000	1.0000	1.0000	1.0000

BAB 6
DEEP LEARNING DAN BIG DATA

Analisis *Big Data* dan *Deep Learning* adalah dua fokus utama ilmu data. *Big Data* telah menjadi penting karena banyak organisasi baik publik maupun swasta telah mengumpulkan sejumlah besar informasi spesifik domain, yang dapat berisi informasi berguna tentang masalah seperti intelijen nasional, keamanan dunia maya, deteksi penipuan, pemasaran, dan informatika medis. Perusahaan seperti Google dan Microsoft menganalisis sejumlah besar data untuk analisis dan keputusan bisnis, yang mempengaruhi teknologi yang ada dan yang akan datang. Algoritma *Deep Learning* mengekstraksi abstraksi tingkat tinggi dan kompleks sebagai representasi data melalui proses pembelajaran hierarkis.

Abstraksi kompleks dipelajari pada level tertentu berdasarkan abstraksi yang relatif lebih sederhana yang diformulasikan pada level sebelumnya dalam hierarki. Manfaat utama *Deep Learning* adalah analisis dan pembelajaran sejumlah besar data tanpa pengawasan, menjadikannya alat berharga untuk analisis *Big Data* di mana data mentah sebagian besar tidak berlabel dan tidak dikategorikan.

6.1 Analisis Big Data

Big Data umumnya mengacu pada data yang melebihi kapasitas penyimpanan, pemrosesan, dan komputasi tipikal dari basis data konvensional dan teknik analisis data (Gambar 6.1) (Najafabadi et al., 2015). Sebagai sumber daya, *Big Data* membutuhkan alat dan metode yang dapat diterapkan untuk menganalisis dan mengekstrak pola dari data skala besar. Munculnya *Big Data* telah disebabkan oleh peningkatan kemampuan penyimpanan data, peningkatan daya pemrosesan komputasi, dan ketersediaan *volume* data yang meningkat, yang memberi organisasi lebih banyak data daripada yang harus diproses sumber daya dan teknologi komputasi. Selain *volume* besar data yang jelas, *Big Data* juga terkait dengan kompleksitas spesifik lainnya, sering disebut sebagai empat V: *Volume*, *Variety*, *Velocity*, dan *Veracity* (Dumbill, 2012).

Gambar 6.1 Hubungan Ilmu Data dan Big Data

Seperti yang dinyatakan sebelumnya, algoritma *Deep Learning* mengekstraksi representasi abstrak bermakna dari data mentah melalui penggunaan pendekatan pembelajaran multi-level hirarkis, di mana pada level yang lebih tinggi representasi yang lebih abstrak dan kompleks dipelajari berdasarkan pada konsep yang kurang abstrak dan representasi di tingkat lebih rendah dari hirarki pembelajaran. Sementara *Deep Learning* dapat diterapkan untuk belajar dari data berlabel jika tersedia

dalam jumlah yang cukup besar, itu menarik untuk belajar dari sejumlah besar data tanpa label/tanpa pengawasan (Bengio & Lecun, 2007)(Bengio, Courville, & Vincent, 2013), membuatnya menarik untuk mengekstraksi representasi yang bermakna dan pola dari *Big Data*.

6.2 Aplikasi *Deep Learning* pada *Big Data*

Mempertimbangkan masing-masing dari empat V karakteristik *Big Data*, yaitu, *Volume, Variety, Velocity,* dan *Veracity*, algoritma dan arsitektur *Deep Learning* lebih tepat untuk mengatasi masalah yang terkait dengan *Volume* dan *Variety* analisis *Big Data*. *Deep Learning* secara inheren mengeksploitasi ketersediaan sejumlah besar data, misalnya *Volume* dalam *Big Data*, di mana algoritma dengan hierarki pembelajaran dangkal gagal mengeksplorasi dan memahami kompleksitas pola data yang lebih tinggi. Terlebih lagi, karena *Deep Learning* berurusan dengan abstraksi data dan representasi, sangat mungkin cocok untuk menganalisis data mentah yang disajikan dalam format yang berbeda dan atau dari sumber yang berbeda, yaitu *Variasi* dalam data besar, dan dapat meminimalkan kebutuhan *input* dari pakar manusia untuk mengekstraksi fitur dari setiap tipe data baru yang diamati dalam *Big Data*.

Sementara menyajikan tantangan berbeda untuk pendekatan analisis data yang lebih konvensional, analisis *Big Data* menyajikan peluang penting untuk mengembangkan algoritma dan model baru untuk mengatasi masalah spesifik terkait *Big Data*. Konsep *Deep Learning* menyediakan satu tempat solusi seperti itu bagi para ahli dan praktisi analitik data. Misalnya, representasi yang diekstraksi oleh *Deep Learning* dapat

dianggap sebagai sumber praktis pengetahuan untuk pengambilan keputusan, pengindeksan semantik, pencarian informasi, dan untuk tujuan lain dalam analisis *Big Data*, dan sebagai tambahan, teknik pemodelan linier sederhana dapat dipertimbangkan untuk analisis *Big Data* ketika data kompleks direpresentasikan dalam bentuk abstraksi yang lebih tinggi.

Berbeda dengan pembelajaran mesin yang lebih konvensional dan algoritme rekayasa fitur, *Deep Learning* memiliki keunggulan berpotensi memberikan solusi untuk mengatasi analisis data dan masalah pembelajaran yang ditemukan dalam *volume* besar data *input*. Lebih khusus, ini membantu dalam secara otomatis mengekstraksi representasi data kompleks dari *volume* besar data yang tidak diawasi. Ini menjadikannya alat yang berharga untuk analisis *Big Data*, yang melibatkan analisis data dari kumpulan data mentah yang sangat besar yang umumnya tidak diawasi dan tidak dikategorikan. Pembelajaran hierarkis dan ekstraksi berbagai tingkat abstraksi yang kompleks, data dalam *Deep Learning* memberikan tingkat penyederhanaan tertentu untuk tugas analisis *Big Data*, terutama untuk menganalisis *volume* besar data, pengindeksan semantik, penandaan data, pencarian informasi, dan tugas diskriminatif seperti klasifikasi dan prediksi.

Dalam konteks membahas karya-karya utama dalam literatur dan memberikan wawasan kami tentang topik-topik spesifik, penelitian ini berfokus pada dua bidang penting yang terkait dengan *Deep Learning* dan *Big Data*: (1) penerapan algoritma dan arsitektur *Deep Learning* untuk analisis *Big Data*, dan (2) bagaimana karakteristik dan masalah

analisis *Big Data* menimbulkan tantangan unik untuk mengadaptasi algoritma *Deep Learning* untuk masalah-masalah tersebut. Survei yang ditargetkan untuk literatur penting dalam penelitian dan aplikasi *Deep Learning* ke berbagai domain disajikan dalam makalah ini sebagai sarana untuk mengidentifikasi bagaimana *Deep Learning* dapat digunakan untuk berbagai tujuan dalam analisis *Big Data*.

Rendahnya kematangan bidang *Deep Learning* menuntut penelitian lebih lanjut yang luas. Secara khusus, lebih banyak pekerjaan diperlukan tentang bagaimana kita dapat mengadaptasi algoritma *Deep Learning* untuk masalah yang terkait dengan *Big Data*, termasuk dimensi tinggi, analisis data *streaming*, skalabilitas model *Deep Learning*, peningkatan formulasi abstraksi data, komputasi terdistribusi, pengindeksan semantik, penandaan data, pengambilan informasi, kriteria untuk mengekstraksi representasi data yang baik, dan adaptasi domain. Pekerjaan di masa depan harus fokus pada mengatasi satu atau lebih dari masalah yang sering terlihat dalam *Big Data*, sehingga memberikan kontribusi untuk penelitian mendalam dan penelitian analisis *Big Data*.

INDEKS

M

N

O

P

R

S

T

U

V

W

DAFTAR PUSTAKA

Abid, A., Fatih, M., & James, B. (2019). Concrete Autoencoders for Differentiable Feature Selection and Reconstruction. *International Conference on Machine Learning*.

Afshar, A., Massoumi, F., Afshar, A., & Mariño, M. A. (2015). State of the art review of ant colony optimization applications in water resource management. *Water Resources Management*, *29*(11), 3891–3904.

Akhouayri, E.-S., Agliz, D., Zonta, D., Atmani, A., & others. (2015). A fuzzy expert system for automatic seismic signal classification. *Expert Systems with Applications*, *42*(3), 1013–1027.

AlJadda, K., Korayem, M., Ortiz, C., Grainger, T., Miller, J. A., Rasheed, K. M., … others. (2018). Mining massive hierarchical data using a scalable probabilistic graphical model. *Information Sciences*, *425*, 62–75.

Amato, F., López, A., Peña-Méndez, E. M., Va\vnhara, P., Hampl, A., & Havel, J. (2013). *Artificial neural networks in medical diagnosis*. Elsevier.

Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2013). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket Daniel. *Choice Reviews Online*, *45*(02), 45-0765-45–0765. https://doi.org/10.5860/choice.45-0765

Arsene, O., Dumitrache, I., & Mihu, I. (2015). Expert system for

medicine diagnosis using software agents. *Expert Systems with Applications*, *42*(4), 1825–1834.

Attia, M., Hossny, M., Nahavandi, S., & Yazdabadi, A. (2017). Skin melanoma segmentation using recurrent and convolutional neural networks. *Proceedings - International Symposium on Biomedical Imaging*, 292–296. https://doi.org/10.1109/ISBI.2017.7950522

Baldi, P. (2012). Autoencoders , Unsupervised Learning , and Deep Architectures. *Journal of Machine Learning Research: Workshop and Conference Proceedings*, *27*, 37–50.

Bazi, Y., & Melgani, F. (2006). Toward an optimal SVM classification system for hyperspectral remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, *44*(11), 3374–3385.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bengio, Y., & Delalleau, O. (2011). On the Expressive Power of Deep Architectures. *Lecture Notes in Computer Sciences*, *6925*, 18–36.

Bengio, Y., & Lecun, Y. (2007). Scaling Learning Algorithms towards AI. In *Large-Scale Kernel Machines* (pp. 1–41). MIT Press.

Bullinaria, J. A. (2015). *Recurrent neural networks*.

Caniani, D., Lioi, D. S., Mancini, I. M., & Masi, S. (2011). Application of fuzzy logic and sensitivity analysis for soil contamination hazard classification. *Waste Management*, *31*(3), 583–594.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *ArXiv Preprint ArXiv:1406.1078*.

Choi, K., Fazekas, G., Sandler, M., & Cho, K. (2017). Convolutional recurrent neural networks for music classification. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2392–2396.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv Preprint ArXiv:1412.3555*.

Courville, A., Bengio, Y., & Goodfellow, I. (2016). *Deep Learning*. MIT Press.

Daamouche, A., Melgani, F., Alajlan, N., & Conci, N. (2013). Swarm optimization of structuring elements for VHR image classification. *IEEE Geoscience and Remote Sensing Letters*, *10*(6), 1334–1338.

den Bergh, F., & Engelbrecht, A. P. (2004). A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, *8*(3), 225–239.

Deng, J. (2009). A large-scale hierarchical image database. *Proc. of IEEE Computer Vision and Pattern Recognition, 2009*.

Dey, A. (2016). Machine Learning Algorithms: A Review. *Vol*, *7*, 1174–1179.

Dumbill, E. (2012). What is big data? An introduction to the big data landscape. *Strata 2012: Making Data Work*.

Eberhart, R. C., & Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. *International Conference on Evolutionary Programming*, 611–616.

Engin, M., & Demira\ug, S. (2003). Fuzzy-hybrid neural network based ECG beat recognition using three different types of feature sets. *Cardiovascular Engineering: An International Journal*, *3*(2), 71–80.

Erhan, D., Courville, A., & Vincent, P. (2010). Why Does Unsupervised Pre-training Help Deep Learning ? *Journal of Machine Learning Research*, *11*, 625–660.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115.

Foo, S. Y., Stuart, G., Harvey, B., & Meyer-Baese, A. (2002). Neural network-based EKG pattern recognition. *Engineering Applications of Artificial Intelligence*, *15*(3–4), 253–260.

Ghamisi, P., & Benediktsson, J. A. (2015). Feature selection based on hybridization of genetic algorithm and particle swarm optimization. *IEEE Geoscience and Remote Sensing Letters*, *12*(2), 309–313.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, *57*, 345–420.

Goldstein, B. A., Navar, A. M., & Carter, R. E. (2016). Moving beyond regression techniques in cardiovascular risk prediction: applying machine learning to address analytic challenges. *European Heart Journal*, *38*(23), 1805–1814.

Gonzalez, C. I., Melin, P., Castro, J. R., Mendoza, O., & Castillo, O. (2016). An improved sobel edge detection method based on generalized type-2 fuzzy logic. *Soft Computing*, *20*(2), 773–784.

Goto, S., Kimura, M., Katsumata, Y., Goto, S., Kamatani, T., Ichihara, G., … Sano, M. (2019). Artificial intelligence to predict needs for urgent revascularization from 12-leads electrocardiography in emergency patients. *PloS One*, *14*(1), e0210103.

Hansen, T., Henriksen, T. B., Bach, C. C., & Matthiesen, N. B. (2017). Congenital Heart Defects and Measures of Prenatal Brain Growth: A Systematic Review. *Pediatric Neurology*, *72*, 7–18.

Hastie, T., Tibshirani, R., Friedman, J., & Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, *27*(2), 83–85.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(July), 504–507.

Jain, L. C., Tan, S. C., & Lim, C. P. (2008). An introduction to

computational intelligence paradigms. In *Computational Intelligence Paradigms* (pp. 1–23). Springer.

Jin, X. (2016). Fault tolerant finite-time leader--follower formation control for autonomous surface vessels with LOS range and angle constraints. *Automatica*, *68*, 228–236.

John Lu, Z. Q. (2010). The elements of statistical learning: data mining, inference, and prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *173*(3), 693–694.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Kocyigit, Y., Alkan, A., & Erol, H. (2008). Classification of EEG recordings by using fast independent component analysis and artificial neural network. *Journal of Medical Systems*, *32*(1), 17–20.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 1097–1105.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., … Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. *International Conference on Machine Learning*, 1378–1387.

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, *8*(1), 98–113.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature Methods*, *13*(1), 35. https://doi.org/10.1038/nmeth.3707

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Li, Q., Rajagopalan, C., & Clifford, G. D. (2014). A machine learning approach to multi-level ECG signal quality classification. *Computer Methods and Programs in Biomedicine*, *117*(3), 435–447.

Littman, M. L., Moore, A. W., & others. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*(11, 28), 237–285.

Lui, H. W., & Chow, K. L. (2018). Multiclass classification of myocardial infarction with convolutional and recurrent neural networks for portable ECG devices. *Informatics in Medicine Unlocked*, *13*, 26–33.

Machine Learning Mastery. (2019). Retrieved September 19, 2018, from https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Minhas, R., Baradarani, A., Seifzadeh, S., & Wu, Q. M. J. (2010). Human action recognition using extreme learning machine based on visual vocabularies. *Neurocomputing*, *73*(10–12), 1906–1917.

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, *49*(1), 8–30.

Mohammed, A. A., Minhas, R., Wu, Q. M. J., & Sid-Ahmed, M. A. (2011). Human face recognition based on multidimensional PCA and extreme learning machine. *Pattern Recognition*, *44*(10–11), 2588–2597.

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, *2*(1), 1.

Naser, S. S. A., & Al-Bayed, M. H. (2016). *Detecting Health Problems Related to Addiction of Video Game Playing Using an Expert System*.

Orhan, U., Hekim, M., & Ozer, M. (2011). EEG signals classification using the K-means clustering and a multilayer perceptron neural network model. *Expert Systems with Applications*, *38*(10), 13475–13481.

Pan, C., Park, D. S., Yang, Y., & Yoo, H. M. (2012). Leukocyte image segmentation by visual attention and extreme learning machine. *Neural Computing and Applications*, *21*(6), 1217–1227.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, 1310–1318.

Patterson, J., & Gibson, A. (2017). Deep Learning: A Practitioner's Approach. In *O'Reilly* (1th ed.). O'Reilly Media, Inc.

Putra, W. S. E. (2016). Klasifikasi Citra Menggunakan Convolutional

Neural Network (CNN) pada Caltech 101. *Jurnal Teknik ITS*, *5*(1).

Quang, D., & Xie, X. (2016). DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, *44*(11), e107--e107.

Richardson, F., Reynolds, D., & Dehak, N. (2015). Deep neural network approaches to speaker and language recognition. *IEEE Signal Processing Letters*, *22*(10), 1671–1675.

Robert, C. (2014). *Machine learning, a probabilistic perspective*. Taylor & Francis.

Rout, A. K., Dash, P. K., Dash, R., & Bisoi, R. (2017). Forecasting financial time series using a low complexity recurrent neural network and evolutionary learning approach. *Journal of King Saud University-Computer and Information Sciences*, *29*(4), 536–552.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … others. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, *115*(3), 211–252.

Rutkowski, L. (2008). *Computational intelligence: methods and techniques*. Springer Science & Business Media.

Sannino, G., & De Pietro, G. (2018). A deep learning approach for ECG-based heartbeat classification for arrhythmia detection. *Future Generation Computer Systems*, *86*, 446–455.

https://doi.org/10.1016/j.future.2018.03.057

Sannino, Giovanna, & De Pietro, G. (2018). A deep learning approach for ECG-based heartbeat classification for arrhythmia detection. *Future Generation Computer Systems*, *86*, 446–455.

Siddique, N., & Adeli, H. (2013). *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing*. John Wiley & Sons.

Simic, V. (2015). Fuzzy risk explicit interval linear programming model for end-of-life vehicle recycling planning in the EU. *Waste Management*, *35*, 265–282.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *ArXiv Preprint ArXiv:1409.1556*.

Singh, S., Pandey, S. K., Pawar, U., & Janghel, R. R. (2018). Classification of ECG Arrhythmia using Recurrent Neural Networks. *Procedia Computer Science*, *132*, 1290–1297.

Stojanovski, D., Strezoski, G., Madjarov, G., & Dimitrovski, I. (2016). Deep learning architecture for twitter sentiment analysis. *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, 149–154.

Strodthoff, N., & Strodthoff, C. (2018). Detecting and interpreting myocardial infarctions using fully convolutional neural networks. *ArXiv Preprint ArXiv:1806.07385*.

Subasi, A. (2007). EEG signal classification using wavelet feature extraction and a mixture of expert model. *Expert Systems with Applications*, *32*(4), 1084–1093.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Vesely, S., Klöckner, C. A., & Dohnal, M. (2016). Predicting recycling behaviour: Comparison of a linear regression model and a fuzzy logic model. *Waste Management*, *49*, 530–536.

Wagenmakers, E.-J., Marsman, M., Jamil, T., Ly, A., Verhagen, J., Love, J., … others. (2018). Bayesian inference for psychology. Part I: Theoretical advantages and practical ramifications. *Psychonomic Bulletin & Review*, *25*(1), 35–57.

Walley, R. J., Smith, C. L., Gale, J. D., & Woodward, P. (2015). Advantages of a wholly Bayesian approach to assessing efficacy in early drug development: a case study. *Pharmaceutical Statistics*, *14*(3), 205–215.

Wiatowski, T., & Bölcskei, H. (2018). A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory*, *64*(3), 1845–1866.

Zen, H., & Sak, H. (2015). Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. *Acoustics, Speech and Signal Processing*

(ICASSP), 2015 IEEE International Conference On, 4470–4474.

Zhang, Q., Zhang, M., Chen, T., Sun, Z., Ma, Y., & Yu, B. (2019). Recent advances in convolutional neural network acceleration. *Neurocomputing*, *323*, 37–51.

Zhang, Z. (2018). Artificial neural network. In *Multivariate Time Series Analysis in Climate and Environmental Research* (pp. 1–35). Springer.

Zhao, Z., & Kumar, A. (2017). Accurate periocular recognition under less constrained environment using semantics-assisted convolutional neural network. *IEEE Transactions on Information Forensics and Security*, *12*(5), 1017–1030.