BAB 2 LANDASAN TEORI

Citra adalah kumpulan dari potong-potongan kecil warna atau intensitas cahaya yang disebut dengan *pixel*. *Pixel-pixel* ini membentuk grid dua dimensi yang menghasilkan pola tertentu sehingga dapat dikenali sebagai suatu objek (misalnya objek kucing). Secara umum citra dapat dibagi ke dalam dua kategori berdasarkan jumlah *channel*, yaitu citra *grayscale* (satu *channel*) atau citra RGB (tiga *channel*). Setiap *pixel* dari setiap *channel* direpresentasikan dengan bilangan antara 0 dan 255, di mana 0 adalah hitam dan 255 adalah warna putih atau warna lain. Komputer "melihat" nilai dari setiap *pixel*, berbeda dengan manusia yang melihat citra sebagai kombinasi intensitas cahaya yang membentuk objek tertentu. Perbedaan bagaimana manusia melihat suatu citra dengan komputer "melihat" citra disebut dengan *semantic gap* yang dapat dilihat pada Gambar 2.1. Pada Gambar 2.1 atas menunjukan bagaimana manusia melihat citra dan bagian bawah menunjukan bagaimana komputer melihat citra [16].

Computer vision adalah suatu bidang ilmu komputer dengan metode tertentu untuk membuat komputer dapat memahami makna dari citra yang diterimanya. Memaknai sebuah citra tidak sulit bagi manusia, namun bagi sebuah komputer adalah suatu tugas yang tidak mudah [16]. Klasifikasi merupakan tugas utama *computer vision* dan algoritma *machine learning* untuk mengekstrak makna atau fitur dari citra agar komputer dapat memahami konten dari citra tersebut. Proses klasifikasi merupakan proses pemberian label terhadap suatu citra ke dalam kategori-kategori tertentu yang bisa dipahami oleh manusia. Dalam penelitian ini klasifikasi yang dilakukan adalah memberikan label terhadap citra *fundus* retina ke dalam lima kelas, yaitu 0-tidak ada DR, 1-DR ringan, 2-DR sedang, 3-DR parah, dan 4-PDR. Proses ekstraksi *fitur* dapat dilakukan dengan pendekatan teknik *machine learning* tradisional seperti *Local Binary Pattern* atau dengan pendekatan *deep learning*. Pendekatan *deep learning*, khususnya *Convolutional Neural Network*, memungkinkan komputer secara otomatis mengekstrak fitur dari citra yang digunakan untuk klasifikasi.

151	121	1	93	165	204	14	214	28	235
62	67	17	234	27	1	221	37	189	141
20	168	155	113	178	228	25	130	139	221
236	136	158	230	10	5	165	17	30	155
174	148	93	70	95	106	151	10	160	214
103	126	58	16	138	136	98	202	42	233
235	103	52	37	94	104	173	86	223	113
212	15	179	139	48	232	194	46	174	37
119	81	241	172	95	170	29	210	22	194
129	19	33	253	229	5	152	233	52	44
88	200	194	185	140	200	223	190	164	102
113	16	220	215	143	104	247	29	97	203
9	210	102	246	75	9	158	104	184	129
124	52	76	148	249	107	65	216	187	181
6	251	52	208	46	65	185	38	77	240
150	194	28	206	148	197	208	28	74	93
33	183	248	153	168	205	146	100	254	218
130	53	128	212	61	226	201	110	140	183
165	246	22	102	151	213	40	138	8	93
152	251	101	230	23	162	70	238	75	24
187	105	152	83	167	98	125	180	136	121
139	197	55	209	28	124	208	208	104	40
123	19	144	223	62	253	202	108	47	242
220	144	31	16	136	123	227	62	183	163

29	142	142	75	22	109	111	28	6	5
137	168	41	206	100	70	219	127	114	191
205	154	226	14	89	86	242	67	203	15
247	47	128	123	253	229	181	251	232	28
68	75	24	99	93	63	215	222	102	180
206	246	85	103	215	3	62	64	77	216
126	80	165	149	196	75	186	60	179	193
44	253	164	253	14	216	175	30	46	254
137	23	33	203	241	21	144	63	244	188
32	214	142	121	249	109	99	232	183	71
45	36	152	27	190	137	61	1	237	247
1	14	241	70	2	30	151	67	169	205
32	80	102	32	99	169	91	166	73	214
186	219	9	203	209	240	40	249	119	122
177	252	38	203	119	0	217	139	139	157
154	145	49	251	150	185	235	23	230	156
157	168	223	60	247	118	5	180	16	206
102	208	195	246	140	138	54	191	139	79
17	233	85	169	166	24	49	40	160	97
84	242	247	144	203	3	19	24	198	88
67	67	185	98	123	106	168	105	127	153
37	113	214	252	203	80	146	211	7	16
142	241	66	86	214	133	146	253	189	200
67	215	174	111	189	54	144	56	59	163

Gambar 2.1 *Semantic* gap manusia dengan komputer. [16]

2.1 *Convolutional Neural Network*

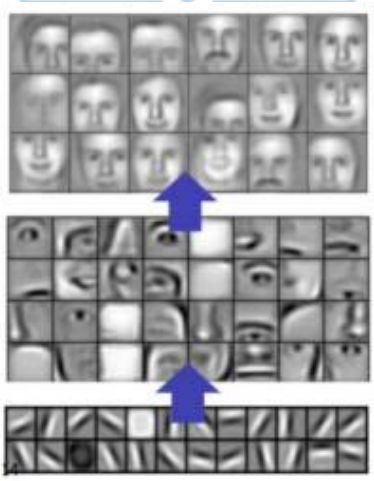
Convolutional Neural Networks (CNN) adalah *Neural Network* (NN) khusus untuk memproses data yang memiliki topologi berbentuk *grid* [17]. Data *time series* adalah data dengan topologi satu dimensi dan citra adalah data dengan topologi dua dimensi. Nama *Convolutional Neural Network* mengindikasikan bahwa *Neural Network* melakukan proses operasi matematika *convolution* yang merupakan operasi linear. *Convolutional Neural Network* secara sederhana adalah *Neural Network* yang menggunakan *convolution* sebagai operasi multiplikasi matrix setidaknya pada salah satu lapisan. CNN terdiri dari lapisan *convolution* dan *pooling* yang memungkinkan fitur diekstrak dari citra.

Convolutional Neural Network (CNN) atau yang sering juga dikenal dengan *ConvNet* merupakan arsitektur *deep feedforward* dan memiliki kemampuan generalisasi yang lebih baik jika dibandingkan dengan *Artificial Neural Network* (ANN). Konsep dari CNN terinspirasi dari cara kerja otak manusia untuk mengenali objek secara hierarki. CNN memiliki kemampuan yang cukup baik karena [18]:

1. CNN menggunakan konsep parameter *sharing* yang menghasilkan kemampuan generalisasi yang baik. Selain itu parameter *sharing* juga mengurangi jumlah parameter yang harus di-*training*.

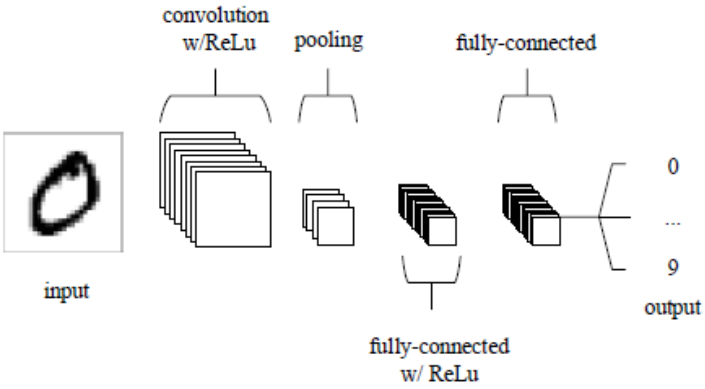
2. Tahap klasifikasi merupakan satu kesatuan dengan tahap ekstraksi fitur yang keduanya merupakan proses *learning*.
3. Model ANN lebih sulit untuk diimplementasikan pada model yang besar jika dibandingkan dengan CNN.

CNN sendiri banyak digunakan pada berbagai bidang karena kemampuannya yang sangat baik, seperti klasifikasi citra, deteksi objek, deteksi wajah, pengenalan suara, pengenalan kendaraan, pengenalan ekspresi wajah dan lainnya. Aspek penting ketika menerapkan CNN untuk menjalankan suatu tugas adalah 1) tidak boleh memiliki fitur yang bergantung secara spasial (*spatially dependent*) dan 2) untuk mendapatkan fitur abstrak ketika *input* diteruskan ke lapisan yang lebih dalam [19]. Sebagai contoh pada klasifikasi citra, lapisan pertama akan mendeteksi garis tepi, lapisan kedua akan mendeteksi bentuk sederhana seperti mata, hidung, telinga, dan pada lapisan yang lebih dalam akan mendeteksi fitur wajah yang bisa dilihat pada Gambar 2.2.



Gambar 2.2 CNN mengekstrak fitur citra secara hierarki [19]

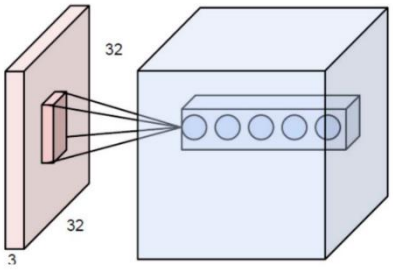
Arsitektur CNN terdiri atas empat komponen utama yang disusun sedemikian rupa sehingga membentuk arsitektur seperti LeNet, VGG-16, AlexNet, GoogLeNet dan lainnya. Komponen utama penyusun arsitektur CNN adalah lapisan *convolution*, lapisan *pooling*, lapisan *fully connected*, dan fungsi aktivasi seperti yang diilustrasikan pada Gambar 2.3.



Gambar 2.3 Arsitektur sederhana dari CNN [20]

2.1.1 **Lapisan *Convolutional***

Ide dari CNN adalah CNN menggunakan wilayah lokal pada sebuah citra, bukan menggunakan keseluruhan citra seperti yang bisa dilihat pada Gambar 2.4. Pada gambar tersebut tampak bahwa *hidden neuron* pada lapisan berikutnya hanya mendapatkan *input* dari bagian tertentu saja dari lapisan sebelumnya (*receptive field*). Ide lain dari CNN adalah menjaga bobot koneksi lokal yang tetap untuk seluruh *neuron* pada lapisan berikutnya [18]. Hal ini akan menghubungkan *neuron* yang bersebelahan pada lapisan berikutnya dengan bobot yang sama kepada wilayah lokal dari lapisan sebelumnya (*local connectivity*). Keuntungan dari koneksi ini adalah pengurangan jumlah parameter jika dibandingkan dengan ANN. CNN mengekstrak fitur dari citra dengan melakukan operasi *convolution* antara *kernel* dengan *input tensor* yang merupakan *subset array pixel* [14]. Produk perkalian *element wise* dari setiap *subset array pixel* dan *kernel* akan dijumlahkan untuk menghasilkan satu nilai numerik dalam *feature map*. CNN memberikan keuntungan untuk mendeteksi dan mengenali fitur terlepas dari posisinya pada citra.



Gambar 2.4 Operasi *convolution* bersifat 3D [19]

Kernel atau matriks *convolution* dikenal juga dengan istilah *filter* karena mereka bertindak seperti *filter* pada pengolahan citra digital. *Kernel* dapat ditambahkan setelah lapisan *input*, di mana setiap lapisan merupakan *kernel* yang berbeda sehingga CNN akan mengekstrak berbagai fitur dari citra *input*. Operasi *convolution* dapat dirumuskan dengan persamaan [18]:

$$a_{ij} = \sigma\big((W * X)_{ij} + b\big) \tag{2.1}$$

Di mana:

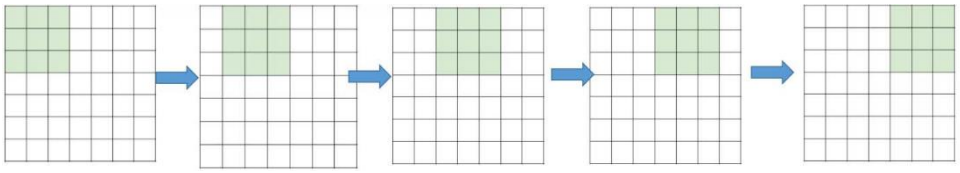
- a_{ij} adalah *output*
- X adalah *input* yang diberikan
- W adalah *filter / kernel / weight*
- b adalah bias
- $*$ merupakan operasi *convolution*
- σ adalah fungsi aktivasi, biasanya ReLU

Stride merupakan langkah dari *kernel* ketika melakukan operasi *convolution*. *Stride* mengatur *overlap* dari *kernel convolution* dan mengatur dimensi *output* dari operasi *convolution*. Pada Gambar 2.5 dapat dilihat citra *input* berukuran 7×7 . Jika *kernel* digeser dengan langkah atau *stride* 1 maka *output* akan berukuran 5×5. Namun jika *kernel* digeser dengan *stride* 2, maka *output* akan berukuran 3×3. Dimensi *output* dari proses *convolution* dengan berbagai ukuran *kernel* dan *stride* dapat dirumuskan dengan persamaan [19]:

$$O = 1 + \frac{N - F}{S} \tag{2.2}$$

Di mana:

- O adalah dimensi *output*
- N adalah dimensi *input*
- F adalah dimensi *kernel*
- S adalah nilai *stride*



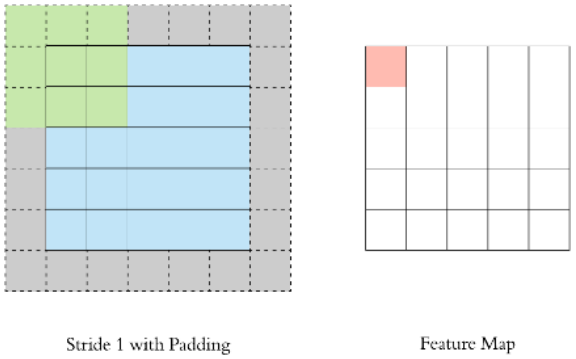
Gambar 2.5 Operasi *convolution* dengan *stride* 1, *kernel* 3 × 4 [19]

Salah satu kekurangan dari operasi *convolution* adalah hilangnya informasi jika informasi ini berada pada bagian tepi citra. Informasi yang berada pada bagian tepi citra hanya ditangkap satu kali ketika posisi *kernel* berada di bagian tepi citra. Cara sederhana namun efisien untuk mengatasi persoalan ini adalah dengan menggunakan *zero-padding*. *Zero-padding* juga mengatasi persoalan dimensi yang semakin kecil jika operasi *convolution* semakin dalam. Sebagai contoh jika citra *input* berukuran 7 × 7, *kernel* 3 × 3, *stride* 1, maka *output* akan berukuran 5 × 5. Namun jika ditambahkan *zero-padding*, maka *output* akan tetap berukuran 7 × 7. Ilustrasi *zero-padding* bisa dilihat pada Gambar 2.6. Dimensi *output* dengan *zero-padding* bisa dirumuskan dengan persamaan [19]:

$$O = 1 + \frac{N + 2P - F}{S} \tag{2.3}$$

Di mana:

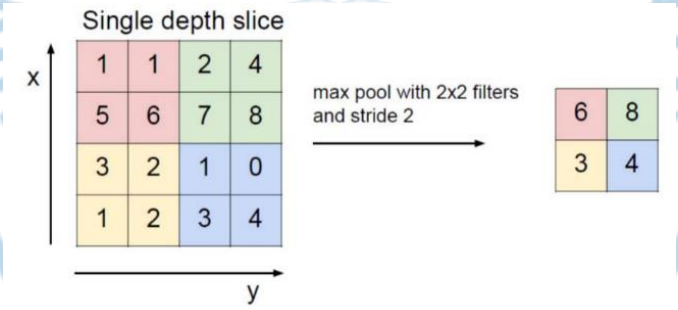
- O adalah dimensi *output*
- N adalah dimensi *input*
- F adalah dimensi *kernel*
- S adalah nilai *stride*
- P adalah ukuran *padding*



Gambar 2.6 Operasi *convolution* dengan zero-*padding* dan *kernel* 3 × 3 [21]

2.1.2 **Lapisan *Pooling***

Lokasi persis dari fitur menjadi kurang penting setelah berhasil dideteksi [18]. Oleh karena itu lapisan *convolution* akan diikuti dengan lapisan *pooling* atau lapisan *sub-sampling*. Keuntungan utama menggunakan teknik *pooling* adalah mengurangi kompleksitas *feature map* untuk lapisan yang berikutnya. Dalam domain pengolahan citra, *pooling* dapat diparalelkan dengan pengurangan resolusi. Terdapat dua metode *pooling* yang umum digunakan, yaitu *max pooling* dan *average pooling*. *Max pooling* membagi citra menjadi beberapa sub-bagian dan mengambil nilai maksimum dari sub-bagian tersebut. Sedangkan *average pooling* membagi citra menjadi beberapa sub-bagian dan mengambil nilai rata-rata dari sub-bagian tersebut. Operasi *max pooling* yang ditunjukkan oleh Gambar 2.7 mengambil nilai maksimum dari setiap sub-bagian citra yang ditandai dengan warna merah, hijau, kuning dan biru. Operasi *pooling* dilakukan dengan memilih ukuran *kernel* yang diterapkan pada matrix *input* untuk menghasilkan *output* berikutnya. Ukuran yang paling umum dari *max pooling* adalah *kernel* berukuran 2×2 dengan *strides* 2.



Gambar 2.7 Operasi *pooling* dengan menggunakan *kernel* berukuran 2x2 [22]

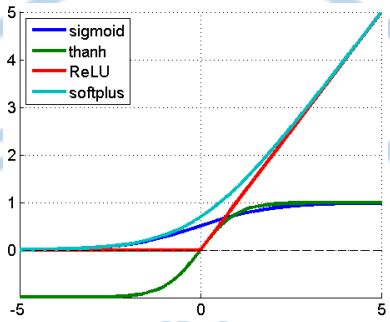
2.1.3 **Fungsi Aktivasi**

Setelah lapisan *convolution* biasanya dilanjutkan dengan lapisan *non-linearity*. Lapisan *non-linearity* digunakan untuk membuat *output* saturasi atau membatasi *output* yang dihasilkan. Lapisan ini lebih sering dikenal dengan fungsi aktivasi. Salah satu fungsi aktivasi yang sering digunakan adalah ReLU atau *Rectified Linear Units*. ReLU banyak digunakan karena beberapa hal [19]:

1. ReLU memiliki definisi yang sederhana baik fungsi dan gradiennya.
   - $ReLU(x) = max(0, x)$

- $\frac{d}{d_x} ReLU(x) = \{1\ if\ x > 0;\ else\ 0\}$

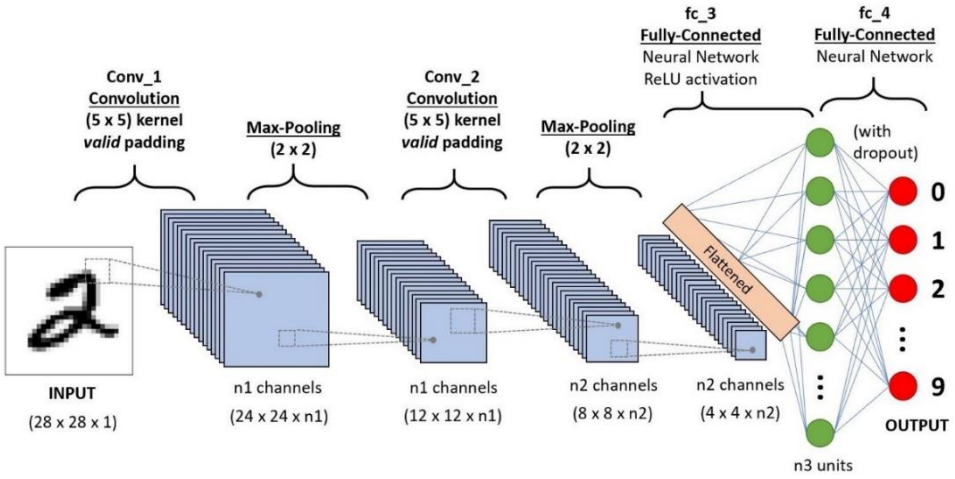
2. Fungsi saturasi seperti *sigmoid* dan *tanh* mengakibatkan permasalah saat *back propagation*. Semakin dalam desain *neural network* maka sinyal gradien mulai menghilang yang dikenal dengan *vanishing gradient*.
3. ReLU membuat representasi *sparse* karena 0 pada gradien menghasilkan nilai 0. Tidak demikian pada *sigmoid* atau *tanh* yang selalu memiliki hasil tidak 0 dari gradien.



Gambar 2.8 Karakter fungsi aktivasi [19]

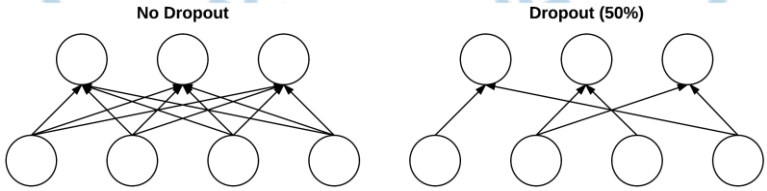
2.1.4 **Lapisan *Fully Connected***

Lapisan *fully connected* merupakan *neuron* yang disusun pada *Neural Network* tradisional. Setiap *node* dalam lapisan *fully connected* secara langsung terhubung dengan setiap *node* baik pada lapisan sebelum maupun setelahnya seperti yang ditunjukan pada Gambar 2.9. Dari gambar ini dapat dilihat bahwa setiap *node* dari lapisan *pooling* terakhir terhubung sebagai vektor ke lapisan pertama dari lapisan *fully connected*. Lapisan *fully connected* memiliki parameter terbanyak pada CNN dan membutuhkan waktu *training* yang lama [19].



Gambar 2.9 Lapisan *convolution* yang diikuti lapisan *fully connected* [23]

Kekurangan lapisan *fully connected* adalah memiliki parameter yang banyak sehingga membutuhkan komputasi kompleks pada sesi *training*. Teknik *dropout* dapat digunakan untuk mengurangi jumlah *node* dan koneksi *fully connected*. Teknik *dropout* mengeliminasi *node* secara acak dengan nilai probabilitas yang ditentukan, sehingga tidak semua *node* ikut dalam proses *forward* dan *backward propagation* untuk setiap *batch* data yang diproses. *Neuron* yang tergantung pada *neuron* yang dieliminasi akan didorong untuk belajar fitur paling kuat sehingga mengurangi *overfit* [19]. Pada Gambar 2.10 dapat dilihat konsep dari *dropout*. Dengan nilai probabilitas 0.5, *dropout* akan memutuskan koneksi di antara dua lapisan *fully connected* secara acak.



Gambar 2.10 Kiri: Dua layer *neural network* tanpa *dropout*. Kanan: dua lapisan *neural network* dengan *dropout* 50% [16]

Lapisan *fully connected* berperan sebagai *classifier* untuk mengklasifikasikan fitur dari citra yang sudah diekstrak ke dalam *k* kelas. Fungsi aktivasi pada bagian *output* dari lapisan *fully connected* biasanya menggunakan *softmax*. *Softmax* menghitung besarnya probabilitas satu *data point* menjadi

anggota dari setiap kelas yang mungkin. Jika nilai *output* atau *scoring function* yang memetakan *input* ke *output* dinyatakan dengan [16]:

$$s = f(x_i, W) \tag{2.4}$$

Di mana:

- s adalah nilai *output* atau nilai prediksi dari model
- x_i adalah data *input* ke i
- W adalah bobot dari model yang memetakan x_i kepada s

Maka *softmax* dari *scoring function* tersebut adalah [16]*:*

$$P(Y = k|X = x_i) = \left(\frac{e^{s_{yi}}}{\sum_j e^{s_j}}\right) \tag{2.5}$$

Di mana:

- $P(Y = k|X = x_i)$ adalah probabilitas x_i diklasifikasikan ke dalam kelas k
- s_{yi} adalah nilai *output* citra i untuk kelas yang sesuai dengan *ground truth*
- s_j adalah nilai *output* citra i untuk semua kelas yang mungkin

Loss function digunakan untuk mengukur seberapa bagus atau tidaknya *output* dari model. Umumnya *softmax* akan berpadanan dengan *loss function categorical cross entropy*. Nilai *loss* menghitung "jarak" antara nilai yang diprediksi oleh model dengan nilai yang seharusnya. Nilai *loss categorical cross entropy* dapat dihitung dengan [16]:

$$L_i = -log\left(\frac{e^{s_{yi}}}{\sum_j e^{s_j}}\right) \tag{2.6}$$

Di mana:

- L_i adalah nilai *loss data point i*
- s_{yi} adalah nilai *output data point* i untuk kelas yang sesuai dengan *ground truth*

- s_j adalah nilai *output data point i* untuk semua kelas yang mungkin

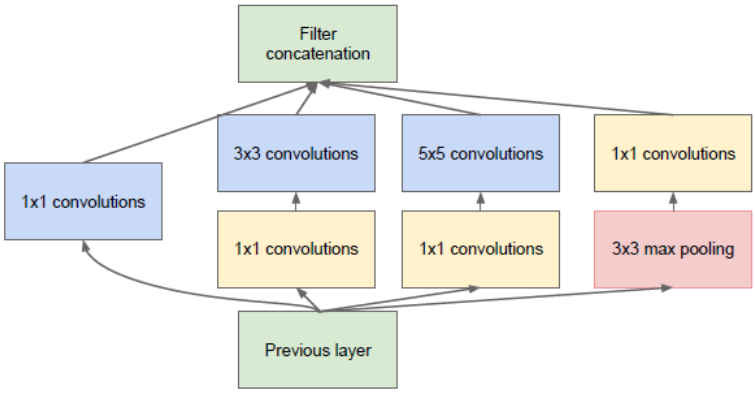
Jika seluruh komponen (*convolution, pooling, fully connected*, dan fungsi aktivasi) yang sudah dibahas disusun sedekian rupa, maka didapatkan arsitektur CNN. Jumlah dari masing-masing komponen dapat berbeda-beda, namun pada umumnya akan memiliki pola tertentu seperti lapisan *convolutional* diikuti dengan lapisan *pooling*, ataupun fungsi aktivasi, sedangkan lapisan *fully connected* biasanya berada pada bagian akhir dari CNN. Kombinasi dari jumlah dan susunan lapisan-lapisan ini akan membentuk berbagai arsitektur CNN seperti LeNet, AlexNet, VGG-16 ataupun GoogLeNet. Secara umum langkah dari CNN adalah sebagai berikut [18]:

1. Menyediakan vektor *input* untuk diteruskan ke CNN.
2. CNN melakukan operasi *convolution* dengan menggunakan *kernel* atau *filter* sehingga dihasilkan *feature map*.
3. *Feature map* tersebut diteruskan melalui ReLU untuk memperkenalkan *non-linearity*.
4. Melakukan operasi *pooling* pada *feature map*.
5. Ulangi tahap 2-4 sesuai dengan jumlah lapisan yang dimiliki.
6. *Feature map* dari tahap 5 (*feature map* terakhir) diteruskan ke lapisan *fully connected* untuk klasifikasi.
7. *Output* dari *fully connected* umumnya akan diikuti dengan fungsi aktivasi *softmax*.
8. Komputer menghitung nilai *error* antara nilai prediksi dengan nilai aktual.
9. Model akan diperbaharui bobotnya dengan *backpropagate* komponen *error* dari langkah 8.
10. Melakukan *forward pass* dan ulangi langkah 2-9 menggunakan parameter baru sampai nilai *error* konvergen.

2.2 **Inception v3**

Inception atau yang dikenal juga dengan GoogLe Net merupakan salah satu aristektur CNN yang dikembangkan oleh Christian Szegedy dan kawan-kawan [11], [24]. Arsitektur Inception v1 didesain untuk bekerja dengan baik dalam

kondisi komputasi yang terbatas [24]. Hal ini dimungkinkan dengan melakukan penggabungan *output* dari *kernel convolution berukuran* 1×1, 3×3, 5×5 dan *pooling* sambil menjaga tinggi dan lebar *output* dari setiap *kernel* tidak berubah [24], [25]. Lapisan *bottleneck* atau operasi *convolution* 1×1 selalu dilakukan sebelum operasi *convolution* 3×3 dan 5×5 untuk mengurangi jumlah operasi matematika sampai dengan faktor 10. Arsitektur Inception tersusun atas kumpulan modul Inception yang bisa dilihat pada Gambar 2.11. Arsitektur ini dikembangkan lebih lanjut oleh Christian Szegedy dan kawan-kawan menjadi Inception v3 [11].

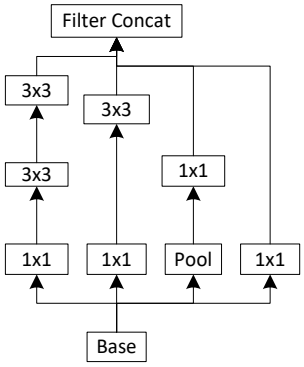


Gambar 2.11 Modul Inception [24]

2.2.1 **Modul Inception v3**

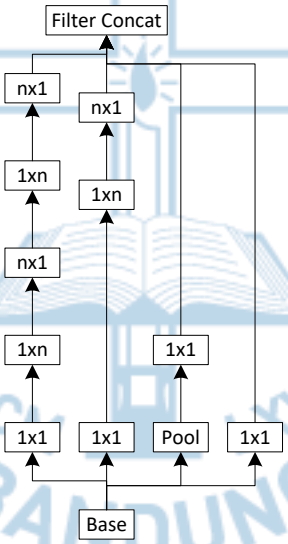
Arsitektur Inception v3 [11] memanfaatkan teknik faktorisasi *convolution* asimetris untuk mengurangi jumlah operasi matematika lebih lanjut. Modul Inception pada Gambar 2.11 dikembangkan lebih lanjut sehingga terdapat tiga tipe modul Inception [26]:

- Modul Inception A dapat dilihat pada Gambar 2.12: modul ini mengganti *convolution* 5×5 pada Inception v1 dengan 2 buah *convolution* 3×3. Jika ditinjau dari total operasi matematika, *convolution* 5×5 membutuhkan 25 operasi matematika, sedangkan 2 buah *convolution* 3×3 membutuhkan 18 operasi matematika.



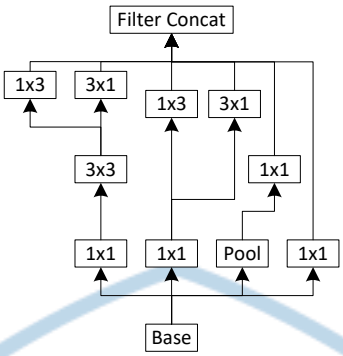
Gambar 2.12 Modul Inception A [11]

- Modul Inception B dapat dilihat pada Gambar 2.13: modul ini menggunakan teknik *convolution* asimetris 1×7 dan 7×1 sebagai ganti *convolution* 7×7.



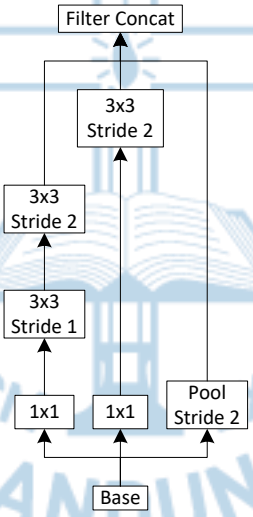
Gambar 2.13 Modul Inception B [11]

- Modul Inception C dapat dilihat pada Gambar 2.14: modul ini digunakan untuk mempromosikan representasi dimensi tinggi.



Gambar 2.14 Modul Inception C [11]

2.2.2 *Grid Size Reduction*

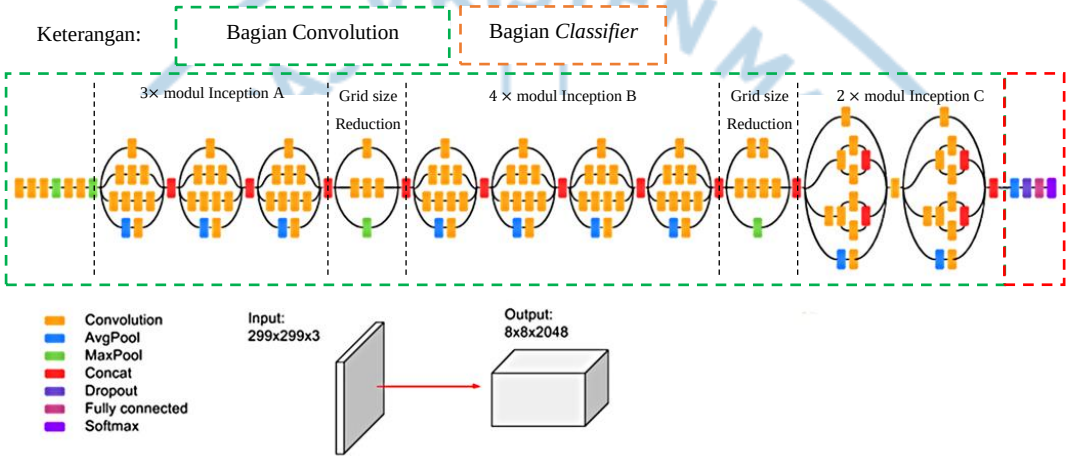


Gambar 2.15 Grid size reduction [11]

Umumnya *ConvNets* menggunakan operasi *pooling* untuk mengurangi ukuran *grid* dari *feature map*. Sebelum menerapkan operasi *pooling*, dimensi dari *kernel network* akan ditingkatkan [11]. Sebagi contoh *grid* berukuran $d \times d$ dengan k *filters*, jika mau sampai di *grid* $\frac{d}{2} \times \frac{d}{2}$ dengan $2k$ *filter* maka perlu melakukan *convolution stride* 1 dengan *filter* sebanyak $2k$ lalu dilanjutkan operasi *pooling*. Hal ini berarti total komputasi yang dibutukan adalah $2d^2k^2$. Jika operasi *pooling* ditukar dengan operasi *convolution*, maka didapati total komputasi $2\left(\frac{d}{2}\right)^2 k^2$.

Inception v3 menggunakan langkah paralel yang menggabungkan operasi *convolution* dan operasi *pooling* seperti pada Gambar 2.15 untuk mendapatkan jumlah *feature map* yang sama dengan operasi *convolution* diikuti *pooling* atau pun sebaliknya namun *Grid Size Reduction* memiliki beban komputasi yang lebih ringan.

Komponen-komponen tersebut (modul Inception v3 dan *grid size reduction*) disusun dengan komposisi yang bisa dilihat pada Gambar 2.16 sehingga membentuk satu kesatuan arsitektur Inception v3. Total parameter dari Inception v3 adalah 23,851,784. Jumlah parameter ini hanya 16.6% jika dibandingkan dengan jumlah parameter dengan arsitektur VGG-19 [27].



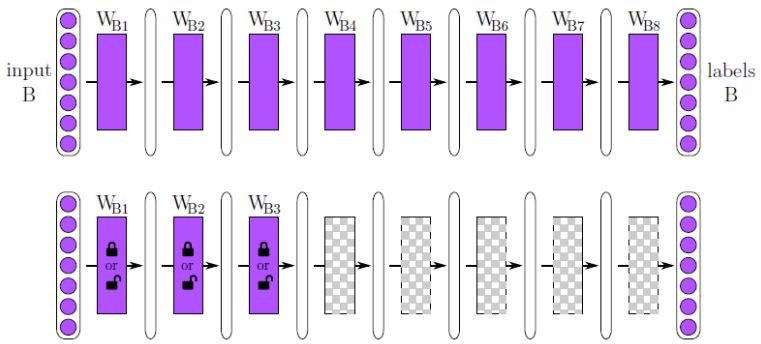
Gambar 2.16 Komponen penyusun Inception v3 [26]

2.3 *Transfer Learning*

Model CNN dapat di-*training* dengan dua cara, yaitu *end to end learning* atau *transfer learning*. Model CNN yang di-*training* dengan *end to end learning* adalah model CNN yang bobotnya diinisialisasi secara acak dan proses *backpropagation* dilakukan untuk memperbaharui bobot *neural network* sampai model ini dapat memberikan *output* klasifikasi dengan tingkat akurasi tertentu. *Transfer learning* berarti sebuah model mula-mula di-*training* dengan cara *end to end learning* menggunakan dataset dari domain tertentu lalu digunakan pada domain dataset yang diinginkan. Saat digunakan pada domain yang diinginkan, model perlu dilakukan *fine tuning* agar model disesuaikan dengan dataset yang baru.

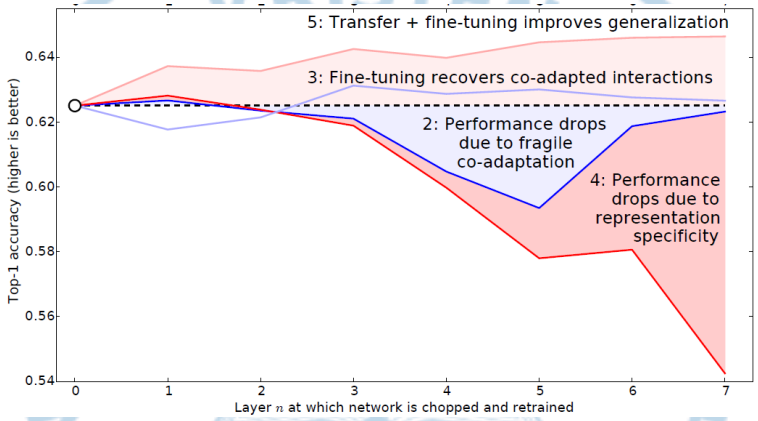
Dalam praktik *deep learning*, peneliti jarang *training* model CNN dengan pendekatan *end to end learning* karena dataset yang digunakan tidak selalu tersedia dalam jumlah yang cukup besar [22]. Lebih umum jika *training* model CNN dengan pendekatan *transfer learning.* Pada umumnya model *pre-trained* di-*training* dengan dataset dari ImageNet yang memiliki 1.2 juta gambar dengan 1.000 kategori. Model *pre-trained* ini lalu digunakan sebagai bobot inisialisasi model untuk tugas lain. Terdapat dua skenario dari *transfer learning*, yaitu [22]:

1. Model *pretrained* CNN sebagai pengekstrak fitur. CNN yang sudah di-*training* pada ImageNet digunakan dengan mengambil bagian *convolution* saja (bagian *classifier* atau *fully connected* tidak diambil) untuk mengekstrak fitur dari dataset. Hal ini diilustrasikan pada Gambar 2.17. Pada gambar tersebut, model mula-mula ditraining dengan *input* B, lalu model ini digunakan untuk dataset lain dengan mengambil sebagian lapisan (lapisan *convolution*) lalu ditambahkan lapisan baru dibagian akhir model (lapisan *fully connected*). Lapisan *fully connected* di-*training* sambil mempertahankan bobot pada lapisan *convolution*.
2. *Fine-tuning* CNN. Model dengan pendekatan *transfer learning* bisa juga dilakukan *fine tuning*. *Fine tuning* dapat dilakukan terhadap semua lapisan CNN atau cukup beberapa lapisan saja karena lapisan awal CNN memiliki fitur-fitur yang umum seperti deteksi garis tepi, deteksi kumpulan warna yang dapat diterapkan secara umum untuk citra apapun. Tapi berbeda dengan lapisan yang lebih dalam karena lapisan-lapisan ini mengekstrak fitur yang lebih spesifik dari sebuah citra [7].



Gambar 2.17 Ilustrasi proses *transfer learning* [7]

Jason Yosinski dan kawan-kawan [7] membuktikan bahwa model CNN yang diawali dengan *transfer learning* atau *transferred features* dapat meningkatkan performa CNN secara umum setelah dilakukan *fine tuning* terhadap dataset baru (lihat Gambar 2.18, garis 5 pada grafik). Pada Gambar 2.18 menunjukan bahwa lapisan awal (1, 2, 3) mengekstrak fitur umum dari citra, sedangkan lapisan akhir (5, 6, 7) mengekstrak fitur spesifik dari citra. Hal ini bisa menjadi kelebihan dan kekurangan dari *transfer learning* tergantung dari ukuran dataset baru apakah cukup untuk melakukan *fine tuning* jika dataset baru berukuran kecil dan dilakukan *fine-tune* terhadap semua lapisan CNN maka model cenderung memberikan hasil *overfit* [22].



Gambar 2.18 Percobaan Jason Yosinski [7]

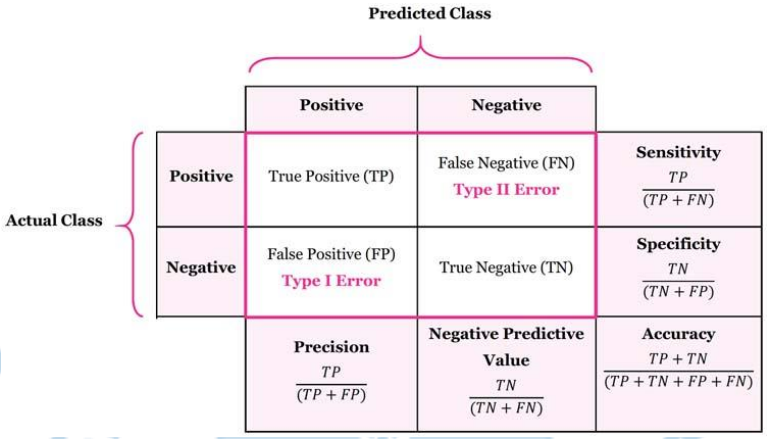
2.4 Metrik Pengukuran

Metrik yang digunakan untuk mengukur performa model berdasarkan *confusion matrix*. *Confusion matrix* adalah tabel yang mencatat jumlah kejadian antara dua nilai, yaitu klasifikasi aktual / *true* dan klasifikasi prediksi. Beberapa metrik bisa diperoleh berdasarkan tabel *confusion matrix* pada Gambar 2.19. Di antaranya adalah:

- *Precision*: $precision = \frac{TP}{TP+FP}$
- *Recall*: $recall = \frac{TP}{TP+FN}$
- *Accuracy*: $acc = \frac{TP+TN}{TP+TN+FP+FN}$
- F1-*score*: $F1 = 2 \times \frac{precision \times recall}{precission+recall}$

Di mana:

- TP (*true positive*) adalah jumlah kejadian klasifikasi positif benar.
- TN (*true negative*) adalah jumlah kejadian klasifikasi benar negatif.
- FP (*false positive*) adalah jumlah kejadian klasifikasi salah positif.
- FN (*false negative*) adalah jumlah kejadian klasifikasi salah negatif.



Gambar 2.19 Confusion matrix [28]

Precision dapat memberikan gambaran seberapa tinggi model itu dapat dipercaya ketika menyatakan seseorang *positive* dan memang *positive* [29]. Sedangkan *recall* mengukur kemampuan model memprediksi kelas *positive* atau kemampuan model untuk menemukan data *positive* dari dataset. Akurasi memberikan informasi mengenai berapa banyak data yang diklasifikasikan secara tepat oleh model. F1-*score* menghitung rata-rata harmonik dari *precision* dan *recall*. Nilai *precision* dan *recall* memiliki rentang dari 0 sampai 1.

Receiver operating characteristic curve (ROC) merepresentasikan grafik tingkat *true positive* dibandingkan dengan tingkat *false positive* atau dengan kata lain ROC menggambarkan relasi antara *sensitivity / recall* dengan $1 - specificity$. *Specificity* bisa dinyatakan dengan $\frac{TN}{TN+FP}$. *Area under ROC curve* (AUC) juga digunakan sebagai metrik performa. AUC menunjukan probabilitas bahwa *classifier* akan memberi peringkat kejadian positif yang dipilih secara acak lebih tinggi dari kejadian negatif yang dipilih secara acak. AUC mengambil nilai antara 0 dan 1, semakin tinggi nilai AUC, semakin baik performanya.