BAB 5: CNN-2: Variasi Fundamental Arsitektur CNN

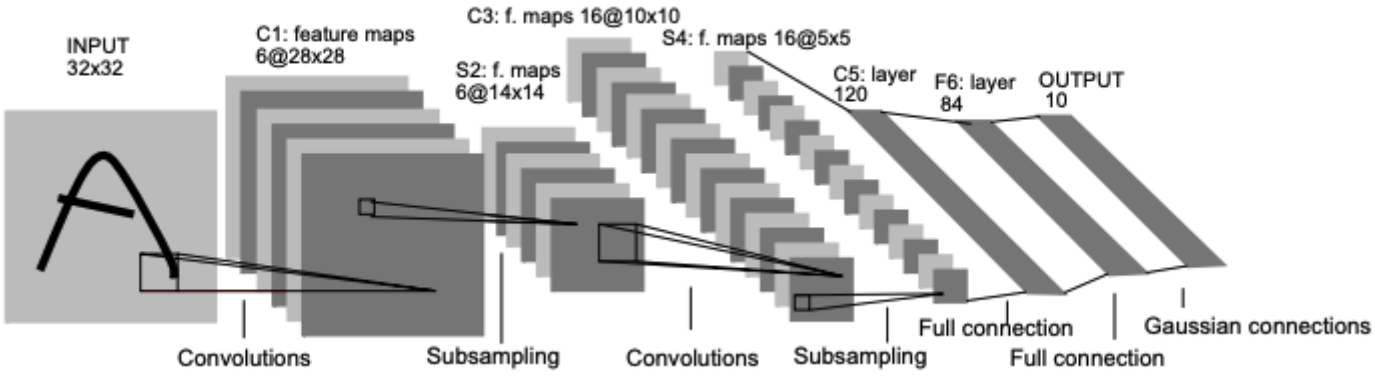
Selama bertahun-tahun, variasi arsitektur fundamental ini telah dikembangkan, menghasilkan kemajuan luar biasa di bidangnya. Ukuran Performance yang baik dari kemajuan ini adalah error rate dalam kompetisi seperti ILSVRC ImageNet Challenge. Dalam kompetisi ini, tingkat error lima besar untuk klasifikasi gambar turun dari lebih dari 26% menjadi kurang dari 2,3% hanya dalam enam tahun. Gambar dengan ukuran besar (tinggi 256 piksel) dan ada 1.000 kelas, beberapa di antaranya sangat halus (coba bedakan 120 ras anjing). Melihat evolusi dari awal pemenang dari challege arsitektur CNN adalah cara yang baik untuk memahami cara kerja CNN.

Pertama-tama kita akan melihat arsitektur LeNet-5 klasik (1998), kemudian tiga pemenang ILSVRC challenge: AlexNet (2012), GoogLeNet (2014), dan ResNet (2015).

LeNet-5

Arsitektur LeNet-5 mungkin merupakan arsitektur CNN yang paling banyak dikenal. Seperti disebutkan sebelumnya, itu dibuat oleh Yann LeCun pada tahun 1998 dan telah banyak digunakan untuk pengenalan digit tulisan tangan (MNIST). Ini terdiri dari layer yang ditunjukkan pada Tabel berikut.

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–



Gambar 1. LeNet5

> 1. Gambar MNIST berukuran 28 × 28 piksel, tetapi gambar tersebut diberikan padding nol p = 2 sehingga input image size menjadi 32 × 32 piksel dan dinormalisasi sebelum dimasukkan ke network. network lainnya tidak menggunakan padding apa pun, itulah sebabnya ukurannya terus menyusut seiring perjalanan gambar melalui network.
> 2. Average pooling layer sedikit lebih kompleks dari biasanya: setiap neuron menghitung rata-rata inputnya, kemudian mengalikan hasilnya dengan koefisien yang dipelajari (satu per peta) dan menambahkan bias yang dipelajari (sekali lagi, satu per peta), lalu akhirnya masuk ke fungsi aktivasi.
> 3. Sebagian besar neuron di peta C3 terhubung ke neuron hanya di tiga atau empat maps S2 (bukan semua enam maps S2). Lihat tabel 1 (halaman 8) di kertas asli10 untuk detailnya.
> 4. layer output agak istimewa: daripada menghitung perkalian matriks dari input dan vektor bobot, setiap neuron menghasilkan kuadrat jarak Euclidian antara vektor input dan vektor bobotnya. Setiap output mengukur berapa banyak gambar milik kelas digit tertentu. Fungsi cross-entropy cost sekarang lebih dipilih, karena lebih banyak mengukur prediksi buruk, menghasilkan gradien yang lebih besar dan konvergen lebih cepat.

Contoh implementasi arsitektur LeNet-5 : link 1 dan link 2

AlexNet

Arsitektur AlexNet CNN memenangkan ILSVRC ImageNet challenge 2012 dengan margin yang besar untuk error rate dengan pemenang keduanya: AlexNet mencapai lima besar error rate sebesar 17%, sementara yang terbaik kedua hanya mencapai 26%. Ini dikembangkan oleh Alex Krizhevsky

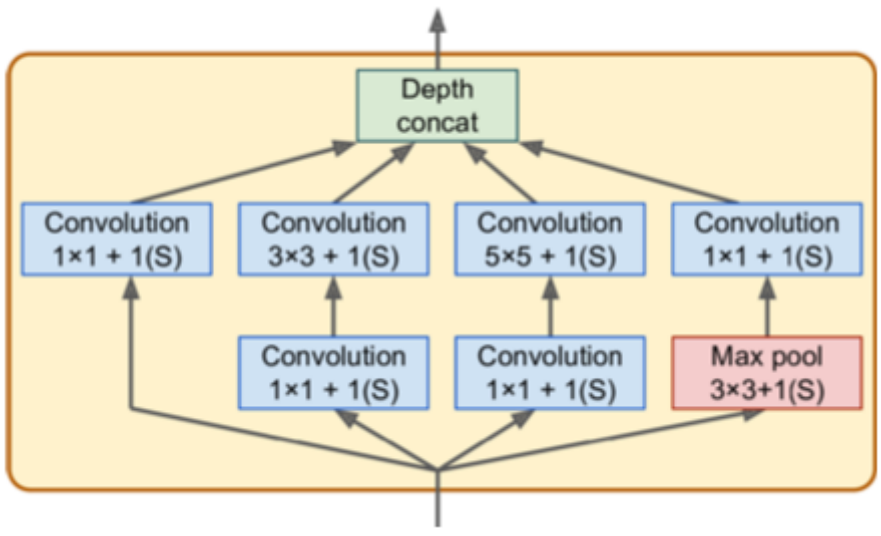
(karena itu namanya), Ilya Sutskever, dan Geoffrey Hinton. Ini mirip dengan LeNet-5, hanya jauh lebih besar dan lebih dalam, dan ini adalah yang pertama menumpuk covolutional leyer secara langsung di atas satu sama lain, daripada menumpuk pooling layer di atas setiap covolutional leyer. Tabel berikut menunjukan arsitektur AlexNet.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F9	Fully connected	–	4,096	–	–	–	ReLU
F8	Fully connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	`same`	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	`valid`	–
C3	Convolution	256	27 × 27	5 × 5	1	`same`	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	`valid`	–
C1	Convolution	96	55 × 55	11 × 11	4	`valid`	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

Untuk mengurangi overfitting, penulis menggunakan dua teknik regularisasi. Pertama, mereka menerapkan dropout dengan tingkat probabilitas dropout sebanyak 50% diantara training ke output layer F8 dan F9. Kedua, mereka melakukan augmentasi data dengan menggeser gambar pelatihan secara acak dengan berbagai offset, membaliknya secara horizontal, dan mengubah kondisi pencahayaan.

GoogleNet

Arsitektur GoogLeNet dikembangkan oleh Christian Szegedy et al. dari Google Research, dan memenangkan ILSVRC 2014 challenge dengan menekan Top-Five error rate di bawah 7%. Performa hebat ini sebagian besar berasal dari fakta bahwa Networknya jauh lebih dalam daripada CNN sebelumnya (seperti yang akan Anda lihat pada Gambar 3). Ini dikarenakan oleh subnetwork yang disebut modul inception, yang memungkinkan GoogLeNet menggunakan parameter jauh lebih efisien daripada arsitektur sebelumnya: GoogLeNet sebenarnya memiliki parameter 10 kali lebih sedikit daripada AlexNet (kira-kira 6 juta, bukan 60 juta). Gambar 2 menunjukkan arsitektur sebuah modul inception. Notasi “3 × 3 + 1(S)” berarti layer menggunakan kernel 3 × 3, stride 1, dan 'same' padding. Sinyal input pertama kali disalin dan dimasukkan ke empat layer berbeda. Semua convolutional layer menggunakan fungsi aktivasi ReLU. Perhatikan bahwa convolutional layer set kedua menggunakan ukuran kernel yang berbeda (1 × 1, 3 × 3, dan 5 × 5), memungkinkan mereka untuk menangkap pola pada skala yang berbeda. Perhatikan juga bahwa setiap layer menggunakan stride 1 dan 'same' padding (bahkan max pooling layer), sehingga semua keluarannya memiliki tinggi dan lebar yang sama dengan masukannya. Hal ini memungkinkan untuk menggabungkan semua keluaran di sepanjang dimensi kedalaman di depth concatenation layer (yaitu, menumpuk peta fitur dari keempat layer konvolusi teratas). layer gabungan ini dapat diimplementasikan di TensorFlow menggunakan tf.concat() operasi.



Gambar 2. Modul Inception

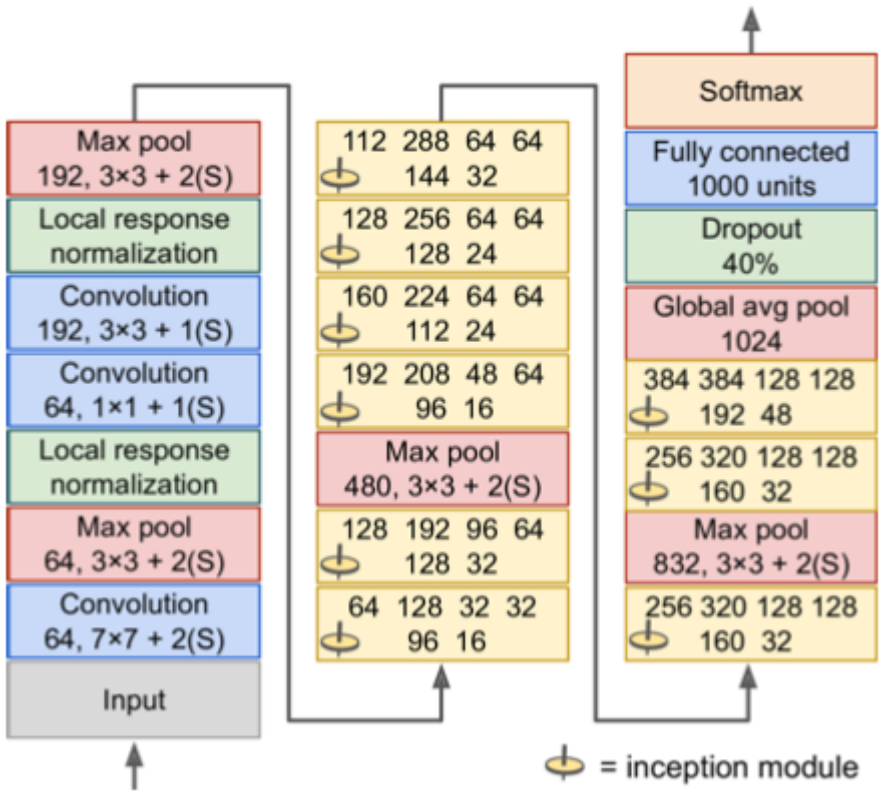
Tujuan dari modul inception adalah:

> 1. Meskipun layer tidak dapat menangkap pola spasial, mereka dapat menangkap pola sepanjang dimensi kedalaman.
> 2. Layer dikonfigurasi untuk menghasilkan lebih sedikit peta fitur daripada inputnya, sehingga Layer berfungsi sebagai bottleneck layer, yang berarti untuk mengurangi dimensi. Ini memotong biaya komputasi dan jumlah parameter, mempercepat pelatihan dan meningkatkan generalisasi.
> 3. Setiap pasang convolutional layer ([1 × 1, 3 × 3] dan [1 × 1, 5 × 5]) bertindak seperti satu convolutional layer yang kuat, yang mampu menangkap pola yang lebih kompleks. Memang, alih-alih menyapu pengklasifikasi linier sederhana di seluruh gambar (seperti yang dilakukan oleh convolutional layer tunggal), pasangan convolutional layer ini menyapu naural Network dua lapis di seluruh gambar. Sehinggal modul inception mampu menghasilkan peta fitur yang menangkap pola kompleks pada berbagai skala.

Sekarang mari kita lihat arsitektur dari GoogLeNet CNN (lihat Gambar 3). Jumlah keluaran feature maps oleh setiap convolution layer dan setiap pooling layer ditampilkan sebelum ukuran kernel. Arsitekturnya sangat dalam sehingga harus direpresentasikan dalam tiga kolom, tetapi GoogLeNet sebenarnya adalah satu tumpukan tinggi, termasuk sembilan modul inception (kotak dengan bagian atas yang berputar). Keenam angka dalam modul

inception mewakili jumlah keluaran feature maps oleh setiap convolution layer dalam modul (dalam urutan yang sama seperti pada Gambar 2). Perhatikan bahwa semua layer convolutional menggunakan fungsi aktivasi ReLU.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



Gambar 3. Arsitektur GoogleNet

Mari kita telusuri network GoogleNet diatas:

- Dua layer pertama membagi tinggi dan lebar gambar dengan 4 (sehingga luasnya dibagi 16), untuk mengurangi beban komputasi. layer pertama menggunakan ukuran kernel yang (7x7 dan 3 x3) sehingga banyak informasi yang disimpan.
- Kemudian local response normalization layer memastikan bahwa layer sebelumnya mempelajari berbagai macam fitur.
- Dua layer convolution mengikuti, di mana yang pertama bertindak seperti layer bottleneck. Seperti yang dijelaskan sebelumnya, kita dapat menganggap pasangan layer convolution ini sebagai satu layer convolutional yang lebih cerdas. (sebagai dimantional reduction dari input sebelumnya).
- Sekali lagi, local response normalization layer memastikan bahwa layer sebelumnya menangkap berbagai macam pola.
- Selanjutnya, sebuah Max pooling layer mengurangi tinggi dan lebar gambar sebanyak 2, sekali lagi untuk mempercepat perhitungan.
- Kemudian muncul tumpukan tinggi sembilan modul inception, diselingi dengan beberapa Max pooling layer untuk mengurangi dimensi dan mempercepat network.
- Selanjutnya, rata-rata global pooling layer menampilkan rata-rata dari setiap feature map: ini menghilangkan informasi spasial yang tersisa, yang baik karena tidak banyak informasi spasial yang tersisa pada titik tersebut. Memang, gambar masukan GoogLeNet biasanya berukuran 224 × 224 piksel, jadi setelah 5 Max pooling layer, masing-masing tinggi dan lebar dibagi dengan 2, feature map turun menjadi 7 × 7. Selain itu, ini adalah tugas klasifikasi, bukan lokalisasi, jadi tidak masalah di mana objeknya berada. Berkat pengurangan dimensi yang dibawa oleh layer ini, tidak perlu memiliki beberapa layer yang terhubung sepenuhnya di bagian atas CNN (seperti di AlexNet), dan ini sangat mengurangi jumlah parameter dalam jaringan dan membatasi risiko overfitting.
- layer terakhir cukup jelas: dropout untuk regularisasi, kemudian layer yang terhubung penuh dengan 1.000 unit (karena ada 1.000 kelas) dan fungsi aktivasi softmax untuk menghasilkan estimasi probabilitas kelas.

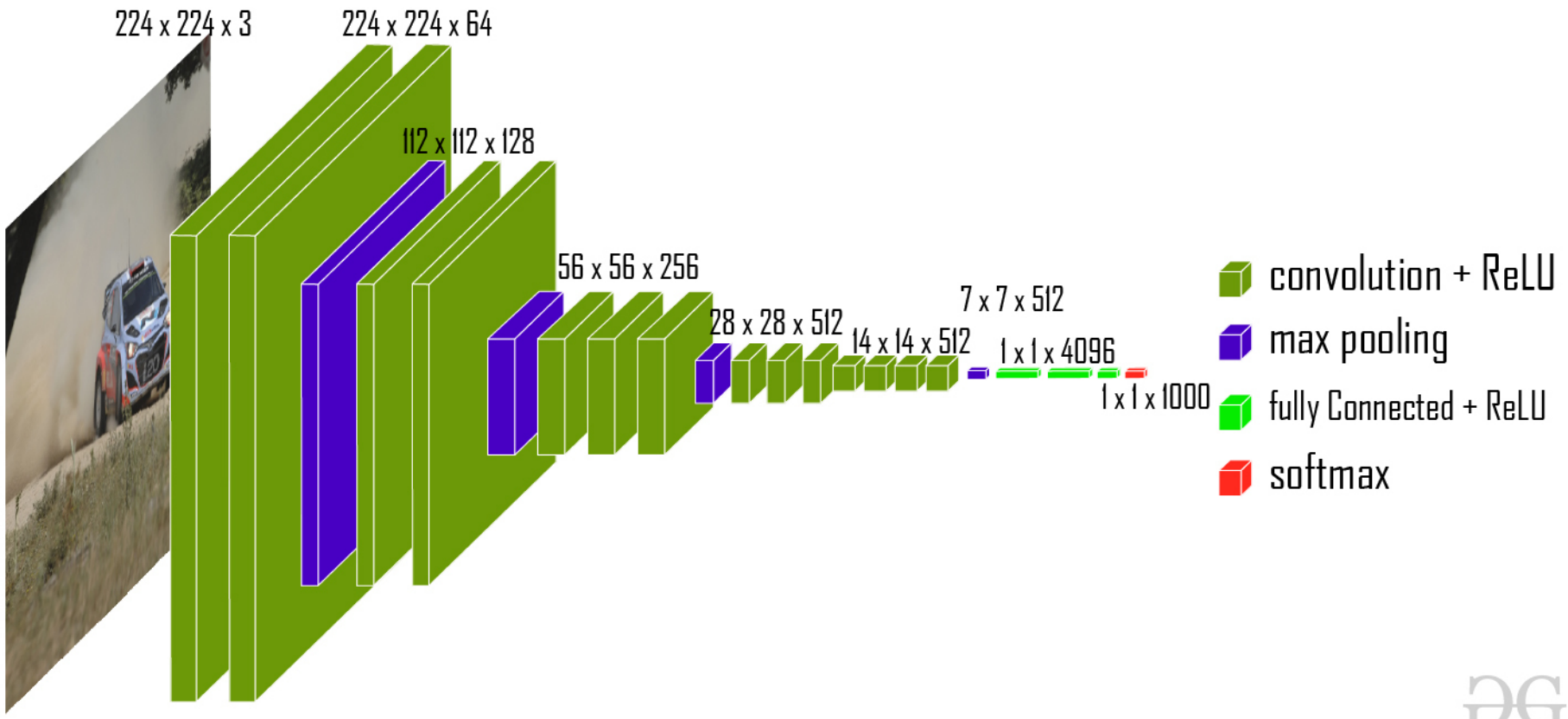
Diagram ini sedikit disederhanakan: arsitektur asli GoogLeNet juga menyertakan dua pengklasifikasi tambahan yang terpasang di atas modul inception ketiga dan keenam. Keduanya terdiri dari satu average pooling layer, satu layer convolutional, dua fully connected layers dan softmax activation layer. Selama proces training, loss yang dihasilkan (diperkecil hingga 70%) ditambahkan ke loss keseluruhan. Tujuannya adalah untuk mengatasi masalah gradien menghilang dan mengatur network. Namun, belakangan diketahui bahwa efeknya relatif kecil.

Beberapa varian arsitektur GoogLeNet kemudian diusulkan oleh peneliti Google, termasuk Inception-v3 dan Inception-v4, menggunakan modul inception yang sedikit berbeda dan mencapai kinerja yang lebih baik lagi.

VGGNet

Runner-up dalam tantangan ILSVRC 2014 adalah VGGNet, yang dikembangkan oleh Karen Simonyan dan Andrew Zisserman dari lab penelitian Visual Geometry Group (VGG) di Universitas Oxford. Itu memiliki arsitektur yang sangat sederhana dan klasik, dengan 2 atau 3 convolutional layer dan pooling layer, kemudian lagi 2 atau 3 convolutional layer dan pooling layer, dan seterusnya (mencapai total hanya 16 atau 19 convolutional layer r, tergantung pada varian VGG), ditambah dense network akhir dengan 2 hidden layer dan output layer. ini hanya menggunakan 3 × 3 filter, tetapi banyak filter. Berikut tabel arsitektur VGG 16 (C dan D) dan 19(E).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

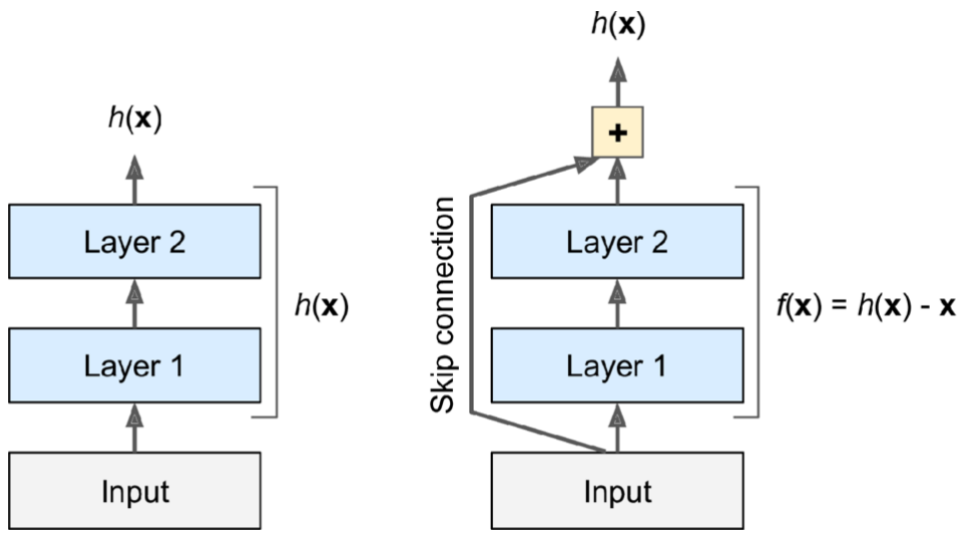


Gambar 4. VGG 16

ResNet

Kaiming He dkk. memenangkan ILSVRC 2015 challage menggunakan Residual Network (atau ResNet), yang memberikan top five error rate yang di bawah 3,6%. Varian network yang menang ini menggunakan CNN yang sangat dalam yang terdiri dari 152 layer (varian lain memiliki 34, 50, dan 101 layer). Ini menegaskan tren umum: model semakin dalam dan dalam, dengan parameter yang semakin sedikit. Kunci untuk dapat melatih network yang dalam adalah dengan menggunakan skip connection (juga disebut shortcut connection): sinyal yang masuk ke dalam layer juga ditambahkan ke output dari layer yang terletak sedikit lebih tinggi dari tumpukan. Mari kita lihat mengapa ini berguna.

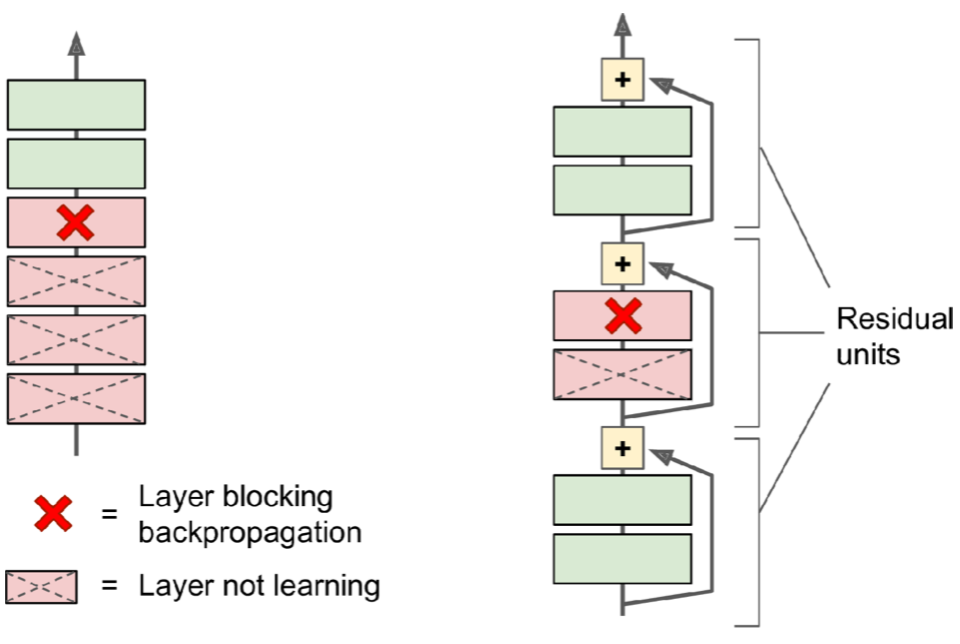
Saat melatih neural network, tujuannya adalah membuatnya memodelkan fungsi target h(x). Jika Anda menambahkan input x ke output jaringan (yaitu, Anda menambahkan skip connection), maka network akan dipaksa untuk memodelkan f(x) = h(x) – x daripada h(x). Ini disebut pembelajaran residual (pada gambar 5).



Gambar 5. Residual Learning

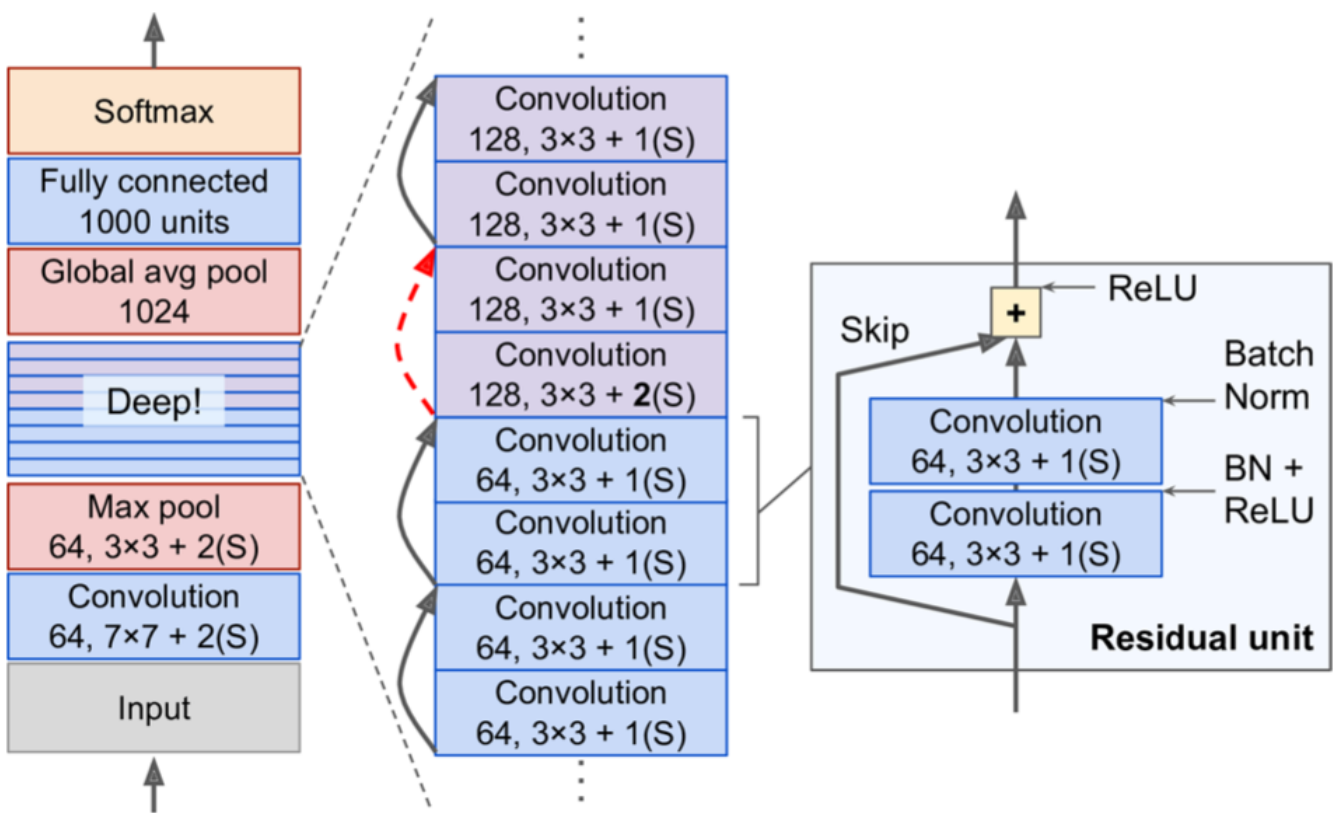
Saat Anda menginisialisasi neural network yang biasanya, bobotnya mendekati nol, sehingga network hanya menghasilkan nilai yang mendekati nol. Jika Anda menambahkan sebuah skip connection, network yang dihasilkan hanya menampilkan salinan inputnya; dengan kata lain, ini awalnya memodelkan fungsi identitas (sama dengan layer sebelum input). Jika fungsi target cukup dekat dengan fungsi identitas (yang sering terjadi), ini akan sangat mempercepat training.

Selain itu, jika Anda menambahkan banyak skip connection, network dapat mulai membuat kemajuan meskipun beberapa layer belum mulai belajar (lihat Gambar 6). Berkat skip connection, sinyal dapat dengan mudah melewati seluruh network. Residual network yang dalam dapat dilihat sebagai tumpukan unit residual (RU), di mana setiap unit residual adalah neural network kecil dengan skip connection.



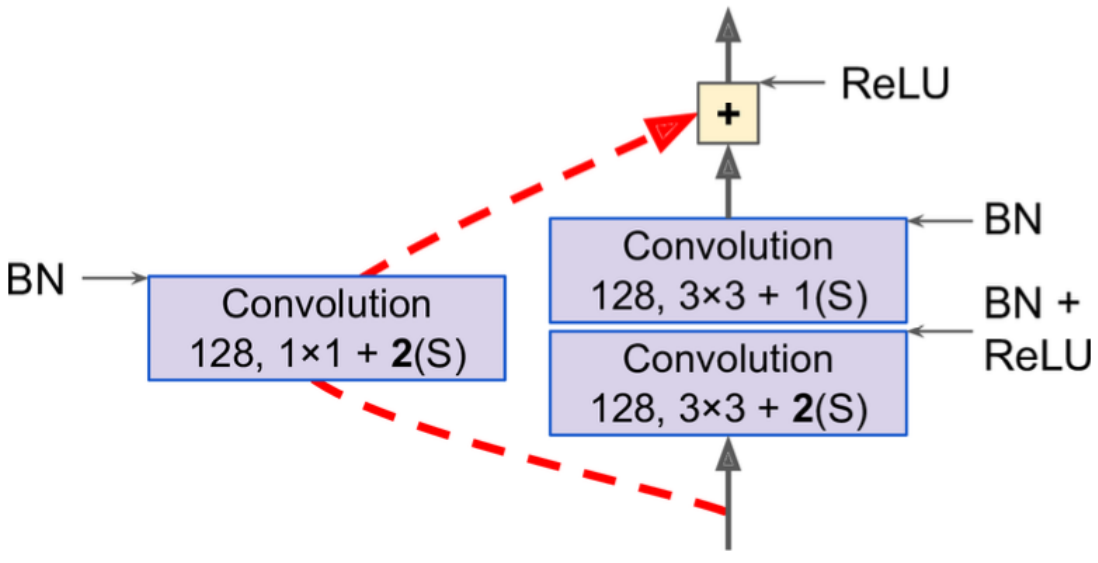
Gambar 6. Regular deep neural network (kiri) and deep residual network (kanan)

Sekarang mari lihat arsitektur ResNet (lihat Gambar 7). Ini sangat sederhana. Itu dimulai dan diakhiri persis seperti GoogLeNet (kecuali tanpa layer dropout), dan di antaranya hanya ada tumpukan unit residu sederhana yang sangat dalam. Setiap unit residu terdiri dari dua convolutional layer (dan tidak ada pooling layer!), Dengan Normalisasi Batch (BN) dan aktivasi ReLU, menggunakan kernel 3 × 3 dan mempertahankan dimensi spasial (straide 1, padding 'sama').



Gambar 7. Arsitektur Resnet

Perhatikan bahwa jumlah feature map digandakan setiap beberapa unit residual, pada saat yang sama tinggi dan lebarnya dibelah dua (menggunakan layer convolutional dengan stride 2). Ketika ini terjadi, input tidak dapat ditambahkan langsung ke output unit residual karena bentuknya tidak sama (misalnya, masalah ini memengaruhi skip connection yang diwakili oleh panah putus-putus pada Gambar 7). Untuk mengatasi masalah ini, masukan dilewatkan melalui layer convolutional 1 × 1 dengan stride 2 dan jumlah feature map output yang sesuai (lihat Gambar 8).



Gambar 8. Skip connection ketika mengubah fature map size dan depth

ResNet-34 adalah ResNet dengan 34 layer (hanya menghitung layer convolutional dan fully connected layer) berisi 3 unit residual yang menghasilkan 64 feature map, 4 RU dengan 128 maps, 6 RU dengan 256 maps, dan 3 RU dengan 512 maps . Dapat dilihat pada tabel berikut. Sedangkan ResNet yang lebih dalam dari itu adalah ResNet-152 yang menggunakan unit residu yang sedikit berbeda. Daripada menggunakan dua layer convolutional 3 × 3 dengan 256 peta fitur, mereka menggunakan tiga convolutional layer: layer pertama konvolusional 1 × 1 dengan hanya 64 feature map (4 kali lebih sedikit), yang bertindak sebagai layer penghambat (bottleneck layer), lalu layer 3 × 3 dengan 64 feature map, dan akhirnya layer konvolusional 1 × 1 lainnya dengan 256 peta fitur (4 kali 64) yang memulihkan kedalaman aslinya. ResNet-152 berisi 3 RU yang menghasilkan 256 maps, lalu 8 RU dengan 512 maps, 36 RU dengan 1.024 maps, dan akhirnya 3 RU dengan 2.048 maps. Semua perbandingan Arsitektur ResNet dapat dilihat pada tabel berikut.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Ada beberapa variasi Arsitektur CNN yang sudah ada dengan bentuk network yang sudah dikembangkan. Berikut arsitektur CNN lainnnya:

> 1. Xceptoion
> 2. SENet
> 3. MobileNet
> 4. EfficientNet
> 5. NasNet, dan lain-lain.

Dokumentasi seluruh arsitektur CNN di TansorFlow : link 1

Implementasi sebuah ResNet-34 CNN dengan Keras

Sebagian besar arsitektur CNN yang dijelaskan sejauh ini cukup mudah diterapkan (walaupun umumnya Anda akan memuat network yang telah dilatih sebelumnya, seperti yang akan kita lihat). Untuk mengilustrasikan prosesnya, mari terapkan ResNet-34 dari awal menggunakan Keras. Pertama, mari buat layer sebuah Unit Residual:

In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models, losses, Model, activations
```

In [2]:
```
class ResidualUnit(layers.Layer):
    def __init__(self, filters, strides=1, activation = "relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = activations.get(activation)
        self.main_layers = [
            layers.Conv2D(filters, 3, strides = strides, padding = "same", use_bias = False),
            layers.BatchNormalization(),
            self.activation,
            layers.Conv2D(filters, 3, strides = 1, padding = "same", use_bias = False),
            layers.BatchNormalization()]
```

```
        self.skip_layers = []
        if strides > 1 :
                self.skip_layers = [
                    layers.Conv2D(filters, 1, strides = strides, padding = "same", use_bias = False),
                    layers.BatchNormalization()]

    def call(self, inputs):
        Z =  inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.main_layers:
            skip_Z = layer(skip_Z)
        return self. activation(Z + skip_Z)
```

Seperti yang Anda lihat, kode ini sangat cocok dengan Gambar 8. Dalam konstruktor, kita membuat semua layer yang kita perlukan: layer utama adalah layer di sisi kanan diagram, dan layer skip adalah layer di sebelah kiri (hanya diperlukan jika langkahnya lebih besar dari 1). Kemudian di call() metode, kami membuat input melewati layer utama dan melewati layer (jika ada), lalu kami menambahkan kedua keluaran dan menerapkan fungsi aktivasi.

Selanjutnya, kita dapat membuat ResNet-34 menggunakan model, karena ini sebenarnya hanyalah rangkaian layer yang panjang (kita dapat memperlakukan setiap unit sisa sebagai satu layer sekarang karena kita memiliki kelas ResidualUnit):

In [3]:
```
model = models.Sequential()
model.add(layers.Conv2D(64, 7, strides = 2, input_shape = [28,28,1],padding="same",use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=3, strides = 2, padding = "same"))
prev_filters = 64
for filters  in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides =  1 if filters ==  prev_filters else 2
    model.add(ResidualUnit(filters, strides= strides))
    prev_filters = filters
model.add(layers.GlobalAvgPool2D())
model.add(layers.Flatten())
model.add(layers.Dense(10, activation = "softmax"))
```

```
Metal device set to: Apple M1

systemMemory: 8.00 GB
maxCacheSize: 2.67 GB
```

Satu-satunya bagian yang sedikit rumit dalam kode ini adalah loop yang menambahkan layer ke model: seperti yang dijelaskan sebelumnya, 3 RU pertama memiliki 64 filter, kemudian 4 RU berikutnya memiliki 128 filter, dan seterusnya. Kemudian kita atur langkahnya menjadi 1 jika jumlah filternya sama dengan di RU sebelumnya, atau kita atur menjadi 2. Kemudian kita tambahkan , dan terakhir kita perbarui prev_filters.

Sungguh menakjubkan bahwa dalam kurang dari 40 baris kode, kami dapat membuat model yang memenangkan tantangan ILSVRC 2015! Ini menunjukkan keanggunan model ResNet dan ekspresi dari Keras API. Menerapkan arsitektur CNN lainnya tidak jauh lebih sulit. Namun, Keras hadir dengan beberapa arsitektur bawaan ini, jadi mengapa tidak menggunakannya saja?

In [4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        3136

 batch_normalization (BatchN  (None, 14, 14, 64)       256
 ormalization)

 activation (Activation)     (None, 14, 14, 64)        0

 max_pooling2d (MaxPooling2D  (None, 7, 7, 64)         0
 )

 residual_unit (ResidualUnit  (None, 7, 7, 64)         74240
 )

 residual_unit_1 (ResidualUn  (None, 7, 7, 64)         74240
 it)

 residual_unit_2 (ResidualUn  (None, 7, 7, 64)         74240
 it)

 residual_unit_3 (ResidualUn  (None, 4, 4, 128)        222208
 it)

 residual_unit_4 (ResidualUn  (None, 4, 4, 128)        295936
 it)

 residual_unit_5 (ResidualUn  (None, 4, 4, 128)        295936
 it)

 residual_unit_6 (ResidualUn  (None, 4, 4, 128)        295936
 it)

 residual_unit_7 (ResidualUn  (None, 2, 2, 256)        886784
 it)

 residual_unit_8 (ResidualUn  (None, 2, 2, 256)        1181696
 it)

 residual_unit_9 (ResidualUn  (None, 2, 2, 256)        1181696
 it)

 residual_unit_10 (ResidualU  (None, 2, 2, 256)        1181696
 nit)

 residual_unit_11 (ResidualU  (None, 2, 2, 256)        1181696
 nit)

 residual_unit_12 (ResidualU  (None, 2, 2, 256)        1181696
 nit)

 residual_unit_13 (ResidualU  (None, 1, 1, 512)        3543040
 nit)

 residual_unit_14 (ResidualU  (None, 1, 1, 512)        4722688
 nit)

 residual_unit_15 (ResidualU  (None, 1, 1, 512)        4722688
 nit)

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 10)                5130

=================================================================
Total params: 21,124,938
Trainable params: 21,109,706
Non-trainable params: 15,232
_________________________________________________________________
```