BAB I
PENDAHULUAN

Bab pendahuluan ini membahas mengenai latar belakang masalah, rumusan masalah, tujuan dan manfaat, ruang lingkup, serta sistematika penulisan yang akan digunakan dalam dokumen skripsi ini.

1.1 Latar Belakang

Kanker kulit merupakan bentuk kanker yang umum dan berpotensi mematikan, terutama pada populasi berkulit terang (Saladi & Persaud, 2005). Ini terjadi ketika sel-sel abnormal tumbuh di kulit, seringkali terkena paparan radiasi ultraviolet dari matahari (Saranya, 2020). Kanker kulit memiliki berbagai jenis pengklasifikasian, namun kanker kulit dapat diklasifikasikan berdasarkan jenis *benign* dan *malignant.*

Sebagian besar kanker kulit bersifat jinak (*benign*). Namun, penting untuk membedakannya dari kanker kulit ganas (*malignant*), yang biasanya ditandai dengan bentuk asimetris, batas yang tidak teratur, perubahan warna, atau diameter lebih dari 6 mm (Perkins & Duffy, 2015). Di sisi lain, jumlah kanker kulit umum pada kulit memiliki keterkaitan dengan risiko peningkatan pengembangan kanker kulit *malignant*, dengan perkiraan 50% kanker kulit *malignant* berpotensi berasal dari kanker kulit *benign*, terutama pada individu muda dan variasi tipe penyebaran (Plasmeijer dkk., 2017). Maka dari itu, deteksi dini *malignant* sangat penting karena sifatnya yang agresif dan resistensi terhadap pengobatan pada stadium lanjut (Jerant dkk., 2000).

Pendekatan *machine learning* telah banyak digunakan dalam klasifikasi kanker kulit, dengan beragam teknik yang menunjukkan potensi dalam mendeteksi kanker kulit. Salah satu pendekatan yang digunakan adalah K-*Nearest Neighbors* (k-NN) mampu mencapai akurasi 75% dalam mengidentifikasi kanker kulit berdasarkan fitur geometris yang relevan, seperti diameter dan bentuk (Savera dkk., 2020). Selain itu, metode *Random Forest* juga telah diaplikasikan dalam penelitian klasifikasi kanker kulit. Dalam penelitian oleh Ashari dkk. (2024) *Random Forest* mencapai akurasi 81% dalam mengidentifikasi fitur penting dari *dataset* yang kompleks. Meskipun *machine learning* memiliki keunggulan dalam hal kecepatan pelatihan, keterbatasan dalam kemampuan mengekstrak fitur secara otomatis

membuat pendekatan ini sering kali kurang efektif dibandingkan dengan teknik yang lebih modern.

Di sisi lain, pendekatan *deep learning*, khususnya *Convolutional Neural Networks* (CNN), telah menunjukkan peningkatan yang signifikan dalam akurasi klasifikasi kanker kulit. Terdapat penelitian mencatat bahwa CNN dapat mencapai akurasi hingga 88,89% dalam mendeteksi kanker kulit (Rupa dkk., 2024). Selain itu, arsitektur VGG-16 menunjukkan akurasi 87,6% dengan penggunaan *data augmentation* untuk menigkatkan akurasi klasifikasi kanker kulit (Malo dkk., 2022). Sementara itu, arsitektur AlexNet yang lebih sederhana dengan akurasi sekitar 91,40% (Yilmaz & Trocan, 2020a). Dalam penelitian lain yang mengevaluasi penggunaan ResNet50 dengan *optimizer* Adam untuk klasifikasi citra kanker kulit, model ini mencapai akurasi 88% (Firasari dkk., 2024). Terdapat pengembangan lanjutan dari arsitektur ResNet50, yaitu ResNet50V2. Penelitian terhadap ResNet50V2 mencatatkan akurasi sebesar 91,85%, yang menunjukkan performa yang sangat baik dalam membedakan *malignant* dan *benign* (Kreouzi dkk., 2024).

Meskipun ResNet50V2 telah terbukti efektif dalam klasifikasi berbasis citra medis, model ini perlu disesuaikan lebih lanjut untuk tugas spesifik. Oleh karena itu, diperlukan *fine-tuning* untuk menyesuaikan model dengan karakteristik khusus dari *dataset* kanker kulit. Penelitian oleh Yosinski dkk. (2014) menunjukkan bahwa *fine-tuning* memungkinkan model *deep learning* untuk beradaptasi dengan lebih baik pada tugas-tugas yang lebih spesifik tanpa memulai pelatihan dari awal. Seperti yang ditunjukkan oleh Pillai dkk. (2024), yang berhasil mencapai akurasi 93,33% dalam klasifikasi kanker kulit setelah melakukan *fine-tuning* pada model dengan arsitektur ResNet50.

Berdasarkan uraian di atas, penelitian ini bertujuan untuk mengembangkan model klasifikasi kanker kulit yang efektif untuk membedakan antara *benign* dan *malignant* menggunakan CNN dengan arsitektur ResNet50V2. Arsitektur ResNet50V2 mampu mengekstrak fitur secara otomatis dari citra. Dalam penelitian ini, dioptimalkan dengan pendekatan *fine-tuning. Fine-tuning* merupakan teknik *transfer learning* yang memanfaatkan model ResNet50V2 yang telah dilatih pada *dataset* besar seperti ImageNet dan kemudian disesuaikan lebih lanjut dengan *dataset* yang lebih spesifik, yaitu *dataset* kanker kulit. Diharapkan bahwa sinergi CNN, khususnya dengan menerapkan *fine-tuning* pada arsitektur ResNet50V2, akan meningkatkan akurasi serta efisiensi dalam klasifikasi

kanker kulit, sehingga dapat berkontribusi pada deteksi dini *malignant* dan meningkatkan peluang hidup pasien.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, maka rumusan masalah dalam penelitian ini adalah bagaimana mengklasifikasi kanker kulit menggunakan CNN dengan menerapkan *fine-tuning* pada arsitektur ResNet50V2.

1.3 Tujuan dan Manfaat

Penelitian ini bertujuan untuk mendapatkan performa model klasifikasi terbaik berbasis CNN dengan menerapkan *fine-tuning* pada arsitektur ResNet50V2 dalam mengklasifikasi kanker kulit.

Penelitian ini diharapkan memberikan manfaat untuk demartologis membantu mendeteksi *malignant* secara dini yang lebih akurat dan efisien, sehingga dapat meningkatkan peluang hidup pasien.

1.4 Ruang Lingkup

Ruang lingkup mempunyai tujuan membeikan batasan dari penelitian ini agar tidak memyimpang dan sesuai dengan target yang diinginkan. Penelitian ini memiliki ruang lingkup sebagai berikut:

1. Data yang digunakan dalam penelitian ini adalah *dataset* citra kanker kulit yang terdiri dari kategori *benign* dan *malignant*, yang diambil dari dataset ISIC 2019.
2. Arsitektur yang digunakan dalam penelitian ini adalah ResNet50V2 dengan penerapan *fine-tuning*.
3. Proses pembuatan model menggunakan bahasa pemrograman Python dan *library* Tensorflow Keras.
4. *Tools* yang digunakan dalam penelitian ini yaitu Kaggle *Notebook* dengan menggunakan Bahasa Python.

1.5 Sistematika Penulisan

Untuk memberikan gambaran jelas mengenai pembahasan penyusunan laporan penelitian, maka dibentuk sistematika sebagai berikut:

BAB I PENDAHULUAN

Bab ini berisi mengenai latar belakang masalah, rumusan masalah, tujuan dan manfaat penelitian, dan sistematika penulisan dari penelitian dengan judul Klasifikasi kanker kulit menggunakan *Convolutional Neural Network* dengan Menerapkan *fine-tuning* pada Arsitektur ResNet50V2.

BAB II LANDASAN TEORI

Bab ini berisi tentang tinjauan pustaka dan dasar teori yang digunakan dalam penelitian ini. Pada bab ini, terdiri dari beberapa sub bab antara lain *state of art*, Kanker Kulit, Citra, Klasifikasi Citra, *Preprocessing*, Augmentasi Data, *Convolutional Neural Network, Transfer Learning, Fine-Tuning,* ResNet50V2, *Batch Normalization, Loss Function, Backpropagation,* Adam *Optimizer*, Evaluasi Model.

BAB III METODOLOGI PENELITIAN

Bab ini berisi rancangan penyelesaian mengenai penelitian dan tahap-tahap penelitian yang dilakukan. Gambaran umum dari tahapan penelitian ini adalah pengumpulan data, *preprocessing data*, *split* data menjadi data *training*, *validation*, dan *testing*, *training* model menggunakan *fine-tuning* arsitektur ResNet50V2, *testing* model, dan evaluasi hasil *training* dan *testing*.

BAB IV HASIL DAN PEMBAHASAN

Bab ini menjelaskan hasil dan analisis yang dilakukan sesuai metodologi yang digunakan.

BAB V PENUTUP

Bab ini memuat kesimpulan dan saran yang diperoleh dari perjelasan bab-bab sebelumnya.

BAB II
LANDASAN TEORI

Bab landasan teori ini berisi tinjauan pustaka dan dasar teori yang digunakan dalam penelitian yang berjudul Klasifikasi Kanker Kulit Menggunakan *Convolutional Neural Network* dengan Penerapan *Fine-Tuning* pada Arsitektur ResNet50V2.

2.1 *Literature Review*

Dalam bidang medis, klasifikasi kanker kulit, merupakan salah satu topik yang menarik perhatian para peneliti. Seiring dengan perkembangan teknologi *deep learning*, berbagai metode telah diusulkan untuk meningkatkan akurasi dan efisiensi dalam proses klasifikasi kanker kulit, dengan tujuan untuk mendukung diagnosis dini kanker kulit seperti *malignant*. Tabel 2.1 mengulas penelitian-penelitian terdahulu yang relevan dengan topik klasifikasi citra kanker kulit serta berbagai teknik ekstraksi fitur yang dapat mendukung peningkatan akurasi klasifikasi kanker kulit.

Tabel 2.1 Daftar Penelitian Berkaitan dengan Klasifikasi Kanker Kulit

No	Penelitian	Topik	Domain Data	Metode	Performa
1.	Rupa dkk., (2024)	Klasifikasi melanoma kulit	ISIC 2017 *dataset*, dengan total citra 2600	CNN	Akurasi: 88,89%
2.	Yilmaz & Trocan, (2020)	Klasifikasi melanoma kulit	ISIC 2018 *dataset*, yang terdiri dari 21.570 citra kulit, dengan 19.373 citra *benign* dan 2.197 citra *malignant*	*Transfer learning* pada arsitektur AlexNet	Akurasi: 91,40%
3.	Malo dkk., (2022)	Klasifikasi melanoma kulit	ISIC *dataset*, dengan total citra 2460	*Transfer learning* pada arsitektur VGG-16	Akurasi: 87,6%
4.	Firasari dkk., (2024)	Klasifikasi kanker kulit	ISIC 2017 *dataset*	*Transfer learning*	Akurasi: 88%

No	Penelitian	Topik	Domain Data	Metode	Performa
			dengan total 3297 citra	pada arsitektur ResNet50	
5.	Kreouzi dkk., (2024)	Klasifikasi melanoma kulit	*DermNet Repository* dengan total 8.825 Citra	*Transfer learning* pada arsitektur ResNet50 V2	Akurasi: 91,85%
6.	Pillai dkk., (2024)	Klasifikasi lesi kulit	*Dataset* ISIC yang terdiri dari 3.297 citra kulit dengan label dengan *benign* dan *malignant*	*Fine-Tuning* pada arsitektur ResNet50	Akurasi: 93,33%

Penelitian yang dilakukan oleh Rupa dkk. (2024) menggunakan CNN untuk mengklasifikasikan melanoma kulit menjadi dua kategori, yaitu melanoma *benign* dan *malignant*, dengan *dataset* ISIC 2017. Penelitian ini mencapai akurasi terbaik sebesar 88,89% pada *epoch* ke-45 dengan *batch size* 2, menggunakan dua lapisan konvolusi dan satu lapisan *max pooling*. Penelitian ini menunjukkan bahwa CNN cukup efektif dalam klasifikasi kanker kulit, namun akurasi yang diperoleh masih memiliki ruang untuk peningkatan.

Studi oleh Yilmaz & Trocan (2020) membahas penggunaan AlexNet dalam mengklasifikasikan melanoma. Penelitian ini menggunakan *dataset* ISIC 2018. Penelitian ini mencapai akurasi klasifikasi sebesar 91,40%. Penelitian ini menunjukkan bahwa AlexNet memiliki keunggulan dalam efisiensi waktu komputasi tetapi memiliki keterbatasan dalam mengidentifikasi fitur kompleks pada gambar melanoma, yang menyebabkan akurasi lebih rendah dibandingkan model CNN lainnya.

Penelitian yang dilakukan oleh Malo dkk. (2022) menerapkan arsitektur VGG-16 dalam klasifikasi melanoma menggunakan dataset ISIC yang terdiri dari 1.800 gambar untuk training dan 660 gambar untuk testing. Hasil penelitian menunjukkan bahwa arsitektur VGG-16 mampu mencapai akurasi sebesar 87,6% yang terbukti efektif dalam pengenalan pola dan memiliki kinerja yang cukup baik pada *dataset* melanoma, meskipun

cukup efektif, model ini cenderung memiliki banyak parameter yang membuatnya cukup berat secara komputasi, sehingga memerlukan waktu pelatihan yang lebih lama dan lebih banyak sumber daya

Firasari dkk. (2024) menggunakan arsitektur ResNet50 dalam klasifikasi kanker kulit dengan menguji optimasi Adam, menghasilkan akurasi terbaik sebesar 88%. Penelitian ini menggunakan *dataset* ISIC 2017. Penelitian ini tampaknya lebih rentan terhadap *overfitting*, meskipun mencapai akurasi pelatihan yang tinggi. Mungkin memerlukan strategi tambahan untuk meningkatkan performa pada data validasi.

Penelitian oleh Kreouzi dkk. (2024) menggunakan *transfer learning* pada arsitektur ResNet50V2 dengan dataset DermNet, menghasilkan akurasi klasifikasi sebesar 91,85%. Walaupun ukuran model relatif besar, hal ini tidak menjadi masalah signifikan untuk klasifikasi kanker kulit karena performa yang dihasilkan tetap tinggi dan akurat dalam membedakan kanker kulit ganas dari jinak.

Pillai dkk. (2024) mengimplementasikan *fine-tuning* pada arsitektur ResNet50 dengan *dataset* ISIC berjumlah 3.297 gambar, mencapai akurasi tertinggi sebesar 93,33%. Penelitian ini secara jelas menunjukkan bahwa pendekatan *fine-tuning* secara signifikan meningkatkan kemampuan model dalam membedakan lesi kulit jinak dan ganas.

Berdasarkan tinjauan dari beberapa penelitian tersebut, dapat disimpulkan bahwa penggunaan CNN, arsitektur ResNet50 dengan metode *fine-tuning*, menghasilkan akurasi tinggi dalam klasifikasi kanker kulit. Pendekatan *fine-tuning* menunjukkan peningkatan performa yang lebih signifikan dibandingkan dengan *transfer learning* standar. Arsitektur ResNet50V2 menunjukan akurasi yang paling unggul. Oleh karena itu, penelitian ini akan mengimplementasikan metode CNN dengan *fine-tuning* pada arsitektur ResNet50V2 untuk mengoptimalkan klasifikasi kanker kulit, mengingat efektivitas dan efisiensi yang telah terbukti dalam literatur sebelumnya.

2.2 Kanker Kulit

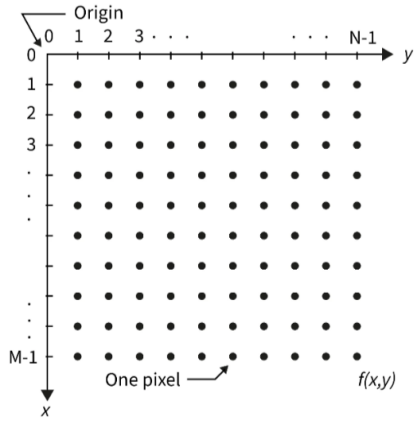
Jumlah kasus kanker kulit di Indonesia cenderung meningkat setiap tahunnya. Sebagian besar kasus ini bermula dari perubahan yang terjadi pada lesi kulit yang awalnya jinak (*benign*), namun berkembang menjadi kanker kulit ganas (*malignant*). Kanker kulit merupakan pertumbuhan tidak terkendali dari sel-sel abnormal pada lapisan kulit, yang

biasanya disebabkan oleh kerusakan DNA akibat paparan sinar ultraviolet dari matahari. Faktor-faktor seperti paparan sinar ultraviolet yang berlebihan dan keturunan dapat mempercepat perkembangan *malignant*. Hal ini menjadi perhatian utama karena banyak individu sering tidak mengenali perubahan lesi kulit sebagai tanda risiko kanker kulit, yang dapat menunda mencari bantuan medis. (Walter dkk., 2010).

2.3 Citra Digital

Konsep citra digital pertama kali diciptakan oleh Russel Kirsch pada tahun 1957, yang mengembangkan teknik untuk menggambarkan objek secara digital. Citra digital merupakan representasi visual dari dunia nyata dalam bentuk digital yang dapat dipahami dan diolah oleh komputer (Rose, 2023). Secara umum, citra digital terdiri dari kumpulan piksel (*pixel*), dimana setiap piksel menyimpan informasi intensitas cahaya dalam satu atau lebih saluran warna. Citra digital dapat diklasifikasikan berdasarkan dimensinya menjadi citra dua dimensi (2D) seperti citra *grayscale* dan citra berwarna (RGB).

Citra digital dapat direpresentasikan secara matematis sebagai fungsi dua variabel diskrit yang memetakan setiap pasangan koordinat piksel ke nilai intensitas tertentu, dapat dilihat pada Gambar 2.1. Sebuah citra digital sering kali disimpan dalam format matriks, dengan ukuran M×N, di mana M adalah jumlah baris (tinggi citra) dan N adalah jumlah kolom (lebar citra). Semakin tinggi nilai M dan N, semakin banyak detail yang dapat disimpan dalam citra.



Gambar 2.1 Representasi Citra Digital dalam Bentuk Koordinat Kartesius
(Suresh & Suresh, 2023)

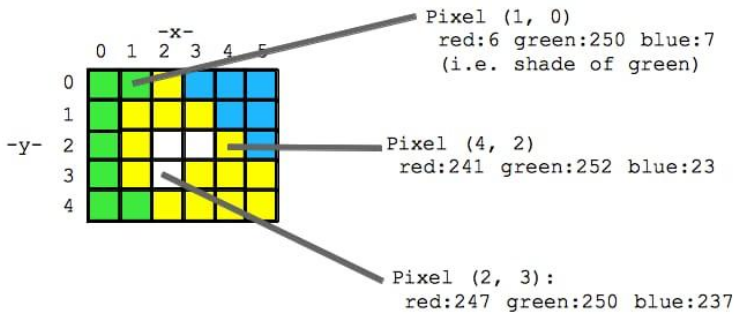
Citra digital dapat digambarkan sebagai sebuah fungsi diskrit $f(x, y)$, yang menggambarkan intensitas citra pada titik koordinat (x, y), di mana x dan y adalah posisi

piksel pada citra. Misalnya, untuk citra *grayscale*, setiap piksel hanya memiliki satu komponen yang menggambarkan tingkat kecerahan. Gambar 2.2 merupakan representasi citra digital dalam bentuk dua dimensi.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Gambar 2.2 Representasi Citra Digital dalam Bentuk Matriks 2 Dimensi
(Rachman dkk., 2024)

Setiap piksel pada citra berwarna dapat direpresentasikan sebagai vektor yang berisi tiga komponen warna, yaitu *Red*, *Green*, dan *Blue* (RGB). Oleh karena itu, citra berwarna dapat dijelaskan dalam bentuk matriks tiga dimensi $f(x, y, c)$, dengan c menunjukkan saluran warna (*Red*, *Green*, *Blue*), dapat dilihat pada Gambar 2.3.



Gambar 2.3 Intensitas Saluran Warna pada Setiap Piksel (Suresh & Suresh, 2023)

2.4 Klasifikasi Citra

Klasifikasi citra merupakan proses pengelompokan citra ke dalam kategori atau kelas tertentu berdasarkan karakteristik atau fitur yang diekstrak dari citra tersebut. Tujuan utama dari klasifikasi citra adalah untuk mengidentifikasi dan membedakan objek atau pola dalam citra secara otomatis menggunakan algoritma komputer. Klasifikasi citra memainkan peran penting dalam berbagai bidang dengan memungkinkan otomatisasi proses identifikasi dan analisis citra yang kompleks (Zhu dkk., 2020).

2.5 *Preprocessing*

Preprocessing merupakan ahap awal yang sangat penting dalam pengolahan data sebelum data tersebut diproses oleh model *machine learning*. Proses ini mencakup berbagai teknik untuk mempersiapkan data agar model dapat mempelajari fitur yang

relevan dengan cara yang optimal (Trojani dkk., 2024). Pada tahap ini, citra yang digunakan untuk pelatihan model perlu dipersiapkan dengan transformasi seperti *resize* dan normalisasi untuk meningkatkan kualitas data dan efisiensi pelatihan.

2.5.1 *Resize*

Resize merupakan teknik *preprocessing* yang digunakan untuk mengubah ukuran citra agar sesuai dengan ukuran input yang diinginkan oleh model. Setiap model *deep learning* yang bekerja dengan citra, memerlukan ukuran *input* yang konsisten. Oleh karena itu, citra yang memiliki ukuran berbeda-beda perlu diubah menjadi ukuran yang seragam. Teknik ini sangat penting dalam meningkatkan efisiensi pelatihan, karena model akan lebih mudah memproses data dengan ukuran yang seragam (Avidan & Shamir, 2007). *Resize* tidak hanya mempengaruhi ukuran citra, tetapi juga dapat membantu dalam menjaga komposisi fitur yang relevan agar model dapat belajar dengan lebih baik

2.5.2 Normalisasi

Normalisasi merupakan salah satu proses *preprocessing* untuk memastikan bahwa nilai piksel citra berada dalam rentang yang sesuai dan dapat diproses secara efisien oleh model. Biasanya, nilai piksel citra berkisar antara 0 hingga 255. Namun, untuk memaksimalkan kinerja model dan menghindari masalah terkait skala, normalisasi dilakukan untuk mengubah rentang nilai piksel tersebut ke dalam rentang [0,1] atau [-1,1]. Proses ini membantu model dalam melakukan pembelajaran dengan lebih efisien, karena nilai yang lebih kecil dan terstandarisasi dapat mempercepat konvergensi selama pelatihan (Singh & Singh, 2020). Normalisasi juga penting untuk menghindari dominasi fitur dengan skala nilai yang besar, sehingga model dapat memproses data secara lebih seimbang. Normalisasi dapat direpresentasikan pada Persamaan 2.1

$$x' = \frac{2(x - \min(x))}{\max(x) - \min(x)} - 1 \quad \text{.....(2.1)}$$

Keterangan:

x : Nilai asli citra yang berada dengan rentang nilai 0 hingga 255

$\min(x)$: Nilai asli citra terkecil dengan rentang nilai 0 hingga 255

$\max(x)$: Nilai asli citra terbesar dengan rentang nilai 0 hingga 255

x' : Nilai citra yang sudah dinormalisasi ke dalam rentang -1 hingga 1

2.6 Augmentasi Data

Augmentasi data merupakan teknik dalam pengolahan citra digital yang digunakan untuk meningkatkan variasi data pelatihan dengan melakukan transformasi pada data yang sudah ada. Augmentasi data sangat penting untuk mengatasi keterbatasan jumlah data pelatihan, yang sering menjadi kendala dalam pengembangan model berbasis *deep learning*. terutama ketika jumlah data asli terbatas. Salah satu teknik augmentasi data dengan menerapkan augmentasi citra secara *real-time* selama pelatihan model. Proses ini dilakukan dengan menerapkan transformasi langsung pada citra saat model dilatih, tanpa menyimpan versi citra yang dimodifikasi ke dalam penyimpanan. Dengan cara ini, efisiensi penggunaan penyimpanan dapat ditingkatkan karena tidak perlu menyimpan semua versi citra hasil augmentasi. Selain itu, memberikan variasi baru di setiap *epoch* pelatihan memungkinkan model belajar dari lebih banyak kombinasi transformasi. Dengan melakukan augmentasi data, model dapat belajar dari berbagai variasi citra yang dihasilkan, sehingga meningkatkan kemampuan generalisasi dan mengurangi risiko *overfitting* (Shorten & Khoshgoftaar, 2019). Augmentasi data dapat direpresentasikan pada Persamaan 2.2 sebagai transformasi T yang diterapkan pada citra asli I.

$$I' = T(I) \quad (2.2)$$

Keterangan :

I' : Citra yang telah diaugmentasi

I : Citra asli

T : Fungsi transformasi metode augmentasi

Beberapa metode umum yang digunakan dalam augmentasi data citra meliputi:

1. *Rotate*

   Rotate merupakan augmentasi data yang memutar citra pada sudut tertentu untuk menciptakan variasi orientasi objek dalam citra. *Rotate* memutar citra sebesar 0 hingga 360 derajat ke arah kiri atau kanan. Teknik ini membantu model mengenali objek dari berbagai sudut pandang, meningkatkan kemampuan deteksi dan klasifikasi objek yang tidak tergantung pada orientasi aslinya. Transformasi rotasi yang diterapkan pada setiap piksel citra asli untuk menghasilkan citra yang telah diputar. Secara matematis, untuk (x, y) merupakan koordinat piksel sebelum rotasi, sedangakan (x', y') merupakan koordinat piksel setelah rotasi. *Rotate* dapat direpresentasikan pada Persamaan 2.3.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{.....(2.3)}$$

Keterangan:

(x', y') : Koordinat citra yang telah dirotasi, x' posisi horizontal dari titik absis dan y' posisi vertikal dari titik ordinat dengan rentang nilai 0 hingga 255

(x, y) : Koordinat citra yang asli, x posisi horizontal dari titik absis dan y posisi vertikal dari titik ordinat dengan rentang nilai 0 hingga 255

θ : Sudut rotasi dalam radian dengan rentang 0 hingga 2π

2. *Width Shift*

Width shift merupakan salah satu teknik augmentasi data untuk memperbesar ukuran dataset dengan menggeser cita secara horizontal. Secara matematis, *width shift* dapat direpresentasikan pada Persamaan 2.4.

$$x' = x + (\alpha_w - w) \quad \text{.....(2.4)}$$

Keterangan:

x' : Koordinat horizontal setelah pergeseran dengan rentang nilai 0 hingga 255

x : Koordinat horizontal yang awal dengan rentang nilai 0 hingga 255

α_w : Proporsi pergeseran horizontal dengan rentang nilai 0 hingga 1

w : Lebar citra dengan rentang nilai 0 hingga 255

3. *Height Shift*

Height shift merupakan salah satu teknik augmentasi data untuk memperbesar ukuran dataset dengan menggeser citra secara vertikal. Secara matematis, *height shift* dapat direpresentasikan pada Persamaan 2.5.

$$y' = y + (\alpha_h - h) \quad \text{.....(2.5)}$$

Keterangan:

y' : Koordinat vertikal setelah pergeseran dengan rentang nilai 0 hingga 255

y : Koordinat vertikal yang awal dengan rentang nilai 0 hingga 255

α_w : Proporsi pergeseran vertikal dengan rentang nilai 0 hingga 1

h : Tinggi citra dengan rentang nilai 0 hingga 255

4. *Shear*

   Shear merupakan salah satu teknik augmentasi data untuk memperbesar ukuran dataset dengan mendistorsi gambar melalui geseran sudut horizontal dan vertikal. *Shear* horizontal dapat direpresentasikan pada Persamaan 2.6.

   $$x' = x + \begin{bmatrix} 1 & \tan(\lambda) \\ 0 & 1 \end{bmatrix} \cdot y \quad (2.6)$$

   Keterangan:

   x' : Koordinat horizontal setelah distorsi dengan rentang nilai 0 hingga 255

   x : Koordinat horizontal yang awal dengan rentang nilai 0 hingga 255

   λ : Pergeseran sudut dengan rentang nilai 0,2 hingga 1

   y : Koordinat vertikal dengan rentang nilai 0 hingga 255

   Untuk *shear* vertikal dapat direpresentasikan pada Persamaan 2.7.

   $$y' = y + \begin{bmatrix} 1 & 0 \\ \tan(\lambda) & 1 \end{bmatrix} \cdot x \quad (2.7)$$

   Keterangan:

   y' : Koordinat vertikal setelah distorsi dengan rentang nilai 0 hingga 255

   y : Koordinat vertikal yang awal dengan rentang nilai 0 hingga 255

   λ : Proporsi pergeseran vertikal dengan rentang nilai 0 hingga 1

   x : Koordinat horizontal dengan rentang nilai 0 hingga 255

5. *Zoom*

   Zoom merupakan salah satu teknik augmentasi data untuk menskalakan data secara memperbesar dan memperkecil. *Zoom* dapat direpresentasikan pada Persamaan 2.8.

   $$x' = z \cdot x \quad (2.8)$$

   Keterangan:

   x' : Koordinat vertikal setelah *zoom* dengan rentang nilai 0 hingga 255

   x : Koordinat vertikal yang awal dengan rentang nilai 0 hingga 255

   z : Representasi faktor *zoom* dari rentang *zoom range*

6. *Horizontal Flip*

   Pembalikan horizontal atau vertikal melibatkan membalik citra secara horizontal atau vertikal. Teknik ini membantu model dalam mengenali objek yang simetris atau memiliki orientasi ganda, meningkatkan akurasi dalam mendeteksi objek yang dapat diputar. Pembalikan horizontal citra dapat direpresentasikan pada Persamaan 2.9.

   $$(x', y') = ((w - 1 - x), y) \quad (2.9)$$

Keterangan:

x' : Koordinat horizontal setelah dibalik dengan rentang nilai 0 hingga 255

x : Koordinat horizontal yang awal dengan rentang nilai 0 hingga 255

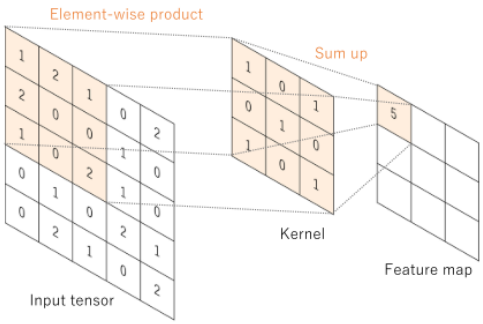
w : Lebar citra dengan rentang nilai 0 hingga 255

2.7 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) adalah salah satu arsitektur jaringan saraf yang paling efektif dan populer dalam bidang pengolahan citra digital. CNN pertama kali dikenalkan oleh Kunihiko Fukushima pada tahun 1979. CNN dirancang khusus untuk mengenali pola spasial dan hierarkis dalam data *grid-like*, seperti citra, dengan menggunakan lapisan konvolusional yang mampu mengekstraksi fitur-fitur penting secara otomatis (Lecun dkk., 1998).

2.7.1 *Convolutional Layer*

Convolution layer terinspirasi secara biologis yang dapat mengekstraksi fitur-fitur lokal dari citra *input* (Lecun dkk., 1998). Lapisan ini menggunakan filter atau kernel yang bergerak melintasi citra untuk menghasilkan *feature map*. Setiap filter menangkap pola tertentu, seperti tepi, tekstur, atau bentuk, yang kemudian digunakan untuk membangun representasi citra yang lebih kompleks pada lapisan berikutnya. Gambar 2.4 merupakan ilustrasi operasi konvolusi.



Gambar 2.4 Operasi *Convolution* (Yamashita dkk., 2018)

Secara matematis, operasi konvolusi dapat direpresentasikan pada Persamaan 2.10. *Input* citra yang akan diproses oleh *convolution layer* dengan ukuran NxN.

$$x_{ij} = \sum_{m=0}^{a-1} \sum_{n=0}^{a-1} K_{mn} I_{i+m,j+n} \quad (2.10)$$

Keterangan:

I : Citra input dengan nilai rentang 0 hingga 255

K : Kernel atau filter konvolusi dengan rentang nilai [-1,1] atau [0,1]

i : Indeks yang menunjukan posisi baris dalam *output* dengan rentang $0 \leq i \leq N - a$

j : Indeks yang menunjukan posisi kolom dalam *output* dengan rentang $0 \leq j \leq N - a$

m : Indeks yang menunjukan posisi baris dalam *filter* dengan rentang $0 \leq m \leq a - 1$

n : Indeks yang menunjukan posisi kolom dalam *filter* dengan rentang $0 \leq n \leq a - 1$

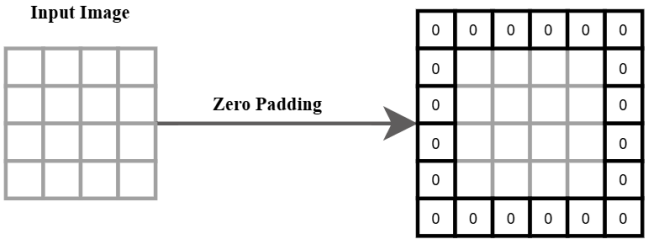
a : Ukuran *filter* konvolusi dengan ukuran $a \times a$

x_{ij} : Nilai *output* untuk *unit* yang terletak di posisi (i, j)

2.7.2 *Padding* dan *Stride*

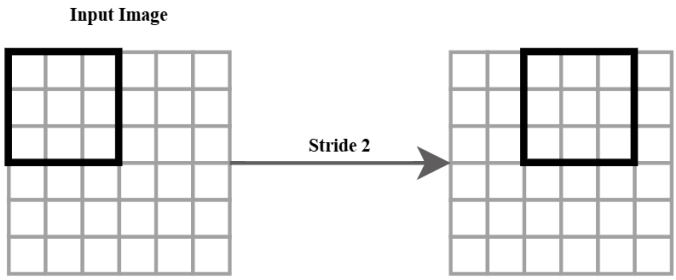
Padding menambahkan lapisan piksel di sekitar tepi citra *input* untuk mengontrol ukuran peta fitur. *Padding* dapat membantu mempertahankan dimensi asli citra setelah operasi konvolusi atau memungkinkan model untuk menangkap informasi tepi yang lebih baik (LeCun dkk., 2010). Secara umum, terdapat dua jenis *padding* yang digunakan, yaitu *valid* padding dan *same padding*.

ResNet50V2 menggunakan strategi *same padding* untuk menjaga dimensi *output* sama dengan dimensi *input* pada lapisan konvolusi. Hal ini sangat penting untuk menjaga konsistensi dimensi pada arsitektur jaringan dengan *skip connection*. Pada ResNet50v2, *same padding* direalisasikan menggunakan *zero padding*, di mana nilai nol ditambahkan di sekitar tepi input sebelum konvolusi. Gambar 2.5 merupakan contoh ilustrasi *zero padding* pada operasi konvolusi.



Gambar 2.5 Ilustrasi *Zero Padding*

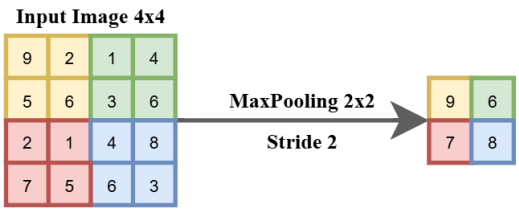
Stride merupakan parameter dalam operasi konvolusi yang menentukan seberapa jauh filter bergerak melintasi citra input selama operasi konvolusi (LeCun dkk., 2010). *Stride* yang lebih besar mengurangi dimensi peta fitur dan meningkatkan kecepatan komputasi, tetapi dapat menyebabkan hilangnya detail. Gambar 2.6 merupakan contoh ilustrasi *stride* 2.



Gambar 2.6 Ilustrasi *Stride* 2

2.7.3 *Pooling Layer*

Pooling layer pertama kali diperkenalkan oleh Kunihiko Fukushima pada tahun 1979. *Pooling layer* bertujuan untuk mengurangi dimensi peta fitur dengan tetap mempertahankan informasi penting dan mengurangi kompleksitas komputasi jaringan (LeCun dkk., 2010). *Pooling* juga membantu membuat jaringan lebih efisien dan meningkatkan kemampuan generalisasi model.



Gambar 2.7 Operasi *Max Pooling* dengan *Filter* Berukuran 2x2 dan *Stride* 2 (Li & Zhou, 2019)

Max pooling merupakan salah satu metode *downsampling* dengan mengambil nilai maksimum dari setiap wilayah kernel dalam *feature map* (Fukushima, 1980). Gambar 2.7 menunjukan contoh operasi *max pooling* dengan filter berukuran 2x2 dan srtide 2. Dalam arsitektur Resnet50v2, *max pooling* digunakan untuk mereduksi ukuran *feature map* secara bertahap tanpa kehilangan informasi penting. Secara matematis, *max pooling* didefinisikan seperti Persamaan 2.11.

$f_{i,j} = \max_{k \times k}(x_{m,n})$.....(2.11)

Keterangan:

k : Ukuran kernel atau filter

m : Indeks baris dalam *filter* dengan rentang $m \in \{1,2,..,k\}$

n : Indeks kolom dalam *filter* dengan rentang $n \in \{1,2,..,k\}$

i : Indeks baris dalam *output poolig* dengan rentang $i \in \{1,2,..,k\}$

j : Indeks kolom dalam *output poolig* dengan rentang $j \in \{1,2,..,k\}$

$f_{i,j}$: *Output* dari operasi *pooling* pada posisi i, j dan saluran k pada *feature map*

2.7.4 *Global Average Pooling*

Global Average Pooling (GAP) adalah teknik yang digunakan untuk mengubah peta fitur menjadi vektor satu dimensi yang dapat digunakan sebagai *input* ke lapisan *fully connected*. *Global average pooling* mengambil rata-rata dari seluruh peta fitur, menghasilkan vektor satu dimensi yang mewakili setiap saluran fitur (Lin dkk., 2013). Secara matematis, GAP didefinisikan seperti Persamaan 2.12.

$S^c = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=0}^{n} x_{i,j}$ .....(2.12)

Keterangan:

M : Jumlah total baris dalam *feature map*

N : Jumlah total kolom dalam *feature map*

m : Jumlah baris dengan nilai rentang $0 \leq m \leq M - 1$

n : Jumlah kolom dengan nilai rentang $0 \leq n \leq N - 1$

i : Nilai pada baris di *feature map* dengan nilai rentang $0 \leq i \leq m - 1$

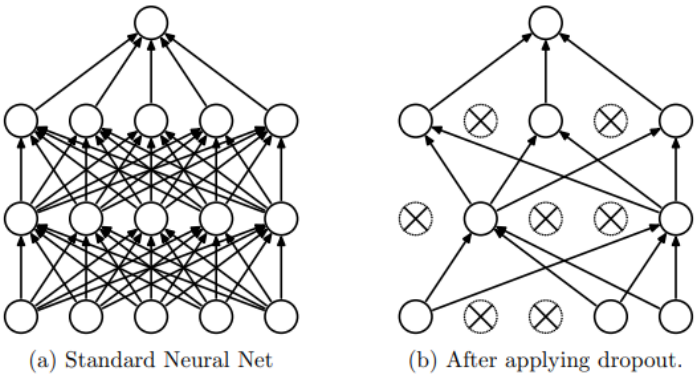
j : Nilai pada kolom di *feature map* dengan nilai rentang $0 \leq j \leq n - 1$

$x_{i,j}$: Bobot elemen matriks pada kolom ke- i dan baris ke- j

S^c : *Output* dari operasi *Global Averaga Pooling*

2.7.5 *Dropout Layer*

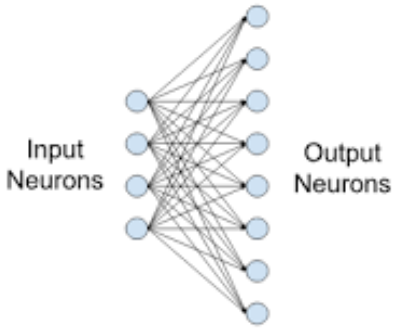
Dropout Layer adalah teknik regularisasi yang digunakan untuk mencegah *overfitting* dalam jaringan neural selama pelatihan (Hinton dkk., 2012). *Dropout* bekerja dengan cara menonaktifkan sejumlah neuron secara acak selama satu iterasi pelatihan, sehingga model tidak bergantung terlalu kuat pada neuron tertentu. Gambar 2.8 merupakan visualisasi *droput.*



Gambar 2.8 Visualisasi Operasi *Dropout* (Srivastava dkk., 2014)

2.7.6 *Fully Connected Layer* (*Dense Layer*)

Konsep *fully connected layer* pertama kali diperkenalkan oleh Frank Rosenblatt pada tahun 1958 dalam pengembangan perceptron, yang merupakan salah satu *neural network* pertama. *Fully connected layer* merupakan lapisan dalam arsitektur jaringan neural di mana setiap neuron di satu lapisan terhubung dengan setiap neuron di lapisan berikutnya. Lapisan ini digunakan di bagian akhir model yang bertanggung jawab untuk menggabungkan fitur-fitur yang telah diekstrak oleh lapisan konvolusi dan *pooling* untuk menjadi representasi akhir untuk tugas klasifikasi (Lecun dkk., 1998). Gambar 2.9 merupakan ilustrasi *fully connected layer*.



Gambar 2.9 *Fully Connected Layer* (Chandra dkk., 2023)

Secara matematis, operasi pada *dense layer* dapat direpresentasikan pada Persamaan 2.13.

$$y = Wx + b \quad (2.13)$$

Keterangan:

y : *Output* dari lapisan *dense* dengan nilai rentang [0,1]

W : Matriks bobot memiliki dimensi $m \times n$

x : *Input* vektor ke lapisan *dense* yang memiliki dimensi $m \times 1$ nilai

b : Vektor bias yang memiliki dimensi $n \times 1$ dengan

2.8 *Transfer Learning*

Transfer learning dikenalkan pertama kali pada tahun 1976 oleh Bozinovski S. dan Fulgosi A. *Transfer learning* merupakan teknik dalam pembelajaran mesin di mana model yang telah dilatih pada satu tugas digunakan kembali atau diadaptasi untuk tugas yang berbeda namun terkait. Pendekatan ini sangat berguna terutama ketika jumlah data yang tersedia untuk tugas target terbatas, karena memungkinkan pemanfaatan pengetahuan yang telah diperoleh dari tugas sumber yang memiliki dataset yang lebih besar atau lebih representatif. *Transfer learning* bertujuan untuk meningkatkan performa pembelajaran pada tugas target dengan memanfaatkan pengetahuan dari tugas sumber (Pan & Yang., 2010).

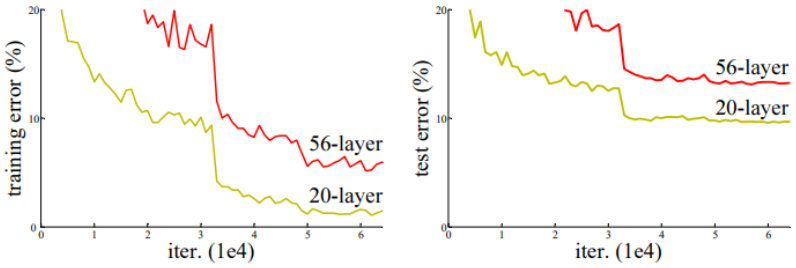
Dalam konteks CNN, *transfer learning* biasanya melibatkan penggunaan arsitektur jaringan yang telah dilatih sebelumnya pada dataset besar seperti ImageNet. Rajaraman dkk. (2018) menyatakan bahwa model yang telah dilatih sebelumnya pada *dataset* skala besar seperti ImageNet dapat berfungsi sebagai ekstraktor fitur yang efektif untuk berbagai tugas visi komputer.

2.9 *Fine-Tuning*

Fine-tuning merupakan teknik dalam *transfer learning* yang melibatkan penyesuaian model yang sudah dilatih sebelumnya yang sudah memiliki kemampuan dasar untuk mengenali pola atau fitur umum dalam data, untuk tugas yang lebih spesifik, dengan tujuan untuk meningkatkan akurasi pada *dataset* yang lebih kecil dan terfokus. Teknik ini memanfaatkan model yang telah dilatih pada *dataset* besar, seperti ImageNet dan disesuaikan lebih lanjut dengan dataset yang lebih kecil yang relevan dengan masalah yang ingin diselesaikan (Vrbančič & Podgorelec, 2020). Dalam *fine-tuning*, biasanya hanya beberapa lapisan terakhir dari model yang dilatih lebih lanjut, sementara lapisan awal yang lebih umum tetap tidak berubah.

2.10 ResNet50V2

ResNet (*Residual Network*) merupakan arsitektur CNN yang diperkenalkan pertama kali oleh Kaiming He, Xiangyu Zhang, Shaoqing Ren, dan Jian Sun pada tahun 2015. Pengembangan arsitektur ini didasari dengan teori semakin bertambahnya *layer* semakin optimal model. Peneliti menemukan bahwa semakin bertambahnya *layer*, maka semakin tinggi masalah *vanishing gradient*, yang mengakibatkan *training error* dan *test error* meningkat yang dapat dilihat pada Gambar 2.10.

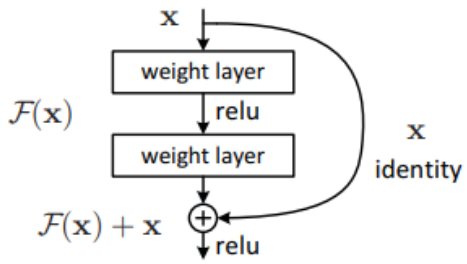


Gambar 2.10 *Training error* dan test error pada 56 dan 20 *layer* (He dkk., 2015)

ResNet dikembangkan untuk mengatasi masalah *vanishing gradient* dan *degradation problem* pada jaringan dalam dengan menggunakan *residual block*, yang memungkinkan model untuk melewati beberapa lapisan tanpa kehilangan informasi penting. Masalah *vanishing gradient* terjadi ketika gradien yang sangat kecil menghambat pembelajaran di lapisan-lapisan yang lebih dalam, sementara *degradation problem* terjadi ketika penambahan lapisan justru menurunkan akurasi model. Dengan pendekatan *residual block*, yang memperkenalkan *shortcut connections*, model dapat menghindari kedua masalah ini dan belajar lebih dalam tanpa mengalami kesulitan dalam mencapai stabilitas dalam pembelajaran.

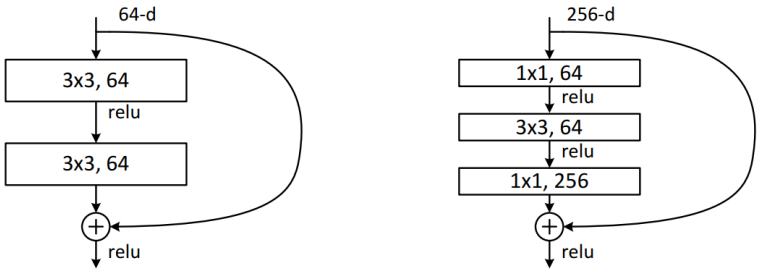
Residual block dalam ResNet dirancang untuk mempermudah proses optimasi dengan mengunakan *shortcut connection* (*identity mapping*). Struktur dasar *residual block* dapat dilihat pada Gambar 2.11. Pada konsep perhitungan fungsi aktivasi yang tadinya $H(x) = f(wx + b)$, menjadi $H(x) = f(x) + x$, jika x dan $f(x)$ memiliki dimensi yang sama. Namun jika x dan $f(x)$ memiliki dimensi yang tidak sama, dapat dilakukan proyeksi linear Ws menjadi $H(x) = f(x) + xWs$. Simbol s pada Ws merujuk pada matriks bobot atau parameter yang digunakan untuk mentransformasikan atau memproyeksikan *input* x sehingga dimensi dari x dan $f(x)$ menjadi kompatibel. Proyeksi linear ini dilakukan dengan mengalikan x dengan matriks Ws, yang berfungsi untuk menyesuaikan dimensi

keduanya agar dapat dijumlahkan. Jadi, Ws merupakan matriks yang digunakan untuk menyesuaikan dimensi x dengan hasil fungsi aktivasi $f(x)$.



Gambar 2.11 Struktur Dasar *Residual Block* (He dkk., 2015)

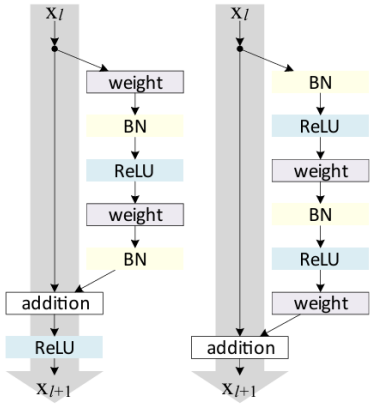
ResNet50 merupakan salah satu arsitektur dari ResNet (*Residual Network*) yang memiliki 50 lapisan. Beberapa aristektur ResNet menggunakan *basic residual block* yang terdiri dari dua *convolution layer*. Sedangkan ResNet50 menggunakan *bottleneck residual block* yang terdiri dari tiga *convolution layer* $(1 \times 1,\ 3 \times 3, 1 \times 1)$, yang dapat dilihat ilustrasinya pada Gambar 2.12. Lapisan pertama (1×1) digunakan untuk reduksi dimensi, lapisan kedua (3×3) untuk ekstraksi fitur, dan lapisan ketiga (1×1) untuk mengembalikan dimensi. Dalam arsitektur ResNet yang ditampilkan, d merupakan jumlah fitur atau saluran (*channel*) yang dihasilkan pada *output* setiap convolution layer.



Gambar 2.12 *Basic residual block* (kiri) dan *bottleneck residual block* (kanan) (He dkk., 2015)

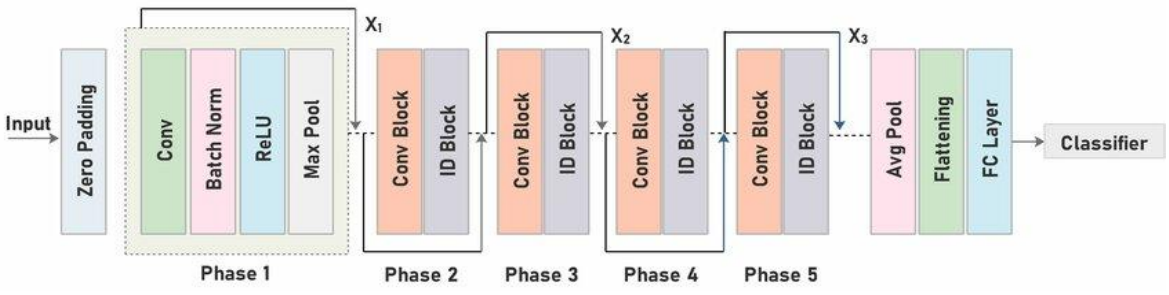
Para peneliti mengembangkan model dengan konsep *pre-activation residual network* sebagai peningkatan dari *residual network* setelah diperkenalkannya ResNet, salah satunya dalam ResNet50 (He dkk., 2016). Maka terbetuknya ResNet50V2 untuk menigkatkan efisiensi *gradient propagation* dan kemudahan optimasi jaringan. Konsep *pre-activation residual network* ini memperbaiki arsitektur *residual netwok* yang sebelumnya posisi BN (*Batch Normalization*) dan ReLU setelah *convolution network*, menjadi sebelumya.

Perbedaan arsitektur residual netwok dengan *pre-activation residual network* dapat dilihat pada Gambar 2.13. Notasi x_l mengacu pada *input* dari *residual block*, yang merupakan *output* dari layer sebelumnya dalam jaringan neural. Sedangkan x_{l+1} menunjukkan *output* dari *residual block* tersebut, yang akan menjadi *input* untuk *residual block* berikutnya dalam jaringan.



Gambar 2.13 *Redual Network* (kiri) dan *Pre-Activation Residual Network* (kanan) (He dkk., 2016)

Arsitektur ResNet50V2 memiliki total 50 lapisan, dengan struktur utama yang terdiri dari *residual block* berbasis *bottleneck*. Gambar 2.14 menggambarkan alur arsitektur ResNet50V2 secara hierarkis, mulai dari *input* hingga klasifikasi akhir. Pada *Phase* 1, data *input* melewati proses *zero padding* untuk menjaga dimensi, kemudian melalui lapisan *convolution* (Conv), *batch nomalization* (BatchNorm), fungsi aktivasi ReLU, dan akhirnya *Max Pooling*. Tahapan ini bertujuan untuk mengekstraksi fitur awal dari citra.



Gambar 2.14 Struktur Arsitektur ResNet50V2 (Patel & Khan, 2023)

Phase 2 hingga *Phase* 5, arsitektur menggunakan kombinasi *residual block* yang terdiri dari dua jenis *block* utama, yaitu Conv *Block* dan *Identity Block* (ID Block). Conv Block digunakan untuk mengubah dimensi dan mengekstraksi fitur lebih kompleks,

sementara ID *Block* mempertahankan dimensi dan membantu dalam pelatihan jaringan yang lebih dalam dengan memanfaatkan *shortcut connection*. *Shortcut connection* ini memungkinkan gradien mengalir langsung ke lapisan awal, sehingga mengatasi masalah *vanishing gradient*. Setelah fase-fase *residual* selesai, data memasuki tahap akhir yang melibatkan *average pooling* (Avg Pool) untuk merangkum informasi fitur secara global. Kemudian, data diflatten menjadi vektor satu dimensi sebelum diteruskan ke *fully connected layer* (FC Layer) untuk menghasilkan *output* klasifikasi akhir.

2.11 Fungsi Aktivasi

Fungsi aktivasi merupakan komponen dalam *neural network* yang bertugas mengubah *input* linear yang berhubungan secara proporsional dengan bobot menjadi *output* non-linear yang memungkinkan model untuk menangkap hubungan kompleks dalam data. Fungsi ini menentukan apakah suatu neuron akan diaktifkan atau tidak, sehingga memungkinkan jaringan memperlajari pola yang kompleks dalam data. Tanpa fungsi aktivasi, *nerual network* hanya akan melakukan transformasi linear, yang berarti tidak dapat menangani data yang dapat dipisahkan secara linear (Parhi & Nowak., 2020). Fungsi aktivasi yang digunakan ResNet50V2 untuk luaran *layer* adalah ReLU. Sedangkan untuk klasifikasi digunakan *sigmoid*.

2.11.1 ReLU

Konsep ReLU (*Rectified Linear Unit*) diperkenalkan oleh Alston Householder pada tahun 1941, yang berfokus pada penggunaan fungsi *rectification* dalam konteks yang lebih umum. Namun, ReLU pertama kali diperkenalkan dalam *deep learning* oleh Kunihiko Fukushima pada tahun 1968. ReLU bekerja dengan mengubah setiap nilai *input* negatif menjadi nol, sementara nilai positif tetap dipertahankan (Fukushima, 1969). Secara matematis, ReLU didefinisikan seperti Persamaan 2.14. Fungsi ini memberikan non-linearitas dalam jaringan saraf, mencegah masalah *vanishing gradient*, den memungkinkan *gradient propagation* yang lebih baik dalam *deep learning*.

$$f(x) = \max(0, x) \quad (2.14)$$

Keterangan:

Max : Nilai maksimum

x : Nilai *input*

$f(x)$: Nilai *output* setelah fungsi aktivasi diterapkan pada *input* x

2.11.2 *Sigmoid*

Sigmoid merupakan salah satu fungsi aktivasi digunakan dalam *neural network* yang memperkenalkan non-linearitas ke dalam model. Fungsi ini mengubah *input* menjadi *output* dalam rentang 0 hingga 1, yang sering digunakan dalam tugas klasifikasi biner (Singh, Y dkk., 2023). Secara matematis, fungsi *sigmoid* dapat didefinisikan pada Persamaan 2.15.

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (2.15)$$

Keterangan:

x : Nilai *Input* hasil perhitungan *linear*, $x \in \mathbb{R}$

e : Bilangan euler yang bernilai 2,718..

$\sigma(x)$: *Output* dari fungsi *Sigmoid* selalu berada dalam rentang (0,1)

2.12 *Batch Normalization*

Batch Normalization merupakan teknik yang digunakan dalam pelatihan *neural network* untuk mempercepat proses konvergensi dan meningkatkan stabilitas model. *Batch Normalization* bertujuan untuk mengatasi masalah *internal covariate shift*, yaitu perubahan distribusi *input* ke setiap lapisan jaringan selama pelatihan (DiIoffe & Szegedy., 2015). Dengan menormalkan *input* setiap *mini-batch*, teknik ini memungkinkan penggunaan tingkat pembelajaran yang lebih tinggi dan mengurangi ketergantungan pada inisialisasi bobot yang cermat.

Proses *Batch Normalization* terdiri dari lima tahapan utama yaitu perhitungan mean, perhitungan varians, normalisasi, pergeseran (*shifting*), dan skala (*scaling*). Tahap pertama dalam *Batch Normalization* adalah menghitung rata-rata (*mean*) dari setiap fitur dalam *mini-batch*. Untuk setiap saluran (*channel*) pada *output* lapisan konvolusi, mean dihitung dengan mempertimbangkan seluruh sampel dalam *batch* serta dimensi spasial. Secara matematis, perhitungan *mean* dapat didefinisikan pada Persamaan 2.16.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad (2.16)$$

Keterangan:

B : *Mini batch,* subset dari *dataset* dengan ukuran m

m : Jumlah sampel dalam satu *mini batch* dengan $m \geq 1$

i : indeks sampel dalam *mini batch* dengan rentang $i \in \{1,2,\dots,m\}$

x_i : *Input* sampel data ke-i dalam *mini batch* yang setiap sampel memiliki d fitur atau dimensi dengan $x_i \in \mathbb{R}^d$

μ_B : Nilai rata-rata dari semua sampel dalam *mini batch*

Varians dari setiap fitur dalam *mini-batch* dihitung untuk mengukur sebaran data setelah mean dihitung. Varians dihitung dengan mempertimbangkan seluruh sampel dalam *batch* serta dimensi fitur. Secara matematis, perhitungan varians dapat didefinisikan pada Persamaan 2.17.

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad (2.17)$$

Keterangan:

B : *Mini batch*, subset dari *dataset* dengan ukuran m

m : Jumlah sampel dalam satu *mini batch* dengan $m \geq 1$

i : indeks sampel dalam *mini batch* dengan rentang $i \in \{1,2,\dots,m\}$

x_i : *Input sampel* data ke-i dalam *mini batch* yang setiap sampel memiliki d fitur atau dimensi dengan $x_i \in \mathbb{R}^d$

μ_B : Nilai rata-rata dari semua sampel dalam *mini batch*

σ_B^2 : Varians *mini-batch* untuk saluran $\boldsymbol{c}$

Varians telah dihitung, kemuudian tahap normalisasi yang bertujuan untuk mengubah distribusi data sehingga memiliki *mean* nol dan varians satu. Proses ini dilakukan dengan menggunakan nilai *mean* dan varians yang telah dihitung. Perhitungan normalisasi dapat didefinisikan pada Persamaan 2.18.

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{(\sigma_B^{(k)})^2 + \epsilon}} \quad (2.18)$$

Keterangan:

i : indeks sampel dalam *mini batch* dengan rentang $i \in \{1,2,\dots,m\}$

k : indeks fitur (dimensi) dari *input* dengan k ditentukan oleh jumlah fitur dalam satu sampel (d) maka dari itu $k \in \{1,2,\dots,d\}$

$x_i^{(k)}$: Nilai fitur ke-k dari sampel ke-i sebelum normalisasi

$\mu_{\boldsymbol{B}}^{(k)}$: *Mean mini batch* untuk fitur ke-k

$\sigma_B^{(k)}$: Standar deviasi *mini bacth* untuk ke-k

ϵ : Konstanta kecil dengan nilai 10^{-3}

$\hat{\mathrm{x}}_i^{(k)}$: Nilai fitur ke-k dari sampel ke-i setelah normalisasi

Normalisasi telah dilakukan, kemudian tahap terakhir dalam *batch normalization* yaitu mentransformasikan kembali nilai yang telah dinormalisasi dengan dua parameter γ (skala) dan β (pergeseran). Pergeseran ini memungkinkan model untuk mengembalikan representasi data jika diperlukan. Sedangkan skala memungkinkan model untuk mengontrol tingkat variasi data setelah normalisasi. Perhitungan transformasi dapat didefinisikan pada Persamaan 2.19.

$$y_i^{(k)} = \gamma^{(k)}\hat{\mathrm{x}}_i^{(k)} + \beta^{(k)} \quad (2.19)$$

Keterangan:

$\gamma^{(k)}$: Parameter skala dengan rentang nilai $\gamma^{(k)} \in \mathbb{R}^+$(Bilangan real positif)

$\hat{\mathrm{x}}_i^{(k)}$: Nilai *input* yang telah dinormalisasi sebelumnya

$\beta^{(k)}$: Parameter pergeseran dengan rentang nilai $\beta^{(k)} \in \mathbb{R}$ (Bilangan real)

$y_i^{(k)}$: Nilai setelah transformasi untuk sampel ke-i pada fitur ke-k

2.13 *Loss Function*

Loss Function merupakan salah satu komponen dalam *neural network* yang bertugas dalam mengukur seberapa baik model membuat prediksi dibandingkan dengan nilai sebenarnya dan memberikan sinyal yang diperlukan untuk memperbarui parameter model selama pelatihan (Goodfellow dkk., 2016). Dengan demikian, fungsi kehilangan berperan sebagai panduan bagi algoritma optimisasi untuk memperbarui bobot model dengan tujuan meminimalkan kesalahan prediksi.

Salah satu *loss function* yang paling umum digunakan dalam tugas klasifikasi biner yaitu *Binary Cross-Entropy Loss Function*. Fungsi ini mengukur perbedaan antara distribusi probabilitas yang diprediksi oleh model dan distribusi probabilitas yang sebenarnya. *Binary Cross-Entropy* efektif dalam mengoptimalkan model untuk menghasilkan prediksi yang akurat dalam konteks di mana setiap sampel hanya dapat dikategorikan ke dalam salah satu dari dua kelas yang saling eksklusif. *Binary Cross-*

Entropy Loss Function dirancang khusus untuk tugas klasifikasi biner, di mana setiap sampel data memiliki label yang bernilai 0 atau 1 (Zhao dkk., 2010). Secara sistematis, *binary cross-entropy loss function* dapat didefinisikan pada Persamaan 2.20.

$$L = -\frac{1}{N}\sum_{j=1}^{N}[t_j \log(o_j) + (1 - t_j)\log(1 - o_j)] \quad (2.20)$$

Keterangan:

j : indeks sampel dalam *batch data*, $j \in [1, N]$

N : Jumlah sampel dalam satu *batch*, $N \geq 1$

t_j : Label kelas sebenarnya (0 atau 1) untuk sampel ke- j

o_j : Probabilitas yang diprediksi oleh model bahwa sampel ke- j termasuk dalam kelas 1 yang diperoleh dari *output* fungsi aktivasi *sigmoid*

2.14 *Backpropagation*

Backpropagation adalah algoritma yang digunakan dalam pelatihan jaringan saraf tiruan. Algoritma ini memungkinkan jaringan untuk memperbarui bobot-bobotnya secara efisien dengan meminimalkan fungsi kehilangan (*loss function*) melalui proses *backpropagation* dari *output* menuju *input*. *Backpropagation* memainkan peran penting dalam memungkinkan model belajar dari data dengan cara mengoptimalkan parameter jaringan untuk meningkatkan akurasi prediksi. Integrasi *backpropagation* dengan arsitektur jaringan yang dalam seperti ResNet-50 memungkinkan model untuk belajar representasi yang kompleks dan mendalam secara efisien (He dkk., 2016).

Secara matematis, proses *backpropagation* melibatkan dua langkah utama, yaitu *forward pass* dan *backward pass*. Pada langkah *forward pass*, *input* data diteruskan melalui setiap lapisan jaringan untuk menghasilkan *output* prediksi berdasarkan bobot-bobot yang ada. Proses ini mencakup perhitungan aktivasi di setiap lapisan hingga mencapai lapisan *output*. Setelah *output* prediksi dihasilkan, langkah *backward pass* dimulai, di mana gradien dari fungsi kehilangan dihitung terhadap setiap bobot dalam jaringan melalui *backpropagation* dari *output* menuju *input*. Perhitungan gradien dilakukan dengan teknik *chain rule* yang dapat didefinisikan pada Persamaan 2.21.

$$\frac{\partial L}{\partial W_{i,j}} = \frac{\partial L}{\partial o_j} \cdot \frac{\partial o_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial W_{i,j}} \quad (2.21)$$

Keterangan:

∂L : Turunan dari L yang merupakan *loss function*

∂o_j : Turunan dari o_j, *output* dari neuron ke-j yang merupakan hasil dari proses aktivasi Neuron

∂z_j : Turunan dari z_j, nilai aktivasi pada neuron j sebelum diterapkan fungsi aktivasi yang diperoleh dengan menjumlahkan hasil perkalian *input* dan bobot.

$\partial W_{i,j}$: Turunan dari $W_{i,j}$ yang merupakan bobot antara neuron i dan neuron j dalam jaringan.

Untuk dapat menghitung Persamaan 2.22, terlebih dahulu meghitung turunan dari tiap komponennya. Penurunan pertama dapat dilakukan dengan nilai *cross-entropy* pada Persamaan 2.23.

$$\frac{\partial L}{\partial o_j} = \frac{\partial}{\partial o_j}(-t_j \log(o_j)) \quad (2.22)$$

Keterangan:

∂o_j : Turunan dari o_j, *output* dari neuron ke-j yang merupakan hasil dari proses aktivasi Neuron

t_j : Label kelas sebenarnya (0 atau 1) untuk sampel ke-j

$\log(o_j)$: Logaritma natural dari o_j, yang merupakan *output* prediksi model

∂L : Turunan dari L yang merupakan *loss function*

$$\frac{\partial L}{\partial o_j} = \frac{-t_j}{o_j} \quad (2.23)$$

Keterangan:

t_j : Label kelas sebenarnya (0 atau 1) untuk sampel ke-j

o_j : Probabilitas yang diprediksi oleh model bahwa sampel ke-j termasuk dalam kelas 1 yang diperoleh dari output fungsi aktivasi *sigmoid*

∂L : Turunan dari L yang merupakan *loss function*

∂o_j : Turunan dari o_j, *output* dari neuron ke-j yang merupakan hasil dari proses aktivasi Neuron

Penurunan kedua dapat dilakukan dengan fungsi aktivasi *sigmoid* pada Persamaan 2.24, sehingga menghasilkan Persamaan 2.25.

$$\frac{\partial o_j}{\partial z_j} = \frac{\partial}{\partial z_j}\frac{e^{z_j}}{\sum_j e^{z_j}} \quad (2.24)$$

Keterangan:

e : Bilangan euler yang bernilai 2,718..

∂o_j : Turunan dari o_j, *output* dari neuron j yang merupakan hasil dari proses aktivasi Neuron

∂z_j : Aktivasi linear sebelum fungsi aktivasi pada neuron ke-j

$$\frac{\partial o_j}{\partial z_j} = o_j(1 - o_j) \quad (2.25)$$

Keterangan:

o_j : Probabilitas yang diprediksi oleh model bahwa sampel ke-j termasuk dalam kelas 1 yang diperoleh dari output fungsi aktivasi *sigmoid*

∂o_j : Turunan dari o_j, *output* dari neuron j yang merupakan hasil dari proses aktivasi Neuron

∂z_j : Aktivasi linear sebelum fungsi aktivasi pada neuron ke-j

Penurunan ketiga dapat dilakukan dengan fungsi aktivasi *sigmoid*, sehingga menghasilkan persamaan 2.26.

$$\frac{\partial z_j}{\partial w_{i,j}} = o_j \quad (2.26)$$

Keterangan:

o_j : Probabilitas yang diprediksi oleh model bahwa sampel ke-j termasuk dalam kelas 1 yang diperoleh dari output fungsi aktivasi *sigmoid*

∂z_j : Aktivasi linear sebelum fungsi aktivasi pada neuron ke-j

$\partial w_{i,j}$: Bobot antara neuron i dan neuron j

Ketiga komponen sudah melakukan penurunan, sehingga hasil penuunan digabungkan yang akan mendapatkan Persamaan 2.27.

$$\frac{\partial L}{\partial w_{i,j}} = \frac{-t_j}{o_j} \cdot o_j(1 - o_j) \cdot o_j \quad (2.27)$$

Keterangan:

t_j : Label kelas sebenarnya (0 atau 1) untuk sampel ke-j

o_j : Probabilitas yang diprediksi oleh model bahwa sampel ke-j termasuk dalam kelas 1 yang diperoleh dari output fungsi aktivasi *sigmoid*

2.15 Adam *Optimizer*

Adam *Optimizer* (*Adaptive Moment Estimation*) merupakan salah satu algoritma optimisasi yang paling populer dan banyak digunakan dalam pelatihan jaringan saraf tiruan, termasuk CNN. Adam menggabungkan keuntungan dari dua metode optimisasi lainnya, yaitu Momentum dan AdaGrad, untuk mempercepat konvergensi dan meningkatkan kinerja model secara keseluruhan (Kingma & Ba, 2014).

Adam *Optimizer* dirancang untuk mengatasi beberapa keterbatasan dari algoritma optimisasi tradisional seperti *Stochastic Gradient Descent* (SGD). Adam memanfaatkan estimasi momen pertama (*mean*) dan momen kedua (varians) dari gradien untuk menyesuaikan laju pembelajaran secara adaptif untuk setiap parameter model. Hal ini memungkinkan Adam untuk melakukan penyesuaian yang lebih tepat dan efisien selama proses pelatihan. Adam bekerja dengan cara memperbarui parameter model berdasarkan estimasi adaptif dari momen pertama dan kedua dari gradien.

Dalam melakukan pembaharuan parameter pada adam *optimizer* terlebih dahulu untuk mencari nilai momen vektor pertama m_t (*mean* dari gradien) dan kedua v_t (*mean* dari kuadrat gradien) secara berurut menggunakan Persamaan 2.28 dan Persamaan 2.29.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (2.28)$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (2.29)$$

Keterangan:

t : Iterasi dalam proses pelatihan

m_t : Perkiraan momen pertama pada iterasi ke- t

v_t : Perkiraan momen kedua pada iterasi ke- t

g_t : *Gradient* pada iterasi ke- t untuk momen pertama

g_t^2 : Kuadrat *gradient* pada iterasi ke- t untuk momen kedua

β_1 : *Decay coefficient* untuk momen pertama dengan rentang $0 \leq \beta_1 < 1$

β_2 : *Decay coefficient* untuk momen kedua dengan rentang $0 \leq \beta_2 < 1$

Kemudian mencari bias *correction* menghitung *moment* vektor pertama dan kedua. Dikarenakan m_t dan v_t diinisialisasi pada nol, dilakukan koreksi bias yang ditunjukan pada Persamaan 2.30 dan 2.31.

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \quad (2.30)$$

$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$.....(2.31)

Keterangan:

m_t : Momen pertama pada iterasi ke- t, yang merupakan mean dari gradien

v_t : Momen kedua pada iterasi ke- t, yang merupakan estimasi mean dari kuadrat gradien

β_1 : *Decay coefficient* untuk momen pertama dengan rentang $0 \leq \beta_1 < 1$

β_2 : *Decay coefficient* untuk momen kedua dengan rentang $0 \leq \beta_2 < 1$

$\hat{m}$: Momen pertama yang telah dikoreksi

$\hat{v}_t$: Momen kedua yang telah dikoreksi

Kemudian memperbaharui nilai bobot yang lama menjadi bobot yang baru menggunakan laju pembelajaran α ditunjukan pada Persamaan 2.32.

$w_t = w_{t-1} - \hat{m}_t \left(\frac{\alpha}{\sqrt{\hat{v}_t+\epsilon}}\right)$.....(2.32)

Keterangan:

w_t : Bobot pada *timestep* ke-t

$\hat{m}_t$: Koreksi momentum pertama

$\hat{v}_t$: Koreksi momentum kedua

α : *Learning rate* (nilai *default* 0.001)

ϵ : Konstanta 10^{-8}

w_{t-1} : Bobot pada *timestep* ke-$t-1$

2.16 Evaluasi Model

Evaluasi model adalah tahap penting dalam pengembangan dan pelatihan model untuk memastikan bahwa model tidak hanya bekerja baik pada data pelatihan tetapi juga mampu menggeneralisasi dengan baik pada data yang belum pernah dilihat sebelumnya. Pemilihan metrik evaluasi yang sesuai sangat penting untuk mengukur kinerja model secara objektif, terutama dalam tugas klasifikasi yang melibatkan ketidakseimbangan kelas (Sokolova & Lapalme, 2009).

Confusion matrix merupakan tabel yang digunakan untuk menggambarkan kinerja model klasifikasi pada set data uji yang dikenal labelnya. Matriks ini menunjukkan jumlah prediksi yang benar dan salah yang dilakukan oleh model dibandingkan dengan label aktual, dapat dilihat pada Gambar 2.15.

		Prediksi	
		Positif (+)	Negatif (-)
Aktual	Positif (+)	TP	FP
	Negatif (-)	FN	TN

Gambar 2.15 *Confusion Matrix* 2x2

TP (*True Positive*) merupakan jumlah kasus di mana model memprediksi kelas positif dan hasilnya sesuai dengan label aktual, yang berarti model berhasil mendeteksi kelas positif dengan benar. TN (*True Negative*) mengacu pada jumlah kasus di mana model memprediksi kelas negatif dan hasilnya juga benar, artinya model berhasil mengidentifikasi kelas negatif dengan akurat. Sebaliknya, FP (*False Positive*) menunjukkan jumlah kasus di mana model memprediksi kelas positif, namun hasilnya salah karena label aktualnya adalah negatif. Terakhir, FN (*False Negative*) adalah jumlah kasus di mana model memprediksi kelas negatif, tetapi hasilnya salah karena label aktualnya adalah positif.

Confusion matrix ini sangat penting untuk mengevaluasi kinerja model klasifikasi dengan menghitung berbagai metrik seperti *accuracy*, *precision*, *recall*, dan *F1-score*, yang memberikan gambaran lebih jelas mengenai seberapa baik model dalam mengklasifikasikan data. *Accuracy* mengukur proporsi prediksi yang benar dari total keseluruhan data, tetapi bisa menjadi kurang representatif ketika data tidak seimbang. Secara sistematis, *accuracy* dapat didefinisikan pada Persamaan 2.33.

$$Accuracy\ (Binary\ Class) = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.33)$$

Keterangan:

TP : *True Positive* jumlah kasus di mana model memprediksi kelas positif dan hasilnya sesuai dengan label aktual

FP : *False Positive* menunjukkan jumlah kasus di mana model memprediksi kelas positif, namun hasilnya salah

TN : *True Negative* jumlah kasus di mana model memprediksi kelas negatif dan hasilnya sesuai dengan label aktual

FN : *False Negative* menunjukkan jumlah kasus di mana model memprediksi kelas negatif, tetapi hasilnya salah

Precision mengukur seberapa tepat model dalam memprediksi kelas positif, yaitu berapa banyak prediksi positif yang benar dibandingkan dengan total prediksi positif yang dibuat. Secara sistematis, *precision* dapat didefinisikan pada Persamaan 2.34.

$$Precision = \frac{TP}{TP+FP} \quad (2.34)$$

Keterangan:

TP : *True Positive* jumlah kasus di mana model memprediksi kelas positif dan hasilnya sesuai dengan label aktual

FP : *False Positive* menunjukkan jumlah kasus di mana model memprediksi kelas positif, namun hasilnya salah

Recall berfokus pada seberapa baik model menangkap semua contoh positif yang ada, yaitu berapa banyak prediksi positif yang benar dibandingkan dengan total data positif yang sebenarnya. Secara sistematis, *recall* dapat didefinisikan pada Persamaan 2.34

$$Recall = \frac{TP}{TP+FN} \quad (2.35)$$

Keterangan:

TP : *True Positive* jumlah kasus di mana model memprediksi kelas positif dan hasilnya sesuai dengan label aktual

FN : *False Negative* menunjukkan jumlah kasus di mana model memprediksi kelas negatif, tetapi hasilnya salah

F1-score merupakan rata-rata geometri dari *precision* dan *recall*, yang memberikan gambaran yang lebih seimbang mengenai kinerja model, terutama saat ada ketidakseimbangan antara kelas positif dan negatif. Secara sistematis, *F1-score* dapat didefinisikan pada Persamaan 2.35.

$$f1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.36)$$

2.17 *Tools* dan *Library*

Tools dan *library* merupakan alat yang digunakan untuk membantu pelaksanaan penelitian ini. *Tools* dan *library* ini mencakup *environment* dan *library-library* dalam bahasa pemograman. Penelitian ini menggunakan *environment* Kaggle *notebook* menggunakan bahasa pemograman Python versi 3.10.14 dengan *library* Tensorflow, Keras, *Numpy*, dan Pandas.

2.17.1 Kaggle *Notebook*

Kaggle *Notebook* merupakan *platform* berbasis *cloud* yang memungkinkan pengguna untuk membuat, mengembangkan, dan berbagi kode Python dalam lingkungan yang mudah diakses. Kaggle menyediakan kernel (*notebook*) yang memungkinkan data *scientists* dan *machine learning practitioners* untuk menulis dan menjalankan kode secara langsung tanpa perlu menyiapkan infrastruktur lokal. *Platform* ini juga menyediakan *dataset* yang dapat diakses langsung untuk analisis dan pengembangan model. Kaggle sangat populer di kalangan komunitas *data science* karena menyediakan berbagai kompetisi yang memotivasi pengembangan model *machine learning* terbaik serta akses ke berbagai *library* dan *tools* seperti TensorFlow dan Keras.

2.17.2 Tensorflow

TensorFlow adalah *library open-source* yang dikembangkan oleh Google untuk pengembangan *machine learning* dan *deep learning*. TensorFlow memungkinkan pembuatan model yang dapat diparalelkan untuk berbagai perangkat keras, mulai dari CPU, GPU, hingga TPU, serta memberikan API yang fleksibel untuk membangun model yang skalabel dan efisien. Dalam jurnal oleh Abadi dkk. (2016), TensorFlow dijelaskan sebagai salah satu *framework* paling populer untuk *deep learning*, yang dirancang untuk komputasi numerik yang besar dan mendalam, serta mendukung model dengan arsitektur yang kompleks. Dalam penelitian ini versi TensorFlow yang digunakan berasal dari lingkungan Kaggle *notebook* yaitu Tensorflow versi 2.16.1.

2.17.3 Keras

Keras adalah *library* tingkat tinggi yang dibuat untuk mempermudah pembuatan dan pelatihan model *deep learning*. Keras dapat berjalan di atas beberapa *backend*, termasuk TensorFlow, dan menawarkan API yang sangat sederhana dan modular. Keras dirancang sederhana dan mudah digunakan, sehingga dapat diakses oleh pemula dan efisien bagi para ahli dapat dipilih dan disesuaikan sesuai kebutuhan model. Dalam penelitian ini versi Keras yang digunakan berasal dari lingkungan Kaggle *notebook* yaitu Keras versi 3.3.3.

2.17.4 Numpy

NumPy adalah *library fundamental* untuk komputasi numerik di Python, menyediakan struktur data seperti *array* yang memungkinkan operasi matematika dan manipulasi data yang cepat dan efisien. NumPy memungkinkan perhitungan matematis dan aljabar linier dengan kecepatan tinggi yang sangat penting dalam pengembangan model *machine learning*. NumPy berfungsi sebagai lapisan interoperabilitas antara berbagai pustaka komputasi *array*, berkat posisinya yang sentral dalam ekosistem Python ilmiah. Hal ini memudahkan integrasi dengan pustaka dan kerangka kerja khusus lainnya, meningkatkan fleksibilitas dan utilitasnya dalam berbagai aplikasi penelitian dan industri (Harris dkk., 2020). Dalam penelitian ini versi Numpy yang digunakan berasal dari lingkungan Kaggle *notebook* yaitu Numpy versi 1.26.4.

2.17.5 Pandas

Pandas merupakan pustaka yang sangat dikenal untuk pengolahan dan analisis data di Python. Dalam penelitian ini, digunakan Pandas versi 2.2.3 yang dijalankan dalam lingkungan Kaggle *notebook*. Dengan versi ini, pengguna dapat dengan mudah mengelola data dalam format tabular seperti CSV, Excel, dan SQL. Selain itu, Pandas juga memfasilitasi pelaksanaan eksplorasi data awal (EDA). Menurut penelitian McKinney (2010), Pandas diakui sebagai alat yang sangat efektif dalam menangani data yang besar dan kompleks. Dengan menyediakan struktur data yang fleksibel seperti DataFrame dan Series, Pandas mempermudah pelaksanaan operasi seperti penyaringan, agregasi, dan transformasi data. Tahapan-tahapan ini merupakan elemen krusial dalam persiapan data sebelum diaplikasikan pada model pembelajaran mesin.