

Basuki Rahmat
Budi Nugroho

PEMROGRAMAN DEEP LEARNING DENGAN PYTHON

Dr. Basuki Rahmat, S.Si, MT
Budi Nugroho, S.Kom, M.Kom

PEMROGRAMAN DEEP LEARNING DENGAN python



 **Indomedia
Pustaka**

Dr. Basuki Rahmat, S.Si, MT
Budi Nugroho, S.Kom, M.Kom

PEMROGRAMAN DEEP LEARNING DENGAN python



PEMROGRAMAN DEEP LEARNING DENGAN PYTHON

Dr. Basuki Rahmat, S.Si, MT
Budi Nugroho, S.Kom, M.Kom



Edisi Asli
Hak Cipta © 2021 pada penulis
Griya Kebonagung 2, Blok I2, No.14
Kebonagung, Sukodono, Sidoarjo
Telp.: 0812-3250-3457
Website: www.indomediapustaka.com
E-mail: indomediapustaka.sby@gmail.com

Hak cipta dilindungi undang-undang. Dilarang memperbanyak sebagian atau seluruh isi buku ini dalam bentuk apa pun, baik secara elektronik maupun mekanik, termasuk memfotokopi, merekam, atau dengan menggunakan sistem penyimpanan lainnya, tanpa izin tertulis dari Penerbit.

UNDANG-UNDANG NOMOR 19 TAHUN 2002 TENTANG HAK CIPTA

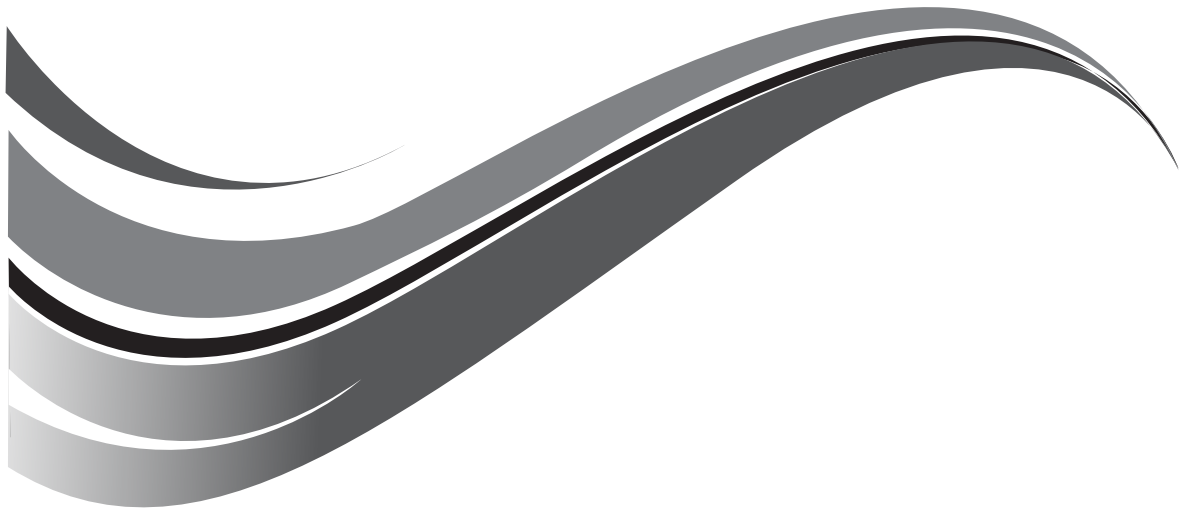
1. Barang siapa dengan sengaja dan tanpa hak mengumumkan atau memperbanyak suatu ciptaan atau memberi izin untuk itu, dipidana dengan pidana penjara paling lama **7 (tujuh) tahun** dan/atau denda paling banyak **Rp 5.000.000.000,00 (lima miliar rupiah)**.
2. Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran Hak Cipta atau Hak Terkait sebagaimana dimaksud pada ayat (1), dipidana dengan pidana penjara paling lama **5 (lima) tahun** dan/atau denda paling banyak **Rp 500.000.000,00 (lima ratus juta rupiah)**.

Rahmat, Basuki
Nugroho, Budi

Pemrograman Deep Learning dengan Python/Basuki Rahmat, Budi Nugroho
Edisi Pertama
—Sidoarjo: Indomedia Pustaka, 2021
Anggota IKAPI No. 195/JTI/2018
1 jil., 17 × 24 cm, 192 hal.

ISBN: 978-623-6133-07-1

- | | |
|----------------|--|
| 1. Pemrograman | 2. Pemrograman Deep Learning dengan Python |
| I. Judul | II. Basuki Rahmat, Budi Nugroho |



Kata Pengantar

Segala puji bagi Allah S.W.T yang telah melimpahkan rahmat, hidayah dan pertolongan-Nya sehingga kami dapat menyelesaikan buku yang berjudul “Pemrograman Deep Learning dengan Python”. Semoga buku ini bisa menjadi salah satu Buku Referensi untuk para dosen, mahasiswa, dan peneliti, serta siapa saja yang ingin mempelajari dan mengembangkan penelitian tentang *Deep Learning*.

Dengan selesainya buku ini, kami mengucapkan banyak terimakasih kepada:

1. Teman-teman di Program Studi Informatika, Program Studi Sistem Informasi, dan Program Studi Sains Data, UPN “Veteran” Jawa Timur.
2. LPPM dan UPN “Veteran” Jawa Timur yang telah mengusahakan dan membiayai penerbitan buku ini, dalam Program Penelitian Mandiri Skim Peningkatan Mutu Pembelajaran (PMP) UPN “Veteran” Jawa Timur Tahun Anggaran 2020.
3. Dan semua pihak yang telah membantu hingga terselesaikannya buku ini.

Kami menyadari bahwa buku ini masih banyak kekurangan. Kritik, saran dan diskusi lebih lanjut, serta peluang kerjasama riset, dan lain-lain, bisa disampaikan melalui alamat email: basukirahmat.if@upnjatim.ac.id. Terimakasih.

Surabaya, Desember 2020

Tim Penulis



Daftar Isi

Kata Pengantar.....	iii
Daftar Isi.....	v
Bab 1 Pendahuluan	1
1.1. Gambaran Isi Buku	1
1.2. Pengantar <i>Machine Learning</i>	4
1.3. Pengantar <i>Deep Learning</i>	6
1.4. Pengembangan <i>Framework Deep Learning</i>	12
Bab 2 Jaringan Saraf Tiruan	25
2.1. Jaringan Saraf Dasar	25
2.2. Simpul Jaringan Saraf	27
2.3. Fungsi Aktivasi	29
2.4. Arsitektur Jaringan Saraf	34
2.4.1. Lapisan Tunggal	34
2.4.2. Multi-Lapisan	35

2.5.	Pelatihan Jaringan Saraf Tiruan.....	36
2.5.1.	Pembelajaran yang Diawasi.....	36
2.5.2.	Pembelajaran Tanpa Pengawasan.....	42
Bab 3	Deep Learning dengan Tensorflow	43
3.1.	Vektor, Skalar, Matriks, dan Tensor.....	43
3.1.1.	Vektor	44
3.1.2.	Skalar.....	45
3.1.3.	Matriks.....	45
3.1.4.	Tensor.....	46
3.2.	Instalasi Python Jupyter Notebook.....	47
3.3.	TensorFlow Dasar	51
3.3.1.	Instalasi TensorFlow.....	52
3.3.2.	TensorFlow Dasar untuk Pengembangan.....	54
3.4.	Pengoptimal di TensorFlow.....	59
3.4.1.	GradientDescentOptimizer	59
3.4.2.	AdagradOptimizer	60
3.4.3.	RMSprop.....	60
3.4.4.	AdadeltaOptimizer	61
3.4.5.	AdamOptimizer.....	61
3.4.6.	MomentumOptimizer dan Nesterov Algorithm.....	62
3.5.	Prediksi XOR Menggunakan TensorFlow.....	63
3.6.	Grafik Komputasi TensorFlow dengan Tensorboard	68
Bab 4	Aplikasi Tensorflow	71
4.1.	Regresi Linear dengan TensorFlow.....	71
4.2.	Regresi Logistik dengan TensorFlow.....	76
4.3.	<i>Convolutional Neural Network</i> dengan TensorFlow	86
4.4.	<i>Convolutional Neural Network</i> dengan TensorFlow Lanjutan.....	95
4.5.	Studi Kasus Sistem Pengenalan Ikan	104
Bab 5	Deep Learning dengan Keras	139
5.1.	Deep Learning dengan Keras Dasar.....	139
5.2.	Prediksi XOR Menggunakan Keras.....	140
5.3.	Studi Kasus Prediksi Sederhana dengan Keras	146

5.3.1.	Penyiapan Data	147
5.3.2.	Penentuan Model Keras.....	149
5.3.3.	Kompilasi Model Keras	150
5.3.4.	Penyesuaian Model Keras	151
5.3.5.	Evaluasi Model Keras.....	152
5.3.6.	Pembuatan Prediksi	152
Bab 6	Aplikasi Deep Learning Studi Kasus	
	Prediksi Tingkat Pengangguran Terbuka	155
6.1.	Pengukuran Kinerja Deep Learning	155
6.2.	Prediksi Tingkat Pengangguran Terbuka (TPT)	
	di Indonesia	156
6.2.1.	Penyiapan Struktur Model Data.....	157
6.2.2.	Normalisasi Data	158
6.2.3.	Perancangan Sistem	159
6.2.4.	Implementasi Sistem	161
Bab 7	Penutup	181
7.1.	Kesimpulan	181
7.2.	Ucapan Terimakasih.....	181
	Daftar Pustaka.....	183



Bab 1

Pendahuluan

1.1. Gambaran Isi Buku

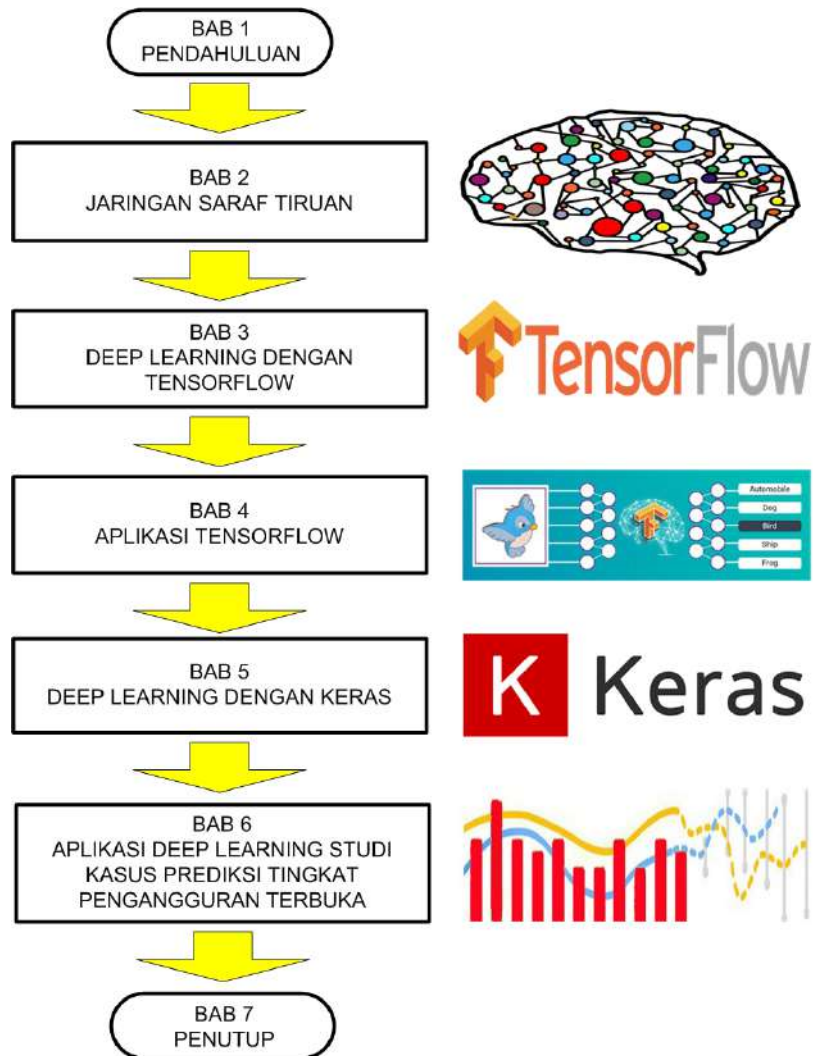
Jika pembaca sedang membaca buku ini, disaat bersamaan mungkin disadari bahwa telah terjadi kemajuan yang luar biasa dari penerapan *Deep Learning* untuk bidang kecerdasan buatan saat ini. Sudah lama kami ingin menuliskan buku ini, namun sepertinya baru terealisasi sekarang ini. Dimana barangkali disaat perkembangannya sudah diluar yang kami pikirkan sewaktu pertama kali ingin menuliskan buku ini. Namun jangan khawatir, semoga buku ini tetap bermanfaat untuk menemani belajar *Deep Learning* untuk meningkatkan riset di bidang kecerdasan buatan. Apalagi kebanyakan literatur-literatur yang ada saat ini tentang *Deep Learning* masih jarang yang berbahasa Indonesia.

Bahasa pemrograman yang kami pilih untuk implementasi pemrograman *Deep Learning* ini adalah bahasa pemrograman favorit kami dan mungkin juga favorit kebanyakan orang-orang saat ini, yaitu Python. Sederetan alasan mengapa harus menggunakan Python, kami ambil dari pythonindo.com antara lain:

- Mudah dipelajari. Sintaksnya jelas dan mudah dibaca. Sangat cocok digunakan, terutama sebagai bahasa pemrograman pertama.

- Sederhana tapi *powerful*. Menulis kodingnya membutuhkan baris perintah yang lebih sedikit dibanding bahasa pemrograman lain.
- Serbaguna, bisa untuk hampir apa saja kebutuhan atau ketertarikan anda. Python bisa dipakai untuk pemrograman dekstop maupun mobile, CLI, GUI, web, otomatisasi, *hacking*, IoT, robotik, dan lain sebagainya.
- Sangat populer. Rangking 3 di TIOBE index tahun 2020. No. 4 bahasa pemrograman paling banyak digunakan menurut Stackoverflow survey 2020. Merupakan no.1 bahasa pemrograman yang paling cepat perkembangannya dan paling diinginkan beberapa tahun belakangan menurut Stackoverflow.
- Siapa yang memakai Python? Python banyak dipakai perusahaan-perusahaan besar dan top di dunia. Google menggunakannya di mesin pencari, di youtube dan lain-lain, microsoft, dropbox, instagram, pinterest, dan lain-lain.
- *Portable*, bisa jalan di multi platform, windows, linux, mac OS, Virtual Machine Java dan .NET.
- Modul (*library*) Python sangat banyak (berlimpah) sehingga dapat memudahkan kita membuat program tanpa harus menulis kode dari dasar.
- Python adalah bahasa pemrograman yang 'kekinian' karena sedang naik daunnya bidang data *science* dan cabangnya seperti AI, *machine learning*, dan *big data*. Python paling populer dan banyak digunakan karena memiliki *library* yang lengkap untuk itu seperti sklearn, pytorch, tensorflow, dan lain sebagainya.
- *Open source*. Python akan terus berkembang karena didukung oleh komunitas yang besar dan Lembaga *Python Software Foundation* (PSF) yang tiap tahunnya mengadakan konferensi internasional.
- Kode Python bisa dijalankan secara interaktif (*mode interactive*) untuk testing, langsung nampak hasilnya.
- Multi paradigma, kita bisa menulis Python dengan gaya pemrograman fungsional dan bisa juga dengan OOP. OOP-nya lebih mudah dan sederhana daripada di bahasa seperti Java atau C++.
- Kode Python bisa di-*embed* ke bahasa lain seperti C dan Java, atau sebaliknya, dari bahasa C atau Java ke Python.
- Python sangat cepat. *Source code* akan dikompilasi jadi bytecode, eksekusi file yang sama untuk kedua kalinya akan lebih cepat.
- Tutorial Python tersedia melimpah, baik berbentuk buku cetak, e-book, artikel, video, dan forum tanya jawab. Semua permasalahan hampir dipastikan akan ada penyelesaiannya di internet.

Kami berusaha menyajikan bagaimana mempelajari konsep dan mengimplementasikan *Deep Learning* ini sebaik-baiknya. Kami berusaha menyajikan secara sistematis agar mudah dipahami. Kami berharap agar buku ini berharga bagi siapa pun yang ingin membangun aplikasi cerdas dan memecahkan masalah-masalah yang penting bagi siapa pun. Secara umum urutan pembahasan Pemrograman *Deep Learning* dengan Python ini seperti diperlihatkan pada Gambar 1.1.



GAMBAR 1.1.
Gambaran Isi Buku

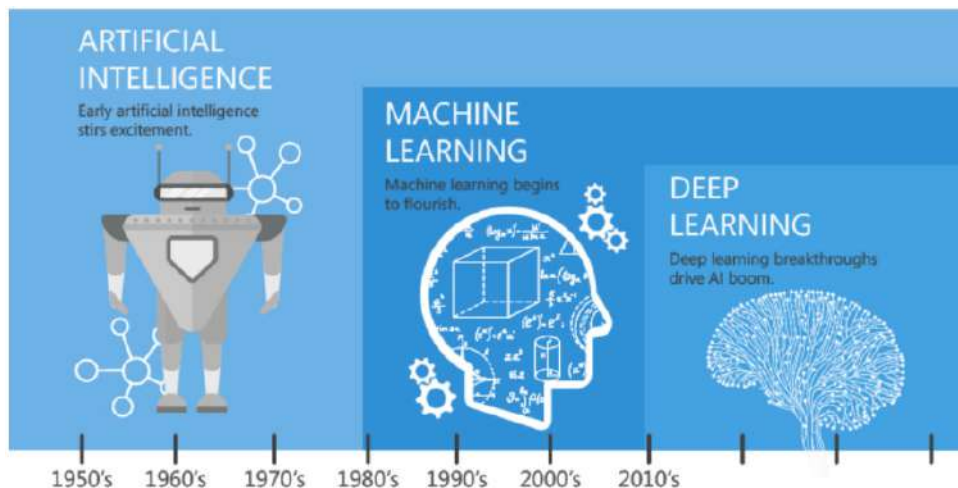
Bab 1 Pendahuluan, berisi tentang Gambaran Isi Buku, Pengantar *Machine Learning*, Pengantar *Deep Learning*, dan Pengembangan *Framework Deep Learning*. Bab 2 Jaringan Saraf Tiruan, dibahas tentang Jaringan Saraf Dasar, Simpul Jaringan Saraf, Fungsi Aktivasi, Arsitektur Jaringan Saraf, dan Pelatihan Jaringan Saraf Tiruan. Bab 3 *Deep Learning* dengan Tensorflow, dibahas tentang Vektor, Skalar, Matriks, dan Tensor. Instalasi Python Jupyter Notebook. TensorFlow Dasar, yang terdiri dari: Instalasi TensorFlow, dan TensorFlow Dasar untuk Pengembangan. Pengoptimal di TensorFlow, yang terdiri dari: GradientDescentOptimizer, AdagradOptimizer, RMSprop, AdadeltaOptimizer, AdamOptimizer, MomentumOptimizer dan Nesterov Algorithm. Prediksi XOR Menggunakan TensorFlow. Dan Bab 3 diakhiri dengan pembahasan tentang Grafik Komputasi TensorFlow dengan Tensorboard. Bab 4 Aplikasi TensorFlow, dibahas tentang Regresi Linear dengan TensorFlow, Regresi Logistik dengan TensorFlow, *Convolutional Neural Network* dengan TensorFlow, *Convolutional Neural Network* dengan TensorFlow Lanjutan, dan Studi Kasus Sistem Pengenalan Ikan. Bab 5 *Deep Learning* dengan Keras, dibahas tentang *Deep Learning* dengan Keras Dasar, Prediksi XOR Menggunakan Keras, dan Studi Kasus Prediksi Sederhana dengan Keras. Bab 6 Aplikasi *Deep Learning* Studi Kasus Prediksi Tingkat Pengangguran Terbuka, dibahas tentang Pengukuran Kinerja *Deep Learning*. Prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia, yang terdiri dari: Penyiapan Struktur Model Data, Normalisasi Data, Perancangan Sistem, dan Implementasi Sistem. Dan terakhir, Bab 7 Penutup, berisi tentang kesimpulan, dan ucapan terimakasih.

1.2. Pengantar *Machine Learning*

Machine Learning merupakan bagian dari Kecerdasan Buatan atau *Artificial Intelligence* (AI). AI didefinisikan sebagai situasi di mana mesin dapat mensimulasikan pikiran manusia dalam pembelajaran dan analisis, yang dengan demikian dapat bekerja dalam pemecahan masalah (Rong *et al.*, 2020). Dengan kata lain situasi dimana mesin seolah bisa menirukan bagaimana manusia berpikir, belajar, menganalisis dan mengambil keputusan untuk memecahkan masalah. AI juga didefinisikan sebagai studi tentang “agen cerdas” yaitu, agen atau perangkat apa pun yang dapat merasakan dan memahami lingkungannya sehingga dapat mengambil tindakan yang tepat agar bisa memaksimalkan peluangnya dalam mencapai tujuannya (Rong *et al.*, 2020).

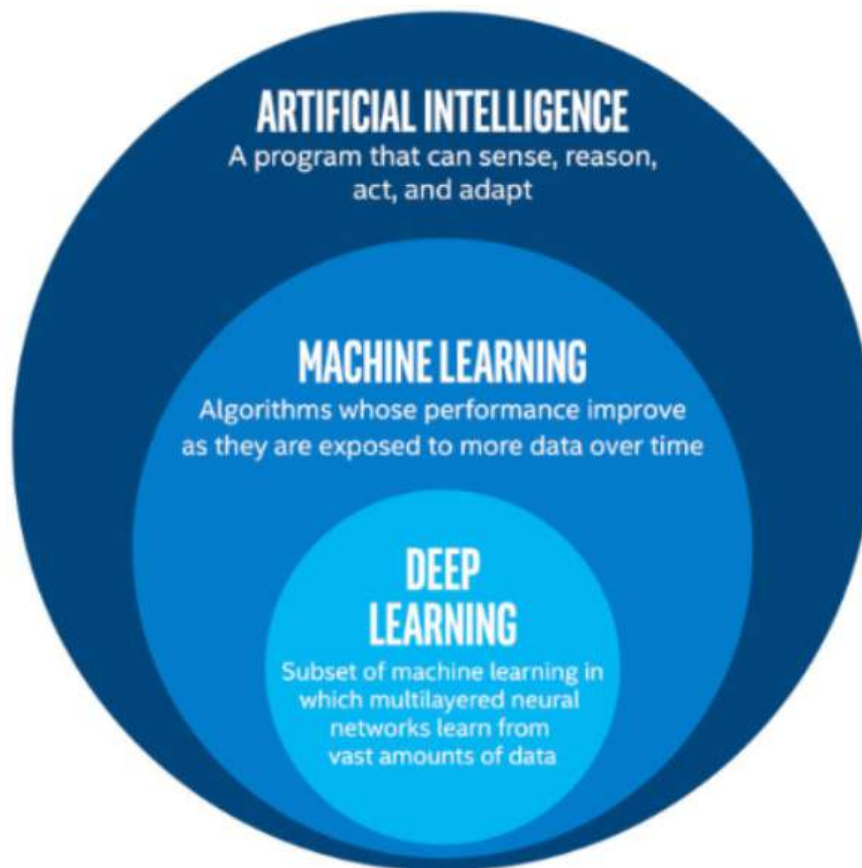
AI mulai dikutip pada tahun 1940-an dalam studi tentang apakah mesin dapat membuat sebuah keputusan (de Sousa *et al.*, 2019). Pada tahun 1970-an, pengembangan solusi AI terapan dimulai, dengan studi di berbagai bidang pengetahuan. Pada tahun 2010-an, studi dan aplikasi AI telah mencakup beberapa fungsi sektor publik, kesehatan masyarakat, transportasi, pendidikan, keamanan, komunikasi, dan bahkan angkatan bersenjata (de Sousa *et al.*, 2019).

Perkembangan lebih lanjut dari AI melahirkan *Machine Learning*. Perkembangan lebih lanjut dari *Machine Learning* melahirkan *Deep Learning*. Secara umum perkembangan AI, *Machine Learning* dan *Deep Learning* dari tahun ke tahun diperlihatkan pada Gambar 1.2 (Oppermann, 2019). Dikaitkan dengan AI dan *Machine Learning*, *Deep Learning* merupakan bagian dari *Machine Learning* dan AI. Hubungan antara AI, *Machine Learning* dan *Deep Learning* diperlihatkan pada Gambar 1.3 (Oppermann, 2019).



GAMBAR 1.2.
Perkembangan Artificial Intelligence, Machine Learning dan Deep Learning (Oppermann, 2019)

Machine Learning sendiri merupakan teknik yang memungkinkan komputer “belajar” dari data-data yang disediakan tanpa pemrograman yang menyeluruh dan eksplisit dari setiap masalah (Meng *et al.*, 2020). Ini bertujuan memodelkan hubungan yang mendalam terhadap input data dan merekonstruksi skema pengetahuan. Hasil dari proses pembelajaran dapat digunakan untuk estimasi, prediksi, dan klasifikasi (Meng *et al.*, 2020).



GAMBAR 1.3.

Deep Learning bagian dari Machine Learning, keduanya bagian dari Artificial Intelligence (Oppermann, 2019)

1.3. Pengantar *Deep Learning*

Deep Learning telah menjadi tantangan untuk didefinisikan oleh banyak orang dan para peneliti karena telah berubah bentuk secara perlahan selama dekade terakhir. Satu definisi yang berguna menetapkan bahwa *Deep Learning* berkaitan dengan “Jaringan Sel Saraf (*Neural Network*) dengan lebih dari dua lapisan.” Namun jika dari definisi ini, maka seolah-olah *Deep Learning* sudah ada sejak tahun 1980-an (lihat Gambar 1.2). Padahal *Deep Learning* dianggap baru muncul pada tahun 2006, setelah Geoffrey Hinton memperkenalkan salah satu varian Jaringan Sel Saraf Tiruan atau biasa disebut Jaringan Saraf Tiruan (*Artificial Neural Network*) yang disebut *Deep Belief Nets* (Hinton, Osindero and Teh, 2006). Ide untuk melatih model Jaringan

Saraf ini adalah dengan melatih dua lapisan kemudian ditambahkan satu lapisan diatasnya, kemudian dilatih hanya lapisan teratas dan begitu seterusnya. Dengan strategi ini didapat pelatihan model Jaringan Saraf yang memiliki lapisan lebih banyak dari model-model sebelumnya. Tulisan Geoffrey Hinton ini merupakan awal populernya istilah *Deep Learning* untuk membedakan arsitektur Jaringan Saraf Tiruan dengan banyak lapisan.

Setelah istilah *Deep Learning* populer, *Deep Learning* belum menjadi daya tarik yang besar bagi para peneliti karena Jaringan Saraf Tiruan dengan banyak lapisan memiliki kompleksitas algoritma yang besar, sehingga membutuhkan komputer dengan spesifikasi tinggi, dan tidak efisien secara komputasi saat itu. Hingga pada tahun 2009 Andrew Y. Ng dkk memperkenalkan penggunaan *Graphics Processing Unit* (GPU) untuk *Deep Learning* (Raina, Madhavan and Ng, 2009). Dengan penggunaan GPU Jaringan Saraf Tiruan dapat berjalan lebih cepat dibanding dengan menggunakan *Central Processing Unit* (CPU). Dengan tersedianya *hardware* yang memadai perkembangan *Deep Learning* mulai pesat, dan menghasilkan produk-produk yang dapat kita nikmati saat ini seperti pengenalan wajah, *self-driving car*, pengenalan suara, dan lain lain.

Perkembangan Jaringan Saraf sudah jauh melampaui arsitektur gaya jaringan sebelumnya (diiringi dengan lebih banyak kekuatan pemrosesan) sebelum menunjukkan hasil spektakuler yang terlihat dalam beberapa tahun terakhir. Berikut ini adalah beberapa aspek dalam evolusi Jaringan Saraf ini, yang menjadi ciri khas dari *Deep Learning*:

- Lebih banyak neuron daripada jaringan sebelumnya.
- Cara yang lebih kompleks untuk menghubungkan lapisan/neuron di Jaringan Saraf.
- Ledakan dalam jumlah daya komputasi yang tersedia untuk pelatihan.
- Ekstraksi fitur otomatis.

Deep Learning mencoba memodelkan abstraksi data skala besar dengan menggunakan arsitektur *Deep Neural Networks* (DNNs) multi lapisan, sehingga mereka dapat memahami gambar, suara, dan teks (Cao et al., 2018). *Deep Learning* umumnya memiliki dua sifat (Cao et al., 2018):

- (1) terdiri dari beberapa lapisan unit pemrosesan nonlinear, dan
- (2) presentasi fitur pada setiap lapisan menggunakan proses pembelajaran yang diawasi atau tidak diawasi.

Selanjutnya, pembahasan tentang *Deep Learning*, tidak dapat dipisahkan dari *Deep Network* itu sendiri yang membahas tentang arsitektur jaringannya. Arsitektur utama *Deep Network* yang terkenal dan banyak digunakan di berbagai bidang penerapan, yaitu (Adam Gibson, 2017):

- a. *Unsupervised Pretrained Networks* (UPNs)
- b. *Convolutional Neural Networks* (CNNs)
- c. *Recurrent Neural Networks*, dan
- d. *Recursive Neural Networks*.

Masing-masing secara ringkas diuraikan sebagai berikut.

a. *Unsupervised Pretrained Networks* (UPNs)

Dalam kelompok ini, setidaknya terdapat tiga arsitektur spesifik, yaitu (Adam Gibson, 2017):

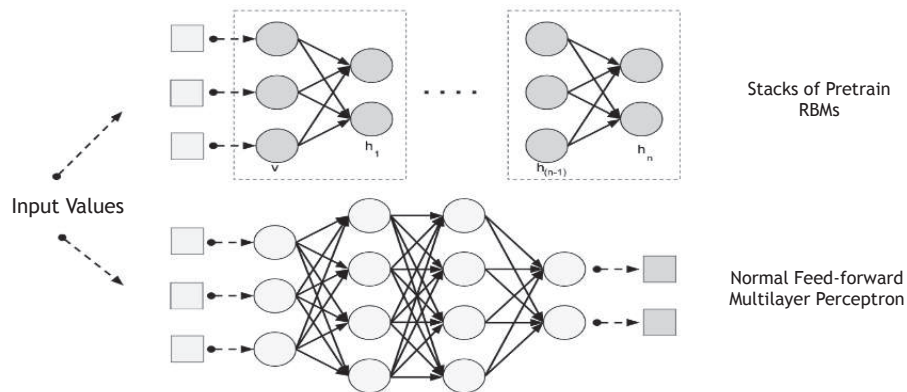
- *Autoencoders*
- *Deep Belief Networks* (DBNs), dan
- *Generative Adversarial Networks* (GANs).

Autoencoders

Autoencoder adalah varian dari Jaringan Saraf umpan-maju yang memiliki bias ekstra untuk menghitung kesalahan merekonstruksi input asli. Setelah pelatihan, *autoencoder* kemudian digunakan sebagai Jaringan Saraf umpan-maju normal untuk aktivasi. Ini adalah bentuk ekstraksi fitur yang tidak diawasi karena Jaringan Saraf hanya menggunakan input asli untuk pembelajaran bobot daripada *backpropagation*, yang memiliki label. *Deep Network* dapat menggunakan *Restricted Boltzmann Machines* (RBMs) atau *autoencoder* sebagai blok penyusun untuk jaringan yang lebih besar (namun jarang ada satu jaringan yang menggunakan keduanya).

***Deep Belief Networks* (DBNs)**

DBN terdiri dari lapisan *Restricted Boltzmann Machines* (RBMs) untuk fase pra-pelatihan dan kemudian jaringan umpan-maju untuk fase *fine-tune*. Gambar 1.4 menunjukkan arsitektur jaringan DBN (Adam Gibson, 2017).



GAMBAR 1.4.
Arsitektur jaringan DBN (Adam Gibson, 2017)

Generative Adversarial Networks (GANs)

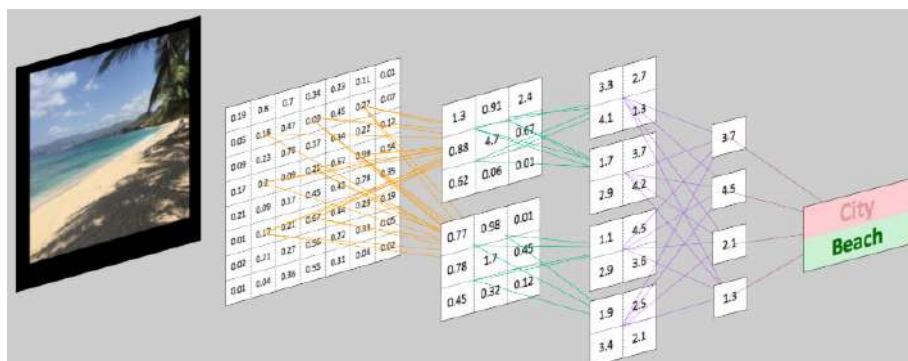
GAN adalah contoh jaringan yang menggunakan pembelajaran tanpa pengawasan untuk melatih dua model secara paralel. Aspek kunci dari GAN (dan model generatif pada umumnya) adalah bagaimana mereka menggunakan jumlah parameter yang secara signifikan lebih kecil dari normal sehubungan dengan jumlah data yang dilatih dalam jaringan. Jaringan dipaksa untuk secara efisien mewakili data pelatihan, membuatnya lebih efektif dalam menghasilkan data yang mirip dengan data pelatihan.

b. *Convolutional Neural Networks (CNNs)*

Tujuan dari CNN adalah mempelajari fitur tingkat tinggi dalam data melalui konvolusi. CNN sangat cocok untuk pengenalan objek melalui gambar dan memegang kompetisi klasifikasi gambar teratas secara konsisten. CNN dapat mengidentifikasi wajah, individu, rambu jalan, platipus, dan banyak aspek lain dari data visual. Untuk analisis teks, CNN tumpang tindih dengan *Optical Character Recognition* (OCR), karena CNN juga mampu menganalisis kata-kata sebagai unit tekstual diskrit. CNN juga pandai menganalisis suara.

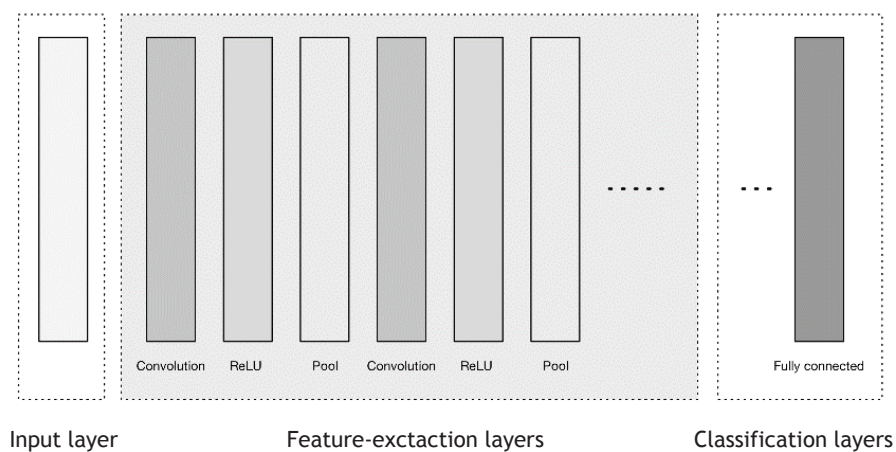
Kemampuan CNN dalam pengenalan gambar adalah salah satu alasan utama mengapa dunia mengakui kehebatan *Deep Learning*. Seperti yang diilustrasikan pada Gambar 1.5, CNN sangat bagus dalam mendapatkan fitur-

fitur dalam posisi tegak dan agak bagus dalam posisi invarian rotasi dari data gambar mentah (Adam Gibson, 2017).



GAMBAR 1.5.
Contoh CNN dalam visi komputer (Adam Gibson, 2017)

CNN mengubah data input dari lapisan input melalui semua lapisan yang terhubung ke satu set skor kelas yang diberikan oleh lapisan output. Ada banyak variasi arsitektur CNN, tetapi umumnya mereka didasarkan pada pola lapisan, seperti yang ditunjukkan pada Gambar 1.6 (Adam Gibson, 2017).



GAMBAR 1.6.
Arsitektur CNN (Adam Gibson, 2017)

Gambar 1.6 menggambarkan tiga kelompok utama:

- Lapisan input
- Lapisan ekstraksi fitur, dan
- Lapisan klasifikasi

Lapisan input menerima input tiga dimensi umumnya dalam bentuk spasial dari ukuran gambar (lebar \times tinggi) dan memiliki kedalaman yang mewakili saluran warna (umumnya tiga, untuk saluran warna *Red Green Blue* (RGB)).

Lapisan ekstraksi fitur memiliki pola pengulangan urutan yang umum:

- Lapisan konvolusi
- Lapisan (fungsi aktivasi) *Rectified Linear Unit* (ReLU), dan
- Lapisan pooling.

Lapisan-lapisan ini menemukan sejumlah fitur dalam gambar dan secara progresif membangun fitur tingkat tinggi.

Lapisan klasifikasi memiliki satu atau lebih lapisan yang terhubung sepenuhnya untuk mengambil fitur tingkat tinggi dan menghasilkan probabilitas atau skor kelas. Lapisan-lapisan ini sepenuhnya terhubung ke semua neuron di lapisan sebelumnya. Keluaran dari lapisan-lapisan ini biasanya menghasilkan keluaran dua dimensi dari dimensi $[b \times N]$, di mana b adalah jumlah contoh-contoh dalam *mini-batch* dan N adalah jumlah kelas dalam penyusunan skor.

c. *Recurrent Neural Networks*

Recurrent Neural Networks berada dalam keluarga Jaringan Saraf umpan-maju. Jaringan yang memungkinkan komputasi paralel dan berurutan. Jaringan ini serupa dengan otak manusia, yang memiliki jaringan umpan balik besar dari neuron yang terhubung. Otak adalah panutan yang luar biasa karena dapat memecahkan banyak masalah yang belum dapat diselesaikan oleh mesin saat ini. Jaringan ini mengambil setiap vektor dari urutan vektor input dan memodelkannya satu per satu. Ini memungkinkan jaringan untuk mempertahankan status, sementara ia memodelkan setiap vektor input

melintasi jendela vektor input. Memodelkan dimensi waktu adalah ciri khas dari jaringan ini.

d. *Recursive Neural Networks*

Recursive Neural Networks, seperti halnya *Recurrent Neural Networks*, dapat menangani input variabel panjang. Bedanya *Recurrent Neural Networks* memiliki kemampuan untuk memodelkan struktur hierarki dalam dataset pelatihan. Lalu apa yang menarik dari jaringan ini? Secara umum, gambar memiliki pemandangan yang terdiri dari banyak objek. Adegan-adegan yang mendekonstruksi sering kali merupakan domain masalah yang menarik. Sifat rekursif dari dekonstruksi inilah yang menantang, karena bukan hanya permasalahan identifikasi objek dalam adegan saja, tetapi juga bagaimana agar objek-objek terkait dapat membentuk adegan.

Arsitektur jaringan ini terdiri dari matriks berbobot bersama dan struktur pohon biner yang memungkinkan jaringan rekursif mempelajari berbagai urutan kata atau bagian gambar. Ini berguna sebagai parser kalimat dan adegan. Jaringan ini menggunakan variasi *backpropagation* yang disebut *backpropagation through structure* (BPTS). Umpan-umpan maju terjadi dari bawah ke atas, dan *backpropagation* adalah dari atas ke bawah. Bisa dianggap, bahwa tujuan adalah seperti bagian atas pohon, sedangkan inputnya adalah bagian bawahnya.

1.4. Pengembangan *Framework Deep Learning*

Sesuai judul buku ini, maka selanjutnya untuk mewujudkan pemrograman *Deep Learning* menggunakan Bahasa Pemrograman Python, sesuai dengan arsitektur-arsitektur *Deep Network*, saat ini tersedia banyak pilihan *framework*, antara lain:

- a. TensorFlow
- b. Keras
- c. PyTorch
- d. Caffe
- e. Sonnet
- f. MXNet
- g. Gluon
- h. Swift

- i. Chainer
- j. DL4J
- k. ONNX
- l. Darknet
- m. Darkflow
- n. YOLO

Masing-masing dijelaskan secara ringkas sebagai berikut. Sebagian besar diambil dari referensi (Kharkovyna, 2019). Sedangkan untuk mempelajari lebih mendalam, akan dibahas di bab tersendiri. Meskipun tidak dibahas semua *framework*. Kami hanya memilih yang cukup populer menurut kami.

a. TensorFlow

TensorFlow adalah *framework* yang dibuat dan dikembangkan oleh Google yang digunakan untuk merancang, membangun, dan melatih model *Deep Learning*. *Framework* ini bisa dibilang paling populer saat ini. Gmail, Uber, Airbnb, Nvidia, dan banyak merek terkenal lainnya menggunakannya.



GAMBAR 1.7.
Logo TensorFlow

Python adalah bahasa klien yang paling nyaman untuk bekerja dengan TensorFlow. Namun, ada juga antarmuka eksperimental yang tersedia di JavaScript, C ++, Java dan Go, C # dan Julia. TensorFlow tidak hanya memperhitungkan kluster komputasi yang kuat tetapi juga kemampuan untuk menjalankan model pada platform seluler seperti iOS dan Android.

TensorFlow membutuhkan banyak pengkodean. Itu tidak akan menghasilkan AI yang kuat dalam semalam, tetapi bisa jadi alat bantu penelitian *Deep Learning* agar sedikit kurang rumit. Perlu berpikir hati-hati tentang arsitektur Jaringan Saraf, penilaian dengan benar terhadap dimensi dan volume data input dan outputnya, agar dihasilkan karya AI yang hebat.

TensorFlow beroperasi dengan grafik perhitungan statis. Yaitu, pertama-tama didefinisikan grafiknya, lalu dijalankan perhitungannya. Jika diperlukan, dibuat perubahan pada arsitekturnya, dan dilatih kembali modelnya. Pendekatan seperti itu dipilih demi efisiensi. Tentunya perhitungan perbaikan dalam proses pembelajaran tanpa harus kehilangan kecepatan belajarnya. Dalam hal ini, pesaing utama TensorFlow adalah PyTorch.

TensorFlow berguna untuk membuat dan bereksperimen dengan arsitektur *Deep Learning*. Formulasinya nyaman untuk integrasi data seperti memasukkan grafik, tabel SQL, dan gambar secara bersamaan. Hal itu didukung oleh Google yang menjamin akan tetap ada untuk sementara waktu. Oleh karena itu masuk akal untuk menginvestasikan waktu dan sumber daya untuk mempelajarinya.

b. Keras

Keras adalah *framework Machine Learning* yang mungkin bisa menjadi teman baru ketika bekerja dengan banyak data, terutama jika mengikuti *state-of-the-art* dari AI: *Deep Learning*. Keras dapat digunakan sebagai API tingkat tinggi di atas *library* tingkat bawah populer lainnya seperti Theano dan CNTK selain Tensorflow. Penciptaan model *Deep Learning* yang sangat besar di Keras direduksi menjadi fungsi garis tunggal. Tetapi strategi ini membuat Keras menjadi lingkungan yang kurang dapat dikonfigurasi dibandingkan dengan *framework* tingkat rendah.



GAMBAR 1.8.
Logo Keras

Keras adalah *framework Deep Learning* yang terbaik bagi mereka yang baru memulai. Ini ideal untuk belajar dan membuat prototipe konsep-konsep sederhana, untuk memahami inti dari berbagai model dan proses pembelajaran mereka.

Keras adalah API yang ditulis dengan indah. Sifat fungsional API membantu sepenuhnya dan sebagai jalan keluar menghasilkan aplikasi yang lebih eksotis.

Keras tidak memblokir akses ke *framework* tingkat bawah. Keras menghasilkan kode yang jauh lebih mudah dibaca dan ringkas. Keras dengan model API serialisasi/deserialisasi, panggilan balik, dan streaming data menggunakan generator Python, sangatlah matang. Namun, Keras tidak dapat dibandingkan dengan Tensorflow karena mereka berada pada level abstraksi yang berbeda.

Tensorflow ada di Level Bawah. Di sinilah *framework* seperti MXNet, Theano, dan PyTorch. Ini adalah tingkat di mana operasi matematika seperti perkalian matriks-matriks umum, dan Jaringan Saraf primitif seperti operasi konvolusi diimplementasikan.

Keras ada di level yang lebih tinggi. Pada level ini, level primitif bawah digunakan untuk mengimplementasikan abstraksi Jaringan Saraf seperti lapisan-lapisan dan model. Secara umum, pada level ini, API memiliki manfaat lainnya seperti penghematan model dan pelatihan model.

c. PyTorch

Alat perangkat lunak utama untuk *Deep Learning* setelah Tensorflow adalah PyTorch. *Framework* PyTorch dikembangkan untuk layanan Facebook tetapi sudah digunakan untuk menyelesaikan tugas-tugasnya sendiri oleh perusahaan seperti Twitter dan Salesforce.

Tidak seperti TensorFlow, *library* PyTorch beroperasi dengan grafik yang diperbarui secara dinamis. Ini berarti dimungkinkan dibuat perubahan pada arsitektur ketika proses sedang berlangsung. Di PyTorch, bisa digunakan *debugger* standar, seperti pdb atau PyCharm.



GAMBAR 1.9.
Logo PyTorch

PyTorch memiliki proses pelatihan Jaringan Saraf yang sederhana dan jelas. Pada saat yang sama, PyTorch mendukung paralelisme data dan model pembelajaran terdistribusi, dan juga berisi banyak model pra-terlatih. PyTorch jauh lebih cocok untuk proyek kecil dan pembuatan prototipe. Namun ketika dibutuhkan solusi lintas platform, maka TensorFlow sepertinya pilihan yang lebih cocok.

d. Caffe

Caffe (*Convolutional Architecture for Fast Feature Embedding*) adalah *framework Deep Learning* yang mendukung banyak antarmuka seperti C, C ++, Python, MATLAB serta Antarmuka Baris Perintah.

Caffe mendukung berbagai jenis arsitektur *Deep Learning* yang diarahkan pada segmentasi gambar dan klasifikasi gambar. Ini dikembangkan oleh *Berkeley AI Research* (BAIR) dan oleh kontributor komunitas.



GAMBAR 1.10.
Logo Caffe

Namun, Caffe tidak mendukung lapisan jaringan granularitas yang baik seperti pada TensorFlow. Mengingat arsitekturnya, dukungan keseluruhan terhadap jaringan berulang dan pemodelan bahasanya cukup buruk, dan pembangunan tipe lapisan kompleks harus dilakukan dalam bahasa tingkat rendah.

Caffe adalah jaringan *Deep Learning* yang populer untuk pengenalan gambar. Ini digunakan oleh para akademisi dan perusahaan pemula (*startup*) tetapi juga beberapa perusahaan besar seperti Yahoo. Caffe juga dapat melakukan komputasi pada *Central Processing Unit* (CPU) dan *Graphics Processing Unit* (GPU).

e. Sonnet

Sonnet adalah *framework Deep Learning* yang dibangun di atas TensorFlow. Ini dirancang untuk membuat Jaringan Saraf dengan arsitektur yang kompleks oleh perusahaan terkenal dunia DeepMind. *Library* Sonnet berorientasi objek tingkat tinggi yang menghasilkan abstraksi ketika mengembangkan Jaringan Saraf atau algoritma *Machine Learning* lainnya.



GAMBAR 1.11.
Logo Sonnet

Ide Sonnet adalah untuk membangun objek Python primer yang sesuai dengan bagian tertentu dari Jaringan Saraf. Selanjutnya, objek-objek ini terhubung secara independen ke komputasi grafik TensorFlow. Pemisahan proses pembuatan objek dan menghubungkannya dengan grafik, dapat menyederhanakan desain arsitektur tingkat tinggi.

Keuntungan utama dari Sonnet, adalah dapat digunakan untuk mereproduksi penelitian-penelitian seperti yang telah ditunjukkan oleh makalah-makalah DeepMind. Hal ini tentunya DeepMind akan lebih memilih menggunakan Sonnet sendiri, daripada menggunakan framework-framework Deep Learning lainnya.

f. MXNet

MXNet adalah *framework Deep Learning* yang sangat skalabel yang dapat digunakan pada berbagai perangkat. Meskipun tampaknya tidak banyak digunakan dibandingkan dengan TensorFlow. Pertumbuhan MXNet kemungkinan banyak didorong oleh perkembangan proyek-proyek Apache.



GAMBAR 1.12.
Logo MXNet

Framework ini awalnya mendukung sejumlah besar bahasa (C ++, Python, R, Julia, JavaScript, Scala, Go, dan bahkan Perl). Penekanan utamanya pada kenyataan bahwa *framework* ini paralel sangat efektif pada banyak GPU dan banyak mesin. Ini, khususnya, telah ditunjukkan oleh karyanya di *Amazon Web Services*.

MXNet mendukung banyak GPU (dengan perhitungan yang dioptimalkan dan pengalihan konteks cepat). Kode bersih dan mudah dirawat (Python, R, Scala, dan API lainnya). Kemampuan pemecahan masalah yang cepat (penting, bagi pemula yang belajar Deep Learning).

Meskipun tidak sepopuler TensorFlow, MXNet memiliki dokumentasi terperinci dan mudah digunakan. Dengan kemampuan untuk memilih antara gaya pemrograman imperatif dan simbolis, menjadikannya kandidat *framework* yang baik bagi para pemula dan programmer yang berpengalaman.

g. Gluon

Gluon adalah *framework Deep Learning* yang luar biasa yang dapat digunakan untuk membuat model sederhana maupun canggih. Kekhususan proyek Gluon adalah antarmuka yang fleksibel yang menyederhanakan prototipe, membangun dan melatih model Deep Learning tanpa mengorbankan kecepatan belajar.



GAMBAR 1.13.
Logo Gluon

Gluon didasarkan pada MXNet dan menawarkan API sederhana yang menyederhanakan pembuatan model Deep Learning. Mirip dengan PyTorch, *framework* Gluon mendukung pekerjaan dengan grafik dinamis. Penggabungan Gluon dengan MXNet akan menghasilkan kinerja yang tinggi. Dari perspektif ini, Gluon terlihat seperti alternatif yang sangat menarik dari Keras untuk komputasi terdistribusi.

Gluon dapat mendefinisikan Jaringan Saraf menggunakan kode sederhana, jelas, dan ringkas. Ini menyatukan algoritma pelatihan dan model Jaringan Saraf, sehingga memberikan fleksibilitas dalam proses pengembangan tanpa mengorbankan kinerja. Gluon memungkinkan untuk mendefinisikan model Jaringan Saraf yang dinamis, artinya mereka dapat dibangun dengan cepat, dengan struktur apa pun, dan menggunakan aliran kontrol asli Python.

h. Swift



GAMBAR 1.14.
Logo Swift

Bagi sebagian programmer, ketika mendengar Swift, mungkin yang dipikirkan adalah pengembangan aplikasi berbasis iOS atau MacOS. Bagi yang tertarik dengan *Deep Learning*, maka seharusnya tidak asing dengan istilah Swift for Tensorflow (disingkat S4TF). Dengan integrasi langsung dengan bahasa pemrograman umum, S4TF memungkinkan menjadi algoritma yang lebih kuat untuk diekspresikan dalam penyelesaian masalah. Dengan menggunakan TensorFlow, API Swift memberi akses transparan ke semua operator TensorFlow tingkat rendah.

i. Chainer

Sampai munculnya DyNet di CMU, dan PyTorch di Facebook, Chainer adalah *framework* Jaringan Saraf terkemuka untuk grafik komputasi dinamis atau jaringan yang memungkinkan input dengan panjang berbeda-beda. Fitur yang populer untuk tugas-tugas *Neuro-linguistic programming* (NLP).

Kode ini ditulis dalam Python murni di atas *library* Numpy dan CuPy. Chainer adalah *framework* pertama yang menggunakan model arsitektur dinamis (seperti pada PyTorch).



GAMBAR 1.15.
Logo Chainer

Chainer beberapa kali memegang rekor mengalahkan *framework* yang lain (TensorFlow, MxNet dan CNTK) dalam hal efektivitas penskalaan ketika masalah pemodelan diselesaikan oleh Jaringan Saraf. Dengan tolok ukur kecepatan proses, Chainer lebih cepat daripada *framework* yang berorientasi Python lainnya. Terutama dengan kinerja pusat data berbasis GPU, Chainer memiliki kecepatan proses yang lebih baik daripada TensorFlow. TensorFlow sendiri dioptimalkan untuk arsitektur TPU.

j. DL4J

Bagi penggemar pemrograman Java, seharusnya tidak asing dengan DL4J (kependekan dari *Deep Learning for Java*). Pelatihan Jaringan Saraf dalam DL4J dilakukan secara paralel melalui iterasi menggunakan kluster. Proses ini didukung oleh arsitektur Hadoop dan Spark.



GAMBAR 1.16.
Logo DL4J

k. ONNX



GAMBAR 1.17.
Logo ONNX

Proyek ONNX lahir dari kolaborasi Microsoft dan Facebook dalam hal pencarian format terbuka untuk presentasi model *Deep Learning*. ONNX menyederhanakan proses transfer model antar algoritma yang bekerja dengan kecerdasan buatan. ONNX memungkinkan model untuk dilatih dalam satu *framework* dan ditransfer ke *framework* yang lain untuk menghasilkan kesimpulan. Model ONNX saat ini didukung oleh Caffe2, Microsoft *Cognitive Toolkit*, MXNet, PyTorch, konektor ke banyak *framework* dan library umum lainnya.

l. Darknet

Darknet adalah kerangka kerja Jaringan Saraf *open source* yang ditulis dalam bahasa pemrograman C dan CUDA. Cepat, mudah dipasang, dan mendukung komputasi CPU dan GPU.



GAMBAR 1.18.
Logo Darknet

m. Darkflow

Darkflow adalah *library Deep Learning* yang menerjemahkan *library* Darknet ke TensorFlow. Darkflow dapat digunakan dalam aplikasi lain yang mendukung bahasa pemrograman Python sebagai alternatif jika suatu perangkat tidak mendukung CUDA dan cuDNN. Melalui konfigurasi yang sederhana, adanya file *cfg* merupakan kode untuk model sedangkan *weights* merupakan bobot hasil pelatihan yang dapat digunakan untuk melakukan *transfer learning*.

n. YOLO

You only look once (YOLO) adalah sistem deteksi objek *real-time* yang canggih. Pada Pascal Titan X dapat memproses gambar pada 30 FPS dan memiliki pemetaan sebesar 57,9% pada COCO *test-dev*. YOLOv3 menggunakan beberapa trik untuk meningkatkan pelatihan dan meningkatkan kinerja, termasuk: prediksi multi-skala, pengklasifikasi *backbone* yang lebih baik, dan banyak lagi.



GAMBAR 1.19.
Logo YOLO

Jika mayoritas sistem deteksi menggunakan pengklasifikasian atau *localizer* untuk melakukan deteksi dengan menerapkan model ke gambar di beberapa lokasi dan skala dan memberi nilai pada gambar sebagai bahan untuk pendeteksian. YOLO menggunakan pendekatan yang berbeda, yakni menerapkan Jaringan Saraf tunggal pada keseluruhan gambar. Jaringan ini

akan membagi gambar menjadi wilayah-wilayah kemudian memprediksi kotak pembatas dan probabilitas. Untuk setiap kotak wilayah pembatas ditimbang probabilitasnya untuk diklasifikasi sebagai objek atau bukan. YOLO memiliki arsitektur CNN.



Bab 2

Jaringan Saraf Tiruan

2.1. Jaringan Saraf Dasar

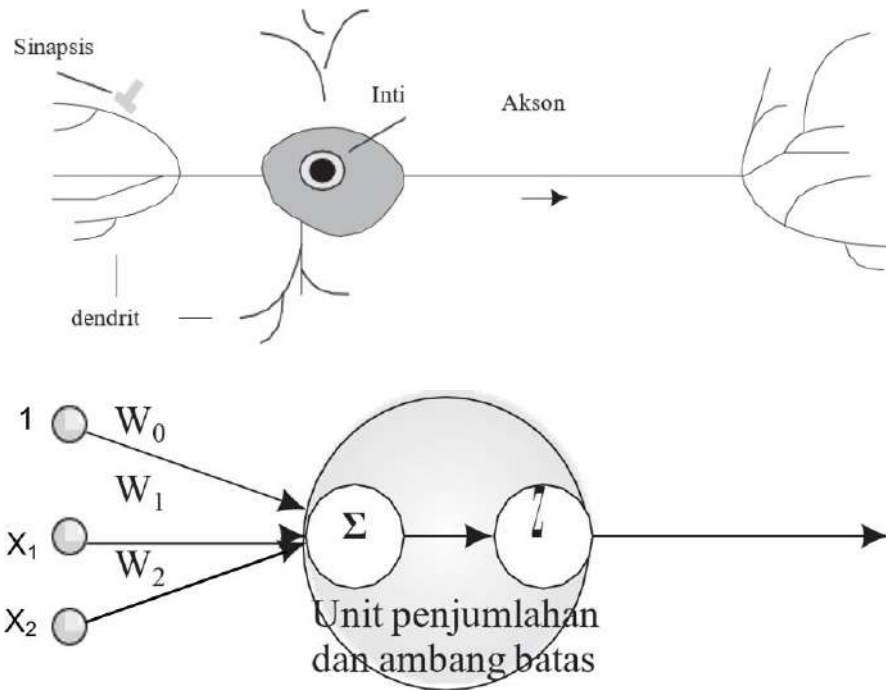
Seperti telah kami sampaikan pada bagian pengantar *Deep Learning* di muka, bahwa definisi yang berguna menetapkan bahwa *Deep Learning* berkaitan dengan “Jaringan Sel Saraf (*Neural Network*) dengan lebih dari dua lapisan.” Meskipun *Deep Learning* sendiri dianggap baru muncul pada tahun 2006, setelah Geoffrey Hinton memperkenalkan salah satu varian Jaringan Saraf Tiruan yang disebut *Deep Belief Nets* (Hinton, Osindero and Teh, 2006). Dengan demikian, mempelajari *Deep Learning* tentunya tidak bisa dipisahkan dengan mengulang kembali pemahaman kita tentang Jaringan Saraf Tiruan. Karena secara umum, konsep Jaringan Saraf Tiruan tetap digunakan pada konsep *Deep Learning*. Meskipun dalam perkembangannya muncul ide-ide baru yang luar biasa untuk efisiensi dan optimasi dalam proses pembelajaran dan penerapannya.

Konsep Jaringan Saraf sendiri sudah ada sejak tahun 1943 ketika Warren McCulloch dan Walter Pitts memperkenalkan perhitungan model Jaringan Saraf Tiruan yang pertama kalinya (Alavala, 2007). Mereka melakukan kombinasi beberapa pemrosesan unit sederhana bersama-sama yang mampu memberikan

peningkatan secara keseluruhan pada kekuatan komputasi. Jaringan lapisan-tunggal (*single-layer*), dengan fungsi aktivasi ambang, diperkenalkan oleh Rosenblatt pada tahun 1959. Jenis jaringan ini disebut *perceptron*. Pada tahun 1960-an secara eksperimental diperlihatkan bahwa *perceptron* dapat memecahkan banyak masalah. Tetapi di sisi lain, banyak masalah, yang tampaknya tidak lebih sulit, tidak dapat diselesaikan. Keterbatasan *perceptron* satu lapis ini secara matematis ditunjukkan oleh Minsky dan Papert dalam buku mereka *Perceptron* [1969] (Alavala, 2007). Setelah publikasi ini Jaringan Saraf seperti kehilangan daya tariknya selama hampir dua dekade. Pada pertengahan 1980-an, algoritma propagasi balik dilaporkan oleh Rumelhart, Hinton, dan Williams [1986] (Alavala, 2007), yang menghidupkan kembali studi tentang Jaringan Saraf. Arti penting dari algoritma baru ini adalah bahwa jaringan multi-pemain (*multiplayer*) dapat dilatih dengan menggunakannya dalam pemecahan masalah.

Seperti yang telah umum diketahui, Jaringan Saraf berusaha mensimulasikan otak manusia. Simulasi ini didasarkan pada pengetahuan saat ini tentang fungsi otak, dan pengetahuan ini bahkan pada primitif terbaiknya. Meskipun Jaringan Saraf Tiruan tidak sepenuhnya mewakili cara kerja atau pengoperasian otak manusia. Operasi otak diyakini didasarkan pada elemen dasar sederhana yang disebut neuron atau sel saraf, yang terhubung satu sama lain dengan jalur transmisi yang disebut akson dan garis reseptif yang disebut dendrit (lihat Gambar 2.1). Pembelajaran dapat didasarkan pada dua mekanisme, yaitu: pembuatan koneksi baru, dan modifikasi koneksi. Setiap neuron memiliki tingkat aktivasi yang berbeda dengan logika Boolean, berkisar antara nilai minimum dan maksimum.

Dalam Jaringan Saraf Tiruan, masukan neuron digabungkan secara linier dengan bobot yang berbeda. Hasil dari kombinasi ini kemudian dimasukkan ke dalam unit aktivasi non-linier (fungsi aktivasi), yang dalam bentuk paling sederhana dapat menggunakan unit ambang batas (Lihat Gambar 2.1). Kemampuan belajar Jaringan Saraf Tiruan dicapai dengan menyajikan sekumpulan pelatihan dari contoh yang berbeda ke jaringan dan menggunakan algoritma pembelajaran, yang mengubah bobot (atau parameter dari fungsi aktivasi) sedemikian rupa sehingga jaringan akan mereproduksi keluaran yang benar atau sesuai target yang diharapkan ketika diberikan nilai masukan yang benar. Kesulitannya adalah bagaimana menjamin generalisasi dan menentukan kapan jaringan dianggap cukup terlatih. Secara umum, Jaringan Saraf menawarkan nonlinier, pemetaan masukan-keluaran, adaptif, dan toleran terhadap kesalahan.



GAMBAR 2.1.
Ilustrasi sederhana neuron biologis dan buatan (*perceptron*)
(Alavala, 2007)

2.2. Simpul Jaringan Saraf

Kapanpun kita mempelajari sesuatu, otak kita menyimpan pengetahuan tersebut. Komputer menggunakan memori untuk menyimpan informasi. Meskipun keduanya menyimpan informasi, mekanismenya sangat berbeda. Komputer menyimpan informasi di lokasi tertentu dari memori, sementara otak menyimpan informasi dengan cara mengubah asosiasi neuron. Neuron itu sendiri tidak memiliki kemampuan penyimpanan; ia hanya mengirimkan sinyal dari satu neuron ke neuron lainnya. Sementara otak adalah jaringan raksasa neuron-neuron ini, dan asosiasi neuron membentuk informasi spesifik.

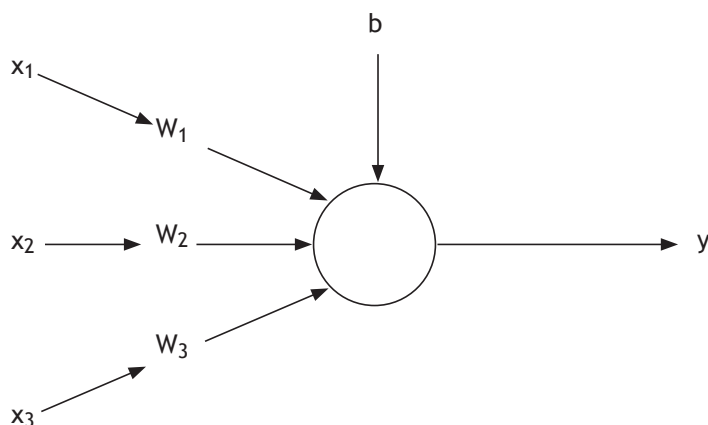
Jaringan Saraf meniru mekanisme otak. Karena otak terdiri dari koneksi banyak neuron, Jaringan Saraf dibangun dengan koneksi *node* atau biasa diterjemahkan simpul, yang merupakan elemen yang sesuai dengan neuron otak. Jaringan Saraf meniru asosiasi neuron, yang merupakan mekanisme terpenting

otak, menggunakan nilai bobot. Tabel 2.1 berikut merangkum analogi antara otak dan Jaringan Saraf.

TABEL 2.1.
Analogi antara otak dan Jaringan Saraf

Otak	Jaringan Saraf
Neuron	Simpul
Koneksi neuron	Bobot koneksi

Contoh sederhana berikut untuk pemahaman yang lebih baik tentang mekanisme Jaringan Saraf. Pertimbangkan simpul yang menerima tiga masukan, seperti yang ditunjukkan pada Gambar 2.2 (Kim, 2017).



GAMBAR 2.2.
Simpul yang menerima tiga masukan (Kim, 2017)

Lingkaran dan panah dari gambar masing-masing menunjukkan simpul dan aliran sinyal. Kemudian x_1 , x_2 , dan x_3 adalah sinyal masukan, w_1 , w_2 , dan w_3 adalah bobot untuk sinyal yang sesuai. Terakhir, b adalah bias, yang merupakan faktor lain yang terkait dengan penyimpanan informasi. Dengan kata lain, informasi Jaringan Saraf disimpan dalam bentuk bobot dan bias.

Sinyal masukan dari luar dikalikan dengan bobot sebelum mencapai simpul. Setelah sinyal tertimbang dikumpulkan di simpul, nilai-nilai ini ditambahkan menjadi jumlah tertimbang. Jumlah tertimbang dari contoh ini dihitung sebagai berikut (Kim, 2017):

$$v = (w_1 x_1) + (w_2 x_2) + (w_3 x_3) + b \quad (2.1)$$

Persamaan ini menunjukkan bahwa sinyal dengan bobot yang lebih besar memiliki efek yang lebih besar. Misalnya, jika bobot w_1 adalah 1, dan w_2 adalah 5, maka sinyal x_2 memiliki efek lima kali lebih besar daripada x_1 . Ketika w_1 adalah nol, x_1 tidak ditransmisikan ke simpul sama sekali. Ini berarti x_1 terputus dari simpul. Contoh ini menunjukkan bahwa bobot Jaringan Saraf meniru cara otak mengubah asosiasi neuron.

Persamaan jumlah tertimbang dapat ditulis dengan matriks sebagai berikut (Kim, 2017):

$$v = wx + b \quad (2.2)$$

dimana w dan x didefinisikan sebagai:

$$w = [w_1 \ w_2 \ w_3] \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (2.3)$$

Akhirnya, simpul memasukkan jumlah tertimbang ke dalam fungsi aktivasi dan menghasilkan keluaran jaringan. Fungsi aktivasi menentukan perilaku simpul, yang dinyatakan seperti Persamaan (2.4).

$$y = \varphi(v) \quad (2.4)$$

dimana dalam persamaan ini adalah fungsi aktivasi.

2.3. Fungsi Aktivasi

Sekarang kita akan melihat sejumlah fungsi aktivasi yang biasa digunakan untuk Jaringan Saraf. Beberapa sifat umum fungsi aktivasi dapat disebutkan sebagai berikut (Ketkar, 2017).

1. Secara teori, Jaringan Saraf dua lapisan yang digunakan untuk mendekati fungsi apa pun (dengan jumlah unit yang cukup di lapisan tersembunyi), digunakan fungsi aktivasi non-linier.

2. Sebuah fungsi yang terus menerus dapat dibedakan memungkinkan untuk menghitung gradien dan metode berbasis gradien digunakan untuk menemukan parameter yang meminimalkan fungsi kerugian kita atas data. Jika suatu fungsi tidak terus menerus dapat dibedakan, maka metode berbasis gradien tidak terlalu dapat membuat kemajuan sesuai yang diharapkan.
3. Fungsi yang jangkauannya terbatas (dibandingkan dengan tak terbatas) mengarah ke kinerja metode berbasis gradien yang lebih stabil.
4. Fungsi halus lebih disukai (bukti empiris) dan fungsi monolitik untuk satu lapisan menyebabkan permukaan kesalahan cembung (ini biasanya bukan menjadi pertimbangan pada *Deep Learning*).
5. Bersifat simetris di sekitar titik asal dan berperilaku seperti fungsi identitas di dekat titik asal ($f(x) = x$).

Macam-macam fungsi aktivasi yang biasa digunakan pada Jaringan Saraf Tiruan dan *Deep Learning*, antara lain:

a. **Fungsi Linear**

Fungsi Linear adalah fungsi paling sederhana yang mengubah masukan menggunakan Persamaan (2.5) (Ketkar, 2017):

$$y = w \cdot x + b \quad (2.5)$$

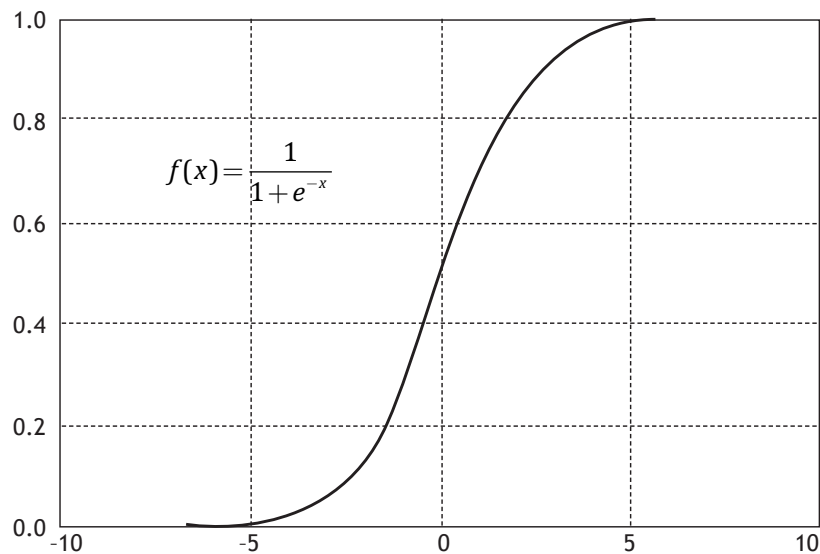
Seperti yang ditunjukkan oleh namanya, fungsi aktivasi ini tidak memiliki perilaku non-linier dan biasanya digunakan untuk menghasilkan rata-rata dari distribusi Gaussian bersyarat. Fungsi linier membuat pembelajaran berbasis gradien menjadi tugas yang cukup mudah.

b. **Fungsi Sigmoid**

Fungsi Sigmoid mengubah masukan menggunakan Persamaan (2.6) (Ketkar, 2017):

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.6)$$

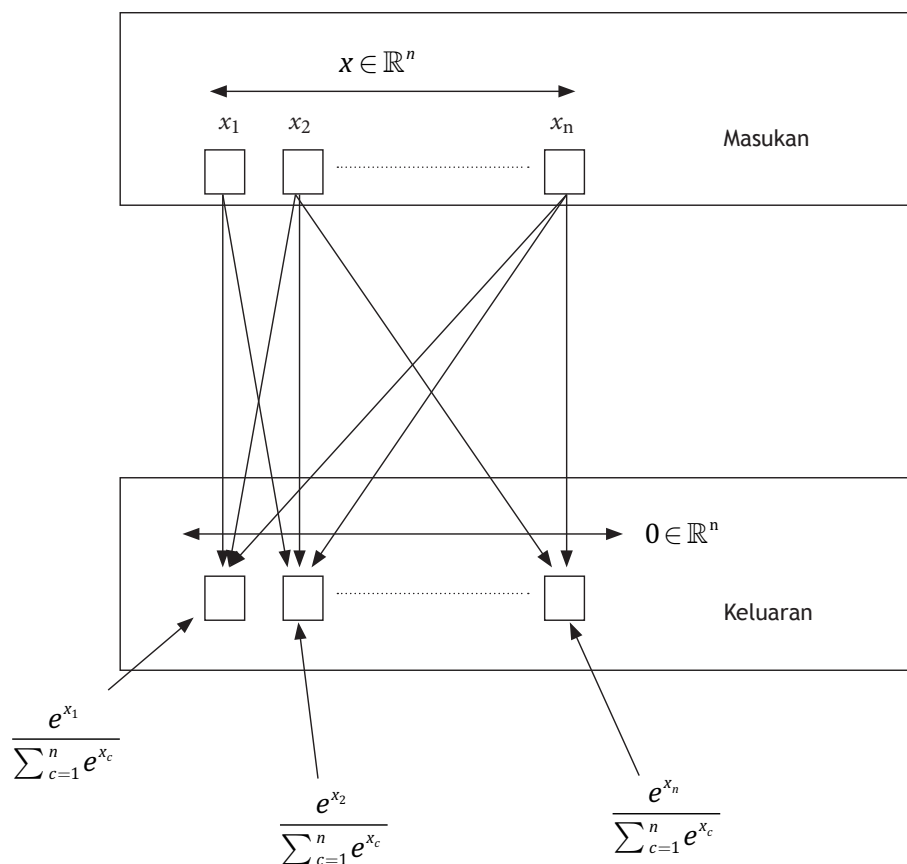
Fungsi sigmoid dapat digunakan pada lapisan keluaran sehubungan dengan entropi silang biner untuk masalah klasifikasi biner. Keluaran fungsi aktivasi ini dapat memodelkan distribusi Bernoulli atas keluaran y yang dikondisikan melalui x .



GAMBAR 2.3.
Fungsi Sigmoid (Ketkar, 2017)

c. Lapisan Softmax

Lapisan Softmax biasanya digunakan sebagai lapisan keluaran untuk tugas-tugas multi-klasifikasi dalam hubungannya dengan fungsi kerugian *Cross Entropy*. Lihat Gambar 2.4. Lapisan Softmax menormalkan keluaran dari lapisan sebelumnya sehingga jumlahnya menjadi satu. Biasanya, unit model lapisan sebelumnya adalah skor yang tidak dinormalisasi tentang seberapa besar kemungkinan masukan tersebut termasuk dalam kelas tertentu. Lapisan softmax menormalkan ini sehingga keluarannya mewakili probabilitas untuk setiap kelas.



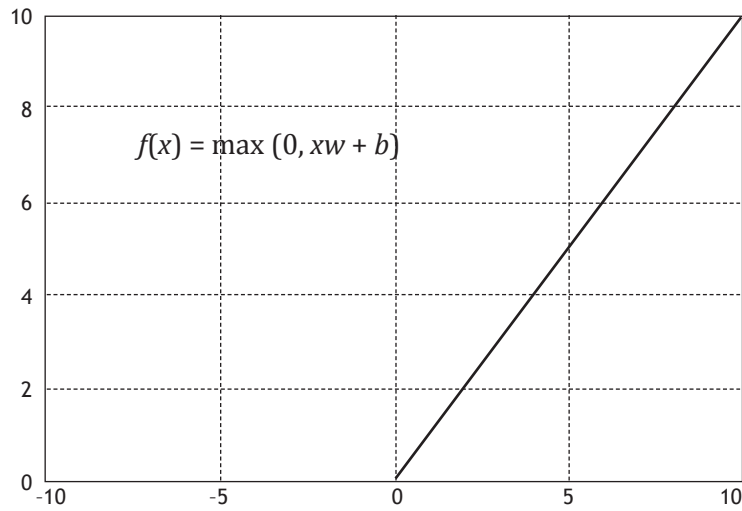
GAMBAR 2.4.
Lapisan Softmax (Ketkar, 2017)

d. ***Rectified Linear Unit (ReLU)***

Fungsi aktivasi ReLU yang digunakan dalam hubungannya dengan transformasi linier mengubah masukan menggunakan Persamaan 2.7 (Ketkar, 2017):

$$f(x) = \max(0, wx + b) \quad (2.7)$$

Fungsi aktivasi yang mendasari adalah $f(x) = \max(0, x)$; lihat Gambar 2.5. Fungsi aktivasi ReLU lebih sering digunakan pada lapisan tersembunyi belakang ini. Hasil menunjukkan bahwa fungsi aktivasi ReLU mengarah ke gradien yang besar dan konsisten, yang membantu pembelajaran berbasis gradien.

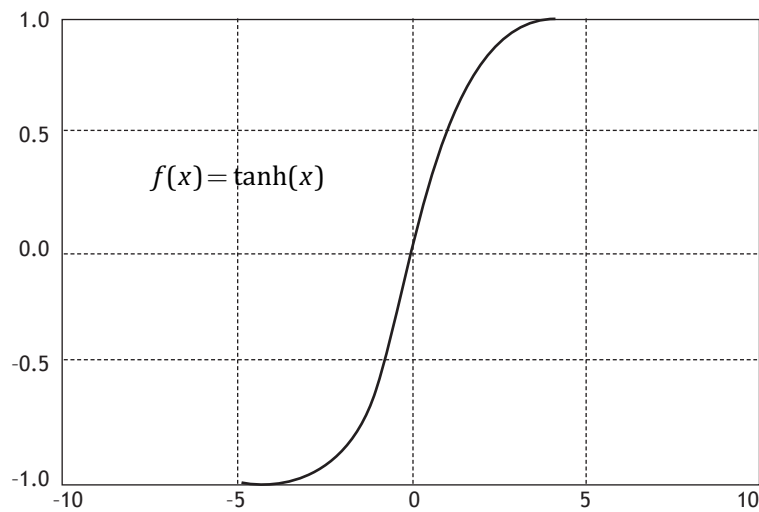


GAMBAR 2.5.
Fungsi aktivasi ReLU (Ketkar, 2017)

e. Tangen Hiperbolik

Fungsi Tangen Hiperbolik mengubah masukan (digunakan bersama dengan transformasi linier) menggunakan Persamaan 2.8 (Ketkar, 2017):

$$f(x) = \tanh(x) \quad (2.8)$$



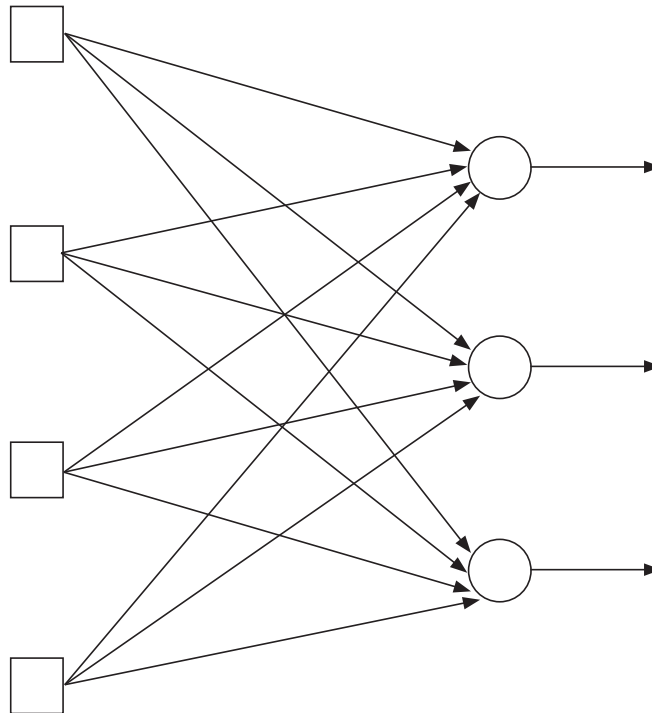
GAMBAR 2.6.
Fungsi aktivasi Tangen Hiperbolik (Ketkar, 2017)

2.4. Arsitektur Jaringan Saraf

Jaringan Saraf tiruan secara arsitektur dapat diklasifikasikan menjadi dua jenis yaitu lapisan tunggal (*single layer*) dan multi-lapisan (*multilayer*).

2.4.1. Lapisan Tunggal

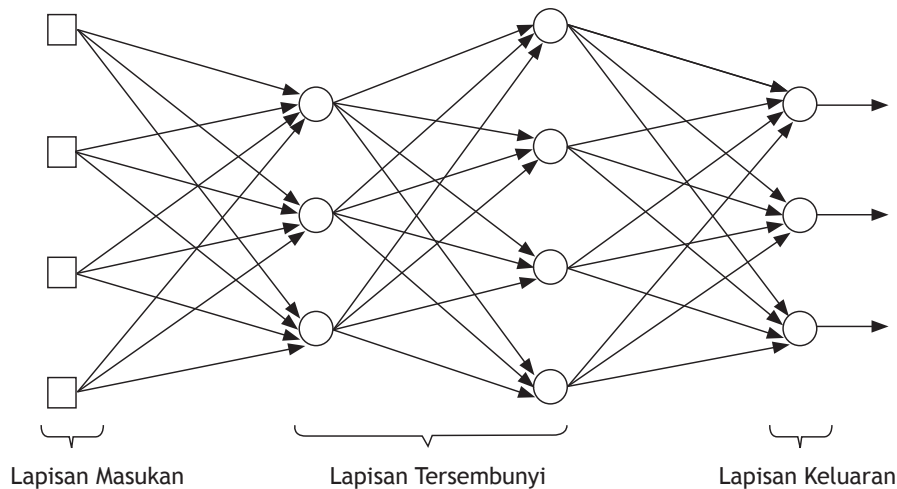
Neuron-neuron dikelompokkan menjadi dua bagian yaitu unit-unit masukan dan unit-unit keluaran. Unit-unit masukan menerima masukan dari luar sedangkan unit-unit keluaran menghasilkan respon sesuai dengan masukannya. Gambaran Jaringan Saraf lapisan tunggal seperti yang ditunjukkan pada Gambar 2.7 (Kim, 2017). Kelompok simpul persegi pada Gambar 2.7 disebut lapisan masukan, sedangkan kelompok simpul lingkaran disebut lapisan keluaran.



GAMBAR 2.7.
Jaringan Saraf lapisan tunggal (Kim, 2017)

2.4.2. Multi-Lapisan

Karena otak adalah jaringan neuron yang sangat besar, Jaringan Saraf adalah jaringan simpul. Berbagai Jaringan Saraf dapat dibuat tergantung pada bagaimana simpul-simpul terhubung. Arsitektur Jaringan Saraf lain selain lapisan tunggal yang paling umum digunakan menggunakan struktur simpul berlapis (multi-lapisan) seperti yang ditunjukkan pada Gambar 2.8 (Kim, 2017).



GAMBAR 2.8.
Struktur multi-lapisan (Kim, 2017)

Kelompok simpul persegi pada Gambar 2.8 disebut lapisan masukan. Simpul dari lapisan masukan hanya bertindak sebagai bagian yang mentransmisikan sinyal masukan ke simpul berikutnya. Oleh karena itu, mereka tidak menghitung jumlah tertimbang dan fungsi aktivasi. Inilah alasan mengapa simpul tersebut diindikasikan dengan kotak dan dibedakan dari simpul lingkaran lainnya. Sebaliknya, grup simpul paling kanan disebut lapisan keluaran. Keluaran dari simpul ini menjadi hasil akhir dari Jaringan Saraf Tiruan. Lapisan di antara lapisan masukan dan keluaran disebut lapisan tersembunyi. Mereka diberi nama ini karena tidak dapat diakses dari luar Jaringan Saraf.

Dengan demikian, Jaringan Saraf telah dikembangkan dari arsitektur sederhana menjadi struktur yang lebih kompleks. Awalnya, pelopor Jaringan Saraf memiliki

arsitektur yang sangat sederhana dengan hanya lapisan masukan dan keluaran seperti pada Gambar 2.7, yang tadi disebut dengan Jaringan Saraf lapisan tunggal. Saat lapisan tersembunyi ditambahkan ke Jaringan Saraf lapisan tunggal, ini menghasilkan Jaringan Saraf lapisan ganda. Oleh karena itu, Jaringan Saraf multi-lapisan terdiri dari lapisan masukan, lapisan tersembunyi, dan lapisan keluaran. Jaringan Saraf yang memiliki satu lapisan tersembunyi disebut Jaringan Saraf dangkal (*shallow neural network*) atau Jaringan Saraf vanilla. Jaringan Saraf multi-lapisan yang berisi dua atau lebih lapisan tersembunyi disebut Jaringan Saraf dalam (*deep neural network*) (Kim, 2017). Sebagian besar Jaringan Saraf kontemporer yang digunakan dalam aplikasi praktis adalah Jaringan Saraf dalam.

2.5. Pelatihan Jaringan Saraf Tiruan

Pelatihan Jaringan Saraf Tiruan bertujuan untuk mencari bobot-bobot yang terdapat pada setiap lapisan. Setidaknya terdapat dua jenis pelatihan yang populer dalam sistem Jaringan Saraf Tiruan (Rahmat and Nugroho, 2019), yaitu:

- Pembelajaran yang Diawasi (*Supervised Learning*).
- Pembelajaran Tanpa Pengawasan (*Unsupervised Learning*).

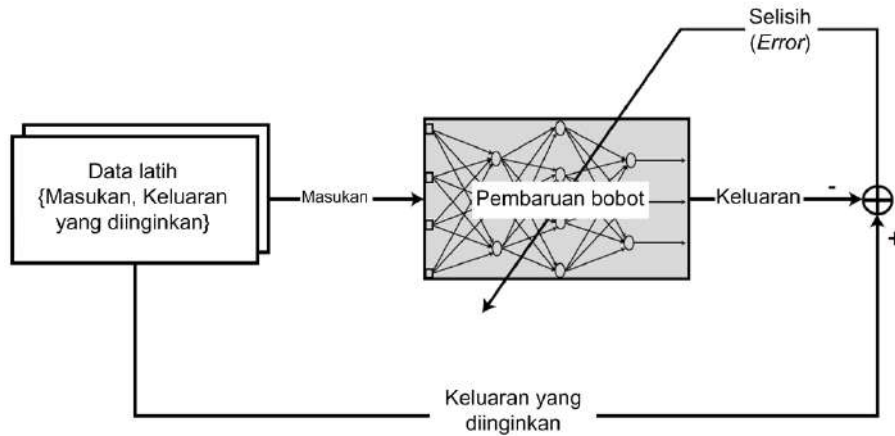
2.5.1. Pembelajaran yang Diawasi

Dalam proses pelatihan ini, jaringan dilatih dengan cara diberikan data-data yang disebut Data Latih yang terdiri atas pasangan data masukan-keluaran yang diinginkan. Jaringan Saraf memiliki memori asosiatif (*associative memory*). Setelah jaringan dilatih, memori asosiatif dapat mengingat suatu pola. Jika jaringan diberi masukan baru, jaringan dapat menghasilkan keluaran seperti yang diinginkan berdasarkan pola yang sudah ada.

Secara umum, pembelajaran yang diawasi dari Jaringan Saraf dilakukan dengan langkah-langkah berikut (Kim, 2017):

- 1) Inisialisasi bobot dengan nilai yang memadai.
- 2) Ambil “masukan” dari data pelatihan, yang diformat sebagai {masukan, keluaran yang diinginkan}, dan masukkan ke dalam Jaringan Saraf. Dapatkan keluaran dari Jaringan Saraf dan hitung kesalahan dari keluaran yang diinginkan.
- 3) Sesuaikan bobot untuk mengurangi kesalahan.
- 4) Ulangi Langkah 2-3 untuk semua data pelatihan.

Gambar 2.9 mengilustrasikan konsep pembelajaran yang diawasi dari Jaringan Saraf (Kim, 2017).



GAMBAR 2.9.
Konsep pembelajaran yang diawasi (Kim, 2017)

Untuk metode pembelajaran yang diawasi yang berguna untuk belajar *Deep Learning*, setidaknya yang terkenal bisa disebutkan:

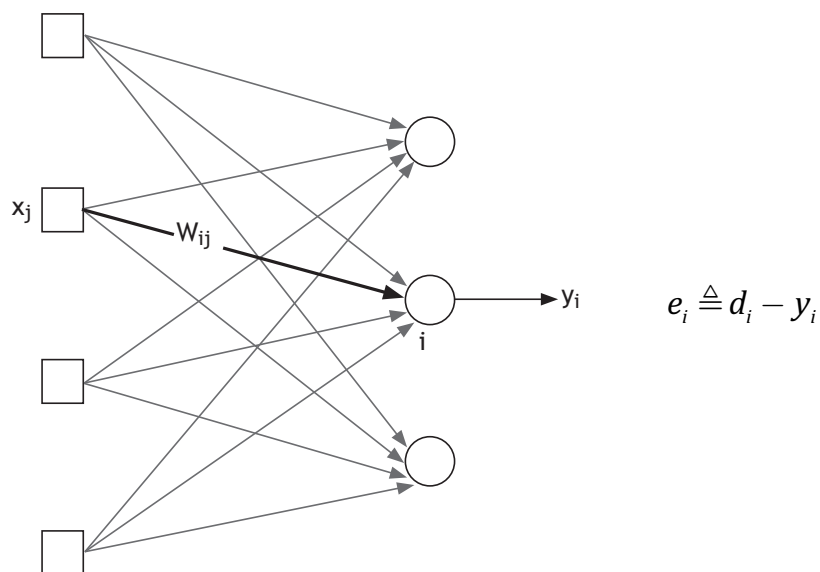
- Aturan Delta (*Delta Rule*), dan
- Algoritma Propagasi Balik (*Back-Propagation Algorithm*).

a. Aturan Delta

Berikut ini akan dijelaskan bagaimana metode pembelajaran dengan menggunakan Aturan Delta pada Jaringan Saraf Lapisan Tunggal. Seperti yang telah dibahas sebelumnya, jaringan saraf tiruan menyimpan informasi dalam bentuk bobot. Oleh karena itu, untuk melatih jaringan saraf dengan informasi baru, bobot harus diubah yang sesuai. Pendekatan sistematis untuk memodifikasi bobot sesuai dengan informasi yang diberikan disebut aturan pembelajaran. Karena pelatihan adalah satu-satunya cara jaringan saraf untuk menyimpan informasi secara sistematis, maka aturan pembelajaran merupakan komponen yang penting dalam penelitian jaringan saraf.

Aturan delta, aturan pembelajaran representatif dari jaringan saraf lapisan tunggal. Meskipun tidak mampu melatih jaringan saraf multi-lapisan, aturan

delta ini sangat berguna untuk mempelajari konsep-konsep penting dari aturan pembelajaran jaringan saraf. Perhatikan jaringan saraf lapisan tunggal, seperti yang ditunjukkan pada Gambar 2.10 (Kim, 2017). Dalam gambar, d_i adalah keluaran yang diinginkan (target) dari simpul keluaran i .



GAMBAR 2.10.
Contoh jaringan saraf lapisan tunggal (Kim, 2017)

Secara ringkas, aturan delta menyesuaikan bobot dengan algoritma berikut (Kim, 2017):

“Jika simpul masukan berkontribusi pada kesalahan keluaran simpul, maka bobot antara dua simpul disesuaikan secara proporsional terhadap nilai masukan, x_j dan kesalahan keluaran, e_i .”

Aturan ini dapat diekspresikan dalam persamaan sebagai (Kim, 2017):

$$w_{ij} \leftarrow w_{ij} + \alpha e_i x_j \quad (2.9)$$

dimana

x_j = Keluaran dari simpul masukan j , ($j = 1, 2, 3$).

e_i = Kesalahan dari simpul keluaran i .

w_{ij} = Bobot antara simpul keluaran i dan masukan simpul j .

α = Kecepatan pembelajaran ($0 < \alpha \leq 1$).

Kecepatan pembelajaran, α , menentukan seberapa banyak bobot berubah setiap waktu. Jika nilai ini terlalu tinggi, keluaran jaringan berkeliaran di sekitar solusi dan gagal untuk menyatu. Sebaliknya, jika terlalu rendah, penghitungan mencapai solusi terlalu lambat.

Dengan demikian, proses pelatihan menggunakan aturan delta untuk jaringan syaraf lapisan tunggal, dijalankan sesuai dengan urutan sebagai berikut:

1. Inisialisasi bobot dengan nilai yang memadai.
2. Ambil “masukan” dari data pelatihan {input, keluaran yang diinginkan} dan masukkan ke jaringan saraf. Hitung kesalahan keluaran, y_i , terhadap keluaran yang diinginkan, d_i .

$$e_i = d_i - y_i \quad (2.10)$$

3. Hitung pembaruan bobot menurut aturan delta sesuai Persamaan (2.9), dalam bentuk lain yaitu:

$$\Delta w_{ij} = \alpha e_i x_j \quad (2.11)$$

4. Sesuaikan bobot dengan cara sebagai berikut:

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad (2.12)$$

5. Lakukan Langkah 2-4 untuk semua data pelatihan.
6. Ulangi Langkah 2-5 hingga kesalahan mencapai tingkat toleransi yang dapat diterima.

Dari aturan delta, ada konsep penting yang disebut Aturan Delta Umum (*Generalized Delta Rule*). Pada bentuk yang lebih umum dari aturan delta, Persamaan (2.9) sebelumnya, diubah menjadi Persamaan (2.13) berikut ini:

$$w_{ij} \leftarrow w_{ij} + \alpha \delta_i x_j \quad (2.13)$$

Ini sama dengan aturan delta pada Persamaan (2.9) sebelumnya, kecuali bahwa e_i diganti dengan δ_i . Dalam persamaan ini, δ_i didefinisikan sebagai (Kim, 2017):

$$\delta_i = \varphi' (v_i) e_i \quad (2.14)$$

dimana

e_i = Kesalahan dari simpul keluaran i .

v_i = Jumlah tertimbang dari simpul keluaran i , sesuai Persamaan (2.2).

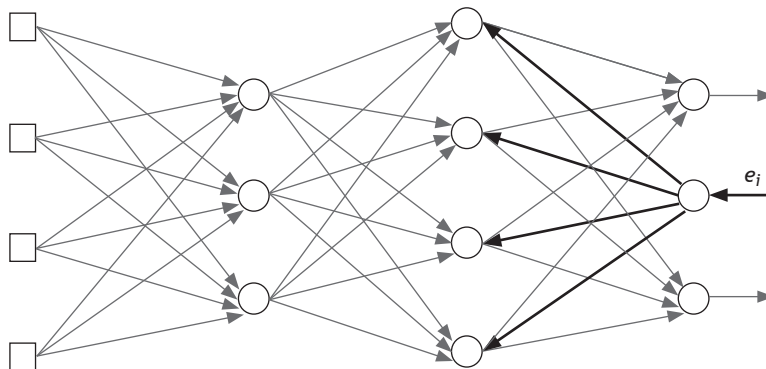
φ' = Turunan dari fungsi aktivasi φ dari node keluaran.

Sehingga bentuk yang lebih umum dari aturan delta menjadi:

$$w_{ij} \leftarrow w_{ij} + \alpha \varphi' (v_i) e_i x_j \quad (2.15)$$

b. Algoritma Propagasi Balik

Selanjutnya, untuk metode pembelajaran yang diawasi yang lain adalah algoritma propagasi balik. Algoritma propagasi balik, merupakan aturan pembelajaran representatif dari jaringan saraf multi-lapisan. Signifikansi dari algoritma propagasi balik adalah menyediakan metode sistematis untuk menentukan kesalahan dari simpul tersembunyi. Setelah kesalahan lapisan tersembunyi ditentukan, aturan delta diterapkan untuk menyesuaikan bobot. Lihat Gambar 2.11.



GAMBAR 2.11.
Ilustrasi propagasi balik (Kim, 2017)

Data masukan dari jaringan saraf berjalan melalui lapisan masukan, lapisan tersembunyi, dan lapisan keluaran. Sebaliknya, dalam algoritma propagasi balik, kesalahan keluaran dimulai dari lapisan keluaran dan bergerak mundur hingga mencapai lapisan tersembunyi paling akhir di sebelah kanan lapisan masukan.

Secara ringkas, proses pelatihan jaringan saraf menggunakan algoritma propagasi balik dijalankan sesuai dengan urutan berikut ini (Kim, 2017):

1. Inisialisasi bobot dengan nilai yang memadai.
2. Masukkan masukan dari data pelatihan {masukan, keluaran yang diinginkan} dan dapatkan keluaran jaringan saraf. Hitung kesalahan keluaran terhadap keluaran yang diinginkan dan delta, δ , dari simpul keluaran.

$$e = d - y$$

$$\delta = \varphi' (v) e$$

3. Menyebarkan delta simpul keluaran, δ , mundur, dan menghitung delta simpul berikutnya.

$$e^{(k)} = W^T \delta$$

$$\delta^{(k)} = \varphi' (v^{(k)}) e^{(k)}$$

4. Ulangi Langkah 3 hingga mencapai lapisan tersembunyi yang aktif selanjutnya.
5. Perbaharui bobot sesuai dengan aturan pembelajaran berikut.

$$\Delta w_{ij} = \alpha \delta_i x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

6. Ulangi Langkah 2-5 untuk semua data pelatihan.
7. Ulangi Langkah 2-6 hingga kesalahan mencapai tingkat toleransi yang dapat diterima.

2.5.2. Pembelajaran Tanpa Pengawasan

Dalam proses pembelajaran ini, jaringan dilatih hanya dengan diberi data masukan yang memiliki kesamaan sifat tanpa disertai keluaran. Seperti namanya, pembelajaran jenis ini dilakukan tanpa pengawasan seorang guru. Proses pembelajaran ini mandiri. Selama pelatihan jaringan saraf tanpa pengawasan, vektor masukan dengan tipe serupa digabungkan untuk membentuk kluster. Ketika pola masukan baru diterapkan, maka jaringan saraf memberikan tanggapan keluaran yang menunjukkan kelas yang memiliki pola masukan. Dalam hal ini, tidak akan ada umpan balik dari lingkungan tentang apa yang seharusnya menjadi keluaran yang diinginkan dan apakah itu benar atau salah. Oleh karena itu, dalam pembelajaran jenis ini jaringan itu sendiri harus menemukan pola, fitur dari data masukan dan hubungan untuk data masukan melalui keluaran.

Jenis jaringan ini didasarkan pada aturan pembelajaran kompetitif dan akan menggunakan strategi di mana ia memilih neuron dengan total masukan terbesar sebagai pemenang. Hubungan antara neuron keluaran menunjukkan persaingan di antara mereka dan salah satunya akan menjadi 'On' yang berarti akan menjadi pemenang dan yang lainnya akan menjadi 'Off'.

Beberapa jaringan yang didasarkan pada konsep pembelajaran tanpa pengawasan dapat disebutkan di sini, namun tidak dibahas secara detail dalam buku ini, yaitu:

- Jaringan Hamming (*Hamming Network*)
- Max Net
- Pembelajaran Kompetitif (*Competitive Learning*)
- Algoritma Pengelompokan K-means (*K-means Clustering Algorithm*), dan
- Neokognitron (*Neocognitron*).



Bab 3

Deep Learning dengan Tensorflow

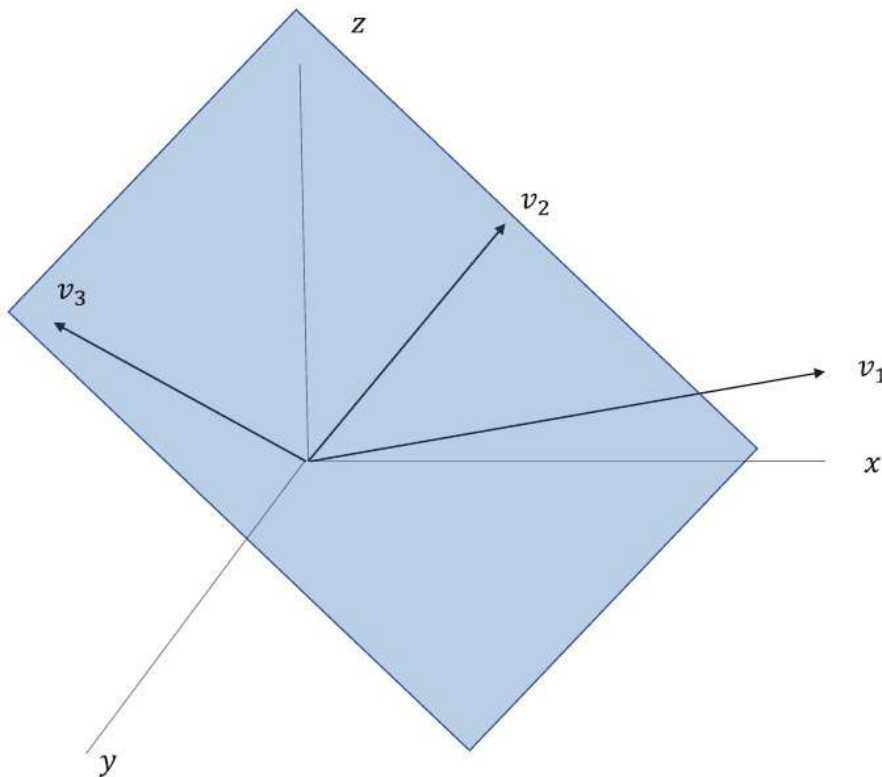
3.1. Vektor, Skalar, Matriks, dan Tensor

Matematika adalah inti dari semua algoritma pembelajaran mesin. Pemahaman yang kuat tentang konsep inti matematika sangat membantu dalam memungkinkan seseorang memilih algoritma yang tepat untuk masalah pembelajaran mesin tertentu, untuk mencapai tujuan akhirnya. Pemahaman konsep matematika yang baik memungkinkan seseorang untuk menyesuaikan model pembelajaran mesin atau *Deep Learning* dengan lebih baik. Termasuk memahami apa yang mungkin menjadi penyebab algoritma tidak berfungsi seperti yang diharapkan.

Matematika sebagai subjek sangat luas, tetapi ada beberapa topik khusus yang harus diperhatikan oleh para profesional atau penggemar pembelajaran mesin dalam hal ini *Deep Learning*. Buku ini tidak membahas kebutuhan konsep matematika penunjang secara lengkap, dapat dibaca pada buku-buku referensi lain yang membahas tentang: Aljabar linier, Probabilitas dan statistik, Kalkulus, serta Teknik Optimasi dan formulasi algoritma pembelajaran mesin lainnya. Namun setidaknya ada beberapa istilah penting seperti Vektor, Skalar, Matriks, dan Tensor coba kami ringkaskan sebelum dimulai pembahasan tentang Tensorflow.

3.1.1. Vektor

Deep Learning berurusan dengan data multidimensi dan manipulasinya, vektor memainkan peran penting di hampir setiap algoritmanya. Vektor merupakan sebuah besaran yang memiliki arah. Vektor digambarkan sebagai panah dengan yang menunjukkan arah vektor dan panjang garisnya disebut besar vektor. Vektor terdiri dari larik angka, baik kontinu maupun diskrit. Ruang yang terdiri dari vektor disebut ruang vektor. Dimensi ruang vektor dapat menjadi terbatas atau tidak terbatas. Diilustrasikan pada Gambar 3.1 adalah ruang vektor tiga dimensi di mana v_1 , v_2 dan v_3 adalah vektor dan P adalah bidang 2-D di dalam ruang vektor tiga dimensi (Pattanayak, 2017).



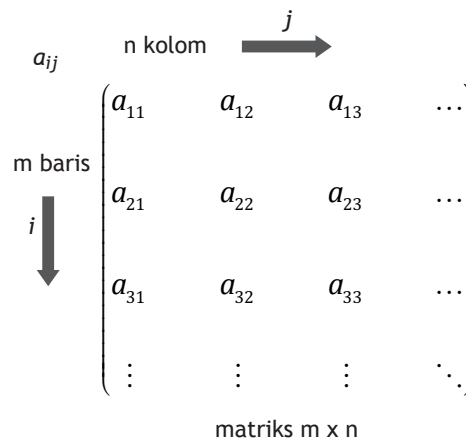
GAMBAR 3.1.
Ruang vektor tiga dimensi dengan vektor dan bidang vektor
(Pattanayak, 2017)

3.1.2. Skalar

Vektor satu dimensi disebut skalar. Skalar adalah besaran yang hanya memiliki besaran dan tidak memiliki arah. Skalar hanya memiliki satu arah yang dapat ia tempuh, dengan arah yang tidak penting. Sehingga yang dipentingkan hanya besarannya. Contoh: muatan listrik, jarak, energi, massa, daya, suhu, waktu, volume, usaha, dan lain-lain.

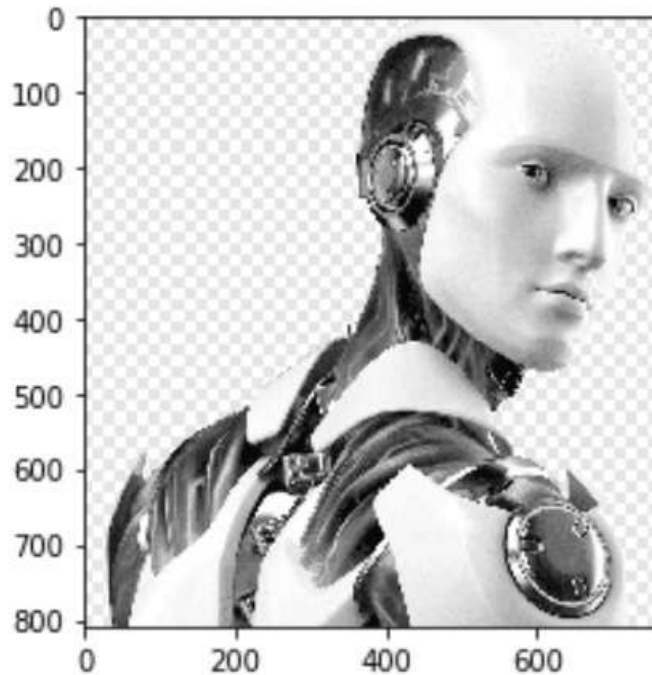
3.1.3. Matriks

Matriks adalah larik angka dua dimensi yang disusun dalam baris dan kolom. Besar kecilnya matriks ditentukan oleh panjang baris dan panjang kolomnya. Jika matriks A memiliki m baris dan n kolom, matriks tersebut dapat direpresentasikan sebagai objek persegi panjang (lihat Gambar 3.2) yang memiliki elemen m x n, dan dapat dilambangkan sebagai A_{mn} .



GAMBAR 3.2.
Struktur matriks

Beberapa vektor yang termasuk dalam ruang vektor yang sama membentuk matriks. Misalnya, gambar dalam skala abu-abu disimpan dalam bentuk matriks. Ukuran gambar menentukan ukuran matriks gambar, dan setiap sel matriks memiliki nilai 0–255 yang mewakili intensitas piksel. Diilustrasikan pada Gambar 3.3 adalah gambar *grayscale* yang diikuti dengan representasi matriksnya.



(a)

```
[ [0.9999  0.9999  0.9999  ... 0.9999  0.9999  0.9999  ]
  [0.9999  0.9999  0.9999  ... 0.9999  0.9999  0.9999  ]
  [0.9999  0.9999  0.9999  ... 0.9999  0.9999  0.9999  ]
  ...
  [0.90187059 0.90187059 0.90187059 ... 0.90187059 0.90187059 0.90187059]
  [0.90187059 0.90187059 0.90187059 ... 0.90187059 0.90187059 0.90187059]
  [0.90187059 0.90187059 0.90187059 ... 0.90187059 0.90187059 0.90187059]]
```

(b)

GAMBAR 3.3.

Contoh representasi matriks dari gambar grayscale

3.1.4. Tensor

Tensor adalah deretan angka multidimensi. Faktanya, vektor dan matriks dapat diperlakukan sebagai tensor 1-D dan 2-D. Dalam *Deep Learning*, tensor banyak digunakan untuk menyimpan dan memproses data. Misal suatu citra dalam RGB disimpan dalam tensor tiga dimensi, dimana sepanjang satu dimensi terdapat sumbu horizontal dan sepanjang dimensi lainnya terdapat sumbu vertikal, dan dimensi ketiga bersesuaian dengan tiga kanal warna yaitu Merah, Hijau, dan Biru.

Contoh lainnya adalah tensor empat dimensi yang digunakan dalam memasukkan gambar melalui tumpukan mini dalam jaringan neural konvolusional. Sepanjang dimensi pertama berupa nomor gambar dalam kelompok, sepanjang dimensi kedua berupa saluran warna, serta dimensi ketiga dan keempat berupa lokasi piksel dalam arah horizontal dan vertikal. Dari istilah Tensor inilah lahir TensorFlow yang dikembangkan oleh Google.

3.2. Instalasi Python Jupyter Notebook

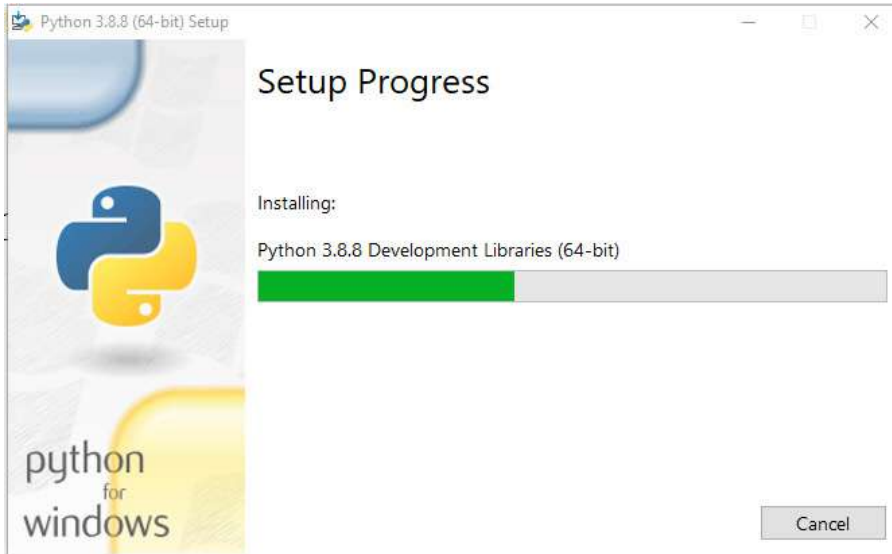
Sebelum kita bersenang-senang bereksperimen dengan *Deep Learning* menggunakan TensorFlow, kita pastikan program Python sudah terinstal di komputer atau laptop kita masing-masing terlebih dahulu. Jika belum memiliki program Python di komputer, maka terlebih dahulu silahkan download Python dari *website* resminya, yaitu yang ada di alamat <https://www.python.org>. Saat buku ini ditulis, TensorFlow versi terbaru yang ada adalah versi 2.4.1. Dan hanya bisa jalan sampai Python 3.8. Oleh karena itu silahkan download Python 3.8 versi terakhir saja, yaitu versi 3.8.8. Silahkan download di alamat: <https://www.python.org/downloads/release/python-388>. Atau untuk versi windows bisa langsung download di alamat: <https://www.python.org/ftp/python/3.8.8/python-3.8.8-amd64.exe>.

Setelah didownload, silahkan dilakukan proses instalasi. Jangan lupa sebagai Administrator (*run as administrator*). Pastikan tampil seperti pada Gambar 3.4.

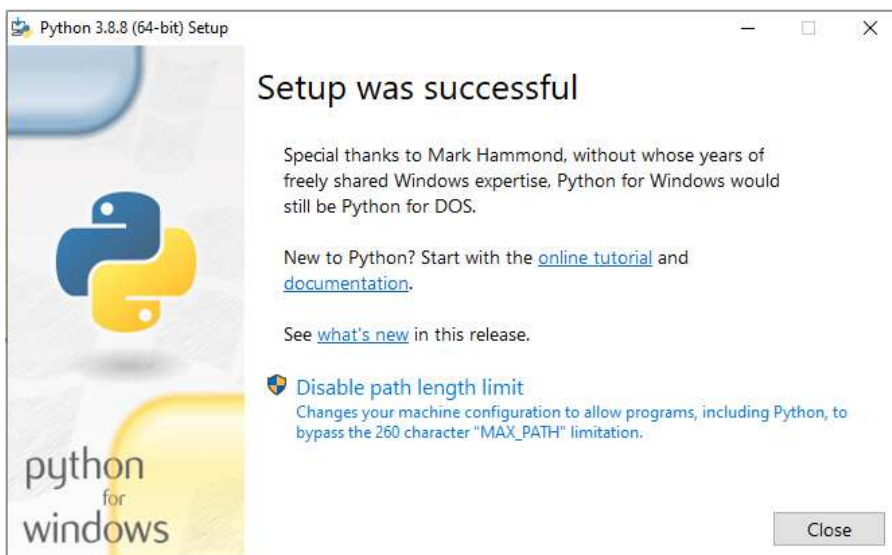


GAMBAR 3.4.
Instalasi Python

Jangan lupa pilih (Check) *Add Python 3.8 to PATH*. Silahkan tekan *Install Now*.
Tunggu sampai proses instalasi selesai.

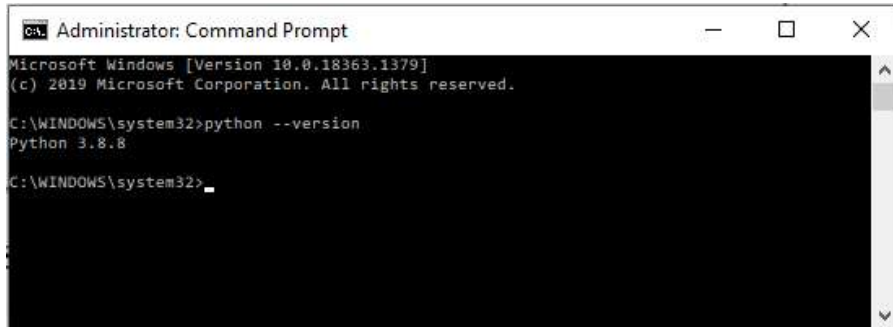


GAMBAR 3.5.
Proses instalasi



GAMBAR 3.6.
Instalasi Python selesai

Cek versi Python yang sudah terinstall di *command prompt* sebagai Administrator:
`python --version`



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

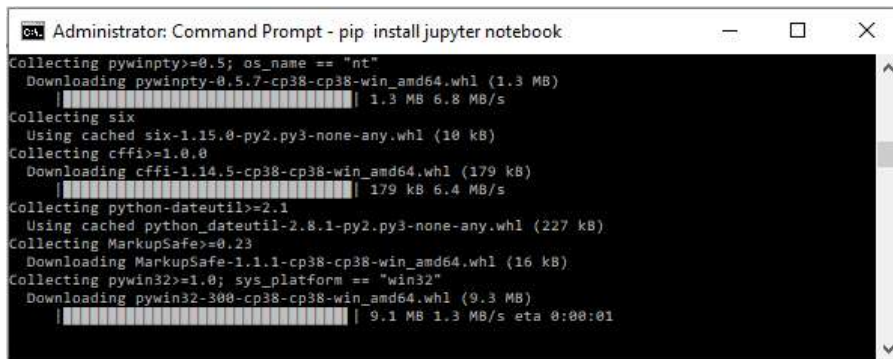
C:\WINDOWS\system32>python --version
Python 3.8.8

C:\WINDOWS\system32>
```

GAMBAR 3.7.
Versi Python

Selanjutnya, untuk menginstal Jupyter Notebook, silahkan ketikkan perintah berikut:

`pip install jupyter notebook`



```
Administrator: Command Prompt - pip install jupyter notebook
Collecting pywinpty>=0.5; os_name == "nt"
  Downloading pywinpty-0.5.7-cp38-cp38-win_amd64.whl (1.3 MB)
    |#####| 1.3 MB 6.8 MB/s
Collecting six
  Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting cffi>=1.0.0
  Downloading cffi-1.14.5-cp38-cp38-win_amd64.whl (179 kB)
    |#####| 179 kB 6.4 MB/s
Collecting python-dateutil>=2.1
  Using cached python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
Collecting MarkupSafe>=0.23
  Downloading MarkupSafe-1.1.1-cp38-cp38-win_amd64.whl (16 kB)
Collecting pywin32>=1.0; sys_platform == "win32"
  Downloading pywin32-300-cp38-cp38-win_amd64.whl (9.3 MB)
    |#####| 9.1 MB 1.3 MB/s eta 0:00:01
```

GAMBAR 3.8.
Instalasi Jupyter Notebook

Tunggu sampai proses instalasi Jupyter Notebook selesai. Setelah selesai, perlu dibaca, biasanya ada pesan-pesan yang harus diketahui. Contohnya saat buku ini ditulis. Akhir dari proses instalasi Jupyter Notebook terdapat pesan berupa peringatan (*Warning*). Contoh kurang lebih misalnya. PERINGATAN: Anda menggunakan pip versi 20.2.3; saat ini, versi 21.0.1 telah tersedia. Anda seharusnya

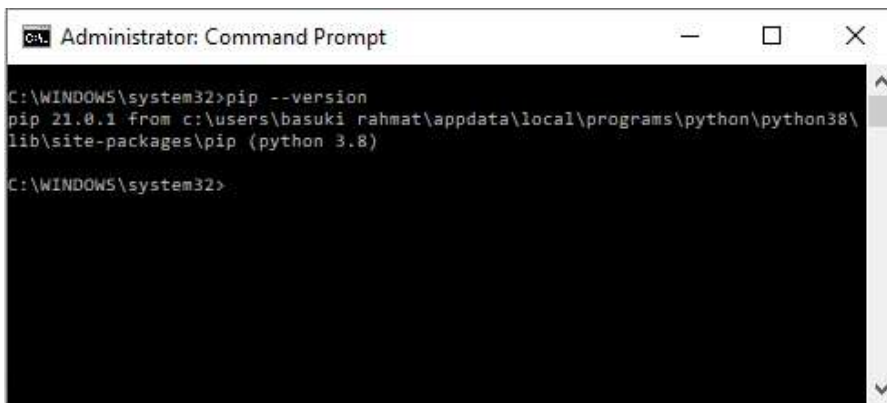
mempertimbangkan untuk memutakhirkan melalui perintah 'python.exe -m pip install --upgrade pip'.

Bisa diikuti pesan tersebut. Untuk memutakhirkan versi pip yang digunakan dapat dijalankan perintah:

```
pip install --upgrade pip
```

Jika berhasil, silahkan dicek versinya.

```
pip --version
```



```
Administrator: Command Prompt
C:\WINDOWS\system32>pip --version
pip 21.0.1 from c:\users\basuki rahmat\appdata\local\programs\python\python38\
lib\site-packages\pip (python 3.8)
C:\WINDOWS\system32>
```

GAMBAR 3.9.
Versi pip terbaru

Untuk menjalankan Jupyter Notebook, dengan menggunakan direktori kerja sesuai yang diinginkan, terlebih dahulu dapat dilakukan *generate* file konfigurasi dulu. Pengaturan ini, hanya dilakukan sekali saja, kecuali jika ingin mengganti direktori kerjanya lagi di lain waktu. Tinggal diganti nama direktori kerjanya saja.

Perintahnya:

```
jupyter notebook --generate-config
```

Selanjutnya, gunakan *File Explorer*, silahkan dicari folder dot jupyter (.jupyter) di folder users/nama_user. Kemudian dapatkan file *jupyter_notebook_config.py*, kemudian silahkan diedit menggunakan Notepad+ misalnya.

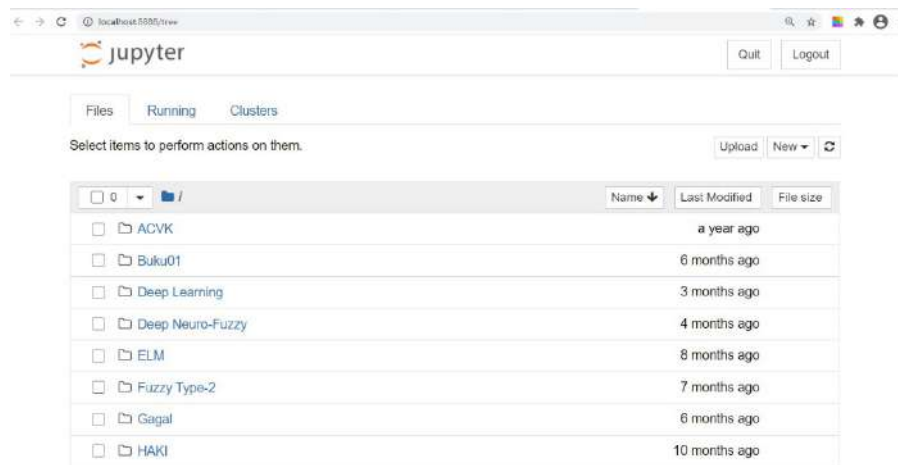
Carilah didalam file tersebut:

```
## The directory to use for notebooks and kernels.  
c.NotebookApp.notebook_dir = 'ganti_dengan_direktori_kerja_yang_diinginkan'
```

Jangan lupa hilangkan (*delete*) tanda komentar (#), kemudian silahkan disimpan. Setelah itu, dilakukan *restart* komputer terlebih dahulu.

Setelah *restart*, untuk menjalankan Jupyter Notebook, silahkan gunakan perintah berikut di *command prompt* sebagai Administrator:

jupyter notebook



GAMBAR 3.10.
Jupyter Notebook siap digunakan

3.3. TensorFlow Dasar

Tensor adalah larik data multidimensi yang digunakan oleh TensorFlow. TensorFlow dikembangkan oleh Google yaitu pustaka sumber terbuka (*library open source*) yang utamanya berfokus pada *Deep Learning* (Pattanayak, 2017). Meskipun selain untuk keperluan *Deep Learning* secara umum TensorFlow adalah *tools* untuk melakukan komputasi numerik secara luas seperti untuk operasi matriks, melakukan (*convex*) *function optimization*, menghitung gradien atau hessian (turunan kedua) dari sebuah fungsi, dsb.

Di *Deep Learning*, TensorFlow menggunakan grafik aliran data komputasi untuk mewakili arsitektur jaringan saraf yang rumit. Simpul dalam grafik menunjukkan komputasi matematis, juga disebut *ops* (operasi), sedangkan *edge* menunjukkan tensor data yang ditransfer di antara keduanya. Selain itu, gradien yang relevan disimpan di setiap simpul dari grafik komputasi, dan selama propagasi balik digabungkan untuk mendapatkan gradien yang terkait dengan bobot.

3.3.1. Instalasi TensorFlow

TensorFlow dapat diinstal dengan mudah di mesin berbasis Linux, Mac OS, dan Windows. Salah satu hal yang perlu diperhatikan untuk penginstalan TensorFlow di Windows mengharuskan versi Python yang digunakan lebih besar dari atau sama dengan 3.5. Batasan seperti itu tidak ada untuk mesin berbasis Linux, atau untuk Mac OS. Detail instalasi TensorFlow didokumentasikan dengan baik di situs resmi TensorFlow: <https://www.tensorflow.org/install>.

Instal TensorFlow 2

TensorFlow diuji dan didukung pada sistem 64-bit berikut:

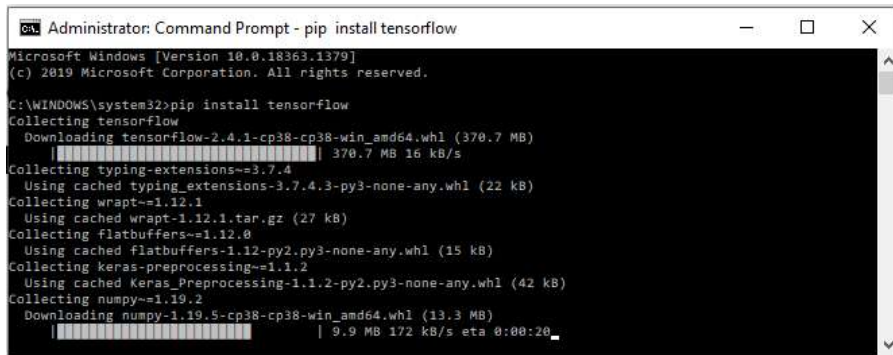
- Python 3.5–3.8
- Ubuntu 16.04 atau yang lebih baru
- Windows 7 atau lebih baru (dengan C ++ dapat didistribusikan ulang)
- macOS 10.12.6 (Sierra) atau lebih baru (tidak ada dukungan GPU)
- Raspbian 9.0 atau yang lebih baru

Instal TensorFlow dengan pengelola paket pip Python.

Paket TensorFlow 2 memerlukan versi pip > 19.0.

Untuk instal TensorFlow terbaru gunakan perintah:

```
pip install tensorflow
```



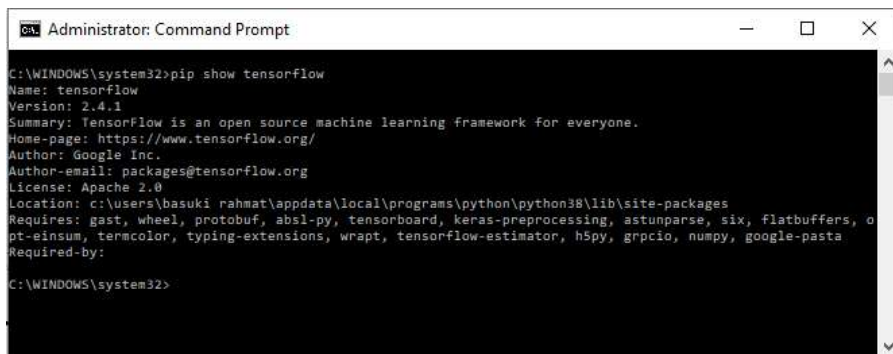
```
Administrator: Command Prompt - pip install tensorflow
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-2.4.1-cp38-cp38-win_amd64.whl (370.7 MB)
    | 370.7 MB 16 kB/s
Collecting typing_extensions==3.7.4
  Using cached typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting wrapt==1.12.1
  Using cached wrapt-1.12.1.tar.gz (27 kB)
Collecting flatbuffers==1.12.0
  Using cached flatbuffers-1.12.py2.py3-none-any.whl (15 kB)
Collecting keras_preprocessing==1.1.2
  Using cached Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
Collecting numpy==1.19.5
  Downloading numpy-1.19.5-cp38-cp38-win_amd64.whl (13.3 MB)
    | 9.9 MB 172 kB/s eta 0:00:20
```

GAMBAR 3.11.
Instalasi TensorFlow

Tunggu sampai proses instalasi TensorFlow selesai. Jika sudah berhasil diinstal, silahkan dicek menggunakan perintah:

`pip show tensorflow`



```
Administrator: Command Prompt

C:\WINDOWS\system32>pip show tensorflow
Name: tensorflow
Version: 2.4.1
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\users\basuki rahmat\appdata\local\programs\python\python38\lib\site-packages
Requires: gast, wheel, protobuf, absl-py, tensorboard, keras-preprocessing, astunparse, six, flatbuffers, opt-einsum, termcolor, typing_extensions, wrapt, tensorflow-estimator, h5py, grpcio, numpy, google-pasta
Required-by:

C:\WINDOWS\system32>
```

GAMBAR 3.12.
Tampilan versi TensorFlow

Tampak TensorFlow yang digunakan dalam buku ini, yaitu versi 2.4.1.

3.3.2. TensorFlow Dasar untuk Pengembangan

TensorFlow memiliki format perintahnya sendiri untuk menentukan dan memanipulasi tensor. Berikut ini adalah contoh-contoh beberapa perintah TensorFlow dasar.

Listing 3.1. Impor TensorFlow dan *Library* Numpy

```
import tensorflow as tf
import numpy as np
```

Listing 3.2. Aktifkan Sesi Interaktif TensorFlow

```
sess=tf.compat.v1.InteractiveSession()
```

Listing 3.3. Mendefinisikan Tensor

```
a = tf.zeros((2,2));
b = tf.ones((2,2))
```

Listing 3.4. Menghitung jumlah elemen di seluruh dimensi tensor.

```
x = tf.constant([[1, 1, 1], [1, 1, 1]])
tf.reduce_sum(x) # 6
```

Perhatikan keluaran dari *Listing 3.4*:

```
Out[19]: <tf.Tensor: shape=(), dtype=int32, numpy=6>
```

GAMBAR 3.13.
Keluaran *Listing 3.4*

Listing 3.5. Periksa Bentuk Tensor

```
a.get_shape()
```

```
Out[20]: TensorShape([2, 2])
```

GAMBAR 3.14.
Keluaran *Listing 3.5*

Listing 3.6. Bentuk ulang Tensor

```
tf.reshape(a,(1,4))
```

```
Out[22]: <tf.Tensor: shape=(1, 4), dtype=float32,
        numpy=array([[0., 0., 0., 0.]], dtype=float32)>
```

GAMBAR 3.15.
Keluaran Listing 3.6

Listing 3.7. Evaluasi Eksplisit di TensorFlow dan Perbedaan dengan Numpy

```
ta = tf.zeros((2,2))
print(ta)
```

```
tf.Tensor(
[[0. 0.]
 [0. 0.]], shape=(2, 2), dtype=float32)
```

GAMBAR 3.16.
Keluaran Listing 3.7

Bandingkan keluarannya jika digunakan Numpy:

```
a = np.zeros((2,2))
print(a)
```

```
[[0. 0.]
 [0. 0.]]
```

GAMBAR 3.17.
Keluaran Listing 3.7 lanjutan

Listing 3.8. Definisikan Konstanta TensorFlow

```
a = tf.constant(1)
b = tf.constant(5)
c= a*b
```

Listing 3.9. Sesi TensorFlow untuk Eksekusi Perintah

```
with tf.compat.v1.Session() as sess:
print(c)
```



```
tf.Tensor(5, shape=(), dtype=int32)
```

GAMBAR 3.18.
Keluaran *Listing 3.9*

Listing 3.10. Definisikan Variabel TensorFlow

```
w = tf.Variable(tf.ones(2,2),name='weights')
```

Listing 3.11. Buat Sesi untuk menjalankan grafik

```
with tf.compat.v1.Session() as sess:  
    # Buat grafik aliran data.  
    c = tf.constant([[1.0, 2.0], [3.0, 4.0]])  
    d = tf.constant([[1.0, 1.0], [0.0, 1.0]])  
    e = tf.matmul(c, d)  
    # Jalankan grafik dan simpan nilai yang diwakili oleh e dalam  
    # hasil.  
    result = sess.run(e)  
    print(result)
```

```
[[1. 3.]  
 [3. 7.]]
```

GAMBAR 3.19.
Keluaran *Listing 3.11*

Sesi TensorFlow umumnya diaktifkan melalui *tf.compat.v1.Session()* seperti yang ditunjukkan pada *Listing 3.11*, dan operasi grafik komputasi (*ops*) dijalankan di bawah sesi yang diaktifkan.

Listing 3.12. Definisikan Variabel TensorFlow dengan Nilai Awal Acak dari Distribusi Normal Standar

```
rw = tf.Variable(tf.random.normal((2,2)),name='random_weights')  
print(rw)
```

```
<tf.Variable 'random_weights:0' shape=(2, 2) dtype=float32, numpy=
array([[ 0.06068836,  1.6037406 ],
       [-0.9951369 ,  0.24026932]], dtype=float32)>
```

GAMBAR 3.20.
Keluaran *Listing 3.12*

Listing 3.13. Panggil Sesi dan Tampilkan Status Awal Variabel

```
with tf.compat.v1.Session() as sess:
    result = sess.run(tf.compat.v1.global_variables_initializer())
    print(result)
```

GAMBAR 3.21.
Keluaran *Listing 3.13*

Seperti yang ditunjukkan pada *Listing 3.11* dan *Listing 3.13*, metode *run* digunakan untuk menjalankan operasi komputasi (*ops*) dalam sesi yang diaktifkan. Dan *tf.compat.v1.global_variables_initializer()* pada *Listing 3.13* adalah untuk mengembalikan operasi komputasi (*ops*) dengan menginisialisasi variabel global.

Listing 3.14. Update Status Variabel TensorFlow

```
tf.compat.v1.disable_eager_execution()
var_1 = tf.Variable(0,name='var_1')
add_op = tf.add(var_1,tf.constant(1))
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    for i in range(5):
        print(sess.run(add_op))
```

1
1
1
1
1

GAMBAR 3.22.
Keluaran *Listing 3.14*

Pada *Listing 3.14* ditambahkan `tf.compat.v1.disable_eager_execution()` untuk mengatasi muncul pesan kesalahan *Tensor.graph is meaningless when eager execution is enabled*.

Listing 3.15. Menampilkan Status Variabel TensorFlow

```
x = tf.constant(1)
y = tf.constant(5)
z = tf.constant(7)
mul_x_y = x*y
final_op = mul_x_y + z
with tf.compat.v1.Session() as sess:
    print(sess.run([mul_x_y,final_op]))
```

[5, 12]

GAMBAR 3.23.
Keluaran *Listing 3.15*

Listing 3.16. Ubah Array Numpy menjadi Tensor

```
a = np.ones((3,3))
b = tf.convert_to_tensor(a)
with tf.compat.v1.Session() as sess:
    print(sess.run(b))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

GAMBAR 3.24.
Keluaran *Listing 3.16*

Listing 3.17. Placeholder dan Kamus Umpan

```
inp1 = tf.compat.v1.placeholder(tf.float32,shape=(1,2))
inp2 = tf.compat.v1.placeholder(tf.float32,shape=(2,1))
output = tf.matmul(inp1,inp2)
with tf.compat.v1.Session() as sess:
    print(sess.run([output],feed_dict={inp1:[[1.,3.]],
    inp2:[[1],[3]]}))
```

```
[array([[10.]], dtype=float32)]
```

GAMBAR 3.25.
Keluaran *Listing 3.17*

Placeholder TensorFlow menentukan variabel, datanya akan ditetapkan di lain waktu. Data umumnya diteruskan ke *placeholder* melalui *feed_dict* saat menjalankan operasi yang melibatkan *placeholder* TensorFlow. Hal ini telah diilustrasikan pada *Listing 3.17*.

3.4. Pengoptimal di TensorFlow

Pengoptimal (*optimizer*) di TensorFlow. TensorFlow memiliki inventaris pengoptimal yang kaya untuk mengoptimalkan fungsi biaya. Semua pengoptimal berbasis gradien, bersama dengan beberapa pengoptimal khusus untuk menangani masalah minimum lokal. Karena kita membahas pengoptimal berbasis gradien paling umum yang digunakan dalam pembelajaran mesin dan *Deep Learning*, di sini ditekankan penyesuaian yang ditambahkan di TensorFlow ke algoritma dasar.

3.4.1. GradientDescentOptimizer

GradientDescentOptimizer mengimplementasikan algoritma dasar penurunan gradien tumpukan penuh dan menggunakan kecepatan pembelajaran sebagai masukan (Pattanayak, 2017). Algoritma penurunan gradien tidak akan mengulang iterasi secara otomatis, jadi logika seperti itu harus ditentukan dalam implementasi.

Metode yang paling penting adalah metode meminimalkan di mana seseorang perlu menentukan fungsi biaya untuk meminimalkan (dilambangkan dengan kerugian) dan daftar variabel (dilambangkan dengan *var_list*) sehubungan dengan fungsi biayanya yang harus diminimalkan. Metode minimalkan secara internal memanggil metode *compute_gradients()* dan *apply_gradients()*. Mendeklarasikan daftar variabel bersifat opsional, dan jika tidak ditentukan, gradien akan dihitung berdasarkan variabel yang didefinisikan sebagai variabel TensorFlow (yaitu, yang dideklarasikan sebagai *tensorflow.Variable()*).

Penggunaan:

```
train_op =  
tf.compat.v1.train.GradientDescentOptimizer(learning_rate).  
minimize(cost)
```

Dimana *learning_rate* adalah kecepatan pembelajaran konstan dan biaya (*cost*) adalah fungsi biaya yang perlu diminimalkan melalui penurunan gradien. Fungsi biaya diminimalkan melalui variabel TensorFlow yang terkait dengan fungsi biaya.

3.4.2. AdagradOptimizer

AdagradOptimizer adalah pengoptimal urutan pertama seperti penurunan gradien tetapi dengan beberapa modifikasi (Pattanayak, 2017). Alih-alih memiliki kecepatan pembelajaran global, kecepatan pembelajaran dinormalisasi untuk setiap dimensi yang bergantung pada fungsi biaya. Kecepatan pembelajaran di setiap iterasi adalah kecepatan pembelajaran global dibagi dengan norma l^2 (l^2 norm) dari gradien sebelumnya hingga iterasi saat ini untuk setiap dimensi.

Penggunaan:

```
train_op = tf.compat.v1.train.AdagradOptimizer.(learning_
rate=0.001, initial_accumulator_value=0.1)
```

Dimana *learning_rate* mewakili η dan *initial_accumulator_value* mewakili faktor normalisasi bukan nol awal untuk setiap bobot.

3.4.3. RMSprop

RMSprop adalah versi *mini-batch* dari teknik optimasi *resilient backpropagation* (Rprop) yang bekerja paling baik untuk pembelajaran *full-batch*. Rprop memecahkan masalah gradien yang tidak mengarah ke minimum dalam kasus di mana kontur fungsi biaya berbentuk elips. Dalam kasus seperti itu, alih-alih aturan pembelajaran global, aturan pembaruan adaptif terpisah untuk setiap bobot akan mengarah pada konvergensi yang lebih baik. Hal yang istimewa dari Rprop adalah tidak menggunakan besaran gradien bobot tetapi hanya tanda-tanda dalam menentukan cara memperbarui setiap bobot.

Penggunaan:

```
train_op = tf.compat.v1.train.RMSPropOptimizer(learning_
rate=0.001, decay =0.9, momentum=0.0, epsilon=1e-10)
```

Dimana *decay* mewakili α , epsilon mewakili ϵ , dan η mewakili kecepatan pembelajaran.

3.4.4. AdadelatOptimizer

AdadelatOptimizer adalah varian *AdagradOptimizer* yang kurang agresif dalam menurunkan kecepatan pembelajaran. Untuk setiap koneksi bobot, *AdagradOptimizer* menskalakan konstanta kecepatan pembelajaran dalam sebuah iterasi dengan membaginya dengan kuadrat rata-rata akar dari semua gradien sebelumnya untuk bobot tersebut hingga iterasi itu. Jadi, kecepatan pembelajaran efektif untuk setiap bobot adalah fungsi angka iterasi yang menurun secara monoton, dan setelah sejumlah besar pengulangan. Kecepatan pembelajaran menjadi sangat kecil. *AdagradOptimizer* mengatasi masalah ini dengan mengambil *mean* atau rata-rata dari gradien kuadrat yang membusuk secara eksponensial untuk setiap bobot atau dimensi. Oleh karena itu, kecepatan pembelajaran yang efektif di *AdadelatOptimizer* tetap lebih dari perkiraan lokal dari gradien saat ini dan tidak menyusut secepat metode *AdagradOptimizer*. Ini memastikan bahwa pembelajaran berlanjut bahkan setelah sejumlah besar iterasi atau *epoch*.

Penggunaan:

```
train_op = tf.compat.v1.train.AdadelatOptimizer(learning_
rate=0.001, rho=0.95, epsilon=1e-08)
```

Dimana *rho* mewakili γ , epsilon mewakili ϵ , dan η mewakili kecepatan pembelajaran. Satu keuntungan signifikan dari *Adadelat* adalah menghilangkan konstanta kecepatan pembelajaran sama sekali. Jika dibandingkan *Adadelat* dan RMSprop, keduanya sama jika dikesampingkan eliminasi konstanta kecepatan pembelajarannya. *Adadelat* dan RMSprop keduanya dikembangkan secara independen sekitar waktu yang sama untuk menyelesaikan masalah peluruhan kecepatan pembelajaran di *Adagrad*.

3.4.5. AdamOptimizer

Adam, atau *Adaptive Moment Estimator*, adalah teknik pengoptimalan lain yang seperti RMSprop atau *Adagrad*, memiliki kecepatan pembelajaran adaptif untuk

setiap parameter atau bobot. *Adam* tidak hanya menyimpan rata-rata gradien kuadrat tetapi juga mempertahankan rata-rata gradien yang lalu.

Penggunaan:

```
train_op = tf.compat.v1.train.AdamOptimizer(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08).minimize(cost)
```

Dimana *learning_rate* adalah kecepatan pembelajaran konstan η dan biaya C adalah fungsi biaya yang perlu diminimalkan melalui *AdamOptimizer*. Parameter β_1 dan β_2 masing-masing sesuai dengan β_1 dan β_2 , sedangkan epsilon mewakili ϵ . Fungsi biaya diminimalkan sehubungan dengan variabel TensorFlow yang terkait dengan fungsi biaya.

3.4.6. MomentumOptimizer dan Nesterov Algorithm

Pengoptimal berbasis momentum telah berevolusi untuk menangani pengoptimalan non-konveks. Setiap kali bekerja dengan jaringan neural, fungsi biaya yang umumnya didapatkan bersifat non-konveks, dan dengan demikian metode pengoptimalan berbasis gradien mungkin terjebak dalam minimum lokal yang buruk. Hal ini sangat tidak diinginkan karena dalam kasus seperti itu didapatkan solusi yang kurang optimal untuk masalah pengoptimalan. Kemungkinan model yang kurang optimal. Selain itu, penurunan gradien mengikuti kemiringan di setiap titik dan membuat kemajuan kecil menuju minimum lokal, tetapi bisa sangat lambat. Metode berbasis momentum memperkenalkan komponen yang disebut kecepatan v yang meredam pembaruan parameter ketika gradien yang dihitung berubah tanda, sedangkan itu mempercepat pembaruan parameter ketika gradien berada dalam arah kecepatan yang sama. Ini menghasilkan konvergensi yang lebih cepat serta osilasi yang lebih sedikit di sekitar minimum global, atau di sekitar minimum lokal yang memberikan generalisasi yang baik.

Penggunaan:

```
train_op = tf.compat.v1.train.MomentumOptimizer.(learning_rate=0.001, momentum=0.9, use_nesterov=False)
```

Dimana *learning_rate* mewakili η , momentum mewakili α , dan *use_nesterov* menentukan apakah akan menggunakan momentum versi Nesterov.

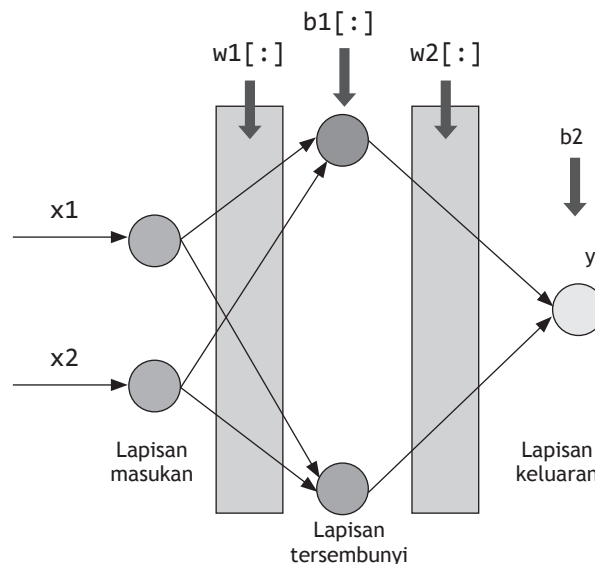
3.5. Prediksi XOR Menggunakan TensorFlow

Selanjutnya akan dicoba TensorFlow untuk menyelesaikan kasus prediksi sederhana. TensorFlow akan digunakan untuk memprediksi atau menirukan cara kerja mesin Gerbang Logika XOR. Pada saat proses pelatihan, TensorFlow akan mempelajari bagaimana pola masukan keluaran berdasarkan Data Latih Gerbang Logika XOR. Selanjutnya, pada saat pengujian, diberikan masukan sebarang, dia akan memprediksi keluarannya. Data Latih proses pelatihan Gerbang Logika XOR, seperti diperlihatkan pada Tabel 3.1.

TABEL 3.1.
Data Latih Gerbang Logika XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Selanjutnya, untuk menyelesaikan permasalahan ini, dirancang arsitektur *Deep Learning* dengan dua masukan dan satu keluaran sesuai dengan *dataset* (Data Latih dan Data Uji) yang digunakan. Rancangan arsitektur *Deep Learning* diperlihatkan pada Gambar 3.26.



GAMBAR 3.26.
Arsitektur Deep Learning dengan TensorFlow

Pemrograman Python untuk menyelesaikan permasalahan Gerbang Logika XOR menggunakan *Deep Learning* dengan TensorFlow, seperti dijabarkan pada program berikut.

#Listing 3.18. Prediksi XOR

```
#-----  
  
import tensorflow as tf  
import numpy as np  
  
#-----  
# Buat placeholder untuk label masukan dan keluaran pelatihan  
#-----  
  
x_ = tf.compat.v1.placeholder(tf.float32, shape=[4,2], name="x-  
input")  
y_ = tf.compat.v1.placeholder(tf.float32, shape=[4,1], name="y-  
output")  
  
#-----  
# Tentukan bobot masing-masing ke lapisan tersembunyi dan keluaran.  
#-----  
  
w1 = tf.Variable(tf.random.uniform([2,2], -1, 1), name="Weights1")  
w2 = tf.Variable(tf.random.uniform([2,1], -1, 1), name="Weights2")  
  
#-----  
# Tentukan bias masing-masing lapisan tersembunyi dan keluaran  
#-----  
  
b1 = tf.Variable(tf.zeros([2]), name="Bias1")  
b2 = tf.Variable(tf.zeros([1]), name="Bias2")  
  
#-----  
# Tentukan hasil akhir melalui operan maju  
#-----
```

```

z2 = tf.sigmoid(tf.matmul(x_, w1) + b1)
pred = tf.sigmoid(tf.matmul(z2,w2) + b2)

#-----
# Tentukan fungsi Biaya Cross-entropy / Log-loss berdasarkan label
# keluaran y_ dan probabilitas yang diprediksi oleh forward pass
#-----

cost = tf.reduce_mean(( (y_ * tf.math.log(pred)) +
((1 - y_) * tf.math.log(1.0 - pred)) ) * -1)
learning_rate = 0.01
train_step = tf.compat.v1.train.GradientDescentOptimizer(learning_
rate).minimize(cost)

#-----
# Sekarang setelah dimiliki semua yang dibutuhkan, dimulai pelatihan
#-----

# Pasangan Data Latih

XOR_X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

XOR_Y = np.array([
    [0],
    [1],
    [1],
    [0]
])

#-----
# Inisialisasi variabel, penyimpanan log, dan proses pelatihan,
# dan proses prediksi
#-----

```

```

init = tf.compat.v1.global_variables_initializer()

sess = tf.compat.v1.Session()

writer = tf.compat.v1.summary.FileWriter("./Downloads/XOR_logs",
sess.graph)

sess.run(init)
for i in range(100000):
    sess.run(train_step, feed_dict={x_: XOR_X, y_: XOR_Y})

Hasil_prediksi = sess.run(pred, feed_dict={x_: XOR_X, y_: XOR_Y})

#-----
print('Final Prediction', Hasil_prediksi)
#-----

```

Keluaran setelah dijalankan

```

Final Prediction [[0.02576846]
 [0.9794625 ]
 [0.9794601 ]
 [0.02292562]]

```

GAMBAR 3.27.

Keluaran hasil prediksi XOR menggunakan Deep Learning dengan TensorFlow

Selanjutnya untuk pengeplotan keluaran hasil prediksi XOR dengan TensorFlow dibandingkan dengan target berupa keluaran yang diinginkan dari gerbang Logika XOR, kita butuh *library* matplotlib. Matplotlib adalah *library* komprehensif untuk membuat visualisasi statis, animasi, dan interaktif dengan Python.

Cara instalasi matplotlib, di *command prompt*:

```

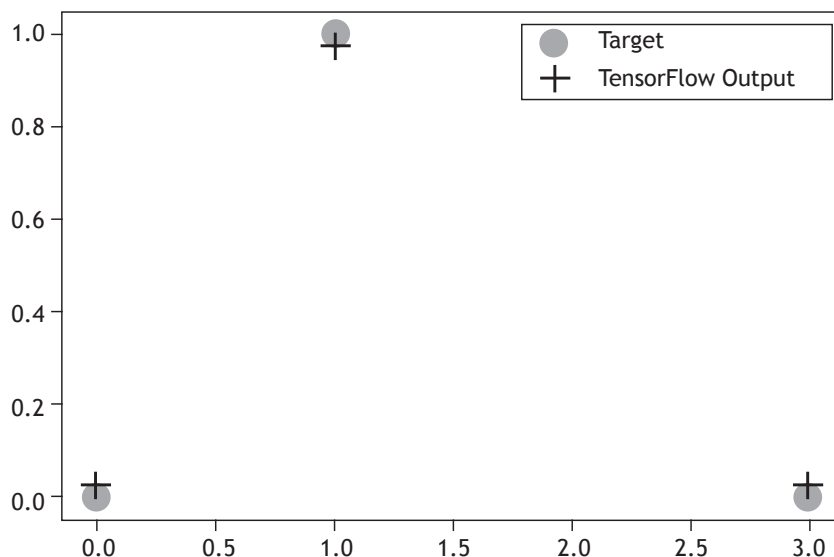
pip install matplotlib

```

Setelah matplotlib berhasil diinstal, selanjutnya silahkan ketikkan program lanjutannya berikut ini.

```
import matplotlib.pyplot as plt
plt.plot(XOR_Y, 'bo', label='Target', linewidth=2, markersize=12)
plt.plot(Hasil_prediksi, 'r+', label='TensorFlow Output',
linewidth=2, markersize=12)
plt.legend(loc='upper right')
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 3.28.



GAMBAR 3.28.
Perbandingan prediksi TensorFlow dan target

Selanjutnya untuk menghitung kesalahan prediksi, digunakan *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE). Berikut ini skrip pythonnya (memanfaatkan *library* scikit-learn).

Cara instalasi scikit-learn, di *command prompt*:

```
pip install scikit-learn
```

Setelah scikit-learn berhasil diinstal, selanjutnya silahkan ketikkan program lanjutannya berikut ini.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse1 = mean_squared_error(XOR_Y, Hasil_prediksi)
rmse1 = sqrt(mean_squared_error(XOR_Y, Hasil_prediksi))
print('MSE =',mse1)
print('RMSE =',rmse1)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 3.29.

```
MSE = 0.0005083181741847298
RMSE = 0.022545912582655193
```

GAMBAR 3.29.

Kesalahan hasil prediksi TensorFlow

3.6. Grafik Komputasi TensorFlow dengan Tensorboard

Dalam pembelajaran mesin, untuk meningkatkan sesuatu, kita sering kali harus dapat mengukurnya. TensorBoard adalah fitur yang menyediakan pengukuran dan visualisasi yang diperlukan selama alur kerja pembelajaran mesin. Ini memungkinkan pelacakan metrik eksperimen seperti kerugian dan akurasi, memvisualisasikan grafik model, memproyeksikan *embedding* ke ruang dimensi yang lebih rendah, dan banyak lagi.

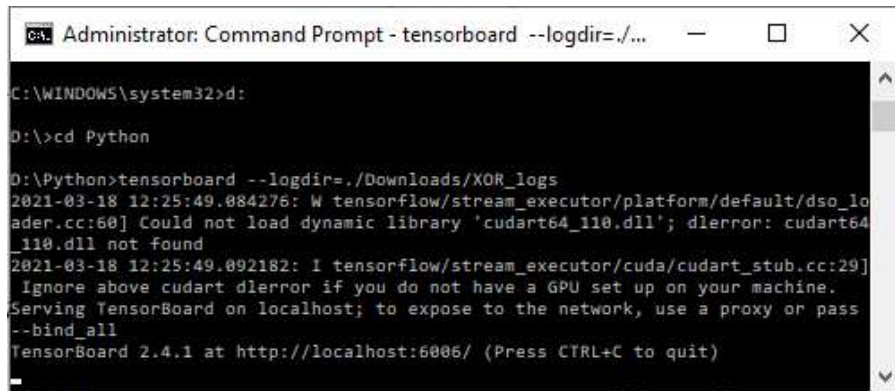
Untuk memulai TensorBoard, perhatikan *Listing 3.18* Prediksi XOR sebelumnya. Perhatikan pada skrip program, terdapat bagian program yang menyatakan:

```
writer = tf.compat.v1.summary.FileWriter("./Downloads/XOR_logs",  
sess.graph)
```

Ini menunjukkan lokasi di mana file *log* ringkasan tersebut telah disimpan. Lokasi ini, bebas ditentukan dimana pun yang kita pilih.

Setelah program sesuai *Listing 3.18* dijalankan, dan sudah keluar hasilnya seperti yang terlihat pada Gambar 3.27. Selanjutnya kita gunakan *File Explorer*, kita cari folder *Downloads/XOR_logs* hasil *generate* program *Listing 3.18*. Supaya lebih mudah, kita pindahkan ke folder yang mudah kita ingat. Misalnya ke direktori *D:\Python*. Sehingga untuk menjalankan Tensorboard, silahkan ke terminal, dan melalui direktori *D:\Python* silahkan jalankan perintah berikut:

```
D:\Python>tensorboard --logdir=./Downloads/XOR_logs
```

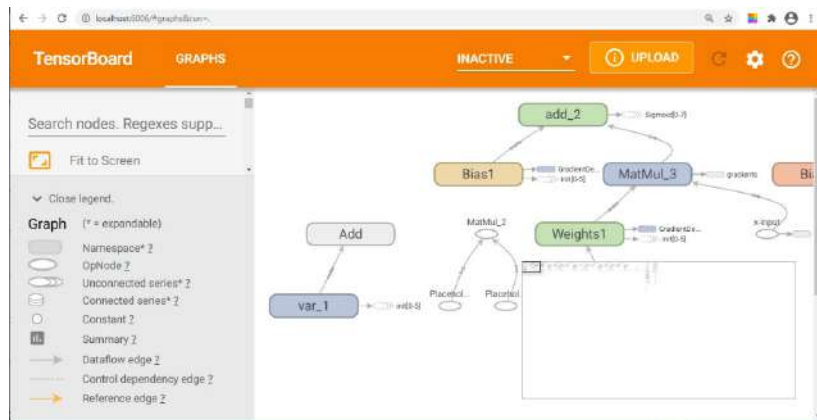


```
C:\WINDOWS\system32>d:  
  
D:\>cd Python  
  
D:\Python>tensorboard --logdir=./Downloads/XOR_logs  
2021-03-18 12:25:49.084276: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found  
2021-03-18 12:25:49.092182: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.  
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all  
TensorBoard 2.4.1 at http://localhost:6006/ (Press CTRL+C to quit)
```

GAMBAR 3.30.
Tensorboard setelah dijalankan

Sesuai informasi yang ada di Gambar 3.30, untuk melihat visualisasi Tensorboard program Pediksi XOR, dapat dilakukan dengan menjalankan:

<http://localhost:6006/>. Setelah dijalankan di browser, maka hasil visualisasinya seperti terlihat pada Gambar 3.31.



GAMBAR 3.31.
Visualisasi Tensorboard program Pediksi XOR



Bab 4

Aplikasi Tensorflow

4.1. Regresi Linear dengan TensorFlow

Dalam Bab 4 ini, kita akan eksplorasi lebih jauh kemampuan *Deep Learning* dengan TensorFlow untuk menyelesaikan berbagai macam permasalahan. Kali ini kita akan mencoba menyelesaikan permasalahan Regresi Linear. Contoh ini kami ambil dari: <https://github.com/aymericdamien/TensorFlow-Examples>.

Regresi Linear adalah sebuah pendekatan untuk memodelkan hubungan antara variabel terikat Y dan satu atau lebih variabel bebas yang disebut x. Salah satu kegunaan dari Regresi Linear adalah untuk melakukan prediksi berdasarkan data-data yang telah dimiliki sebelumnya. Hubungan di antara variable-variabel tersebut disebut sebagai model Regresi Linear. Persamaan Regresi Linear sederhana, dapat dinyatakan sebagai berikut (Bruce and Bruce, 2017):

$$Y = Wx + b \quad (4.1)$$

Dimana W sebagai koefisien kemiringan untuk x , dan b sebagai konstanta. Variabel Y dikenal sebagai respon atau variabel dependen (terikat) karena bergantung pada x . Variabel x dikenal sebagai prediktor atau variabel independen (bebas). Komunitas pembelajaran mesin cenderung menggunakan istilah lain, menyebut Y sebagai target dan x sebagai vektor fitur (Bruce and Bruce, 2017). Demikian juga, untuk W koefisien kemiringan untuk x , dan b sebagai konstanta, cenderung lebih suka digunakan istilah Bobot dan Bias.

Berikut ini, contoh pemrograman penyelesaian Regresi Linear menggunakan TensorFlow, untuk mendapatkan W sebagai koefisien kemiringan atau Bobot, dan b sebagai konstanta atau Bias, jika pasangan *dataset* x dan Y diberikan. Sehingga diperoleh hasil prediksi yang paling mendekati nilai Y yang diharapkan. Dalam hal ini ditunjukkan dengan kesalahan prediksi dalam *Mean Square Error* (MSE), *Root Mean Square Error* (RMSE), dan *Mean Absolute Percentage Error* (MAPE), serta Persentase_keberhasilan prediksi.

Pertama, import *library* yang dibutuhkan.

```
# Import Library yang dibutuhkan:

from __future__ import absolute_import, division, print_function
import tensorflow as tf
import numpy as np
```

Kemudian, tentukan parameter-parameter yang digunakan.

```
# Parameter-parameter yang digunakan:

kecepatan_pembelajaran = 0.01
langkah_pelatihan = 5000
tampilkan_langkah = 50
```

Tentukan Data Latih yang digunakan.

```
# Data Latih yang digunakan.

X = np.array([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
              7.042,10.791,5.313,7.997,5.654,9.27,3.1])
Y = np.array([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
              2.827,3.465,1.65,2.904,2.42,2.94,1.3])
```

Tentukan Bobot dan Bias, diinisialisasi secara acak, persamaan Regresi Linear, *Mean Square Error* (MSE), *Root Mean Square Error* (RMSE), *Mean Absolute Percentage Error* (MAPE), dan Pengoptimal penurunan Gradien *Stochastic*.

```
# Bobot dan Bias, diinisialisasi secara acak.
rng = np.random
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Regresi linier (Wx + b).
def Regresi_Linier(x):
    return W * x + b

# Mean square error (MSE).
def mean_square(y_pred, y_true):
    return tf.reduce_mean(tf.square(y_pred - y_true))

# Mean Absolute Percentage Error (MAPE).
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Pengoptimal Penurunan Gradien Stochastic.
optimizer = tf.optimizers.SGD(kecepatan_pembelajaran)
```

Kemudian, buat fungsi untuk menjalankan proses pengoptimalan.

```
# Fungsi untuk Proses pengoptimalan.

from math import sqrt
def run_optimization():
    # Bungkus komputasi di dalam GradientTape untuk
    # diferensiasi otomatis.
    with tf.GradientTape() as g:
        pred = Regresi_Linier(X)
        mse = mean_square(pred, Y)
        rmse = sqrt(mse)
        MAPE = mean_absolute_percentage_error(Y, pred)
        Persentase_keberhasilan = 100-MAPE

    # Hitung gradien.
    gradients = g.gradient(mse, [W, b])

    # Perbarui W dan b menggunakan gradien berikut.
    optimizer.apply_gradients(zip(gradients, [W, b]))
```

Selanjutnya, jalankan pelatihan untuk sejumlah langkah tertentu.

```
# Jalankan pelatihan untuk sejumlah langkah tertentu:
for langkah in range(1, langkah_pelatihan + 1):
    # Jalankan pengoptimalan untuk memperbarui nilai W dan b.
    run_optimization()

    if langkah % tampilkan_langkah == 0:
        pred = Regresi_Linier(X)
        mse = mean_square(pred, Y)
        rmse = sqrt(mse)
        MAPE = mean_absolute_percentage_error(Y, pred)
        Persentase_keberhasilan = 100-MAPE
```

```
print("langkah: %i, mse: %f, rmse: %f, MAPE: %f, Persentase_
keberhasilan: %f, W: %f, b: %f" % (langkah, mse, rmse,
MAPE, Persentase_keberhasilan, W.numpy(), b.numpy()))
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 4.1.

```
langkah: 4650, mse: 0.153858, rmse: 0.392247, MAPE: 14.001456, Persentase_kebe
rhasilan: 85.998544, W: 0.251640, b: 0.798766
langkah: 4700, mse: 0.153858, rmse: 0.392247, MAPE: 14.001471, Persentase_kebe
rhasilan: 85.998529, W: 0.251639, b: 0.798769
langkah: 4750, mse: 0.153858, rmse: 0.392247, MAPE: 14.001485, Persentase_kebe
rhasilan: 85.998515, W: 0.251639, b: 0.798772
langkah: 4800, mse: 0.153858, rmse: 0.392247, MAPE: 14.001498, Persentase_kebe
rhasilan: 85.998502, W: 0.251639, b: 0.798775
langkah: 4850, mse: 0.153858, rmse: 0.392247, MAPE: 14.001515, Persentase_kebe
rhasilan: 85.998485, W: 0.251638, b: 0.798778
langkah: 4900, mse: 0.153858, rmse: 0.392247, MAPE: 14.001529, Persentase_kebe
rhasilan: 85.998471, W: 0.251638, b: 0.798781
langkah: 4950, mse: 0.153858, rmse: 0.392247, MAPE: 14.001543, Persentase_kebe
rhasilan: 85.998457, W: 0.251637, b: 0.798784
langkah: 5000, mse: 0.153858, rmse: 0.392247, MAPE: 14.001557, Persentase_kebe
rhasilan: 85.998443, W: 0.251637, b: 0.798787
```

GAMBAR 4.1.
Proses pelatihan TensorFlow

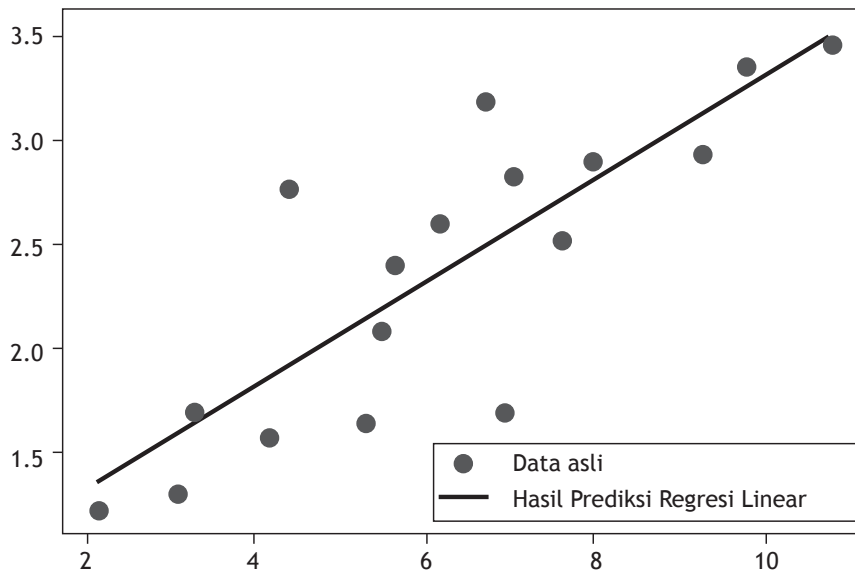
Dari Gambar 4.1, dapat dilihat setelah iterasi atau langkah ke-5000, diperoleh kesalahan prediksi sebesar MSE = 0.153858, RMSE = 0.392247, MAPE = 14.001557, dan Persentase_keberhasilan: 85.998443%.

Selanjutnya untuk pengeplotan hasil prediksi terhadap nilai Y yang diharapkan, sesuai Data Latih yang digunakan, programnya sebagai berikut.

```
# Tampilan grafis

import matplotlib.pyplot as plt
plt.plot(X, Y, 'ro', label='Data asli')
plt.plot(X, np.array(W * X + b), label='Hasil Prediksi Regresi
Linear')
plt.legend()
plt.show()
```

Jika dijalankan, diperoleh hasil seperti terlihat pada Gambar 4.2.



GAMBAR 4.2.
Hasil prediksi Regresi Linear

4.2. Regresi Logistik dengan TensorFlow

Regresi Linear sesuai Persamaan (4.1), telah disebutkan, bahwa regresi ini bisa digunakan untuk menyelesaikan permasalahan prediksi berdasarkan data-data yang telah dimiliki sebelumnya. Meskipun bentuknya hanya berupa garis lurus. Berbekal persamaan Regresi Linear ini, selanjutnya bisa dikembangkan menjadi Regresi Logistik, sehingga selain memiliki kemampuan prediksi dia juga bisa memiliki kemampuan klasifikasi.

Regresi Logistik (kadang disebut model logistik atau model logit), dalam statistika digunakan untuk prediksi probabilitas kejadian suatu peristiwa dengan mencocokkan data pada fungsi logit kurva logistik. Metode ini merupakan model linier umum yang digunakan untuk regresi binomial. Seperti analisis regresi pada umumnya, metode ini menggunakan beberapa variabel prediktor, baik numerik maupun kategori.

Regresi Logistik bisa digunakan untuk memprediksi hasil biner berdasarkan satu set variabel independen. Regresi Logistik juga biasa digunakan ketika variabel dependen (target) bersifat kategorikal. Sebagai contoh, bagaimana kita bisa mengetahui atau memprediksi apakah email terkena spam (1) atau bukan (0). Untuk kasus lain, misalnya untuk memprediksi apakah tanaman akan hidup (1) atau mati (0). Dan lain-lain. Selain untuk klasifikasi biner, bisa juga, nilai diantara 0 dan 1. Yang nanti akan kita gunakan untuk contoh prediksi atau pengenalan angka tulisan tangan.

Mirip persamaan Regresi Linear pada Persamaan (4.1), persamaan Regresi Logistik sederhana, dapat dinyatakan sebagai berikut:

$$\text{Ln}\left(\frac{p}{1-p}\right) = Wx + b \quad (4.2)$$

atau

$$e^{(Wx+b)} = \frac{p}{1-p} \quad (4.3)$$

Penyelesaian Persamaan (4.3) untuk mendapatkan nilai p, diperoleh:

$$p = \frac{e^{(Wx+b)}}{1 + e^{(Wx+b)}} \quad (4.4)$$

Dimana p sebagai peluang yang nilainya berkisar antara 0 – 1. W sebagai koefisien kemiringan untuk x atau Bobot, dan b sebagai konstanta atau Bias.

Selanjutnya, berikut ini, kita akan gunakan Regresi Logistik sederhana dengan TensorFlow, untuk mengenali angka tulisan tangan yang sangat populer, milik *Modified National Institute of Standards and Technology* (MNIST). *Dataset*-nya berisi 60.000 contoh angka tulisan tangan untuk pelatihan, dan 10.000 contoh angka tulisan tangan untuk pengujian. Gambar angka tulisan tangan telah dinormalisasi ukurannya dan dipusatkan pada gambar berukuran tetap (28x28 piksel) dengan

nilai dari 0 hingga 255. Dalam contoh ini, setiap gambar akan diubah menjadi *float32*, dinormalisasi menjadi $[0, 1]$ dan diratakan menjadi larik 1-D dari 784 fitur ($28 * 28$).



GAMBAR 4.3.
Angka tulisan tangan MNIST (<http://yann.lecun.com/exdb/mnist>)

Berikut ini, contoh pemrogramannya.
Pertama, import *library* yang dibutuhkan.

```
# Import Library yang dibutuhkan:  
  
from __future__ import absolute_import, division, print_function  
import tensorflow as tf  
import numpy as np
```

Kemudian, tentukan parameter-parameter yang digunakan.

```
# Parameter-parameter dataset MNIST.  
jumlah_kelas = 10 # tulisan angka 0 to 9  
jumlah_fitur = 784 # 28*28
```

```
# Parameter pelatihan.  
kecepatan_pembelajaran = 0.01  
langkah_pelatihan = 1000  
ukuran_batch = 256  
tampilkan_langkah = 50
```

Selanjutnya, siapkan data angka tulisan tangan MNIST.

```
# Siapkan data MNIST.  
from tensorflow.keras.datasets import mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
# Ubah menjadi float32.  
x_train, x_test = np.array(x_train, np.float32), np.array(x_test,  
np.float32)  
  
# Ratakan gambar ke vektor 1-D dari 784 fitur (28 * 28).  
x_train, x_test = x_train.reshape([-1, jumlah_fitur]), x_test.  
reshape([-1, jumlah_fitur])  
  
# Normalisasi nilai gambar dari [0, 255] hingga [0, 1].  
x_train, x_test = x_train / 255., x_test / 255.
```

Kemudian, gunakan API `tf.data` untuk mengacak dan mengumpulkan data.


```
# Gunakan API tf.data untuk mengacak dan mengumpulkan data.

train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_data = train_data.repeat().shuffle(5000).batch(ukuran_batch).
prefetch(1)
```

Tentukan Bobot W , Bias b , fungsi Regresi Logistik, Fungsi kerugian *Cross-Entropy*, Metrik akurasi, dan Pengoptimal penurunan gradien stokastik.

```
# Bobot bentuk [784, 10], fitur gambar 28 * 28, dan jumlah total
kelas.
W = tf.Variable(tf.ones([jumlah_fitur, jumlah_kelas]),
name="weight")

# Bias bentuk [10], jumlah kelas.
b = tf.Variable(tf.zeros([jumlah_kelas]), name="bias")

# Regresi Logistik ( $Wx + b$ ).
def Regresi_Logistik(x):
    # Terapkan softmax untuk menormalkan logit ke
    # distribusi probabilitas.
    return tf.nn.softmax(tf.matmul(x, W) + b)

# Fungsi kerugian Cross-Entropy.
def cross_entropy(y_pred, y_true):
    # Menyandakan label ke satu vektor panas.
    y_true = tf.one_hot(y_true, depth=jumlah_kelas)

    # Nilai prediksi klip untuk menghindari kesalahan log (0).
    y_pred = tf.clip_by_value(y_pred, 1e-9, 1.)
```

```

    # Hitung cross-entropy.
    return tf.reduce_mean(-tf.reduce_sum(y_true * tf.math.log(y_
pred),1))

# Metrik akurasi.
def accuracy(y_pred, y_true):
    # Kelas prediksi adalah indeks nilai tertinggi dalam vektor prediksi
    # (yaitu argmax).
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true,
tf.int64))
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# Pengoptimal penurunan gradien stokastik.
optimizer = tf.optimizers.SGD(kecepatan_pembelajaran)

```

Selanjutnya, buat fungsi untuk menjalankan proses pengoptimalan.

```

# Fungsi untuk menjalankan proses pengoptimalan.
def run_optimization(x, y):
    # Bungkus komputasi di dalam GradientTape untuk diferensiasi
    # otomatis.
    with tf.GradientTape() as g:
        pred = Regresi_Logistik(x)
        loss = cross_entropy(pred, y)

    # Hitung gradien.
    gradients = g.gradient(loss, [W, b])

    # Perbarui W dan b menggunakan gradien berikut.
    optimizer.apply_gradients(zip(gradients, [W, b]))

```

Selanjutnya, silahkan jalankan pelatihan untuk sejumlah langkah tertentu.

```
# Jalankan pelatihan untuk sejumlah langkah tertentu.
for step, (batch_x, batch_y) in enumerate(train_data.take(langkah_
pelatihan), 1):
    # Jalankan pengoptimalan untuk memperbarui nilai W dan b.
    run_optimization(batch_x, batch_y)

    if step % tampilkan_langkah == 0:
        pred = Regresi_Logistik(batch_x)
        loss = cross_entropy(pred, batch_y)
        acc = accuracy(pred, batch_y)
        print("step: %i, loss: %f, accuracy: %f" % (step, loss, acc))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.4.

```
step: 300, loss: 0.365288, accuracy: 0.906250
step: 350, loss: 0.385452, accuracy: 0.886719
step: 400, loss: 0.434739, accuracy: 0.886719
step: 450, loss: 0.286779, accuracy: 0.917969
step: 500, loss: 0.388428, accuracy: 0.902344
step: 550, loss: 0.370555, accuracy: 0.894531
step: 600, loss: 0.366227, accuracy: 0.910156
step: 650, loss: 0.367984, accuracy: 0.902344
step: 700, loss: 0.353479, accuracy: 0.898438
step: 750, loss: 0.323127, accuracy: 0.925781
step: 800, loss: 0.332522, accuracy: 0.914062
step: 850, loss: 0.346961, accuracy: 0.917969
step: 900, loss: 0.226002, accuracy: 0.945312
step: 950, loss: 0.364847, accuracy: 0.886719
step: 1000, loss: 0.364202, accuracy: 0.906250
```

GAMBAR 4.4.
Proses pelatihan TensorFlow

Berikutnya, silahkan dilakukan uji model pada sekumpulan data validasi.

```
# Uji model pada set validasi.  
pred = Regresi_Logistik(x_test)  
print("Akurasi Tes: %f" % accuracy(pred, y_test))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.5.

Akurasi Tes: 0.906300

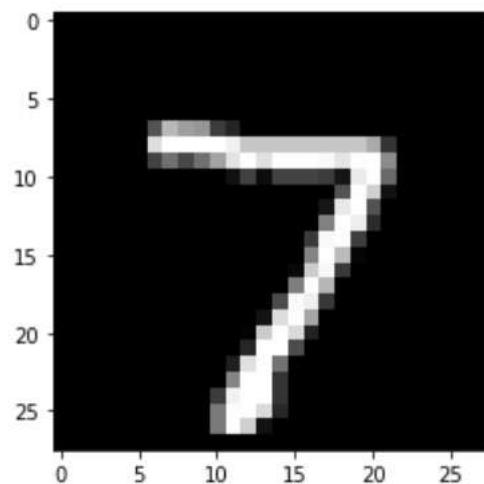
GAMBAR 4.5.

Hasil uji model pada sekumpulan data validasi

Tampak dari Gambar 4.5, hasil akurasi pengujian lebih dari 90 persen. Sudah bagus. Langkah terakhir, untuk menampilkan visualisasi hasil prediksi pengenalan angka tulisan tangan MNIST, bisa diambil 5 contoh gambar angka yang ingin dikenali.

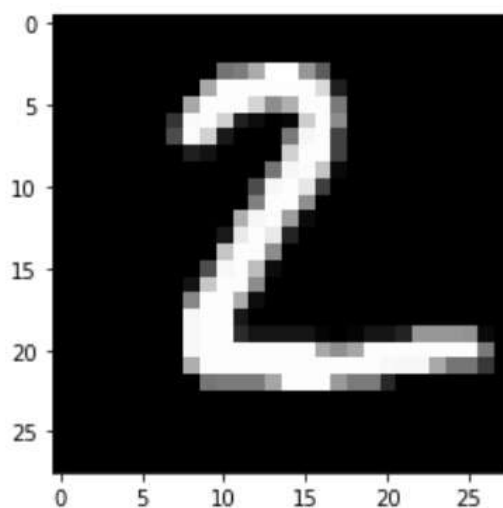
```
# Visualisasikan prediksi.  
import matplotlib.pyplot as plt  
  
# Memprediksi 5 gambar dari set validasi.  
n_images = 5  
test_images = x_test[:n_images]  
predictions = Regresi_Logistik(test_images)  
  
# Menampilkan gambar dan prediksi model.  
for i in range(n_images):  
    plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray')  
    plt.show()  
    print("Prediksi model: %i" % np.argmax(predictions.numpy()[i]))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.6 sampai dengan Gambar 4.10.



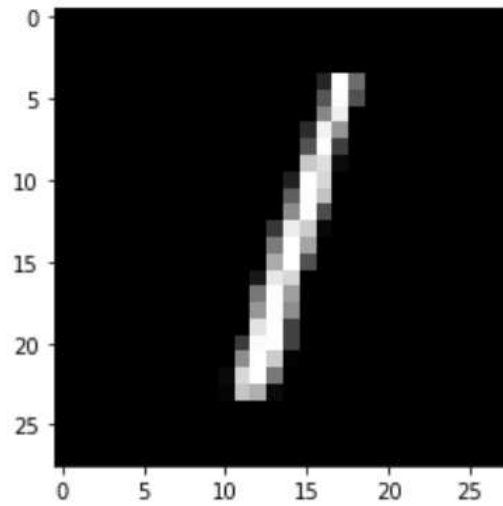
Prediksi model: 7

GAMBAR 4.6.
Hasil pengenalan angka 7 tulisan tangan MNIST



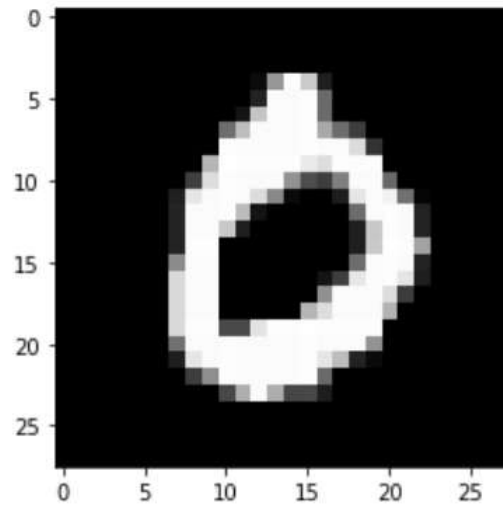
Prediksi model: 2

GAMBAR 4.7.
Hasil pengenalan angka 2 tulisan tangan MNIST



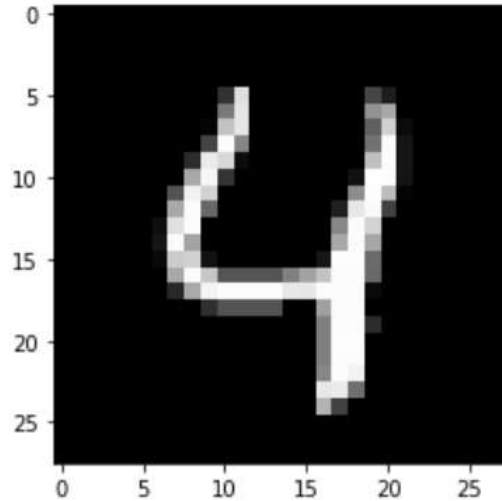
Prediksi model: 1

GAMBAR 4.8.
Hasil pengenalan angka 1 tulisan tangan MNIST



Prediksi model: 0

GAMBAR 4.9.
Hasil pengenalan angka 0 tulisan tangan MNIST



Prediksi model: 4

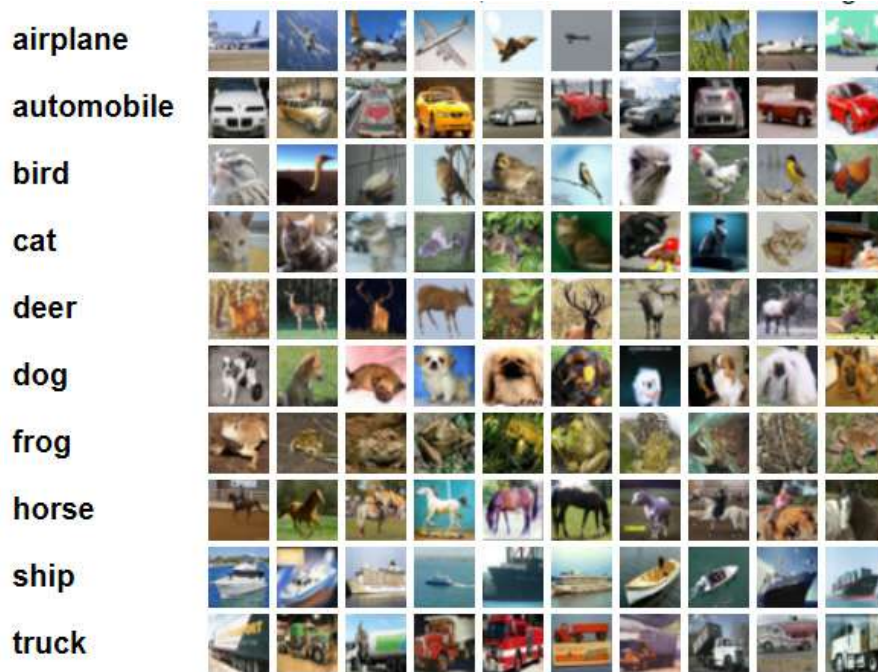
GAMBAR 4.10.
Hasil pengenalan angka 4 tulisan tangan MNIST

4.3. *Convolutional Neural Network* dengan TensorFlow

Contoh aplikasi TensorFlow selanjutnya, akan dijelaskan bagaimana melakukan pemrograman *Deep Learning* dengan salah satu arsitektur utama *Deep Network* yang terkenal dan banyak digunakan di berbagai bidang penerapan, yaitu *Convolutional Neural Networks* (CNNs). Mengenai penjelasan tentang konsep atau teori CNN ini telah dibahas di Bab 1. Di Sub Bab ini, akan didemonstrasikan bagaimana pelatihan CNN sederhana untuk mengklasifikasikan gambar CIFAR. CIFAR-10 dan CIFAR-100 diberi label *subset* dari 80 juta kumpulan gambar kecil. Mereka dikumpulkan oleh Alex Krizhevsky, Vinod Nair, dan Geoffrey Hinton yang beralamat di <https://www.cs.toronto.edu/~kriz/cifar.html>.

Dataset CIFAR-10 terdiri dari 60.000 gambar berwarna 32x32 dalam 10 kelas, dengan 6000 gambar per kelas. Ada 50.000 gambar pelatihan dan 10.000 gambar uji. *Dataset* dibagi menjadi lima kelompok pelatihan dan satu kelompok pengujian, masing-masing dengan 10.000 gambar. Kelompok pengujian berisi tepat 1000 gambar yang dipilih secara acak dari setiap kelas. Kelompok pelatihan berisi gambar yang tersisa dalam urutan acak, tetapi beberapa kelompok pelatihan mungkin berisi lebih banyak gambar dari satu kelas daripada yang lain. Di antara mereka, *batch*

pelatihan berisi tepat 5.000 gambar dari setiap kelas. Berikut ini langkah-langkah pemrogramannya.



GAMBAR 4.11.
Dataset CIFAR-10

Pertama, import *library* yang dibutuhkan.

```
# Impor TensorFlow
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Kemudian, silahkan unduh dan siapkan kumpulan data CIFAR10. *Dataset* CIFAR10 berisi 60.000 gambar berwarna dalam 10 kelas, dengan 6.000 gambar di setiap kelas. *Dataset* dibagi menjadi 50.000 gambar pelatihan dan 10.000 gambar

pengujian. Kelas-kelas tersebut saling eksklusif dan tidak ada tumpang tindih di antara mereka.

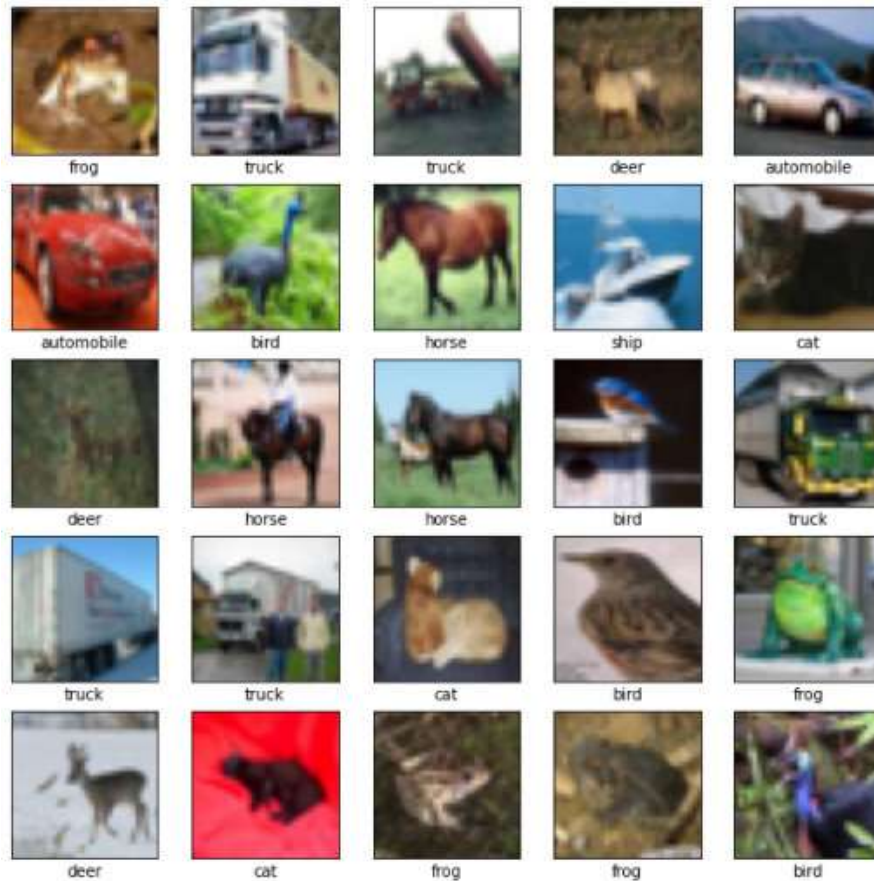
```
(train_images, train_labels), (test_images, test_labels) = datasets.  
cifar10.load_data()  
  
# Normalisasi nilai piksel menjadi antara 0 dan 1  
train_images, test_images = train_images / 255.0, test_images /  
255.0
```

Selanjutnya, silahkan dilakukan verifikasi datanya. Untuk memverifikasi bahwa *dataset* terlihat benar, mari kita plot 25 gambar pertama dari *set* pelatihan dan menampilkan nama kelas di bawah setiap gambar.

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',  
               'dog', 'frog', 'horse', 'ship', 'truck']
```

```
plt.figure(figsize=(10,10))  
for i in range(25):  
    plt.subplot(5,5,i+1)  
    plt.xticks([])  
    plt.yticks([])  
    plt.grid(False)  
    plt.imshow(train_images[i], cmap=plt.cm.binary)  
    # The CIFAR labels happen to be arrays,  
    # which is why you need the extra index  
    plt.xlabel(class_names[train_labels[i][0]])  
plt.show()
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.12.



GAMBAR 4.12.
Plot 25 gambar pertama dari set pelatihan

Selanjutnya, buat basis konvolusional. Enam baris kode di bawah ini mendefinisikan dasar konvolutional menggunakan pola umum: setumpuk lapisan *Conv2D* dan *MaxPooling2D*. Sebagai masukan, CNN mengambil bentuk tensor (tinggi_gambar, lebar_gambar, saluran_warna), mengabaikan ukuran tumpukan. Pada dimensi ini, *color_channels* merujuk ke (R, G, B). Dalam contoh ini, akan dikonfigurasi CNN untuk memproses masukan dalam bentuk (32, 32, 3), yang merupakan format gambar CIFAR. Hal ini dapat dilakukan dengan meneruskan argumen *input_shape* ke lapisan pertama.

```

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

```

Selanjutnya, marilah kita tampilkan arsitektur model kita sejauh ini.

```

model.summary()

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.13.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
Total params: 56,320		
Trainable params: 56,320		
Non-trainable params: 0		

GAMBAR 4.13.
Arsitektur model

Dari Gambar 4.13, dapat dilihat bahwa keluaran dari setiap lapisan *Conv2D* dan *MaxPooling2D* adalah bentuk tensor 3D (tinggi, lebar, saluran). Dimensi lebar dan tinggi cenderung menyusut saat masuk lebih dalam di jaringan. Jumlah saluran keluaran untuk setiap lapisan *Conv2D* dikontrol oleh argumen pertama (misalnya, 32 atau 64). Biasanya, saat lebar dan tinggi menyusut, dapat (secara komputasi) untuk ditambahkan lebih banyak saluran keluaran di setiap lapisan *Conv2D*.

Kemudian, tambahkan lapisan padat di atas untuk melengkapi model kita. Masukkan tensor keluaran terakhir dari basis konvolusional (berbentuk (4, 4, 64)) ke dalam satu atau lebih lapisan padat, untuk melakukan klasifikasi. Lapisan padat mengambil vektor sebagai masukan (yaitu 1D), sedangkan keluaran saat ini adalah tensor 3D. Pertama, akan diratakan (atau dibuka gulungan) keluaran 3D ke 1D, lalu ditambahkan satu atau lebih lapisan padat di atasnya. CIFAR memiliki 10 kelas keluaran, jadi digunakan lapisan *Dense* akhir dengan 10 keluaran.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Sehingga, arsitektur lengkap model kita saat ini, menjadi berikut ini.

```
model.summary()
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.14.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 10)	650
flatten_1 (Flatten)	(None, 10)	0
dense_2 (Dense)	(None, 64)	704
dense_3 (Dense)	(None, 10)	650
Total params: 123,924		
Trainable params: 123,924		
Non-trainable params: 0		

GAMBAR 4.14.
Arsitektur model lengkap

Seperti yang kita lihat pada Gambar 4.14, keluaran (4, 4, 64) diratakan menjadi vektor bentuk (1024) sebelum melewati dua lapisan *Dense*. Selanjutnya dilakukan pelatihan modelnya, dan dilihat hasil akurasi validasi menggunakan Data Uji.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy
                (from_logits=True), metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.15.

```
Epoch 5/10
1563/1563 [=====] - 88s 56ms/step - loss: 0.8515 - acc
uracy: 0.6974 - val_loss: 0.9011 - val_accuracy: 0.6890
Epoch 6/10
1563/1563 [=====] - 86s 55ms/step - loss: 0.7918 - acc
uracy: 0.7211 - val_loss: 0.9358 - val_accuracy: 0.6742
Epoch 7/10
1563/1563 [=====] - 87s 56ms/step - loss: 0.7339 - acc
uracy: 0.7397 - val_loss: 0.9502 - val_accuracy: 0.6742
Epoch 8/10
1563/1563 [=====] - 87s 56ms/step - loss: 0.6775 - acc
uracy: 0.7586 - val_loss: 0.8768 - val_accuracy: 0.7013
Epoch 9/10
1563/1563 [=====] - 88s 56ms/step - loss: 0.6372 - acc
uracy: 0.7744 - val_loss: 0.9008 - val_accuracy: 0.7035
Epoch 10/10
1563/1563 [=====] - 88s 56ms/step - loss: 0.5935 - acc
uracy: 0.7905 - val_loss: 0.8728 - val_accuracy: 0.7056
```

GAMBAR 4.15.

Proses pelatihan CNN dan hasil akurasi validasi menggunakan Data Uji

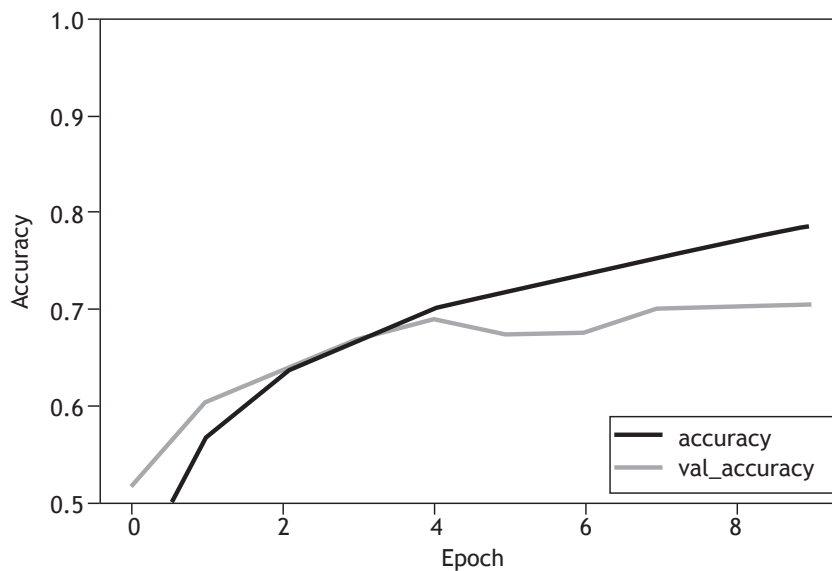
Hasil proses pelatihan CNN dan hasil akurasi validasi menggunakan Data Uji juga dapat ditampilkan dalam bentuk grafik. Selanjutnya dapat dilakukan evaluasi modelnya.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images, test_labels,
verbose=2)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.16.

313/313 - 5s - loss: 0.8728 - accuracy: 0.7056



GAMBAR 4.16.

Grafik hasil proses pelatihan CNN dan hasil akurasi validasi menggunakan Data Uji

Langkah terakhir, silahkan dicetak hasil akurasi pengujiannya.

```
print(test_acc)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.17.

0.7056000232696533

GAMBAR 4.17.

Hasil akurasi pengujian

Dari Gambar 4.17 terlihat CNN telah mencapai akurasi pengujian lebih dari 70%. Lumayan untuk pengenalan pertama dengan CNN menggunakan TensorFlow.

4.4. *Convolutional Neural Network* dengan TensorFlow Lanjutan

Contoh aplikasi TensorFlow selanjutnya, masih menerapkan *Convolutional Neural Networks* (CNNs) yang akan kita gunakan untuk mengenali angka tulisan tangan milik *Modified National Institute of Standards and Technology* (MNIST). Supaya hasilnya nanti bisa kita bandingkan dengan metode sebelumnya, yaitu Regresi Logistik. Langkah-langkah pemrogramannya diuraikan sebagai berikut.

Pertama, import *library* yang dibutuhkan.

```
from __future__ import absolute_import, division, print_function

import tensorflow as tf
from tensorflow.keras import Model, layers
import numpy as np
```

Kemudian, siapkan parameter-parameter yang dibutuhkan.

```
# Parameter-parameter yang digunakan.
jumlah_kelas = 10 # Angka 0-9.

# Parameter Pelatihan.
kecepatan_pembelajaran = 0.001
langkah_pelatihan = 1000
ukuran_batch = 128
tampilkan_langkah = 50
```



```
# Parameter Jaringan.  
conv1_filters = 32 # jumlah filter untuk lapisan konv. pertama.  
conv2_filters = 64 # jumlah filter untuk lapisan konv. ke-2.  
fc1_units = 1024 # jumlah neuron untuk lapisan terkoneksi penuh pertama.
```

Selanjutnya, silahkan siapkan data MNIST.

```
# Siapkan data MNIST.  
from tensorflow.keras.datasets import mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
# Ubah menjadi float32.  
x_train, x_test = np.array(x_train, np.float32), np.array(x_test,  
np.float32)  
  
# Normalisasi nilai gambar dari [0, 255] hingga [0, 1].  
x_train, x_test = x_train / 255., x_test / 255.
```

Gunakan API `tf.data` untuk mengacak dan mengumpulkan data.

```
# Gunakan API tf.data.  
train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))  
train_data = train_data.repeat().shuffle(5000).batch(ukuran_batch).  
prefetch(1)
```

Kemudian, silahkan buat model CNN.

```

# Buat Model TF.
class ConvNet(Model):
    # Setel lapisan.
    def __init__(self):
        super(ConvNet, self).__init__()
        # Convolution Layer dengan 32 filter dan ukuran kernel 5.
        self.conv1 = layers.Conv2D(32, kernel_size=5, activation=tf.
nn.relu)
        # Max Pooling (down-sampling) dengan ukuran kernel 2 dan langkah
        2.
        self.maxpool1 = layers.MaxPool2D(2, strides=2)

        # Convolution Layer dengan 64 filter dan ukuran kernel 3.
        self.conv2 = layers.Conv2D(64, kernel_size=3, activation=tf.
nn.relu)
        # Max Pooling (down-sampling) dengan ukuran kernel 2 dan langkah
        2.
        self.maxpool2 = layers.MaxPool2D(2, strides=2)
        # Ratakan data menjadi vektor 1-D untuk lapisan yang terhubung
        sepenuhnya.
        self.flatten = layers.Flatten()

        # Lapisan terhubung sepenuhnya.
        self.fc1 = layers.Dense(1024)
        # Terapkan Dropout (jika is_training False, dropout tidak diterapkan).
        self.dropout = layers.Dropout(rate=0.5)

        # Lapisan keluaran, prediksi kelas.
        self.out = layers.Dense(jumlah_kelas)

# Atur operan maju.
def call(self, x, is_training=False):
    x = tf.reshape(x, [-1, 28, 28, 1])
    x = self.conv1(x)
    x = self.maxpool1(x)
    x = self.conv2(x)

```

```

x = self.maxpool2(x)
x = self.flatten(x)
x = self.fc1(x)
x = self.dropout(x, training=is_training)
x = self.out(x)
if not is_training:
    # tf cross entropy mengharapkan logits tanpa softmax, jadi hanya
    # terapkan softmax saat tidak berlatih.
    x = tf.nn.softmax(x)
return x

# Buat model jaringan Deep Learning.
conv_net = ConvNet()

```

Siapkan fungsi untuk kerugian *Cross-Entropy*.

```

# Kerugian Cross-Entropy.
# Perhatikan bahwa ini akan menerapkan 'softmax' ke logits.
def cross_entropy_loss(x, y):
    # Ubah label menjadi int 64 untuk fungsi tf cross-entropy.
    y = tf.cast(y, tf.int64)
    # Terapkan softmax ke logits dan hitung cross-entropy.
    loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
logits=x)
    # Kerugian rata-rata di seluruh kelompok.
    return tf.reduce_mean(loss)

# Metrik akurasi.
def accuracy(y_pred, y_true):
    # Kelas prediksi adalah indeks nilai tertinggi dalam vektor
    prediksi (yaitu argmax).
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true,
tf.int64))
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32),
axis=-1)

# Pengoptimal penurunan gradien stokastik.
optimizer = tf.optimizers.Adam(kecepatan_pembelajaran)

```

Silahkan buat fungsi untuk menjalankan proses pengoptimalan.

```
# Proses pengoptimalan.
def run_optimization(x, y):
    # Bungkus komputasi di dalam GradientTape untuk diferensiasi otomatis.
    with tf.GradientTape() as g:

        # Umpan maju.
        pred = conv_net(x, is_training=True)

        # Hitung kerugian.
        loss = cross_entropy_loss(pred, y)

        # Variabel yang akan diperbarui, yaitu variabel yang dapat dilatih.
        trainable_variables = conv_net.trainable_variables

        # Hitung gradien.
        gradients = g.gradient(loss, trainable_variables)

        # Perbarui W dan b gradien berikut.
        optimizer.apply_gradients(zip(gradients, trainable_variables))
```

Selanjutnya, silahkan jalankan pelatihan untuk sejumlah langkah tertentu.

```
# Jalankan pelatihan untuk sejumlah langkah tertentu.
for step, (batch_x, batch_y) in enumerate(train_data.take(langkah_pelatihan), 1):
    # Jalankan pengoptimalan untuk memperbarui nilai W dan b.
    run_optimization(batch_x, batch_y)

    if step % tampilkan_langkah == 0:
        pred = conv_net(batch_x)
        loss = cross_entropy_loss(pred, batch_y)
        acc = accuracy(pred, batch_y)
        print("step: %i, loss: %f, accuracy: %f" % (step, loss, acc))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.18.

```
step: 100, loss: 1.463799, accuracy: 1.000000
step: 150, loss: 1.466812, accuracy: 1.000000
step: 200, loss: 1.463316, accuracy: 1.000000
step: 250, loss: 1.480136, accuracy: 0.992188
step: 300, loss: 1.472813, accuracy: 0.992188
step: 350, loss: 1.478996, accuracy: 0.992188
step: 400, loss: 1.475980, accuracy: 0.984375
step: 450, loss: 1.469335, accuracy: 0.992188
step: 500, loss: 1.466777, accuracy: 1.000000
step: 550, loss: 1.462899, accuracy: 1.000000
step: 600, loss: 1.470868, accuracy: 0.992188
step: 650, loss: 1.466607, accuracy: 1.000000
step: 700, loss: 1.465705, accuracy: 1.000000
step: 750, loss: 1.470385, accuracy: 1.000000
step: 800, loss: 1.475498, accuracy: 0.992188
step: 850, loss: 1.466452, accuracy: 1.000000
step: 900, loss: 1.464823, accuracy: 1.000000
step: 950, loss: 1.469388, accuracy: 0.992188
step: 1000, loss: 1.470226, accuracy: 0.992188
```

GAMBAR 4.18.
Proses pelatihan CNN

Selanjutnya, silahkan lakukan uji model pada sekumpulan data validasi.

```
# Uji model pada set validasi.
pred = conv_net(x_test)
print("Akurasi Tes: %f" % accuracy(pred, y_test))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.19.

Akurasi Tes: 0.906300

GAMBAR 4.19.
Hasil uji model pada sekumpulan data validasi

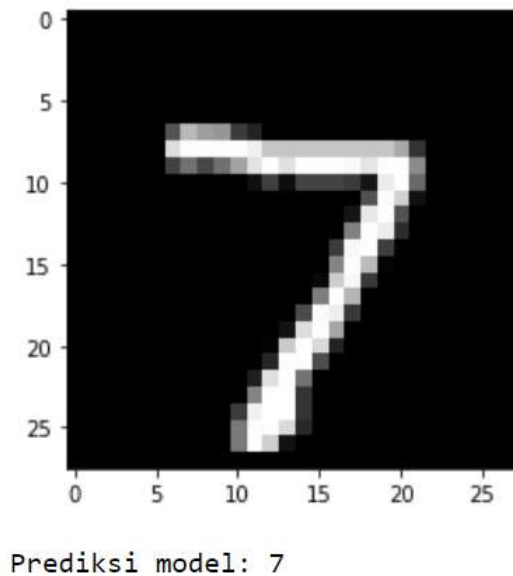
Langkah terakhir, untuk menampilkan visualisasi hasil prediksi pengenalan angka tulisan tangan MNIST, bisa diambil 5 contoh gambar angka yang ingin dikenali.

```
# Visualisasikan prediksi.
import matplotlib.pyplot as plt

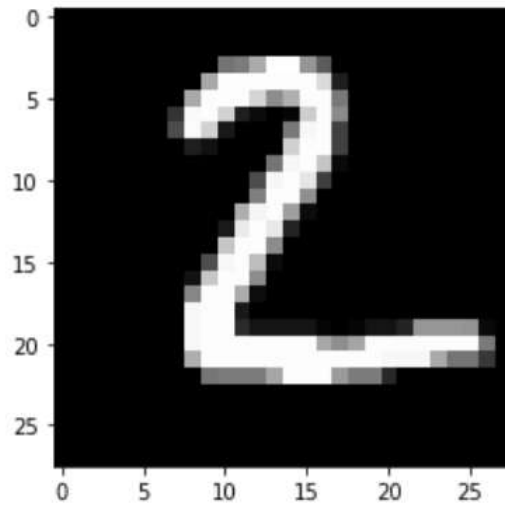
# Memprediksi 5 gambar dari set validasi.
n_images = 5
test_images = x_test[:n_images]
predictions = conv_net(test_images)

# Menampilkan gambar dan prediksi model.
for i in range(n_images):
    plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray')
    plt.show()
    print("Prediksi model: %i" % np.argmax(predictions.numpy()[i]))
```

Jika dijalankan, maka hasilnya seperti diperlihatkan pada Gambar 4.20 sampai dengan Gambar 4.24.

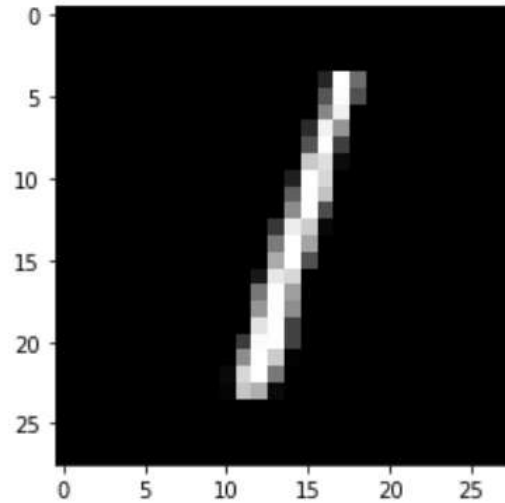


GAMBAR 4.20.
Hasil pengenalan angka 7 tulisan tangan MNIST



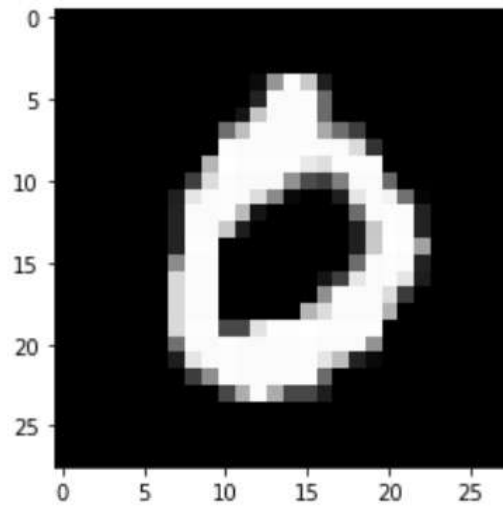
Prediksi model: 2

GAMBAR 4.21.
Hasil pengenalan angka 2 tulisan tangan MNIST



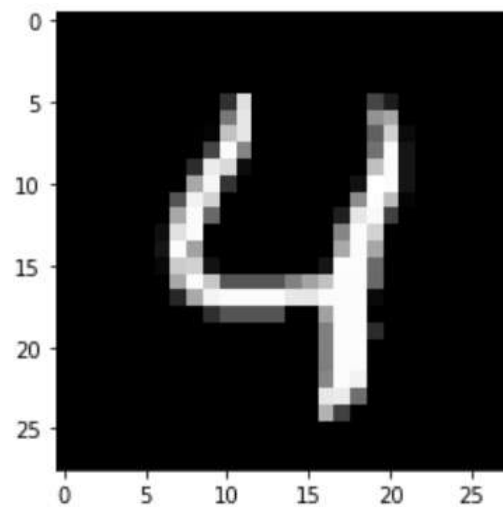
Prediksi model: 1

GAMBAR 4.22.
Hasil pengenalan angka 1 tulisan tangan MNIST



Prediksi model: 0

GAMBAR 4.23.
Hasil pengenalan angka 0 tulisan tangan MNIST



Prediksi model: 4

GAMBAR 4.24.
Hasil pengenalan angka 4 tulisan tangan MNIST

Dengan demikian, jika dibandingkan, hasil kedua metode pengenalan angka tulisan tangan MNIST, menggunakan Regresi Logistik sebelumnya dengan menggunakan CNN, dapat ditabelkan seperti pada Tabel 4.1.

TABEL 4.1.
Perbandingan hasil pengenalan angka tulisan tangan MNIST

Metrik	Regresi Logistik	CNN	Keterangan
Akurasi Pelatihan	0.906250	0.992188	Jumlah langkah pelatihan (iterasi) = 1000
Akurasi Pengujian	0.906300	0.991900	

Dari hasil perbandingan pengenalan angka tulisan tangan MNIST pada Tabel 4.1, terlihat hasil pengenalan angka menggunakan CNN lebih baik dibandingkan dengan menggunakan metode Regresi Logistik, ditunjukkan dengan akurasi pelatihan dan pengujian yang lebih baik. Dengan jumlah iterasi pelatihan yang sama.



4.5. Studi Kasus Sistem Pengenalan Ikan

Setelah kita bereksperimen dengan TensorFlow untuk menyelesaikan berbagai macam permasalahan, saatnya kita gunakan untuk menyelesaikan permasalahan yang lebih menarik dan menantang. Kita masih menggunakan arsitektur utama *Deep Network* yang terkenal dan banyak digunakan di berbagai bidang penerapan, yaitu *Convolutional Neural Networks* (CNNs). Kali ini CNN akan kita gunakan untuk mengenali gambar sesuai *dataset* (Data Latih dan Data Uji) hasil koleksi kita sendiri. Barangkali inilah contoh yang ditunggu-tunggu para pembaca (GR sedikit). Karena biasanya para peneliti bereksperimen dengan datanya sendiri, agar lebih leluasa untuk mempublikasikan hasil simulasinya. Sehingga misalnya kalo kita cari di mesin pencari google, kata kunci andalannya adalah *image classification using own dataset*. Bahkan sampai hafal biasanya diakhiri dengan kata kunci *github download* (senyum sedikit).




Baiklah, untuk contoh kita kali ini, *dataset* yang kami maksud adalah hasil koleksi berupa hasil pencarian jenis-jenis ikan yang ada di internet. Pencarian dan pengumpulan *dataset* ini dilakukan oleh salah satu mahasiswa kami, untuk penelitian Skripsi-nya. Penelitiannya tentang pengenalan jenis ikan, menggunakan metode *Support Vector Machine* (SVM) (Ferdiansyah, 2020). Kali ini kita akan gunakan *dataset* ini, untuk kita kenali jenis-jenis ikan tersebut menggunakan CNN dengan TensorFlow.

Dataset yang akan digunakan berjumlah 250 data, yang digunakan untuk proses pelatihan (Data Latih) sebanyak 200 data, dan pengujian (Data Uji), sebanyak 50 data. Data ini terbagi secara merata untuk 5 (lima) jenis ikan. Sehingga untuk masing-masing jenis ikan, terdapat 40 Data Latih dan 10 Data Uji. Berikut ini adalah lima jenis ikan hias yang digunakan sebagai bahan penelitian (Ferdiansyah, 2020).

TABEL 4.2.
Dataset jenis ikan hias

Jenis Ikan Hias	
	<p>Ikan Komet (<i>Carassius auratus auratus</i>)</p> <p>Morfologi:</p> <ul style="list-style-type: none"> - Bentuk tubuh ikan komet cenderung memanjang, memipih tegak atau dikenal dengan istilah compressed. - Mulut ikan komet ada pada bagian tengah ujung kepala terminal atau berada tepat di ujung hidung. - Sirip ekor pada ikan komet berbentuk berlekuk tunggal. - Sirip punggung ikan komet agak memanjang. - Ikan komet memiliki bentuk sirip yang lebih panjang dari ikan mas, meskipun jika didekatkan keduanya akan sangat mirip.
	<p>Ikan Maanfish (<i>Pterophyllum scalare</i>)</p> <p>Morfologi:</p> <ul style="list-style-type: none"> - Bentuk tubuh pipih, dengan tubuh berbentuk seperti anak panah. - Memiliki warna dan jenis yang bervariasi seperti warna putih, hitam, dan kuning. - Pada bagian dadanya terdapat dua buah sirip yang panjangnya menjuntai sampai ke bagian ekor.



Jenis Ikan Hias	
	<p>Ikan Molly (<i>Poecilia sphenops</i>)</p> <p>Morfologi:</p> <ul style="list-style-type: none">- panjang tubuhnya sekita 5-7 cm.- Bentuk tubuhnya cukup unik moncong kedepan yang terlihat dari bagian kepala hingga mulut.- bentuk sirip ekor agak berbeda terutama pada molly jantan, bentuk unik seperti sabit.
	<p>Ikan Redfine (<i>Epalzeorhynchus frenatum</i>)</p> <p>Morfologi:</p> <ul style="list-style-type: none">- Memiliki ukuran tubuh yang tergolong sedang dengan panjang sekitar 12 cm.- Bentuk tubuhnya tampak memanjang dengan bagian badan yang terlihat lebih besar.- Warna yang tampak di bagian sirip dan ekornya hanya warna merah cerah.
	<p>Ikan Zebra (<i>Danio rerio</i>)</p> <p>Morfologi:</p> <ul style="list-style-type: none">- Pada tubuh ikan ini ditutupi oleh garis-garis berwarna putih kekuningan dan hitam yang berawal dari pangkal ekor.- Bentuk tubuh pipih dengan perut sedikit membesar.- Warna pada ikan jantan terlihat lebih cerah dan menarik dibandingkan dengan ikan betina.

Baiklah, sekarang saatnya kita menuliskan pemrogramannya, sesuai dengan urutan langkah-langkah berikut ini.

Pertama, seperti biasanya, kita *import library-library* yang dibutuhkan.



```
# Library yang dibutuhkan

import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
import glob
import time
import cv2
import os

%matplotlib inline
```

Selanjutnya, siapkan Data Latih dan Data Uji. Baca dari direktori penyimpanan.

```
# Panggil Dataset (Data Latih)
# Urutan Data Ikan
# komet
# maanfish
# molly
# redfine
# zebra

dir_komet = "./Dataset/training-data/komet/*.jpg"
dir_maanfish = "./Dataset/training-data/maanfish/*.jpg"
dir_molly = "./Dataset/training-data/molly/*.jpg"
dir_redfine = "./Dataset/training-data/redfine/*.jpg"
dir_zebra = "./Dataset/training-data/zebra/*.jpg"
```

```

files = glob.glob(dir_komet)
komet = []
for f1 in files:
    img = cv2.imread(f1)
    komet.append(img)

files = glob.glob(dir_maanfish)
maanfish = []
for f1 in files:
    img = cv2.imread(f1)
    maanfish.append(img)

files = glob.glob(dir_molly)
molly = []

for f1 in files:
    img = cv2.imread(f1)
    molly.append(img)

files = glob.glob(dir_redfine)
redfine = []
for f1 in files:
    img = cv2.imread(f1)
    redfine.append(img)

files = glob.glob(dir_zebra)
zebra = []
for f1 in files:
    img = cv2.imread(f1)
    zebra.append(img)

# Panggil Dataset (Data Uji)
dir_komet_uji = "./Dataset/testing-data/komet/*.jpg"
dir_maanfish_uji = "./Dataset/testing-data/maanfish/*.jpg"
dir_molly_uji = "./Dataset/testing-data/molly/*.jpg"
dir_redfine_uji = "./Dataset/testing-data/redfine/*.jpg"
dir_zebra_uji = "./Dataset/testing-data/zebra/*.jpg"

```

```

files = glob.glob(dir_komet_uji)
komet_uji = []
for f1 in files:
    img = cv2.imread(f1)
    komet_uji.append(img)

files = glob.glob(dir_maanfish_uji)
maanfish_uji = []
for f1 in files:
    img = cv2.imread(f1)
    maanfish_uji.append(img)

files = glob.glob(dir_molly_uji)
molly_uji = []
for f1 in files:
    img = cv2.imread(f1)
    molly_uji.append(img)

files = glob.glob(dir_redfine_uji)
redfine_uji = []
for f1 in files:
    img = cv2.imread(f1)
    redfine_uji.append(img)

files = glob.glob(dir_zebra_uji)
zebra_uji = []
for f1 in files:
    img = cv2.imread(f1)
    zebra_uji.append(img)

```

Setelah itu, bisa dilakukan pengecekan ukuran Data Latih.

```

# Cek ukuran Data Latih
[np.shape(komet), np.shape(maanfish), np.shape(molly),
np.shape(redfine), np.shape(zebra)]

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.25.

```
Out[49]: [(40, 100, 100, 3),
          (40, 100, 100, 3),
          (40, 100, 100, 3),
          (40, 100, 100, 3),
          (40, 100, 100, 3)]
```

GAMBAR 4.25.
Ukuran Data Latih

Terlihat dari Gambar 4.25, Data Latih masing-masing jenis ikan sebanyak 40 gambar. Ukuran masing-masing 100x100 piksel. Angka 3 menunjukkan saluran warna R, G, B. Nantinya menjadi dasar CNN mengambil bentuk tensor (tinggi_gambar, lebar_gambar, saluran_warna R, G, B).

Dengan cara yang sama, bisa dilakukan untuk pengecekan ukuran Data Uji.

```
# Cek ukuran Data Uji
[np.shape(komet_uji), np.shape(maanfish_uji), np.shape(molly_uji),
 np.shape(redfine_uji), np.shape(zebra_uji)]
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.26.

```
Out[50]: [(10, 100, 100, 3),
          (10, 100, 100, 3),
          (10, 100, 100, 3),
          (10, 100, 100, 3),
          (10, 100, 100, 3)]
```

GAMBAR 4.26.
Ukuran Data Uji

Terlihat dari Gambar 4.26, Data Uji masing-masing jenis ikan sebanyak 10 gambar. Ukuran masing-masing 100x100 piksel. Angka 3 menunjukkan saluran warna R, G, B.

Langkah selanjutnya, Data Latih semua jenis ikan dijadikan dalam satu variabel, yaitu X.

```
# Data Latih semua jenis ikan digabung
X = np.vstack((komet, maanfish, molly, redfine, zebra)).astype(np.float32)
```

Setelah digabung, untuk memastikan ukuran Data Latih, dapat dicek dengan menggunakan perintah berikut.

```
# Cek ukuran Data Latih setelah digabung
X.shape
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.27.

```
Out[99]: (200, 100, 100, 3)
```

GAMBAR 4.27.

Ukuran Data Latih setelah semua jenis ikan digabung

Terlihat dari Gambar 4.27, jumlah Data Latih sebanyak 200 data. Setelah itu, dapat dilakukan pengecekan nilai X (Data Latih) sebelum dilakukan normalisasi.

```
# Tampilkan sebelum dinormalisasi
print(X)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.28.


```

[[209. 216. 201.]
 [209. 216. 201.]
 [209. 216. 201.]
 ...
 [209. 216. 201.]
 [209. 216. 201.]
 [209. 216. 201.]]

[[209. 216. 201.]
 [209. 216. 201.]
 [209. 216. 201.]
 ...
 [209. 216. 201.]
 [209. 216. 201.]
 [209. 216. 201.]]]]

```

GAMBAR 4.28.
Data Latih sebelum dinormalisasi

Setelah itu, dilakukan normalisasi Data Latih.

```

# Data Latih semua jenis ikan dinormalisasi
X = X / 255.0

```

Setelah dilakukan normalisasi Data Latih, dapat dilakukan pengecekan nilainya.

```

# Tampilkan setelah dinormalisasi
print(X)

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.29.

```

[[0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 ...
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]]

[[0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 ...
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]
 [0.81960785 0.84705883 0.7882353 ]]]]

```

GAMBAR 4.29.
Data Latih setelah dinormalisasi

Dengan cara yang sama, dilakukan penggabungan untuk Data Uji.

```

# Data Uji semua jenis ikan digabung
X_uji = np.vstack((komet_uji, maanfish_uji, molly_uji, redfine_uji,
 zebra_uji)).astype(np.float32)

```

Pengecekan ukuran Data Uji.

```

# Cek ukuran Data Uji
X_uji.shape

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.30.

```
Out[58]: (50, 100, 100, 3)
```

GAMBAR 4.30.
Data Uji setelah dinormalisasi

Terlihat dari Gambar 4.30, terdapat 50 Data Uji.

Kemudian, untuk meyakinkan, bisa dilakukan pengecekan Data Uji sebelum dinormalisasi.

```
# Data Uji sebelum dinormalisasi  
print(X_uji)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.31.

```
[[ 3.  29.  0.]  
 [ 6.  33.  0.]  
 [ 9.  36.  0.]  
 ...  
 [ 9.  43.  3.]  
 [ 4.  36.  1.]  
 [ 0.  32.  0.]]  
  
[[ 2.  28.  0.]  
 [ 6.  33.  0.]  
 [ 8.  35.  0.]  
 ...  
 [ 9.  43.  3.]  
 [ 4.  36.  1.]  
 [ 2.  34.  0.]]]]
```

GAMBAR 4.31.
Data Uji sebelum dinormalisasi

Kemudian dilakukan normalisasi Data Uji.

```
# Data Uji semua jenis ikan dinormalisasi
X_uji = X_uji / 255.0
```

Pengecekan Data Uji setelah dinormalisasi.

```
# Cek setelah dinormalisasi
print(X_uji)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.32.

```
[[0.01176471 0.11372549 0.          ]
 [0.02352941 0.12941177 0.          ]
 [0.03529412 0.14117648 0.          ]
 ...
 [0.03529412 0.16862746 0.01176471]
 [0.01568628 0.14117648 0.00392157]
 [0.          0.1254902  0.          ]]]

[[0.00784314 0.10980392 0.          ]
 [0.02352941 0.12941177 0.          ]
 [0.03137255 0.13725491 0.          ]
 ...
 [0.03529412 0.16862746 0.01176471]
 [0.01568628 0.14117648 0.00392157]
 [0.00784314 0.13333334 0.          ]]]]
```

GAMBAR 4.32.
Data Uji setelah dinormalisasi

Selanjutnya, dilakukan pemberian label sesuai jenis ikan pada Data Latih. Ikan komet (diberi label: 0), maanfish (diberi label: 1), molly (diberi label: 2), redfine (diberi label: 3), dan zebra (diberi label: 4).


```
# Semua label ikan Data Latih digabung
y = np.hstack((label_komet, label_maanfish, label_molly, label_
redfine, label_zebra))
```

Setelah digabung dalam satu variabel, yaitu y, dilakukan pengecekan ukuran label ikan Data Latih.

```
# Cek ukuran label ikan Data Latih
y.shape
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.34.

Out[65]: (200,)

GAMBAR 4.34.
Ukuran label Data Latih

Terlihat dari Gambar 4.34, setelah digabung label ikan untuk Data Latih, terdapat 200 data label ikan. Sesuai dengan jumlah Data Latihnya.

Dengan cara yang sama, dilakukan pemberian label ikan untuk Data Uji. Ikan komet (diberi label: 0), maanfish (diberi label: 1), molly (diberi label: 2), redfine (diberi label: 3), dan zebra (diberi label: 4). Masing-masing sebanyak 10 label ikan untuk 10 Data Uji.

```
# Label ikan Data Uji
label_komet_uji = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
label_maanfish_uji = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
label_molly_uji = np.array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
label_redfine_uji = np.array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
label_zebra_uji = np.array([4, 4, 4, 4, 4, 4, 4, 4, 4, 4])
```

Dengan cara yang sama yang dilakukan pada label ikan Data Latih, dilakukan pengecekan ukuran label ikan untuk Data Uji.

```
# Cek ukuran label Data Uji
[np.shape(label_komet_uji), np.shape(label_maanfiah_uji),
np.shape(label_molly_uji), np.shape(label_redfine_uji),
np.shape(label_zebra_uji)]
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.35.

```
Out[67]: [(10,), (10,), (10,), (10,), (10,)]
```

GAMBAR 4.35.
Ukuran masing-masing label Data Uji

Kemudian, seperti pada label ikan Data Latih, dilakukan penggabungan semua label ikan untuk Data Uji.

```
# Semua label ikan Data Uji digabung
y_uji = np.hstack((label_komet_uji, label_maanfiah_uji, label_molly_
uji, label_redfine_uji, label_zebra_uji))
```

Setelah digabung dalam satu variabel, yaitu `y_uji`, dilakukan pengecekan ukuran label ikan Data Uji.

```
# Cek ukuran label ikan Data Uji
y_uji.shape
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.36.

```
Out[69]: (50,)
```

GAMBAR 4.36.
Ukuran label Data Uji

Terlihat dari Gambar 4.36, setelah digabung label ikan untuk Data Uji, terdapat 50 data label ikan. Sesuai dengan jumlah Data Ujinya.

Berikutnya, untuk menampilkan beberapa gambar dari Data Latih yang digunakan, bisa ditampilkan beberapa sampel. Masing-masing jenis ikan ditampilkan sebanyak 5 data.

```
# Tampilkan sampel Data Latih

class_names = ['komet', 'maanfish', 'molly', 'redfine', 'zebra']

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(komet[i], cmap=plt.cm.binary)
    plt.xlabel(label_komet[i])
    plt.ylabel(class_names[0])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(2,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(maanfish[i], cmap=plt.cm.binary)
    plt.xlabel(label_maanfish[i])
    plt.ylabel(class_names[1])
plt.show()
```



```

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(3,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(molly[i], cmap=plt.cm.binary)
    plt.xlabel(label_molly[i])
    plt.ylabel(class_names[2])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(redfine[i], cmap=plt.cm.binary)
    plt.xlabel(label_redfine[i])
    plt.ylabel(class_names[3])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(zebra[i], cmap=plt.cm.binary)
    plt.xlabel(label_zebra[i])
    plt.ylabel(class_names[4])
plt.show()

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.37.



GAMBAR 4.37.
Sampel Data Latih

Dengan cara yang sama, untuk menampilkan beberapa gambar dari Data Uji yang digunakan, bisa ditampilkan beberapa sampel. Masing-masing jenis ikan, ditampilkan sebanyak 5 data.

```

# Tampilkan sampel Data Uji

class_names = ['komet', 'maanfish', 'molly', 'redfine', 'zebra']

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(komet_uji[i], cmap=plt.cm.binary)
    plt.xlabel(label_komet_uji[i])
    plt.ylabel(class_names[0])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(2,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(maanfish_uji[i], cmap=plt.cm.binary)
    plt.xlabel(label_maanfish_uji[i])
    plt.ylabel(class_names[1])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(3,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(molly_uji[i], cmap=plt.cm.binary)
    plt.xlabel(label_molly_uji[i])
    plt.ylabel(class_names[2])
plt.show()

```

```

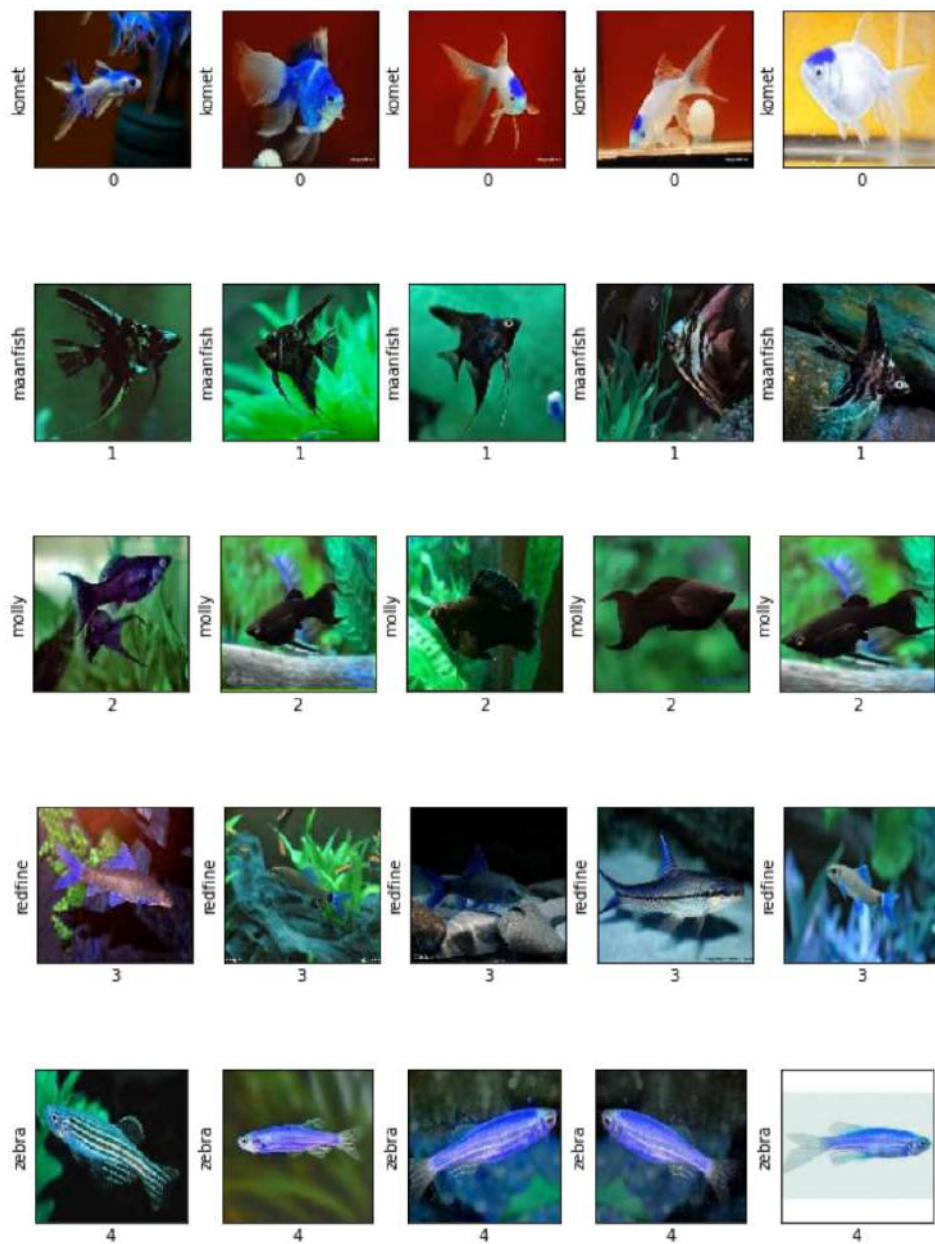
plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(redfine_uji[i], cmap=plt.cm.binary)
    plt.xlabel(label_redfine_uji[i])
    plt.ylabel(class_names[3])
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(zebra_uji[i], cmap=plt.cm.binary)
    plt.xlabel(label_zebra_uji[i])
    plt.ylabel(class_names[4])
plt.show()

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.38.

Tahap selanjutnya, mulai masuk ke sistem CNN. Buat basis konvolusional 10 baris kode di bawah ini untuk mendefinisikan dasar convolutional menggunakan pola umum: setumpuk lapisan *Conv2D* dan *MaxPooling2D*. Sebagai masukan, CNN mengambil bentuk tensor (tinggi_gambar, lebar_gambar, saluran_warna), mengabaikan ukuran tumpukan. *Color_channels* merujuk ke (R, G, B). CNN untuk memproses masukan dalam bentuk (100, 100, 3), sesuai Data Latih yang digunakan. Dapat dilakukan dengan meneruskan argumen *input_shape* ke lapisan pertama kita.



GAMBAR 4.38.
Sampel Data Uji

```

model = models.Sequential()
model.add(layers.Conv2D(100, (3, 3), activation='relu', input_
shape=(100, 100, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(200, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(200, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(200, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(200, (3, 3), activation='relu'))

```

Setelah itu, dapat ditampilkan arsitektur model kita sejauh ini.

```

# Menampilkan arsitektur model CNN
model.summary()

```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.39.

Dari Gambar 4.39, dapat dilihat bahwa keluaran dari setiap lapisan *Conv2D* dan *MaxPooling2D* adalah bentuk tensor 3D (tinggi, lebar, saluran). Dimensi lebar dan tinggi cenderung menyusut saat masuk lebih dalam di jaringan. Biasanya, saat lebar dan tinggi menyusut, dapat (secara komputasi) untuk ditambahkan lebih banyak saluran keluaran di setiap lapisan *Conv2D*.

Tambahkan lapisan padat di atas untuk melengkapi model kita. Dapat dimasukkan tensor keluaran terakhir dari basis konvolusional (berbentuk (2, 2, 200)) ke dalam satu atau lebih lapisan padat untuk melakukan klasifikasi. Lapisan padat mengambil vektor sebagai masukan (yaitu 1D), sedangkan keluaran saat ini adalah tensor 3D. Pertama, akan diratakan (atau dibuka gulungan) keluaran 3D ke 1D, lalu ditambahkan satu atau lebih lapisan padat di atasnya. Sistem memiliki 5 kelas keluaran (5 jenis ikan), jadi digunakan lapisan *Dense* akhir dengan 5 keluaran.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 98, 98, 100)	2800
max_pooling2d_4 (MaxPooling2D)	(None, 49, 49, 100)	0
conv2d_6 (Conv2D)	(None, 47, 47, 200)	180200
max_pooling2d_5 (MaxPooling2D)	(None, 23, 23, 200)	0
conv2d_7 (Conv2D)	(None, 21, 21, 200)	360200
max_pooling2d_6 (MaxPooling2D)	(None, 10, 10, 200)	0
conv2d_8 (Conv2D)	(None, 8, 8, 200)	360200
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 200)	0
conv2d_9 (Conv2D)	(None, 2, 2, 200)	360200
Total params: 1,263,600		
Trainable params: 1,263,600		
Non-trainable params: 0		

GAMBAR 4.39.
Arsitektur model CNN

```
model.add(layers.Flatten())
model.add(layers.Dense(200, activation='relu'))
model.add(layers.Dense(5))
```

Sehingga, arsitektur lengkap model CNN bisa ditampilkan.

```
# menampilkan arsitektur lengkap model CNN
model.summary()
```


Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.40.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 98, 98, 100)	2800
max_pooling2d_4 (MaxPooling2D)	(None, 49, 49, 100)	0
conv2d_6 (Conv2D)	(None, 47, 47, 200)	180200
max_pooling2d_5 (MaxPooling2D)	(None, 23, 23, 200)	0
conv2d_7 (Conv2D)	(None, 21, 21, 200)	360200
max_pooling2d_6 (MaxPooling2D)	(None, 10, 10, 200)	0
conv2d_8 (Conv2D)	(None, 8, 8, 200)	360200
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 200)	0
conv2d_9 (Conv2D)	(None, 2, 2, 200)	360200
flatten_1 (Flatten)	(None, 800)	0
dense_2 (Dense)	(None, 200)	160200
dense_3 (Dense)	(None, 5)	1005
Total params: 1,424,805		
Trainable params: 1,424,805		
Non-trainable params: 0		

GAMBAR 4.40.
Arsitektur lengkap model CNN

Selanjutnya, dapat dilakukan pelatihan modelnya, misalkan untuk 20 *epoch* (iterasi). Sekaligus model divalidasi menggunakan Data Uji. Skrip programnya seperti berikut ini.


```
# Kumpulkan dan latih modelnya
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

history = model.fit(X, y, epochs=20,
validation_data=(X_uji, y_uji))
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.41.

```
Epoch 15/20
7/7 [=====] - 38s 5s/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 1.7153 - val_accuracy: 0.8400
Epoch 16/20
7/7 [=====] - 33s 5s/step - loss: 8.8894e-04 - accurac
y: 1.0000 - val_loss: 1.7691 - val_accuracy: 0.8400
Epoch 17/20
7/7 [=====] - 33s 5s/step - loss: 2.8335e-04 - accurac
y: 1.0000 - val_loss: 1.8056 - val_accuracy: 0.8200
Epoch 18/20
7/7 [=====] - 39s 6s/step - loss: 3.3932e-04 - accurac
y: 1.0000 - val_loss: 1.8019 - val_accuracy: 0.8200
Epoch 19/20
7/7 [=====] - 42s 6s/step - loss: 1.7715e-04 - accurac
y: 1.0000 - val_loss: 1.7970 - val_accuracy: 0.8400
Epoch 20/20
7/7 [=====] - 40s 5s/step - loss: 1.4630e-04 - accurac
y: 1.0000 - val_loss: 1.8307 - val_accuracy: 0.8400
```

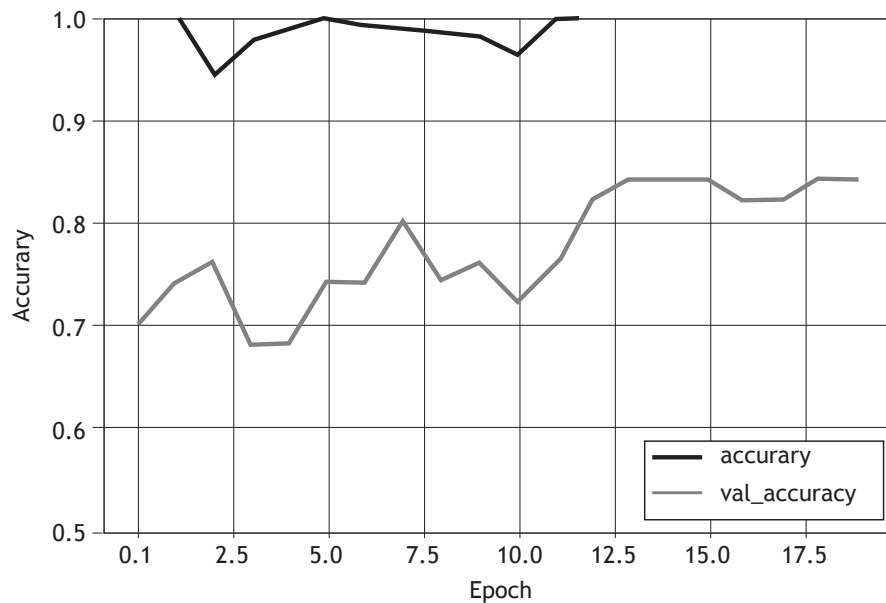
GAMBAR 4.41.

Hasil proses pelatihan CNN dan akurasi validasi dengan Data Uji

Setelah dilakukan pelatihan dan dilihat hasil akurasi validasi menggunakan Data Uji, jika hasilnya kurang memuaskan, prosesnya bisa diulang-ulang. Jika ingin menampilkan dalam bentuk grafik, untuk memudahkan evaluasi model CNN, dapat diketikkan skrip program berikut ini.

```
# Tampilkan grafik hasil pelatihan CNN dan akurasi validasi dengan
# Data Uji
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.grid()
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.42.



GAMBAR 4.42.

Grafik hasil proses pelatihan CNN dan akurasi validasi dengan Data Uji

Dari hasil proses pelatihan CNN dan akurasi validasi dengan Data Uji pada Gambar 4.41 dan Gambar 4.42, selama iterasi berlangsung, menunjukkan fluktuasi yang bagus. Pencapaian nilai akurasi validasi terbaik, diperoleh pada nilai 84 persen. Hal ini sudah cukup bagus.

Kemudian, untuk menampilkan hasil akurasi pengujiannya, dapat dilakukan dengan cara mengevaluasi modelnya. Skrip programnya seperti berikut ini.

```
# Evaluasi modelnya
test_loss, test_acc = model.evaluate(X_uji, y_uji, verbose=2)
# Tampilkan hasil tes akurasi
print('Hasil akurasi pengujian =', test_acc)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 4.43.

```
2/2 - 2s - loss: 1.8307 - accuracy: 0.8400
Hasil akurasi pengujian = 0.8399999737739563
```

GAMBAR 4.43.
Hasil akurasi pengujian CNN

Dari Gambar 4.43 terlihat hasil akurasi pengujian CNN sebesar 0.8399999737739563, atau 84 persen.

Langkah terakhir, untuk menampilkan visualisasi hasil prediksi pengenalan ikan, untuk ke-50 Data Uji, skrip programnya seperti berikut ini.

```
# Visualisasikan hasil prediksi pengenalan ikan.

import matplotlib.pyplot as plt

# Memprediksi 50 gambar dari Data Uji.
n_images = 50
test_images = X_uji[:n_images]
test_label = y_uji[:n_images]
```

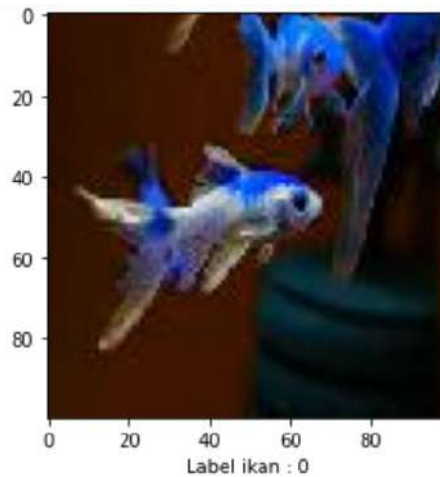
```

predictions = model.predict(test_images)

# Menampilkan gambar dan prediksi model.
for i in range(n_images):
    plt.imshow(test_images[i])
    plt.xlabel("Label ikan: %i" % test_label[i])
    plt.show()
    print("Hasil prediksi model: %i" % np.argmax(predictions[i]))

```

Jika dijalankan, maka beberapa hasil pengenalan masing-masing jenis ikan seperti diperlihatkan pada Gambar 4.44 sampai dengan Gambar 4.53.

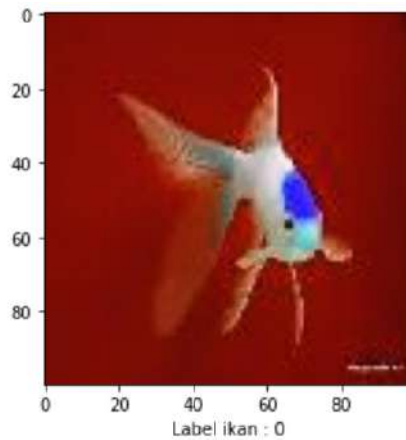


Hasil prediksi model: 0

GAMBAR 4.44.

Contoh visualisasi hasil prediksi pengenalan ikan komet yang benar

Tampak dari Gambar 4.44, CNN bisa memprediksi dengan tepat, jenis ikannya. Label ikan 0, diprediksi sebagai 0, yaitu ikan komet.

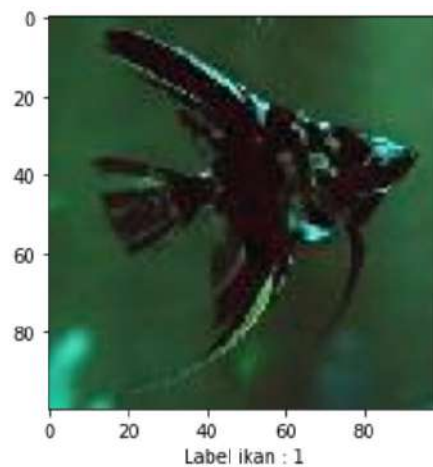


Hasil prediksi model: 2

GAMBAR 4.45.

Contoh visualisasi hasil prediksi pengenalan ikan komet yang salah

Tampak dari Gambar 4.45, CNN tidak bisa memprediksi dengan benar jenis ikannya. Label ikan 0, diprediksi sebagai 2, yaitu ikan komet, diprediksi sebagai ikan molly.

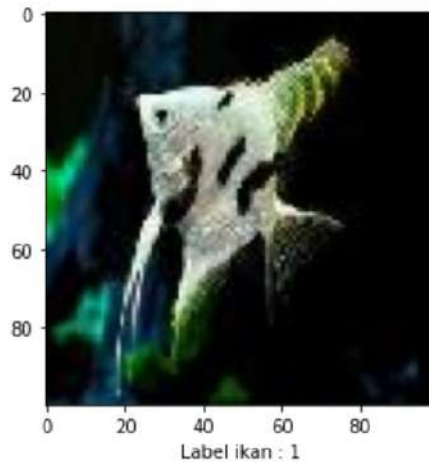


Hasil prediksi model: 1

GAMBAR 4.46.

Contoh visualisasi hasil prediksi pengenalan ikan maanfish yang benar

Tampak dari Gambar 4.46, CNN bisa memprediksi dengan tepat, jenis ikannya. Label ikan 1, diprediksi sebagai 1, yaitu ikan maanfish.

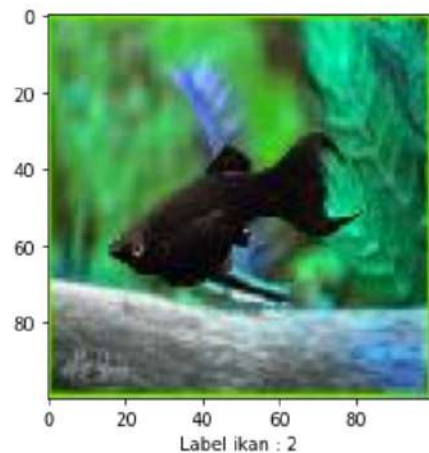


Hasil prediksi model: 2

GAMBAR 4.47.

Contoh visualisasi hasil prediksi pengenalan ikan maanfish yang salah

Tampak dari Gambar 4.47, CNN tidak bisa memprediksi dengan benar jenis ikannya. Label ikan 1, diprediksi sebagai 2, yaitu ikan maanfish, diprediksi sebagai ikan molly.

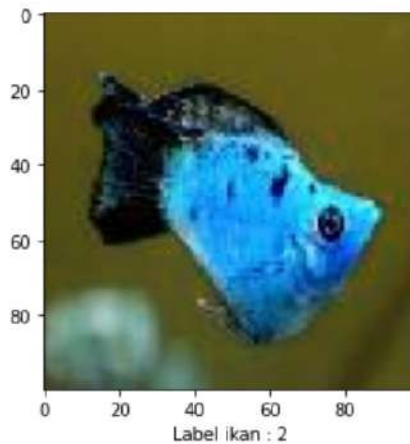


Hasil prediksi model: 2

GAMBAR 4.48.

Contoh visualisasi hasil prediksi pengenalan ikan molly yang benar

Tampak dari Gambar 4.48, CNN bisa memprediksi dengan tepat, jenis ikannya. Label ikan 2, diprediksi sebagai 2, yaitu ikan molly.

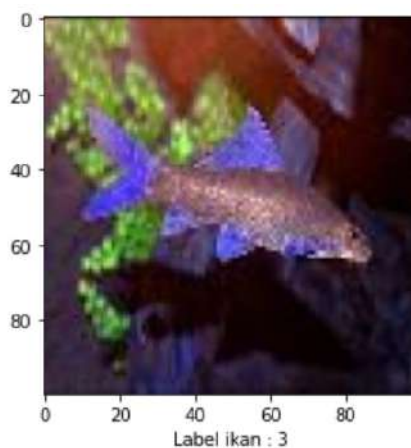


Hasil prediksi model: 0

GAMBAR 4.49.

Contoh visualisasi hasil prediksi pengenalan ikan molly yang salah

Tampak dari Gambar 4.49, CNN tidak bisa memprediksi dengan benar jenis ikannya. Label ikan 2, diprediksi sebagai 0, yaitu ikan molly, diprediksi sebagai ikan komet.

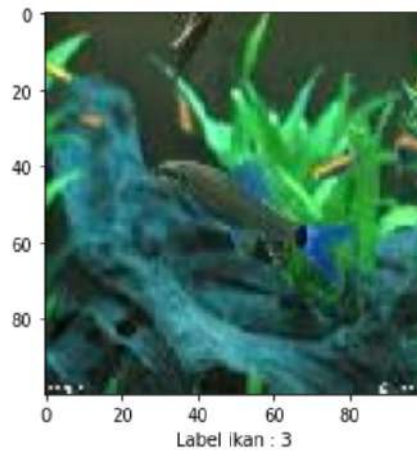


Hasil prediksi model: 3

GAMBAR 4.50.

Contoh visualisasi hasil prediksi pengenalan ikan redfine yang benar

Tampak dari Gambar 4.50, CNN bisa memprediksi dengan tepat, jenis ikannya. Label ikan 3, diprediksi sebagai 3, yaitu ikan redfine.

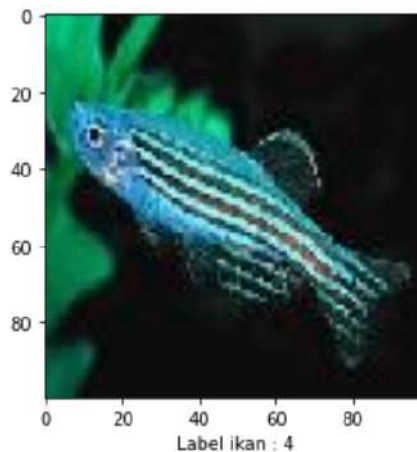


Hasil prediksi model: 4

GAMBAR 4.51.

Contoh visualisasi hasil prediksi pengenalan ikan redfine yang salah

Tampak dari Gambar 4.51, CNN tidak bisa memprediksi dengan benar jenis ikannya. Label ikan 3, diprediksi sebagai 4, yaitu ikan redfine, diprediksi sebagai ikan zebra.

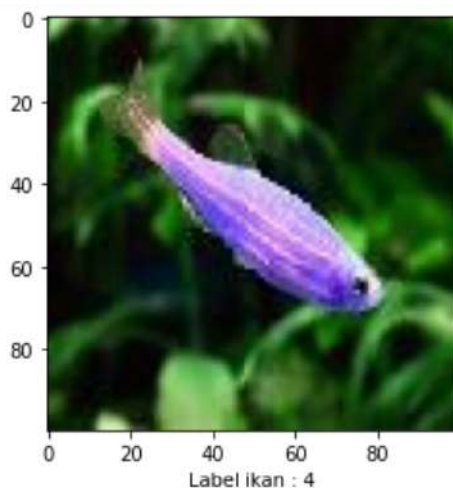


Hasil prediksi model: 4

GAMBAR 4.52.

Contoh visualisasi hasil prediksi pengenalan ikan zebra yang benar

Tampak dari Gambar 4.52, CNN bisa memprediksi dengan tepat, jenis ikannya. Label ikan 4, diprediksi sebagai 4, yaitu ikan zebra.



Hasil prediksi model: 4


GAMBAR 4.53.





Contoh visualisasi hasil prediksi pengenalan ikan zebra yang juga benar

Tampak dari Gambar 4.53, CNN juga bisa memprediksi dengan benar jenis ikannya. Label ikan 4, diprediksi sebagai 4, yaitu ikan zebra. Dan memang untuk pengujian pengenalan jenis ikan zebra, CNN bisa memprediksi semua dengan benar (100%).

Hasil pengujian secara lengkap seperti diperlihatkan pada Tabel 4.3.

TABEL 4.3.
Hasil pengujian pengenalan ikan menggunakan CNN

No	Pengujian Ikan		Hasil Prediksi Benar	Hasil Prediksi Salah	Akurasi (Persentase keberhasilan) (%)
	Gambar	Jenis Ikan			
1		Komet	7	3	70

No	Pengujian Ikan		Hasil Prediksi Benar	Hasil Prediksi Salah	Akurasi (Persentase keberhasilan) (%)
	Gambar	Jenis Ikan			
2		Maanfish	9	1	90
3		Molly	7	3	70
4		Redfine	9	1	90
5		Zebra	10	0	100
Rata-rata					84

Dari Tabel 4.3 terlihat, hasil akurasi pengujian rata-rata, diperoleh nilai sebesar 84%. Hasil ini sama dengan nilai hasil akurasi pengujian CNN sebesar 0.8399999737739563, atau 84 persen yang ditunjukkan pada Gambar 4.43.



Bab 5

Deep Learning dengan Keras

5.1. Deep Learning dengan Keras Dasar

Dalam Bab 5 ini, kita akan mempelajari *framework Deep Learning* lain, selain TensorFlow, yaitu Keras. Keras adalah *library* Python *open source* gratis yang kuat dan mudah digunakan untuk mengembangkan dan mengevaluasi model *Deep Learning* (Brownlee, 2019). Keras membungkus *library* perhitungan numerik yang efisien Theano dan TensorFlow yang memungkinkan digunakan untuk mendefinisikan dan melatih model jaringan saraf hanya dalam beberapa baris kode (Brownlee, 2019).

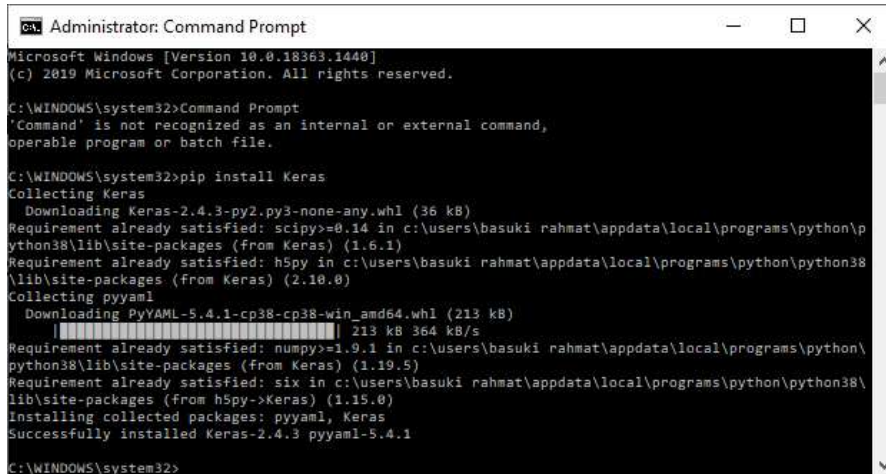
Cara Instalasi Keras, di *command prompt*:

```
pip install Keras
```

Berikut ini adalah langkah-langkah pemrograman python dengan memanfaatkan *library* Keras untuk menyelesaikan permasalahan sistem prediksi. Langkah-langkah pemrograman python dengan Keras adalah sebagai berikut (Brownlee, 2019):

- 1) Penyiapan Data.
- 2) Penentuan Model Keras.

- 3) Kompilasi Model Keras.
- 4) Penyesuaian Model Keras.
- 5) Evaluasi Model Keras, dan
- 6) Pembuatan prediksi



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.1440]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>Command Prompt
'Command' is not recognized as an internal or external command,
operable program or batch file.

C:\WINDOWS\system32>pip install Keras
Collecting Keras
  Downloading Keras-2.4.3-py2.py3-none-any.whl (36 kB)
Requirement already satisfied: scipy>=0.14 in c:\users\basuki rahmat\appdata\local\programs\python\python38\lib\site-packages (from Keras) (1.6.1)
Requirement already satisfied: h5py in c:\users\basuki rahmat\appdata\local\programs\python\python38\lib\site-packages (from Keras) (2.10.0)
Collecting pyyaml
  Downloading PyYAML-5.4.1-cp38-cp38-win_amd64.whl (213 kB)
    |#####| 213 kB 364 kB/s
Requirement already satisfied: numpy>=1.9.1 in c:\users\basuki rahmat\appdata\local\programs\python\python38\lib\site-packages (from Keras) (1.19.5)
Requirement already satisfied: six in c:\users\basuki rahmat\appdata\local\programs\python\python38\lib\site-packages (from h5py->Keras) (1.15.0)
Installing collected packages: pyyaml, Keras
Successfully installed Keras-2.4.3 pyyaml-5.4.1

C:\WINDOWS\system32>
```

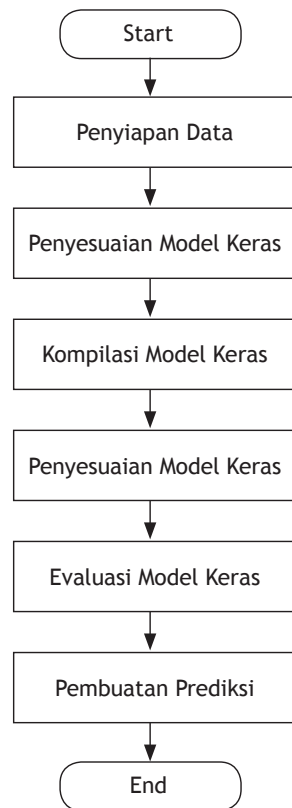
GAMBAR 5.1.
Instalasi Keras

Sehingga secara umum langkah-langkah atau alur penyelesaian permasalahan prediksi menggunakan *Deep Learning* dengan *library* Keras ini seperti diperlihatkan pada Gambar 5.2.

5.2. Prediksi XOR Menggunakan Keras

Selanjutnya, akan kita coba menggunakan Keras untuk menyelesaikan Prediksi XOR. Nanti hasilnya kita bandingkan dengan yang sudah kita kerjakan menggunakan TensorFlow. Data Latih dan arsitektur *Deep Learning* yang digunakan sama dengan sewaktu kita menyelesaikan Prediksi XOR dengan TensorFlow. Tabel 3.1 Data Latih Gerbang Logika XOR dan Gambar 3.26 Arsitektur *Deep Learning* dengan TensorFlow. Penyelesaian menggunakan *Deep Learning* dengan Keras, bisa dilihat pada program berikut.

Panggil *library* yang dibutuhkan, inialisasi konstruktor, dan pembuatan struktur *Deep Learning*.



GAMBAR 5.2.

Diagram alir penyelesaian prediksi menggunakan Deep Learning dengan Keras

```
# Impor `Sequential` dari `keras.models`  
from keras.models import Sequential  
  
# Impor `Dense` dari `keras.layers`  
from keras.layers import Dense  
  
# Inisialisasi konstruktor  
model = Sequential()  
  
# Tambahkan lapisan masukan  
model.add(Dense(2, activation='sigmoid', input_shape=(2,)))
```

```
# Tambahkan satu lapisan tersembunyi
model.add(Dense(2, activation='sigmoid'))

# Tambahkan lapisan keluaran
model.add(Dense(1, activation='sigmoid'))
```

Selanjutnya, ketikkan skrip berikut ini, untuk model *Deep Learning*-nya, dapatkan bobot-bobot dan bias awal.

```
# Bentuk keluaran model
model.output_shape

# Ringkasan model
model.summary()

# Konfigurasi model
model.get_config()

# Buat daftar semua tensor bobot
model.get_weights()
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 5.3.

```
Model: "sequential_1"

Layer (type)                 Output Shape                 Param #
=====
dense_3 (Dense)              (None, 2)                   6
dense_4 (Dense)              (None, 2)                   6
dense_5 (Dense)              (None, 1)                   3
=====
Total params: 15
Trainable params: 15
Non-trainable params: 0
```

```
Out[7]: [array([[ -0.00594807, -1.1232877 ],
               [-1.211806 ,  0.38877928]], dtype=float32),
         array([0., 0.], dtype=float32),
         array([[ 0.6797166 , -0.9488363 ],
               [-0.70436096,  0.28295314]], dtype=float32),
         array([0., 0.], dtype=float32),
         array([[ 0.59368217,
                  -0.5525098 ]], dtype=float32),
         array([0.], dtype=float32)]
```

GAMBAR 5.3.
Model, bobot dan bias awal

Untuk pelatihan *Deep Learning* silahkan ketikkan skrip berikut.

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

model.fit(XOR_X, XOR_Y, epochs=100000, batch_size=1, verbose=1)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.4.

```
Epoch 99992/100000
4/4 [=====] - 0s 2ms/step - loss: 0.6884 - accuracy: 0.5333
Epoch 99993/100000
4/4 [=====] - 0s 2ms/step - loss: 0.4075 - accuracy: 0.9000
Epoch 99994/100000
4/4 [=====] - 0s 2ms/step - loss: 0.4013 - accuracy: 0.7333
Epoch 99995/100000
4/4 [=====] - 0s 2ms/step - loss: 0.2857 - accuracy: 0.9000
Epoch 99996/100000
4/4 [=====] - 0s 2ms/step - loss: 0.4072 - accuracy: 0.9000
Epoch 99997/100000
4/4 [=====] - 0s 2ms/step - loss: 0.3319 - accuracy: 0.8333
Epoch 99998/100000
4/4 [=====] - 0s 2ms/step - loss: 0.6616 - accuracy: 0.5333
Epoch 99999/100000
4/4 [=====] - 0s 2ms/step - loss: 0.6885 - accuracy: 0.5333
Epoch 100000/100000
4/4 [=====] - 0s 2ms/step - loss: 0.6884 - accuracy: 0.5333

Out[8]: <tensorflow.python.keras.callbacks.History at 0x23f21b40e50>
```

GAMBAR 5.4.
Proses pelatihan Deep Learning menggunakan Keras

Untuk menguji coba *Deep Learning* menggunakan Keras ini, silahkan diberikan masukan sebarang, misalkan sama seperti pada cara skrip sebelumnya. Silahkan ketikkan skrip berikut.

```
Hasil_Prediksi_Keras = model.predict(XOR_X)
print(Hasil_Prediksi_Keras)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.5.

```
[[0.3335482]
 [0.3335482]
 [0.9997659]
 [0.3335482]]
```

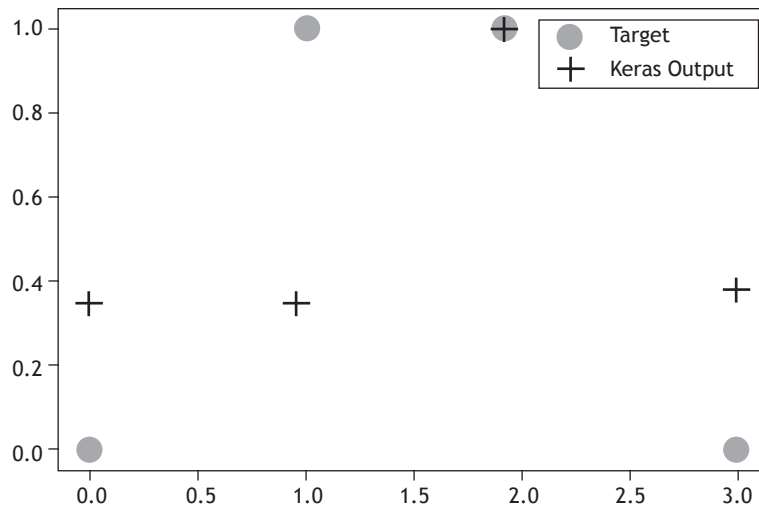
GAMBAR 5.5.

Keluaran hasil prediksi XOR menggunakan Deep Learning dengan Keras

Selanjutnya untuk pengeplotan keluaran hasil prediksi XOR dengan Keras dibandingkan dengan target berupa keluaran yang diinginkan dari gerbang Logika XOR, silahkan ketikkan skrip berikut ini.

```
import matplotlib.pyplot as plt
plt.plot(XOR_Y, 'bo', label='Target', linewidth=2, markersize=12)
plt.plot(Hasil_Prediksi_Keras, 'r+', label='Keras Output',
         linewidth=2, markersize=12)
plt.legend(loc='upper right')
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.6.



GAMBAR 5.6.
Perbandingan prediksi Keras dan target

Selanjutnya untuk menghitung kesalahan prediksi XOR dengan Keras, digunakan *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE). Berikut ini skrip pythonnya.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse2 = mean_squared_error(XOR_Y, Hasil_Prediksi_Keras)
rmse2 = sqrt(mean_squared_error(XOR_Y, Hasil_Prediksi_Keras))
print('MSE =',mse2)
print('RMSE =',rmse2)
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 5.7.

```
MSE = 0.1666667149924983
RMSE = 0.4082483496506732
```

GAMBAR 5.7.
Kesalahan hasil prediksi Keras

Sehingga jika dibandingkan, hasil kedua cara pemrograman *Deep Learning* menggunakan *library* TensorFlow dan Keras, dapat ditabelkan seperti pada Tabel 5.1.

TABEL 5.1.
Perbandingan hasil prediksi XOR

Jumlah iterasi (<i>epoch</i>)	Ukuran Kesalahan Prediksi	TensorFlow	Keras
100.000	MSE	0.00051	0.16667
100.000	RMSE	0.02255	0.40825

Dari hasil perbandingan prediksi XOR pada Tabel 5.1, terlihat hasil menggunakan TensorFlow lebih baik dibandingkan dengan menggunakan Keras, ditunjukkan dengan MSE dan RMSE yang lebih kecil. Dengan jumlah iterasi pelatihan yang sama.

5.3. Studi Kasus Prediksi Sederhana dengan Keras

Sesuai diagram alir penyelesaian prediksi menggunakan *Deep Learning* dengan Keras pada Gambar 5.2, sekarang akan kita coba untuk menyelesaikan kasus prediksi sederhana. Kasus prediksi sederhana yang dimaksud yaitu prediksi pasien apakah menderita penyakit diabetes atau tidak. Dalam contoh ini, digunakan *dataset* diabetes Pima Indian. Ini adalah *dataset* pembelajaran mesin standar dari repositori Pembelajaran Mesin *University of California* (UCI) (Brownlee, 2019). Ini menggambarkan data rekam medis pasien untuk orang India Pima dan apakah mereka memiliki diabetes dalam lima tahun. Dengan demikian, ini adalah masalah klasifikasi biner (terkena diabetes 1 atau tidak 0). Semua variabel masukan yang menggambarkan setiap pasien adalah numerik. Ini membuatnya lebih mudah digunakan secara langsung didalam jaringan saraf atau jaringan *Deep Learning* yang mengharapakan masukan numerik dan nilai-nilai keluaran.

Berikut ini tahapan-tahapan contoh penerapan *Deep Learning* menggunakan Keras untuk menyelesaikan sistem prediksi pasien apakah terkena diabetes atau tidak (Brownlee, 2019).

5.3.1. Penyiapan Data

Langkah pertama adalah mendefinisikan fungsi dan kelas yang akan digunakan. Digunakan *library* NumPy untuk memuat *dataset* dan digunakan dua kelas dari *library* Keras untuk mendefinisikan model yang digunakan. Impor yang diperlukan tercantum di bawah ini.

```
# Pendefinisian fungsi dan kelas yang digunakan
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

Dataset yang digunakan yaitu *dataset* diabetes Pima Indian (pima-indians-diabetes.csv). Dapat diunduh pada alamat berikut:

<https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv>

Setelah *dataset* diunduh dan disimpan dengan nama file: pima-indians-diabetes.csv, silahkan diletakkan di direktori kerja lokal di lokasi yang sama dengan file program python berada. Berikut ini adalah isi dari file pima-indians-diabetes.csv.

```
# Isi file pima-indians-diabetes.csv
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
...
```

Selanjutnya dapat dimuat file sebagai matriks angka menggunakan fungsi NumPy `loadtxt()`. Terdapat delapan variabel masukan dan satu variabel keluaran (kolom terakhir). Akan dipelajari sebuah model untuk memetakan baris variabel masukan (X) ke variabel keluaran (y), yang sering diringkas sebagai $y = f(X)$.

Variabel Masukan (X):

- Jumlah kali hamil
- Konsentrasi glukosa plasma 2 jam dalam tes toleransi glukosa oral
- Tekanan darah diastolik (mm Hg)
- Ketebalan lipatan kulit trisep (mm)
- Insulin serum 2 jam ($\mu\text{U} / \text{ml}$)
- Indeks massa tubuh (berat dalam kg / (tinggi dalam m) 2)
- Fungsi silsilah diabetes
- Umur (tahun)

Variabel Keluaran (y):

- Variabel kelas (0 atau 1)

Yang berarti 1 (terkena diabetes), 0 (tidak).

Setelah file CSV dimuat ke dalam memori, selanjutnya dapat dibagi kolom data menjadi variabel masukan dan keluaran. Data akan disimpan dalam larik 2D tempat dimensi pertama adalah baris dan dimensi kedua adalah kolom, mis. [baris, kolom]. Array dapat dibagi menjadi dua array dengan memilih subset kolom menggunakan operator *slice* NumPy standar atau ":" Kita dapat memilih 8 kolom pertama dari indeks 0 hingga indeks 7 melalui slice 0: 8. Kemudian dapat dipilih kolom keluaran (variabel ke-9) melalui indeks 8. Contoh perintahnya di Python sebagai berikut.

```
# Pemanggilan dataset yang digunakan
# memuat dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# dipecah menjadi variabel input (X) dan output (y)
X = dataset[:,0:8]
y = dataset[:,8]
...
```

Sekarang siap untuk didefinisikan model jaringan *Deep Learning* yang sesuai dengan *dataset* yang digunakan. Catatan, *dataset* memiliki 9 kolom dan kisaran 0:8 akan dipilih kolom dari 0 hingga 7, berhenti sebelum indeks 8.

5.3.2. Penentuan Model Keras

Model dalam Keras didefinisikan sebagai urutan lapisan (*sequence of layers*). Dibuat model sequensial dan ditambahkan lapisan satu per satu sampai diperoleh arsitektur jaringan yang sesuai. Hal pertama yang harus dilakukan adalah dipastikan lapisan masukan memiliki jumlah fitur masukan yang tepat. Ini dapat ditentukan saat dibuat lapisan pertama dengan argumen *input_dim* dan diatur sampe ke-8 untuk 8 variabel masukan. Sedangkan untuk penentuan lapisan dalam atau lapisan tersembunyi penentuan jumlah lapisannya bisa menggunakan cara heuristik atau melalui proses percobaan dan kesalahan percobaan berkali-kali.

Secara umum, diperlukan jaringan yang cukup besar untuk menangkap struktur masalah. Dalam contoh ini, digunakan struktur jaringan yang sepenuhnya terhubung dengan tiga lapisan. Lapisan yang terhubung sepenuhnya ditentukan menggunakan kelas *Dense*. Dapat ditentukan jumlah *neuron* atau simpul di lapisan seperti argumen pertama, dan penentuan fungsi aktivasi menggunakan argumen aktivasi.

Akan digunakan fungsi aktivasi unit linear yang diperbaiki yang disebut sebagai ReLU pada dua lapisan pertama dan fungsi Sigmoid di lapisan keluaran. Dulu fungsi aktivasi Sigmoid dan Tanh lebih disukai untuk semua lapisan. Saat ini, kinerja yang lebih baik dicapai menggunakan fungsi aktivasi ReLU. Pada contoh ini digunakan sigmoid pada lapisan keluaran untuk memastikan keluaran jaringannya antara 0 dan 1 dan mudah dipetakan ke probabilitas kelas 1 atau *snip* ke klasifikasi keras dari kedua kelas dengan ambang batas standar 0.5.

Model mengharapkan deretan data dengan 8 variabel (argumen *input_dim* = 8). Lapisan tersembunyi pertama memiliki 12 simpul dan menggunakan fungsi aktivasi ReLU. Lapisan tersembunyi kedua memiliki 8 simpul dan menggunakan fungsi aktivasi ReLU. Lapisan keluaran memiliki satu simpul dan menggunakan fungsi aktivasi sigmoid. Contoh pemrogramannya dengan python sebagai berikut:

```
# Pemodelan Keras
# Definisikan model keras
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
...
```

5.3.3. Kompilasi Model Keras

Setelah model didefinisikan, selanjutnya dilakukan kompilasi. Proses kompilasi model menggunakan perpustakaan numerik yang efisien di bawah selimut (yang disebut *backend*) seperti Theano atau TensorFlow. *Backend* secara otomatis memilih cara terbaik untuk mewakili jaringan untuk pelatihan dan membuat prediksi untuk berjalan pada perangkat keras, seperti CPU atau GPU atau bahkan perangkat keras terdistribusi.

Saat melakukan proses kompilasi, harus ditentukan beberapa properti tambahan yang diperlukan saat melatih jaringan. Yang perlu diingat dalam proses pelatihan jaringan, berarti akan ditemukan set bobot terbaik saat memetakan masukan ke keluaran dalam Data Latih yang digunakan. Harus ditentukan fungsi kerugian yang digunakan untuk evaluasi satu set bobot, *optimizer* digunakan untuk mencari melalui berbagai bobot jaringan dan metrik opsional apa pun yang ingin dikumpulkan dan dilaporkan selama pelatihan.

Dalam hal ini, bisa digunakan *cross entropy* sebagai argumen *loss*. Kerugian ini untuk masalah klasifikasi biner dan didefinisikan dalam Keras sebagai “*binary_crossentropy*”. Cara pemilihan fungsi kehilangan saat pelatihan *Deep Neural Networks* didefinisikan *optimizer* sebagai algoritma penurunan gradien stokastik efisien “adam”. Ini adalah penurunan gradien yang populer karena secara otomatis menyetel sendiri dan memberikan hasil yang baik dalam berbagai masalah. Contoh pemrograman kompilasi model Keras dengan python sebagai berikut:

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

5.3.4. Penyesuaian Model Keras

Setelah ditetapkan model dan dilakukan proses kompilasi selanjutnya siap untuk dilakukan perhitungan yang efisien. Saatnya dilakukan eksekusi model pada beberapa data. Dapat dilatih atau disesuaikan model terhadap data yang dimuat dengan memanggil fungsi *fit()* pada model. Proses pelatihan akan berjalan untuk sejumlah iterasi melalui Data Latih yang disebut *epochs*. Harus ditentukan menggunakan argumen *epochs*. Harus diatur jumlah baris Data Latih yang dipertimbangkan sebelum bobot model diperbarui dalam setiap *epoch*, yang disebut ukuran *batch* dan ditetapkan menggunakan argumen *batch_size*.

Untuk masalah ini, akan dijalankan sejumlah kecil *epoch*(150) dan digunakan ukuran batch yang relatif kecil yaitu 10. Konfigurasi ini dapat dipilih secara eksperimental dengan coba-coba. Pelatihan model cukup sehingga ia belajar pemetaan baris masukan data yang baik (atau cukup baik) ke klasifikasi keluaran. Model akan selalu memiliki beberapa kesalahan, tetapi jumlah kesalahan akan keluar setelah beberapa titik untuk konfigurasi model yang diberikan. Ini disebut model konvergensi. Contoh pemrograman penyesuaian model Keras dengan python sebagai berikut:

```
# Penyesuaian model Keras
# Menyesuaikan model keras pada dataset
model.fit(X, y, epochs=150, batch_size=10)
...
```


5.3.5. Evaluasi Model Keras

Setelah dilakukan pelatihan jaringan saraf (*Deep Learning*) pada seluruh Data Latih, selanjutnya dapat dilakukan evaluasi kinerja jaringan pada Data Latih yang sama. Ini hanya akan memberi gambaran tentang seberapa baik telah dimodelkan Data Latih yang digunakan. Tetapi tidak tahu seberapa baik algoritma dapat bekerja pada data yang baru. Biasanya *dataset* disiapkan ke dalam sekumpulan Data Latih dan Data Uji untuk pelatihan dan evaluasi model yang telah dirancang. Proses evaluasi model pada *dataset* saat proses pelatihan bisa digunakan fungsi *evaluate()* pada model dengan diberikan masukan dan keluaran yang sama dengan yang digunakan saat pelatihan model. Ini akan menghasilkan prediksi untuk setiap pasangan masukan dan keluaran dan mengumpulkan skor, termasuk kehilangan rata-rata dan metrik yang telah dikonfigurasi, seperti akurasi. Fungsi *evaluate()* akan mengembalikan daftar dengan dua nilai. Yang pertama adalah hilangnya model pada *dataset* dan yang kedua adalah akurasi model pada *dataset*. Contoh pemrograman evaluasi model Keras dengan python sebagai berikut:

```
# Evaluasi model keras
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

5.3.6. Pembuatan Prediksi

Setelah dilakukan proses pelatihan model, selanjutnya digunakan untuk prediksi pada data baru. Untuk melakukan proses prediksi tinggal dipanggil fungsi *predict()* pada model. Bisa digunakan fungsi aktivasi sigmoid pada lapisan keluaran, sehingga prediksi akan menjadi probabilitas dalam kisaran antara 0 dan 1. Agar keluarannya menjadi prediksi biner yang tajam maka tinggal dibulatkan. Contoh pemrograman prediksi *Deep Learning* menggunakan Keras dengan python sebagai berikut:

```
# Pembuatan prediksi menggunakan Deep Learning dengan Keras
# Pembuatan prediksi kelas dengan model
predictions = model.predict_classes(X)
# Meringkas 5 kasus pertama
for i in range(5):
    print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i],
    y[i]))
```

Untuk contoh prediksi pasien penderita diabetes ini, diperoleh hasil seperti Gambar 5.4.

```
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
```

GAMBAR 5.8.

Contoh hasil prediksi pasien penderita diabetes



Bab 6

Aplikasi Deep Learning Studi Kasus Prediksi Tingkat Pengangguran Terbuka

6.1. Pengukuran Kinerja Deep Learning

Dalam Bab 6 ini, akan dijelaskan tentang studi kasus prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia. Seperti umumnya sistem prediksi, untuk mengukur keberhasilan proses prediksi, kita membutuhkan metrik yang berhubungan dengan ukuran kinerja *Deep Learning*. Pengukuran kinerja keberhasilan Tahap Pelatihan maupun Tahap Pengujian *Deep Learning*, terutama untuk penyelesaian permasalahan sistem prediksi, biasanya paling tidak dinyatakan dalam:

- 1) *Mean Square Error* (MSE)
- 2) *Root Mean Square Error* (RMSE)
- 3) *Mean Absolute Percentage Error* (MAPE), dan
- 4) Persentase Keberhasilan.

Masing-masing dirumuskan sebagai berikut. Jika e_i menyatakan *error* atau selisih antara target atau nilai yang diinginkan dibandingkan dengan nilai hasil prediksi, dengan jumlah data sebanyak n data, maka:

Mean Square Error (MSE) dinyatakan dalam:

$$MSE = \frac{\sum_{i=1}^n e_i^2}{n} \quad (6.1)$$

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{MSE} \quad (6.2)$$

Mean Absolute Percentage Error (MAPE) dinyatakan dalam (Afiadi, 2015):

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{\text{Target}_i - \text{Prediksi}_i}{\text{Target}_i} \right| \times 100 \quad (6.3)$$

Persentase Keberhasilan:

$$\text{Persentase Keberhasilan} = (100 - \text{MAPE}) \% \quad (6.4)$$

6.2. Prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia

Sekarang kita akan mencoba menyelesaikan kasus prediksi yang lebih lengkap rancangan sistem dan implementasinya. Studi kasus yang diambil, yaitu prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia menggunakan *Deep Learning*.

Berdasarkan informasi dari Badan Pusat Statistik (BPS) Indonesia, yang beralamat di <https://sirusa.bps.go.id/sirusa/index.php/indikator/44>, Tingkat Pengangguran Terbuka (TPT) adalah persentase jumlah pengangguran terhadap jumlah angkatan kerja, yang dinyatakan dalam Persamaan (6.5):

$$TPT = \frac{a}{b} \times 100\% \quad (6.5)$$

dimana:

a = Jumlah Pengangguran.

b = Jumlah Angkatan Kerja.

Kegunaan TPT untuk mengindikasikan besarnya persentase angkatan kerja yang termasuk dalam pengangguran. Interpretasi TPT yang tinggi menunjukkan bahwa terdapat banyak angkatan kerja yang tidak terserap pada pasar kerja.

6.2.1. Penyiapan Struktur Model Data

Dataset yang digunakan untuk eksperimen sistem prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia, kami peroleh dan kami rekap secara manual dari Badan Pusat Statistik (BPS) Indonesia yang berada di alamat:

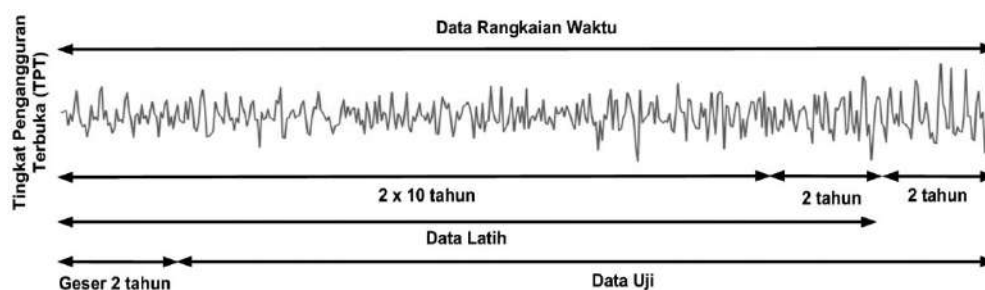
<https://www.bps.go.id/indicator/6/543/1/tingkat-pengangguran-terbuka-menurut-provinsi.html>

Dari data TPT asli yang kami peroleh dari BPS ini, terdapat beberapa perlakuan, sehubungan dengan ketersediaan data:

1. Dari data TPT mulai tahun 1986-2020, terdapat data yang tidak ada, yaitu pada tahun 1995. Sehingga data yang bisa digunakan adalah data TPT tahun 1996 s.d 2020. Karena jumlah datanya ganjil, untuk kemudahan format Data Latih dan Data Uji, diambil data TPT dari tahun 1997 s.d 2020.
2. Dari data TPT tahun 1997 s.d 2020, data TPT tahun 1997 s.d 2004 datanya adalah data tahunan. Sedangkan mulai tahun 2005 s.d 2020 tidak dimunculkan data tahunan, tetapi terdapat data TPT bulan februari dan agustus. Agar diperoleh format data yang sama, yaitu tahunan, maka dari data TPT bulan februari dan agustus diambil nilai rata-rata.

Dari data-data ini sebagian digunakan untuk Tahap Pelatihan dan sisanya untuk Tahap Pengujian prediksi. Data TPT yang digunakan untuk Tahap Pelatihan (Data Latih), yaitu data TPT mulai tahun 1997 sampai dengan tahun 2016 untuk memprediksi TPT tahun 2017 dan 2018. Sedangkan data TPT yang digunakan untuk Tahap Pengujian (Data Uji), tinggal digeser dua tahun kedepan, yaitu data TPT mulai tahun 1999 sampai dengan tahun 2018 untuk memprediksi TPT tahun 2019 dan 2020.

Dengan demikian, kebutuhan sistem prediksi ini, dirancang untuk memprediksi TPT dua tahun kedepan, berdasarkan data TPT dua puluh tahun sebelumnya. Jika masing-masing data TPT dua tahunan sepanjang tahun 1997 sampai dengan 2016 dikelompokkan, maka diperoleh sepuluh kelompok tahun. Dengan cara yang sama, untuk data TPT dua tahun yang diprediksi dijadikan satu kelompok prediksi. Rancangan ini selanjutnya dijadikan model sistem, yaitu sistem dengan sepuluh masukan dan satu keluaran. Sehingga struktur model Data Latih dan Data Uji, sesuai rancangan ini seperti diperlihatkan pada Gambar 6.1.



GAMBAR 6.1.
Struktur model Data Latih dan Data Uji Prediksi
Tingkat Pengangguran Terbuka (TPT)

6.2.2. Normalisasi Data

Data yang digunakan untuk sistem prediksi Tingkat Pengangguran Terbuka (TPT) sesuai Persamaan 6.5 dinyatakan dalam persen. Nilainya berkisar diantara 0 s.d 100. Sedangkan keluaran *Deep Learning* sesuai fungsi aktivasi yang digunakan. Dalam contoh buku ini, misalnya di simpul keluaran, menggunakan fungsi aktivasi Sigmoid. Maka sesuai dengan persamaan fungsi aktivasi Sigmoid pada Persamaan (2.6), keluaran dari simpul ini, adalah nilai diantara 0 s.d 1. Oleh karena itu, data TPT ini, perlu dilakukan normalisasi. Supaya bisa masuk kedalam jangkauan kemampuan *Deep Learning*. Nantinya jika akan dibandingkan dengan target atau nilai TPT yang diinginkan sesungguhnya, hasil prediksi keluaran dari *Deep Learning* harus dilakukan denormalisasi data terlebih dahulu. Untuk keperluan normalisasi data, umumnya digunakan Persamaan (6.6) berikut ini.

$$X' = \frac{0,8 (X - b)}{(a - b)} + 0,1 \quad (6.6)$$

Dimana:

X' = data hasil normalisasi.

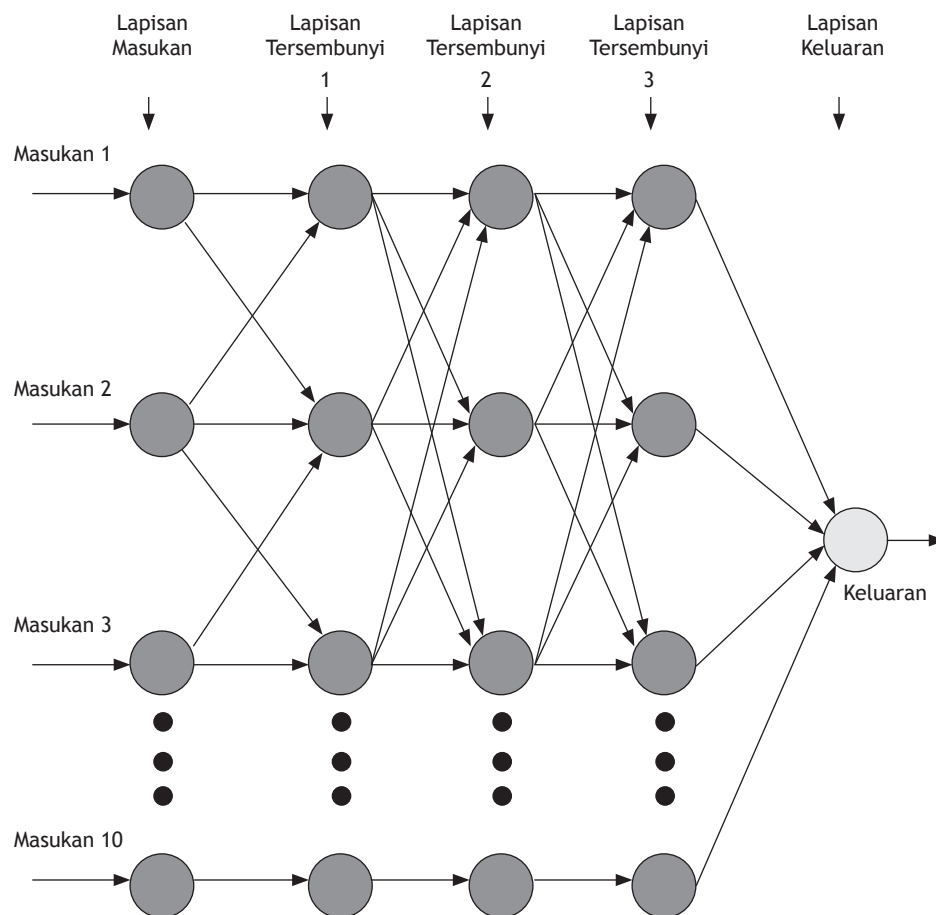
X = data asli/data awal.

a = nilai maksimum data asli.

b = nilai minimum data asli.

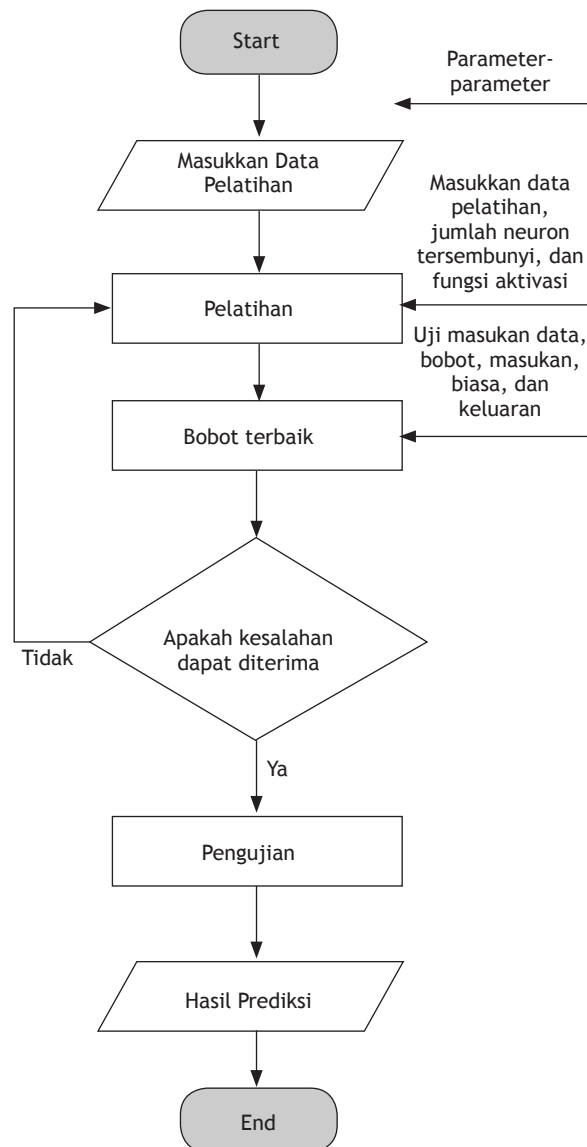
6.2.3. Perancangan Sistem

Sesuai dengan struktur model sistem prediksi TPT dengan sepuluh masukan dan satu keluaran, maka rancangan arsitektur sistem *Deep learning* yang sesuai dengan permasalahan ini untuk contoh buku ini diperlihatkan pada Gambar 6.2.



GAMBAR 6.2.
Arsitektur Deep Learning

Dari Gambar 6.2, simpul di lapisan masukan dan lapisan tersembunyi dirancang untuk menggunakan fungsi aktivasi ReLU, sedangkan simpul di lapisan keluaran dirancang menggunakan fungsi aktivasi Sigmoid. Dengan menggunakan rancangan arsitektur jaringan *Deep Learning* sesuai pada Gambar 6.2, maka selanjutnya proses pelatihan dan pengujian untuk *Deep Learning* ini dirancang seperti yang ditunjukkan pada Gambar 6.3.



GAMBAR 6.3.
Diagram alir proses pelatihan dan pengujian Deep Learning

Sesuai dengan diagram alir proses pelatihan dan pengujian *Deep Learning* pada Gambar 6.3, Jaringan *Deep Learning* dilatih menggunakan Data Latih dalam bentuk pasangan data masukan dan keluaran yang diinginkan. Dalam proses pelatihan, diolah dengan bobot dan bias menggunakan fungsi aktivasi yang sesuai. Kemudian keluaran jaringan *Deep Learning* dibandingkan dengan target atau keluaran yang diinginkan, hingga kesalahan bisa diterima. Selanjutnya bobot terbaik hasil proses pelatihan, digunakan untuk proses pengujian. Dalam proses pengujian, data baru diberikan, kemudian diproses pada setiap lapisan sampai lapisan keluaran, sehingga diperoleh keluaran jaringan berupa prediksi TPT. Selanjutnya hasil penilaian kinerja dalam proses prediksi TPT dinyatakan dalam bentuk *Mean Square Error* (MSE), *Root Mean Square Error* (RMSE), *Mean Absolute Percentage Error* (MAPE), dan Persentase Keberhasilan (%).

6.2.4. Implementasi Sistem

a. Tahap Pelatihan

Implementasi sistem untuk studi kasus prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia menggunakan Python dengan *Deep Learning* ini, untuk Tahap Pelatihan, direalisasikan sesuai urutan langkah-langkah berikut ini.

- 1) Penyiapan Data.
- 2) Penentuan Model Keras.
- 3) Kompilasi Model Keras.
- 4) Penyesuaian Model Keras.
- 5) Evaluasi Model Keras.
- 6) Pembuatan prediksi.

Masing-masing dijabarkan sebagai berikut.

1) **Penyiapan Data**

Langkah pertama sebelum penyiapan Data Latih, adalah mendefinisikan fungsi dan kelas yang akan digunakan. Digunakan *library* NumPy untuk memuat Data Latih dan digunakan *library* Keras untuk mendefinisikan model yang digunakan. Impor yang diperlukan tercantum di bawah ini.

```
# Pendefinisian fungsi dan kelas yang digunakan
# Import library keras dan lain-lain

import numpy as np # For matrix math
from numpy import loadtxt
from keras.models import Sequential
import matplotlib.pyplot as plt # For plotting
import keras
from keras.utils import to_categorical
from keras.layers import Flatten, Dense, Activation
from IPython.display import clear_output
```

Selanjutnya siapkan fungsi untuk menghitung data hasil normalisasi. Fungsi normalisasi bisa dibuat seperti program berikut ini.

```
# Fungsi normalisasi

def normalisasi(x):
    a = np.max(x)
    b = np.min(x)
    for i in x:
        hasil_normalisasi = (0.8 * (x - b))/(a-b) + 0.1
    return hasil_normalisasi
```

Selanjutnya panggil Data Latih dan Data Uji. Programnya sebagai berikut.

```
# Panggil Data Latih
# Data Tingkat Pengangguran Terbuka (TPT) mulai Tahun 1997
# sampai dengan Tahun 2020

TPT = loadtxt('data_pengangguran.csv', delimiter=',')
TPT_normal = normalisasi(TPT)

TPT_Target = TPT[:,10]
TPT_Target_Uji = TPT[:,11]
```

```

# DATA LATIH =====
# Data dibagi menjadi 10 variabel input (X) dan 1 variabel
output (Y)
# Input: mulai tahun 1997 sampai tahun 2016

X = TPT_normal[:,0:10]

# Output: data TPT tahun 2017 dan 2018

Y = TPT_normal[:,10]

# DATA UJI PREDIKSI=====
# Geser dua tahun ke depan
# Input: mulai tahun 1999 sampai tahun 2018
# Untuk memprediksi TPT Tahun 2019 dan 2020

X1 = TPT_normal[:,1:11]

# Urut tahun, menggantikan indeks dari 0 s.d 1
urut = [1,
        2
        ]

```

Misalkan ingin ditampilkan Data Latih dan Data Ujinya, tinggal ketikkan:

```
print(TPT)
```

Hasilnya seperti terlihat pada Gambar 6.4.

```

[[ 4.69  6.36  8.1   9.67 10.75  9.43  8.01  7.22  6.03  5.99  5.42  5.11]
 [ 5.46  6.08  9.06  9.86 10.36  8.43  7.28  6.25  5.82  5.56  5.2   6.01]]

```

GAMBAR 6.4.
Data Latih dan Data Uji

Dengan cara yang sama, misalkan ingin ditampilkan Data Latih dan Data Uji setelah dilakukan normalisasi, tinggal ketikkan:

```
TPT_normal = normalisasi(TPT)
print(TPT_normal)
```

Hasilnya seperti terlihat pada Gambar 6.5.

```
[[0.1          0.32046205 0.55016502 0.75742574 0.9          0.72574257
  0.53828383 0.4339934  0.27689769 0.27161716 0.19636964 0.15544554]
 [0.20165017 0.28349835 0.67689769 0.78250825 0.84851485 0.59372937
  0.44191419 0.30594059 0.24917492 0.21485149 0.16732673 0.27425743]]
```

GAMBAR 6.5.
Data Latih dan Data Uji setelah dinormalisasi

Selanjutnya jika ingin dilihat dimensi matriksnya, tinggal ketikkan:

```
baris, kolom = TPT.shape
print("barisnya =",baris)
print("kolomnya =",kolom)
```

Hasilnya seperti terlihat pada Gambar 6.6.

```
barisnya = 2
kolomnya = 12
```

GAMBAR 6.6.
Dimensi matriks Data Latih dan Data Uji

Dengan cara yang sama, untuk lihat dimensi matriks setelah dinormalisasi, tinggal ketikkan:

```
baris, kolom = TPT_normal.shape
print("barisnya =", baris)
print("kolomnya =", kolom)
```

Hasilnya seperti terlihat pada Gambar 6.7.

```
barisnya = 2
kolomnya = 12
```

GAMBAR 6.7.

Dimensi matriks Data Latih dan Data Uji setelah dinormalisasi

Dengan cara yang sama, untuk lihat dimensi matriks data masukan pelatihan, yang diambilkan dari data TPT setelah dinormalisasi, tinggal ketikkan:

```
baris, kolom = X.shape
print("barisnya =", baris)
print("kolomnya =", kolom)
```

Jika dijalankan, maka hasilnya seperti ditampilkan pada Gambar 6.8.

```
barisnya = 2
kolomnya = 10
```

GAMBAR 6.8.

Dimensi matriks data masukan pelatihan

Terlihat ukuran dimensinya, 2x10, seperti terlihat pada Gambar 6.8.

2) Penentuan Model Keras

Langkah kedua adalah penentuan model Keras. Model Keras didefinisikan sebagai urutan lapisan (*sequence of layers*). Dibuat model sequensial dan ditambahkan lapisan satu per satu sampai diperoleh arsitektur jaringan yang sesuai. Arsitektur *Deep Learning* yang digunakan, yaitu sistem dengan 10 masukan dan 1 keluaran. Terdapat 1 lapisan masukan, 3 lapisan tersembunyi dan 1 lapisan keluaran. Masing-masing dengan jumlah simpul 10, 10, 10, 10 dan 1 seperti terlihat pada Gambar 6.2. Simpul pada lapisan masukan dan lapisan tersembunyi, dirancang menggunakan fungsi aktivasi ReLU, sedangkan simpul pada lapisan keluaran dirancang menggunakan fungsi aktivasi Sigmoid. Implementasi kodingnya seperti berikut ini.

```
# Mendefinisikan model keras

model = Sequential()
model.add(Dense(10, input_dim=10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

3) Kompilasi Model Keras

Langkah ketiga setelah model didefinisikan, selanjutnya dilakukan kompilasi. Proses kompilasi model menggunakan perpustakaan numerik yang efisien di bawah selimut (yang disebut *backend*) seperti Theano atau TensorFlow. Proses kompilasi adalah proses pelatihan jaringan. Proses pelatihan yaitu proses menemukan set bobot terbaik saat memetakan masukan ke keluaran dalam Data Latih yang digunakan. Fungsi kerugian yang digunakan untuk evaluasi satu set bobot, *optimizer* digunakan untuk mencari melalui berbagai bobot jaringan dan metrik opsional yang sesuai, Dalam hal ini digunakan *Mean Squared Error* (MSE) sebagai argumen *loss*. Implementasi programnya seperti berikut ini.

```
# Proses kompilasi (pelatihan) Deep Learning

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

4) Penyesuaian Model Keras

Langkah selanjutnya adalah penyesuaian atau pencarian model terbaik dari Keras. Penyesuaian model Keras menggunakan fungsi *fit()* pada model. Proses pelatihan akan berjalan untuk sejumlah iterasi melalui Data Latih yang disebut *epochs*. Sebelumnya disiapkan kelas untuk pengeplotan grafik penurunan *error (loss)* berupa *Mean Squared Error (MSE)* selama proses pelatihan *Deep Learning*. Implementasi programnya seperti berikut ini.

```
# Kelas untuk pengeplotan grafik penurunan error (loss)

class PlotLosses(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []
        self.fig = plt.figure()
        self.logs = []

    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.i += 1
```



```
clear_output(wait=True)
plt.plot(self.x, self.losses, label="loss")
plt.plot(self.x, self.val_losses, label="val_loss")
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show();

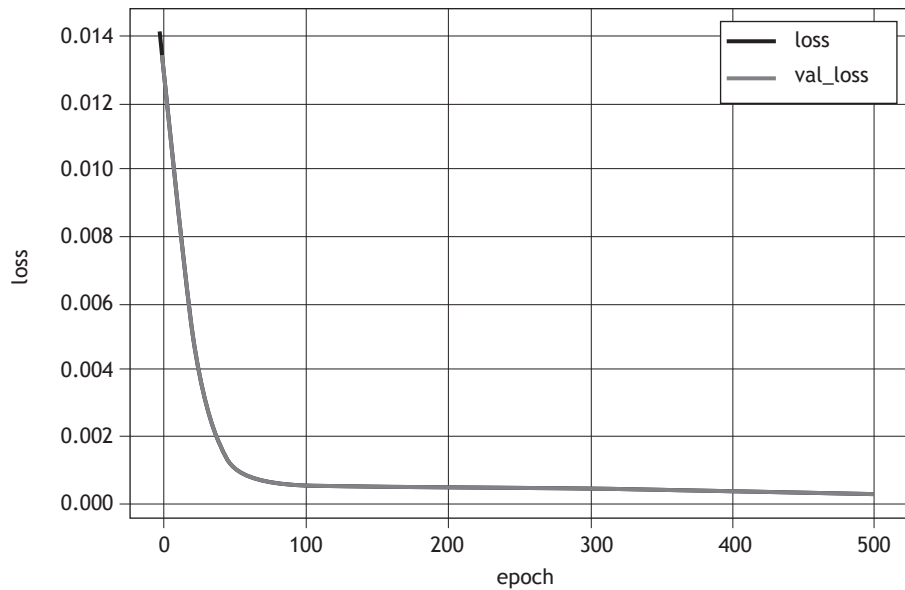
plot_losses = PlotLosses()
```

Kemudian penyesuaian atau pencarian model terbaik dari Keras, menggunakan fungsi *fit()* pada model, agar proses pelatihan berjalan untuk sejumlah iterasi, implementasi programnya seperti berikut ini.

```
# Penyesuaian atau pencarian model terbaik dari Keras

model.fit(X, Y,
          epochs=500,
          validation_data=(X, Y),
          callbacks=[plot_losses],
          verbose=0)
```

Jika dijalankan, dengan jumlah iterasi 500 *epoch*, contoh grafik hasil penurunan *loss* berupa MSE selama pelatihan, hasilnya seperti terlihat pada Gambar 6.9.



GAMBAR 6.9.
Penurunan loss berupa MSE selama proses pelatihan

5) Evaluasi Model Keras

Setelah dilakukan pelatihan jaringan *Deep Learning* pada seluruh Data Latih, selanjutnya dapat dilakukan evaluasi kinerja jaringan pada Data Latih yang sama. Ini hanya akan memberi gambaran tentang seberapa baik telah dimodelkan Data Latih yang digunakan. Implementasi programnya seperti berikut ini.

```
# Evaluate model Keras

_, MSE = model.evaluate(X, Y, batch_size=1)
print('MSE: %.2f' % (MSE))
```

Hasilnya seperti terlihat pada Gambar 6.10.

```
2/2 [=====] - 0s 4ms/step - loss: 2.4841e-04 - mean_squared_error: 2.4841e-04  
MSE : 0.00
```

GAMBAR 6.10.
MSE hasil proses pelatihan

6) Pembuatan prediksi

Langkah terakhir setelah diperoleh hasil pelatihan terbaik, selanjutnya jaringan *Deep Learning* digunakan untuk memprediksi. Proses prediksi dengan cara diberikan masukan data baru. Untuk melakukan proses prediksi tinggal dipanggil fungsi *predict()* pada model.

```
# Menjalankan fungsi prediksi  
  
y_pred = model.predict(X)  
print(y_pred)
```

Contoh, jika diberikan masukan yang sama seperti saat pelatihan, hasilnya seperti terlihat pada Gambar 6.11.

```
[[0.18087065]  
 [0.1833452 ]]
```

GAMBAR 6.11.
Contoh hasil prediksi Deep Learning Tahap Pelatihan

Selanjutnya dari hasil prediksi *Deep Learning*, akan dibandingkan dengan target. Data target yang digunakan, adalah data TPT yang sesungguhnya. Yaitu data TPT tahun 2017 dan 2018. Maka keluaran hasil prediksi jaringan *Deep Learning* terlebih dahulu harus dilakukan denormalisasi. Program denormalisasi seperti diperlihatkan pada program berikut.

```

# Denormalisasi data sebelum dibandingkan

a = np.max(TPT)
b = np.min(TPT)

# Rumus
# x_denormalisasi = (((x_normalisasi - 0.1)*(a - b))/0.8) +
b

y_prediksi = (((y_pred - 0.1)*(a - b))/0.8) + b

print("y_prediksi =", y_prediksi)

```

Contoh hasilnya seperti terlihat pada Gambar 6.12.

```

y_prediksi = [[5.302595]
               [5.32134 ]]

```

GAMBAR 6.12.
Hasil prediksi Deep Learning setelah denormalisasi

Selanjutnya akan diplot hasil prediksi jaringan *Deep Learning*, dan dibandingkan hasilnya dengan target. Sebelumnya, urutan pengeplotan disesuaikan dengan tahun yang sesuai. Programnya sebagai berikut.

```

# Urut tahun, menggantikan indeks dari 0 s.d 1
urutan1 = ['2017',
           '2018'
          ]

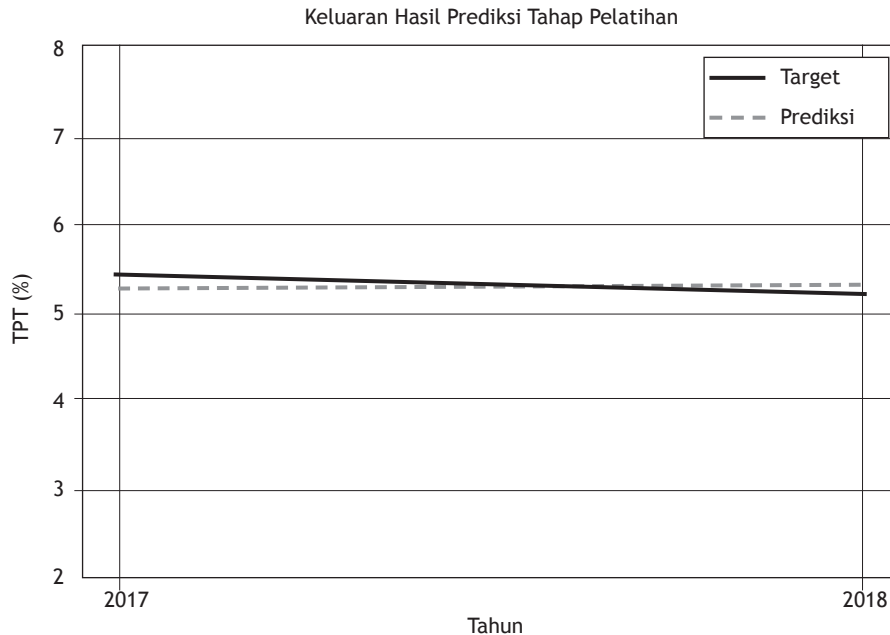
```

Sehingga program untuk menampilkan keluaran hasil prediksi Tahap Pelatihan, sebagai berikut.

```
# Plot keluaran hasil prediksi Tahap Pelatihan

plt.title('Keluaran Hasil Prediksi Tahap Pelatihan')
plt.plot(urutan1, TPT_Target, '-b',
label='Target',linewidth=3, markersize=12)
plt.plot(urutan1, y_prediksi, '--r', label='Prediksi',
linewidth=3, markersize=12)
plt.legend();
plt.xlabel('Tahun')
plt.ylabel('TPT (%)')
plt.ylim((2,8))
plt.grid()
```

Hasilnya seperti terlihat pada Gambar 6.13.



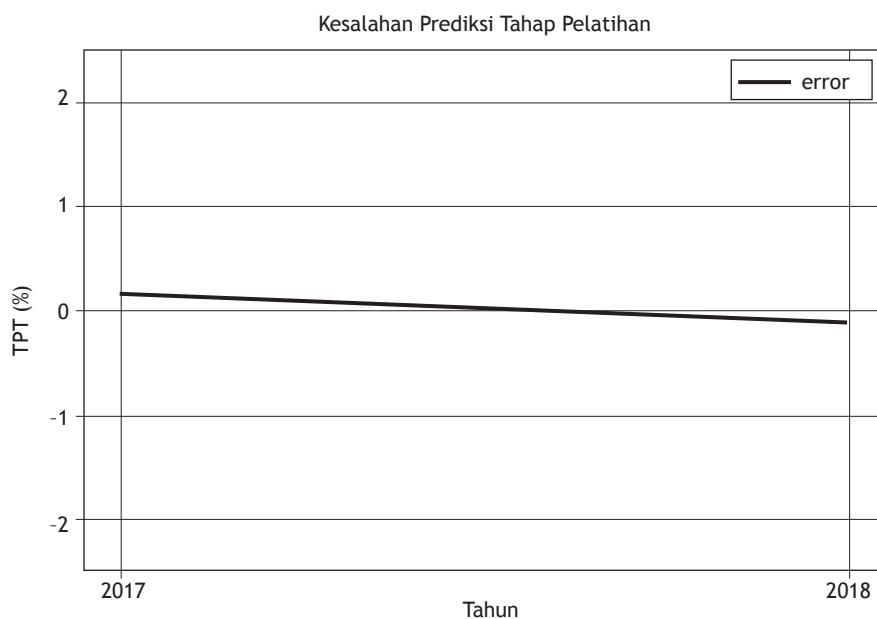
GAMBAR 6.13.
Keluaran hasil prediksi Tahap Pelatihan

Dan untuk menampilkan grafik kesalahan prediksi Tahap Pelatihan, programnya sebagai berikut.

```
# Plot kesalahan prediksi Tahap Pelatihan

selisih = TPT_Target - y_prediksi.transpose()
plt.title('Kesalahan Prediksi Tahap Pelatihan')
plt.plot(urutan1, selisih.transpose(), '-m', label='error',
linewidth=3, markersize=12)
plt.legend();
plt.xlabel('Tahun')
plt.ylabel('TPT (%)')
plt.ylim((-2.5,2.5))
plt.grid()
```

Hasilnya seperti terlihat pada Gambar 6.14.



GAMBAR 6.14.
Kesalahan prediksi Tahap Pelatihan

Selanjutnya untuk mengukur kinerja *Deep Learning* Tahap Pelatihan berupa kesalahan prediksi dalam MSE, RMSE, dan MAPE, serta Persentase keberhasilan prediksi, programnya sebagai berikut.

```
# Pengukuran kinerja Deep Learning

from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(TPT_Target, y_prediksi)
rmse = sqrt(mse)
print("mse =",mse)
print("rmse =",rmse)

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

MAPE = mean_absolute_percentage_error(TPT_Target, y_
prediksi)
print("MAPE =",MAPE)

Persentase_keberhasilan = 100-MAPE
print("Persentase_keberhasilan =", Persentase_keberhasilan)
```

Hasilnya seperti terlihat pada Gambar 6.15.

```
mse = 0.014253658747846353
rmse = 0.11938868768793111
MAPE = 2.0732203989654225
Persentase_keberhasilan = 97.92677960103458
```

GAMBAR 6.15.
Hasil kinerja Deep Learning Tahap Pelatihan

b. Tahap Pengujian

Selanjutnya untuk Tahap Pengujian, akan dilakukan sesuai dengan Data Uji. Berdasarkan data TPT mulai tahun 1999 sampai dengan tahun 2018, akan digunakan untuk memprediksi TPT tahun 2019 dan 2020. Langkah-langkah pemrogramannya, sebagai berikut.

Siapkan urutan tahun pengeplotan, menggantikan indeks 0 s.d 1.

```
# Urut tahun, menggantikan indeks dari 0 s.d 1
urutan2 = ['2019',
           '2020'
          ]
```

Selanjutnya, untuk memprediksi TPT tahun 2019 dan 2020, programnya cukup dituliskan seperti baris kode program berikut ini (sebelum denormalisasi).

```
# Ujicoba prediksi
y_uji = model.predict(X1)
print('Hasil prediksi sebelum denormalisasi =', y_uji)
```

Hasilnya seperti terlihat pada Gambar 6.16.

```
Hasil prediksi sebelum denormalisasi = [[0.1748051]
[0.1862019]]
```

GAMBAR 6.16.
Hasil prediksi sebelum denormalisasi

Selanjutnya dilakukan denormalisasi, programnya sebagai berikut.

```
# Denormalisasi data sebelum dibandingkan

a = np.max(TPT)
b = np.min(TPT)

# Rumus
# x_denormalisasi = (((x_normalisasi - 0.1)*(a - b))/0.8) + b

y_pengujian = (((y_uji - 0.1)*(a - b))/0.8) + b

print("y_pengujian =", y_pengujian)
```

Hasilnya seperti terlihat pada Gambar 6.17.

```
y_pengujian = [[5.2566485]
               [5.3429794]]
```

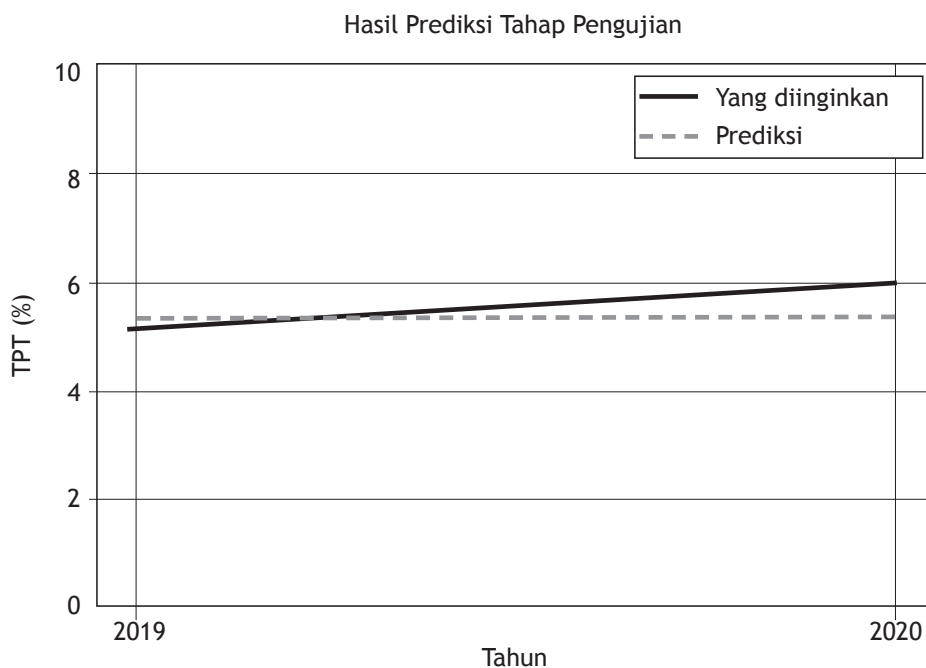
GAMBAR 6.17.
Hasil prediksi setelah denormalisasi

Selanjutnya sudah bisa dilakukan pengeplotan grafik hasil prediksi Tahap Pengujian. Programnya sebagai berikut.

```
# Plot keluaran hasil prediksi Tahap Pengujian

plt.title('Hasil Prediksi Tahap Pengujian')
plt.plot(urutan2, TPT_Target_Uji, '-b', label='Yang
diinginkan',linewidth=3, markersize=12)
plt.plot(urutan2, y_pengujian, '--r', label='Prediksi',
linewidth=3, markersize=12)
plt.legend();
plt.xlabel('Tahun')
plt.ylabel('TPT (%)')
plt.ylim((0,10))
plt.grid()
```

Hasilnya seperti terlihat pada Gambar 6.18.



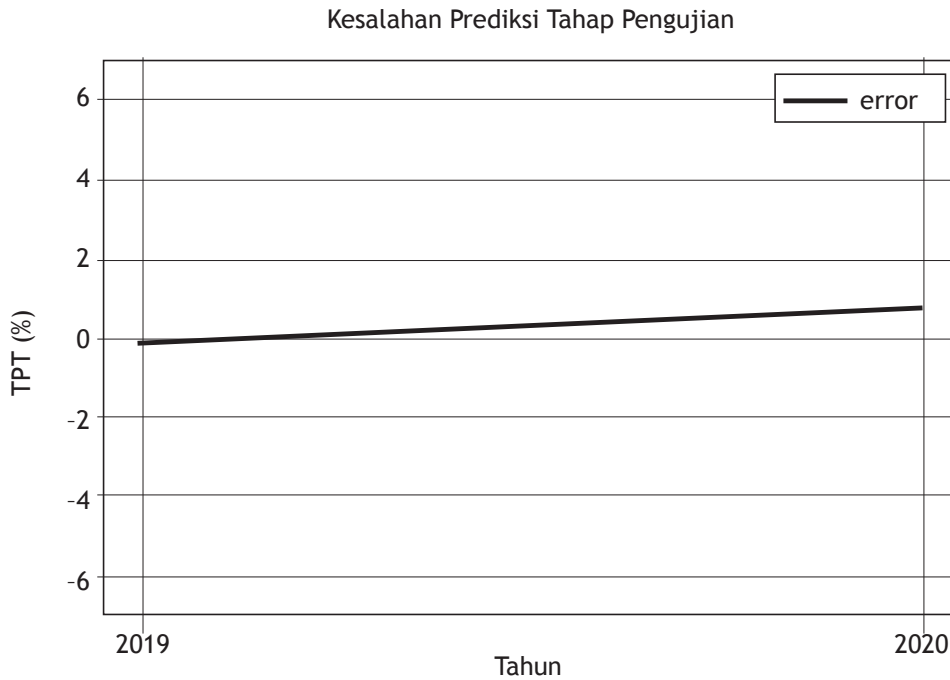
GAMBAR 6.18.
Hasil Prediksi Tahap Pengujian

Dan untuk pengeplotan kesalahan prediksi Tahap Pengujian, programnya sebagai berikut.

```
# Plot kesalahan prediksi Tahap Pengujian

selisih_pengujian = TPT_Target_Uji - y_pengujian.transpose()
plt.title('Kesalahan Prediksi Tahap Pengujian')
plt.plot(urutan2, selisih_pengujian.transpose(), '-m',
label='error', linewidth=3, markersize=12)
plt.legend();
plt.xlabel('Tahun')
plt.ylabel('TPT (%)')
plt.ylim((-7,7))
plt.grid()
```

Hasilnya seperti terlihat pada Gambar 6.19.



GAMBAR 6.19.
Kesalahan prediksi Tahap Pengujian

Selanjutnya langkah terakhir, untuk mengukur kinerja *Deep Learning* Tahap Pengujian berupa kesalahan prediksi dalam MSE, RMSE, dan MAPE, serta Persentase keberhasilan prediksi, programnya sebagai berikut.

```
# Pengukuran kinerja Deep Learning Tahap Pengujian

from sklearn.metrics import mean_squared_error
from math import sqrt
mse1 = mean_squared_error(TPT_Target_Uji, y_pengujian)
rmse1 = sqrt(mse1)
print("mse =",mse1)
print("rmse =",rmse1)
```

```

MAPE = mean_absolute_percentage_error(TPT_Target_Uji, y_
penguji)
print("MAPE =",MAPE)

Persentase_keberhasilan = 100-MAPE
print("Persentase_keberhasilan =", Persentase_keberhasilan)

```

Hasilnya seperti terlihat pada Gambar 6.20.

```

mse = 0.23321111684784993
rmse = 0.48291936888868925
MAPE = 7.765649207032838
Persentase_keberhasilan = 92.23435079296716

```

GAMBAR 6.20.

Hasil kinerja Deep Learning Tahap Pengujian

Secara lengkap, masing-masing hasil eksperimen sistem prediksi Tingkat Pengangguran Terbuka (TPT) di Indonesia, Tahap Pelatihan dan Tahap Pengujian disajikan pada Tabel 6.1.

TABEL 6.1.
Hasil eksperimen sistem Prediksi
Tingkat Pengangguran Terbuka (TPT) di Indonesia

Tahap	MSE	RMSE	MAPE	Persentase Keberhasilan (%)
Pelatihan	0.01425	0.11939	2.07322	97.9268
Pengujian	0.23321	0.48292	7.76565	92.23435



Bab 7

Penutup

7.1. Kesimpulan

Telah diuraikan dalam buku ini bagaimana melakukan teknik pemrograman *Deep Learning* dengan Python dan menerapkannya untuk berbagai macam permasalahan.

7.2. Ucapan Terimakasih

Terimakasih kepada Kementerian Pendidikan dan Kebudayaan Universitas Pembangunan Nasional “Veteran” Jawa Timur yang telah memberikan dana penelitian untuk penerbitan buku ini sesuai dengan Surat Perjanjian Penugasan dalam Rangka Pelaksanaan Program Penelitian Internal Batch II Skim Peningkatan Mutu dan Pembelajaran UPN “Veteran” Jawa Timur Tahun Anggaran 2020 Nomor: SPP/ 96 /UN.63.8/LT/VII/2020.



Daftar Pustaka

- Adam Gibson, J. P. (2017) 'Deep Learning', *Deep Learning*. O'Reilly Media, Inc. Available at: <https://learning.oreilly.com/library/view/deep-learning/9781491924570/>.
- Afiadi, F. (2015) 'Model Estimasi Variasi Spasial Seismisitas Sebagai Prekursor Gempa Bumi Kuat Menggunakan Extreme Learning Machine (ELM)', in *TESIS Program Studi Instrumentasi dan Kontrol*. ITB.
- Alavala, C. R. (2007) *Fuzzy Logic and Neural Networks Basic Concepts & Applications*. New Age International Publishers.
- Brownlee, J. (2019) 'Your First Deep Learning Project in Python with Keras Step-By-Step', *Machine Learning Mastery*.
- Bruce, P. and Bruce, A. (2017) 'Practical Statistics for Data Scientists'. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
- Ferdiansyah, F. F. (2020) 'Klasifikasi dan Pengenalan Objek Ikan Menggunakan Algoritma Support Vector Machine (SVM)', in *SKRIPSI Program Studi Informatika*. UPN.

- Hinton, G. E., Osindero, S. and Teh, Y.-W. (2006) 'A Fast Learning Algorithm for Deep Belief Nets', *Neural Comput.* Cambridge, MA, USA: MIT Press, 18(7), pp. 1527–1554. doi: 10.1162/neco.2006.18.7.1527.
- Ketkar, N. (2017) 'Deep Learning with Python: A Hands-on Introduction'. Apress.
- Kharkovyna, O. (2019) 'Top 10 Best Deep Learning Frameworks in 2019'. Towards Data Science. Available at: <https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de>.
- Kim, P. (2017) *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*. Apress.
- Meng, T. et al. (2020) 'A survey on machine learning for data fusion', *Information Fusion*, 57, pp. 115–129. doi: <https://doi.org/10.1016/j.inffus.2019.12.001>.
- Oppermann, A. (2019) 'Artificial Intelligence vs. Machine Learning vs. Deep Learning', in. DeepLearning Academy. Available at: <https://www.deeplearning-academy.com/p/ai-wiki-machine-learning-vs-deep-learning>.
- Pattanayak, S. (2017) 'Pro Deep Learning with TensorFlow : A Mathematical Approach to Advanced Artificial Intelligence in Python'. Apress.
- Rahmat, B. and Nugroho, B. (2019) 'Pemrograman Fuzzy dan Jaringan Syaraf Tiruan Untuk Sistem Kendali Cerdas Dilengkapi dengan Contoh Pemrograman Python'. Indomedia Pustaka.
- Raina, R., Madhavan, A. and Ng, A. Y. (2009) 'Large-Scale Deep Unsupervised Learning Using Graphics Processors', in *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery (ICML '09), pp. 873–880. doi: 10.1145/1553374.1553486.
- Rong, G. et al. (2020) 'Artificial Intelligence in Healthcare: Review and Prediction Case Studies', *Engineering*. doi: <https://doi.org/10.1016/j.eng.2019.08.015>.
- de Sousa, W. G. et al. (2019) 'How and where is artificial intelligence in the public sector going? A literature review and research agenda', *Government Information Quarterly*, 36(4), p. 101392. doi: <https://doi.org/10.1016/j.giq.2019.07.004>.