BAB 2
LANDASAN TEORI

2.1 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) merupakan arsitektur *deep learning* yang dirancang khusus untuk memproses data dengan struktur *grid* seperti gambar. CNN terdiri dari beberapa komponen utama yang bekerja secara hierarkis. *Convolutional layers* bertugas mengekstrak fitur lokal dari gambar menggunakan operasi konvolusi antara *filter* yang dapat dipelajari dan *feature map* masukan. *Pooling layers* bertugas mengurangi dimensi spasial *feature map* sehingga representasi fitur menjadi lebih ringkas dan invariant terhadap pergeseran kecil. *Fully connected layers* pada bagian akhir arsitektur bertugas melakukan klasifikasi berdasarkan fitur yang telah diekstrak oleh lapisan sebelumnya [13].

Keunggulan CNN dibandingkan jaringan *fully connected* biasa terletak pada konsep *parameter sharing* dan *local connectivity*. Dengan *parameter sharing*, bobot *filter* yang sama diterapkan pada seluruh bagian gambar, sehingga jumlah parameter yang perlu dipelajari jauh lebih sedikit. Karakteristik ini menjadikan CNN lebih efisien secara komputasi dan lebih mudah dilatih [13]. Salah satu komponen penting dalam arsitektur CNN modern adalah *batch normalization* [14]. Teknik ini menormalkan aktivasi setiap lapisan menggunakan statistik dari *mini-batch* selama proses pelatihan, sehingga mempercepat konvergensi dan meningkatkan stabilitas pelatihan. *Batch normalization* memungkinkan penggunaan *learning rate* yang lebih tinggi dan mengurangi sensitivitas model terhadap inisialisasi parameter [14].

Rumus 2.1 menunjukkan operasi dasar konvolusi pada CNN yang ditulis oleh Goodfellow et al. tahun 2016 [13]. Pada proses ini, sebuah *kernel* atau *filter* K digeser di atas gambar masukan I untuk menghitung nilai pada setiap posisi keluaran $S(i,j)$. Nilai keluaran diperoleh dari penjumlahan hasil perkalian elemen-elemen yang bersesuaian antara *kernel* dan bagian gambar yang sedang diamati. Proses ini memungkinkan CNN mengekstraksi berbagai pola lokal, seperti tepi, tekstur, maupun bentuk, yang selanjutnya direpresentasikan dalam bentuk *feature map* [13].

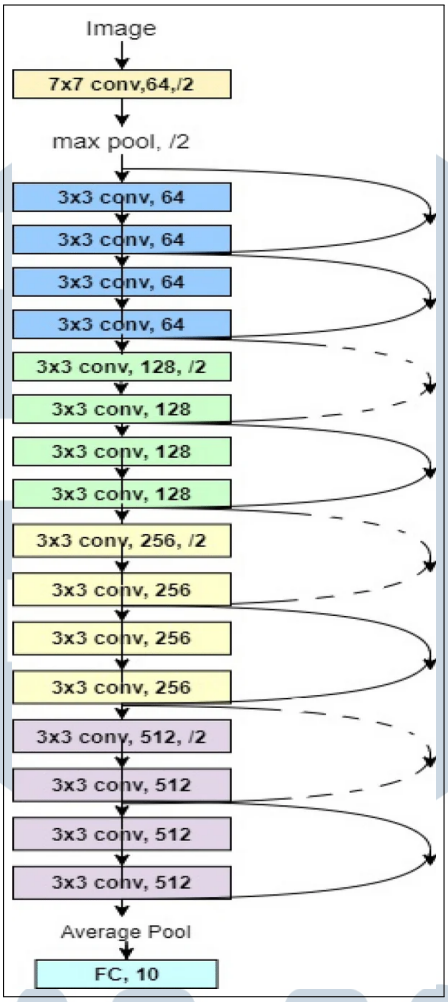
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m,\ j+n)\, K(m,n) \tag{2.1}$$

- I merupakan gambar masukan (*input image*) atau *feature map* dari lapisan sebelumnya.
- K merupakan *kernel* atau *filter* konvolusi yang digunakan untuk mengekstraksi fitur.
- $S(i, j)$ merupakan nilai keluaran (*output feature map*) pada posisi (i, j).
- m dan n merupakan indeks elemen pada *kernel*.

2.2 ResNet (*Residual Network*)

ResNet diperkenalkan oleh He et al. pada tahun 2016 sebagai solusi terhadap permasalahan degradasi performa pada jaringan yang semakin dalam [11]. Sebelum ResNet, penambahan kedalaman jaringan sering mengakibatkan penurunan akurasi *training* yang dikenal sebagai *degradation problem*. Masalah ini disebabkan oleh sulitnya mengoptimasi jaringan yang sangat dalam melalui metode *gradient descent* konvensional.

Inovasi utama ResNet adalah konsep *residual learning* melalui penggunaan *skip connection*. Pada arsitektur konvensional, setiap lapisan mempelajari pemetaan langsung dari masukan ke keluaran. Pada ResNet, setiap *residual block* mempelajari fungsi *residual*, yaitu selisih antara keluaran yang diinginkan dan masukan yang diteruskan langsung. Masukan tersebut kemudian ditambahkan langsung ke keluaran *block* melalui *skip connection*, sehingga jaringan dapat dengan mudah mempelajari fungsi identitas apabila lapisan tambahan tidak memberikan manfaat [11].



Gambar 2.1. Diagram arsitektur dari ResNet-18

Sumber: [15]

Gambar 2.1 menunjukkan ilustrasi arsitektur ResNet-18 yang digunakan pada penelitian ini. ResNet-18 merupakan salah satu varian ResNet yang terdiri dari 18 lapisan dengan sekitar 11,7 juta parameter. Arsitektur ini diawali dengan satu *convolutional layer* berukuran kernel 7×7 dan *stride* 2, dilanjutkan dengan *max pooling* berukuran 3×3, kemudian empat *residual stage* dengan konfigurasi blok $[2, 2, 2, 2]$. Setiap *stage* menggunakan jumlah filter berturut-turut sebesar 64, 128, 256, dan 512, kemudian diakhiri dengan *global average pooling* dan satu *fully connected layer* untuk proses klasifikasi [11]. ResNet-18 banyak digunakan sebagai *baseline* dalam berbagai penelitian karena memberikan keseimbangan yang baik antara akurasi dan kompleksitas komputasi.

Konsep *residual learning* pada ResNet dinyatakan melalui Rumus 2.2 yang diperkenalkan oleh He et al. pada tahun 2016 [11].

$$y = F(x, \{W_i\}) + x \tag{2.2}$$

Pada Rumus 2.2, x merupakan masukan (*input*) dari suatu *residual block*, sedangkan $F(x, \{W_i\})$ merupakan fungsi residual yang dipelajari oleh beberapa lapisan konvolusi dengan parameter atau bobot $\{W_i\}$. Keluaran akhir *block*, yaitu y, diperoleh dengan menjumlahkan hasil fungsi residual dengan masukan awal melalui *skip connection*. Dengan mekanisme ini, jaringan tidak perlu selalu mempelajari pemetaan baru secara langsung, tetapi cukup mempelajari perubahan (*residual*) yang diperlukan terhadap masukan. Pendekatan tersebut membantu mempermudah proses optimasi dan mengurangi permasalahan *vanishing gradient* pada jaringan yang dalam [11].

- x merupakan masukan (*input*) dari *residual block*.
- $F(x, \{W_i\})$ merupakan fungsi residual yang dipelajari oleh lapisan konvolusi dengan parameter $\{W_i\}$.
- $\{W_i\}$ merupakan himpunan bobot pada lapisan konvolusi di dalam *residual block*.
- y merupakan keluaran (*output*) dari *residual block*.

2.3 MobileNet

MobileNet merupakan keluarga arsitektur CNN yang dirancang khusus untuk aplikasi *mobile* dan *embedded vision* dengan fokus pada efisiensi komputasi dan ukuran model yang kecil [16]. Arsitektur ini diperkenalkan sebagai respons terhadap kebutuhan *deployment* model *deep learning* pada perangkat dengan keterbatasan sumber daya.

2.3.1 *Depthwise Separable Convolution*

Inovasi utama MobileNet adalah penggunaan *depthwise separable convolution* sebagai pengganti *standard convolution* [16]. Operasi ini memecah konvolusi standar menjadi dua tahap terpisah. Pertama, *depthwise convolution* menerapkan satu filter pada setiap *channel* masukan secara independen. Kedua,

pointwise convolution menggunakan filter 1×1 untuk mengkombinasikan hasil *depthwise convolution* antar *channel*.

Pemisahan ini secara drastis mengurangi jumlah parameter dan operasi komputasi dibandingkan *standard convolution*, dengan pengurangan hingga 8–9 kali lipat untuk kernel berukuran 3×3. Hal ini menjadikan MobileNet jauh lebih ringan tanpa kehilangan kemampuan ekstraksi fitur yang signifikan [16].

Pada *depthwise separable convolution*, proses konvolusi dibagi menjadi dua tahap, yaitu *depthwise convolution* dan *pointwise convolution*. Tahap *depthwise convolution* bertugas mengekstraksi fitur pada setiap kanal masukan secara independen menggunakan satu *kernel* untuk setiap kanal. Dengan demikian, proses ekstraksi fitur dapat dilakukan dengan jumlah operasi komputasi yang lebih sedikit dibandingkan konvolusi standar. Rumus 2.3 menunjukkan operasi *depthwise convolution* yang digunakan pada arsitektur MobileNet sebagaimana diperkenalkan oleh Howard et al. pada tahun 2017 [16].

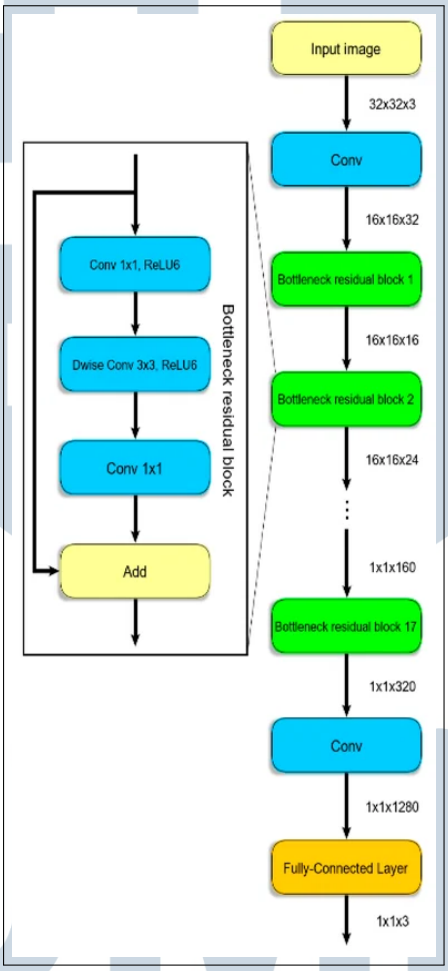
$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,\, l+j-1,\, m} \tag{2.3}$$

Pada Rumus 2.3, $\hat{K}$ menyatakan *depthwise convolution kernel* berukuran $D_K \times D_K \times M$, sedangkan F merupakan *input feature map*. Berbeda dengan konvolusi standar yang menggunakan satu *kernel* untuk seluruh kanal masukan, *depthwise convolution* menerapkan satu *kernel* tersendiri pada setiap kanal masukan. Hasil konvolusi pada setiap kanal kemudian membentuk *output feature map* $\hat{G}$ dengan jumlah kanal yang tetap sama seperti masukan. Pendekatan ini mampu mengurangi jumlah parameter dan operasi komputasi tanpa menghilangkan kemampuan model dalam mengekstraksi fitur spasial [16].

- $\hat{G}_{k,l,m}$ merupakan nilai keluaran (*output feature map*) pada posisi (k,l) untuk kanal ke-m.
- $\hat{K}_{i,j,m}$ merupakan *kernel depthwise convolution* pada kanal ke-m.
- $F_{k+i-1,l+j-1,m}$ merupakan nilai *input feature map* pada kanal ke-m.
- D_K merupakan ukuran *kernel* konvolusi.
- M merupakan jumlah kanal (*channels*) pada *input feature map*.

2.3.2 MobileNetV2

MobileNetV2 diperkenalkan oleh Sandler et al. pada tahun 2018 sebagai pengembangan dari MobileNetV1 [12]. MobileNetV2 memperkenalkan dua inovasi utama, yaitu *inverted residual structure* dan *linear bottleneck*.



Gambar 2.2. Diagram arsitektur dari MobileNetV2

Sumber: [12]

Pada Gambar 2.2 adalah ilustrasi/diagram dari arsitektur MobileNetV2. Pada *inverted residual block*, dimensi *channel* terlebih dahulu diperluas menggunakan *expansion layer* 1×1, kemudian *depthwise convolution* diterapkan pada representasi berdimensi tinggi tersebut, dan akhirnya dimensi direduksi kembali menggunakan *projection layer* 1×1. *Skip connection* ditambahkan antara masukan dan keluaran *block* apabila dimensinya sama [12].

Linear bottleneck menghilangkan fungsi aktivasi non-linear pada *projection*

layer terakhir setiap *block*. Hal ini didasarkan pada observasi bahwa fungsi aktivasi ReLU dapat merusak informasi ketika diterapkan pada representasi berdimensi rendah [12]. MobileNetV2 memiliki sekitar 3,4 juta parameter, jauh lebih sedikit dibandingkan ResNet-18, sehingga cocok untuk *deployment* pada perangkat dengan sumber daya terbatas.

2.4 *Neural Network Quantization*

Quantization adalah teknik kompresi model yang mengurangi presisi numerik representasi parameter dalam *neural network* [5]. Proses ini mengkonversi nilai *floating-point* presisi tinggi menjadi representasi dengan presisi lebih rendah, sehingga mengurangi ukuran model dan berpotensi mempercepat inferensi [5]. Menurut Jacob *et al.*, proses kuantisasi menggunakan pemetaan (*affine mapping*) antara nilai bilangan bulat hasil kuantisasi dengan nilai riil yang dinyatakan pada Persamaan 2.4.

$$r = S(q - Z) \tag{2.4}$$

Pada Persamaan 2.4, r merupakan nilai riil (*real value*) yang direpresentasikan dalam format *floating-point*, sedangkan q merupakan nilai hasil kuantisasi dalam bentuk bilangan bulat. Pada kuantisasi INT8, nilai q direpresentasikan sebagai bilangan bulat 8-bit. Parameter S (*scale*) adalah bilangan riil positif yang menentukan faktor skala untuk mengonversi nilai kuantisasi kembali ke domain riil, sedangkan parameter Z (*zero-point*) adalah bilangan bulat yang merepresentasikan nilai riil nol pada domain kuantisasi. Dengan adanya *zero-point*, nilai nol dapat direpresentasikan secara tepat sehingga operasi seperti *zero-padding* pada jaringan saraf dapat dilakukan tanpa menimbulkan kesalahan representasi [5].

Dalam skema ini, setiap array bobot (*weights*) maupun aktivasi (*activations*) menggunakan satu pasang parameter kuantisasi, yaitu *scale* (S) dan *zero-point* (Z), sedangkan array yang berbeda dapat memiliki pasangan parameter yang berbeda sesuai dengan rentang nilainya [5].

2.4.1 *Post-Training Quantization* (PTQ)

Post-Training Quantization (PTQ) merupakan metode *quantization* yang diterapkan pada model yang sudah terlatih tanpa memerlukan proses pelatihan ulang [5]. PTQ lebih efisien secara komputasi karena tidak memerlukan data pelatihan dalam jumlah besar maupun proses *training* yang panjang. Namun, PTQ dapat mengakibatkan penurunan akurasi yang lebih besar dibandingkan metode lain, terutama untuk tingkat presisi yang sangat rendah [7].

2.4.2 *Quantization-Aware Training* (QAT)

Quantization-Aware Training (QAT) mengintegrasikan simulasi *quantization* ke dalam proses pelatihan, memungkinkan model untuk beradaptasi dengan efek *quantization* selama pembelajaran [5]. QAT umumnya menghasilkan akurasi yang lebih tinggi dibandingkan PTQ karena model dapat mempelajari representasi yang lebih *robust* terhadap *quantization noise*. Namun, *trade-off* utama QAT adalah kebutuhan komputasi yang lebih besar karena memerlukan proses pelatihan ulang [4].

2.4.3 Format Representasi Numerik

Setiap nilai numerik dalam model *neural network* disimpan dalam format bit tertentu. Semakin banyak bit yang digunakan, semakin tinggi presisi nilai yang dapat direpresentasikan, namun semakin besar pula kebutuhan memori. Perbandingan ketiga format presisi yang digunakan dalam penelitian ini ditunjukkan pada Tabel 2.1.

Tabel 2.1. Perbandingan format representasi numerik

Format	Total Bit	Sign	Eksponen	Mantissa
FP32	32	1	8	23
FP16	16	1	5	10
INT8	8	1	–	–

A *Floating Point* 32-bit (FP32)

FP32 merupakan format representasi numerik standar yang digunakan dalam pelatihan dan inferensi *neural network* [17]. Format ini mengikuti standar

IEEE 754 dengan struktur 1 bit *sign*, 8 bit *exponent*, dan 23 bit *mantissa*, seperti yang ditunjukkan pada Persamaan 2.5.

$$x_{\text{FP32}} = (-1)^s \times 2^{e-127} \times (1+f) \tag{2.5}$$

- s merupakan bit *sign* (0 untuk positif, 1 untuk negatif).
- e merupakan nilai eksponen 8-bit (bias 127).
- f merupakan nilai *mantissa* 23-bit yang merepresentasikan bagian pecahan.

Format ini menyediakan presisi tinggi dan rentang dinamis yang luas, sehingga ideal untuk proses pelatihan yang membutuhkan stabilitas numerik. Namun, setiap parameter membutuhkan 32 bit atau 4 byte memori, yang menjadi kendala untuk *deployment* pada perangkat dengan memori terbatas.

B *Floating Point* 16-bit (FP16)

FP16 atau *half-precision floating-point* menggunakan 16 bit per nilai mengikuti standar IEEE 754, dengan struktur 1 bit *sign*, 5 bit *exponent*, dan 10 bit *mantissa* [17]. Nilai FP16 dinyatakan pada Persamaan 2.6.

$$x_{\text{FP16}} = (-1)^s \times 2^{e-15} \times (1+f) \tag{2.6}$$

- s merupakan bit *sign*.
- e merupakan nilai eksponen 5-bit (bias 15).
- f merupakan nilai *mantissa* 10-bit.

Konversi ke FP16 dilakukan melalui proses *type casting* yang sederhana tanpa memerlukan kalibrasi. Karena bit *mantissa* berkurang dari 23 menjadi 10, terjadi pembulatan nilai bobot yang menyebabkan sedikit penurunan presisi. FP16 dapat mengurangi ukuran model hingga 50% dengan degradasi akurasi yang minimal pada sebagian besar arsitektur CNN [8]. Keunggulan kecepatan FP16 umumnya terwujud pada GPU modern yang memiliki unit komputasi *half-precision*

secara *native*, seperti *Tensor Cores* pada GPU NVIDIA [17, 18]. Sebaliknya, pada CPU yang tidak memiliki dukungan *hardware* FP16 secara *native*, operasi FP16 dijalankan melalui emulasi yang justru menambah *overhead* komputasi dan dapat memperlambat inferensi [10].

C *Integer* 8-bit (INT8)

INT8 *quantization* merepresentasikan nilai menggunakan bilangan bulat 8-bit dengan rentang $[-128, 127]$ untuk format *signed* [5]. Berbeda dengan FP32 dan FP16 yang menggunakan format *floating-point*, INT8 menggunakan representasi bilangan bulat sehingga konversi dari FP32 ke INT8 memerlukan proses *mapping* menggunakan *scale factor* (S) dan *zero-point* (Z) seperti yang dinyatakan pada Persamaan 2.4. INT8 berpotensi mengurangi ukuran model hingga 75% dan memberikan percepatan inferensi pada *hardware* yang mendukung operasi *integer arithmetic* [5, 8].

Terdapat dua pendekatan utama dalam INT8 *quantization* menggunakan PyTorch. *Static quantization* mengkuantisasi seluruh lapisan termasuk *convolutional layers* dan memerlukan proses kalibrasi menggunakan data representatif. *Dynamic quantization* mengkuantisasi bobot secara statis namun mengkuantisasi aktivasi secara dinamis saat inferensi, dan dalam implementasi PyTorch standar hanya menargetkan lapisan *Linear* [5].

2.5 CIFAR-10 dan CIFAR-100

CIFAR-10 dan CIFAR-100 merupakan dua *benchmark dataset* standar yang banyak digunakan dalam penelitian *computer vision* [19]. Keduanya terdiri dari 60.000 gambar berwarna dengan resolusi 32×32 piksel, yang dibagi menjadi 50.000 gambar untuk data pelatihan dan 10.000 gambar untuk data pengujian. Dalam penelitian ini, CIFAR-10 digunakan sebagai *dataset* utama untuk mengevaluasi dampak kuantisasi terhadap akurasi, ukuran model, dan latensi inferensi. Sementara itu, CIFAR-100 digunakan sebagai *dataset* tambahan untuk menguji apakah pola hasil yang diperoleh tetap konsisten pada tugas klasifikasi dengan tingkat kesulitan yang lebih tinggi. Untuk melihat perbandingan karakteristik secara langsung, berikut adalah tabel perbandingan antara CIFAR-10 dan CIFAR-100 pada tabel 2.2

Tabel 2.2. Perbandingan karakteristik dataset CIFAR-10 dan CIFAR-100

Karakteristik	**CIFAR-10**	**CIFAR-100**
Jumlah gambar	60.000	60.000
Ukuran gambar	32×32 piksel	32×32 piksel
Jumlah kelas	10	100
Gambar per kelas	6.000	600
Data pelatihan	50.000	50.000
Data pengujian	10.000	10.000

Meskipun memiliki jumlah gambar yang sama, kedua *dataset* memiliki karakteristik yang berbeda. CIFAR-10 terdiri atas 10 kelas objek dengan masing-masing 6.000 gambar, sedangkan CIFAR-100 terdiri atas 100 kelas dengan hanya 600 gambar pada setiap kelas. Jumlah kelas yang lebih banyak dan jumlah sampel per kelas yang lebih sedikit menjadikan CIFAR-100 lebih menantang dibandingkan CIFAR-10 [19].

Gambar 2.3. Contoh gambar pada dataset CIFAR10
Sumber: [20]

Gambar 2.3 memperlihatkan contoh gambar atau data pada dataset CIFAR-10, yaitu *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, dan *truck*. Setiap kelas diwakili oleh tiga contoh gambar untuk menunjukkan variasi bentuk, warna, dan sudut pengambilan objek. Meskipun resolusi setiap gambar hanya sebesar 32×32 piksel, objek masih dapat dikenali dengan cukup baik sehingga CIFAR-10 banyak digunakan sebagai *benchmark* dalam penelitian klasifikasi citra.

Gambar 2.4. Contoh gambar pada dataset CIFAR100

Sumber: [20]

Gambar 2.4 menampilkan contoh gambar dari 10 kelas pertama pada dataset CIFAR-100 dengan tiga contoh gambar pada setiap kelas. Berbeda dengan CIFAR-10 yang hanya memiliki 10 kelas, CIFAR-100 terdiri atas 100 kelas objek yang lebih beragam dan spesifik. Variasi kelas yang lebih banyak, ditambah dengan jumlah sampel yang lebih sedikit pada setiap kelas, menyebabkan proses klasifikasi menjadi lebih menantang. Oleh karena itu, CIFAR-100 digunakan dalam penelitian ini untuk mengevaluasi apakah dampak kuantisasi yang diamati pada CIFAR-10 tetap konsisten pada dataset dengan tingkat kesulitan yang lebih tinggi.

Selain karakteristik dari data, perlu diketahui terlebih dahulu struktur data dari *dataset* yang digunakan dalam penelitian. Struktur data menjelaskan atribut atau *feature* yang dimiliki oleh setiap sampel pada *dataset*, beserta tipe data dan bentuk representasinya. Informasi ini penting karena menjadi dasar dalam proses pembacaan, transformasi, dan pemrosesan data sebelum digunakan pada tahap pelatihan maupun evaluasi model. Struktur data untuk dataset CIFAR-10 dan CIFAR-100 ditunjukkan pada Tabel 2.3 dan Tabel 2.4.

Tabel 2.3. Struktur data dataset CIFAR-10

Feature	**Class**	**Shape**	**Dtype**
id	Text	–	string
image	Image	$(32, 32, 3)$	uint8
label	ClassLabel	–	int64

Tabel 2.4. Struktur data dataset CIFAR-100

Feature	Class	Shape	Dtype
coarse_label	ClassLabel	–	int64
id	Text	–	string
image	Image	$(32, 32, 3)$	uint8
label	ClassLabel	–	int64

Berdasarkan Tabel 2.3, setiap sampel pada dataset CIFAR-10 terdiri atas tiga *feature*, yaitu `id`, `image`, dan `label`. *Feature* `id` merupakan identitas unik untuk setiap citra, sedangkan `image` menyimpan citra RGB berukuran 32×32 piksel dengan tiga kanal warna yang direpresentasikan dalam bentuk *unsigned 8-bit integer* (*uint8*). Selanjutnya, `label` merupakan penanda kelas yang bertipe *int64*, yang menunjukkan salah satu dari sepuluh kategori objek pada dataset CIFAR-10.

Sementara itu, berdasarkan Tabel 2.4, struktur data CIFAR-100 memiliki satu *feature* tambahan dibandingkan CIFAR-10, yaitu `coarse_label`. *Feature* ini merepresentasikan 20 *superclass* yang mengelompokkan 100 kelas objek ke dalam kategori yang lebih umum. Selain itu, CIFAR-100 juga memiliki `id`, `image`, dan `label` dengan karakteristik yang sama seperti pada CIFAR-10. Meskipun demikian, pada penelitian ini proses pelatihan, *fine-tuning*, dan evaluasi model hanya memanfaatkan `image` sebagai data masukan dan `label` sebagai target klasifikasi. `id` dan `coarse_label` tidak digunakan dalam proses eksperimen.

2.6 Gap Penelitian

Meskipun banyak penelitian telah dilakukan mengenai *quantization* pada CNN, masih terdapat *gap* dalam perbandingan sistematis mengenai dampak pengurangan presisi pada arsitektur dengan filosofi desain yang berbeda. Sebagian besar penelitian berfokus pada satu arsitektur atau satu metode *quantization* tertentu, sehingga menyulitkan penarikan kesimpulan umum mengenai *trade-off* antara efisiensi dan akurasi [7, 8]. Analisis komparatif yang terkontrol antara arsitektur berorientasi akurasi seperti ResNet dan arsitektur berorientasi efisiensi seperti MobileNet, menggunakan metode *quantization* yang identik dalam kondisi eksperimen yang sama, masih terbatas. Penelitian ini bertujuan untuk mengisi *gap* tersebut dengan analisis yang sistematis mengenai dampak pengurangan presisi terhadap kedua arsitektur di bawah kondisi eksperimen yang terkontrol.