

Jaringan Neural Konvolusional Praktis

Menerapkan model pembelajaran mendalam tingkat lanjut menggunakan Python

Mohit Sewak
Md. Rezaul Karim
Pradeep Pujari



BIRMINGHAM - MUMBAI

Jaringan Neural Konvolusional Praktis

Hak Cipta © 2018 Packt Publishing

Seluruh hak cipta. Tidak ada bagian dari buku ini yang boleh direproduksi, disimpan dalam sistem pengambilan, atau dikirim dalam bentuk apa pun atau dengan cara apa pun, tanpa izin tertulis sebelumnya dari penerbit, kecuali dalam kasus kutipan singkat yang disematkan dalam artikel atau ulasan penting.

Segala upaya telah dilakukan dalam penyusunan buku ini untuk memastikan keakuratan informasi yang disajikan. Namun, informasi yang terkandung dalam buku ini dijual tanpa jaminan, baik tersurat maupun tersirat. Baik penulis, maupun Packt Publishing atau dealer dan distributornya, tidak akan bertanggung jawab atas segala kerusakan yang disebabkan atau diduga disebabkan secara langsung atau tidak langsung oleh buku ini.

Packt Publishing telah berusaha untuk memberikan informasi merek dagang tentang semua perusahaan dan produk yang disebutkan dalam buku ini dengan penggunaan modal yang tepat. Namun, Packt Publishing tidak dapat menjamin keakuratan informasi ini.

Commissioning Editor: Sunith Shetty

Akuisisi Editor: Vinay Argekar

Pengembangan Konten Editor: Cheryl Dsa

Teknis Editor: Sagar Sawant

Copy Editor: Vikrant Phadke, Safis Editing

Koordinator Proyek: Nidhi Joshi

Proofreader: Safis Editing

Indexer: Tejal Daruwale Soni

Graphics: Tania Dutta

Koordinator Produksi: Arvindkumar Gupta

Pertama kali diterbitkan: Februari 2018

Referensi produksi: 1230218

Diterbitkan oleh Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, Inggris.

ISBN 978-1-78839-230-3



mapt.io

Mapt adalah perpustakaan digital online yang memberi Anda akses penuh ke lebih dari 5.000 buku dan video, serta alat industri terkemuka untuk membantu Anda merencanakan pengembangan pribadi dan memajukan karier Anda. Untuk informasi lebih lanjut, silakan kunjungi situs web kami.

Mengapa berlangganan?

- Habiskan lebih sedikit waktu untuk belajar dan lebih banyak waktu coding dengan eBook dan Video praktis dari lebih dari 4.000 profesional industri
- Tingkatkan pembelajaran Anda dengan Skill Plans yang dibuat khusus untuk Anda
- Dapatkan eBook atau video gratis setiap bulan
- Mapt sepenuhnya dapat dicari
- Salin dan tempel, cetak, dan tandai konten

PacktPub.com

Tahukah Anda bahwa Packt menawarkan versi eBook dari setiap buku yang diterbitkan, dengan file PDF dan ePUB tersedia? Anda dapat meningkatkan ke versi eBook di www.PacktPub.com dan sebagai pelanggan buku cetak, Anda berhak mendapatkan diskon untuk salinan eBook. Hubungi kami di service@packtpub.com untuk lebih jelasnya.

Di www.PacktPub.com, Anda juga dapat membaca kumpulan artikel teknis gratis, mendaftar ke berbagai buletin gratis, dan menerima diskon dan penawaran eksklusif untuk buku dan eBook Packt.

Kontributor

Tentang Penulis

Mohit Sewak adalah ilmuwan data kognitif senior di IBM, dan sarjana PhD dalam AI dan CS di BITS Pilani. Dia memegang beberapa paten dan publikasi dalam AI, pembelajaran mendalam, dan pembelajaran mesin. Dia telah menjadi ilmuwan data utama untuk beberapa perangkat lunak AI / ML global yang sangat sukses dan solusi industri dan sebelumnya terlibat dalam solusi dan penelitian untuk lini produk Watson Cognitive Commerce. Dia memiliki 14 tahun pengalaman yang kaya dalam merancang dan menyelesaikan solusi dengan TensorFlow, Torch, Caffe, Theano, Keras, Watson, dan banyak lagi.

Md. Rezaul Karim adalah seorang peneliti di Fraunhofer FIT, Jerman. Ia juga seorang kandidat PhD di RWTH Aachen University, Jerman. Sebelum bergabung dengan FIT, dia bekerja sebagai peneliti di Insight Center for Data Analytics, Irlandia. Dia adalah seorang insinyur utama di Samsung Electronics, Korea.

Dia memiliki 9 tahun pengalaman R&D dengan C ++, Java, R, Scala, dan Python. Dia telah menerbitkan makalah penelitian tentang bioinformatika, data besar, dan pembelajaran mendalam. Dia memiliki pengalaman kerja praktis dengan Spark, Zeppelin, Hadoop, Keras, Scikit-Learn, TensorFlow, Deeplearning4j, MXNet, dan H2O.

Pradeep Pujari adalah insinyur pembelajaran mesin di Walmart Labs dan anggota ACM terkemuka . Keahlian domain intinya adalah dalam pengambilan informasi, pembelajaran mesin, dan pemrosesan bahasa alami. Di waktu luangnya, dia suka menjelajahi teknologi AI, membaca, dan membimbing.

Tentang pengulas

Sumit Pal adalah penulis terbitan Apress. Dia memiliki lebih dari 22 tahun pengalaman dalam perangkat lunak, dari start-up hingga perusahaan, dan merupakan konsultan independen yang bekerja dengan data besar, visualisasi data, dan ilmu data. Dia membangun sistem analitik berbasis data ujung ke ujung.

Dia telah bekerja untuk Microsoft (SQLServer), Oracle (OLAP Kernel), dan Verizon. Dia memberi saran kepada klien tentang arsitektur data mereka dan membangun solusi di Spark dan Scala. Dia telah berbicara di banyak konferensi di Amerika Utara dan Eropa dan telah mengembangkan pelatihan analis data besar untuk Experfy. Dia memiliki MS dan BS dalam ilmu komputer.

Packt sedang mencari penulis seperti Anda

Jika Anda tertarik menjadi penulis untuk Packt, silakan kunjungi author.packtpub.com dan daftar sekarang. Kami telah bekerja dengan ribuan pengembang dan profesional teknologi, seperti Anda, untuk membantu mereka berbagi wawasan dengan komunitas teknologi global. Anda dapat membuat lamaran umum, melamar topik hangat tertentu yang kami rekrut sebagai penulis, atau mengirimkan ide Anda sendiri.

Daftar Isi

| |
|---|
| Judul Halaman |
| Hak Cipta dan Kredit |
| Jaringan Neural Konvolusional Praktis |
| Packt Upsell |
| Mengapa berlangganan? |
| PacktPub.com |
| Kontributor |
| Tentang Penulis |
| Tentang pengulas |
| Packt sedang mencari penulis seperti Anda |
| Kata pengantar |
| Untuk siapa buku ini |
| Apa yang dicakup buku ini |
| Untuk mendapatkan hasil maksimal dari buku ini |
| Unduh file kode contoh |
| Unduh gambar berwarna |
| Konvensi digunakan |
| Berhubungan |
| Ulasan |
| 1. Deep Neural Networks - Gambaran Umum |
| Blok penyusun jaringan neural |
| Pengantar TensorFlow |
| Menginstal TensorFlow |
| Untuk varian macOS X / Linux |
| Dasar-dasar TensorFlow |
| Matematika dasar dengan TensorFlow |
| Softmax di TensorFlow |
| Pengantar set data MNIST |
| Jaringan syaraf tiruan paling sederhana |
| Membangun jaringan neural lapisan tunggal dengan TensorFlow |
| Ringkasan pustaka deep learning dengan keras |
| Lapisan dalam model Keras |
| Pengenalan nomor tulisan tangan dengan Keras dan MNIST |
| Mengambil data pelatihan dan pengujian |
| Data yang diratakan |
| Memvisualisasikan data pelatihan |
| Membangun jaringan |
| Melatih jaringan |

Menguji

Memahami propagasi mundur

Ringkasan

2. Pengantar Jaringan Neural Konvolusional

Sejarah CNN

Jaringan saraf konvolusional

Bagaimana komputer menginterpretasikan gambar?

Kode untuk memvisualisasikan gambar

Keluar

Lapisan masukan

Lapisan konvolusional

Lapisan konvolusional di Keras

Lapisan penggabungan

Contoh praktis - klasifikasi gambar

Augmentasi gambar

Ringkasan

3. Buat CNN Pertama dan Pengoptimalan Kinerja Anda

Arsitektur CNN dan kekurangan DNN

Operasi konvolusional

Operasi penggabungan, langkah, dan bantalan

Lapisan terhubung sepenuhnya

Operasi konvolusi dan penggabungan di TensorFlow

Menerapkan operasi penggabungan di TensorFlow

Operasi konvolusi di TensorFlow

Melatih CNN

Inisialisasi bobot dan bias

Regularisasi

Fungsi aktivasi

Menggunakan sigmoid

Menggunakan tanh

Menggunakan ULT

Membangun, melatih, dan mengevaluasi CNN pertama kami

Deskripsi kumpulan data

Langkah 1 - Memuat paket yang diperlukan

Langkah 2 - Memuat gambar pelatihan / pengujian untuk menghasilkan set pelatihan / pengujian

Langkah 3 - Mendefinisikan hyperparameter CNN

Langkah 4 - Membangun lapisan CNN

Langkah 5 - Mempersiapkan grafik TensorFlow

Langkah 6 - Membuat model CNN

Langkah 7 - Jalankan grafik TensorFlow untuk melatih model CNN

Langkah 8 - Evaluasi model

Model optimasi kinerja

Jumlah lapisan tersembunyi

Jumlah neuron per lapisan tersembunyi

Normalisasi batch

Regularisasi lanjutan dan menghindari overfitting

Menerapkan operasi dropout dengan TensorFlow

Pengoptimal mana yang akan digunakan?

Penyetelan memori

Penempatan lapisan yang tepat

Membangun CNN kedua dengan menyatukan semuanya

Deskripsi set data dan pemrosesan awal

Membuat model CNN

Melatih dan mengevaluasi jaringan

Ringkasan

4. Arsitektur Model CNN Populer

Pengantar ImageNet

LeNet

Arsitektur AlexNet

Pengklasifikasi tanda lalu lintas menggunakan AlexNet

Arsitektur VGGNet

Contoh kode klasifikasi gambar VGG16

Arsitektur GoogLeNet

Wawasan arsitektur

Modul awal

Arsitektur ResNet

Ringkasan

5. Transfer Pembelajaran

Pendekatan ekstraksi fitur

Dataset target kecil dan mirip dengan set data pelatihan asli

Dataset target kecil tetapi berbeda dari set data pelatihan asli

Set data target berukuran besar dan mirip dengan set data pelatihan asli

Set data target berukuran besar dan berbeda dari set data pelatihan asli

Contoh pembelajaran transfer

Pembelajaran multi-tugas

Ringkasan

6. Autoencoder untuk CNN

Memperkenalkan autoencoders

Autoencoder konvolusional

Aplikasi

Contoh kompresi

Ringkasan

7. Deteksi Objek dan Segmentasi Instance dengan CNN

Perbedaan antara deteksi objek dan klasifikasi citra

Mengapa deteksi objek jauh lebih menantang daripada klasifikasi gambar?

Pendekatan tradisional nonCNN untuk deteksi objek

Fitur Haar, pengklasifikasi berjenjang, dan algoritme Viola-Jones

Fitur Haar

Pengklasifikasi bertingkat

Algoritma Viola-Jones

R-CNN - Wilayah dengan fitur CNN

Fast R-CNN - CNN berbasis wilayah cepat

R-CNN Lebih Cepat - CNN berbasis jaringan proposal region yang lebih cepat

Mask R-CNN - Segmentasi instance dengan CNN

Segmentasi instance dalam kode

Menciptakan lingkungan

Menginstal dependensi Python (lingkungan Python2)

Mengunduh dan menginstal COCO API dan pustaka detektor (perintah shell OS)

Mempersiapkan struktur folder dataset COCO

Menjalankan model terlatih pada set data COCO

Referensi

Ringkasan

8. GAN: Menghasilkan Gambar Baru dengan CNN

Pix2pix - Terjemahan Gambar-ke-Gambar GAN

CycleGAN

Melatih model GAN

GAN - contoh kode

Menghitung kerugian

Menambahkan pengoptimal

Pembelajaran semi-supervisi dan GAN

Pencocokan fitur

Klasifikasi semi-terbimbing menggunakan contoh GAN

GAN konvolusional yang dalam

Normalisasi batch

Ringkasan

9. Mekanisme Perhatian untuk CNN dan Model Visual

Mekanisme perhatian untuk pembuatan teks gambar

Jenis Perhatian

Perhatian Keras

Perhatian Lembut

Menggunakan perhatian untuk meningkatkan model visual

Alasan untuk performa model CNN visual yang kurang optimal

Model perhatian visual yang berulang

Menerapkan RAM pada sampel MNIST yang berisik

Sekilas Sensor dalam kode

Referensi

Ringkasan

Buku Lain yang Mungkin Anda Nikmati

Tinggalkan ulasan - beri tahu pembaca lain apa yang Anda pikirkan

Kata pengantar

CNN merevolusi beberapa domain aplikasi, seperti sistem pengenalan visual, mobil tanpa pengemudi, penemuan medis, e-commerce inovatif, dan banyak lagi. Buku ini membantu Anda memulai dengan blok bangunan CNN, sekaligus memandu Anda melalui praktik terbaik untuk menerapkan model dan solusi CNN kehidupan nyata. Anda akan belajar membuat solusi inovatif untuk analisis gambar dan video guna memecahkan masalah pembelajaran mesin dan visi komputer yang kompleks.

Buku ini dimulai dengan ikhtisar jaringan neural dalam, dengan contoh klasifikasi gambar, dan memandu Anda membangun model CNN pertama Anda. Anda akan mempelajari konsep-konsep seperti pembelajaran transfer dan pembuat kode otomatis dengan CNN yang akan memungkinkan Anda membuat model yang sangat kuat, bahkan dengan data pelatihan yang diawasi (gambar berlabel) terbatas .

Kemudian kami mengembangkan pembelajaran ini untuk mencapai algoritme dan solusi terkait visi yang canggih untuk deteksi objek, segmentasi instance, jaringan generatif (bermusuhan), teks gambar, mekanisme perhatian, dan model perhatian berulang untuk penglihatan.

Selain memberi Anda pengalaman langsung dengan model dan arsitektur visi yang paling menarik, buku ini mengeksplorasi penelitian mutakhir dan terbaru di bidang CNN dan visi komputer. Hal ini memungkinkan pengguna untuk meramalkan masa depan dalam bidang ini dan memulai perjalanan inovasi mereka dengan cepat menggunakan solusi CNN yang canggih.

Di akhir buku ini, Anda harus siap untuk mengimplementasikan model CNN yang canggih, efektif, dan efisien dalam proyek profesional atau inisiatif pribadi Anda saat mengerjakan kumpulan data gambar dan video yang kompleks.

Untuk siapa buku ini

Buku ini ditujukan bagi para ilmuwan data, pembelajaran mesin, dan praktisi pembelajaran mendalam, serta penggemar kognitif dan kecerdasan buatan yang ingin selangkah lebih maju dalam membangun CNN. Dapatkan pengalaman langsung dengan kumpulan data ekstrem dan arsitektur CNN yang berbeda untuk membangun model ConvNet yang efisien dan cerdas. Pengetahuan dasar tentang konsep pembelajaran mendalam dan bahasa pemrograman Python sangat diharapkan.

Apa yang dicakup buku ini

[Bab 1](#) , *Deep Neural Networks - Gambaran umum* , ini memberikan penyegaran cepat dari ilmu jaringan neural dalam dan kerangka kerja berbeda yang dapat digunakan untuk mengimplementasikan jaringan tersebut, dengan matematika di belakangnya.

[Bab 2](#) , *Pengantar Jaringan Neural Konvolusional* , memperkenalkan pembaca ke jaringan saraf konvolusional dan menunjukkan bagaimana pembelajaran mendalam dapat digunakan untuk mengekstrak wawasan dari gambar.

[Bab 3](#) , *Membuat CNN Pertama Anda dan Pengoptimalan Performa* , membuat model CNN sederhana untuk klasifikasi gambar dari awal, dan menjelaskan cara menyesuaikan hyperparameter dan mengoptimalkan waktu pelatihan dan performa CNN untuk masing-masing meningkatkan efisiensi dan akurasi.

[Bab 4](#) , *Arsitektur Model CNN Populer* , menunjukkan keunggulan dan cara kerja arsitektur CNN populer (dan pemenang penghargaan) yang berbeda, bagaimana mereka berbeda satu sama lain, dan bagaimana menggunakannya.

[Bab 5](#) , *Pembelajaran Mentransfer* , mengajarkan Anda untuk mengambil jaringan yang telah dilatih sebelumnya dan menyesuaikannya dengan kumpulan data baru dan

berbeda. Ada juga masalah klasifikasi khusus untuk aplikasi kehidupan nyata menggunakan teknik yang disebut **pembelajaran transfer**.

[Bab 6](#), *Utoencoders untuk CNN*, memperkenalkan teknik pembelajaran tanpa pengawasan yang disebut **autoencoders**. Kami berjalan melalui berbagai aplikasi autoencoder untuk CNN, seperti kompresi gambar.

[Bab 7](#), *Deteksi Objek dan Segmentasi Instance dengan CNN*, mengajarkan perbedaan antara deteksi objek, segmentasi instance, dan klasifikasi gambar. Kami kemudian mempelajari beberapa teknik untuk deteksi objek dan segmentasi contoh dengan CNN.

[Bab 8](#), *GAN — Menghasilkan Gambar Baru dengan CNN*, menjelajahi Jaringan CNN generatif, dan kemudian kami menggabungkannya dengan jaringan CNN diskriminatif yang kami pelajari untuk membuat gambar baru dengan CNN / GAN.

[Bab 9](#), *Mekanisme Perhatian untuk Model CNN dan Visual*, mengajarkan intuisi di balik perhatian dalam pembelajaran mendalam dan mempelajari bagaimana model berbasis perhatian digunakan untuk mengimplementasikan beberapa solusi lanjutan (teks gambar dan RAM). Kami juga memahami berbagai jenis perhatian dan peran pembelajaran penguatan sehubungan dengan mekanisme perhatian keras.

Untuk mendapatkan hasil maksimal dari buku ini

Buku ini difokuskan untuk membangun CNN dengan bahasa pemrograman Python. Kami telah menggunakan Python versi 2.7 (2x) untuk membangun berbagai aplikasi dan sumber terbuka serta perangkat lunak profesional yang siap untuk perusahaan menggunakan Python, Spyder, Anaconda, dan PyCharm. Banyak contoh juga kompatibel dengan Python 3x. Sebagai praktik yang baik, kami mendorong pengguna untuk menggunakan lingkungan virtual Python untuk mengimplementasikan kode-kode ini.

Kami fokus pada cara menggunakan berbagai Python dan pustaka pembelajaran mendalam (Keras, TensorFlow, dan Caffe) dengan cara terbaik untuk membangun aplikasi dunia nyata. Dalam semangat itu, kami telah mencoba untuk menjaga agar semua kode tetap ramah dan mudah dibaca. Kami merasa bahwa ini akan memungkinkan pembaca kami untuk dengan mudah memahami kode dan dengan mudah menggunakan kodenya dalam skenario yang berbeda.

Unduh file kode contoh

Anda dapat mendownload file kode contoh untuk buku ini dari akun Anda di www.packtpub.com. Jika Anda membeli buku ini di tempat lain, Anda dapat mengunjungi www.packtpub.com.

[com/support](http://www.packtpub.com/support) dan mendaftar agar file dikirim langsung ke email Anda.

Anda dapat mengunduh file kode dengan mengikuti langkah-langkah berikut:

1. Masuk atau daftar di www.packtpub.com .
2. Pilih tab DUKUNGAN .
3. Klik pada Download Kode & Errata .
4. Masukkan nama buku di kotak Pencarian dan ikuti petunjuk di layar.

Setelah file diunduh, pastikan Anda mengekstrak atau mengekstrak folder menggunakan versi terbaru:

- WinRAR / 7-Zip untuk Windows
- Zipeg / iZip / UnRarX untuk Mac
- 7-Zip / PeaZip untuk Linux

Paket kode untuk buku juga dihosting di GitHub di <https://github.com/PacktPublishing/Practical-Convolutional-Neural-Networks> . Jika ada pembaruan pada kode, itu akan diperbarui di repositori GitHub yang ada.

Kami juga memiliki bundel kode lain dari katalog kaya buku dan video kami yang tersedia di <https://github.com/PacktPublishing/> . Periksa mereka!

Unduh gambar berwarna

Kami juga menyediakan file PDF yang memiliki gambar berwarna dari screenshot / diagram yang digunakan dalam buku ini. Anda dapat mengunduhnya di sini: http://www.packtpub.com/sites/default/files/downloads/PracticalConvolutionalNeuralNetworks_ColorImages.pdf .

Konvensi digunakan

Ada sejumlah konvensi teks yang digunakan di seluruh buku ini.

CodeInText: Menunjukkan kata-kata kode dalam teks, nama tabel database, nama folder, nama file, ekstensi file, nama jalur, URL tiruan, input pengguna, dan pegangan Twitter. Berikut ini contohnya: "Pasang `WebStorm-10*.dmg` file image disk yang didownload sebagai disk lain di sistem Anda."

Satu blok kode diatur sebagai berikut:

```
impor tensorflow sebagai tf
#Membuat objek TensorFlow
hello_constant = tf.constant ( 'Hello World!', Name = 'hello_constant' )
#Membuat objek sesi untuk menjalankan grafik komputasi
dengan tf.Session () sebagai sess:
```

Jika kami ingin menarik perhatian Anda ke bagian tertentu dari blok kode, baris atau item yang relevan dicetak tebal:

```
| x = tf. kurangi (1, 2 , name = None) # -1  
| y = tf. multiply (2, 5, name = None) # 10
```

Tebal : Menunjukkan istilah baru, kata penting, atau words yang Anda lihat di layar. Misalnya, kata-kata dalam menu atau kotak dialog muncul dalam teks seperti ini. Berikut ini contohnya: "Pilih info Sistem dari panel Administrasi . "

Peringatan atau catatan penting muncul seperti ini.

Tip dan trik muncul seperti ini.

Berhubungan

Umpang balik dari pembaca kami selalu diterima.

Umpang balik umum : Email feedback@packtpub.com dan sebutkan judul buku di subjek pesan Anda. Jika Anda memiliki pertanyaan tentang aspek apa pun dari buku ini, silakan email kami di questions@packtpub.com.

Errata : Meskipun kami telah berhati-hati untuk memastikan keakuratan konten kami, kesalahan bisa saja terjadi. Jika Anda menemukan kesalahan dalam buku ini, kami akan berterima kasih jika Anda melaporkannya kepada kami. Silakan kunjungi www.packtpub.com/submit-errata , pilih buku Anda, klik tautan Formulir Pengiriman Errata, dan masukkan detailnya.

Pembajakan : Jika Anda menemukan salinan ilegal dari karya kami dalam bentuk apa pun di Internet, kami akan berterima kasih jika Anda memberi kami alamat lokasi atau nama situs web. Silakan hubungi kami di copyright@packtpub.com dengan tautan ke materi.

Jika Anda tertarik menjadi seorang penulis : Jika ada topik yang Anda kuasai dan Anda tertarik untuk menulis atau berkontribusi pada sebuah buku, silakan kunjungi auth.or.packtpub.com .

Ulasan

Silakan tinggalkan ulasan. Setelah Anda membaca dan menggunakan buku ini, mengapa tidak meninggalkan ulasan di situs tempat Anda membelinya? Pembaca potensial kemudian dapat melihat dan menggunakan opini Anda yang tidak bias untuk

membuat keputusan pembelian, kami di Packt dapat memahami apa yang Anda pikirkan tentang produk kami, dan penulis kami dapat melihat tanggapan Anda pada buku mereka. Terima kasih!

Untuk informasi lebih lanjut tentang Packt, silakan kunjungi packtpub.com.

Deep Neural Networks - Gambaran Umum

Dalam beberapa tahun terakhir, kami telah melihat kemajuan luar biasa di bidang AI (pembelajaran mendalam). Saat ini, pembelajaran mendalam adalah landasan dari banyak teknologi majuaplikasi, dari mobil self-driving hingga menghasilkan seni dan musik. Ilmuwan bertujuan untuk membantu komputer tidak hanya memahami ucapan tetapi juga berbicara dalam bahasa alami. Pembelajaran mendalam adalah sejenis metode pembelajaran mesin yang didasarkan pada representasi data pembelajaran sebagai lawan dari algoritma khusus tugas. Pembelajaran mendalam memungkinkan komputer untuk membangun konsep yang kompleks dari konsep yang lebih sederhana dan lebih kecil. Misalnya, sistem pembelajaran mendalam mengenali citra seseorang dengan menggabungkan tepi dan sudut label bawah dan menggabungkannya menjadi beberapa bagian tubuh secara hierarkis. Tidak lama lagi saat pembelajaran mendalam akan diperluas ke aplikasi yang memungkinkan mesin untuk berpikir sendiri.

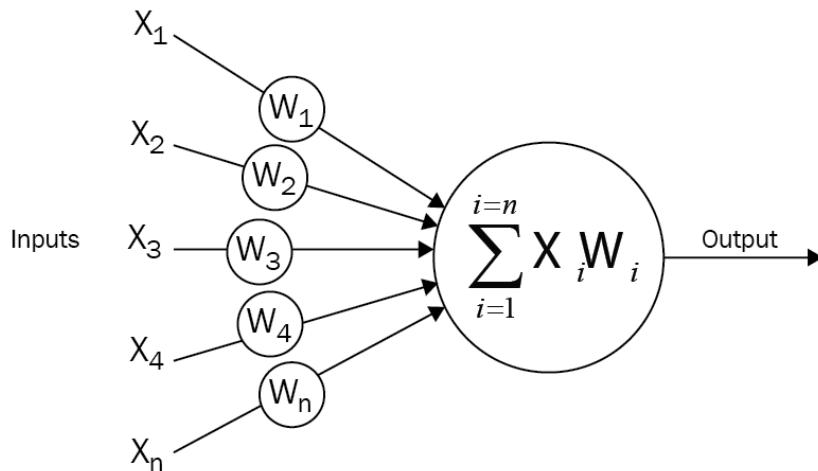
Dalam bab ini, kami akan membahas topik-topik berikut:

- Blok penyusun jaringan neural
- Pengantar TensorFlow
- Pengantar Keras
- Propagasi mundur

Blok penyusun jaringan neural

Jaringan saraf terdiri dari banyak neuron buatan. Apakah itu representasi otak atau apakah itu representasi matematis dari beberapa pengetahuan? Di sini, kami hanya akan mencoba memahami bagaimana jaringan saraf digunakan dalam praktik. Sebuah **jaringan saraf convolutional (CNN)** adalah jenis yang sangat khusus multi-layer jaringan saraf. CNN dirancang untuk mengenali pola visual langsung dari gambar dengan pemrosesan minimal. Representasi grafis dari jaringan ini dihasilkan pada gambar berikut. Bidang jaringan saraf awalnya terinspirasi oleh tujuan pemodelan sistem saraf biologis, tetapi sejak itu telah bercabang ke arah yang berbeda dan telah menjadi masalah teknik dan mencapai hasil yang baik dalam tugas pembelajaran mesin.

Neuron buatan adalah fungsi yang menerima masukan dan menghasilkan keluaran. Jumlah neuron yang digunakan bergantung pada tugas yang dihadapi. Itu bisa serendah dua atau sebanyak beberapa ribu. Ada banyak cara untuk menghubungkan neuron buatan untuk membuat CNN. Salah satu topologi yang umum digunakan dikenal sebagai jaringan **umpang-maju** :



Setiap neuron menerima masukan dari neuron lain. Efek dari setiap jalur masukan pada neuron dikontrol oleh beratnya. Bobotnya bisa positif atau negatif. Seluruh jaringan saraf belajar melakukan komputasi yang berguna untuk mengenali objek dengan memahami bahasanya. Sekarang, kita dapat menghubungkan neuron-neuron tersebut ke dalam jaringan yang dikenal sebagai jaringan umpan-maju. Ini berarti bahwa neuron di setiap lapisan memasukkan keluarannya ke lapisan berikutnya sampai kita mendapatkan hasil akhir. Ini dapat dituliskan sebagai berikut:

$$w_1x_1 + w_2x_2 + \dots + w_nx_n$$

$$\sum_{i=1}^{i=n} w_i x_i = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Neuron yang merambat maju sebelumnya dapat diimplementasikan sebagai berikut:

```
import numpy as np
import math

class Neuron (object):
    def __init__ (self):
        self.weights = np.array ([1.0, 2.0])
        self.bias = 0.0
    def forward (self, input):
        """
        Dengan asumsi bahwa input dan bobot adalah array numpy 1-D dan biasnya adalah angka """
        a_cell_sum = np.sum (input * self.weights) + self.bias
        result = 1.0 / (1.0 + math.exp (-a_cell_sum)) # ini adalah sigmoid fungsi aktivasi
        hasil kembali
neuron = neuron ()
output = neuron.forward (np.array ([1,1]))
print (output)
```

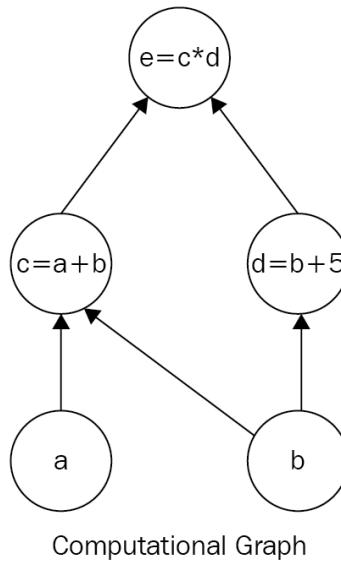
Pengantar TensorFlow

TensorFlow didasarkan pada komputasi berbasis grafik. Perhatikan ekspresi matematika berikut, misalnya:

$$c = (a + b), \quad d = b + 5,$$

$$e = c * d$$

Di TensorFlow, ini direpresentasikan sebagai grafik komputasi, seperti yang ditunjukkan di sini. Ini hebat karena komputasi dilakukan secara paralel:



Menginstal TensorFlow

Ada dua cara mudah untuk menginstal TensorFlow:

- Menggunakan lingkungan virtual (direkomendasikan dan dijelaskan di sini)
- Dengan gambar Docker

Untuk varian macOS X / Linux

Cuplikan kode berikut membuat lingkungan virtual Python dan menginstal TensorFlow di lingkungan itu. Anda harus menginstal Anaconda sebelum Anda menjalankan kode ini:

```
#Membuat lingkungan virtual bernama "tensorflow_env" dengan asumsi bahwa versi python 3.7 telah
diinstal.
conda create -n tensorflow_env python = 3.7
#Aktifkan titik ke lingkungan bernama "tensorflow" sumber aktifkan tensorflow_env

conda install pandas matplotlib jupyter notebook scipy scikit-learn
#installs versi tensorflow terbaru ke lingkungan tensorflow_env
pip3 memasang tensorflow
```

Silakan lihat update terbaru di halaman TensorFlow resmi, <https://www.tensorflow.org/install/1/>.

Coba jalankan kode berikut di konsol Python Anda untuk memvalidasi instalasi Anda. Konsol harus mencetak `Hello World!` jika TensorFlow diinstal dan berfungsi:

```
| impor tensorflow sebagai tf
|
| #Membuat objek TensorFlow
| hello_constant = tf.constant ( 'Hello World!', Name = 'hello_constant' )
| #Membuat objek sesi untuk menjalankan grafik komputasi
| dengan tf.Session () sebagai sess:
|     #Implementing operasi tf.constant dalam sesi
|     output = sess.run (hello_constant)
|     cetak (keluaran)
```

Dasar-dasar TensorFlow

Di TensorFlow, data tidak disimpan sebagai integer, float, string, atau primitif lainnya. Nilai-nilai ini dikemas dalam objek yang disebut **tensor**. Ini terdiri dari satu set nilai primitif yang dibentuk menjadi larik dengan sejumlah dimensi. Jumlah dimensi dalam tensor disebut **rank**-nya . Pada contoh sebelumnya, `hello_constant` adalah tensor string konstan dengan pangkat nol. Beberapa contoh tensor konstan adalah sebagai berikut:

```
| # A adalah tensor int32 dengan rank = 0
| A = tf.constant (123)
| # B adalah tensor int32 dengan dimensi 1 (rank = 1)
| B = tf.constant ([123.456.789])
| # C adalah int32 2- tensor dimensi
| C = tf. konstan ([[123.456.789], [222.333.444]])
```

Program inti TensorFlow didasarkan pada ide grafik komputasi. Grafik komputasi adalah grafik berarah yang terdiri dari dua bagian berikut:

- Membangun grafik komputasi
- Menjalankan grafik komputasi

Grafik komputasi dijalankan dalam satu **sesi** . Sesi TensorFlow adalah lingkungan waktu proses untuk grafik komputasi. Ini mengalokasikan CPU atau GPU dan mempertahankan status waktu proses TensorFlow. Kode berikut membuat contoh sesi bernama `sess` menggunakan `tf.Session`. Kemudian `sess.run()` fungsi tersebut mengevaluasi tensor dan mengembalikan hasil yang disimpan dalam `output` variabel. Akhirnya dicetak sebagai `Hello World!`:

```
| dengan tf.Session () sebagai sess:
|     # Jalankan operasi tf.constant dalam sesi
|     output = sess.run (hello_constant)
|     print (output)
```

Dengan menggunakan TensorBoard, kita dapat memvisualisasikan grafik. Untuk menjalankan TensorBoard, gunakan perintah berikut:

```
| tensorboard --logdir = jalur / ke / direktori-log
```

Mari buat potongan kode penjumlahan sederhana sebagai berikut. Buat bilangan bulat konstan `x` dengan nilai 5, setel nilai variabel baru `y` setelah menambahkannya 5, dan cetak:

```
| konstanta_x = tf. konstanta (5, nama = 'konstanta_x')
| variabel_y = tf. Variabel (x + 5, nama = 'variabel_y')
| cetak (variabel_y)
```

Perbedaannya adalah `variabel_y` tidak diberi nilai saat ini `x + 5` sebagaimana mestinya dalam kode Python. Sebaliknya, ini adalah persamaan; itu berarti, ketika `variabel_y` dihitung, ambil nilai `x` pada saat itu dan tambahkan `5` padanya. Penghitungan nilai `variabel_y` tidak pernah benar-benar dilakukan dalam kode sebelumnya. Potongan kode ini sebenarnya milik bagian pembuatan grafik komputasi dari program TensorFlow biasa. Setelah menjalankan ini, Anda akan mendapatkan sesuatu seperti `<tensorflow.python.ops.variables.Variable object at 0x7f074bfd9ef0>` dan bukan nilai sebenarnya dari `variabel_y` as `10`. Untuk memperbaikinya, kita harus mengeksekusi bagian kode dari grafik komputasi, yang terlihat seperti ini:

```
| #inisialisasi semua variabel
| init = tf.global_variables_initializer ()
| # Semua variabel sekarang diinisialisasi
|
| dengan tf.Session () sebagai sess:
|     sess.run (init)
|     print (sess.run (variabel_y))
```

Berikut adalah pelaksanaan beberapa fungsi matematika dasar, seperti penjumlahan, pengurangan, perkalian, dan pembagian dengan tensor. Untuk lebih banyak fungsi matematika, silakan lihat dokumentasinya:

Untuk fungsi matematika TensorFlow, buka https://www.tensorflow.org/versions/r0.12/api_docs/python/math_ops/basic_math_functions.

Matematika dasar dengan TensorFlow

The `tf.add()` Fungsi mengambil dua angka, dua tensor, atau salah satu dari masing-masing, dan ia mengembalikan jumlah mereka sebagai tensor a:

```
| Selain
| x = tf.add (1, 2, nama = None) # 3
```

Berikut adalah contoh pengurangan dan perkalian:

```
| x = tf. kurangi (1, 2, name = None) # -1
| y = tf. multiply (2, 5, name = None) # 10
```

Bagaimana jika kita ingin menggunakan non-konstanta? Bagaimana cara memasukkan set data input ke TensorFlow? Untuk ini, TensorFlow menyediakan API `tf.placeholder()`, dan penggunaan `feed_dict`.

A `placeholder` adalah variabel yang datanya ditugaskan nanti di `tf.session.run()` fungsi. Dengan bantuan ini, operasi kami dapat dibuat dan kami dapat membangun grafik komputasi kami tanpa memerlukan data. Setelah itu, data ini dimasukkan ke dalam

grafik melalui placeholder ini dengan bantuan `feed_dict` parameter `tf.Session.run()` untuk mengatur `placeholder` tensor. Dalam contoh berikut, tensor `x` disetel ke string `Hello World` sebelum sesi berjalan:

```
x = tf.placeholder (tf.string)
dengan tf.Session () sebagai sess:
    output = sess.run (x, feed_dict = {x: 'Hello World'})
```

Anda juga dapat menyetel lebih dari satu tensor menggunakan `feed_dict`, sebagai berikut:

```
x = tf.placeholder (tf.string)
y = tf.placeholder (tf.int32, None)
z = tf.placeholder (tf.float32, None)

dengan tf.Session () sebagai sess:
    output = sess.run (x, feed_dict = {x: 'Selamat datang di CNN', y: 123, z: 123.45})
```

Placeholder juga dapat memungkinkan penyimpanan array dengan bantuan berbagai dimensi. Silakan lihat contoh berikut:

```
impor tensorflow sebagai tf
x = tf.placeholder ("float", [None, 3])
y = x * 2

dengan tf.Session () sebagai sesi:
    input_data = [[1, 2, 3],
                  [4, 5, 6],]
    result = session.run (y, feed_dict = {x: input_data})
print (hasil)
```

Ini akan memunculkan kesalahan seperti `ValueError: invalid literal for... dalam kasus di mana data yang diteruskan ke feed_dict parameter tidak cocok dengan jenis tensor dan tidak dapat dimasukkan ke dalam jenis tensor.`

The `tf.truncated_normal()` mengembalikan fungsi tensor dengan nilai acak dari distribusi normal. Ini sebagian besar digunakan untuk inisialisasi bobot di jaringan:

```
n_features = 5
n_labels = 2
weights = tf.truncated_normal ((n_features, n_labels))
dengan tf.Session () as sess:
    print (sess.run (weights))
```

Softmax di TensorFlow

Fungsi Softmax mengkonversi input, yang dikenal sebagai **logit** atau **logit skor**, menjadi antara 0 dan 1, dan juga menormalkan output sehingga mereka semua jumlah sampai dengan 1. Dengan kata lain, Softmax fungsi ternyata logits Anda ke dalam probabilitas. Secara matematis, yang fungsi Softmax didefinisikan sebagai berikut:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

Di TensorFlow, fungsi softmax diimplementasikan. Ini mengambil logits dan kembali Softmax aktivasi yang memiliki tipe yang sama dan bentuk sebagai logits input, seperti yang ditunjukkan pada gambar berikut:

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

Kode berikut digunakan untuk mengimplementasikan ini:

```
logit_data = [2.0, 1.0, 0.1]
logits = tf.placeholder(tf.float32)
softmax = tf.nn.softmax(logits)

dengan tf.Session () sebagai sess:
    output = sess.run (softmax, feed_dict = {logits : logit_data})
    print (keluaran)
```

Cara kami merepresentasikan label secara matematis sering disebut **enkode one-hot**. Setiap label diwakili oleh vektor yang memiliki 1,0 untuk label yang benar dan 0,0 untuk yang lainnya. Ini bekerja dengan baik untuk sebagian besar kasus masalah. Namun, ketika masalah memiliki jutaan label, pengkodean one-hot tidak efisien, karena sebagian besar elemen vektor adalah nol. Kami mengukur kesamaan jarak antara dua vektor probabilitas, yang dikenal sebagai **cross-entropi** dan dilambangkan dengan **D**.

Entropi silang tidak simetris. Artinya: $D(S, L) \neq D(L, S)$

Dalam pembelajaran mesin, kami mendefinisikan apa artinya model menjadi buruk biasanya dengan fungsi matematika. Fungsi ini disebut fungsi **kerugian**, **biaya**, atau **tujuan**. Salah satu fungsi yang sangat umum digunakan untuk menentukan kerugian model disebut **kerugian cross-entropy**. Konsep ini berasal dari teori informasi (untuk lebih lanjut tentang ini, silakan merujuk ke Teori Informasi Visual di <https://colah.github.io/posts/2015-09-Visual-Information/>). Secara intuitif, kerugian akan tinggi jika model melakukan klasifikasi yang buruk pada data pelatihan, dan jika tidak, akan rendah, seperti yang ditunjukkan di sini:

$$\begin{bmatrix} \hat{y} \\ 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \xrightarrow{D(\hat{y}, y)} \begin{bmatrix} y \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$D(\hat{y}, y) = -\sum_j y_j \ln \hat{y}_j$$

Fungsi kerugian cross-entropy

Di TensorFlow, kita bisa menulis fungsi lintas-entropi menggunakan `tf.reduce_sum()`; ia mengambil larik angka dan mengembalikan jumlahnya sebagai tensor (lihat blok kode

berikut):

```
x = tf.constant ([[1,1,1], [1,1,1]])
dengan tf.Session () sebagai sess:
    print (sess.run (tf.reduce_sum ([1,2,3] ))) #returns 6
    print (sess.run (tf.reduce_sum (x, 0))) #sum sepanjang sumbu x, mencetak [2,2,2]
```

Namun dalam praktiknya, saat menghitung fungsi softmax, suku perantara mungkin sangat besar karena adanya eksponensial. Jadi, membagi angka besar bisa jadi tidak stabil secara numerik. Kita harus menggunakan softmax dan API kerugian lintas entropi yang disediakan TensorFlow. Cuplikan kode berikut secara manual menghitung kehilangan lintas entropi dan juga mencetaknya menggunakan TensorFlow API:

```
impor tensorflow sebagai tf

softmax_data = [0.1,0.5,0.4]
onehot_data = [0.0,1.0,0.0]

softmax = tf.placeholder (tf.float32)
onehot_encoding = tf.placeholder (tf.float32)

cross_entropy = - tf.reduce_sum (tf .multiply (onehot_encoding, tf.log (softmax)))

cross_entropy_loss = tf.nn.softmax_cross_entropy_with_logits (logits = tf.log (softmax), label =
onehot_encoding)

dengan tf.Session () sebagai sesi:
    print (session.run (cross_entropyun) , feed_dict = {softmax: softmax_data, onehot_encoding:
onehot_data}))
    print (session.run (cross_entropy_loss, feed_dict = {softmax: softmax_data, onehot_encoding:
onehot_data}))
```

Pengantar set data MNIST

Di sini kami menggunakan **MNIST (Institut Standar dan Teknologi Nasional yang Dimodifikasi)**, yang terdiri dari gambar nomor tulisan tangan dan labelnya. Sejak dirilis pada tahun 1998, kumpulan data klasik ini digunakan untuk mengukur algoritme klasifikasi.

File data `train.csv` dan `test.csv` terdiri dari angka-angka yang digambar tangan, dari 0 sampai 9 dalam bentuk gambar skala abu-abu. Sebuah citra digital merupakan fungsi matematis yang berbentuk $f(x, y) = \text{nilai piksel}$. Gambarnya dua dimensi.

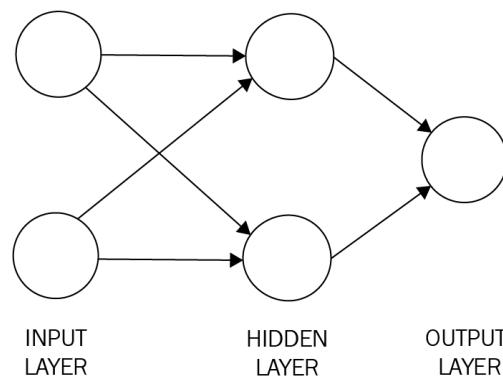
Kami dapat melakukan fungsi matematika apa pun pada gambar. Dengan menghitung gradien pada gambar, kita dapat mengukur seberapa cepat nilai piksel berubah dan arah perubahannya. Untuk pengenalan gambar, kami mengubah gambar menjadi skala abu-abu untuk kesederhanaan dan memiliki satu saluran warna. Representasi **RGB** dari suatu gambar terdiri dari tiga saluran warna, **MERAH**, **BIRU**, dan **HIJAU**. Dalam skema warna RGB, sebuah gambar adalah tumpukan tiga gambar MERAH, BIRU, dan HIJAU. Dalam skema warna grayscale, warna tidak penting. Gambar berwarna lebih sulit dianalisis secara komputasi karena membutuhkan lebih banyak ruang dalam memori. Intensitas, yaitu ukuran terang dan gelap suatu gambar, sangat berguna untuk mengenali objek. Pada beberapa aplikasi, misalnya untuk mendeteksi garis jalur pada aplikasi mobil tanpa pengemudi, warna menjadi penting karena harus membedakan

jalur kuning dan jalur putih. Gambar grayscale tidak memberikan informasi yang cukup untuk membedakan antara garis jalur putih dan kuning.

Setiap gambar grayscale diinterpretasikan oleh komputer sebagai matriks dengan satu entri untuk setiap piksel gambar. Setiap gambar berukuran tinggi dan lebar 28 x 28 piksel, menghasilkan jumlah 784 piksel. Setiap piksel memiliki satu nilai piksel yang terkait dengannya. Nilai ini menunjukkan terang atau gelapnya piksel tersebut. Nilai piksel ini adalah bilangan bulat mulai dari 0 hingga 255, di mana nilai nol berarti paling gelap dan 255 adalah yang paling putih, dan piksel abu-abu antara 0 dan 255.

Jaringan syaraf tiruan paling sederhana

Gambar berikut mewakili jaringan saraf dua lapisan sederhana:



Jaringan saraf dua lapis sederhana

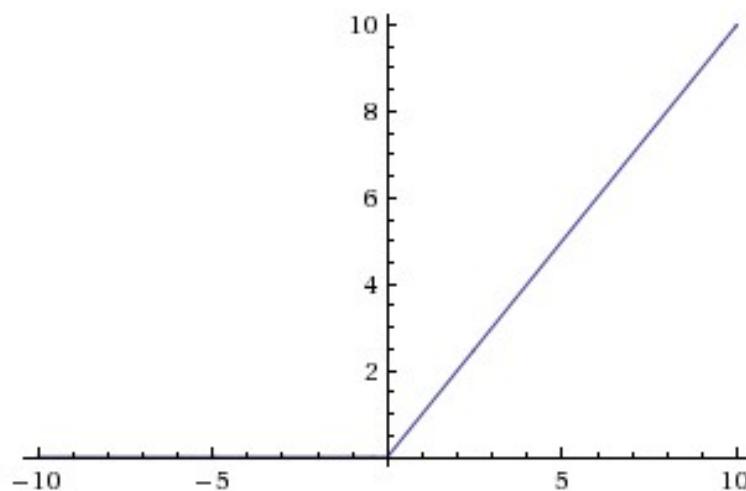
Lapisan pertama adalah lapisan **masukan** dan lapisan terakhir adalah **lapisan keluaran**. Lapisan tengah adalah **lapisan tersembunyi**. Jika ada lebih dari satu lapisan tersembunyi, maka jaringan tersebut adalah jaringan neural dalam.

Masukan dan keluaran setiap neuron di lapisan tersembunyi terhubung ke setiap neuron di lapisan berikutnya. Ada sejumlah neuron di setiap lapisan tergantung pada masalahnya. Mari kita perhatikan sebuah contoh. Contoh sederhana yang mungkin sudah Anda ketahui adalah pengenalan digit tulisan tangan populer yang mendekripsi angka, katakanlah 5. Jaringan ini akan menerima gambar 5 dan akan mengeluarkan 1 atau 0. A 1 adalah untuk menunjukkan gambar tersebut sebenarnya adalah 5 dan 0 sebaliknya. Setelah jaringan dibuat, itu harus dilatih. Kita dapat menginisialisasi dengan bobot acak dan kemudian memasukkan sampel masukan yang dikenal sebagai set **data pelatihan**. Untuk setiap sampel masukan, kami memeriksa keluaran, menghitung tingkat kesalahan dan kemudian menyesuaikan bobot sehingga setiap kali ia melihat 5 ia mengeluarkan 1 dan untuk yang lainnya ia mengeluarkan nol. Jenis pelatihan ini disebut **pembelajaran yang diawasi** dan metode penyesuaian bobot disebut **propagasi mundur**. Saat membangun model jaringan saraf tiruan, salah satu pertimbangan utama adalah bagaimana memilih fungsi aktivasi untuk lapisan

tersembunyi dan keluaran. Tiga fungsi aktivasi yang paling umum digunakan adalah fungsi sigmoid, fungsi tangen hiperbolik, dan **Rectified Linear Unit (ReLU)**. Keindahan dari fungsi sigmoid adalah turunannya dievaluasi pada z dan hanya z dikalikan dengan 1-minus z . Itu berarti:

$$dy/dx = \sigma(x)(1 - \sigma(x))$$

Ini membantu kami menghitung gradien yang digunakan di jaringan saraf secara efisien dengan cara yang nyaman. Jika aktivasi umpan maju dari fungsi logistik untuk lapisan tertentu disimpan dalam memori, gradien untuk lapisan tertentu itu dapat dievaluasi dengan bantuan perkalian dan pengurangan sederhana daripada menerapkan dan mengevaluasi kembali fungsi sigmoid, karena memerlukan eksponen ekstra. Gambar berikut menunjukkan kepada kita fungsi aktivasi ULT , yaitu nol ketika $x < 0$ dan kemudian linier dengan kemiringan 1 ketika $x > 0$:



ULT adalah fungsi nonlinier yang menghitung fungsi $f(x) = \max(0, x)$. Itu berarti fungsi ULT adalah 0 untuk input negatif dan x untuk semua input $x > 0$. Ini berarti aktivasi dibatasi pada nol (lihat gambar sebelumnya di sebelah kiri). TensorFlow mengimplementasikan fungsi ULT di `tf.nn.relu()`:

$$\text{relu}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Propagasi mundur, singkatan dari "propagasi kesalahan ke belakang", adalah metode umum untuk melatih jaringan saraf tiruan yang digunakan bersama dengan metode pengoptimalan seperti penurunan gradien. Metode ini menghitung gradien fungsi kerugian sehubungan dengan semua bobot di jaringan. Metode pengoptimalan diberi makan dengan gradien dan menggunakannya untuk memperbarui bobot guna mengurangi fungsi kerugian.

Membangun jaringan neural lapisan tunggal dengan TensorFlow

Mari kita buat neural net satu lapis dengan TensorFlow langkah demi langkah. Dalam contoh ini, kami akan menggunakan kumpulan data MNIST. Dataset ini adalah kumpulan gambar skala abu-abu berukuran 28 x 28 piksel dari digit tulisan tangan. Dataset ini terdiri dari 55.000 data latih, 10.000 data uji, dan 5.000 data validasi. Setiap titik data MNIST memiliki dua bagian: gambar digit tulisan tangan dan label yang sesuai. Blok kode berikut memuat data. `one_hot=True` berarti bahwa label adalah vektor yang dikodekan one-hot, bukan digit sebenarnya dari label. Misalnya, jika labelnya adalah 2, Anda akan melihat [0,0,1,0,0,0,0,0,0,0]. Ini memungkinkan kami untuk langsung menggunakanannya di lapisan keluaran jaringan:

```
| dari tensorflow.examples.tutorials.mnist import input_data  
| mnist = input_data.read_data_sets ("MNIST_data /", one_hot = True)
```

Menyiapkan placeholder dan variabel dilakukan sebagai berikut:

```
| # Semua pixel pada gambar (28 * 28 = 784)  
| features_count = 784  
| # ada 10 digit yaitu label  
| labels_count = 10  
| batch_size = 128  
| epochs = 10  
| learning_rate = 0,5  
  
| fitur = tf . placeholder ( tf . float32 , [ None , features_count ])  
| label = tf . placeholder ( tf . float32 , [ Tidak ada , labels_count ])  
  
| #Set bobot dan bias  
| bobot tensor = tf . Variabel ( tf . Truncated_normal (( features_count , labels_count )))  
| bias = tf . Variabel ( tf . Nol ( labels_count ), nama = 'bias' )
```

Mari kita siapkan pengoptimal di TensorFlow:

```
| kerugian,  
| pengoptimal = tf . melatih . GradientDescentOptimizer ( learning_rate ) . meminimalkan ( kerugian )
```

Sebelum memulai pelatihan, mari kita siapkan operasi inisialisasi variabel dan operasi untuk mengukur keakuratan prediksi kita, sebagai berikut:

```
| # Fungsi Linear WX + b  
| logits = tf . add ( tf . matmul ( fitur , bobot ), bias )  
  
| prediksi = tf . nn . softmax ( logits )  
  
| # Entropi silang  
| cross_entropy = - tf . reduce_sum ( label * tf . log ( prediksi ), reduction_indices = 1 )  
  
| # Kerugian  
| kerugian pelatihan = tf . pengurangan_berarti ( persilangan_entropi )  
  
| # Menginisialisasi semua variabel  
| init = tf . global_variables_initializer ()  
  
| # Menentukan apakah prediksi akurat  
| is_correct_prediction = tf . sama ( tf . argmax ( prediksi , 1 ), tf . argmax ( label , 1 ))  
| # Menghitung akurasi  
| akurasi prediksi = tf . reduce_mean ( tf . cast ( is_correct_prediction , tf . float32 ))
```

Sekarang kita dapat mulai melatih model, seperti yang ditunjukkan pada cuplikan kode berikut:

```
| #Memulai sesi  
| dengan tf . Session () sebagai sess :  
|     # menginisialisasi semua variabel
```

```

sess . menjalankan ( init )
total_batch = int ( len ( mnist . kereta . label ) / batch_size )
untuk zaman di kisaran ( zaman ):
    avg_cost = 0
    untuk i di kisaran ( total_batch ):
        batch_x , batch_y = mnist . melatih . next_batch ( batch_size = batch_size )
        _, c = sess . jalankan ([ pengoptimal , rugi ], feed_dict = { fitur : batch_x ,
label : batch_y })
        avg_cost += c / total_batch
    print ( "Epoch:" , ( epoch + 1 ), "cost =" , " {:.3f} " . format ( avg_cost ))
    print ( sess . run ( akurasi , feed_dict = { fitur : mnist . test . images , labels : mnist .
test . labels }))

```

Ringkasan pustaka deep learning dengan keras

Keras adalah API jaringan neural dalam tingkat tinggi dengan Python yang berjalan di atas TensorFlow, CNTK, atau Theano.

Berikut beberapa konsep inti yang perlu Anda ketahui untuk bekerja dengan Keras. TensorFlow adalah pustaka pembelajaran mendalam untuk komputasi numerik dan kecerdasan mesin. Ini adalah open source dan menggunakan grafik aliran data untuk komputasi numerik. Operasi matematika diwakili oleh node dan array data multidimensi; yaitu, tensor diwakili oleh tepi grafik. Kerangka kerja ini sangat teknis dan oleh karena itu mungkin sulit bagi analis data. Keras membuat pengkodean jaringan neural dalam menjadi sederhana. Ini juga berjalan mulus pada mesin CPU dan GPU.

Sebuah **Model** adalah struktur data inti Keras . Model sekuensial, yang terdiri dari tumpukan linier lapisan, adalah jenis model yang paling sederhana. Ini menyediakan fungsi umum, seperti `fit()`, `evaluate()`, dan `compile()`.

Anda dapat membuat model sekuensial dengan bantuan baris kode berikut:

```

dari keras.models import Sequential
# Membuat para Sequential Model
Model = Sequential ()

```

Lapisan dalam model Keras

Lapisan Keras sama seperti lapisan jaringan saraf. Ada lapisan yang sepenuhnya terhubung, lapisan kolam maksimum, dan lapisan aktivasi. Sebuah lapisan dapat ditambahkan ke model menggunakan fungsi model `add()`. Misalnya, model sederhana dapat direpresentasikan sebagai berikut:

```

dari keras.models import Sequential
dari keras.layers.core import Dense, Activation, Flatten
# Membuat para Sequential Model
Model = Sequential ()
#Lapisan 1 - Menambahkan model lapisan yang rata

```

```
. tambahkan (Ratakan (bentuk_input = ( 32 , 32 , 3 )))

#Layer 2 - Menambahkan model lapisan yang terhubung sepenuhnya
. tambahkan (Padat ( 100 ))

#Layer 3 - Menambahkan model lapisan aktivasi ULT
. tambahkan (Aktivasi ( 'relu' ))

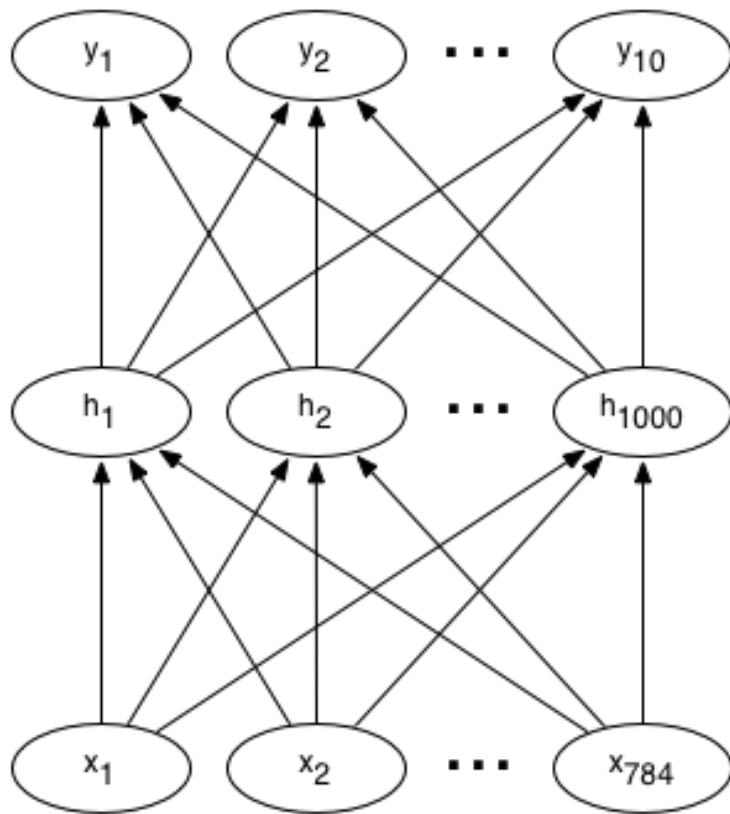
#Layer 4- Menambahkan model lapisan yang terhubung sepenuhnya
. tambahkan (Padat ( 60 ))

#Layer 5 - Menambahkan model lapisan aktivasi ULT
. tambahkan (Aktivasi ( 'relu' ))
```

Keras akan secara otomatis menyimpulkan bentuk semua lapisan setelah lapisan pertama. Ini berarti Anda hanya perlu mengatur dimensi masukan untuk lapisan pertama. Lapisan pertama dari cuplikan kode sebelumnya ```model.add(Flatten(input_shape=(32, 32, 3)))``` menetapkan dimensi masukan ke (32, 32, 3) dan dimensi keluaran ke (3072 = 32 x 32 x 3). Lapisan kedua menerima keluaran dari lapisan pertama dan menyetel dimensi keluaran ke (100). Rantai penerusan keluaran ke lapisan berikutnya berlanjut hingga lapisan terakhir, yang merupakan keluaran model.

Pengenalan nomor tulisan tangan dengan Keras dan MNIST

Jaringan saraf tipikal untuk pengenal digit mungkin memiliki 784 piksel input yang terhubung ke 1.000 neuron di lapisan tersembunyi, yang kemudian menghubungkan ke 10 target keluaran - satu untuk setiap digit. Setiap lapisan terhubung sepenuhnya ke lapisan di atas. Representasi grafis dari jaringan ini ditunjukkan sebagai berikut, di mana `xinput`, `hneuron` tersembunyi, dan `yvariabel kelas output`:



Di notebook ini, kami akan membangun jaringan saraf yang akan mengenali angka tulisan tangan dari 0-9.

Jenis jaringan saraf yang kami bangun digunakan di sejumlah aplikasi dunia nyata, seperti mengenali nomor telepon dan menyortir surat pos berdasarkan alamat. Untuk membangun jaringan ini, kami akan menggunakan dataset **MNIST**.

Kami akan mulai seperti yang ditunjukkan pada kode berikut dengan mengimpor semua modul yang diperlukan, setelah itu data akan dimuat, dan akhirnya membangun jaringan:

```
# Impor data Numpy, keras dan MNIST
import numpy sebagai np
import matplotlib.pyplot sebagai plt

dari keras.datasets import mnist
dari keras.models import Sequential
dari keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

Mengambil data pelatihan dan pengujian

Dataset MNIST sudah terdiri dari data pelatihan dan pengujian. Ada 60.000 titik data data latih dan 10.000 titik data uji. Jika Anda tidak memiliki file data secara lokal di `'~/.keras/datasets/'` jalur, itu dapat diunduh di lokasi ini.

Setiap titik data MNIST memiliki:

- Gambar digit tulisan tangan
- Label yang sesuai yaitu angka dari 0-9 untuk membantu mengidentifikasi gambar

Gambar akan dipanggil, dan akan menjadi masukan ke jaringan saraf kita, **X** ; label yang sesuai adalah **y** .

Kami ingin label kami sebagai vektor one-hot. Vektor satu-panas adalah vektor banyak nol dan satu. Paling mudah untuk melihat ini sebagai contoh. Angka 0 direpresentasikan sebagai [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dan 4 direpresentasikan sebagai [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] sebagai vektor satu panas.

Data yang diratakan

Kami akan menggunakan data yang diratakan dalam contoh ini, atau representasi gambar MNIST dalam satu dimensi daripada dua juga dapat digunakan. Dengan demikian, setiap gambar angka 28 x 28 piksel akan direpresentasikan sebagai larik 1 dimensi 784 piksel.

Dengan meratakan data, informasi tentang struktur 2D gambar dilempar; namun, data kami disederhanakan. Dengan bantuan ini, semua data pelatihan kami dapat disimpan dalam satu larik bentuk (60.000, 784), di mana dimensi pertama mewakili jumlah gambar pelatihan dan yang kedua menggambarkan jumlah piksel di setiap gambar. Jenis data ini mudah dianalisis menggunakan jaringan neural sederhana, sebagai berikut:

```
# Mengambil data pelatihan dan pengujian
(X_train, y_train), (X_test, y_test) = mnist.load_data()

mencetak ('X_train bentuk:' , X_train . bentuk )
print ('bentuk X_test:' , X_test . bentuk )
print ('bentuk y_train:' , y_train . bentuk )
print ('bentuk y_test:' , y_test . bentuk )
```

Memvisualisasikan data pelatihan

Fungsi berikut akan membantu Anda memvisualisasikan data MNIST. Dengan meneruskan indeks contoh pelatihan, fungsi tersebut akan menampilkan gambar pelatihan itu bersama dengan label yang sesuai di judul:`show_digit`

```
# Visualisasikan
impor data matplotlib.pyplot sebagai plt
% matplotlib inline

#Menampilkan gambar pelatihan dengan indeksnya di set MNIST
def display_digit ( indeks ):
    label = y_train [ index ] . argmax ( axis = 0 )
    image = X_train [ indeks ]
    plt . title ( 'Data pelatihan, indeks: % d , Label: % d ' % ( indeks , label ))
    plt . imshow ( gambar , cmap = 'grey_r' )
    plt . tampilan ()
```

```

# Menampilkan gambar pelatihan
display_digit (0) pertama (indeks 0 )

X_train = X_train . membentuk kembali ( 60000 , 784 )
X_test = X_test . membentuk kembali ( 10000 , 784 )
X_train = X_train . astype ( 'float32' )
X_test = X_test . astype ( 'float32' )
X_train /= 255
X_test /= 255
print ( "Latih bentuk matriks" , bentuk X_train . )
print ( "Uji bentuk matriks" , bentuk X_test . )

#One Hot encoding label.
dari keras.utils import np_utils
print ( y_train . shape )
y_train = np_utils . to_categorical ( y_train , 10 )
y_test = np_utils . to_categorical ( y_test , 10 )
print ( y_train . shape )

```

Membangun jaringan

Untuk contoh ini, Anda akan menentukan yang berikut:

- Lapisan masukan, yang Anda harapkan untuk setiap bagian data MNIST , karena memberi tahu jaringan jumlah masukan
- Lapisan tersembunyi, karena mereka mengenali pola dalam data dan juga menghubungkan lapisan masukan ke lapisan keluaran
- Lapisan keluaran, yang mendefinisikan bagaimana jaringan belajar dan memberi label sebagai keluaran untuk gambar yang diberikan, sebagai berikut:

```

# Mendefinisikan jaringan neural
def build_model ():
    model = Sequential ()
    model . tambahkan model ( Dense ( 512 , input_shape = ( 784 , )) )
    . add ( Activation ( 'relu' )) # Sebuah "aktivasi" hanyalah fungsi non-linier yang diterapkan
    ke output # dari lapisan di atas. Dalam kasus ini, dengan "unit linier yang diperbaiki", # kita
    melakukan penjepitan pada semua nilai di bawah 0 hingga 0.

    model . add ( Dropout ( 0.2 )) #Dengan bantuan Dropout membantu kita dapat melindungi model
    dari menghafal atau "overfitting"
    model data pelatihan . tambahkan model ( Dense ( 512 ))
    . tambahkan model ( Aktivasi ( 'relu' )) . tambahkan model ( Dropout ( 0.2 )) . tambahkan model
    ( Dense ( 10 )) . tambahkan ( Aktivasi (
        'softmax' )) # Aktivasi "softmax" khusus ini, # Ini
        juga memastikan bahwa output adalah distribusi probabilitas yang valid, #
        Berarti bahwa nilai yang diperoleh semuanya non-negatif dan berjumlah 1.
        model pengembalian

#Membangun model
model = build_model ()

model . kompilasi ( pengoptimal = 'rmsprop' ,
    loss = 'kategorikal_crossentropy' ,
    metrik = [ 'akurasi' ])

```

Melatih jaringan

Sekarang setelah kami membangun jaringan, kami memberinya makan dengan data dan melatihnya, sebagai berikut:

```
| #  
| Model pelatihan . fit ( X_train , y_train , batch_size = 128 , nb_epoch = 4 , verbose = 1 ,  
| validation_data = ( X_test , y_test ))
```

Menguji

Setelah Anda puas dengan output dan akurasi pelatihan, Anda dapat menjalankan jaringan pada **set data pengujian** untuk mengukur performanya!

Ingatlah untuk melakukan ini hanya setelah Anda menyelesaikan pelatihan dan puas dengan hasilnya.

Hasil yang baik akan memperoleh akurasi **lebih dari 95%**. Beberapa model sederhana telah dikenal untuk mencapai akurasi hingga 99,7%! Kami dapat menguji model, seperti yang ditunjukkan di sini:

```
| # Membandingkan label yang diprediksi oleh model kita dengan label sebenarnya  
| skor = model . evaluasi ( X_test , y_test , batch_size = 32 , verbose = 1 , sample_weight =  
| None )  
| # Mencetak hasil  
| cetak ( 'Test score:' , score [ 0 ])  
| print ( 'Test akurasi:' , skor [ 1 ])
```

Memahami propagasi mundur

Di bagian ini, kita akan memahami intuisi tentang propagasi mundur. Ini adalah cara menghitung gradien menggunakan aturan rantai. Memahami proses ini dan kehalusannya sangat penting bagi Anda untuk dapat memahami dan mengembangkan, merancang, dan men-debug jaringan neural secara efektif.

Secara umum, diberikan fungsi $f(x)$, di mana x adalah vektor input, kita ingin menghitung gradien dari f pada x yang dilambangkan dengan $\nabla(f(x))$. Ini karena dalam kasus jaringan neural, fungsi f pada dasarnya adalah fungsi kerugian (L) dan input x adalah kombinasi bobot dan data pelatihan. Simbol ∇ diucapkan sebagai **nabla**:

$$(x_i, y_i) \quad i = 1 \dots N$$

Mengapa kita mengambil gradien pada parameter bobot?

Diketahui bahwa data pelatihan biasanya tetap dan parameternya adalah variabel yang dapat kita kendalikan. Kami biasanya menghitung gradien parameter sehingga kami

dapat menggunakannya untuk pembaruan parameter. Gradien ∇f adalah vektor turunan parsial, yaitu:

$$\nabla f = [df/dx, df/dy] = [y, x]$$

Singkatnya, propagasi mundur akan terdiri dari:

- Melakukan operasi umpan maju
- Membandingkan keluaran model dengan keluaran yang diinginkan
- Menghitung kesalahan
- Menjalankan operasi umpan maju mundur (propagasi mundur) untuk menyebarluaskan kesalahan ke setiap bobot
- Menggunakan ini untuk memperbarui bobot, dan mendapatkan model yang lebih baik
- Melanjutkan ini hingga kami memiliki model yang bagus

Kami akan membangun jaringan saraf yang mengenali angka dari 0 hingga 9. Aplikasi jaringan semacam ini digunakan untuk menyortir surat pos dengan kode pos, mengenali nomor telepon dan nomor rumah dari gambar, mengekstrak jumlah paket dari gambar paket dan sebagainya .

Dalam banyak kasus, propagasi mundur diterapkan dalam kerangka kerja, seperti TensorFlow. Namun, tidak selalu benar bahwa hanya dengan menambahkan sejumlah sembarang lapisan tersembunyi, propagasi mundur akan bekerja secara ajaib pada kumpulan data. Faktanya adalah jika inisialisasi bobot ceroboh, fungsi non linieritas ini dapat menjenuhkan dan berhenti belajar. Itu berarti kerugian pelatihan akan datar dan menolak untuk turun. Ini dikenal sebagai **masalah gradien lenyap** .

Jika bobot matriks W Anda diinisialisasi terlalu besar, hasil perkalian matriks juga mungkin memiliki rentang yang sangat besar, yang pada gilirannya akan membuat semua keluaran dalam vektor z hampir biner: baik 1 atau 0. Namun, jika ini adalah kasus, maka $z * (1-z)$, yang merupakan gradien lokal dari non-linearitas sigmoid, akan menjadi *nol* (menghilang) dalam kedua kasus , yang akan membuat gradien untuk x dan W juga nol. Lewat mundur lainnya juga akan menghasilkan nol mulai dari titik ini dan seterusnya karena perkalian dalam aturan rantai.

Fungsi aktivasi nonlinier lainnya adalah ReLU, yang membatasi neuron pada nol yang ditunjukkan sebagai berikut. Penerusan maju dan mundur untuk lapisan yang sepenuhnya terhubung yang menggunakan ULT pada intinya meliputi:

```
| z = np.maximum (0, np.dot (W, x)) #Merepresentasikan penerusan maju  
| dW = np.outer (z> 0, x) #Merepresentasikan lintasan mundur: gradien lokal untuk W
```

Jika Anda mengamati ini sebentar, Anda akan melihat bahwa jika sebuah neuron dijepit ke nol pada operasi maju (yaitu, $z = 0$, ia tidak menembak), maka bobotnya akan mendapatkan gradien nol. Hal ini dapat menyebabkan apa yang disebut masalah **ULT yang mati** . Ini berarti jika neuron ReLU sayangnya diinisialisasi sedemikian rupa sehingga tidak pernah aktif, atau jika bobot neuron pernah terlempar dengan

pembaruan besar selama pelatihan ke dalam rezim ini, dalam kasus seperti itu neuron ini akan tetap mati secara permanen. Ini mirip dengan kerusakan otak permanen yang tidak bisa dipulihkan. Kadang-kadang, Anda bahkan dapat meneruskan seluruh rangkaian pelatihan melalui jaringan yang terlatih dan akhirnya menyadari bahwa sebagian besar (sekitar 40%) neuron Anda nol sepanjang waktu.

Dalam kalkulus, yang rantai aturan yang digunakan untuk menghitung turunan dari komposisi dari dua atau lebih fungsi. Artinya, jika kita memiliki dua fungsi sebagai f dan g , maka aturan rantai mewakili turunan dari komposisinya $f \circ g$. Itu berfungsi yang memetakan x ke $f(g(x))$ dalam hal derivatif dari f dan g dan produk fungsi dinyatakan sebagai berikut:

$$(f \circ g)' = (f' \circ g) \cdot g'.$$

Ada cara yang lebih eksplisit untuk merepresentasikan ini dalam istilah variabel. Misalkan $F = f \circ g$, atau ekuivalen, $F(x) = f(g(x))$ untuk semua x . Kemudian seseorang juga bisa menulis:

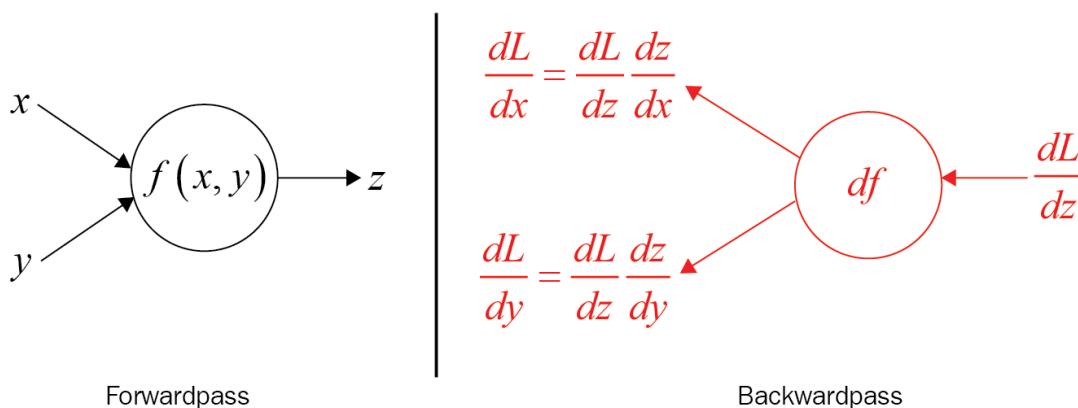
$$F'(x) = f'(g(x)) g'(x).$$

Aturan rantai dapat ditulis dengan bantuan notasi Leibniz dengan cara berikut. Jika sebuah variabel z bergantung pada variabel y , yang pada gilirannya bergantung pada variabel x (seperti y dan z adalah variabel dependen), maka z bergantung pada x juga melalui y antara . Aturan rantai kemudian menyatakan:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}.$$

```
z = 1 / (1 + np.exp(-np.dot(W, x))) # forward pass
dx = np.dot(W.T, z * (1-z)) # backward pass: gradien lokal untuk x
dW = np.outer(z * (1-z), x) # backward pass: gradien lokal untuk W
```

Forward pass di sebelah kiri pada gambar berikut menghitung z sebagai fungsi $f(x, y)$ menggunakan variabel input x dan y . Sisi kanan gambar mewakili umpan mundur. Menerima dL/dz , gradien fungsi kerugian sehubungan dengan z , gradien x dan y pada fungsi kerugian dapat dihitung dengan menerapkan aturan rantai, seperti yang ditunjukkan pada gambar berikut:



Ringkasan

Dalam bab ini, kami meletakkan dasar jaringan saraf dan berjalan melalui jaringan saraf tiruan yang paling sederhana. Kami mempelajari cara membuat jaringan neural lapisan tunggal menggunakan TensorFlow.

Kami mempelajari perbedaan lapisan dalam model Keras dan mendemonstrasikan pengenalan angka tulisan tangan yang terkenal dengan Keras dan MNIST.

Akhirnya, kami memahami apa itu propagasi mundur dan menggunakan kumpulan data MNIST untuk membangun jaringan kami dan melatih serta menguji data kami.

Di bab selanjutnya, kami akan memperkenalkan Anda ke CNN.

Pengantar Jaringan Neural Konvolusional

Convolutional Neural Networks (CNNs) ada di mana-mana. Dalam lima tahun terakhir, kami telah melihat peningkatan dramatis dalam kinerja sistem pengenalan visual karena pengenalan arsitektur mendalam untuk pembelajaran dan klasifikasi fitur. CNN telah mencapai kinerja yang baik di berbagai bidang, seperti pemahaman ucapan otomatis, visi komputer, terjemahan bahasa, mobil tanpa pengemudi, dan game seperti Alpha Go. Jadi, aplikasi CNN hampir tidak terbatas. DeepMind (dari Google) baru-baru ini menerbitkan WaveNet, yang menggunakan CNN untuk menghasilkan ucapan yang meniru suara manusia (<https://deepmind.com/blog/wavenet-generative-model-raw-audio/>).

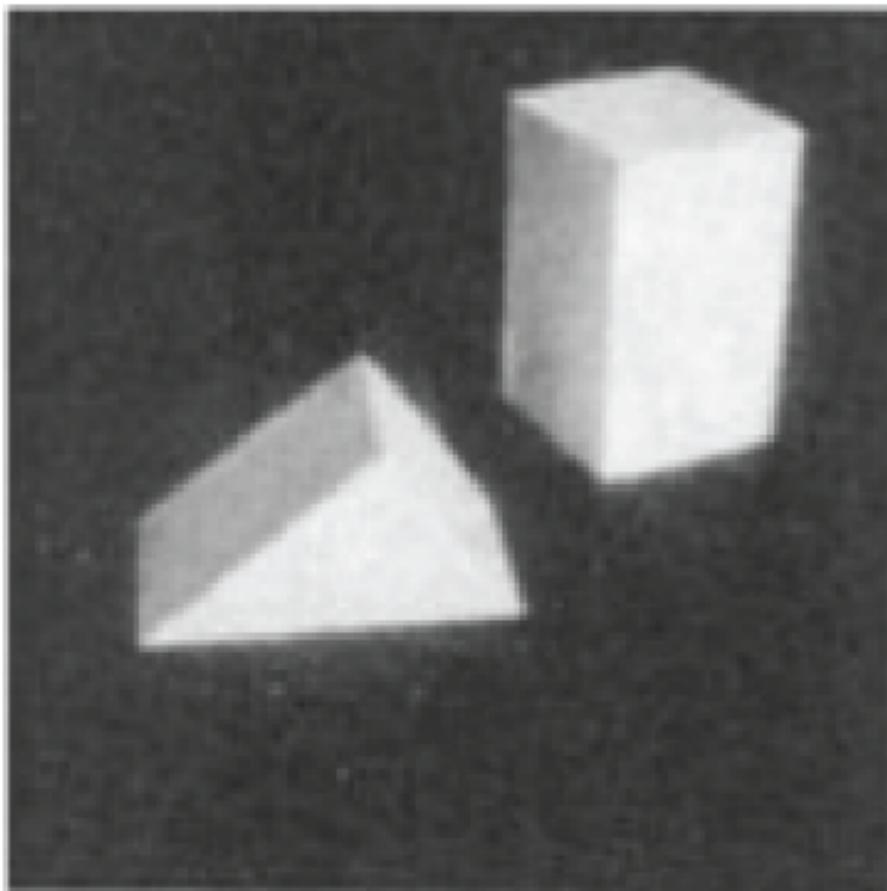
Dalam bab ini, kami akan membahas topik-topik berikut:

- Sejarah CNN
- Gambaran umum CNN
- Augmentasi gambar

Sejarah CNN

Ada banyak upaya untuk mengenali gambar dengan mesin selama beberapa dekade. Merupakan tantangan untuk meniru sistem pengenalan visual dari otak manusia di komputer. Penglihatan manusia adalah yang paling sulit untuk ditiru dan sistem kognitif sensorik otak yang paling kompleks. Kami tidak akan membahas neuron biologis di sini, yaitu korteks visual primer, melainkan fokus pada neuron buatan. Benda-benda di dunia fisik adalah tiga dimensi, sedangkan gambar dari benda-benda itu berbentuk dua dimensi. Dalam buku ini, kami akan memperkenalkan jaringan saraf tanpa menarik analogi otak. Pada tahun 1963, ilmuwan komputer Larry Roberts, yang

juga dikenal sebagai **bapak computer vision**, mendeskripsikan kemungkinan mengekstraksi informasi geometris 3D dari tampilan blok perspektif 2D. dalam disertasi penelitiannya yang berjudul **BLOCK WORLD**. Ini adalah terobosan pertama dalam dunia computer vision. Banyak peneliti di seluruh dunia dalam pembelajaran mesin dan kecerdasan buatan mengikuti pekerjaan ini dan mempelajari visi komputer dalam konteks BLOCK WORLD. Manusia dapat mengenali balok apa pun orientasi atau perubahan pencahayaan yang mungkin terjadi. Dalam disertasi ini, menurutnya penting untuk memahami bentuk sederhana yang menyerupai tepi pada gambar. Dia mengekstrak bentuk seperti tepi ini dari balok untuk membuat komputer mengerti bahwa kedua balok ini sama terlepas dari orientasinya:



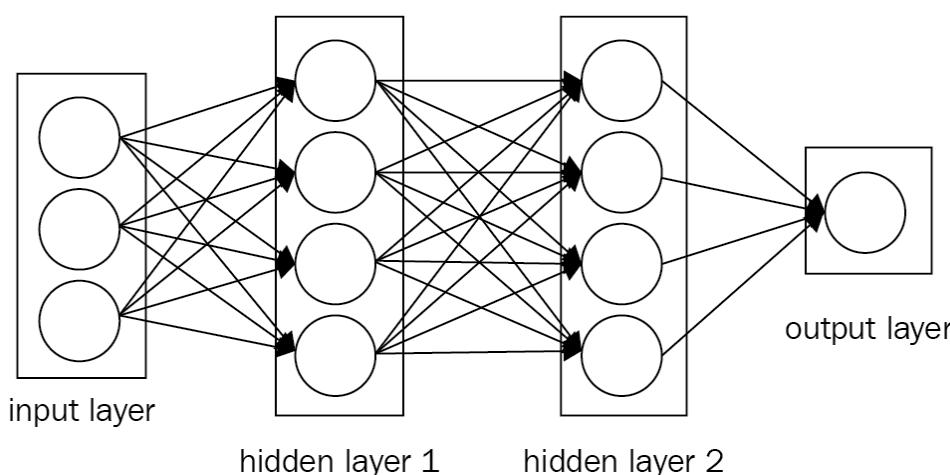
Input Image

Visi dimulai dengan struktur sederhana. Ini adalah awal dari visi komputer sebagai model rekayasa. David Mark, seorang ilmuwan computer vision MIT, memberi kami konsep penting berikutnya, yaitu visi itu hierarkis. Dia menulis sebuah buku yang sangat berpengaruh bernama *VISION*. Ini adalah buku sederhana. Ia mengatakan bahwa suatu citra terdiri dari beberapa lapisan. Kedua prinsip ini membentuk dasar arsitektur pembelajaran mendalam, meskipun tidak memberi tahu kita model matematika seperti apa yang akan digunakan.

Pada 1970-an, algoritme pengenalan visual pertama, yang dikenal sebagai **model silinder umum**, berasal dari lab AI di Universitas Stanford. Ideanya di sini adalah bahwa dunia terdiri dari bentuk-bentuk sederhana dan objek dunia nyata apa pun adalah kombinasi dari bentuk-bentuk sederhana ini. Pada saat yang sama, model lain, yang dikenal sebagai **model struktur bergambar**, diterbitkan dari SRI Inc. Konsepnya masih sama dengan model silinder umum, tetapi bagian-bagiannya dihubungkan oleh pegas; dengan demikian, ini memperkenalkan konsep variabilitas. Algoritma pengenalan visual pertama digunakan di kamera digital oleh Fujifilm pada tahun 2006.

Jaringan saraf konvolisional

CNN, atau ConvNets, sangat mirip dengan jaringan neural biasa. Mereka masih terdiri dari neuron dengan bobot yang dapat dipelajari dari data. Setiap neuron menerima beberapa masukan dan melakukan perkalian titik. Mereka masih memiliki fungsi kerugian pada lapisan terakhir yang terhubung sepenuhnya. Mereka masih dapat menggunakan fungsi nonlinier. Semua tip dan teknik yang kita pelajari dari bab sebelumnya masih berlaku untuk CNN. Seperti yang kita lihat di bab sebelumnya, jaringan saraf biasa menerima data masukan sebagai vektor tunggal dan melewati serangkaian lapisan tersembunyi.. Setiap lapisan tersembunyi terdiri dari satu set neuron, di mana setiap neuron terhubung sepenuhnya ke semua neuron lain di lapisan sebelumnya. Dalam satu lapisan, setiap neuron benar-benar independen dan mereka tidak berbagi koneksi apa pun. Lapisan terkoneksi penuh terakhir, juga disebut **lapisan keluaran**, berisi skor kelas dalam kasus masalah klasifikasi gambar. Umumnya, ada tiga lapisan utama dalam ConvNet sederhana. Mereka adalah **lapisan konvolusi**, **lapisan penggabungan**, dan **lapisan yang sepenuhnya terhubung**. Jaringan saraf sederhana dapat kita lihat pada gambar berikut:



Jaringan neural tiga lapis biasa

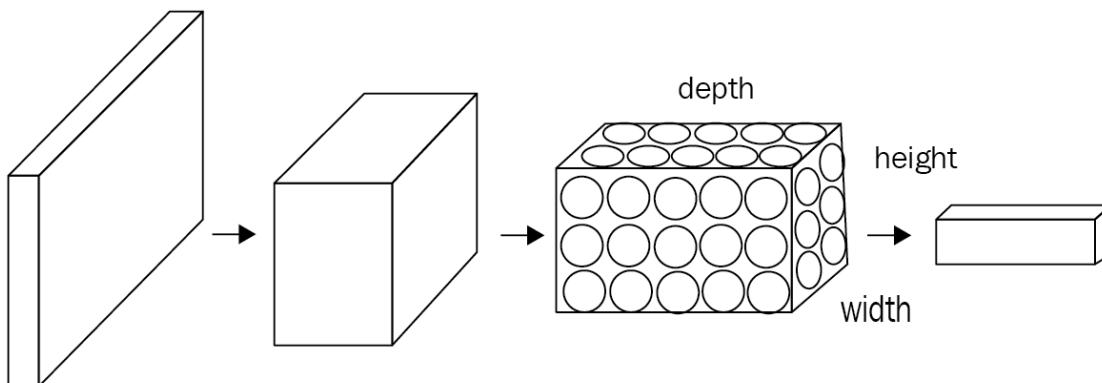
Jadi, apa yang berubah? Karena CNN kebanyakan mengambil gambar sebagai masukan, ini memungkinkan kita untuk menyandikan beberapa properti ke dalam jaringan, sehingga mengurangi jumlah parameter.

Dalam kasus data gambar dunia nyata, CNN berkinerja lebih baik daripada **Multi-Layer Perceptrons (MLPs)**. Ada dua alasan untuk ini:

- Pada bab terakhir, kita melihat bahwa untuk memasukkan gambar ke MLP, kita mengubah matriks masukan menjadi vektor numerik sederhana tanpa struktur spasial. Tidak diketahui bahwa angka-angka ini disusun secara spasial. Jadi, CNN dibuat karena alasan ini; yaitu untuk menjelaskan pola dalam data multidimensi. Tidak seperti MLP, CNN memahami fakta bahwa piksel gambar yang berdekatan satu sama lain lebih terkait erat daripada piksel yang jaraknya lebih jauh:

$$CNN = \text{Lapisan masukan} + \text{lapisan tersembunyi} + \text{lapisan yang terhubung sepenuhnya}$$

- CNN berbeda dari MLP dalam jenis lapisan tersembunyi yang dapat disertakan dalam model. Sebuah ConvNet mengatur neuronnya dalam tiga dimensi: **lebar**, **tinggi**, dan **kedalaman**. Setiap lapisan mengubah volume masukan 3D menjadi volume keluaran 3D neuron menggunakan fungsi aktivasi. Misalnya, pada gambar berikut, input layer merah menahan gambar. Jadi lebar dan tingginya adalah dimensi gambar, dan kedalamannya tiga karena ada saluran Merah, Hijau, dan Biru:



ConvNets adalah jaringan neural dalam yang membagikan parameternya di seluruh ruang.

Bagaimana komputer menginterpretasikan gambar?

Pada dasarnya, setiap gambar dapat direpresentasikan sebagai matriks nilai piksel.

Dengan kata lain, gambar dapat dianggap sebagai fungsi (f) yang memetakan dari R^2 ke R .

$f(x, y)$ memberikan nilai intensitas pada posisi (x, y) . Dalam praktiknya, nilai fungsinya hanya berkisar dari 0 hingga 255. Demikian pula, gambar berwarna dapat direpresentasikan sebagai tumpukan tiga fungsi. Kita dapat menulis ini sebagai vektor:

$$f(x, y) = [r(x, y) \ g(x, y) \ b(x, y)]$$

Atau kita bisa menulis ini sebagai pemetaan:

$$f: R \times R \rightarrow R^3$$

Jadi, gambar berwarna juga merupakan suatu fungsi, tetapi dalam hal ini, nilai pada setiap posisi (x, y) bukanlah angka tunggal. Sebaliknya ini adalah vektor yang memiliki tiga intensitas cahaya berbeda yang sesuai dengan tiga saluran warna. Berikut ini adalah kode untuk melihat detail suatu gambar sebagai input ke komputer.

Kode untuk memvisualisasikan gambar

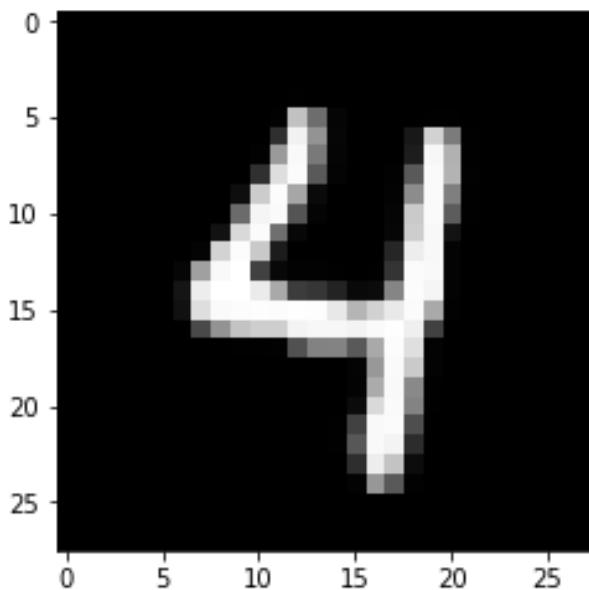
Mari kita lihat bagaimana sebuah gambar dapat divisualisasikan dengan kode berikut:

```
#import semua lib
impor yang diperlukan matplotlib.pyplot sebagai plt
%matplotlib
impor sebaris numpy sebagai np
dari skimage.io impor imread
dari skimage.transform impor ubah ukuran

# Muat gambar berwarna dalam grayscale
image = imread ('sample_digit.png', as_grey = True)
image = resize (image, (28,28), mode = 'reflect')
print ('This image is:', type ( image),
      'dengan dimensi:', image.shape)

plt.imshow (image, cmap = 'grey')
```

Kami mendapatkan gambar berikut sebagai hasilnya:



```
def visualize_input (img, ax):  
    ax.imshow (img, cmap = 'grey')  
    width, height = img.shape  
    thresh  
    = img.max () / 2.5  
    untuk x dalam rentang (lebar):  
        untuk y dalam rentang (tinggi ):  
            ax.annotate (str (round (img [x] [y], 2)), xy = (y, x),  
                         horizontalalignment = 'center',  
                         verticalalignment = 'center',  
                         color = 'white' if img [ x] [y] <thresh else 'black')  
  
fig = plt.figure (figsize = (12,12))  
ax = fig.add_subplot (111)  
visualize_input (image, ax)
```

Hasil berikut diperoleh:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|-------|------|------|------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|--|--|
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 0.75 | 0.43 | 0.02 | 0.0 | 0.0 | 0.0 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.18 | 0.94 | 0.58 | 0.03 | 0.0 | 0.0 | 0.0 | 0.09 | 0.81 | 0.48 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 0.6 | 0.97 | 0.47 | 0.02 | 0.0 | 0.0 | 0.11 | 0.96 | 0.69 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.19 | 0.86 | 0.97 | 0.35 | 0.02 | 0.0 | 0.0 | 0.01 | 0.35 | 0.97 | 0.7 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.06 | 0.79 | 0.98 | 0.64 | 0.04 | 0.0 | 0.0 | 0.01 | 0.54 | 0.97 | 0.49 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 10 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.42 | 0.95 | 0.97 | 0.33 | 0.02 | 0.0 | 0.0 | 0.02 | 0.79 | 0.97 | 0.36 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.07 | 0.84 | 0.97 | 0.43 | 0.03 | 0.0 | 0.0 | 0.03 | 0.8 | 0.96 | 0.08 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.02 | 0.82 | 0.98 | 0.78 | 0.11 | 0.0 | 0.0 | 0.0 | 0.15 | 0.92 | 0.96 | 0.02 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.02 | 0.62 | 0.97 | 0.99 | 0.26 | 0.03 | 0.01 | 0.01 | 0.0 | 0.01 | 0.21 | 0.97 | 0.97 | 0.02 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.08 | 0.92 | 0.99 | 0.99 | 0.91 | 0.66 | 0.24 | 0.21 | 0.14 | 0.04 | 0.05 | 0.52 | 0.98 | 0.95 | 0.02 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 15 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.07 | 0.86 | 0.97 | 0.98 | 0.98 | 0.98 | 0.99 | 0.98 | 0.9 | 0.7 | 0.81 | 0.9 | 0.98 | 0.66 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.29 | 0.57 | 0.76 | 0.8 | 0.8 | 0.94 | 0.96 | 0.96 | 0.95 | 0.97 | 0.99 | 0.93 | 0.26 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 0.01 | 0.32 | 0.5 | 0.51 | 0.35 | 0.66 | 0.98 | 0.87 | 0.02 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.02 | 0.59 | 0.97 | 0.79 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.02 | 0.66 | 0.97 | 0.53 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| 20 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.04 | 0.84 | 0.97 | 0.51 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 0.26 | 0.97 | 0.97 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 0.34 | 0.97 | 0.95 | 0.18 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.018 | 0.92 | 0.76 | 0.05 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.001 | 0.59 | 0.35 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | |
| 25 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | |
| 0 | 0 | 5 | 10 | 15 | 20 | 25 | | | | | | | | | | | | | | | | | | | | | | | | | |

Pada bab sebelumnya, kami menggunakan pendekatan berbasis MLP untuk mengenali gambar. Ada dua masalah dengan pendekatan itu:

- Ini meningkatkan jumlah parameter
- Ia hanya menerima vektor sebagai input, yaitu meratakan matriks menjadi vektor

Artinya kita harus menemukan cara baru untuk memproses gambar, di mana informasi 2D tidak hilang sama sekali. CNN mengatasi masalah ini. Selanjutnya, CNN menerima matriks sebagai masukan. Lapisan konvolusional mempertahankan struktur spasial. Pertama, kita mendefinisikan jendela konvolusi, juga disebut **filter**, atau **kernel**; lalu geser ini ke atas gambar.

Keluar

Jaringan saraf dapat dianggap sebagai masalah penelusuran. Setiap node dalam jaringan saraf mencari korelasi antara data masukan dan data keluaran yang benar.

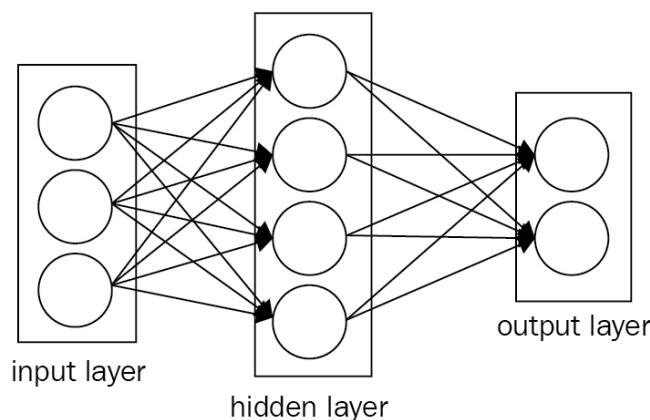
Putus sekolah secara acak mematikan node saat menyebar ke depan dan dengan demikian membantu mencegah bobot agar tidak berkumpul ke posisi yang identik. Setelah ini selesai, ini akan mengaktifkan semua node dan melakukan propagasi balik.

Demikian pula, kita dapat menetapkan beberapa nilai lapisan ke nol secara acak selama propagasi maju untuk melakukan pelepasan pada lapisan.

Gunakan dropout hanya selama pelatihan. Jangan gunakan saat runtime atau di set data pengujian Anda.

Lapisan masukan

The **lapisan input** memegang data gambar. Pada gambar berikut, lapisan masukan terdiri dari tiga masukan. Dalam **lapisan yang sepenuhnya terhubung**, neuron di antara dua lapisan yang berdekatan terhubung sepenuhnya secara berpasangan tetapi tidak berbagi koneksi apa pun di dalam lapisan. Dengan kata lain, neuron di lapisan ini memiliki koneksi penuh ke semua aktivasi di lapisan sebelumnya. Oleh karena itu, aktivasi mereka dapat dihitung dengan perkalian matriks sederhana, secara opsional menambahkan suku bias. Perbedaan antara lapisan yang sepenuhnya terhubung dan konvolusional adalah bahwa neuron dalam lapisan konvolusional terhubung ke wilayah lokal dalam masukan, dan bahwa mereka juga berbagi parameter:



Lapisan konvolusional

Tujuan utama konvolusi dalam kaitannya dengan ConvNet adalah untuk mengekstrak fitur dari gambar masukan. Lapisan ini melakukan sebagian besar komputasi di ConvNet. Kami tidak akan membahas detail matematis dari konvolusi di sini tetapi akan mendapatkan pemahaman tentang cara kerjanya di atas gambar.

Fungsi aktivasi ULT sangat berguna di CNN.

Lapisan konvolusional di Keras

Untuk membuat lapisan konvolusional di Keras, Anda harus mengimpor modul yang diperlukan sebagai berikut:

```
| dari keras.layers import Conv2D
```

Kemudian, Anda dapat membuat lapisan konvolusional dengan menggunakan format berikut:

```
| Konv2D (filter, ukuran_kernel, langkah, padding, aktivasi = 'relu' , bentuk_input)
```

Anda harus meneruskan argumen berikut:

- `filters`: Jumlah filter.
- `kernel_size`: Jumlah yang menentukan tinggi dan lebar jendela konvolusi (persegi). Ada juga beberapa argumen opsional tambahan yang mungkin ingin Anda sesuaikan.
- `strides`: Langkah konvolusi. Jika Anda tidak menentukan apa pun, ini disetel ke satu.
- `padding`: Ini adalah salah satu `valid` atau `same`. Jika Anda tidak menentukan apa pun, pengisi disetel ke `valid`.
- `activation`: Ini biasanya `relu`. Jika Anda tidak menentukan apa pun, tidak ada aktivasi yang diterapkan. Anda sangat dianjurkan untuk menambahkan fungsi aktivasi ULT ke setiap lapisan konvolusional di jaringan Anda.

Dimungkinkan untuk merepresentasikan keduanya `kernel_size` dan `strides` sebagai angka atau tupel.

Saat menggunakan lapisan konvolusional Anda sebagai lapisan pertama (muncul setelah lapisan masukan) dalam model, Anda harus memberikan `input_shape` argumen tambahan—`input_shape`. Ini adalah tupel yang menentukan tinggi, lebar, dan kedalaman (dalam urutan itu) dari input.

Harap pastikan bahwa `input_shape` argumen tidak disertakan jika lapisan konvolusional bukan lapisan pertama di jaringan Anda.

Ada banyak argumen lain yang dapat disetel untuk mengubah perilaku lapisan konvolusional Anda:

- **Contoh 1 :** Untuk membangun CNN dengan lapisan masukan yang menerima gambar berukuran 200 x 200 piksel dalam skala abu-abu. Dalam kasus seperti itu, lapisan berikutnya akan menjadi lapisan konvolusional 16 filter dengan lebar dan tinggi 2. Saat kita melanjutkan dengan konvolusi, kita dapat mengatur filter untuk melompati 2 piksel bersama-sama. Oleh karena itu, kita dapat membuat lapisan konvolusional dengan filter yang tidak menutupi gambar dengan angka nol dengan kode berikut:

```
| Conv2D (filter = 16, kernel_size = 2, step = 2, activation = 'relu', input_shape = (200, 200, 1))
```

- **Contoh 2 :** Setelah kita membangun model CNN kita, kita dapat memiliki lapisan berikutnya di dalamnya menjadi lapisan konvolusional. Lapisan ini akan memiliki 32 filter dengan lebar dan tinggi 3, yang akan mengambil lapisan yang dibangun pada contoh sebelumnya sebagai masukannya. Di sini, saat kita melanjutkan dengan konvolusi, kita akan menyetel filter untuk melompat satu piksel pada satu waktu, sehingga lapisan konvolusional akan dapat melihat semua daerah dari lapisan sebelumnya juga. Lapisan konvolusional seperti itu dapat dibangun dengan bantuan kode berikut:

```
| Konv2D (filter = 32, kernel_size = 3, padding = 'same', activation = 'relu')
```

- **Contoh 3 :** Anda juga dapat membuat lapisan konvolusional di Keras berukuran 2×2 , dengan 64 filter dan fungsi aktivasi ULT. Di sini, konvolusi menggunakan langkah 1 dengan padding disetel ke `valid` dan semua argumen lain disetel ke nilai defaultnya. Lapisan konvolusional seperti itu dapat dibangun menggunakan kode berikut:

```
| Konv2D (64, (2,2), aktivasi = 'relu')
```

Lapisan penggabungan

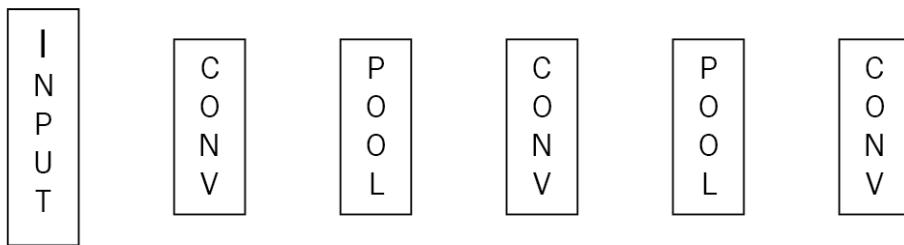
Seperti yang telah kita lihat, lapisan konvolusional adalah tumpukan peta fitur, dengan satu peta fitur untuk setiap filter. Lebih banyak filter meningkatkan dimensi konvolusi. Dimensi yang lebih tinggi menunjukkan lebih banyak parameter. Jadi, lapisan penyatuhan mengontrol overfitting dengan secara progresif mengurangi ukuran spasial representasi untuk mengurangi jumlah parameter dan komputasi. Lapisan penggabungan sering kali menggunakan lapisan konvolusional sebagai masukan. Pendekatan penggabungan yang paling umum digunakan adalah **penggabungan maksimal**. Selain penggabungan maksimal, unit penggabungan juga dapat menjalankan fungsi lain seperti **penggabungan rata - rata**. Di CNN, kita dapat mengontrol perilaku lapisan konvolusional dengan menentukan ukuran setiap filter dan jumlah filter. Untuk meningkatkan jumlah node dalam lapisan konvolusional, kita dapat menambah jumlah filter, dan untuk meningkatkan ukuran pola, kita dapat meningkatkan ukuran filter. Ada juga beberapa hyperparameter lain yang bisa disetel. Salah satunya adalah langkah konvolusi. Langkah adalah jumlah saat filter meluncur di atas gambar. Langkah 1 memindahkan filter sebesar 1 piksel secara horizontal dan vertikal. Di sini, konvolusi menjadi sama dengan lebar dan kedalaman gambar masukan. Langkah 2 membuat lapisan konvolusional setengah dari lebar dan tinggi gambar. Jika filter meluas ke luar gambar, maka kita bisa abaikan nilai yang tidak diketahui ini atau gantikan dengan nol. Ini dikenal sebagai **bantalan**. Di Keras, kita dapat mengatur `padding = 'valid'` apakah boleh kehilangan beberapa nilai. Jika tidak, setel `padding = 'same'`:

Filter



ConvNet yang sangat sederhana terlihat seperti ini:

|N|



Contoh praktis - klasifikasi gambar

Lapisan konvolusional membantu mendeteksi pola regional pada gambar. Lapisan penyatuan maksimal, yang ada setelah lapisan konvolusional , membantu mengurangi dimensi. Berikut adalah contoh klasifikasi gambar menggunakan semua prinsip yang telah kita pelajari di bagian sebelumnya. Satu gagasan penting adalah pertama-tama membuat semua gambar menjadi ukuran standar sebelum melakukan hal lain. Lapisan konvolusi pertama membutuhkan tambahan `input.shape()` parameter. Pada bagian ini, kami akan melatih CNN untuk mengklasifikasikan gambar dari database CIFAR-10. CIFAR-10 adalah kumpulan data 60.000 gambar berwarna dengan ukuran 32 x 32. Gambar-gambar ini diberi label ke dalam 10 kategori dengan masing-masing 6.000 gambar. Kategori tersebut adalah pesawat terbang, mobil, burung, kucing, anjing, rusa, katak, kuda, kapal, dan truk. Mari kita lihat bagaimana melakukan ini dengan kode berikut:

```

impor keras
import numpy sebagai np
import matplotlib.pyplot sebagai plt
%matplotlib inline

fig = plt.figure (figsize = (20,5))
untuk i dalam range (36):
    ax = fig.add_subplot (3, 12, i + 1 , xticks = [], yticks = [])
    ax.imshow (np.squeeze (x_train [i])) dari keras.datasets import cifar10

# rescale [0,255] -> [0,1]
x_train = x_train.astype ('float32') / 255
dari keras.utils import np_utils

# one-hot menyandikan label
num_classes = len (np.unique (y_train))
y_train = keras.utils.to_categorical (y_train, num_classes)
y_test = keras.utils.to_categorical (y_test, num_classes)

# hentikan set pelatihan menjadi set pelatihan dan validasi
(x_train, x_valid) = x_train [5000:], x_train [: 5000]
(y_train, y_valid) = y_train [5000:], y_train [: 5000]

# print bentuk set pelatihan
('x_train shape:', x_train.shape)

# jumlah pencetakan pelatihan, validasi, dan uji gambar
cetak (x_train.shape [0], 'melatih sampel')
cetak (x_test.shape [0], 'sampel uji')
print (x_valid.shape [0], 'validation samples') x_test = x_test.astype ('float32') / 255

dari keras.models import Sequential
dari keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential ( )
model.add (Conv2D (filter = 16, kernel_size = 2, padding = 'same', activation = 'relu',
                  input_shape = (32, 32, 3)))
model.add (MaxPooling2D (pool_size = 2))

```

```

model.add(Conv2D(filter = 32, kernel_size = 2, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2))
model.add(Conv2D(filter = 64, kernel_size = 2), padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2))
model.add(Conv2D(filter = 32, kernel_size = 2, padding = 'sama', aktivasi = 'relu'))
model.add(MaxPooling2D(pool_size = 2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Padat(10, aktivasi = 'softmax'))

model.summary()

# kompilasi model
model.compile(loss = 'kategorikal_crossentropy', pengoptimal = 'rmsprop',
               metrik = ['akurasi'])
dari keras.callbacks import ModelCheckpoint

# latih model
checkpointer = ModelCheckpoint(filepath = 'model.weights.best.hdf5', verbose = 1,
                               save_best_only = True)
hist = model.fit(x_train, y_train, batch_size = 32, epochs = 100,
                  validation_data = (x_valid, y_valid), callbacks = [checkpointer],
                  verbose = 2, acak = True)

```

Augmentasi gambar

Saat melatih model CNN, kami tidak ingin model mengubah prediksi apa pun berdasarkan ukuran, sudut, dan posisi gambar. Gambar direpresentasikan sebagai matriks nilai piksel, sehingga ukuran, sudut, dan posisi berpengaruh besar terhadap nilai piksel. Untuk membuat model lebih invariant-ukuran, kita dapat menambahkan berbagai ukuran gambar ke set pelatihan. Demikian pula, untuk membuat model lebih invariant-rotasi, kita dapat menambahkan gambar dengan sudut yang berbeda. Proses ini dikenal sebagai **augmentasi data gambar**. Ini juga membantu menghindari overfitting. Overfitting terjadi saat model diekspos ke sangat sedikit sampel. Augmentasi data gambar adalah salah satu cara untuk mengurangi overfitting, tetapi itu mungkin tidak cukup karena gambar yang diperbesar masih berkorelasi. Keras menyediakan kelas augmentasi gambar yang disebut `ImageDataGenerator` yang menentukan konfigurasi untuk augmentasi data gambar. Ini juga menyediakan fitur lain seperti:

- Standardisasi berdasarkan sampel dan fitur
- Rotasi acak, pergeseran, geser, dan zoom gambar
- Balik horizontal dan vertikal
- ZCA pemutih
- Penataan ulang dimensi
- Menyimpan perubahan ke disk

Objek generator gambar tambahan dapat dibuat sebagai berikut:

```
| imagedatagen = ImageDataGenerator()
```

API ini menghasilkan kumpulan data gambar tensor dalam augmentasi data waktu nyata, alih-alih memproses seluruh kumpulan data gambar dalam memori. API ini dirancang untuk membuat data gambar tambahan selama proses pemasangan model.

Jadi, ini mengurangi overhead memori tetapi menambahkan beberapa biaya waktu untuk pelatihan model.

Setelah dibuat dan dikonfigurasi, Anda harus menyesuaikan data Anda. Ini menghitung statistik apa pun yang diperlukan untuk melakukan transformasi ke data gambar. Ini dilakukan dengan memanggil `fit()` fungsi pada generator data dan meneruskannya ke set data pelatihan, sebagai berikut:

```
| imagedatagen . pas ( train_data )
```

Ukuran kumpulan dapat dikonfigurasi, pembuat data dapat disiapkan, dan kumpulan gambar dapat diterima dengan memanggil `flow()` fungsi:

```
| imagedatagen.flow (x_train, y_train, batch_size = 32 )
```

Terakhir, panggil `fit_generator()` fungsi alih-alih memanggil `fit()` fungsi pada model:

```
| fit_generator (gambaragen, sample_per_epoch = len (X_train), epochs = 200)
```

Mari kita lihat beberapa contoh untuk memahami cara kerja API augmentasi gambar di Keras. Kami akan menggunakan tugas pengenalan digit tulisan tangan MNIST dalam contoh ini.

Mari kita mulai dengan melihat sembilan gambar pertama dalam set data pelatihan:

```
#Plot gambar
dari keras.datasets import mnist
dari matplotlib import pyplot #loading
data
(X_train, y_train), (X_test, y_test) = mnist.load_data ()
#creating a grid of 3x3 images
for i in range (0, 9):
    pyplot.subplot (330 + 1 + i)
    pyplot.imshow (X_train [i], cmap = pyplot.get_cmap ('grey'))
#Menampilkan plot
pyplot.show ()
```

Cuplikan kode berikut membuat gambar tambahan dari set data CIFAR-10. Kami akan menambahkan gambar-gambar ini ke set pelatihan dari contoh terakhir dan melihat bagaimana akurasi klasifikasi meningkat:

```
dari keras.preprocessing.image import ImageDataGenerator
# membuat dan mengkonfigurasi augmented image generator
datagen_train = ImageDataGenerator (
width_shift_range = 0,1, # menggeser gambar secara acak secara horizontal (10% dari total lebar)
height_shift_range = 0,1, # menggeser gambar secara acak secara vertikal (10% dari total tinggi )
horizontal_flip = True) # membalik gambar secara acak secara horizontal
# membuat dan mengonfigurasi generator gambar tambahan
datagen_valid = ImageDataGenerator (
width_shift_range = 0,1, # menggeser gambar secara acak secara horizontal (10% dari total lebar)
height_shift_range = 0,1, # menggeser gambar secara acak secara vertikal (10% dari tinggi total)
horizontal_flip = True) # membalik gambar secara acak secara horizontal
# pas augmented image generator pada data
datagen_train.fit (x_train)
datagen_valid.fit (x_valid)
```

Ringkasan

Kami memulai bab ini dengan melihat secara singkat sejarah CNN. Kami memperkenalkan Anda pada penerapan visualisasi gambar.

Kami mempelajari klasifikasi gambar dengan bantuan contoh praktis, menggunakan semua prinsip yang kami pelajari di bab ini. Terakhir, kami mempelajari bagaimana augmentasi gambar membantu kami menghindari overfitting dan mempelajari berbagai fitur lain yang disediakan oleh augmentasi gambar.

Pada bab berikutnya, kita akan mempelajari cara membuat model CNN pengklasifikasi gambar sederhana dari awal.

Buat CNN Pertama dan Pengoptimalan Kinerja Anda

Sebuah **jaringan saraf convolutional** (CNN) adalah jenis **jaringan saraf umpan-maju** (FNN) di mana pola konektivitas antara neuron yang terinspirasi oleh korteks visual hewan. Dalam beberapa tahun terakhir, CNN telah mendemonstrasikan kinerja manusia super dalam layanan pencarian gambar, mobil tanpa pengemudi, klasifikasi video otomatis, pengenalan suara, dan **pemrosesan bahasa alami** (NLP).

Mempertimbangkan motivasi tersebut, dalam bab ini, kami akan membuat model CNN sederhana untuk klasifikasi gambar dari awal, diikuti oleh beberapa aspek teoretis, seperti operasi konvolusional dan penggabungan. Kemudian kita akan membahas cara menyetel hyperparameter dan mengoptimalkan waktu pelatihan CNN untuk meningkatkan akurasi klasifikasi. Terakhir, kami akan membangun model CNN kedua dengan mempertimbangkan beberapa praktik terbaik. Singkatnya, topik-topik berikut akan dibahas dalam bab ini:

- Arsitektur CNN dan kekurangan DNN
- Operasi konvolusi dan lapisan penyatuhan
- Membuat dan melatih CNN untuk klasifikasi gambar
- Model optimasi kinerja
- Membuat CNN yang ditingkatkan untuk kinerja yang dioptimalkan

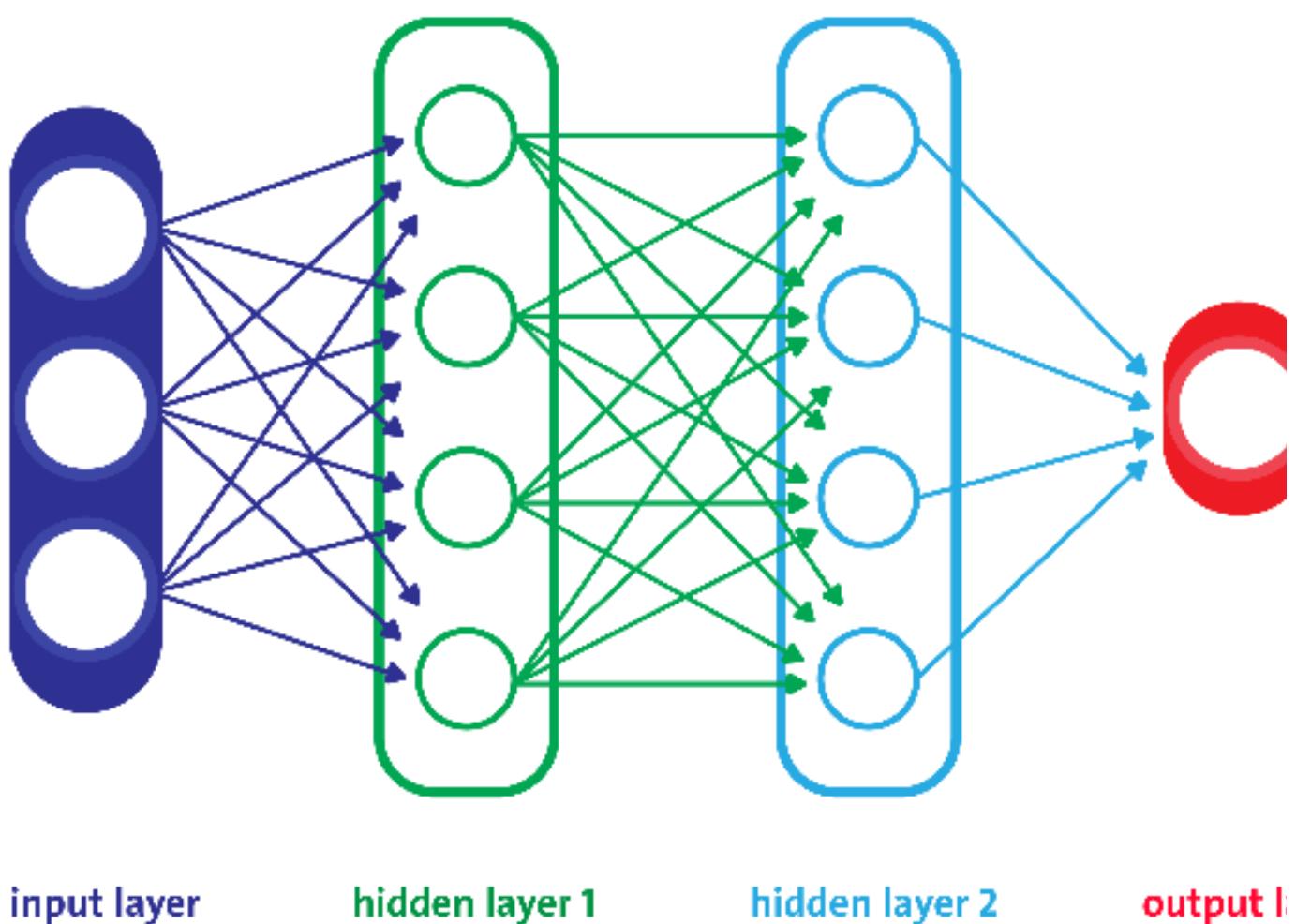
Arsitektur CNN dan kekurangan DNN

Dalam [Bab 2](#) , *Pengantar Jaringan Neural Konvolusional* , kita membahas bahwa perceptron multilayer biasa berfungsi dengan baik untuk gambar kecil (misalnya, MNIST atau CIFAR-10). Namun, ini rusak untuk gambar yang lebih besar karena banyaknya parameter yang diperlukan. Misalnya, gambar 100×100 memiliki 10.000 piksel, dan jika lapisan pertama hanya memiliki 1.000 neuron (yang sudah sangat

membatasi jumlah informasi yang dikirimkan ke lapisan berikutnya), ini berarti 10 juta koneksi; dan itu hanya untuk lapisan pertama.

CNN memecahkan masalah ini menggunakan lapisan yang terhubung sebagian. Karena lapisan berurutan hanya terhubung sebagian dan karena sangat banyak menggunakan kembali bobotnya, CNN memiliki parameter yang jauh lebih sedikit daripada DNN yang terhubung sepenuhnya, yang membuatnya lebih cepat untuk dilatih, mengurangi risiko overfitting, dan memerlukan lebih sedikit data pelatihan. Selain itu, ketika CNN mempelajari kernel yang dapat mendeteksi fitur tertentu, CNN dapat mendeteksi fitur tersebut di mana saja pada gambar. Sebaliknya, ketika DNN mempelajari fitur di satu lokasi, DNN dapat mendeteksinya hanya di lokasi tersebut.

Karena gambar biasanya memiliki fitur yang sangat berulang, CNN dapat menggeneralisasi jauh lebih baik daripada DNN untuk tugas pemrosesan gambar seperti klasifikasi, menggunakan lebih sedikit contoh pelatihan. Yang penting, DNN tidak memiliki pengetahuan sebelumnya tentang bagaimana piksel diatur; itu tidak tahu bahwa piksel terdekat dekat. Arsitektur CNN menyematkan pengetahuan sebelumnya ini. Lapisan bawah biasanya mengidentifikasi fitur di area kecil gambar, sementara lapisan yang lebih tinggi menggabungkan fitur tingkat bawah menjadi fitur yang lebih besar. Ini berfungsi dengan baik pada sebagian besar gambar alami, memberi CNN permulaan yang menentukan dibandingkan dengan DNN:



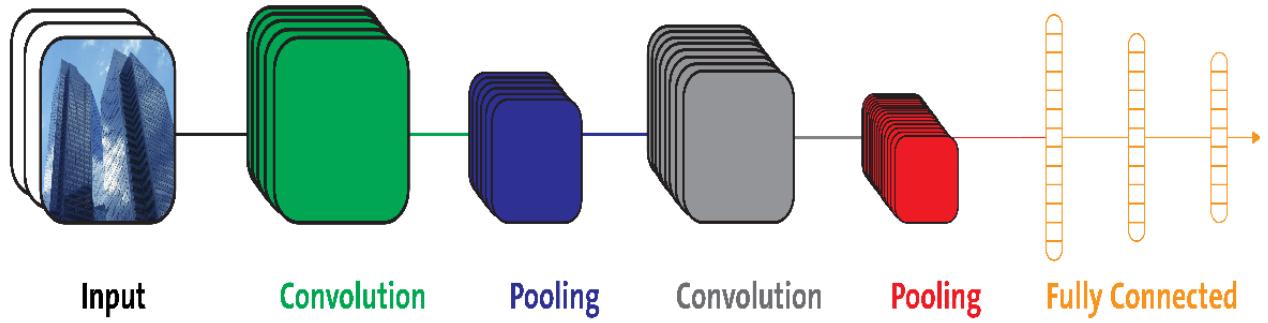
Misalnya, pada *Gambar 1*, di sebelah kiri, Anda dapat melihat jaringan neural tiga lapis biasa. Di sebelah kanan, ConvNet mengatur neuronnya dalam tiga dimensi (lebar, tinggi, dan kedalaman) seperti yang divisualisasikan di salah satu lapisan. Setiap lapisan ConvNet mengubah volume masukan 3D menjadi volume keluaran 3D dari aktivasi neuron. Lapisan masukan merah menampung gambar, jadi lebar dan tingginya akan menjadi dimensi gambar, dan kedalamannya akan menjadi tiga (saluran merah, hijau, dan biru). Oleh karena itu, semua jaringan saraf multilayer yang kami lihat memiliki lapisan yang terdiri dari garis panjang neuron, dan kami harus meratakan gambar atau data masukan ke 1D sebelum memasukkannya ke jaringan saraf.

Namun, apa yang terjadi setelah Anda mencoba memberi mereka gambar 2D secara langsung? Jawabannya adalah bahwa di CNN, setiap lapisan direpresentasikan dalam 2D, yang membuatnya lebih mudah untuk mencocokkan neuron dengan input yang sesuai. Kita akan melihat contohnya di bagian selanjutnya. Fakta penting lainnya adalah bahwa semua neuron dalam peta fitur memiliki parameter yang sama, sehingga secara dramatis mengurangi jumlah parameter dalam model; tetapi yang lebih penting, ini berarti bahwa setelah CNN belajar mengenali pola di satu lokasi, CNN dapat mengenalinya di lokasi lain.

Sebaliknya, setelah DNN biasa belajar mengenali pola di satu lokasi, DNN hanya dapat mengenalinya di lokasi tersebut. Dalam jaringan multilayer seperti MLP atau DBN, keluaran dari semua neuron dari lapisan masukan terhubung ke setiap neuron di lapisan tersembunyi, dan kemudian keluaran akan kembali bertindak sebagai masukan ke lapisan yang sepenuhnya terhubung. Di jaringan CNN, skema koneksi yang mendefinisikan lapisan konvolusional sangat berbeda. Lapisan konvolusional adalah jenis lapisan utama dalam CNN, di mana setiap neuron terhubung ke wilayah tertentu dari area masukan yang disebut **bidang reseptif**.

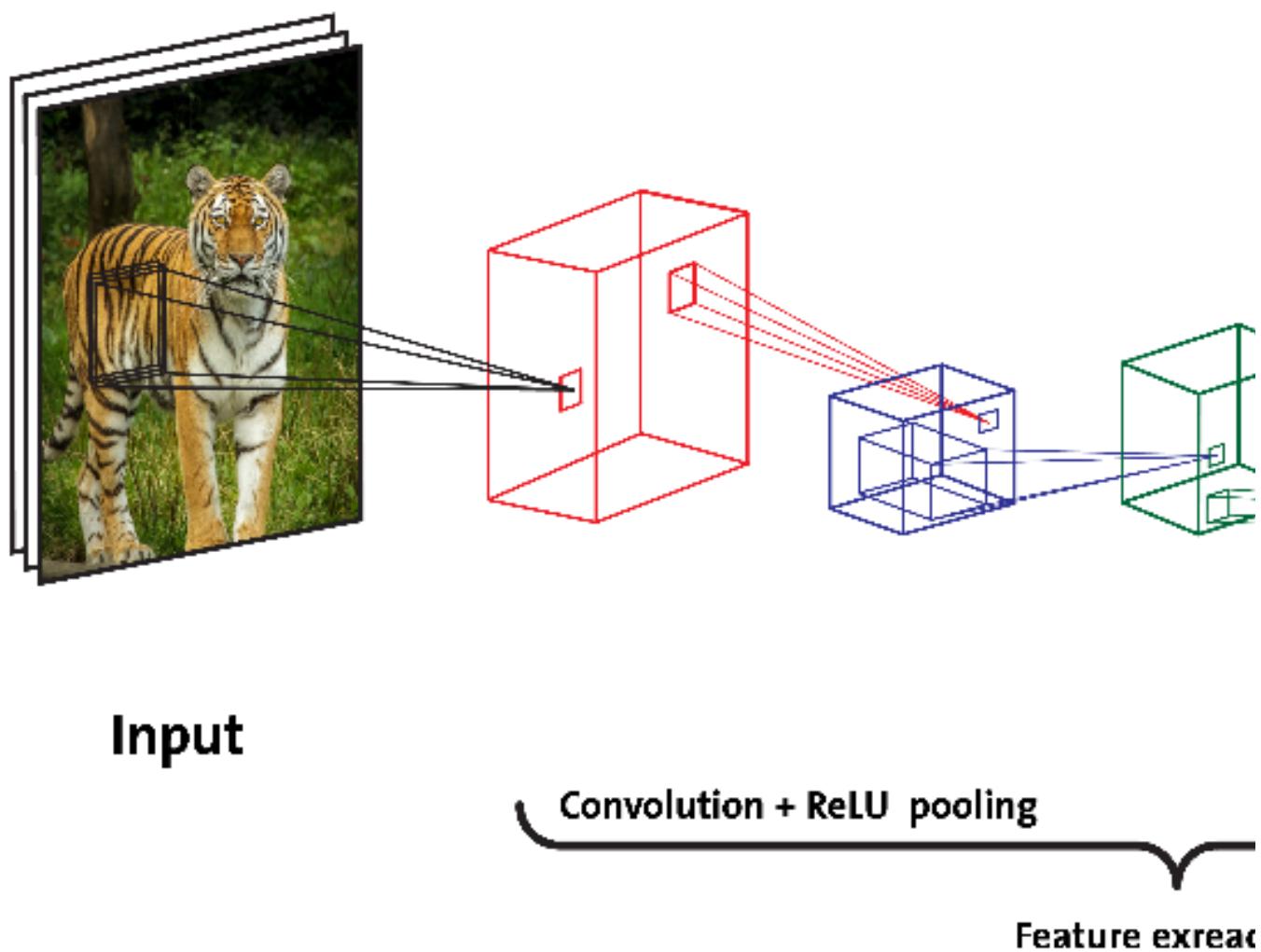
Dalam arsitektur CNN yang khas, beberapa lapisan konvolusional dihubungkan dalam gaya kaskade. Setiap lapisan diikuti oleh lapisan **Rectified Linear Unit (ReLU)**, lalu lapisan penyatuan, lalu satu atau lebih lapisan konvolusional (+ ULANG), lalu lapisan penyatuan lainnya, dan terakhir satu atau lebih lapisan yang terhubung sepenuhnya. Tergantung pada jenis masalahnya, jaringannya mungkin dalam. Output dari setiap lapisan konvolusi adalah sekumpulan objek yang disebut **peta fitur**, yang dihasilkan oleh filter kernel tunggal. Kemudian peta fitur dapat digunakan untuk menentukan masukan baru ke lapisan berikutnya.

Setiap neuron dalam jaringan CNN menghasilkan keluaran, diikuti oleh ambang aktivasi, yang sebanding dengan masukan dan tidak terikat:



Gambar 2: Arsitektur konseptual CNN

Seperti yang Anda lihat pada *Gambar 2* , lapisan penyatuan biasanya ditempatkan setelah lapisan konvolusional (misalnya, antara dua lapisan konvolusional) . Lapisan penggabungan menjadi subkawasan kemudian membagi wilayah konvolusional. Kemudian, satu nilai perwakilan dipilih, baik menggunakan teknik penggabungan maksimal atau penggabungan rata-rata, untuk mengurangi waktu komputasi lapisan berikutnya. Dengan cara ini, CNN dapat dianggap sebagai ekstraktor fitur. Untuk memahami ini lebih jelas, lihat gambar berikut:



Dengan cara ini, ketahanan fitur sehubungan dengan posisi spasialnya juga meningkat. Untuk lebih spesifiknya, ketika peta fitur digunakan sebagai properti gambar dan melewati gambar grayscale, itu menjadi semakin kecil dan semakin kecil saat berkembang melalui jaringan; tetapi biasanya juga semakin dalam, karena lebih banyak peta fitur akan ditambahkan.

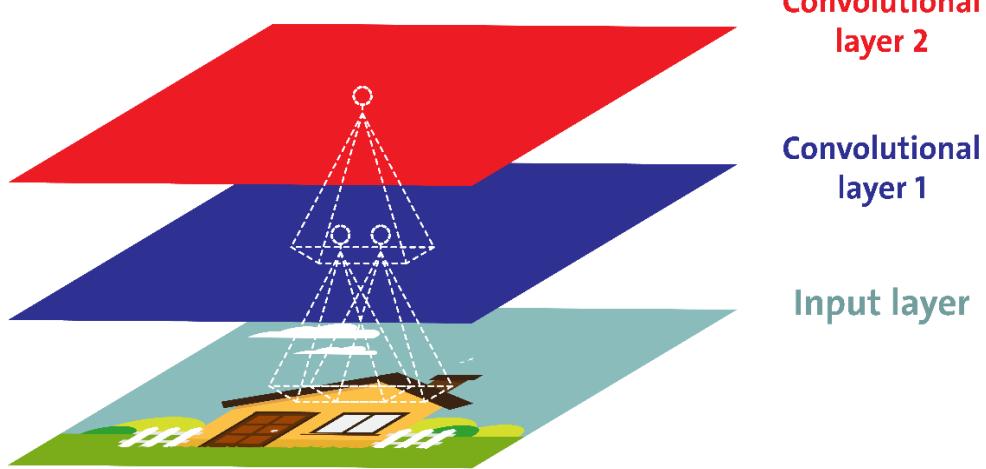
Kita telah membahas batasan FFNN semacam itu - yaitu, jumlah neuron yang sangat tinggi akan diperlukan, bahkan dalam arsitektur yang dangkal, karena ukuran input yang sangat besar yang terkait dengan gambar, di mana setiap piksel adalah variabel yang relevan. Operasi konvolusi memberikan solusi untuk masalah ini karena mengurangi jumlah parameter bebas, memungkinkan jaringan menjadi lebih dalam dengan lebih sedikit parameter.

Operasi konvolusional

Konvolusi adalah operasi matematika yang menggeser satu fungsi ke fungsi lainnya dan mengukur integral dari perkalian pointwise. Ini memiliki koneksi yang dalam dengan transformasi Fourier dan transformasi Laplace dan banyak digunakan dalam pemrosesan sinyal. Lapisan konvolusional sebenarnya menggunakan korelasi silang, yang sangat mirip dengan konvolusi.

Dalam matematika, konvolusi adalah operasi matematika pada dua fungsi yang menghasilkan fungsi ketiga — yaitu, versi modifikasi (berbelit-belit) dari salah satu fungsi aslinya. Fungsi yang dihasilkan memberikan integral dari perkalian titik dua fungsi sebagai fungsi dari jumlah yang salah satu fungsi asli diterjemahkan. Pembaca yang tertarik dapat merujuk ke URL ini untuk informasi lebih lanjut: <http://en.wikipedia.org/wiki/Convolution> .

Jadi, blok penyusun CNN yang paling penting adalah lapisan konvolusional. Neuron di lapisan konvolusional pertama tidak terhubung ke setiap piksel dalam gambar input (seperti FNN — misalnya, MLP dan DBN), tetapi hanya ke piksel di bidang reseptifnya. Lihat *Gambar 3* . Pada gilirannya, setiap neuron di lapisan konvolusional kedua hanya terhubung ke neuron yang terletak di dalam persegi panjang kecil di lapisan pertama:



Gambar 3: Setiap neuron konvolusional memproses data hanya untuk bidang reseptifnya

Dalam Bab 2 , *Pengantar Jaringan Neural Konvolusional* , kita telah melihat bahwa semua jaringan saraf multilayer (misalnya, MLP) memiliki lapisan yang terdiri dari begitu banyak neuron, dan kita harus meratakan gambar masukan ke 1D sebelum memasukkannya ke jaringan saraf. Sebaliknya, dalam CNN, setiap lapisan direpresentasikan dalam 2D, yang membuatnya lebih mudah untuk mencocokkan neuron dengan input yang sesuai.

Konsep bidang reseptif digunakan oleh CNN untuk mengeksplorasi lokalitas spasial dengan menerapkan pola koneksi lokal antara neuron dari lapisan yang berdekatan.

Arsitektur ini memungkinkan jaringan untuk berkonsentrasi pada fitur tingkat rendah di lapisan tersembunyi pertama, dan kemudian merangkainya menjadi fitur tingkat yang lebih tinggi di lapisan tersembunyi berikutnya, dan seterusnya. Struktur hierarki ini biasa terjadi pada gambar dunia nyata, yang merupakan salah satu alasan mengapa CNN bekerja dengan sangat baik untuk pengenalan gambar.

Akhirnya, ini tidak hanya membutuhkan jumlah neuron yang rendah tetapi juga mengurangi jumlah parameter yang dapat dilatih secara signifikan. Misalnya, terlepas dari ukuran gambarnya, wilayah bangunan berukuran 5×5 , masing-masing dengan bobot yang sama, hanya memerlukan 25 parameter yang dapat dipelajari. Dengan cara ini, ini menyelesaikan masalah gradien yang menghilang atau meledak dalam melatih jaringan saraf multilayer tradisional dengan banyak lapisan dengan menggunakan propagasi mundur.

Operasi penggabungan, langkah, dan bantalan

Setelah Anda memahami cara kerja lapisan konvolusional, lapisan penyatuhan cukup mudah dipahami. Lapisan penggabungan biasanya bekerja pada setiap saluran masukan secara independen, sehingga kedalaman keluaran sama dengan kedalaman masukan. Sebagai alternatif, Anda dapat menggabungkan dimensi kedalaman, seperti yang akan kita lihat selanjutnya, dalam hal ini dimensi spasial gambar (misalnya, tinggi dan lebar) tetapi tidak berubah tetapi jumlah saluran berkurang. Mari kita lihat definisi formal lapisan penggabungan dari situs web TensorFlow yang terkenal:

"Operasi penggabungan menyapu jendela persegi panjang di atas tensor masukan, menghitung operasi pengurangan untuk setiap jendela (rata-rata, maks, atau maks dengan argmax). Setiap operasi penggabungan menggunakan jendela persegi panjang berukuran yang disebut ksize yang dipisahkan oleh langkah offset. Misalnya, jika langkah semuanya satu, setiap jendela digunakan, jika langkah semuanya berpasangan, setiap jendela digunakan di setiap dimensi, dan seterusnya. "

Oleh karena itu, secara ringkas, seperti halnya lapisan konvolusional, setiap neuron dalam lapisan penyatuhan terhubung ke keluaran sejumlah neuron terbatas di lapisan sebelumnya, yang terletak di dalam bidang reseptif persegi panjang kecil. Namun, kita harus menentukan ukurannya, langkahnya, dan jenis paddingnya. Jadi secara ringkas, outputnya bisa dihitung sebagai berikut:

```
| keluaran [i] = kurangi (nilai [langkah * i: langkah * i + ksize]),
```

Di sini, indeks juga mengambil nilai-nilai bantalan menjadi pertimbangan .

Neuron penyatuhan tidak memiliki bobot. Oleh karena itu, yang dilakukannya hanyalah menggabungkan input menggunakan fungsi agregasi seperti max atau mean.

Dengan kata lain, tujuan dari penggunaan pooling adalah untuk mensubsampel gambar input untuk mengurangi beban komputasi, penggunaan memori, dan jumlah parameter. Ini membantu untuk menghindari overfitting pada tahap pelatihan. Mengurangi ukuran gambar masukan juga membuat jaringan saraf mentolerir sedikit pergeseran gambar. Semantik spasial operasi konvolusi bergantung pada skema bantalan yang dipilih.

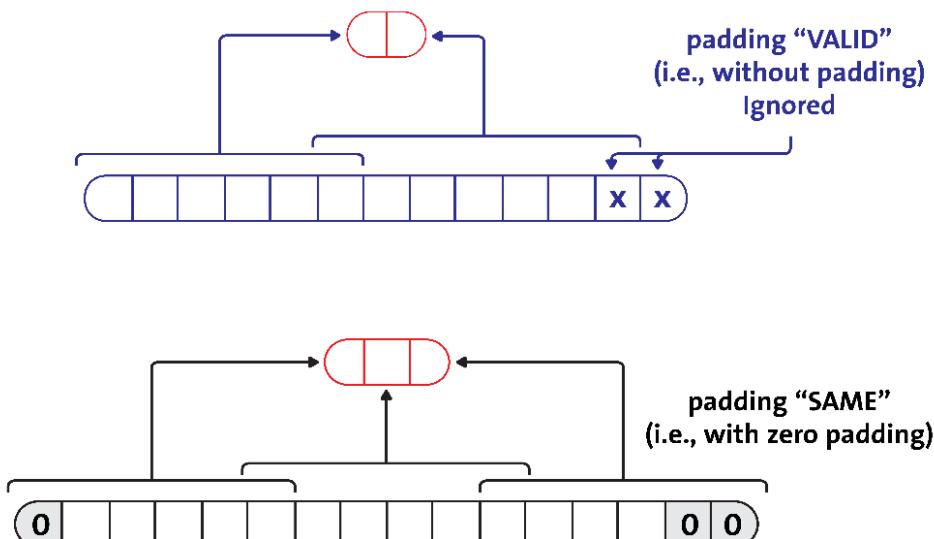
Padding adalah operasi untuk menambah ukuran data masukan. Dalam kasus data satu dimensi, Anda cukup menambahkan / menambahkan array dengan konstanta; dalam data dua dimensi, Anda mengelilingi matriks dengan konstanta ini. Dalam n-dimensi, Anda mengelilingi hypercube n-dimensi Anda dengan konstanta. Dalam kebanyakan kasus, konstanta ini adalah nol dan disebut **bantalan nol** :

- **Padding VALID** : Hanya meletakkan kolom paling kanan (atau baris paling bawah)
- **Padding yang sama** : Mencoba mengisi secara merata ke kiri dan kanan, tetapi jika jumlah kolom yang akan ditambahkan ganjil, kolom tambahan akan ditambahkan ke kanan, seperti yang terjadi pada contoh ini

Mari kita jelaskan definisi sebelumnya secara grafis, pada gambar berikut. Jika kita ingin sebuah layer memiliki tinggi dan lebar yang sama dengan layer sebelumnya, adalah umum untuk menambahkan angka nol di sekitar input, seperti yang ditunjukkan pada diagram. Ini disebut **SAMA** atau **padding nol**.

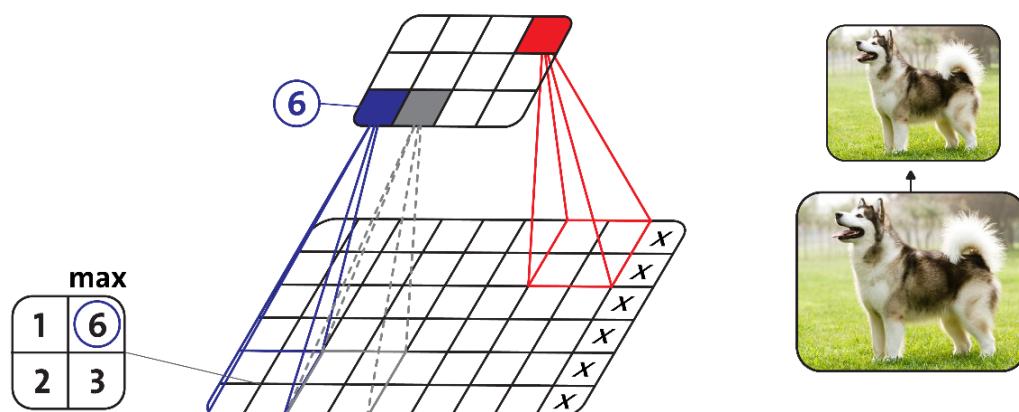
*Istilah **SAMA** berarti peta fitur keluaran memiliki dimensi spasial yang sama dengan peta fitur masukan.*

Di sisi lain, zero padding diperkenalkan untuk membuat bentuk sesuai dengan kebutuhan, secara merata di setiap sisi peta masukan. **VALID** berarti tidak ada padding dan hanya meletakkan kolom paling kanan (atau baris paling bawah):



Gambar 4: Padding SAMA versus VALID dengan CNN

Pada contoh berikut (*Gambar 5*), kami menggunakan pooling kernel 2x2 dan langkah 2 tanpa padding. Hanya nilai input **maksimum** di setiap kernel yang berhasil mencapai lapisan berikutnya karena input lain dijatuhkan (kita akan melihatnya nanti):



Gambar 5: Contoh penggunaan penggabungan maksimal, yaitu subsampling

Lapisan terhubung sepenuhnya

Di bagian atas tumpukan, lapisan terhubung penuh reguler (juga dikenal sebagai **FNN** atau **lapisan padat**) ditambahkan; itu bertindak mirip dengan MLP, yang mungkin terdiri dari beberapa lapisan yang sepenuhnya terhubung (+ ULANG). Lapisan terakhir mengeluarkan prediksi (misalnya, softmax). Contohnya adalah lapisan softmax yang mengeluarkan perkiraan probabilitas kelas untuk klasifikasi multikelas.

Lapisan yang terhubung sepenuhnya menghubungkan setiap neuron dalam satu lapisan ke setiap neuron di lapisan lain. Meskipun FNN yang terhubung sepenuhnya dapat digunakan untuk mempelajari fitur serta mengklasifikasikan data, tidak praktis untuk menerapkan arsitektur ini pada gambar.

Operasi konvolusi dan penggabungan di TensorFlow

Sekarang kita telah melihat bagaimana operasi konvolusional dan penggabungan dilakukan secara teoritis, mari kita lihat bagaimana kita dapat melakukan operasi ini secara langsung menggunakan TensorFlow. Jadi mari kita mulai.

Menerapkan operasi penggabungan di TensorFlow

Dengan menggunakan TensorFlow, lapisan subsampling biasanya dapat direpresentasikan dengan `max_pool` operasi dengan mempertahankan parameter awal lapisan tersebut. Sebab `max_pool`, ia memiliki tanda tangan berikut di TensorFlow:

```
| tf.nn.max_pool (nilai, ksize, langkah, padding, data_format, nama)
```

Sekarang mari kita pelajari cara membuat fungsi yang menggunakan tanda tangan sebelumnya dan mengembalikan tensor dengan tipe `tf.float32`, yaitu tensor keluaran gabungan maksimal:

```
| impor tensorflow sebagai tf
|
| def maxpool2d (x, k = 2):
|     return tf.nn.max_pool (x,
|                           ksize = [1, k, k, 1],
|                           langkah = [1, k, k, 1],
|                           padding = 'SAMA')
```

Di segmen kode sebelumnya, parameternya dapat dijelaskan sebagai berikut:

- `value`: Ini adalah tensor 4D `float32` elemen dan bentuk (panjang tumpukan, tinggi, lebar, dan saluran)
- `ksize`: Daftar bilangan bulat yang mewakili ukuran jendela pada setiap dimensi
- `strides`: Langkah jendela bergerak di setiap dimensi
- `data_format`: `NHWC`, `NCHW` dan `NCHW_VECT_C` didukung
- `ordering`: `NHWC` atau `NCHW`
- `padding`: `VALID` atau `SAME`

Namun, bergantung pada struktur pelapisan di CNN, ada operasi penggabungan lain yang didukung oleh TensorFlow, sebagai berikut:

- `tf.nn.avg_pool`: Ini mengembalikan tensor yang dikurangi dengan rata-rata setiap jendela
- `tf.nn.max_pool_with_argmax`: Ini mengembalikan `max_pool` tensor dan tensor dengan indeks rata `max_value`
- `tf.nn.avg_pool3d`: Ini melakukan `avg_pool` operasi dengan kubik jendela; masukan memiliki kedalaman tambahan
- `tf.nn.max_pool3d`: Ini menjalankan fungsi yang sama seperti (...) tetapi menerapkan operasi maks

Sekarang mari kita lihat contoh konkret tentang cara kerja padding di TensorFlow. Misalkan kita memiliki gambar input `x` dengan bentuk `[2, 3]` dan satu saluran. Sekarang kami ingin melihat efek keduanya `VALID` dan `SAME` paddings:

- `valid_pad`: Max pool dengan 2×2 kernel, stride 2, dan `VALID` padding
- `same_pad`: Max pool dengan 2×2 kernel, stride 2, dan `SAME` padding

Mari kita lihat bagaimana kita bisa mencapai ini dengan Python dan TensorFlow. Misalkan kita memiliki citra masukan bentuk `[2, 4]`, yaitu satu saluran:

```
| import tensorflow sebagai tf
| x = tf. konstan ([[2., 4., 6., 8.,],
| [10., 12., 14., 16.]])
```

Sekarang mari kita berikan bentuk yang diterima oleh `tf.nn.max_pool`:

```
| x = tf. bentuk kembali (x, [1, 2, 4, 1])
```

Jika kita ingin mengaplikasikan `VALID` padding dengan max pool dengan kernel 2×2 , langkah 2 :

```
| VALID = tf.nn.max_pool (x, [1, 2, 2, 1], [1, 2, 2, 1], padding = 'VALID')
```

Di sisi lain, menggunakan kumpulan maks dengan kernel 2×2 , langkah 2 dan `SAME` padding:

```
| SAMA = tf.nn.max_pool (x, [1, 2, 2, 1], [1, 2, 2, 1], padding = 'SAMA')
```

Untuk `VALID` padding, karena tidak ada padding, bentuk keluarannya adalah `[1, 1]`. Namun, untuk `SAME` padding, karena kita memadatkan gambar ke bentuk `[2, 4]` (dengan -

`inf`) dan kemudian menerapkan kumpulan maksimal, bentuk keluarannya adalah [1, 2]. Mari kita validasi:

```
| cetak (VALID.get_shape ())
| cetak (SAMA.get_shape ())
|
| >>>
| (1, 1, 2, 1)
| (1, 1, 2, 1)
```

Operasi konvolusi di TensorFlow

TensorFlow menyediakan berbagai metode konvolusi. Bentuk kanonik diterapkan oleh `conv2d` operasi. Mari kita lihat penggunaan operasi ini:

```
conv2d (
    masukan,
    filter,
    langkah,
    padding,
    use_cudnn_on_gpu = True,
    data_format = 'NHWC',
    dilations = [1, 1, 1, 1],
    name = None
)
```

Parameter yang kami gunakan adalah sebagai berikut:

- `input`: Operasi akan diterapkan ke tensor asli ini. Ini memiliki format empat dimensi yang pasti, dan urutan dimensi default ditampilkan berikutnya.
- `filter`: Ini adalah tensor yang mewakili kernel atau filter. Ini memiliki metode yang sangat generik: (`filter_height`, `filter_width`, `in_channels`, dan `out_channels`).
- `strides`: Ini adalah daftar empat `int` tipe data tensor, yang menunjukkan jendela geser untuk setiap dimensi.
- `padding`: Ini bisa menjadi `SAME` atau `VALID`. `SAME` akan mencoba untuk menghemat dimensi tensor awal, tetapi `VALID` akan memungkinkannya untuk berkembang jika ukuran keluaran dan bantalan dihitung. Kita akan melihat nanti bagaimana melakukan padding bersama dengan layer pooling.
- `use_cudnn_on_gpu`: Ini menunjukkan apakah akan menggunakan `CUDA GPU CNN` pustaka untuk mempercepat penghitungan.
- `data_format`: Ini menentukan urutan pengorganisasian data (`NHWC` atau `NCWH`).
- `dilations`: Ini menandakan daftar opsional `int`s. Ini defaultnya ke `(1, 1, 1, 1)`. Tensor 1D dengan panjang 4. Faktor dilasi untuk setiap dimensi masukan. Jika disetel ke $k > 1$, akan ada sel $k-1$ yang dilewati di antara setiap elemen filter pada dimensi itu. Urutan dimensi ditentukan oleh nilai `data_format`; lihat contoh kode sebelumnya untuk detailnya. Pelebaran dalam dimensi batch dan kedalaman harus 1.
- `name`: Nama operasi (opsional).

Berikut ini adalah contoh lapisan konvolusional. Ini menggabungkan konvolusi, menambahkan jumlah parameter bias, dan akhirnya mengembalikan fungsi aktivasi

yang telah kita pilih untuk seluruh lapisan (dalam hal ini, operasi ULT, yang sering digunakan):

```
def conv_layer (data, bobot, bias, langkah = 1):
    x = tf.nn.conv2d (x,
                     bobot,
                     langkah = [1, langkah, langkah, 1],
                     padding = 'SAMA')
    x = tf.nn.bias_add (x, bias)
    kembali tf.nn.relu (x)
```

Di sini, x adalah input ensor 4D (ukuran batch, tinggi, lebar, dan saluran). TensorFlow juga menawarkan beberapa jenis lapisan konvolusional lainnya. Sebagai contoh:

- `tf.layers.conv1d()` membuat lapisan konvolusional untuk input 1D. Ini berguna, misalnya, di NLP, di mana kalimat dapat direpresentasikan sebagai larik kata 1D, dan bidang reseptif mencakup beberapa kata yang berdekatan.
- `tf.layers.conv3d()` membuat lapisan konvolusional untuk input 3D.
- `tf.nn.atrous_conv2d()` membuat lapisan konvolusional trous (*a trous* adalah bahasa Prancis dengan lubang). Ini sama dengan menggunakan lapisan konvolusional biasa dengan filter yang dilebarkan dengan memasukkan baris dan kolom nol. Misalnya, filter 1×3 sama dengan $(1, 2, 3)$ dapat dilatasi dengan laju dilatasi 4, menghasilkan filter dilatasi $(1, 0, 0, 0, 2, 0, 0, 0, 3)$. Hal ini memungkinkan lapisan konvolusional untuk memiliki bidang reseptif yang lebih besar tanpa harga komputasi dan tidak menggunakan parameter tambahan.
- `tf.layers.conv2d_transpose()` membuat lapisan konvolusional transpose, terkadang disebut **lapisan dekonvolusional**, yang mengambil sampel gambar. Ia melakukannya dengan menyisipkan nol di antara masukan, sehingga Anda dapat menganggapnya sebagai lapisan konvolusional biasa menggunakan langkah pecahan.
- `tf.nn.depthwise_conv2d()` menciptakan lapisan konvolusional yang sangat bijaksana yang menerapkan setiap filter ke setiap saluran masukan secara terpisah. Dengan demikian, jika ada f_n filter dan f_n saluran input, maka ini akan menampilkan $f_n \times f_n$ peta fitur.
- `tf.layers.separable_conv2d()` membuat lapisan konvolusional yang dapat dipisahkan yang pertama bertindak seperti lapisan konvolusional yang bijaksana dan kemudian menerapkan lapisan konvolusional 1×1 ke peta fitur yang dihasilkan. Ini memungkinkan untuk menerapkan filter ke kumpulan saluran input yang berubah-ubah.

Melatih CNN

Di bagian sebelumnya, kita telah melihat bagaimana membuat CNN dan menerapkan operasi yang berbeda pada lapisannya yang berbeda. Sekarang ketika datang untuk melatih CNN, itu jauh lebih rumit karena membutuhkan banyak pertimbangan untuk mengontrol operasi tersebut seperti menerapkan fungsi aktivasi yang sesuai, inisialisasi bobot dan bias, dan tentu saja, menggunakan pengoptimal secara cerdas.

Ada juga beberapa pertimbangan lanjutan seperti penyetelan hyperparameter untuk dioptimalkan juga. Namun, itu akan dibahas di bagian selanjutnya. Kami pertama kali memulai diskusi kami dengan bobot dan inisialisasi bias.

Inisialisasi bobot dan bias

Salah satu teknik inisialisasi yang paling umum dalam melatih DNN adalah inisialisasi acak. Ide untuk menggunakan inisialisasi acak hanyalah mengambil sampel setiap bobot dari distribusi normal set data masukan dengan deviasi rendah. Nah, deviasi rendah memungkinkan Anda untuk membuat jaringan bias ke solusi sederhana 0.

Tapi apa artinya? Masalahnya, inisialisasi dapat diselesaikan tanpa dampak buruk dari benar-benar menginisialisasi bobot menjadi 0. Kedua, inisialisasi Xavier sering digunakan untuk melatih CNN. Ini mirip dengan inisialisasi acak tetapi seringkali ternyata bekerja jauh lebih baik. Sekarang izinkan saya menjelaskan alasannya:

- Bayangkan Anda menginisialisasi bobot jaringan secara acak tetapi ternyata mulai terlalu kecil. Kemudian sinyal menyusut saat melewati setiap lapisan sampai terlalu kecil untuk digunakan.
- Di sisi lain, jika bobot dalam jaringan mulai terlalu besar, maka sinyal tumbuh saat melewati setiap lapisan hingga terlalu besar untuk digunakan.

Hal baiknya adalah menggunakan inisialisasi Xavier memastikan bobotnya tepat, menjaga sinyal dalam kisaran nilai yang wajar melalui banyak lapisan. Singkatnya, secara otomatis dapat menentukan skala inisialisasi berdasarkan jumlah neuron input dan output.

*Pembaca yang tertarik harus merujuk ke publikasi ini untuk informasi rinci: Xavier Glorot dan Yoshua Bengio, Memahami kesulitan pelatihan FNN mendalam , Prosiding Konferensi Internasional ke-13 tentang **Kecerdasan Buatan dan Statistik (AISTATS)** 2010, Chia Laguna Resort, Sardinia, Italia. Volume 9 JMLR: W&CP.*

Terakhir, Anda dapat mengajukan pertanyaan cerdas, *Tidak bisakah saya menyingkirkan inisialisasi acak saat melatih DNN biasa (misalnya, MLP atau DBN) ?* Nah, baru-baru ini, beberapa peneliti telah membicarakan tentang inisialisasi matriks ortogonal acak yang berkinerja lebih baik daripada inisialisasi acak apa pun untuk pelatihan DNN:

- **Ketika datang untuk menginisialisasi bias** , adalah mungkin dan umum untuk menginisialisasi bias menjadi nol karena pemecahan asimetri disediakan oleh bilangan acak kecil dalam bobot. Menyetel bias ke nilai konstan kecil seperti 0,01 untuk semua bias memastikan bahwa semua unit ULT dapat menyebarkan beberapa gradien. Namun, itu tidak berkinerja baik dan juga tidak melakukan peningkatan yang konsisten. Oleh karena itu, disarankan tetap menggunakan nol.

Regularisasi

Ada beberapa cara untuk mengontrol pelatihan CNN untuk mencegah overfitting dalam fase pelatihan. Misalnya, regularisasi L2 / L1, batasan norma maks, dan putus:

- **Regularisasi L2** : Ini mungkin bentuk regularisasi yang paling umum. Ini dapat diterapkan dengan menghukum besaran kuadrat dari semua parameter secara langsung di tujuan. Misalnya, dengan menggunakan pembaruan parameter penurunan gradien, regularisasi L2 pada akhirnya berarti bahwa setiap bobot berkurang secara linier: $W' = -\lambda * W$ menuju nol.
- **Regularisasi L1** : Ini adalah bentuk lain yang relatif umum dari regularisasi, di mana untuk setiap bobot w kita menambahkan istilah $\lambda|w|$ ke tujuan. Namun, dimungkinkan juga untuk menggabungkan regularisasi L1 dengan regularisasi L2: $\lambda_1|w_1| + \lambda_2|w_2|$, yang umumnya dikenal sebagai **regularisasi jaring-elastis**.
- **Batasan norma-maks** : Bentuk lain dari regularisasi adalah dengan memaksakan batas atas absolut pada besaran vektor bobot untuk setiap neuron dan menggunakan penurunan gradien yang diproyeksikan untuk memaksakan pembatas.

Terakhir, pelolosan adalah varian lanjutan dari regularisasi, yang akan dibahas nanti di bab ini.

Fungsi aktivasi

Operasi aktivasi menyediakan berbagai jenis nonlinier untuk digunakan di jaringan saraf. Ini termasuk nonlinier halus, seperti `sigmoid`, `tanh`, `elu`, `softplus`, dan `softsign`. Di sisi lain, beberapa fungsi kontinu tetapi tidak-differentiable yang dapat digunakan adalah `relu`, `relu6`, `crelu`, dan `relu_x`. Semua operasi aktivasi menerapkan secara komponen dan menghasilkan tensor dengan bentuk yang sama seperti tensor masukan. Sekarang mari kita lihat cara menggunakan beberapa fungsi aktivasi yang umum digunakan dalam sintaks TensorFlow.

Menggunakan sigmoid

Di TensorFlow, tanda tangan `tf.sigmoid(x, name=None)` menghitung sigmoid dari segi x elemen menggunakan $y = 1 / (1 + \exp(-x))$ dan menampilkan tensor dengan jenis yang sama x . Berikut adalah deskripsi parameternya:

- x : Tensor. Ini harus menjadi salah satu jenis berikut: `float32`, `float64`, `int32`, `complex64`, `int64`, atau `qint32`.
- `name`: Nama operasi (opsional).

Menggunakan tanh

Di TensorFlow, tanda tangan `tf.tanh(x, name=None)` menghitung tangen hiperbolik dari segi `x` elemen dan menampilkan tensor dengan jenis yang sama `x`. Berikut adalah deskripsi parameternya:

- `x`: Tensor atau jarang. Ini adalah tensor dengan jenis `float`, `double`, `int32`, `complex64`, `int64`, atau `qint32`.
- `name`: Nama operasi (opsional).

Menggunakan ULT

Di TensorFlow, tanda tangan `tf.nn.relu(features, name=None)` menghitung linier yang diperbaiki menggunakan `max(features, 0)` dan menampilkan tensor yang memiliki jenis yang sama dengan fitur. Berikut adalah deskripsi parameternya:

- `features`: Tensor. Ini harus menjadi salah satu jenis berikut: `float32`, `float64`, `int32`, `int64`, `uint8`, `int16`, `int8`, `uint16`, dan `half`.
- `name`: Nama operasi (opsional).

Untuk informasi lebih lanjut tentang cara menggunakan fungsi aktivasi lainnya, lihat situs web TensorFlow. Hingga saat ini, kami memiliki pengetahuan teoritis yang minim untuk membangun jaringan CNN pertama kami untuk membuat prediksi.

Membangun, melatih, dan mengevaluasi CNN pertama kami

Pada bagian selanjutnya, kita akan melihat bagaimana mengklasifikasikan dan membedakan antara anjing dari kucing berdasarkan gambar mentahnya. Kami juga akan melihat bagaimana menerapkan model CNN pertama kami untuk menangani gambar mentah dan berwarna yang memiliki tiga saluran. Desain dan implementasi jaringan ini tidak langsung; API level rendah TensorFlow akan digunakan untuk ini. Namun, jangan khawatir; Nanti di bab ini, kita akan melihat contoh lain dari penerapan CNN menggunakan API kontrib tingkat tinggi TensorFlow. Sebelum kita memulai secara formal, penjelasan singkat tentang dataset adalah sebuah mandat.

Deskripsi kumpulan data

Untuk contoh ini, kami akan menggunakan kumpulan data anjing versus kucing dari Kaggle yang disediakan untuk masalah klasifikasi Anjing versus Kucing yang terkenal

sebagai kompetisi taman bermain dengan kernel diaktifkan. Dataset dapat diunduh dari <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>.

Map kereta berisi 25.000 gambar anjing dan kucing. Setiap gambar di folder ini memiliki label sebagai bagian dari nama file. Folder tes berisi 12.500 gambar, dinamai menurut ID numerik. Untuk setiap gambar dalam set pengujian, Anda harus memprediksi probabilitas bahwa gambar tersebut adalah seekor anjing (1 = anjing, 0 = kucing); yaitu, masalah klasifikasi biner. Untuk contoh ini, ada tiga skrip Python.

Langkah 1 - Memuat paket yang diperlukan

Di sini kami mengimpor paket dan pustaka yang diperlukan. Perhatikan bahwa bergantung pada platformnya, impor Anda mungkin berbeda:

```
waktu impor
impor matematika
impor acak
impor os
impor panda sebagai pd
impor numpy sebagai np
impor matplotlib.pyplot sebagai plt
impor tensorflow sebagai tf
impor Preprocessor
impor cv2
import LayersConstructor
dari sklearn.metrics, impor confusion_matrix
dari datetime import timedelta
dari sklearn.metrics.classification import akurasi_score
dari sklearn.metrics impor presisi_recall_fscore_support
```

Langkah 2 - Memuat gambar pelatihan / pengujian untuk menghasilkan set pelatihan / pengujian

Kami mengatur jumlah saluran warna sebagai 3 untuk gambar. Di bagian sebelumnya, kita telah melihat bahwa nilainya harus 1 untuk gambar grayscale:

```
| num_channels = 3
```

Untuk kesederhanaan, kami berasumsi bahwa dimensi gambar harus berupa persegi saja. Mari kita atur ukurannya menjadi 128:

```
| img_size = 128
```

Sekarang kita memiliki ukuran gambar (yaitu, 128) dan jumlah saluran (yaitu, 3), ukuran gambar ketika diratakan menjadi satu dimensi akan menjadi perkalian dari dimensi gambar dan jumlah saluran, sebagai berikut:

```
| img_size_flat = img_size * img_size * num_channels
```

Perhatikan bahwa, pada tahap selanjutnya, kita mungkin perlu membentuk ulang gambar untuk penyatuan maksimal dan lapisan konvolusional, jadi kita perlu membentuk kembali gambar. Untuk kasus kami, ini akan menjadi tupel dengan tinggi dan lebar gambar yang digunakan untuk membentuk kembali array:

```
| img_shape = (img_size, img_size)
```

Kita harus secara eksplisit mendefinisikan label (yaitu, kelas) karena kita hanya memiliki gambar berwarna mentah, sehingga gambar tersebut tidak memiliki label seperti kumpulan data pembelajaran mesin numerik lainnya. Mari secara eksplisit mendefinisikan info kelas sebagai berikut:

```
| kelas = ['anjing', 'kucing']
| num_classes = len(kelas)
```

Kita perlu menentukan ukuran batch yang perlu dilatih pada model CNN kita nanti:

```
| batch_size = 14
```

Perhatikan bahwa kita juga dapat menentukan bagian dari set pelatihan yang akan digunakan sebagai pemisahan validasi. Mari kita asumsikan bahwa 16% akan digunakan, untuk kesederhanaan:

```
| validation_size = 0.16
```

Satu hal penting yang harus disetel adalah berapa lama menunggu setelah kehilangan validasi berhenti meningkat sebelum menghentikan pelatihan. Kita harus menggunakan none jika kita tidak ingin menerapkan penghentian awal:

```
| early_stopping = Tidak ada
```

Sekarang, unduh kumpulan data dan Anda harus melakukan satu hal secara manual: pisahkan gambar anjing dan kucing dan letakkan di dua folder terpisah. Untuk lebih spesifik, misalkan Anda meletakkan set pelatihan Anda di bawah jalur `/home/DoG_CaT/data/train/`. Di folder kereta, buat dua folder terpisah `dogs` dan `cats` tetapi hanya tampilkan jalur ke `DoG_CaT/data/train/`. Kami juga berasumsi bahwa set pengujian kami ada di `/home/DoG_CaT/data/test/direktori`. Selain itu, Anda dapat menentukan direktori pos pemeriksaan tempat log dan file model pos pemeriksaan akan ditulis:

```
| train_path = '/ home / DoG_CaT / data / train /'
| test_path = '/ home / DoG_CaT / data / test /'
| checkpoint_dir = "model /"
```

Kemudian kita mulai membaca set pelatihan dan mempersiapkannya untuk model CNN. Untuk memproses set pengujian dan pelatihan, kami memiliki skrip lain

Preprocessor.py. Meskipun demikian, akan lebih baik untuk mempersiapkan set pengujian juga :

```
| data = Preprocessor.read_train_sets (train_path, img_size, kelas, validation_size =
| validation_size)
```

Baris kode sebelumnya membaca gambar mentah kucing dan anjing dan membuat set pelatihan. The `read_train_sets()` fungsi berjalan sebagai berikut:

```
| def read_train_sets (train_path, image_size, kelas, validation_size = 0):
|     kelas Dataset (objek):
|         lulus
|         data_sets = DataSets ()
|         gambar, label, id, cls = load_train (train_path, image_size, kelas)
|         gambar, label, id, cls = shuffle (gambar, label, id, cls)
|
|         if isinstance (validation_size, float):
|             validation_size = int (validation_size * images.shape [0])
|             validation_images = gambar [: validation_size]
|             validation_labels = label [: validation_size]
|             validation_ids = ids [: validation_size]
|             validation_cls = cls [: validation_size]
|             train_images = gambar [validation_size:]
|             train_labels = label [validation_size:]
|             train_ids = id [validation_size:]
|             train_cls = cls [validation_size:]
|             data_sets.train = DataSet (train_images, train_labels, train_id, train_cls)
|             data_sets.valid = DataSet (validation_images, validation_labels, validation_ids,
| validation_cls)
|         mengembalikan data_sets
```

Di segmen kode sebelumnya, kami telah menggunakan metode `load_train()` untuk memuat gambar yang merupakan turunan dari kelas yang disebut `DataSet`:

```
| def load_train (train_path, image_size, kelas):
|     gambar = []
|     label = []
|     ids = []
|     cls = []
|
|     print ('Membaca gambar pelatihan')
|     untuk fld di kelas:
|         index = class.index (fld)
|         print ('Memuat {} file (Indeks: {})'. format (fld, indeks))
|         path = os.path.join (train_path, fld, '* g')
|         files = glob.glob (path)
|         untuk fl dalam file:
|             gambar = cv2.imread (fl)
|             image = cv2.resize (image, (image_size, image_size), cv2.INTER_LINEAR)
|             images.append (gambar)
|             label = np.zeros (len (kelas))
|             label [indeks] = 1,0
|             labels.append (label)
|             flbase = os.path.basename (fl)
|             ids.append (flbase)
|             cls.append (fld)
|             gambar = np.array (gambar)
|             label = np.array (label)
|             ids = np.array (ids)
|             cls = np.array (cls)
|             mengembalikan gambar, label, id, cls
```

The `DataSet` kelas, yang digunakan untuk menghasilkan batch training set, adalah sebagai berikut:

```
| kelas DataSet (objek):
|
|     def next_batch (self, batch_size):
|         """ Kembalikan contoh` batch_size` berikutnya dari kumpulan data ini. """
|         start = self._index_in_epoch
```

```

    self._index_in_epoch += batch_size
    jika self._index_in_epoch > self._num_examples:
        # Epoch selesai
        self._epochs_completed += 1
        mulai = 0
        self._index_in_epoch = batch_size
        nyatakan batch_size <= self._num_examples
    end = self._index_in_epoch
    return self._images [start: end], self._labels [start: end], self._ids [start: end], self._cls
[start: end]

```

Kemudian, kami juga menyiapkan set pengujian dari gambar pengujian yang dicampur (anjing dan kucing):

```
| test_images, test_ids = Preprocessor.read_test_set (test_path, img_size)
```

Kami memiliki `read_test_set()` fungsi untuk kemudahan, sebagai berikut:

```

def read_test_set (test_path, image_size):
    gambar, ids = load_test (test_path, image_size)
    mengembalikan gambar, id

```

Sekarang, mirip dengan set pelatihan, kami memiliki fungsi khusus yang dipanggil `load_test ()` untuk memuat set pengujian, yang berjalan sebagai berikut:

```

def load_test (test_path, image_size):
    path = os.path.join (test_path, '* g')
    files = diurutkan (glob.glob (path))

    X_test = []
    X_test_id = []
    print ("Membaca gambar uji")
    untuk f1 dalam file:
        flbase = os.path.basename (f1)
        img = cv2.imread (f1)
        img = cv2.resize (img, (image_size, image_size), cv2.INTER_LINEAR)
        X_test.append (img)
        X_test_id.append (flbase)
    X_test = np.array (X_test, dtype = np.uint8)
    X_test = X_test.astype ('float32')
    X_test = X_test / 255
    mengembalikan X_test, X_test_id

```

Sudah selesai dilakukan dengan baik! Kami sekarang dapat melihat beberapa gambar yang dipilih secara acak. Untuk ini, kami memiliki fungsi helper yang dipanggil `plot_images()`; ini membuat gambar dengan sub-plot 3 x 3. Jadi, secara keseluruhan, sembilan gambar akan diplot, bersama dengan label aslinya. Ini berjalan sebagai berikut:

```

def plot_images (gambar, cls_true, cls_pred = Tidak ada):
    jika len (gambar) == 0:
        cetak ("tidak ada gambar untuk ditampilkan")
        kembali
    lain:
        random_indices = random.sample (range (len (gambar)), min (len (gambar), 9))
        gambar, cls_true = zip (* [(gambar [i], cls_true [i]) untuk i dalam indeks_akur])
        ara, sumbu = plt.subplots (3, 3)
        fig.subplots_adjust (hspace = 0.3, wspace = 0.3)
        untuk i, axes in enumerate (axes.flat):
            # Gambar plot.
            ax.imshow (gambar [i] .reshape (img_size, img_size, num_channels))
            jika cls_pred adalah None:
                xlabel = "Benar: {0}" . format (cls_true [i])
            lain:
                xlabel = "Benar: {0}, Pred: {1}" . format (cls_true [i], cls_pred [i])
            ax.set_xlabel (xlabel)
            ax.set_xticks ([])


```

```
|     ax.set_yticks ([])
|     plt.show ()
```

Mari kita dapatkan beberapa gambar acak dan labelnya dari set kereta:

```
| gambar, cls_true = data.train.images, data.train.cls
```

Terakhir, kami memplot gambar dan label menggunakan fungsi helper kami di kode sebelumnya:

```
| plot_images (gambar = gambar, cls_true = cls_true)
```

Baris kode sebelumnya menghasilkan label sebenarnya dari gambar yang dipilih secara acak:



Gambar 6: Label sebenarnya dari gambar yang dipilih secara acak

Terakhir, kami dapat mencetak statistik set data:

```
| print ("Ukuran:")
| print (" - Training-set: tt {}". format (len (data.train.labels)))
| print (" - Test-set: tt {}". format (len (test_images)))
| print (" - Validation-set: t {}". format (len (data.valid.labels)))

| >>>
| Membaca gambar pelatihan
| Memuat file anjing (Indeks: 0)
| Memuat file kucing (Indeks: 1)
| Membaca gambar uji
| Ukuran:
| - Set-pelatihan: 21000
| - Set-uji: 12500
| - Set validasi: 4000
```

Langkah 3- Mendefinisikan hyperparameter CNN

Sekarang setelah kita memiliki set pelatihan dan pengujian, sekarang waktunya untuk menentukan hyperparameter untuk model CNN sebelum kita mulai membuat. Pada lapisan konvolusional pertama dan kedua, kami menentukan lebar dan tinggi setiap filter, yaitu 3, dengan jumlah filter 32:

```
| filter_size1 = 3  
| num_filters1 = 32  
| filter_size2 = 3  
| num_filters2 = 32
```

Lapisan konvolusional ketiga memiliki dimensi yang sama tetapi dua kali filter; yaitu, 64filter:

```
| filter_size3 = 3  
| num_filters3 = 64
```

Dua lapisan terakhir adalah lapisan yang sepenuhnya terhubung, menentukan jumlah neuron:

```
| fc_size = 128
```

Sekarang mari kita buat pelatihan lebih lambat untuk pelatihan yang lebih intensif dengan menyetel nilai yang lebih rendah dari kecepatan pemelajaran, sebagai berikut:

```
| learning_rate = 1e-4
```

Langkah 4 - Membangun lapisan CNN

Setelah kita mendefinisikan hyperparameter CNN, tugas selanjutnya adalah mengimplementasikan jaringan CNN. Seperti yang bisa Anda tebak, untuk tugas kami, kami akan membangun jaringan CNN yang memiliki tiga lapisan konvolusional, lapisan yang diratakan, dan dua lapisan yang terhubung sepenuhnya (lihat `LayersConstructor.py`). Selain itu, kita perlu mendefinisikan bobot dan biasnya juga. Selain itu, kami juga akan memiliki lapisan penyatuan maksimal implisit. Pertama, mari kita tentukan bobotnya. Berikut ini, kami memiliki `new_weights()`metode yang meminta bentuk gambar dan mengembalikan bentuk normal yang terpotong:

```
| def new_weights (bentuk):  
|     return tf.Variable (tf.truncated_normal (bentuk, stddev = 0,05))
```

Kemudian kami mendefinisikan bias menggunakan `new_biases()`metode:

```
| def new_biases (panjang):  
|     return tf.Variable (tf.constant (0,05, bentuk = [panjang]))
```

Sekarang mari kita definisikan metode `new_conv_layer()`, untuk membangun lapisan konvolusional. Metode ini mengambil batch masukan, jumlah saluran masukan, ukuran filter, dan jumlah filter dan juga menggunakan penggabungan maksimal (jika

benar, kami menggunakan penggabungan maks 2×2) untuk membangun lapisan konvolusional baru. Alur kerja metode ini adalah sebagai berikut:

1. Tentukan bentuk bobot filter untuk konvolusi, yang ditentukan oleh TensorFlow API.
2. Buat bobot baru (yaitu, filter) dengan bentuk yang diberikan dan bias baru, satu untuk setiap filter.
3. Buat operasi TensorFlow untuk konvolusi yang langkahnya disetel ke 1 di semua dimensi. Langkah pertama dan terakhir harus selalu 1, karena langkah pertama untuk nomor citra dan langkah terakhir untuk saluran input. Misalnya, langkah = $(1, 2, 2, 1)$ berarti filter dipindahkan dua piksel melintasi sumbu x dan sumbu y gambar.
4. Tambahkan bias ke hasil konvolusi. Kemudian nilai bias ditambahkan ke setiap saluran filter.
5. Kemudian menggunakan penggabungan untuk menurunkan resolusi gambar. Ini adalah penggabungan maksimal 2×2 , yang berarti kami mempertimbangkan 2×2 jendela dan memilih nilai terbesar di setiap jendela. Kemudian kami memindahkan dua piksel ke jendela berikutnya.
6. ULT kemudian digunakan untuk menghitung $\max(x, 0)$ untuk setiap piksel input x . Seperti yang dinyatakan sebelumnya, ULT biasanya dijalankan sebelum penggabungan, tetapi karena `relu(max_pool(x)) == max_pool(relu(x))` kita dapat menghemat 75% dari operasi-relu dengan penyatuhan maks terlebih dahulu.
7. Terakhir, ini mengembalikan layer yang dihasilkan dan bobot filter karena kita akan memplot bobotnya nanti.

Sekarang kita mendefinisikan fungsi untuk membangun lapisan konvolusional yang akan digunakan:

```
def new_conv_layer (masukan, num_input_channels, filter_size, num_filters, use_pooling = True):  
    bentuk = [ukuran_filter, ukuran_filter, jumlah_input_saluran, jumlah_filter]  
    bobot = bobot_baru (bentuk = bentuk)  
    bias = new_biases (length = num_filters)  
    layer = tf.nn.conv2d (masukan = masukan,  
                        filter = bobot,  
                        langkah = [1, 1, 1, 1],  
                        padding = 'SAMA')  
  
    layer += bias  
    jika use_pooling:  
        layer = tf.nn.max_pool (nilai = lapisan,  
                               ksize = [1, 2, 2, 1],  
                               langkah = [1, 2, 2, 1],  
                               padding = 'SAMA')  
  
    layer = tf.nn.relu (lapisan)  
    lapisan kembali, bobot
```

Tugas selanjutnya adalah menentukan lapisan yang diratakan:

1. Dapatkan bentuk dari lapisan masukan.
2. Jumlah fiturnya adalah `img_height * img_width * num_channels`. The `get_shape()` Fungsi TensorFlow digunakan untuk menghitung ini.
3. Ini kemudian akan membentuk kembali lapisan menjadi (
 `num_images` dan `num_features`). Kami hanya menetapkan ukuran dimensi kedua menjadi `num_features` dan ukuran dimensi pertama menjadi -1, yang berarti ukuran

dalam dimensi tersebut dihitung sehingga ukuran total tensor tidak berubah dari pembentukan kembali.

4. Akhirnya, ini mengembalikan lapisan yang diratakan dan jumlah fitur.

Kode berikut berfungsi persis sama seperti yang dijelaskan sebelumnya

```
def flatten_layer(layer):
```

```
    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer_flat = tf.reshape(layer, [-1, num_features])
    kembalikan layer_flat, num_features
```

Akhirnya, kita perlu membangun lapisan yang terhubung sepenuhnya. Fungsi berikut `new_fc_layer()`, mengambil batch input, jumlah batch, dan jumlah output (yaitu, kelas yang diprediksi) dan menggunakan ULT. Ini kemudian membuat bobot dan bias berdasarkan metode yang kita tentukan sebelumnya di langkah ini. Akhirnya, ini menghitung lapisan sebagai perkalian matriks dari input dan bobot, dan kemudian menambahkan nilai bias:

```
def new_fc_layer(masukan, num_inputs, num_outputs, use_relu = True):
    bobot = bobot_baru(bentuk = [num_inputs, num_outputs])
    bias = new_biases(length = num_outputs)
    layer = tf.matmul(masukan, bobot) + bias
    jika use_relu:
        layer = tf.nn.relu(layer)
    lapisan kembali
```

Langkah 5 - Mempersiapkan grafik TensorFlow

Kami sekarang membuat placeholder untuk grafik TensorFlow:

```
x = tf.placeholder(tf.float32, bentuk = [Tidak ada, img_size_flat], name = 'x')
x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
y_true = tf.placeholder(tf.float32, shape = [Tidak ada, num_classes], name = 'y_true')
y_true_cls = tf.argmax(y_true, sumbu = 1)
```

Langkah 6 - Membuat model CNN

Sekarang kami memiliki masukan; yaitu, `x_image` siap untuk diumpulkan ke lapisan konvolusional. Kami secara resmi membuat lapisan konvolusional, diikuti oleh penggabungan maksimal:

```
layer_conv1, weights_conv1 =
    LayersConstructor.new_conv_layer(masukan = x_image,
        num_input_channels = num_channels,
        filter_size = filter_size1,
        num_filters = num_filters1,
        use_pooling = Benar)
```

Kita harus memiliki lapisan konvolusional kedua, di mana masukannya adalah lapisan konvolusional pertama `layer_conv1`, diikuti oleh penggabungan maksimal:

```
| layer_conv2, weights_conv2 =  
|     LayersConstructor.new_conv_layer (input = layer_conv1,  
|         num_input_channels = num_filters1,  
|         filter_size = filter_size2,  
|         num_filters = num_filters2,  
|         use_pooling = Benar)
```

Kami sekarang memiliki lapisan konvolusional ketiga di mana inputnya adalah output dari lapisan konvolusional kedua, yaitu, `layer_conv2` diikuti oleh penggabungan maksimal:

```
| layer_conv3, weights_conv3 =  
|     LayersConstructor.new_conv_layer (input = layer_conv2,  
|         num_input_channels = num_filters2,  
|         filter_size = filter_size3,  
|         num_filters = num_filters3,  
|         use_pooling = Benar)
```

Setelah lapisan konvolusional ketiga dibuat, kami kemudian membuat instance lapisan yang diratakan sebagai berikut:

```
| layer_flat, num_features = LayersConstructor.flatten_layer (layer_conv3)
```

Setelah kita meratakan gambar, mereka siap untuk diumpulkan ke lapisan pertama yang terhubung sepenuhnya. Kami menggunakan ULT:

```
| layer_fc1 = LayersConstructor.new_fc_layer (masukan = layer_flat,  
|         num_inputs = num_features,  
|         num_outputs = fc_size,  
|         use_relu = Benar)
```

Terakhir, kita harus memiliki lapisan terhubung penuh kedua dan terakhir di mana masukannya adalah keluaran dari lapisan terhubung penuh pertama:

```
| layer_fc2 = LayersConstructor.new_fc_layer (masukan = layer_fc1,  
|         num_inputs = fc_size,  
|         num_outputs = num_classes,  
|         use_relu = Salah)
```

Langkah 7 - Jalankan grafik TensorFlow untuk melatih model CNN

Langkah-langkah berikut digunakan untuk melakukan pelatihan. Kode-kodenya cukup jelas, seperti yang telah kita gunakan dalam contoh sebelumnya. Kami menggunakan softmax untuk memprediksi kelas dengan membandingkannya dengan kelas yang sebenarnya:

```
| y_pred = tf.nn.softmax (layer_fc2)  
| y_pred_cls = tf.argmax (y_pred, axis = 1)  
| cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2 (logits = layer_fc2, label = y_true)
```

Kami mendefinisikan `cost` fungsi dan kemudian pengoptimal (pengoptimal Adam dalam hal ini). Kemudian kami menghitung akurasi:

```
| cost_op = tf.reduce_mean (cross_entropy)
| optimizer = tf.train.AdamOptimizer (learning_rate = learning_rate). minimalkan (cost_op)
| correct_prediction = tf.equal (y_pred_cls, y_true_cls)
| akurasi = tf.reduce_mean (tf.cast (prediksi_koreksi, tf.float32))
```

Kemudian kami menginisialisasi semua operasi menggunakan `global_variables_initializer()` fungsi dari TensorFlow:

```
| init_op = tf.global_variables_initializer ()
```

Kemudian kami membuat dan menjalankan sesi TensorFlow untuk menjalankan pelatihan di seluruh tensor:

```
| sesi = tf.Session ()
| session.run (init_op)
```

Kami kemudian memasukkan data pelatihan sehingga ukuran batch menjadi 32 (lihat *Langkah 2*):

```
| train_batch_size = batch_size
```

Kami menyimpan dua daftar untuk melacak pelatihan dan akurasi validasi:

```
| acc_list = []
| val_acc_list = []
```

Kami kemudian menghitung jumlah total iterasi yang dilakukan sejauh ini dan membuat daftar kosong untuk melacak semua iterasi:

```
| total_iterations = 0
| iter_list = []
```

Kami secara resmi memulai pelatihan dengan menjalankan `optimize()` fungsi, yang membutuhkan sejumlah iterasi. Dibutuhkan dua:

- The `x_batch` contoh pelatihan yang memegang batch gambar dan
- `y_true_batch`, label sebenarnya untuk gambar tersebut

Ini kemudian mengubah bentuk setiap gambar dari (`num` contoh, baris, kolom, kedalaman) menjadi (`num` contoh, bentuk gambar yang diratakan). Setelah itu, kami memasukkan batch ke dalam `dict` variabel placeholder di grafik TensorFlow. Nanti, kami menjalankan pengoptimal pada kumpulan data pelatihan.

Kemudian, TensorFlow menetapkan variabel `feed_dict_train` ke variabel placeholder. Pengoptimal kemudian dijalankan untuk mencetak status di akhir setiap epoch. Terakhir, ini memperbarui jumlah total iterasi yang kami lakukan:

```
def optimalkan (jumlah_iterasi):
    total_iterations_global
    best_val_loss = float ("inf")
    kesabaran = 0
    untuk i dalam rentang (aksara_total, aksara_total + aksara_jumlah):
        x_batch, y_true_batch, _, cls_batch = data.train.next_batch (train_batch_size)
        x_valid_batch, y_valid_batch, _, valid_cls_batch = data.valid.next_batch (train_batch_size)
        x_batch = x_batch.reshape (train_batch_size, img_size_flat)
        x_valid_batch = x_valid_batch.reshape (ukuran_batch_train, img_size_flat)
        feed_dict_train = {x: x_batch, y_true: y_true_batch}
        feed_dict_validate = {x: x_valid_batch, y_true: y_valid_batch}
```

```

        session.run (pengoptimal, feed_dict = feed_dict_train)

    jika i% int (data.train.num_examples / batch_size) == 0:
        val_loss = session.run (biaya, feed_dict = feed_dict_validate)
        epoch = int (i / int (data.train.num_examples / batch_size))
        acc, val_acc = print_progress (epoch, feed_dict_train, feed_dict_validate, val_loss)
        acc_list.append (acc)
        val_acc_list.append (val_acc)
        iter_list.append (epoch + 1)

    jika early_stopping:
        jika val_loss <best_val_loss:
            best_val_loss = val_loss
            kesabaran = 0
        lain:
            kesabaran += 1
        jika kesabaran == early_stopping:
            istirahat
    total_iterations += num_iterations

```

Kami akan menunjukkan bagaimana pelatihan kami berjalan di bagian selanjutnya.

Langkah 8 - Evaluasi model

Kami telah berhasil menyelesaikan pelatihan. Saatnya mengevaluasi model. Sebelumnya, kita mulai mengevaluasi model, mari terapkan beberapa fungsi tambahan untuk memplot kesalahan contoh dan mencetak akurasi validasi. The `plot_example_errors()` membutuhkan dua parameter. Yang pertama adalah `cls_pred`, laris nomor kelas yang diprediksi untuk semua gambar dalam set pengujian.

Parameter kedua `correct,,` adalah `boolean` laris untuk memprediksi apakah kelas yang diprediksi sama dengan `true` kelas untuk setiap gambar dalam set pengujian. Pada awalnya, ia mendapatkan gambar dari set pengujian yang telah diklasifikasikan secara salah. Kemudian ia mendapatkan kelas yang diprediksi dan yang sebenarnya untuk gambar tersebut, dan akhirnya ia memplot sembilan gambar pertama dengan kelasnya (yaitu, label prediksi versus label sebenarnya):

```

def plot_example_errors (cls_pred, benar):
    salah = (benar == Salah)
    images = data.valid.images [salah]
    cls_pred = cls_pred [salah]
    cls_true = data.valid.cls [salah]
    plot_images (gambar = gambar [0: 9], cls_true = cls_true [0: 9], cls_pred = cls_pred [0: 9])

```

Fungsi tambahan kedua disebut `print_validation_accuracy()`; itu mencetak akurasi validasi. Ini mengalokasikan sebuah array untuk kelas yang diprediksi, yang akan dihitung dalam batch dan diisi ke dalam array ini, dan kemudian menghitung kelas yang diprediksi untuk batch tersebut:

```

def print_validation_accuracy (show_example_errors = False, show_confusion_matrix = False):
    num_test = len (data.valid.images)
    cls_pred = np.zeros (shape = num_test, dtype = np.int)
    i = 0
    sementara saya <num_test:
        # Indeks akhir untuk kelompok berikutnya dilambangkan dengan j.
        j = min (i + batch_size, num_test)
        images = data.valid.images [i: j,:]. reshape (batch_size, img_size_flat)
        label = data.valid.labels [i: j,:]
        feed_dict = {x: gambar, y_true: label}
        cls_pred [i: j] = session.run (y_pred_cls, feed_dict = feed_dict)
        i = j

```

```

cls_true = np.array (data.valid.cls)
cls_pred = np.array ([kelas [x] untuk x di cls_pred])
benar = (cls_true == cls_pred)
benar_sum = benar.sum ()
acc = float (jumlah_koreksi) / num_test

msg = "Akurasi pada Set-Pengujian: {0: .1%} ({1} / {2})"
cetak (format pesan (acc, correct_sum, num_test))

jika show_example_errors:
    print ("Contoh kesalahan:")
    plot_example_errors (cls_pred = cls_pred, benar = benar)

```

Sekarang setelah kita memiliki fungsi tambahan, kita dapat memulai pengoptimalan. Pertama-tama, mari kita ulangi penyetelan 10.000 kali dan lihat kinerjanya:

```
| optimalkan (jumlah_iterasi = 1000)
```

Setelah 10.000 iterasi, kami mengamati hasil berikut:

```

Akurasi pada Test-Set: 78,8% (3150/4000)
Presisi: 0.793378626929
Ingat: 0,7875
Skor F1: 0.786639298213

```

Ini berarti akurasi pada set tes sekitar 79%. Juga, mari kita lihat seberapa baik performa pengklasifikasi kita pada gambar contoh:



Gambar 7: Prediksi acak pada set pengujian (setelah 10.000 iterasi)

Setelah itu, kami mengulangi pengoptimalan lebih lanjut hingga 100.000 kali dan mengamati akurasi yang lebih baik:



Gambar 8: Prediksi acak pada set pengujian (setelah 100.000 iterasi)

```
>>>
Akurasi pada Test-Set: 81.1% (3244/4000)
Presisi: 0.811057239265
Ingat: 0.811
Skor F1: 0.81098298755
```

Jadi itu tidak meningkat sebanyak itu tetapi peningkatan 2% pada akurasi keseluruhan. Sekarang saatnya mengevaluasi model kita untuk satu gambar. Untuk mempermudah, kami akan mengambil dua gambar acak dari seekor anjing dan seekor kucing dan melihat kekuatan prediksi model kami:



Gambar 9: Contoh gambar untuk kucing dan anjing yang akan diklasifikasikan

Pertama, kami memuat dua gambar ini dan menyiapkan set pengujian yang sesuai, seperti yang telah kita lihat pada langkah sebelumnya dalam contoh ini:

```
test_cat = cv2.imread ('Test_image / cat.jpg')
test_cat = cv2.resize (test_cat, (img_size, img_size), cv2.INTER_LINEAR) / 255
preview_cat = plt.imshow (test_cat.reshape (img_size, img_size, num_channels))

test_dog = cv2.imread ('Test_image / dog.jpg')
test_dog = cv2.resize (test_dog, (img_size, img_size), cv2.INTER_LINEAR) / 255
preview_dog = plt.imshow (test_dog.reshape (img_size, img_size, num_channels))
```

Kemudian kita memiliki fungsi berikut untuk membuat prediksi:

```
def sample_prediction (test_im):
    feed_dict_test = {
        x: test_im.reshape (1, img_size_flat),
```

```
    y_true: np.array ([[1, 0]])
}
test_pred = session.run (y_pred_cls, feed_dict = feed_dict_test)
mengembalikan kelas [test_pred [0]]
print ("Kelas yang diprediksi untuk test_cat: {}". format (sample_prediction (test_cat)))
print ("Kelas yang diprediksi untuk test_dog: {}". format (sample_prediction (test_dog)))

>>>
Kelas yang diprediksi untuk test_cat: kucing
Kelas yang diprediksi untuk test_dog: dogs
```

Terakhir, setelah selesai, kami menutup sesi TensorFlow dengan menjalankan `close()` metode:

```
| session.close ()
```

Model optimasi kinerja

Karena CNN berbeda dari perspektif arsitektur layering, mereka memiliki persyaratan yang berbeda serta kriteria penyetelan. Bagaimana Anda mengetahui kombinasi hyperparameter yang terbaik untuk tugas Anda? Tentu saja, Anda dapat menggunakan penelusuran petak dengan validasi silang untuk menemukan hyperparameter yang tepat untuk model pembelajaran mesin linier.

Namun, untuk CNN, ada banyak hyperparameter yang harus disetel, dan karena melatih jaringan neural pada kumpulan data yang besar membutuhkan banyak waktu, Anda hanya dapat menjelajahi sebagian kecil dari ruang hyperparameter dalam waktu yang wajar. Berikut beberapa wawasan yang bisa diikuti.

Jumlah lapisan tersembunyi

Untuk banyak masalah, Anda bisa mulai dengan satu lapisan tersembunyi dan Anda akan mendapatkan hasil yang wajar. Sebenarnya telah ditunjukkan bahwa MLP dengan hanya satu lapisan tersembunyi dapat memodelkan bahkan fungsi yang paling kompleks asalkan memiliki cukup neuron. Untuk waktu yang lama, fakta ini meyakinkan para peneliti bahwa tidak perlu menyelidiki jaringan saraf yang lebih dalam. Namun, mereka mengabaikan fakta bahwa jaringan dalam memiliki efisiensi parameter yang jauh lebih tinggi daripada jaringan dangkal; mereka dapat memodelkan fungsi kompleks menggunakan neuron yang jauh lebih sedikit daripada jaringan dangkal, membuatnya jauh lebih cepat untuk dilatih.

Perlu dicatat bahwa ini mungkin tidak selalu terjadi. Namun, secara ringkas, untuk banyak masalah, Anda bisa mulai hanya dengan satu atau dua lapisan tersembunyi. Ini akan bekerja dengan baik menggunakan dua lapisan tersembunyi dengan jumlah total neuron yang sama, dalam jumlah waktu pelatihan yang kira-kira sama. Untuk masalah yang lebih kompleks, Anda dapat secara bertahap menambah jumlah lapisan tersembunyi, hingga Anda mulai menyesuaikan set pelatihan. Tugas yang sangat

kompleks, seperti klasifikasi gambar besar atau pengenalan ucapan, biasanya memerlukan jaringan dengan lusinan lapisan dan data pelatihan dalam jumlah besar.

Jumlah neuron per lapisan tersembunyi

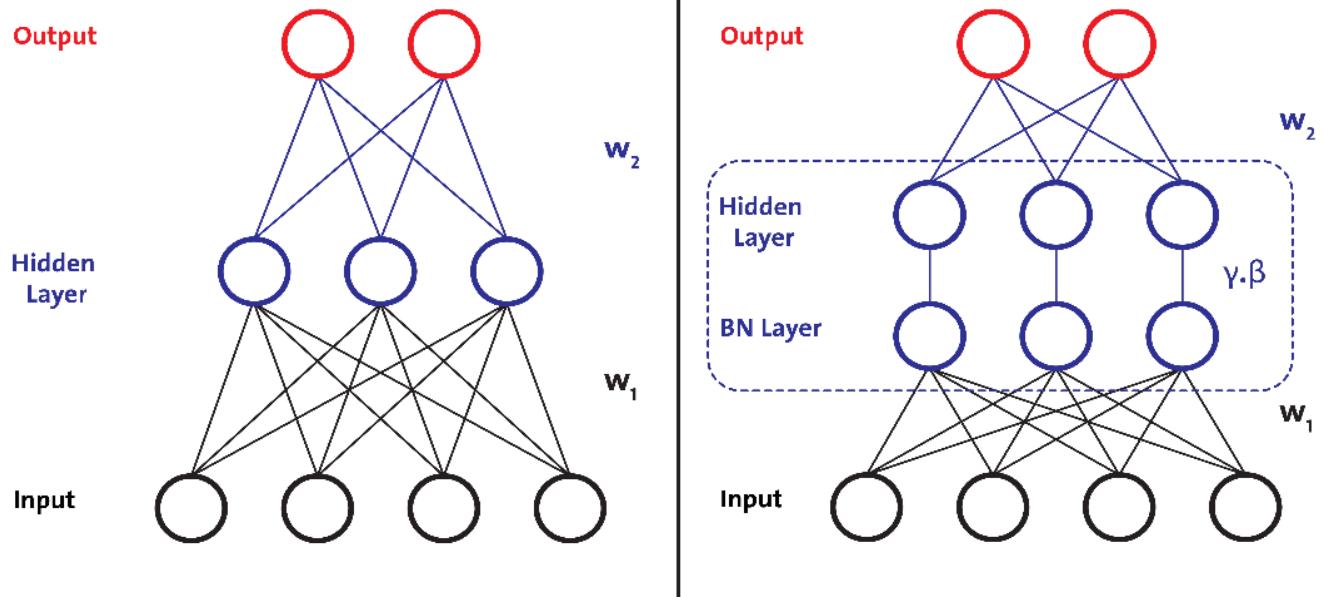
Jelas, jumlah neuron di lapisan masukan dan keluaran ditentukan oleh jenis masukan dan keluaran yang dibutuhkan tugas Anda. Misalnya, jika dataset Anda memiliki bentuk 28×28 , ia diharapkan memiliki neuron masukan dengan ukuran 784 dan neuron keluaran harus sama dengan jumlah kelas yang akan diprediksi. Sedangkan untuk lapisan tersembunyi, praktik yang umum adalah mengukurnya untuk membentuk corong, dengan semakin sedikit neuron di setiap lapisan, alasannya adalah bahwa banyak fitur tingkat rendah dapat bergabung menjadi fitur tingkat tinggi yang jauh lebih sedikit. Namun, praktik ini tidak umum sekarang, dan Anda dapat menggunakan ukuran yang sama untuk semua lapisan tersembunyi.

Jika ada empat lapisan konvolusional dengan 256 neuron, itu hanya satu hyperparameter untuk disetel, bukan satu per lapisan. Sama seperti jumlah lapisan, Anda dapat mencoba meningkatkan jumlah neuron secara bertahap hingga jaringan mulai overfitting. Pertanyaan penting lainnya adalah: kapan Anda ingin menambahkan lapisan penggabungan maksimal daripada lapisan konvolusional dengan langkah yang sama? Masalahnya adalah bahwa lapisan penyatuan maksimal tidak memiliki parameter sama sekali, sedangkan lapisan konvolusional memiliki cukup banyak.

Kadang-kadang, menambahkan lapisan normalisasi respons lokal yang membuat neuron yang paling kuat mengaktifkan, menghambat neuron di lokasi yang sama tetapi di peta fitur yang berdekatan, mendorong peta fitur yang berbeda untuk mengkhususkan dan memisahkannya, memaksa mereka untuk menjelajahi fitur yang lebih luas. Ini biasanya digunakan di lapisan bawah untuk memiliki kumpulan fitur tingkat rendah yang lebih besar yang dapat dibangun lapisan atas.

Normalisasi batch

Normalisasi batch (BN) adalah metode untuk mengurangi pergeseran kovariat internal sambil melatih DNN biasa. Ini juga dapat diterapkan ke CNN. Karena normalisasi, BN lebih jauh mencegah perubahan yang lebih kecil pada parameter untuk memperkuat dan dengan demikian memungkinkan kecepatan pembelajaran yang lebih tinggi, membuat jaringan lebih cepat:



Idenya adalah menempatkan langkah tambahan di antara lapisan, di mana keluaran dari lapisan sebelumnya dinormalisasi. Untuk lebih spesifik, dalam kasus operasi non-linier (misalnya, ULT), transformasi BN harus diterapkan pada operasi non-linier. Biasanya, keseluruhan proses memiliki alur kerja berikut:

- Mengubah jaringan menjadi jaringan BN (lihat *Gambar 1*)
- Kemudian melatih jaringan baru
- Mengubah statistik batch menjadi statistik populasi

Dengan cara ini, BN dapat sepenuhnya mengambil bagian dalam proses propagasi mundur. Seperti yang ditunjukkan pada *Gambar 1*, BN dilakukan sebelum proses jaringan lainnya di lapisan ini diterapkan. Namun, semua jenis penurunan gradien (misalnya, **penurunan gradien stokastik (SGD)** dan variannya) dapat diterapkan untuk melatih jaringan BN.

Pembaca yang tertarik dapat merujuk ke makalah asli untuk mendapatkan informasi lebih lanjut: Ioffe, Sergey, dan Christian Szegedy. Normalisasi batch: Mempercepat pelatihan jaringan dalam dengan mengurangi pergeseran kovariat internal . arXiv pracetak arXiv: 1502.03167 (2015).

Sekarang pertanyaan yang valid adalah: di mana menempatkan layer BN? Nah, untuk mengetahui jawabannya, evaluasi singkat performa layer BatchNorm pada ImageNet-2012 (<https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md>) menunjukkan benchmark berikut:

| Name | Accuracy | LogLoss | Comments |
|---------------------------|--------------|---------|-------------|
| Before | 0.474 | 2.35 | As in paper |
| Before + scale&bias layer | 0.478 | 2.33 | As in paper |
| After | 0.499 | 2.21 | |
| After + scale&bias layer | 0.493 | 2.24 | |

Dari tabel sebelumnya terlihat bahwa penempatan BN setelah non-linearitas adalah cara yang tepat. Pertanyaan kedua adalah: fungsi aktivasi apa yang harus digunakan dalam lapisan BN? Nah, dari benchmark yang sama, kita bisa melihat hasil sebagai berikut:

| Name | Accuracy | LogLoss | Comments |
|---------|--------------|-------------|----------|
| ReLU | 0.499 | 2.21 | |
| RReLU | 0.500 | 2.20 | |
| PReLU | 0.503 | 2.19 | |
| ELU | 0.498 | 2.23 | |
| Maxout | 0.487 | 2.28 | |
| Sigmoid | 0.475 | 2.35 | |
| TanH | 0.448 | 2.50 | |
| No | 0.384 | 2.96 | |

Dari tabel sebelumnya, kita dapat berasumsi bahwa menggunakan ULT atau variannya adalah ide yang lebih baik. Sekarang, pertanyaan lain adalah bagaimana menggunakannya menggunakan perpustakaan pembelajaran yang mendalam. Nah, di TensorFlow, ini adalah:

```
training = tf.placeholder(tf.bool)
x = tf.layers.dense(input_x, units = 100)
x = tf.layers.batch_normalization(x, training = training)
x = tf.nn.relu(x)
```

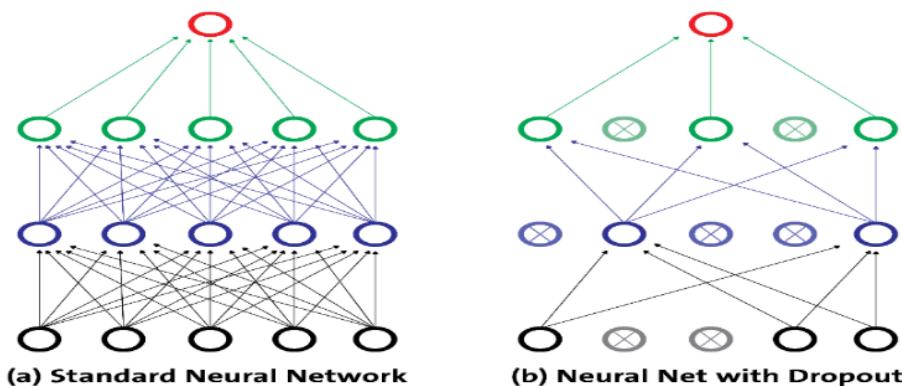
Peringatan umum: setel ini ke `True` untuk pelatihan dan `False` untuk pengujian. Namun, penambahan sebelumnya memperkenalkan operasi tambahan yang akan dijalankan pada grafik, yang memperbarui variabel mean dan variansnya sedemikian rupa sehingga tidak akan menjadi dependensi operasi pelatihan Anda. Untuk melakukannya, kita bisa menjalankan ops secara terpisah, sebagai berikut:

```
extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
sess.run ([train_op, extra_update_ops], ...)
```

Regularisasi lanjutan dan menghindari overfitting

Seperti disebutkan di bab sebelumnya, salah satu kelemahan utama yang diamati selama pelatihan jaringan neural besar adalah overfitting, yaitu menghasilkan perkiraan yang sangat baik untuk data pelatihan tetapi mengeluarkan noise untuk zona di antara titik tunggal. Ada beberapa cara untuk mengurangi atau bahkan mencegah masalah ini, seperti dropout, penghentian awal, dan pembatasan jumlah parameter.

Dalam kasus overfitting, model secara khusus disesuaikan dengan set data pelatihan, sehingga tidak akan digunakan untuk generalisasi. Oleh karena itu, meskipun performanya baik pada set pelatihan, performanya pada set data pengujian dan pengujian selanjutnya buruk karena tidak memiliki properti generalisasi:



Gambar 10: Putus sekolah versus tanpa putus sekolah

Keuntungan utama dari metode ini adalah ia menghindari menahan semua neuron dalam satu lapisan untuk mengoptimalkan bobotnya secara sinkron. Adaptasi yang dibuat dalam kelompok acak ini mencegah semua neuron menyatu ke tujuan yang sama, sehingga mengurangi korelasi bobot yang diadaptasi. Properti kedua yang ditemukan di aplikasi dropout adalah aktivasi unit tersembunyi menjadi jarang, yang juga merupakan karakteristik yang diinginkan.

Pada gambar sebelumnya, kami memiliki representasi dari jaringan neural multilayer asli yang terhubung sepenuhnya dan jaringan terkait dengan putus sekolah yang terhubung. Akibatnya, kira-kira setengah dari masukan menjadi nol (contoh ini dipilih untuk menunjukkan bahwa probabilitas tidak selalu memberikan empat nol yang diharapkan). Salah satu faktor yang dapat mengejutkan Anda adalah faktor skala yang diterapkan pada elemen yang tidak dijatuhkan.

Teknik ini digunakan untuk memelihara jaringan yang sama, dan mengembalikannya ke arsitektur asli saat pelatihan, menggunakan `dropout_keep_prob` sebagai 1. Kelemahan utama penggunaan putus sekolah adalah tidak memiliki manfaat yang sama untuk lapisan konvolusional, di mana neuron tidak sepenuhnya terhubung. Untuk mengatasi masalah ini, ada beberapa teknik yang bisa diterapkan, seperti DropConnect dan stochastic pooling:

- DropConnect mirip dengan dropout karena memperkenalkan ketersebaran dinamis di dalam model, tetapi berbeda dalam hal ketersebaran pada bobot, bukan vektor keluaran dari sebuah lapisan. Masalahnya adalah bahwa lapisan

yang sepenuhnya terhubung dengan DropConnect menjadi lapisan yang terhubung secara jarang di mana koneksi dipilih secara acak selama tahap pelatihan.

- Dalam penggabungan stokastik, operasi penggabungan deterministik konvensional diganti dengan prosedur stokastik, di mana aktivasi dalam setiap wilayah penggabungan dipilih secara acak sesuai dengan distribusi multinomial, yang diberikan oleh aktivitas dalam wilayah penggabungan. Pendekatan ini bebas hyperparameter dan dapat dikombinasikan dengan pendekatan regularisasi lainnya, seperti putus sekolah dan augmentasi data.

Penggabungan stokastik versus penggabungan maks standar:

Penggabungan stokastik setara dengan penggabungan maks standar tetapi dengan banyak salinan gambar input, masing-masing memiliki deformasi lokal kecil.

Kedua, salah satu metode paling sederhana untuk mencegah overfitting pada jaringan adalah dengan menghentikan pelatihan sebelum overfitting terjadi. Ini datang dengan kerugian bahwa proses pembelajaran terhenti. Ketiga, membatasi jumlah parameter terkadang membantu dan membantu menghindari overfitting. Dalam hal pelatihan CNN, ukuran filter juga memengaruhi jumlah parameter. Jadi, membatasi jenis parameter ini akan membatasi daya prediksi jaringan secara langsung, mengurangi kompleksitas fungsi yang dapat dijalankannya pada data, dan membatasi jumlah overfitting.

Menerapkan operasi dropout dengan TensorFlow

Jika kita menerapkan operasi pelepasan ke vektor sampel, itu akan bekerja pada transmisi pelepasan ke semua unit yang bergantung pada arsitektur. Untuk menerapkan operasi pelepasan, TensorFlow mengimplementasikan `tf.nn.dropout` metode yang bekerja sebagai berikut:

```
| tf.nn.dropout (x, keep_prob, noise_shape, seed, name)
```

Dimana `x` tensor aslinya. The `keep_prob` berarti kemungkinan menjaga neuron dan faktor dimana node yang tersisa dikalikan. Itu `noise_shape` menandakan daftar empat elemen yang menentukan apakah dimensi akan menerapkan nol secara independen atau tidak. Mari kita lihat segmen kode ini:

```
| impor tensorflow sebagai tf X = [1.5, 0.5, 0.75, 1.0, 0.75, 0.6, 0.4, 0.9]
| drop_out = tf.nn.dropout (X, 0.5)
| sess = tf.Session () dengan sess.as_default ():
|     print (drop_out.eval ())
| sess.close ()
|
| [3. 0. 1.5 0. 0. 1.2000005 0. 1.7999995]
```

Dalam contoh sebelumnya, Anda dapat melihat hasil penerapan dropout ke variabel x , dengan probabilitas 0,5 nol; dalam kasus di mana itu tidak terjadi, nilainya digandakan (dikalikan dengan $1 / 1,5$, probabilitas putus sekolah).

Pengoptimal mana yang akan digunakan?

Saat menggunakan CNN, karena salah satu fungsi tujuan adalah untuk meminimalkan biaya yang dievaluasi, kita harus menentukan pengoptimal. Menggunakan pengoptimal yang paling umum, seperti SGD, kecepatan pembelajaran harus diskalakan dengan $1 / T$ untuk mendapatkan konvergensi, di mana T adalah jumlah iterasi. Adam atau RMSProp mencoba untuk mengatasi batasan ini secara otomatis dengan menyesuaikan ukuran langkah sehingga langkah tersebut berada pada skala yang sama dengan gradien. Selain itu, pada contoh sebelumnya, kami telah menggunakan pengoptimal Adam, yang berkinerja baik dalam banyak kasus.

Namun demikian, jika Anda melatih jaringan neural tetapi menghitung gradien adalah wajib, menggunakan `RMSPropOptimizer` fungsi (yang mengimplementasikan `RMSProp` algoritme) adalah ide yang lebih baik karena ini akan menjadi cara belajar yang lebih cepat dalam pengaturan batch mini. Peneliti juga merekomendasikan penggunaan pengoptimal momentum, sambil melatih CNN atau DNN yang dalam. Secara teknis, `RMSPropOptimizer` adalah bentuk lanjutan penurunan gradien yang membagi kecepatan pembelajaran dengan rata-rata gradien kuadrat yang menurun secara eksponensial. Nilai pengaturan yang disarankan untuk parameter peluruhan adalah 0,9, sedangkan nilai default yang baik untuk kecepatan pemelajaran adalah 0,001. Misalnya, di TensorFlow, `tf.train.RMSPropOptimizer()` membantu kami menggunakan ini dengan mudah:

```
| optimizer = tf.train.RMSPropOptimizer (0,001, 0,9). minimalkan (cost_op)
```

Penyetelan memori

Pada bagian ini, kami mencoba memberikan beberapa wawasan. Kami mulai dengan masalah dan solusinya; Lapisan konvolusional memerlukan RAM dalam jumlah besar, terutama selama pelatihan, karena umpan balik dari propagasi mundur memerlukan semua nilai perantara yang dihitung selama lintasan maju. Selama inferensi (yaitu, saat membuat prediksi untuk instance baru), RAM yang ditempati oleh satu lapisan dapat dilepaskan segera setelah lapisan berikutnya dihitung, jadi Anda hanya memerlukan RAM sebanyak yang diperlukan oleh dua lapisan yang berurutan.

Namun demikian, selama pelatihan, semua yang dihitung selama forward pass perlu dipertahankan untuk reverse pass, jadi jumlah RAM yang dibutuhkan adalah (setidaknya) jumlah total RAM yang dibutuhkan oleh semua lapisan. Jika GPU Anda

kehabisan memori saat melatih CNN, berikut lima hal yang dapat Anda coba untuk menyelesaikan masalah (selain membeli GPU dengan lebih banyak RAM):

- Kurangi ukuran tumpukan mini
- Kurangi dimensi menggunakan langkah yang lebih besar dalam satu atau lebih lapisan
- Hapus satu atau lebih lapisan
- Gunakan pelampung 16-bit, bukan 32-bit
- Distribusikan CNN di beberapa perangkat (lihat selengkapnya di <https://www.tensorflow.org/deploy/distributed>)

Penempatan lapisan yang tepat

Pertanyaan penting lainnya adalah: kapan Anda ingin menambahkan lapisan penggabungan maksimal daripada lapisan konvolusional dengan langkah yang sama? Masalahnya adalah bahwa lapisan penyatuan maksimal tidak memiliki parameter sama sekali, sedangkan lapisan konvolusional memiliki cukup banyak.

Bahkan menambahkan lapisan normalisasi respons lokal terkadang membuat neuron yang paling kuat mengaktifkan, menghambat neuron di lokasi yang sama tetapi di peta fitur yang berdekatan, yang mendorong peta fitur yang berbeda untuk mengkhususkan dan memisahkannya, memaksa mereka untuk menjelajahi fitur yang lebih luas. Ini biasanya digunakan di lapisan bawah untuk memiliki kumpulan fitur tingkat rendah yang lebih besar yang dapat dibangun lapisan atas.

Membangun CNN kedua dengan menyatukan semuanya

Sekarang kita tahu bagaimana mengoptimalkan struktur pelapisan di CNN dengan menambahkan penginisialisasi dropout, BN, dan bias, seperti Xavier. Mari kita coba menerapkan ini ke CNN yang tidak terlalu rumit. Melalui contoh ini, kita akan melihat bagaimana menyelesaikan masalah klasifikasi kehidupan nyata. Untuk lebih spesifiknya, model CNN kita akan dapat mengklasifikasikan rambu lalu lintas dari sekumpulan gambar.

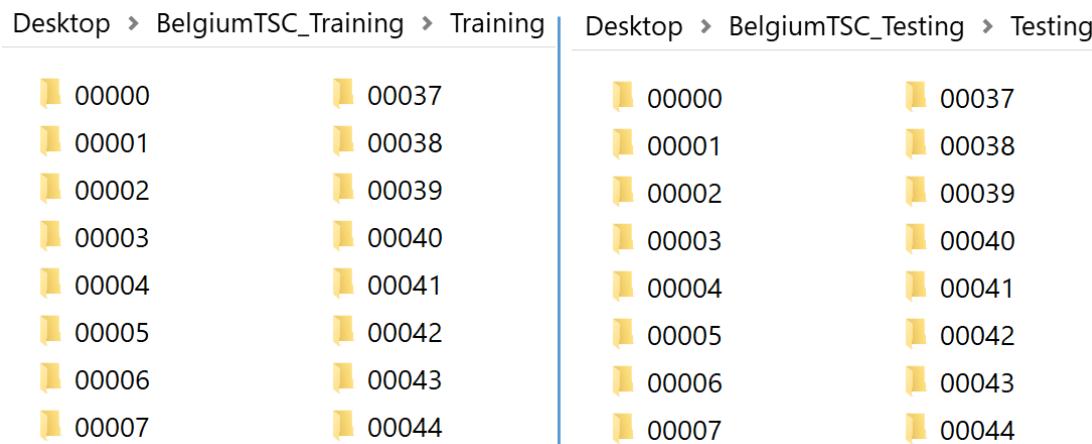
Deskripsi set data dan pemrosesan awal

Untuk ini kami akan menggunakan dataset lalu lintas Belgia (BelgiumTS untuk Klasifikasi (gambar yang dipotong)). Dataset ini dapat diunduh dari <http://btsd.ethz.ch/sh>

[areddata/](#). Berikut adalah sekilas tentang konvensi rambu lalu lintas di Belgia:

- Rambu lalu lintas Belgia biasanya dalam bahasa Belanda dan Prancis. Ini bagus untuk diketahui, tetapi untuk kumpulan data yang akan Anda gunakan, itu tidak terlalu penting!
- Ada enam kategori rambu lalu lintas di Belgia: rambu peringatan, rambu prioritas, rambu larangan, rambu wajib, rambu yang terkait dengan parkir dan berdiri diam di jalan dan, terakhir, rambu penunjuk.

Setelah kita mengunduh dataset tersebut, kita akan melihat struktur direktori berikut (kiri pelatihan, tes kanan):



Gambar dalam .ppm format; jika tidak, kita bisa menggunakan pemuat gambar bawaan TensorFlow (contoh, `tf.image.decode_png`). Namun, kita bisa menggunakan `skimage` paket Python.

Di Python 3, jalankan `$ sudo pip3 install scikit-image` untuk `skimage` menginstal dan menggunakan paket ini. Jadi mari kita mulai dengan menunjukkan jalur direktori sebagai berikut:

```
| Train_IMAGE_DIR = "<path> / BelgiumTSC_Training /"
| Test_IMAGE_DIR = "<path> / BelgiumTSC_Testing /"
```

Kemudian mari kita tulis fungsi menggunakan `skimage` pustaka untuk membaca gambar dan mengembalikan dua daftar:

- `images`: Daftar array Numpy, masing-masing mewakili gambar
- `labels`: Daftar nomor yang mewakili label gambar

```
def load_data (data_dir):
    # Semua subdirektori, di mana setiap folder mewakili
    # direktori label unik = [d untuk d di os.listdir (data_dir) jika os.path.isdir (os.path.join
    # (data_dir, d))]

    # Iterasi direktori label dan kumpulkan data dalam dua daftar, label dan gambar.
    label = []
    gambar = []
    untuk d dalam direktori: label_dir = os.path.join (data_dir, d)
    nama_file = [os.path.join (label_dir, f)
        untuk f di os.listdir (label_dir) jika f. endswith (" . ppm")]

    # Untuk setiap label, muat gambarnya dan tambahkan ke daftar gambar.
    # Dan tambahkan nomor label (yaitu nama direktori) ke daftar label.
```

```
untuk f di nama_file: images.append (skimage.data.imread (f))
labels.append (int (d))
mengembalikan gambar, label
```

Blok kode sebelumnya sangat mudah dan berisi komentar sebaris. Bagaimana kalau menampilkan statistik terkait tentang gambar? Namun, sebelum itu, mari aktifkan fungsi sebelumnya:

```
# Memuat pelatihan dan menguji set data.
train_data_dir = os.path.join (Train_IMAGE_DIR, "Training")
test_data_dir = os.path.join (Test_IMAGE_DIR, "Testing")

gambar, label = load_data (train_data_dir)
```

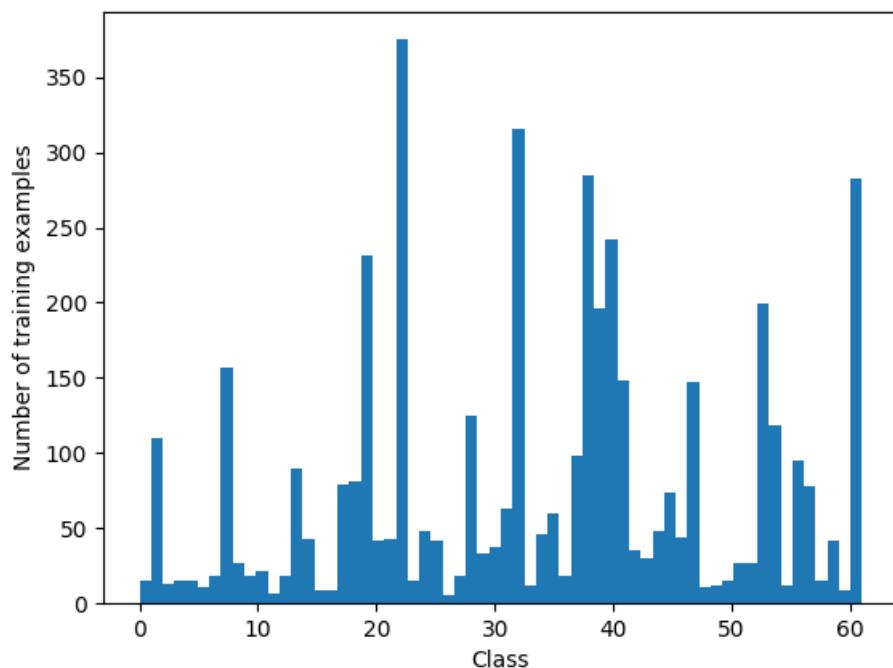
Kemudian mari kita lihat beberapa statistik:

```
print ("Kelas unik: {0} \ nTotal Gambar: {1} ". format (len (set (label)), len (gambar)))

>>>
Kelas unik: 62
Total Gambar: 4575
```

Jadi kita memiliki 62 kelas untuk diprediksi (yaitu, masalah klasifikasi gambar multikelas) dan kita memiliki banyak gambar juga yang seharusnya cukup untuk memenuhi CNN yang lebih kecil. Sekarang mari kita lihat distribusi kelas secara visual:

```
# Buat histogram dengan 62 bin dari data `label` dan tunjukkan plotnya:
plt.hist (label, 62)
plt.xlabel ('Kelas')
plt.ylabel ('Jumlah contoh pelatihan')
plt.show ()
```



Oleh karena itu, dari gambar di atas, terlihat bahwa kelas-kelas sangat timpang. Namun, untuk membuatnya lebih sederhana, kami tidak akan menangani ini, tetapi selanjutnya, akan sangat bagus untuk memeriksa beberapa file secara visual, misalnya menampilkan gambar pertama dari setiap label:

```

def display_images_and_labels (gambar, label):
    unique_labels = set (labels)
    plt.figure (figsize = (15, 15))
    i = 1
    untuk label di unique_labels:
        # Pilih gambar pertama untuk setiap label.
        image = images [labels.index (label)]
        plt.subplot (8, 8, i) #
        Petak 8 baris x 8 kolom splt.axis ('off')
        plt.title ("Label {0} ({1 }) ". format (label, labels.count (label)))
        i += 1
        _ = plt.imshow (gambar)
    plt.show ()
display_images_and_labels (gambar, label)

```



Sekarang Anda dapat melihat dari gambar sebelumnya bahwa gambar memiliki ukuran dan bentuk yang berbeda. Apalagi kita bisa melihatnya menggunakan kode Python, sebagai berikut:

```

untuk img dalam gambar [: 5]:
    print ("shape: {0}, min: {1}, maks: {2} ". format (img.shape, img.min (), img.max ()))

>>>
bentuk: (87, 84, 3), min: 12, maks: 255
bentuk: (289, 169, 3), min: 0, maks: 255
bentuk: (205, 76, 3), min: 0 , maks: 255
bentuk: (72, 71, 3), min: 14, maks: 185
bentuk: (231, 228, 3), min: 0, maks: 255

```

Oleh karena itu, kita perlu menerapkan beberapa pra-pemrosesan seperti mengubah ukuran, membentuk kembali, dan sebagainya ke setiap gambar. Misalkan setiap gambar akan memiliki ukuran 32 x 32:

```

images32 = [skimage.transform.resize (img, (32, 32), mode = 'constant')

untuk img dalam gambar] untuk img dalam gambar32 [: 5]:
    print ("shape: {0}, min: {1 }, maks: {2} ". format (img.shape, img.min (), img.max ()))

>>>
bentuk: (32, 32, 3), min: 0.06642539828431372, maks: 0.9704350490196079
bentuk: (32, 32, 3), min: 0.0, max: 1.0

```

```
bentuk: (32, 32, 3), min: 0.03172870710784261 , maks: 1.0
bentuk: (32, 32, 3), min: 0.059474571078431314, maks: 0.7036305147058846
bentuk: (32, 32, 3), min: 0.01506204044117481, maks: 1.0
```

Sekarang, semua gambar kita memiliki ukuran yang sama. Tugas selanjutnya adalah mengonversi label dan fitur gambar sebagai `numpyarray`:

```
labels_array = np.array (label)
images_array = np.array (images32)
print ("label:", labels_array.shape, "nimages:", images_array.shape)

>>>
label: (4575,)
gambar: (4575, 32, 32, 3)
```

Fantastis! Tugas selanjutnya adalah membuat CNN kedua kita, tetapi kali ini kita akan menggunakan `contrib`paket TensorFlow , yang merupakan API tingkat tinggi yang mendukung operasi pelapisan.

Membuat model CNN

Kami akan membangun jaringan yang kompleks. Namun, ia memiliki arsitektur yang lugas. Pada awalnya, kami menggunakan Xavier sebagai penginisialisasi jaringan. Setelah kami menginisialisasi bias jaringan menggunakan penginisialisasi Xavier. Lapisan masukan diikuti oleh lapisan konvolusional (lapisan konvolusional 1), yang lagi-lagi diikuti oleh lapisan BN (yaitu, lapisan BN 1). Lalu ada lapisan penyatuan dengan langkah dua dan ukuran kernel dua. Kemudian lapisan BN lainnya mengikuti lapisan konvolusional kedua. Selanjutnya, ada layer pooling kedua dengan langkah dua dan ukuran kernel dua. Nah, maka lapisan polling maks diikuti oleh lapisan perataan yang meratakan masukan dari (Tidak ada, tinggi, lebar, saluran) ke (Tidak ada, tinggi * lebar * saluran) == (Tidak ada, 3072).

Setelah perataan selesai, input dimasukkan ke dalam lapisan pertama yang terhubung sepenuhnya 1. Kemudian BN ketiga diterapkan sebagai fungsi normalizer. Kemudian kita akan memiliki lapisan putus sekolah sebelum kita memberi makan jaringan yang lebih ringan ke lapisan 2 yang terhubung sepenuhnya yang menghasilkan ukuran logit (Tidak ada, 62). Terlalu banyak makan? Jangan khawatir; kita akan melihatnya selangkah demi selangkah. Mari mulai pengkodean dengan membuat grafik komputasi, membuat kedua fitur, dan memberi label placeholder:

```
graph = tf.Graph ()
dengan graph.as_default ():

    # Placeholder untuk input dan label.
    images_X = tf.placeholder (tf.float32, [None, 32, 32, 3]) # setiap gambar berukuran 32x32
    labels_X = tf.placeholder (tf.int32, [None])

    # Initializer: Xavier
    biasInit = tf.contrib.layers .xavier_initializer (uniform = True, seed = None, dtype =
tf.float32)

    # Convolution layer 1: jumlah neuron 128 dan ukuran kernel 6x6.
    conv1 = tf.contrib.layers.conv2d (images_X, num_outputs = 128, kernel_size = [6, 6],
        biases_initializer = biasInit)

    # Batch normalization layer 1: dapat diterapkan sebagai
    fungsi normalizer # untuk conv2d dan fully_connected
    bn1 = tf.contrib.layers.batch_norm (conv1, center = True, scale = True, is_training = True)
```

```

# Max Pooling (down sampling) dengan langkah 2 dan ukuran kernel 2
pool1 = tf.contrib.layers.max_pool2d ( bn1, 2, 2)

# Convolution layer 2: jumlah neuron 256 dan ukuran kernel 6x6.
conv2 = tf.contrib.layers.conv2d (pool1, num_outputs = 256, kernel_size = [4, 4], stride = 2,
    biases_initializer = biasInit)

# Batch normalization layer 2:
bn2 = tf.contrib.layers.batch_norm (conv2, center = True, scale = True, is_training = True)

# Max Pooling (down-sampling) dengan langkah 2 dan ukuran kernel 2
pool2 = tf.contrib.layers.max_pool2d (bn2, 2, 2)

# Ratakan masukan dari [Tidak ada, tinggi, lebar, saluran] ke
# [Tidak ada, tinggi * lebar * saluran] == [Tidak ada, 3072]
images_flat = tf.contrib.layers.flatten (pool2)

# Lapisan terhubung sepenuhnya 1
fc1 = tf.contrib.layers.fully_connected (images_flat, 512, tf.nn.relu)

# Batch normalization layer 3
bn3 = tf.contrib.layers.batch_norm (fc1, center = True, scale = True, is_training = True)

# berlaku dropout, jika is_training adalah False, dropout tidak diterapkan
fc1 = tf.layers.dropout (bn3, rate = 0.25, training = True)

# Fully connected layer 2 yang menghasilkan ukuran logits [None, 62].
# Di sini 62 berarti jumlah kelas yang akan diprediksi.
logits = tf.contrib.layers.fully_connected (fc1, 62, tf.nn.relu)

```

Sampai saat ini, kami telah berhasil menghasilkan logits of size (None, 62). Kemudian kita perlu mengonversi logit ke label indexes (int) dengan shape (None), yang merupakan vektor 1D length == batch_size:predicted_labels = tf.argmax(logits, axis=1). Kemudian kami mendefinisikan cross-entropy sebagai loss fungsi, yang merupakan pilihan yang baik untuk klasifikasi:

```

loss_op = tf.reduce_mean (tf.nn.sparse_softmax_cross_entropy_with_logits (logits = logits, labels =
    labels_X))

```

Sekarang salah satu bagian terpenting adalah memperbarui operasi dan membuat pengoptimal (Adam dalam kasus kami):

```

update_ops = tf.get_collection (tf.GraphKeys.UPDATE_OPS)
dengan tf.control_dependencies (update_ops):
    # Buat pengoptimal, yang bertindak sebagai op.train pelatihan =
    tf.train.AdamOptimizer (learning_rate = 0.10). minimalkan (loss_op)

```

Akhirnya, kami menginisialisasi semua operasi:

```

init_op = tf.global_variables_initializer ()

```

Melatih dan mengevaluasi jaringan

Kita mulai dengan membuat sesi untuk menjalankan grafik yang kita buat. Perhatikan bahwa untuk pelatihan yang lebih cepat, kita harus menggunakan GPU. Namun, jika Anda tidak memiliki GPU, cukup setel log_device_placement=False:

```

session = tf.Session (graph = graph, config = tf.ConfigProto (log_device_placement = True))
session.run (init_op)
untuk i dalam range (300):
    _, loss_value = session.run ([train, loss_op], feed_dict = {images_X: images_array, labels_X:
        labels_array})

```

```

jika i% 10 == 0:
    print ("Loss:", loss_value)

>>>
Kerugian: 4.7910895
Kerugian: 4.3410876
Kerugian: 4.0275432
...
Kerugian: 0.523456

```

Setelah pelatihan selesai, mari kita pilih 10 gambar acak dan lihat kekuatan prediksi model kita:

```

random_indexes = random.sample (range (len (images32)), 10)
random_images = [images32 [i]
for i in random_index]
    random_labels = [label [i]
untuk i di random_index]

```

Lalu mari kita jalankan `predicted_labels` op:

```

prediksi = session.run ([predict_labels], feed_dict = {images_X: random_images}) [0]
print (random_labels)
print (prediksi)

>>>
[38, 21, 19, 39, 22, 22, 45, 18, 22, 53]
[20 21 19 51 22 22 45 53 22 53]

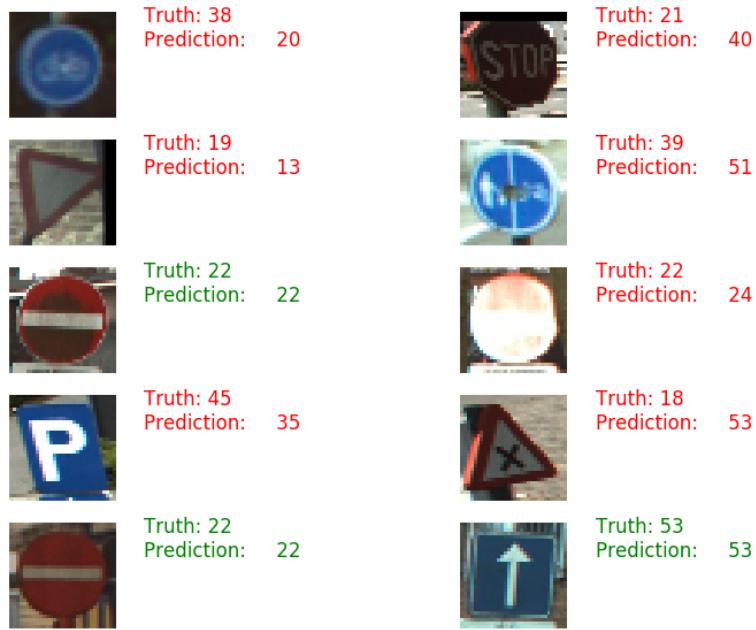
```

Jadi kita dapat melihat bahwa beberapa gambar diklasifikasikan dengan benar dan beberapa salah. Namun, inspeksi visual akan lebih membantu. Jadi, mari kita tunjukkan prediksi dan kebenaran dasarnya:

```

fig = plt.figure (figsize = (5, 5))
untuk i dalam range (len (random_images)):
    truth = random_labels [i]
    prediksi = prediksi [i]
    plt.subplot (5, 2, 1 + i)
    plt .axis ('off') color = 'green'
    if truth == prediction
    else
        'red'plt.text (40, 10, "Truth: {0} nPrediction: {1} ". format (truth, prediction), fontsize =
12,
        color = color)
    plt.imshow (random_images [i])
">>>
"

```



Akhirnya, kami dapat mengevaluasi model kami menggunakan set pengujian. Untuk melihat kekuatan prediksi, kami menghitung keakuratannya:

```
# Muat set data pengujian.
test_X, test_y = load_data (test_data_dir)

# Transformasi gambar, seperti yang kita lakukan dengan set pelatihan.
test_images32 = [skimage.transform.resize (img, (32, 32), mode = 'constant')
untuk img dalam test_X]
    display_images_and_labels (test_images32, test_y)

# Jalankan prediksi terhadap pengujian
setpredicted = session.run ([predict_labels], feed_dict = {images_X: test_images32}) [0]

# Hitung berapa banyak kecocokan
match_count = sum ([int (y == y_) untuk y, y_ in zip (test_y, prediksi)])
akurasi = match_count / len (test_y) print ("Akurasi: {:.3f}"). Format (akurasi)

>>
Akurasi: 87.583"
"
```

Tidak seburuk itu dalam hal akurasi. Selain itu, kami juga dapat menghitung metrik kinerja lainnya seperti presisi, perolehan, pengukuran f1 dan juga memvisualisasikan hasil dalam matriks kebingungan untuk menunjukkan jumlah label yang diprediksi versus jumlah label yang sebenarnya. Meski demikian, kami tetap dapat meningkatkan akurasi dengan menyetel jaringan dan hyperparameter. Tapi saya serahkan ini kepada para pembaca.

Akhirnya, kita selesai, jadi tutup sesi TensorFlow:

```
| session.close ()
```

Ringkasan

Dalam bab ini, kita membahas cara menggunakan CNN, yang merupakan jenis jaringan saraf tiruan umpan maju di mana pola konektivitas antar neuron diilhami oleh

organisasi korteks visual hewan. Kami melihat bagaimana membuat serangkaian lapisan untuk membuat CNN dan melakukan operasi yang berbeda di setiap lapisan. Lalu kami melihat bagaimana melatih CNN. Selanjutnya, kita membahas cara mengoptimalkan hyperparameter CNN dan pengoptimalan.

Terakhir, kami membuat CNN lain, tempat kami memanfaatkan semua teknik pengoptimalan. Model CNN kami tidak mencapai akurasi yang luar biasa karena kami mengulang kedua CNN beberapa kali dan bahkan tidak menerapkan teknik pencarian grid; itu berarti kami tidak mencari kombinasi terbaik dari hyperparameter. Oleh karena itu, kesimpulannya adalah menerapkan rekayasa fitur yang lebih canggih dalam gambar mentah, mengulangi pelatihan untuk lebih banyak epoch dengan hyperparameter terbaik, dan mengamati performanya.

Aku n bab berikutnya, kita akan melihat bagaimana menggunakan beberapa lebih dalam dan populer CNN arsitektur, seperti ImageNet, AlexNet, VGG, GoogLeNet, dan ResNet. Kita akan melihat bagaimana memanfaatkan model terlatih ini untuk pembelajaran transfer.

Arsitektur Model CNN Populer

Dalam bab ini, akan memperkenalkan database gambar ImageNet dan juga mencakup arsitektur dari model CNN populer berikut:

- LeNet
- AlexNet
- VGG
- GoogLeNet
- ResNet

Pengantar ImageNet

ImageNet adalah basis data lebih dari 15 juta gambar beresolusi tinggi berlabel tangan dalam sekitar 22.000 kategori. Basis data ini diatur seperti hierarki WordNet, di mana setiap konsep juga disebut sebagai **synset** (yaitu, **rangkaian sinonim**). Setiap synset adalah sebuah node dalam hierarki ImageNet. Setiap node memiliki lebih dari 500 gambar.

The **ImageNet Skala Besar Visual Recognition Tantangan (ILSVRC)** didirikan pada tahun 2010 untuk meningkatkan state-of-the-art teknologi untuk deteksi objek dan klasifikasi citra dalam skala besar:

Start exploring here

Numbers in brackets: (the number of synsets in the subtree).

- ↳ ImageNet 2011 Fall Release (32326)
 - ↳ plant, flora, plant life (4486)
 - ↳ geological formation, formation (175)
 - ↳ natural object (1112)
 - ↳ sport, athletics (176)
 - ↳ artifact, artefact (10504)
 - ↳ fungus (308)
 - ↳ person, individual, someone, somebody (10000)
 - ↳ animal, animate being, beast, brute, creature, fauna (10000)
 - ↳ Misc (20400)

Popular Synsets

Animal

fish
bird
mammal
invertebrate

Instrumen

utensil
appliance
tool
musical instrument

Plant

tree
flower
vegetable

Scene

room
geological formation

Activity

sport

Food

beverage

Material

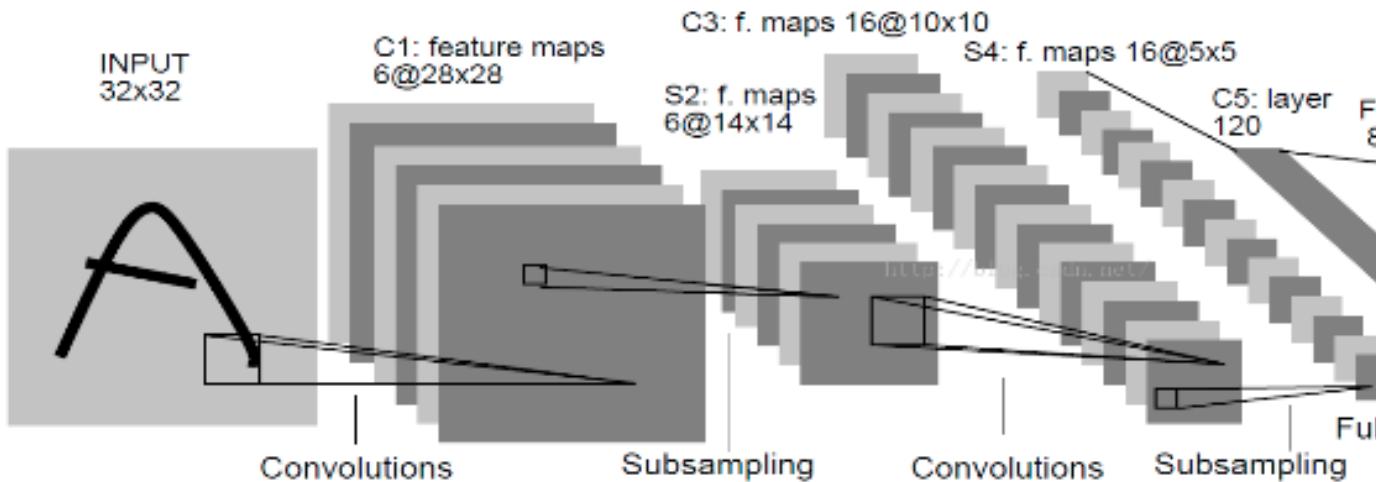
fabric

Mengikuti gambaran umum ImageNet ini, sekarang kita akan melihat berbagai arsitektur model CNN.

LeNet

Pada tahun 2010, sebuah tantangan dari ImageNet (dikenal sebagai **ILSVRC 2010**) muncul dengan arsitektur CNN, LeNet 5, yang dibangun oleh Yann Lecun. Jaringan ini mengambil gambar 32×32 sebagai masukan, yang menuju ke lapisan konvolusi (C1)

dan kemudian ke lapisan subsampling (**S2**). Saat ini, lapisan subsampling digantikan oleh lapisan penggabungan. Kemudian, ada urutan lain dari lapisan konvolusi (**C3**) diikuti oleh lapisan penggabungan (yaitu, subsampling) (**S4**). Terakhir, ada tiga lapisan yang terhubung sepenuhnya, termasuk **OUTPUT** lapisan di bagian akhir. Jaringan ini digunakan untuk pengenalan kode pos di kantor pos. Sejak itu, setiap tahun berbagai arsitektur CNN diperkenalkan dengan bantuan kompetisi ini:



LeNet 5 - Arsitektur CNN dari artikel Yann Lecun pada tahun 1998

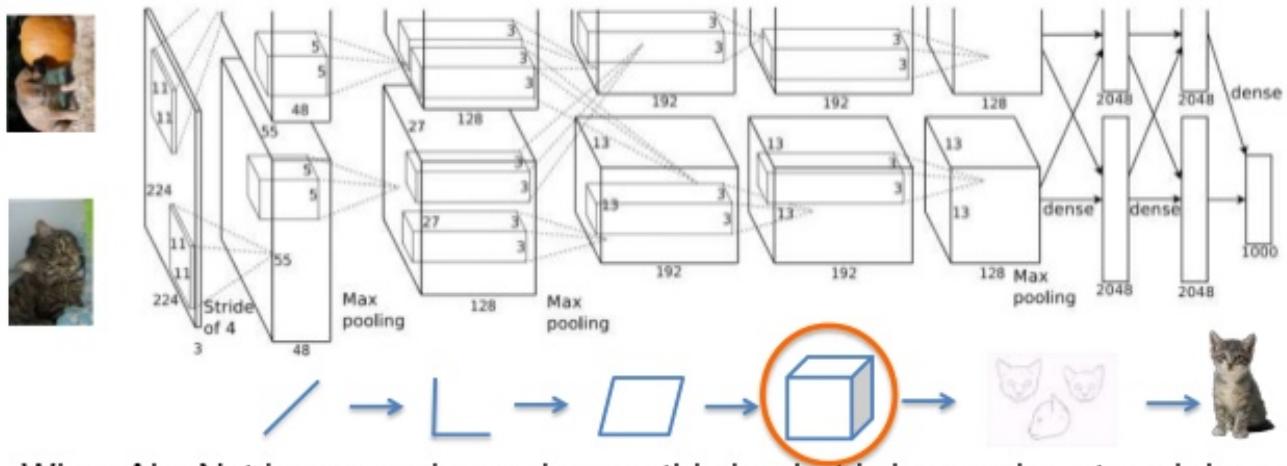
Oleh karena itu, kita dapat menyimpulkan poin-poin berikut:

- Input ke jaringan ini adalah citra grayscale 32 x 32
- Arsitektur yang diimplementasikan adalah layer CONV, diikuti oleh POOL dan layer yang terhubung sepenuhnya
- Filter CONV berukuran 5 x 5, diterapkan dengan kecepatan 1

Arsitektur AlexNet

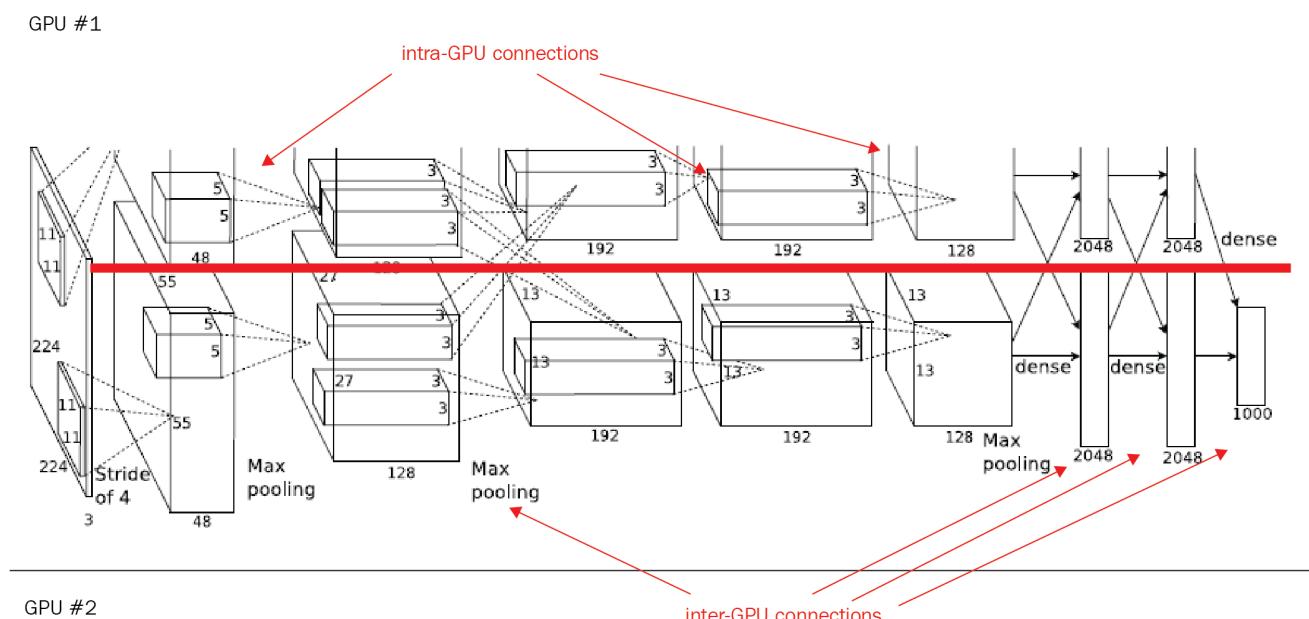
Terobosan pertama dalam arsitektur CNN datang pada tahun 2012. Arsitektur CNN pemenang penghargaan ini disebut **AlexNet**. Ini dikembangkan di Universitas Toronto oleh Alex Krizhevsky dan profesornya, Jeffry Hinton.

Dalam proses pertama, fungsi aktivasi ULT dan penurunan 0,5 digunakan di jaringan ini untuk melawan overfitting. Seperti yang bisa kita lihat pada gambar berikut, ada lapisan normalisasi yang digunakan dalam arsitektur, tetapi ini tidak digunakan lagi dalam praktik karena menggunakan augmentasi data yang berat. AlexNet masih digunakan hingga saat ini meskipun terdapat jaringan yang lebih akurat, karena strukturnya yang relatif sederhana dan kedalamannya yang kecil. Ini banyak digunakan dalam visi komputer:



When AlexNet is processing an image, this is what is happening at each layer.

AlexNet dilatih pada database ImageNet menggunakan dua GPU terpisah, kemungkinan karena keterbatasan pemrosesan dengan koneksi antar-GPU pada saat itu, seperti yang ditunjukkan pada gambar berikut:



Pengklasifikasi tanda lalu lintas menggunakan AlexNet

Dalam contoh ini, kami akan menggunakan pembelajaran transfer untuk ekstraksi fitur dan set data tanda lalu lintas Jerman untuk mengembangkan pengklasifikasi. Yang digunakan di sini adalah implementasi AlexNet oleh Michael Guerzhoy dan Davi Frossard, dan bobot AlexNet berasal dari visi Berkeley dan pusat Pembelajaran. Kode lengkap dan dataset dapat diunduh dari sini.

AlexNet mengharapkan gambar 227 x 227 x 3 piksel, sedangkan gambar rambu lalu lintas adalah 32 x 32 x 3 piksel. Untuk memasukkan gambar tanda lalu lintas ke AlexNet, kita perlu mengubah ukuran gambar ke dimensi yang diharapkan AlexNet, yaitu 227 x 227 x 3:

```
| original_image = tf.placeholder (tf.float32, ( Tidak ada , 32 , 32 , 3 ))
| resized_image = tf.image.resize_images ( original_imag , ( 227 , 227 ))
```

Kita bisa melakukannya dengan bantuan `tf.image.resize_images` metode TensorFlow. Masalah lain di sini adalah bahwa AlexNet dilatih pada kumpulan data ImageNet, yang memiliki 1.000 kelas gambar. Jadi, kami akan mengganti lapisan ini dengan lapisan klasifikasi 43-neuron. Untuk melakukan ini, cari tahu ukuran keluaran dari lapisan yang terhubung sepenuhnya terakhir; karena ini adalah lapisan yang sepenuhnya terhubung dan begitu juga bentuk 2D, elemen terakhir adalah ukuran keluaran. `fc7.get_shape().as_list()[-1]` melakukan triknya; menggabungkan ini dengan jumlah kelas untuk dataset rambu lalu lintas untuk mendapatkan bentuk akhir lapisan terhubung sepenuhnya: `shape = (fc7.get_shape().as_list()[-1], 43)`. Sisa kode lainnya hanyalah cara standar untuk menentukan lapisan yang terhubung sepenuhnya di TensorFlow. Terakhir, hitung probabilitas dengan `softmax`:

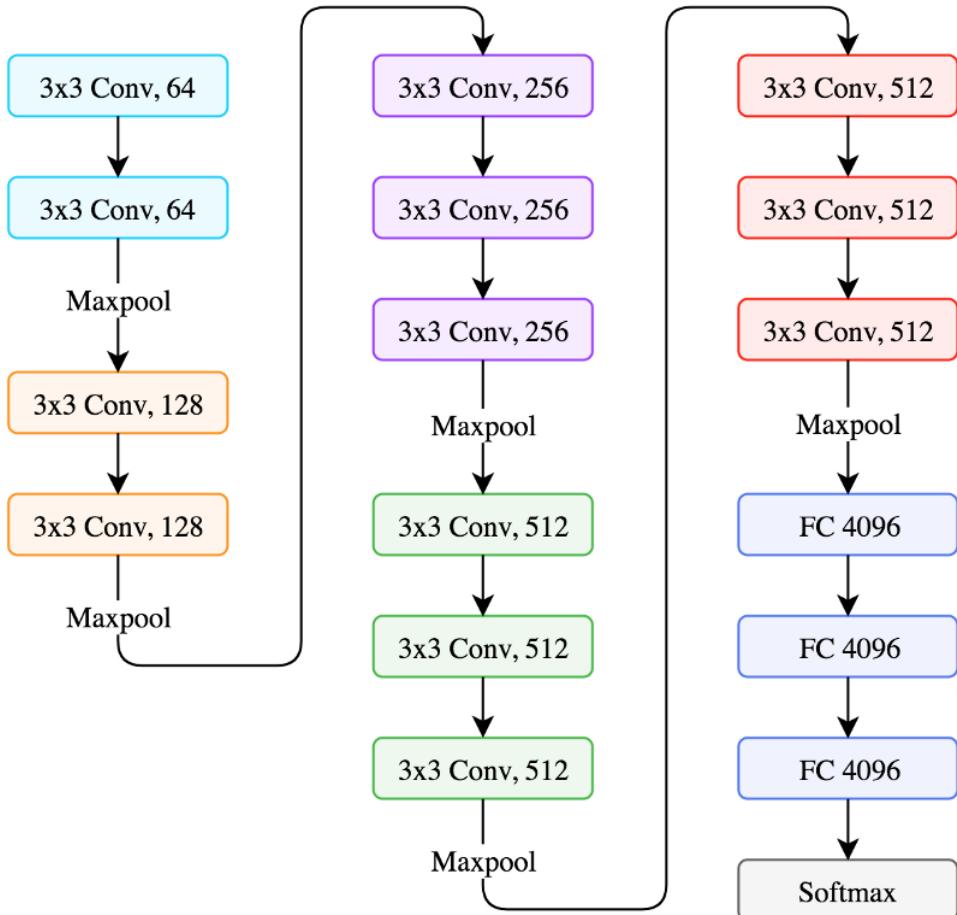
```
| #Refer kode implementasi AlexNet, mengembalikan lapisan terakhir yang terhubung sepenuhnya
| fc7 = AlexNet (diubah ukurannya, feature_extract = True )
| bentuk = (fc7.get_shape () . as_list () [- 1 ], 43)
| fc8_weight = tf.Variable (tf.truncated_normal (bentuk, stddev = 1e-2 ))
| fc8_b = tf.Variabel (tf.zeros (43))
| logits = tf.nn.xw_plus_b (fc7, fc8_weight, fc8_b)
| probs = tf.nn.softmax (logits)
```

Arsitektur VGGNet

Runner-up dalam tantangan ImageNet 2014 adalah VGGNet dari grup geometris visual di Universitas Oxford. Jaringan saraf konvolusional ini adalah arsitektur yang sederhana dan elegan dengan tingkat kesalahan 7,3%. Ini memiliki dua versi: VGG16 dan VGG19.

VGG16 adalah jaringan saraf 16 lapisan, tidak termasuk lapisan penggabungan maks dan lapisan softmax. Karenanya, ini dikenal sebagai VGG16. VGG19 terdiri dari 19 lapisan. Model terlatih tersedia di Keras untuk backend Theano dan TensorFlow.

Pertimbangan desain utama di sini adalah kedalaman. Peningkatan kedalaman jaringan dicapai dengan menambahkan lebih banyak lapisan konvolusi, dan itu dilakukan karena filter konvolusi 3 x 3 kecil di semua lapisan. Ukuran input default dari sebuah gambar untuk model ini adalah 224 x 224 x 3. Gambar tersebut melewati tumpukan lapisan konvolusi dengan langkah 1 piksel dan bantalan 1. Model ini menggunakan konvolusi 3 x 3 di seluruh jaringan. Penyatuan maksimum dilakukan di atas jendela 2 x 2 piksel dengan langkah 2, lalu tumpukan lapisan konvolusi lainnya diikuti oleh tiga lapisan yang terhubung sepenuhnya. Dua lapisan terhubung penuh pertama masing-masing memiliki 4.096 neuron, dan lapisan ketiga yang sepenuhnya terhubung bertanggung jawab untuk klasifikasi dengan 1.000 neuron. Lapisan terakhir adalah lapisan softmax. VGG16 menggunakan jendela konvolusi 3 x 3 yang jauh lebih kecil, dibandingkan dengan AlexNet 's jendela konvolusi 11 x 11 jauh lebih besar. Semua lapisan tersembunyi dibangun dengan fungsi aktivasi ULT. Arsitekturnya terlihat seperti ini:



Arsitektur jaringan VGG16

*Karena filter konvolusi 3 x 3 yang kecil, kedalaman VGGNet ditingkatkan. Jumlah parameter dalam jaringan ini kira-kira 140 juta, kebanyakan dari lapisan pertama yang terkoneksi penuh. Dalam arsitektur zaman akhir, lapisan VGGNet yang terhubung sepenuhnya diganti dengan lapisan **penyatuan rata-rata global (GAP)** untuk meminimalkan jumlah parameter.*

Pengamatan lain adalah bahwa jumlah filter meningkat seiring dengan penurunan ukuran gambar.

Contoh kode klasifikasi gambar VGG16

Modul Aplikasi Keras memiliki model jaringan neural terlatih, bersama dengan bobot terlatihnya yang dilatih di ImageNet. Model ini dapat digunakan secara langsung untuk prediksi, ekstraksi fitur, dan penyempurnaan:

```

# import model jaringan VGG16 dan pustaka lain yang diperlukan
dari keras.applications.vgg16 import VGG16
dari keras.preprocessing import image
dari keras.applications.vgg16 import preprocess_input
import numpy as np

# Instantiate VGG16 dan kembalikan instance model

```

```

vgg16 vgg16_model = VGG16 (weights = 'imagenet' , include_top = False )
#include_top: apakah akan menyertakan 3 lapisan yang terhubung sepenuhnya di bagian atas jaringan.
#Ini harus True untuk klasifikasi dan False untuk ekstraksi fitur. Mengembalikan contoh model
#weights: 'imagenet' berarti model melakukan pra-pelatihan pada data ImageNet.
model = VGG16 (bobot = 'imagenet', include_top = True)
model.summary ()

#image file name to classify
image_path = 'jumping_dolphin.jpg'
#load gambar masukan dengan utilitas keras helper dan ubah ukuran gambar.
#Ukuran masukan default untuk model ini adalah 224x224 piksel.
img = image.load_img (image_path, target_size = (224, 224))
#convert PIL (Python Image Library ??) image ke numpy array
x = image.img_to_array (img)
print (x.shape)

#image sekarang diwakili oleh bentuk array NumPy (224, 224, 3),
# tetapi kita perlu memperluas dimensinya menjadi (1, 224, 224, 3) sehingga kita dapat
# melewatkannya melalui jaringan - kita juga akan memproses gambar dengan
# mengurangi intensitas piksel RGB rata-rata dari dataset
ImageNet # Akhirnya, kita dapat memuat jaringan Keras kita dan mengklasifikasikan gambar:

x = np.expand_dims (x, axis = 0)
print (x.shape)

preprocessed_image = preprocess_input (x)

preds = model.predict (preprocessed_image)
print ('Prediksi:', decode_predictions (preds, top = 2) [0] )

```

Saat pertama kali menjalankan skrip sebelumnya, Keras akan secara otomatis mengunduh dan menyimpan bobot arsitektur ke dalam cache di direktori. Proses selanjutnya akan lebih cepat. `~/.keras/models`

Arsitektur GoogLeNet

Pada tahun 2014, ILSVRC, Google menerbitkan jaringannya sendiri yang dikenal sebagai **GoogLeNet**. Kinerjanya sedikit lebih baik dari VGGNet; Kinerja GoogLeNet adalah 6,7% dibandingkan dengan kinerja VGGNet sebesar 7,3%. Fitur utama yang menarik dari GoogLeNet adalah bahwa ia berjalan sangat cepat karena pengenalan konsep baru yang disebut **modul awal**, sehingga mengurangi jumlah parameter menjadi hanya 5 juta; itu 12 kali lebih kecil dari AlexNet. Ini memiliki penggunaan memori yang lebih rendah dan penggunaan daya yang lebih rendah juga.

Ini memiliki 22 lapisan, jadi ini adalah jaringan yang sangat dalam. Menambahkan lebih banyak lapisan akan meningkatkan jumlah parameter dan kemungkinan jaringan terlalu banyak. Akan ada lebih banyak komputasi, karena peningkatan linier pada filter menghasilkan peningkatan kuadratik dalam komputasi. Jadi, perancang menggunakan modul awal dan GAP. Lapisan yang sepenuhnya terhubung di akhir jaringan diganti dengan lapisan GAP karena lapisan yang terhubung sepenuhnya biasanya cenderung overfitting. GAP tidak memiliki parameter untuk dipelajari atau dioptimalkan.

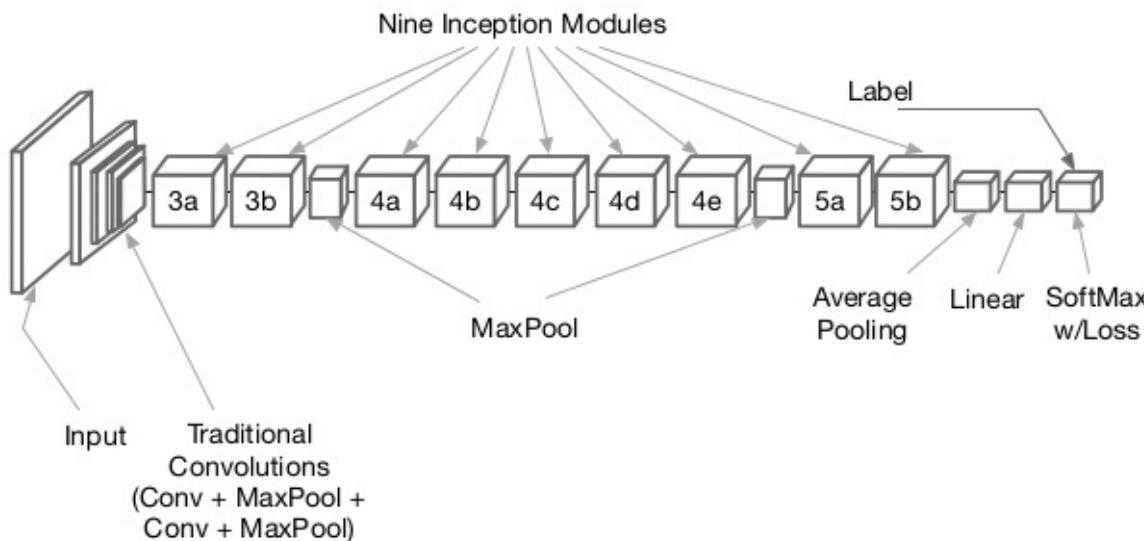
Wawasan arsitektur

Alih-alih memilih ukuran filter tertentu seperti pada arsitektur sebelumnya, desainer GoogLeNet menerapkan ketiga filter ukuran 1 x 1, 3 x 3, dan 5 x 5 pada tambalan yang

sama, dengan penggabungan dan penggabungan maks 3×3 ke dalam vektor keluaran tunggal.

Penggunaan konvolusi 1×1 mengurangi dimensi di mana pun komputasi dinaikkan dengan konvolusi 3×3 dan 5×5 yang mahal. Konvolusi 1×1 dengan fungsi aktivasi ULT digunakan sebelum konvolusi 3×3 dan 5×5 yang mahal.

Di GoogLeNet, modul awal ditumpuk satu sama lain. Penumpukan ini memungkinkan kita untuk memodifikasi setiap modul tanpa mempengaruhi lapisan selanjutnya. Misalnya, Anda dapat menambah atau mengurangi lebar lapisan apa pun:



Arsitektur GoogLeNet

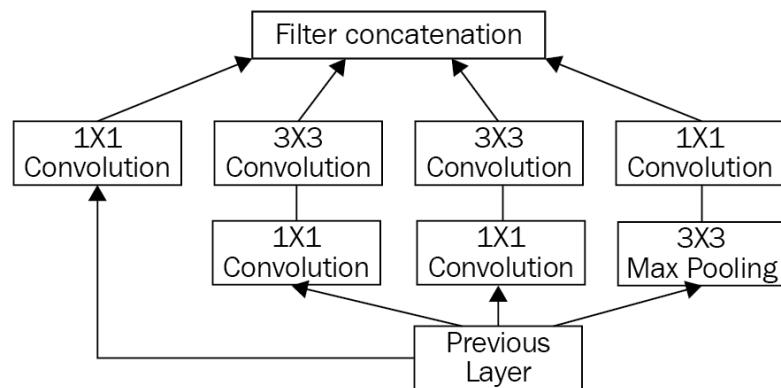
Jaringan dalam juga menderita ketakutan akan apa yang dikenal sebagai masalah **gradien menghilang** selama propagasi mundur. Ini dihindari dengan menambahkan pengklasifikasi tambahan ke lapisan perantara. Juga, selama pelatihan, kerugian menengah ditambahkan ke kerugian total dengan faktor diskon 0,3.

Karena lapisan yang sepenuhnya terhubung cenderung mengalami overfitting, maka lapisan tersebut diganti dengan lapisan GAP. Penggabungan rata-rata tidak mengecualikan penggunaan dropout, metode regularisasi untuk mengatasi overfitting di jaringan neural dalam. GoogLeNet menambahkan lapisan linier setelah 60, lapisan GAP untuk membantu orang lain menggesek pengklasifikasi mereka sendiri menggunakan teknik pembelajaran transfer.

Modul awal

Gambar berikut adalah contoh modul awal:

Inception Module Example:



Inception Module with dimension reductions: 1 X 1 Convolutions

Arsitektur ResNet

Setelah kedalaman tertentu, menambahkan lapisan tambahan ke konvnet umpan maju menghasilkan kesalahan pelatihan yang lebih tinggi dan kesalahan validasi yang lebih tinggi. Saat menambahkan lapisan, kinerja hanya meningkat hingga kedalaman tertentu, lalu menurun dengan cepat. Dalam **makalah ResNet (Residual Network)**, penulis berpendapat bahwa underfitting ini tidak mungkin terjadi karena masalah gradien menghilang, karena ini terjadi bahkan ketika menggunakan teknik normalisasi batch. Oleh karena itu, mereka menambahkan konsep baru yang disebut **blok residual**. Tim ResNet menambahkan koneksi yang dapat melewati lapisan:

ResNet menggunakan konvNet standar dan menambahkan koneksi yang melewati beberapa lapisan konvolusi sekaligus. Setiap bypass memberikan blok sisa.

Input

Convolution

Batch Norm

ReLU

Convolution

Batch Norm

Addition

ReLU

Output

Blok sisa

Di ImageNet ILSVRC 2015 Kompetisi, pemenangnya adalah ResNet dari Microsoft, dengan tingkat kesalahan 3,57%. ResNet adalah sejenis VGG dalam arti bahwa struktur yang sama diulang terus menerus untuk membuat jaringan lebih dalam. Tidak seperti VGGNet, VGGNet memiliki variasi kedalaman yang berbeda, seperti 34, 50, 101, dan 152 lapisan. Ini memiliki 152 lapisan kekalahan dibandingkan dengan AlexNet 8, 19 lapisan VGGNet, dan 22 lapisan GoogLeNet. Arsitektur ResNet adalah tumpukan blok residual. Ide utamanya adalah melewati lapisan dengan menambahkan koneksi ke jaringan saraf. Setiap blok sisa memiliki lapisan konvolusi 3 x 3. Setelah lapisan konv terakhir, lapisan GAP ditambahkan. Hanya ada satu lapisan yang terhubung sepenuhnya untuk mengklasifikasikan 1.000 kelas. Ini memiliki variasi kedalaman yang berbeda, seperti 34, 50, 101, atau 152 lapisan untuk dataset ImageNet. Untuk jaringan yang lebih dalam, katakanlah lebih dari 50 lapisan, ia menggunakan konsep fitur **bottleneck** untuk meningkatkan efisiensi. Tidak ada putus sekolah yang digunakan di jaringan ini.

Arsitektur jaringan lain yang harus diperhatikan termasuk:

- Jaringan dalam Jaringan
- Di luar ResNet
- FractalNet, jaringan neural ultra-dalam tanpa residu

Ringkasan

Dalam bab ini, kita belajar tentang arsitektur CNN yang berbeda. Model ini adalah model yang sudah ada sebelumnya dan berbeda dalam arsitektur jaringan. Masing-masing jaringan ini dirancang untuk memecahkan masalah khusus untuk arsitekturnya. Jadi, di sini kami menjelaskan perbedaan arsitektur mereka.

Kami juga memahami bagaimana arsitektur CNN kami sendiri, seperti yang dijelaskan di bab sebelumnya, berbeda dari yang lanjutan ini.

Pada bab berikutnya, kita akan mempelajari bagaimana model yang telah dilatih sebelumnya ini dapat digunakan untuk pembelajaran transfer.

Transfer Pembelajaran

Pada bab sebelumnya, kita mengetahui bahwa CNN terdiri dari beberapa lapisan. Kami juga mempelajari arsitektur CNN yang berbeda, menyetel hyperparameter yang berbeda, dan mengidentifikasi nilai untuk langkah, ukuran jendela, dan padding. Kemudian kami memilih fungsi kerugian yang benar dan mengoptimalkannya. Kami melatih arsitektur ini dengan sejumlah besar gambar. Jadi, pertanyaannya di sini adalah, bagaimana kita memanfaatkan pengetahuan ini dengan kumpulan data yang berbeda? Alih-alih membangun arsitektur CNN dan melatihnya dari awal, jaringan yang sudah dilatih sebelumnya dapat diambil dan diadaptasikan ke kumpulan data baru dan berbeda melalui teknik yang disebut **pembelajaran transfer**. Kami dapat melakukannya melalui ekstraksi fitur dan fine tuning.

Pembelajaran transfer adalah proses menyalin pengetahuan dari jaringan yang sudah terlatih ke jaringan baru untuk memecahkan masalah serupa.

Dalam bab ini, kami akan membahas topik-topik berikut:

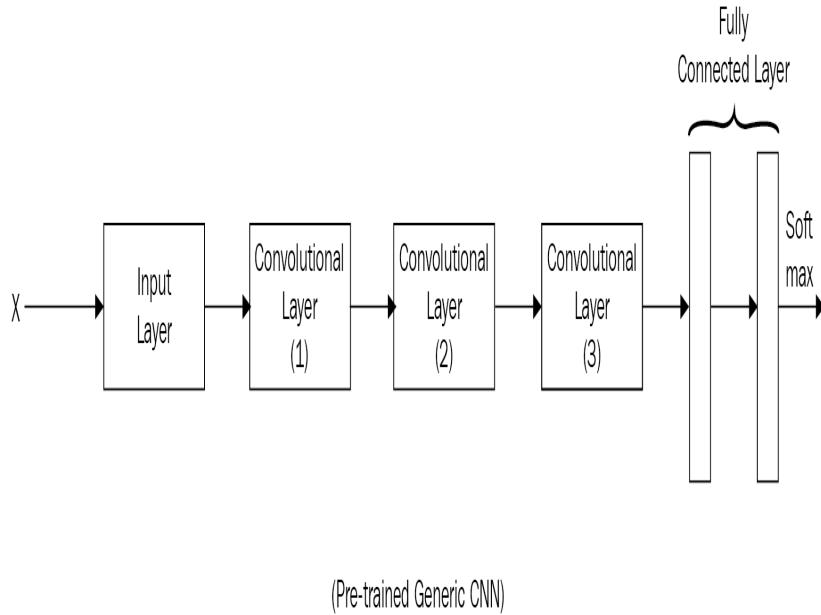
- Pendekatan ekstraksi fitur
- Contoh pembelajaran transfer
- Pembelajaran multi-tugas

Pendekatan ekstraksi fitur

Dalam pendekatan ekstraksi fitur, kami hanya melatih jaringan tingkat atas; sisa jaringan tetap diperbaiki. Pertimbangkan pendekatan ekstraksi fitur ketika kumpulan data baru relatif kecil dan mirip dengan kumpulan data asli. Dalam kasus seperti itu, fitur tingkat yang lebih tinggi yang dipelajari dari kumpulan data asli harus ditransfer dengan baik ke kumpulan data baru.

Pertimbangkan pendekatan penyesuaian jika kumpulan data baru berukuran besar dan mirip dengan kumpulan data asli. Mengubah bobot asli seharusnya aman karena jaringan kemungkinan tidak akan terlalu menyesuaikan set data baru yang besar.

Mari kita pertimbangkan jaringan neural konvolisional terlatih, seperti yang ditunjukkan pada diagram berikut. Dengan menggunakan ini kita dapat mempelajari bagaimana transfer pengetahuan dapat digunakan dalam berbagai situasi:



Kapan kita harus menggunakan pembelajaran transfer? Pembelajaran transfer dapat diterapkan dalam situasi berikut, tergantung pada:

- Ukuran kumpulan data (target) baru
- Kesamaan antara kumpulan data asli dan target

Ada empat kasus penggunaan utama:

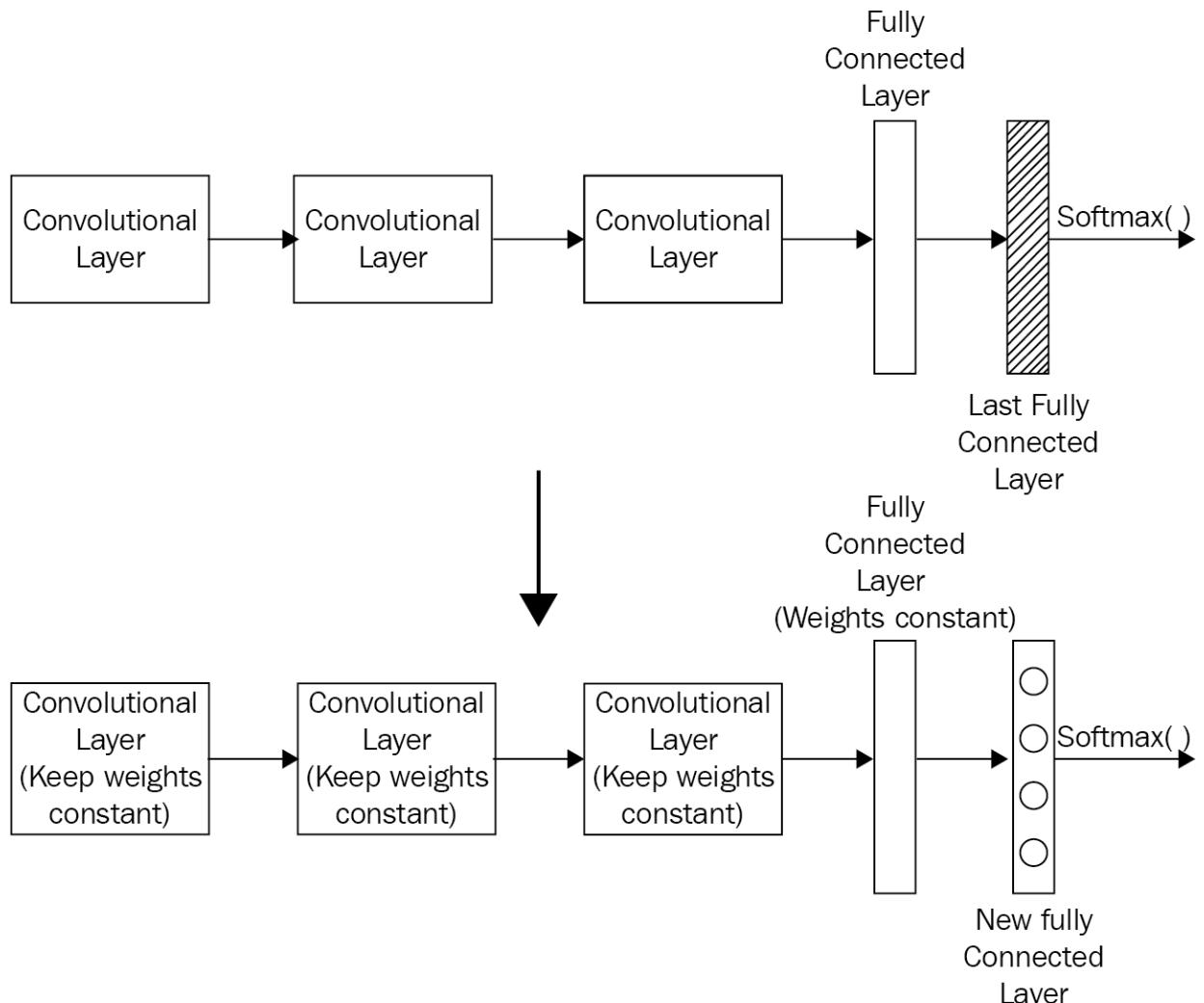
- **Kasus 1** : Dataset (target) baru berukuran kecil dan mirip dengan set data pelatihan asli
- **Kasus 2** : Dataset (target) baru berukuran kecil tetapi berbeda dari set data pelatihan asli
- **Kasus 3** : Dataset (target) baru berukuran besar dan mirip dengan set data pelatihan asli
- **Kasus 4** : Dataset (target) baru berukuran besar dan berbeda dari set data pelatihan asli

Sekarang, mari kita bahas setiap kasus secara mendetail di bagian berikut.

Dataset target kecil dan mirip dengan set data pelatihan asli

Jika kumpulan data target kecil dan mirip dengan kumpulan data asli:

- Dalam kasus ini, ganti layer terkoneksi penuh terakhir dengan layer terkoneksi penuh baru yang cocok dengan jumlah kelas dari dataset target
- Inisialisasi bobot lama dengan bobot acak
- Latih jaringan untuk memperbarui bobot lapisan baru yang terhubung sepenuhnya:



(Transfer Learning : Small data set and Similar data)

Pembelajaran transfer dapat digunakan sebagai strategi untuk menghindari overfitting, terutama bila ada set data yang kecil.

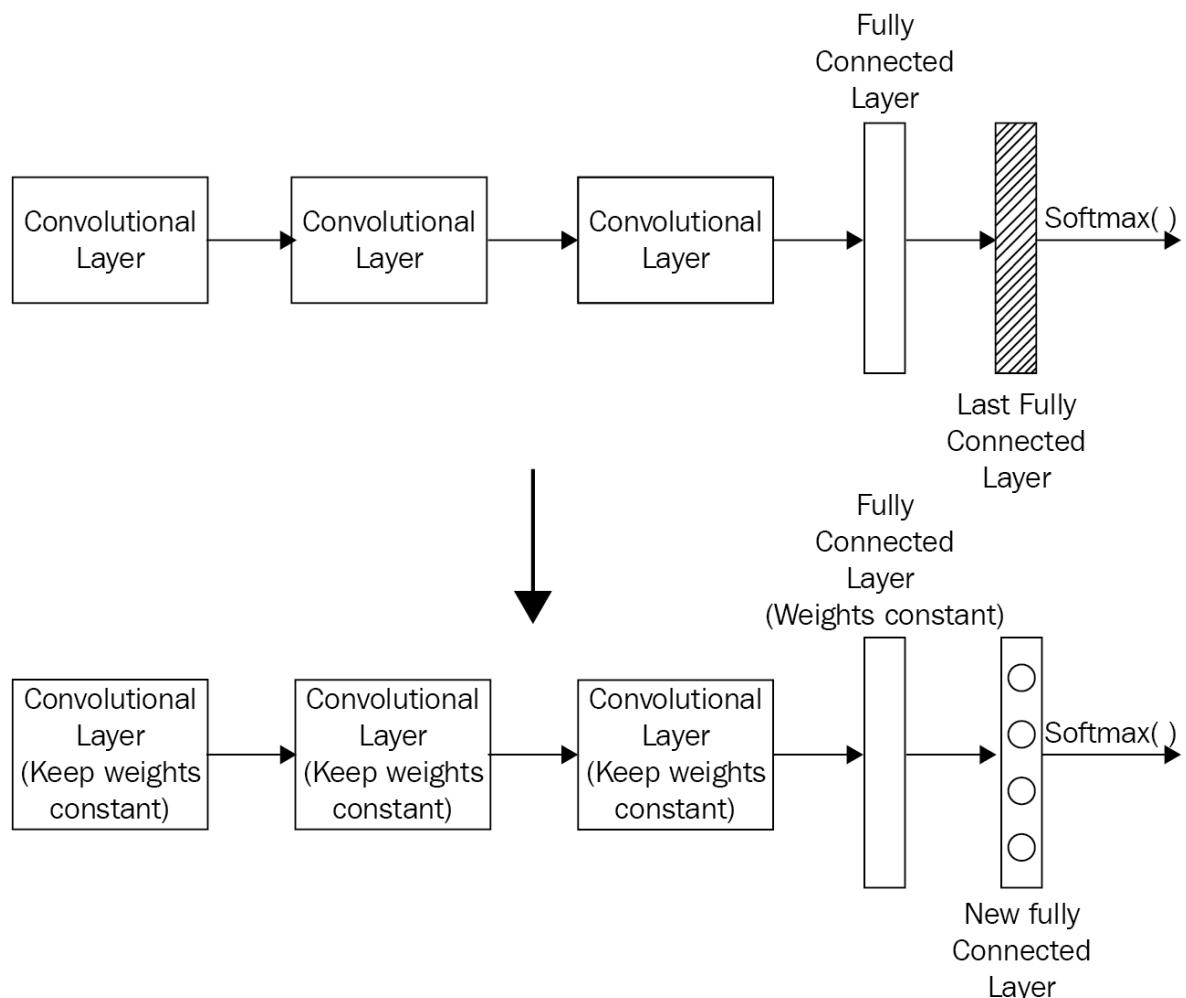
Dataset target kecil tetapi berbeda dari set data pelatihan asli

Jika kumpulan data target kecil tetapi jenisnya berbeda dengan aslinya - misalnya, kumpulan data asli adalah gambar anjing dan kumpulan data baru (target) adalah

gambar bunga - maka lakukan hal berikut:

- Potong sebagian besar lapisan awal jaringan
- Tambahkan ke lapisan terlatih yang tersisa, lapisan baru yang sepenuhnya terhubung yang cocok dengan jumlah kelas dari kumpulan data target
- Acak bobot dari lapisan baru yang terhubung sepenuhnya dan bekukan semua bobot dari jaringan terlatih
- Latih jaringan untuk memperbarui bobot dari lapisan baru yang terhubung sepenuhnya

Karena set data kecil, overfitting masih menjadi perhatian di sini. Untuk mengatasinya, kami akan menjaga bobot jaringan asli yang dilatih sebelumnya tetap sama dan hanya memperbarui bobot dari lapisan baru yang terhubung sepenuhnya:



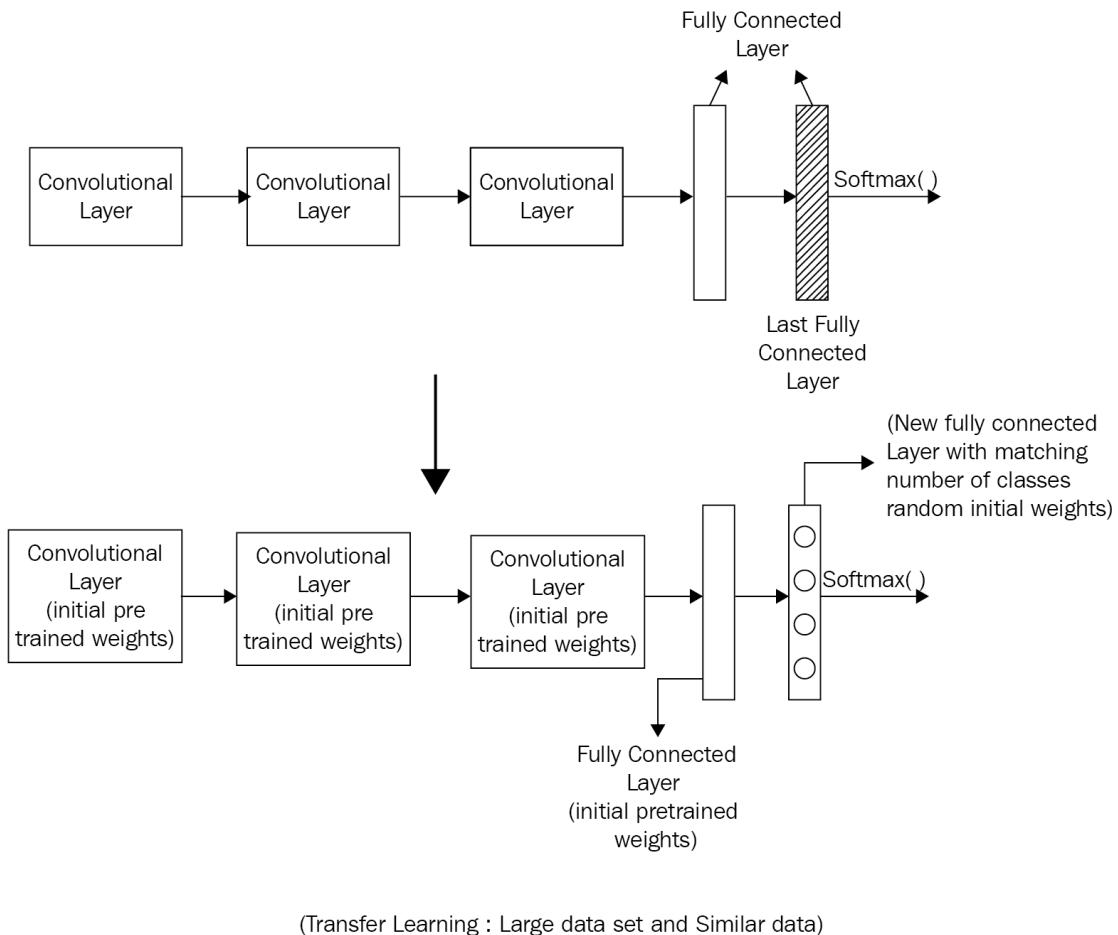
(Transfer Learning : Small data set and Similar data)

Hanya sempurnakan porsi level yang lebih tinggi dari jaringan. Ini karena lapisan awal dirancang untuk mengekstrak fitur yang lebih umum. Secara umum, lapisan pertama jaringan neural konvolusional tidak dikhawasukan untuk kumpulan data.

Set data target berukuran besar dan mirip dengan set data pelatihan asli

Di sini kami tidak memiliki masalah overfitting, karena datasetnya besar. Jadi, dalam kasus ini, kita dapat melatih ulang seluruh jaringan:

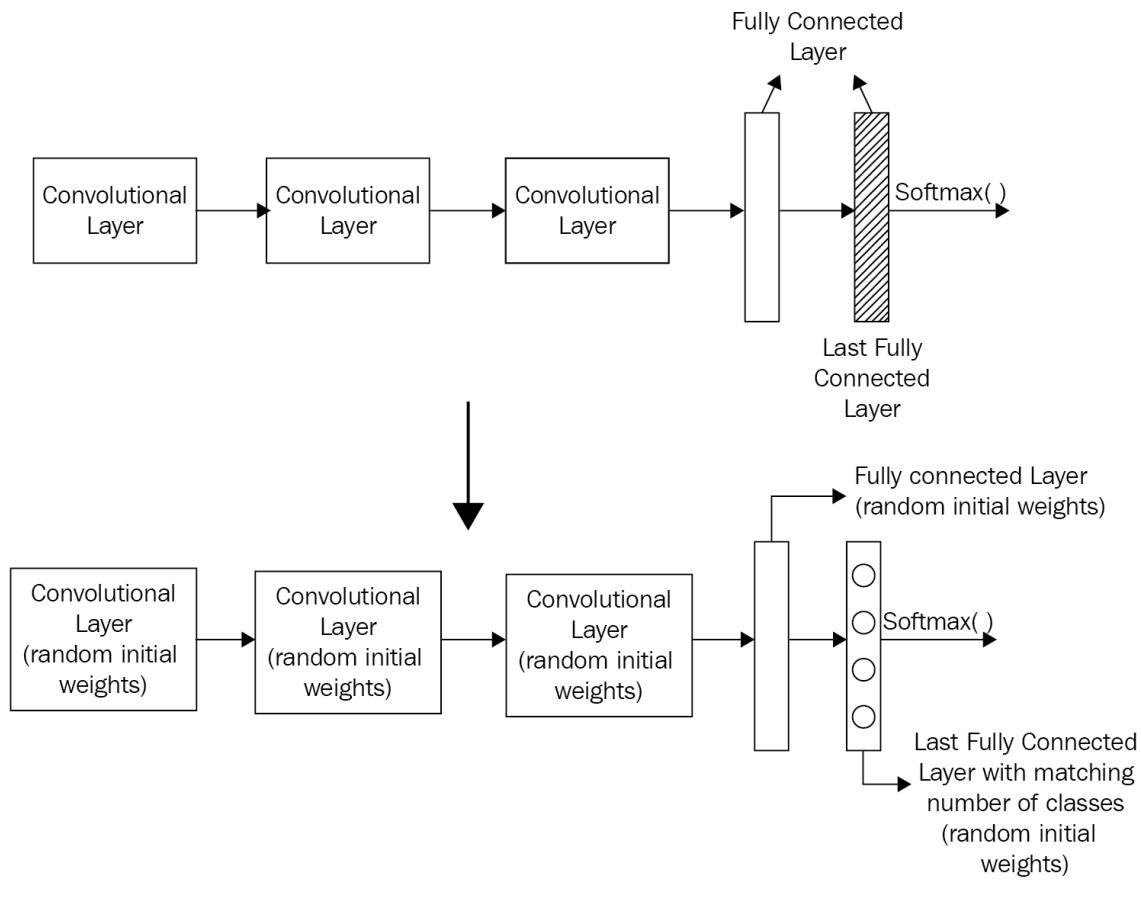
- Hapus layer terhubung penuh terakhir dan ganti dengan layer terhubung penuh yang sesuai dengan jumlah kelas dalam dataset target
- Inisialisasi bobot dari lapisan yang baru ditambahkan dan terhubung sepenuhnya ini secara acak
- Inisialisasi anak timbangan lainnya dengan anak timbangan yang telah dilatih sebelumnya
- Latih seluruh jaringan:



Set data target berukuran besar dan berbeda dari set data pelatihan asli

Jika kumpulan data target besar dan berbeda dari aslinya:

- Hapus layer terhubung penuh terakhir dan ganti dengan layer terhubung penuh yang sesuai dengan jumlah kelas dalam dataset target
- Latih seluruh jaringan dari awal dengan bobot yang diinisialisasi secara acak:



(Transfer Learning : Large data set and Different data)

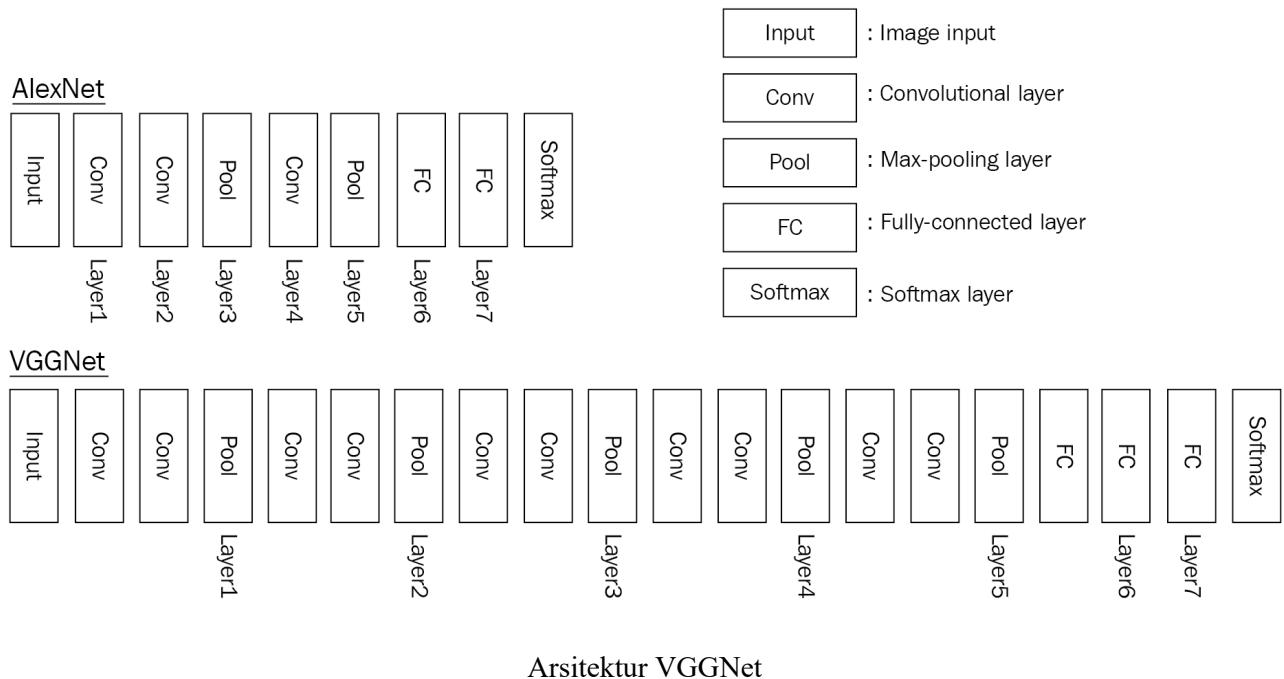
The caffe perpustakaan memiliki ModelZoo, di mana satu dapat berbagi beban jaringan.

Pertimbangkan untuk melatih dari awal jika kumpulan data besar dan sangat berbeda dari kumpulan data asli. Dalam kasus ini, kami memiliki cukup data untuk dilatih dari awal tanpa takut overfitting. Namun, bahkan dalam kasus ini, mungkin bermanfaat untuk menginisialisasi seluruh jaringan dengan bobot terlatih dan menyempurnakannya pada kumpulan data baru.

Contoh pembelajaran transfer

Dalam contoh ini, kita akan mengambil VGGNet terlatih dan menggunakan pembelajaran transfer untuk melatih pengklasifikasi CNN yang memprediksi ras anjing, dengan diberi gambar anjing. Keras berisi banyak model terlatih, bersama dengan kode yang memuat dan memvisualisasikannya. Lainnya adalah kumpulan data bunga yang dapat diunduh di sini. Dataset jenis anjing memiliki 133 kategori jenis anjing dan 8.351 gambar anjing. Unduh kumpulan data Dog breed di sini dan salin ke

folder Anda. VGGNet memiliki 16 konvolusional dengan lapisan penyatuan dari awal hingga akhir dan tiga lapisan yang terhubung sepenuhnya diikuti oleh sebuah softmaxfungsi. Tujuan utamanya adalah untuk menunjukkan bagaimana kedalam jaringan memberikan kinerja terbaik. Itu berasal dari **Visual Geometric Group (VGG)** di Oxford. Jaringan berkinerja terbaik mereka adalah VGG16. Dataset Dog breed relatif kecil dan sedikit tumpang tindih dengan `imageNet`dataset. Jadi, kita dapat menghapus lapisan terakhir yang terhubung sepenuhnya setelah lapisan konvolusional dan menggantinya dengan milik kita. Bobot dari lapisan konvolusional dipertahankan konstan. Gambar masukan dilewatkan melalui lapisan konvolusional dan berhenti di lapisan ke-16:



Kami akan menggunakan fitur bottleneck dari jaringan VGG16 terlatih - jaringan semacam itu telah mempelajari fitur-fitur dari `imageNet`kumpulan data. Karena `imageNet`kumpulan data sudah berisi beberapa gambar anjing, model jaringan VGG16 telah mempelajari fitur utama untuk klasifikasi. Demikian pula, arsitektur CNN terlatih lainnya juga dapat dianggap sebagai latihan untuk menyelesaikan tugas klasifikasi gambar lainnya.

Unduh `bottleneck_features`VGG16 di sini, salin ke folder Anda sendiri, dan muat:

```
bottleneck_features = np.load ('bottleneck_features / DogVGG16Data.npz')
train_vgg16 = bottleneck_features ['train']
valid_vgg16 = bottleneck_features ['valid']
test_vgg16 = bottleneck_features ['test']
```

Sekarang tentukan arsitektur model:

```
dari keras.layers import GlobalAveragePooling2D

model = Sequential ()
model.add (GlobalAveragePooling2D (input_shape = (7, 7, 512)))
model.add (Dense (133, activation = 'softmax'))
model.summary ()

Lapisan (tipe) Parameter Bentuk Output # Terhubung ke
=====
globalaveragepooling2d_1 (Global (Tidak ada, 512) 0 globalaveragepooling2d_input_1 [0]
```

```
dense_2 (Padat) (Tidak ada, 133) 68229 globalaveragepooling2d_1 [0] [0]
=====
Total params: 68.229
Parameter yang bisa dilatih: 68.229
Parameter yang tidak bisa dilatih: 0
```

Kompilasi model dan latih:

```
model.compile (loss = 'kategorikal_crossentropy', pengoptimal = 'rmsprop',
               metrik = ['akurasi'])
dari keras.callbacks import ModelCheckpoint

# latih model
checkpointer = ModelCheckpoint (filepath = 'dogvgg16.weights.best.hdf5', verbose = 1,
                                 save_best_only = True)
model.fit (train_vgg16, train_target, nb_epoch = 20, validation_data = (valid_vgg16, valid_target),
           callbacks = [checkpointer], verbose = 1, shuffle = True)
```

Muat model dan hitung akurasi klasifikasi pada set pengujian:

```
# memuat bobot yang menghasilkan model akurasi validasi
terbaik.load_weights ('dogvgg16.weights.best.hdf5')
# dapatkan indeks prediksi ras anjing untuk setiap gambar dalam set pengujian
vgg16_predictions = [np.argmax (model.predict (np. expand_dims (fitur, sumbu = 0)))
                      untuk fitur dalam test_vgg16]

# laporan akurasi uji
test_accuracy = 100 * np.sum (np.array (vgg16_predictions) ==
                           np.argmax (test_target, axis = 1)) / len (vgg16_predictions )
print ('\nAkurasi pengujian: % .4f %%' % test_accuracy)
```

Pembelajaran multi-tugas

Dalam pembelajaran multi-tugas, pembelajaran transfer terjadi dari satu model yang dilatih sebelumnya ke banyak tugas secara bersamaan. Misalnya, pada mobil yang mengemudi sendiri, jaringan saraf dalam mendekripsi rambu lalu lintas, pejalan kaki, dan mobil lain di depan secara bersamaan. Pengenalan ucapan juga mendapat manfaat dari pembelajaran multi-tugas.

Ringkasan

Dalam beberapa kasus tertentu, arsitektur jaringan neural konvolusional yang dilatih pada gambar memungkinkan kami menggunakan kembali fitur yang dipelajari di jaringan baru. Manfaat kinerja dari mentransfer fitur menurun semakin banyak perbedaan tugas dasar dan tugas target. Mengejutkan mengetahui bahwa menginisialisasi jaringan saraf konvolusional dengan fitur yang ditransfer dari hampir semua lapisan dapat menghasilkan peningkatan kinerja generalisasi setelah menyempurnakan set data baru.

Autoencoder untuk CNN

Dalam bab ini, kami akan membahas topik-topik berikut:

- Memperkenalkan Autoencoders
- Autoencoder Konvolusional
- Aplikasi Autoencoders
- Contoh kompresi

Memperkenalkan autoencoders

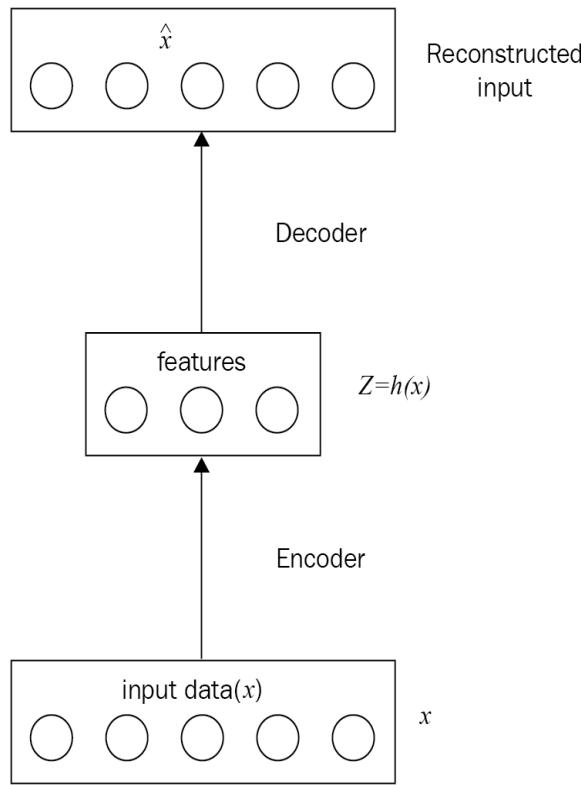
Sebuah autoencoder adalah jaringan saraf yang teratur, model pembelajaran tanpa pengawasan yang mengambil input dan menghasilkan input yang sama pada lapisan output. Jadi, tidak ada label terkait dalam data pelatihan. Umumnya, autoencoder terdiri dari dua bagian:

- Jaringan encoder
- Jaringan decoder

Ia mempelajari semua fitur yang diperlukan dari data pelatihan tak berlabel, yang dikenal sebagai representasi fitur dimensi yang lebih rendah. Pada gambar berikut, data masukan (x) dilewatkan melalui encoder yang menghasilkan representasi terkompresi dari data masukan. Secara matematis, dalam persamaan, $z = h(x)$, z adalah vektor fitur, dan biasanya berdimensi lebih kecil dari x .

Kemudian, kami mengambil fitur-fitur yang dihasilkan dari data input dan meneruskannya melalui jaringan decoder untuk merekonstruksi data asli.

Encoder dapat berupa jaringan neural yang terhubung sepenuhnya atau **jaringan neural konvolusional (CNN)**. Sebuah decoder juga menggunakan jenis yang sama jaringan sebagai encoder. Di sini, kami telah menjelaskan dan mengimplementasikan fungsi encoder dan decoder menggunakan ConvNet:



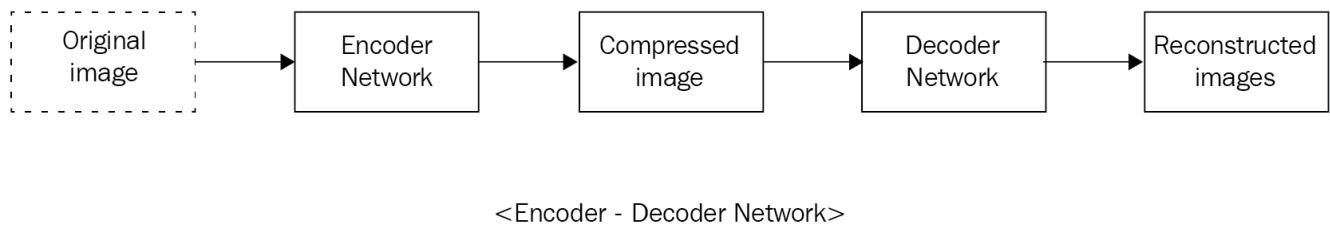
Fungsi kerugian: $\| x - \hat{x} \|_2^2$

Dalam jaringan ini, ukuran lapisan masukan dan keluaran sama.

Autoencoder konvolusional

Autoencoder konvolusional adalah jaringan neural (kasus khusus model pembelajaran tanpa pengawasan) yang dilatih untuk mereproduksi gambar masukannya di lapisan keluaran. Gambar dilewatkan melalui encoder, yang merupakan ConvNet yang menghasilkan representasi gambar berdimensi rendah. Dekoder, yang merupakan contoh ConvNet lainnya, mengambil gambar yang dikompresi ini dan merekonstruksi gambar aslinya.

Encoder digunakan untuk mengompres data dan decoder digunakan untuk mereproduksi gambar asli. Oleh karena itu, autoencoder dapat digunakan untuk data, kompresi. Logika kompresi adalah data-spesifik, artinya ia dipelajari dari data daripada algoritma kompresi yang telah ditentukan sebelumnya seperti JPEG, MP3, dan sebagainya. Aplikasi autoencoder lainnya dapat berupa denoising gambar (menghasilkan gambar yang lebih bersih dari gambar yang rusak), pengurangan dimensi, dan pencarian gambar:



Ini berbeda dari ConvNets biasa atau jaringan neural dalam arti bahwa ukuran masukan dan ukuran target harus sama.

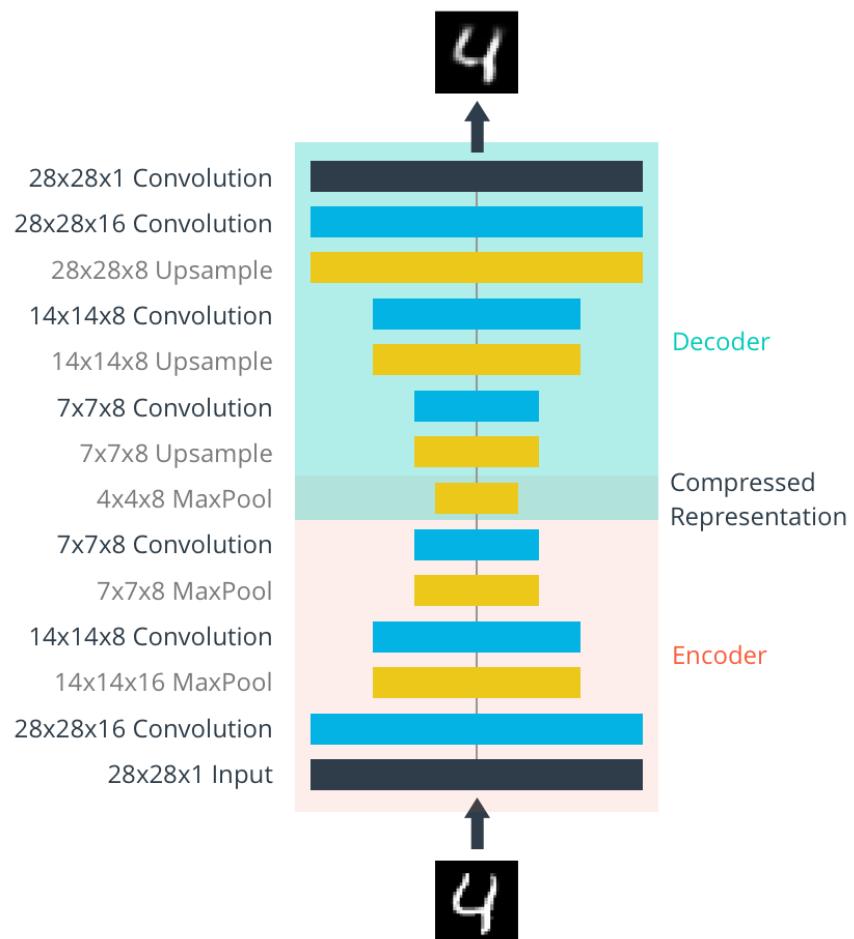
Aplikasi

Autoencoder digunakan untuk reduksi dimensi, atau kompresi data, dan denoising gambar. Pengurangan dimensi, pada gilirannya, membantu meningkatkan kinerja waktu proses dan menghabiskan lebih sedikit memori. Pencarian gambar bisa menjadi sangat efisien dalam ruang berdimensi rendah.

Contoh kompresi

Arsitektur Jaringan terdiri dari jaringan encoder, yang merupakan piramida konvolusional yang khas. Setiap lapisan konvolusional diikuti oleh lapisan penyatuan maksimal; ini mengurangi dimensi lapisan.

Dekoder mengubah masukan dari representasi renggang menjadi gambar rekonstruksi lebar. Skema jaringan ditampilkan di sini:



Ukuran gambar keluaran lapisan encoder adalah $4 \times 4 \times 8 = 128$. Ukuran gambar asli adalah $28 \times 28 \times 1 = 784$, sehingga vektor gambar yang dikompres kira-kira 16% dari ukuran gambar aslinya.

Biasanya, Anda akan melihat lapisan konvolusi yang dialihkan yang digunakan untuk meningkatkan lebar dan tinggi lapisan. Mereka bekerja hampir persis sama dengan lapisan konvolusional tetapi sebaliknya. Langkah pada lapisan masukan menghasilkan langkah yang lebih besar pada lapisan konvolusi yang dialihkan. Misalnya, jika Anda memiliki kernel 3×3 , tambalan 3×3 di lapisan masukan akan dikurangi menjadi satu unit di lapisan konvolusional. Secara komparatif, satu unit di lapisan masukan akan diperluas menjadi jalur 3×3 dalam lapisan konvolusi yang dialihkan. API TensorFlow memberi kita cara mudah untuk membuat lapisan:

`tf.nn.conv2d_transpose` klik di sini https://www.tensorflow.org/api_docs/python/tf/nn/conv2d_transpose.

Ringkasan

Kami memulai bab ini dengan pengenalan singkat tentang autoencoder, dan kami mengimplementasikan fungsi encoder dan decoder dengan bantuan ConvNets.

Kami kemudian beralih ke autoencoder konvolusional dan mempelajari perbedaannya dari ConvNets biasa dan jaringan neural.

Kami menelusuri berbagai aplikasi autoencoders, dengan sebuah contoh, dan melihat bagaimana autoencoder meningkatkan efisiensi pencarian gambar di ruang berdimensi rendah.

Pada bab selanjutnya, kita akan mempelajari deteksi objek dengan CNN dan mempelajari perbedaan antara deteksi objek dan klasifikasi objek.

Deteksi Objek dan Segmentasi Instance dengan CNN

Sampai sekarang, dalam buku ini, kami sebagian besar menggunakan **jaringan saraf konvolusional (CNN)** untuk klasifikasi. Klasifikasi mengklasifikasikan seluruh citra ke dalam salah satu kelas sehubungan dengan entitas yang memiliki probabilitas deteksi maksimum pada citra. Tetapi bagaimana jika tidak ada satu, tetapi beberapa entitas yang diminati dan kita ingin gambarnya terkait dengan semuanya? Salah satu cara untuk melakukannya adalah dengan menggunakan tag alih-alih kelas, di mana tag ini adalah semua kelas dari lapisan klasifikasi Softmax kedua dari belakang dengan probabilitas di atas ambang yang diberikan. Namun, probabilitas pendekripsi di sini sangat bervariasi berdasarkan ukuran dan penempatan entitas, dan dari gambar berikut, kita sebenarnya dapat mengatakan, *Seberapa yakin model bahwa entitas yang diidentifikasi adalah yang diklaim?* Bagaimana jika kita sangat yakin bahwa ada entitas, katakanlah seekor anjing, dalam gambar, tetapi skala dan posisinya dalam gambar tidak menonjol seperti pemiliknya, entitas *Pribadi*? Jadi, *Tag Multikelas* adalah cara yang valid tetapi bukan yang terbaik untuk tujuan ini:



Dalam bab ini, kami akan membahas topik-topik berikut:

- Perbedaan antara deteksi objek dan klasifikasi citra
- Pendekatan tradisional non-CNN untuk deteksi objek
- CNN berbasis wilayah dan fitur-fiturnya
- Cepat R-CNN
- R-CNN lebih cepat
- Masker R-CNN

Perbedaan antara deteksi objek dan klasifikasi citra

Mari kita ambil contoh lain. Anda sedang menonton film *101 Dalmatians*, dan Anda ingin tahu berapa banyak Dalmatians yang dapat Anda hitung dalam adegan film tertentu dari film itu. Klasifikasi Gambar dapat, paling baik, memberi tahu Anda bahwa setidaknya ada satu anjing atau satu *Dalmatian* (tergantung pada level mana Anda telah melatih pengklasifikasi Anda), tetapi tidak persis berapa banyak dari mereka.

Masalah lain dengan model berbasis klasifikasi adalah bahwa model tersebut tidak memberi tahu Anda di mana lokasi entitas yang teridentifikasi dalam gambar. Sering kali, ini sangat penting. Misalnya, Anda melihat anjing tetangga Anda bermain dengan dia (*Orang*) dan kucingnya. Anda mengambil foto mereka dan ingin mengekstrak gambar anjing dari sana untuk mencari di web untuk jenisnya atau anjing serupa yang serupa. Satu-satunya masalah di sini adalah bahwa mencari seluruh gambar mungkin tidak berfungsi, dan tanpa mengidentifikasi objek individu dari gambar, Anda harus melakukan tugas pencarian potong-ekstrak secara manual untuk tugas ini, seperti yang ditunjukkan pada gambar berikut:



Jadi, pada dasarnya Anda memerlukan teknik yang tidak hanya mengidentifikasi entitas dalam gambar, tetapi juga memberi tahu Anda penempatannya di gambar. Inilah yang disebut **deteksi objek**. Deteksi objek memberi Anda kotak pembatas dan label kelas (bersama dengan kemungkinan deteksi) dari semua entitas yang diidentifikasi dalam gambar. Output dari sistem ini dapat digunakan untuk memberdayakan beberapa kasus penggunaan lanjutan yang bekerja pada kelas tertentu dari objek yang terdeteksi.

Ambil contoh, fitur Pengenalan Wajah yang Anda miliki di Facebook, Google Foto, dan banyak aplikasi serupa lainnya. Di dalamnya, sebelum Anda mengidentifikasi *siapa yang* ada di gambar yang diambil di sebuah pesta, Anda perlu mendeteksi semua wajah di gambar itu; kemudian Anda dapat meneruskan wajah-wajah ini melalui modul pengenalan / klasifikasi wajah Anda untuk mendapatkan / mengklasifikasikan

namanya. Jadi, nomenklatur Objek dalam deteksi objek tidak terbatas pada entitas linguistik tetapi mencakup segala sesuatu yang memiliki batasan tertentu dan data yang cukup untuk melatih sistem, seperti yang ditunjukkan pada gambar berikut:



Sekarang, jika Anda ingin mengetahui berapa banyak tamu yang hadir di pesta Anda yang benar-benar **menikmatinya**, Anda bahkan dapat menjalankan deteksi objek untuk **Wajah Tersenyum** atau **Detektor Senyuman**. Ada model pendekripsi objek terlatih yang sangat kuat dan efisien yang tersedia untuk sebagian besar bagian tubuh manusia yang dapat dideteksi (mata, wajah, tubuh bagian atas, dan sebagainya), ekspresi populer manusia (seperti senyuman), dan banyak objek umum lainnya juga. . Jadi, lain kali Anda menggunakan **Rana Senyuman** di ponsel cerdas Anda (fitur yang dibuat untuk mengeklik gambar secara otomatis saat sebagian besar wajah dalam pemandangan terdeteksi sebagai tersenyum), Anda tahu apa yang memberdayakan fitur ini.

Mengapa deteksi objek jauh lebih menantang daripada klasifikasi gambar?

Dari pemahaman kita tentang CNN dan klasifikasi gambar sejauh ini, mari kita coba memahami bagaimana kita dapat mendekati masalah deteksi objek, dan hal itu secara logis mengarahkan kita pada penemuan kompleksitas dan tantangan yang mendasarinya. Asumsikan kita berurusan dengan gambar monokromatik untuk kesederhanaan.

Deteksi objek apa pun pada tingkat tinggi dapat dianggap sebagai kombinasi dari dua tugas (kami akan membantahnya nanti):

- Mendapatkan kotak pembatas yang tepat (atau sebanyak mungkin untuk difilter nanti)

- Mengklasifikasikan objek dalam kotak pembatas (sambil mengembalikan efektivitas klasifikasi untuk pemfilteran)

Jadi, deteksi objek tidak hanya harus memenuhi semua tantangan klasifikasi gambar (tujuan kedua), tetapi juga menghadapi tantangan baru dalam menemukan kotak pembatas yang tepat, atau sebanyak mungkin. Seperti yang telah kita ketahui cara menggunakan CNN untuk tujuan klasifikasi gambar, dan tantangan terkait, sekarang kita dapat berkonsentrasi pada tugas pertama kita dan mengeksplorasi seberapa efektif (akurasi klasifikasi) dan efisien (kompleksitas komputasi) pendekatan kita — atau lebih tepatnya seberapa menantang tugas ini akan menjadi.

Jadi, kita mulai dengan membuat kotak pembatas secara acak dari gambar. Bahkan jika kita tidak khawatir tentang beban komputasi untuk menghasilkan begitu banyak kotak kandidat, yang secara teknis disebut sebagai **Proposal Wilayah** (wilayah yang kami kirim sebagai proposal untuk mengklasifikasikan objek), kami masih perlu memiliki beberapa mekanisme untuk menemukan nilai terbaik untuk parameter berikut :

- Koordinat awal (atau tengah) untuk mengekstrak / menggambar kotak pembatas kandidat
- Panjang kotak pembatas kandidat
- Lebar kotak pembatas kandidat
- Langkah melintasi setiap sumbu (jarak dari satu lokasi awal ke lokasi lainnya pada sumbu x - horizontal dan sumbu y - vertikal)

Mari kita asumsikan bahwa kita dapat menghasilkan algoritme yang dapat memberi kita nilai paling optimal dari parameter ini. Namun, akankah satu nilai untuk parameter ini berfungsi di sebagian besar kasus, atau pada kenyataannya, dalam beberapa kasus umum? Dari pengalaman kami, kami tahu bahwa setiap objek akan memiliki skala yang berbeda, jadi kami tahu bahwa satu nilai tetap untuk L dan W untuk kotak ini tidak akan berfungsi. Juga, kita dapat memahami bahwa objek yang sama, katakanlah Anjing, mungkin ada dalam proporsi / skala dan posisi yang berbeda dalam gambar yang berbeda, seperti dalam beberapa contoh sebelumnya. Jadi ini menegaskan keyakinan kami bahwa kami membutuhkan kotak tidak hanya dengan skala berbeda tetapi juga ukuran berbeda.

Mari kita asumsikan bahwa, mengoreksi dari analogi sebelumnya, kita ingin mengekstrak N jumlah kotak kandidat per koordinat awal pada gambar, di mana N mencakup sebagian besar ukuran / skala yang mungkin sesuai dengan masalah klasifikasi kita. Meskipun itu tampaknya menjadi pekerjaan yang agak menantang, mari kita asumsikan kita memiliki angka ajaib itu dan itu jauh dari kombinasi L [l -image] $\times W$ [l , w -image] (semua kombinasi L dan W di mana panjang adalah himpunan semua bilangan bulat antara 1 dan panjang gambar aktual dan lebarnya dari 1 sampai lebar gambar); yang akan mengarahkan kita ke kotak $l * w$ per koordinat:



Kemudian, pertanyaan lain adalah tentang berapa banyak koordinat awal yang perlu kita kunjungi dalam gambar kita dari mana kita akan mengekstrak masing-masing kotak N ini , atau Stride. Menggunakan langkah yang sangat besar akan mengarahkan kita untuk mengekstrak sub-gambar itu sendiri, alih-alih satu objek homogen yang dapat diklasifikasikan secara efektif dan digunakan untuk tujuan mencapai beberapa tujuan dalam contoh sebelumnya. Sebaliknya, langkah yang terlalu pendek (katakanlah, 1 piksel di setiap arah) dapat berarti banyak kandidat kotak.

Dari ilustrasi sebelumnya, kita dapat memahami bahwa bahkan setelah menghilangkan sebagian besar kendala secara hipotetis, kita masih jauh dari membuat sistem yang dapat kita muat di Smartphone kita untuk mendekripsi selfie yang tersenyum atau bahkan wajah yang cerah secara real time (bahkan setelah satu jam di fakta). Robot dan mobil self-driving kami juga tidak dapat mengidentifikasi objek saat mereka bergerak (dan menavigasi jalan mereka dengan menghindarinya). Intuisi ini akan membantu kita menghargai kemajuan di bidang deteksi objek dan mengapa hal itu menjadi area kerja yang berdampak.

Pendekatan tradisional nonCNN untuk deteksi objek

Library seperti OpenCV dan beberapa lainnya melihat inklusi yang cepat dalam bundel perangkat lunak untuk Smartphone, proyek Robotic, dan banyak lainnya, untuk memberikan kemampuan deteksi objek tertentu (wajah, senyuman, dan sebagainya), dan manfaat seperti Computer Vision, meskipun dengan beberapa kendala bahkan sebelum adopsi CNN yang produktif.

Penelitian berbasis CNN di bidang deteksi objek dan Segmentasi Instans ini memberikan banyak kemajuan dan peningkatan kinerja untuk bidang ini, tidak hanya memungkinkan penerapan sistem dalam skala besar tetapi juga membuka jalan bagi banyak solusi baru. Tetapi sebelum kita berencana untuk terjun ke kemajuan berbasis CNN, akan menjadi ide yang baik untuk memahami bagaimana tantangan yang dikutip di bagian sebelumnya dijawab untuk memungkinkan deteksi objek di tempat pertama (bahkan dengan semua batasan), dan kemudian kita akan melakukannya logis mulai

diskusi kita tentang peneliti yang berbeda dan penerapan CNN untuk memecahkan masalah lain yang masih bertahan dengan penggunaan pendekatan tradisional.

Fitur Haar, pengklasifikasi berjenjang, dan algoritme Viola-Jones

Tidak seperti CNN, atau pembelajaran terdalam dalam hal ini, yang dikenal karena kemampuannya dalam menghasilkan fitur konseptual yang lebih tinggi secara otomatis, yang pada gilirannya memberikan dorongan besar untuk pengklasifikasi, dalam kasus aplikasi pembelajaran mesin tradisional, fitur tersebut perlu ditangani. dibuat oleh UKM.

Seperti yang mungkin juga kami pahami dari pengalaman kami mengerjakan pengklasifikasi pembelajaran mesin berbasis CPU, kinerjanya dipengaruhi oleh dimensi tinggi dalam data dan ketersediaan terlalu banyak fitur untuk diterapkan ke model, terutama dengan beberapa pengklasifikasi yang sangat populer dan canggih seperti sebagai **Support Vector Machines (SVM)**, yang dulu dianggap canggih hingga beberapa waktu lalu.

Pada bagian ini, kita akan memahami beberapa ide inovatif yang menarik inspirasi dari berbagai bidang sains dan matematika yang mengarah pada penyelesaian beberapa tantangan yang dikutip di atas, untuk membawa konsep deteksi objek waktunya dalam sistem non-CNN.

Fitur Haar

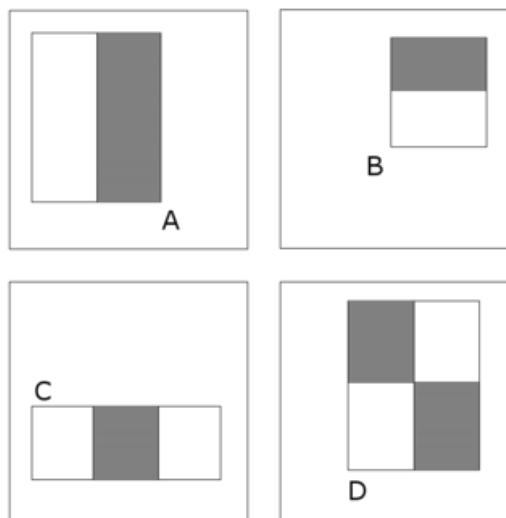
Fitur Haar atau Haar-like adalah bentukan persegi panjang dengan kepadatan piksel yang bervariasi. Fitur Haar merangkum intensitas piksel di wilayah persegi panjang yang berdekatan di lokasi tertentu di wilayah deteksi. Berdasarkan perbedaan antara jumlah intensitas piksel di seluruh wilayah, mereka mengkategorikan subbagian gambar yang berbeda.

Fitur mirip Haar memiliki nama yang dikaitkan dengan istilah matematika Haar wavelet, yang merupakan urutan fungsi berbentuk persegi berskala ulang yang bersama-sama membentuk keluarga atau basis wavelet .

Karena fitur mirip Haar berfungsi pada perbedaan antara intensitas piksel di seluruh wilayah, fitur ini berfungsi paling baik dengan gambar monokrom. Ini juga alasan mengapa gambar yang digunakan sebelumnya dan di bagian ini juga monokrom untuk intuisi yang lebih baik.

Kategori-kategori ini dapat dikelompokkan menjadi tiga kelompok besar, sebagai berikut:

- Dua fitur persegi panjang
- Tiga fitur persegi panjang
- Empat fitur persegi panjang



Fitur seperti Haar

Dengan beberapa trik mudah, penghitungan intensitas yang berbeda-beda di seluruh gambar menjadi sangat efisien dan dapat diproses dengan kecepatan yang sangat tinggi dalam waktu nyata.

Pengklasifikasi bertingkat

Bahkan jika kami dapat mengekstrak fitur Haar dari wilayah tertentu dengan sangat cepat, hal itu tidak menyelesaikan masalah penggalian fitur tersebut dari banyak tempat berbeda dalam gambar; di sinilah konsep fitur berjenjang masuk untuk membantu. Teramati bahwa hanya 1 dari 10.000 sub-wilayah yang ternyata positif untuk wajah dalam klasifikasi, tetapi kami harus mengekstrak semua fitur dan menjalankan seluruh pengklasifikasi di semua wilayah. Lebih lanjut, diamati bahwa dengan hanya menggunakan beberapa fitur (dua di lapisan pertama kaskade), pengklasifikasi dapat menghilangkan sebagian besar wilayah (50% di wilayah pertama kaskade). Juga jika sample terdiri dari just sampel wilayah yang dikurangi ini, maka hanya sedikit lebih banyak fitur (10 fitur di lapisan kedua dari kaskade) yang diperlukan untuk pengklasifikasi yang dapat menyingkirkan lebih banyak kasus, dan seterusnya. Jadi kami melakukan klasifikasi dalam lapisan, dimulai dengan pengklasifikasi yang memerlukan daya komputasi yang sangat rendah untuk menyingkirkan sebagian besar subkawasan, secara bertahap meningkatkan beban komputasi yang diperlukan untuk subset yang tersisa, dan seterusnya.

Algoritma Viola-Jones

Pada tahun 2001, Paul Viola dan Michael Jones mengusulkan solusi yang dapat bekerja dengan baik untuk menjawab beberapa tantangan sebelumnya, tetapi dengan beberapa kendala. Meskipun ini adalah algoritme yang berusia hampir dua dekade, beberapa perangkat lunak computer vision paling populer hingga saat ini, atau setidaknya hingga saat ini, digunakan untuk menyematkannya dalam beberapa bentuk atau lainnya. Fakta ini membuatnya sangat penting untuk memahami algoritme yang sangat sederhana, namun andal ini sebelum kita beralih ke pendekatan berbasis CNN untuk Proposal Wilayah.

OpenCV, salah satu pustaka perangkat lunak paling populer untuk computer vision, menggunakan pengklasifikasi berjenjang sebagai mode utama untuk deteksi objek, dan pengklasifikasi Cascade yang mirip Haar sangat populer dengan OpenCV. Banyak pengklasifikasi Haar yang telah dilatih sebelumnya tersedia untuk ini untuk beberapa jenis objek umum.

Algoritme ini tidak hanya mampu memberikan deteksi dengan **TPR** tinggi (**True Positive Rate**) dan **FPR** rendah (**False Positive Rates**), tetapi juga dapat bekerja secara real time (memproses setidaknya dua frame per detik).

TPR tinggi yang dikombinasikan dengan FPR rendah merupakan kriteria yang sangat penting untuk menentukan ketahanan suatu algoritma.

Batasan algoritma yang mereka usulkan adalah sebagai berikut:

- Ini hanya bisa bekerja untuk mendeteksi, tidak mengenali wajah (mereka mengusulkan algoritma untuk wajah, meskipun hal yang sama dapat digunakan untuk banyak objek lain).
- Wajah harus hadir dalam gambar sebagai tampilan frontal. Tidak ada tampilan lain yang dapat dideteksi.

Inti dari algoritma ini adalah Haar (like) Features dan Cascading Classifiers. Fitur Haar dijelaskan nanti dalam sub-bagian. Algoritme Viola-Jones menggunakan subset fitur Haar untuk menentukan fitur umum pada sebuah wajah seperti:

- Mata (ditentukan oleh fitur dua persegi panjang (horizontal) , dengan persegi panjang horizontal gelap di atas mata yang membentuk alis, diikuti oleh persegi panjang yang lebih terang di bawah)
- Hidung (fitur tiga persegi panjang (vertikal) , dengan hidung sebagai pusat persegi panjang terang dan satu persegi panjang yang lebih gelap di kedua sisi pada hidung, membentuk pelipis), dan seterusnya

Fitur cepat untuk mengekstrak ini kemudian dapat digunakan untuk membuat pengklasifikasi untuk mendeteksi (membedakan) wajah (dari non-wajah).

Fitur Haar, dengan beberapa trik, sangat cepat dihitung.



Algoritma Viola-Jones dan Fitur mirip Haar untuk mendeteksi wajah

Fitur mirip Haar ini kemudian digunakan dalam pengklasifikasi berjenjang untuk mempercepat masalah deteksi tanpa kehilangan ketahanan deteksi.

Dengan demikian, Haar Features dan cascading classifiers menghasilkan beberapa pendekripsi objek individual yang sangat kuat, efektif, dan cepat dari generasi sebelumnya. Tapi tetap saja, pelatihan kaskade ini untuk objek baru sangat memakan waktu, dan mereka memiliki banyak kendala, seperti yang disebutkan sebelumnya. Di sialah detektor objek berbasis CNN generasi baru datang untuk menyelamatkan.

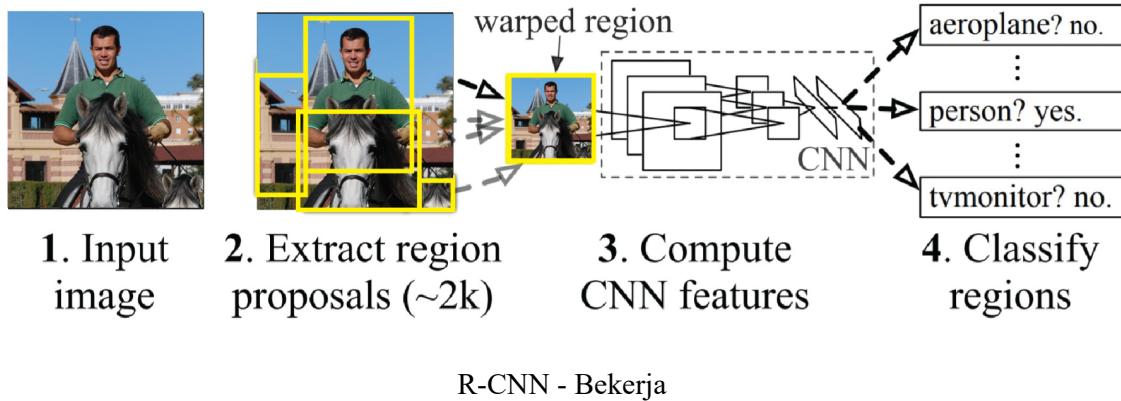
Dalam bab ini, kita hanya membahas dasar fitur Haar-Cascades atau Haar (dalam kategori non-CNN) karena fitur ini tetap dominan untuk waktu yang lama dan merupakan dasar dari banyak jenis baru. Pembaca didorong untuk juga menjelajahi beberapa fitur / kaskade berbasis SIFT dan HOG yang lebih efektif (makalah terkait diberikan di bagian Referensi).

R-CNN - Wilayah dengan fitur CNN

Dalam pertanyaan 'Mengapa deteksi objek jauh lebih menantang daripada klasifikasi gambar?' Pada bagian ini, kami menggunakan metode non-CNN untuk menggambar proposal wilayah dan CNN untuk klasifikasi, dan kami menyadari bahwa ini tidak akan berfungsi dengan baik karena wilayah yang dihasilkan dan dimasukkan ke CNN tidak optimal. R-CNN atau region dengan fitur CNN, seperti namanya, membalik contoh itu sepenuhnya dan menggunakan CNN untuk menghasilkan fitur yang diklasifikasikan menggunakan teknik (non-CNN) yang disebut **SVM (Support Vector Machines)**

R-CNN menggunakan metode jendela geser (seperti yang telah kita diskusikan sebelumnya, menggunakan beberapa $L \times W$ dan langkah) untuk menghasilkan sekitar 2.000 wilayah yang diminati, dan kemudian mengubahnya menjadi fitur untuk klasifikasi menggunakan CNN. Ingat apa yang kita bahas dalam bab pemelajaran transfer— lapisan rata terakhir (sebelum klasifikasi atau lapisan softmax) dapat

diekstraksi untuk mentransfer pembelajaran dari model yang dilatih pada data generalistik, dan selanjutnya melatihnya (seringkali membutuhkan lebih sedikit data dibandingkan dengan model dengan kinerja serupa yang telah dilatih dari awal menggunakan data khusus domain) untuk membuat model model khusus domain. R-CNN juga menggunakan mekanisme serupa untuk meningkatkan efektivitasnya pada deteksi objek tertentu:



*Makalah asli pada R-CNN mengklaim bahwa pada dataset PASCAL VOC 2012, telah meningkatkan **rata-rata presisi rata-rata (mAP)** lebih dari 30% dibandingkan dengan hasil terbaik sebelumnya pada data tersebut sambil mencapai peta 53,3%.*

Kami melihat angka presisi sangat tinggi untuk latihan klasifikasi gambar (menggunakan CNN) di atas data ImageNet. Jangan gunakan angka itu dengan statistik perbandingan yang diberikan di sini, karena tidak hanya kumpulan data yang digunakan berbeda (dan karenanya tidak dapat dibandingkan), tetapi juga tugas yang ada (klasifikasi versus deteksi objek) sangat berbeda, dan deteksi objek jauh lebih menantang tugas daripada klasifikasi gambar.

PASCAL VOC (Visual Object Challenge): Setiap bidang penelitian memerlukan semacam set data standar dan KPI standar untuk membandingkan hasil di berbagai studi dan algoritme. Imagenet, kumpulan data yang kami gunakan untuk klasifikasi gambar, tidak dapat digunakan sebagai kumpulan data standar untuk deteksi objek, karena pendekslsian objek memerlukan data (train, test, dan validation set) yang diberi label tidak hanya dengan kelas objek tetapi juga posisinya. ImageNet tidak menyediakan ini. Oleh karena itu, di sebagian besar studi deteksi objek, kami mungkin melihat penggunaan dataset deteksi objek standar, seperti PASCAL VOC. Dataset PASCAL VOC sejauh ini memiliki 4 varian, VOC2007, VOC2009, VOC2010, dan VOC2012. VOC2012 adalah yang terbaru (dan terkaya) dari semuanya.

Tempat lain yang kami temukan adalah perbedaan skala (dan lokasi) dari wilayah yang diminati, *pengenalan menggunakan wilayah* . Inilah yang disebut tantangan **lokalisasi** ; itu diselesaikan dalam R-CNN dengan menggunakan berbagai bidang reseptif, mulai

dari wilayah setinggi 195 x 195 piksel dan 32 x 32 langkah, hingga yang lebih rendah ke bawah.

Pendekatan ini disebut pengenalan menggunakan wilayah .

Tunggu sebentar! Apakah itu membunyikan bel? Kami mengatakan bahwa kami akan menggunakan CNN untuk menghasilkan fitur dari wilayah ini, tetapi CNN menggunakan masukan berukuran konstan untuk menghasilkan lapisan pipih berukuran tetap. Kami memang memerlukan fitur ukuran tetap (ukuran vektor rata) sebagai masukan ke SVM kami, tetapi di sini ukuran wilayah masukan berubah. Jadi bagaimana cara kerjanya? R-CNN menggunakan teknik populer yang disebut **Affine Image Warping** untuk menghitung input CNN ukuran tetap dari setiap proposal wilayah, apa pun bentuk wilayahnya.

Dalam geometri, transformasi affine adalah nama yang diberikan untuk fungsi transformasi antara ruang affine yang mempertahankan titik, garis lurus, dan bidang. Ruang Affine adalah struktur yang menggeneralisasi properti ruang Euclidian sambil mempertahankan properti yang terkait dengan paralelisme dan skala masing-masing .

Selain tantangan yang telah kami bahas, ada tantangan lain yang patut disebutkan. Wilayah kandidat yang kami hasilkan di langkah pertama (tempat kami melakukan klasifikasi di langkah kedua) tidak terlalu akurat, atau tidak memiliki batas yang ketat di sekitar objek yang diidentifikasi. Jadi kami menyertakan tahap ketiga dalam metode ini, yang meningkatkan akurasi kotak pembatas dengan menjalankan fungsi regresi (disebut **regressor kotak pembatas**) untuk mengidentifikasi batas pemisahan.

R-CNN terbukti sangat berhasil jika dibandingkan dengan pendekatan non-CNN end-to-end sebelumnya. Tapi itu menggunakan CNN hanya untuk mengonversi wilayah menjadi fitur. Seperti yang kami pahami, CNN sangat kuat untuk klasifikasi gambar juga, tetapi karena CNN kami hanya akan bekerja pada gambar wilayah masukan dan bukan pada fitur wilayah yang diratakan, kami tidak dapat menggunakannya di sini secara langsung. Di bagian selanjutnya, kita akan melihat bagaimana mengatasi kendala ini.

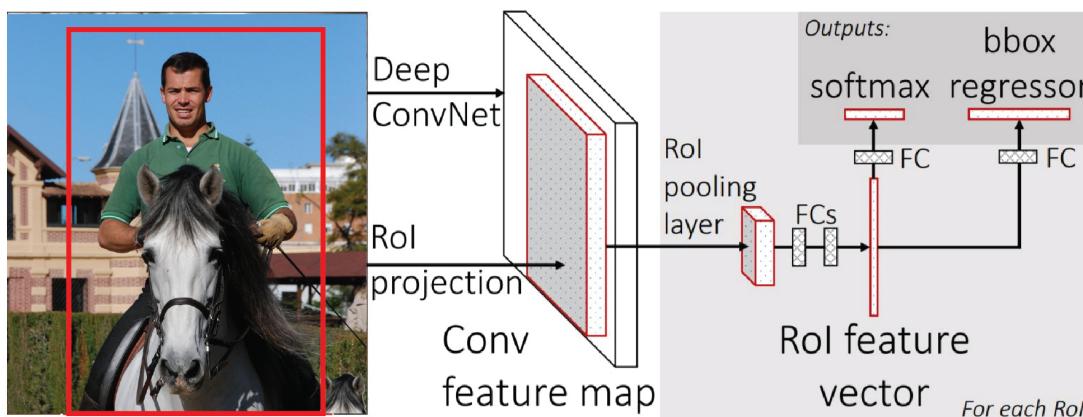
R-CNN sangat penting untuk menutupi dari perspektif pemahaman penggunaan latar belakang CNN dalam deteksi objek karena telah menjadi lompatan besar dari semua pendekatan berbasis non-CNN. Tetapi karena perbaikan lebih lanjut dalam deteksi objek berbasis CNN, seperti yang akan kita bahas selanjutnya, R-CNN tidak aktif bekerja sekarang dan kode tidak dipertahankan lagi.

Fast R-CNN - CNN berbasis wilayah cepat

Fast R-CNN, atau metode CNN berbasis Wilayah Cepat, merupakan peningkatan dari R-CNN yang dicakup sebelumnya. Tepatnya tentang statistik peningkatan, dibandingkan dengan R-CNN, adalah:

- 9x lebih cepat dalam pelatihan
- 213x lebih cepat dalam penilaian / servis / pengujian (0,3 dtk per pemrosesan gambar), mengabaikan waktu yang dihabiskan untuk proposal wilayah
- Memiliki peta yang lebih tinggi sebesar 66% pada dataset PASCAL VOC 2012

Jika R-CNN menggunakan CNN (lima lapis) yang lebih kecil, Fast R-CNN menggunakan jaringan VGG16 yang lebih dalam, yang meningkatkan akurasinya. Selain itu, R-CNN lambat karena menjalankan penerusan ConvNet untuk setiap proposal objek tanpa berbagi komputasi:



Fast R-CNN: Bekerja

Di Fast R-CNN, Deep VGG16 CNN menyediakan perhitungan penting untuk semua tahapan, yaitu:

- Perhitungan **Region of Interest (RoI)**
- Objek Klasifikasi (atau latar belakang) untuk konten wilayah
- Regresi untuk menyempurnakan kotak pembatas

Masukan ke CNN, dalam hal ini, bukanlah wilayah mentah (kandidat) dari gambar, tetapi gambar (lengkap) itu sendiri; keluarannya bukanlah lapisan pipih terakhir tetapi lapisan konvolusi (peta) sebelumnya. Dari peta konvolusi yang dihasilkan, lapisan penyatuhan RoI (varian penyatuhan maksimal) digunakan untuk menghasilkan RoI dengan panjang tetap yang diratakan yang sesuai dengan setiap proposal objek yang dihasilkan, yang kemudian diteruskan melalui beberapa yang **sepenuhnya terhubung** (**FC**) lapisan.

Penggabungan RoI adalah varian dari penggabungan maksimal (yang kita gunakan di bab awal dalam buku ini), di mana ukuran keluaran ditetapkan dan persegi panjang masukan adalah parameternya.

Lapisan penggabungan RoI menggunakan penggabungan maksimal untuk mengubah fitur di dalam wilayah minat yang valid menjadi peta fitur kecil dengan tingkat spasial tetap.

Output dari lapisan FC kedua dari belakang kemudian digunakan untuk keduanya:

- Klasifikasi (lapisan SoftMax) dengan jumlah kelas sebanyak proposal objek, +1 kelas tambahan untuk latar belakang (tidak ada kelas yang ditemukan di wilayah tersebut)
- Kumpulan regresi yang menghasilkan empat angka (dua angka yang menunjukkan koordinat x, y dari sudut kiri atas kotak untuk objek itu, dan dua angka berikutnya yang sesuai dengan tinggi dan lebar objek yang ditemukan di wilayah itu) untuk setiap objek-proposal yang diperlukan untuk membuat kotak pembatas tepat untuk objek tertentu itu

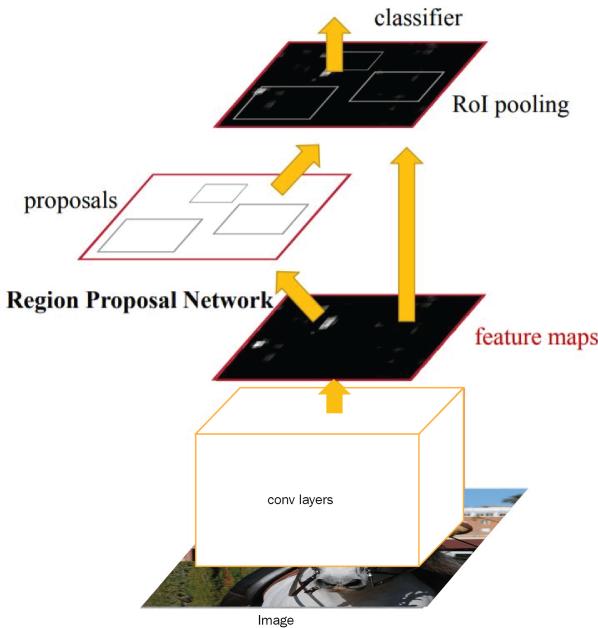
Hasil yang dicapai dengan Fast R-CNN sangat bagus. Yang lebih hebat lagi adalah penggunaan jaringan CNN yang kuat untuk menyediakan fitur yang sangat efektif untuk ketiga tantangan yang perlu kita atasi. Tetapi masih ada beberapa kekurangan, dan ada ruang untuk perbaikan lebih lanjut seperti yang akan kita pahami di bagian selanjutnya tentang Faster R-CNN.

R-CNN Lebih Cepat - CNN berbasis jaringan proposal region yang lebih cepat

Kita telah melihat di bagian sebelumnya bahwa Fast R-CNN menurunkan waktu yang dibutuhkan untuk menilai (pengujian) gambar secara drastis, tetapi pengurangan tersebut mengabaikan waktu yang diperlukan untuk membuat Proposal Wilayah, yang menggunakan mekanisme terpisah (meskipun menarik dari peta konvolusi dari CNN) dan terus membuktikan adanya hambatan. Juga, kami mengamati bahwa meskipun ketiga tantangan diselesaikan menggunakan fitur umum dari peta-konvolusi di Fast R-CNN, mereka menggunakan mekanisme / model yang berbeda.

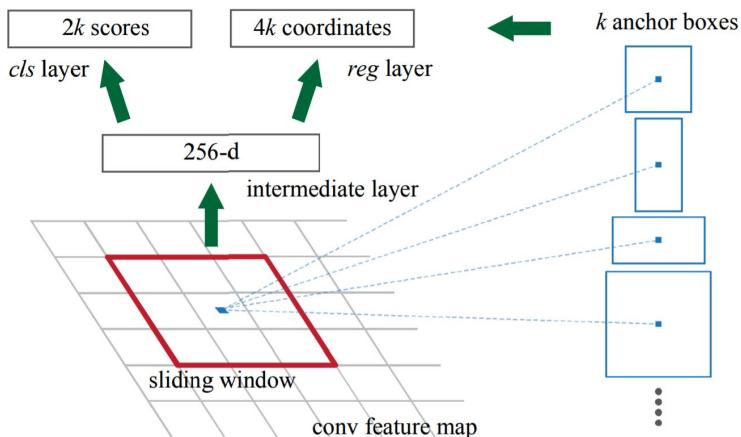
R-CNN yang lebih cepat memperbaiki kekurangan ini dan mengusulkan konsep **Jaringan Proposal Wilayah (RPN)**, menurunkan waktu penilaian (pengujian) menjadi 0,2 detik per gambar, bahkan termasuk waktu untuk Proposal Wilayah.

Fast R-CNN melakukan scoring (pengujian) dalam 0,3 detik per gambar, itu juga tidak termasuk waktu yang dibutuhkan untuk proses yang setara dengan Region Proposal.



R-CNN Lebih Cepat: Bekerja - Jaringan Proposial Wilayah bertindak sebagai Mekanisme Perhatian

Seperti yang ditunjukkan pada gambar sebelumnya, CNN VGG16 (atau lainnya) bekerja langsung pada gambar, menghasilkan peta konvolusional (mirip dengan apa yang dilakukan di Fast R-CNN). Hal-hal berbeda dari sini, di mana sekarang ada dua cabang, satu dimasukkan ke RPN dan yang lainnya ke Jaringan deteksi. Ini lagi - lagi merupakan perpanjangan dari CNN yang sama untuk prediksi, yang mengarah ke a **Jaringan Konvolusional Penuh (FCN)**. The RPN bertindak sebagai Mekanisme Perhatian dan juga berbagi penuh gambar fitur konvolusional dengan jaringan deteksi. Selain itu, sekarang karena semua bagian dalam jaringan dapat menggunakan komputasi berbasis GPU yang efisien, hal ini mengurangi waktu keseluruhan yang diperlukan:

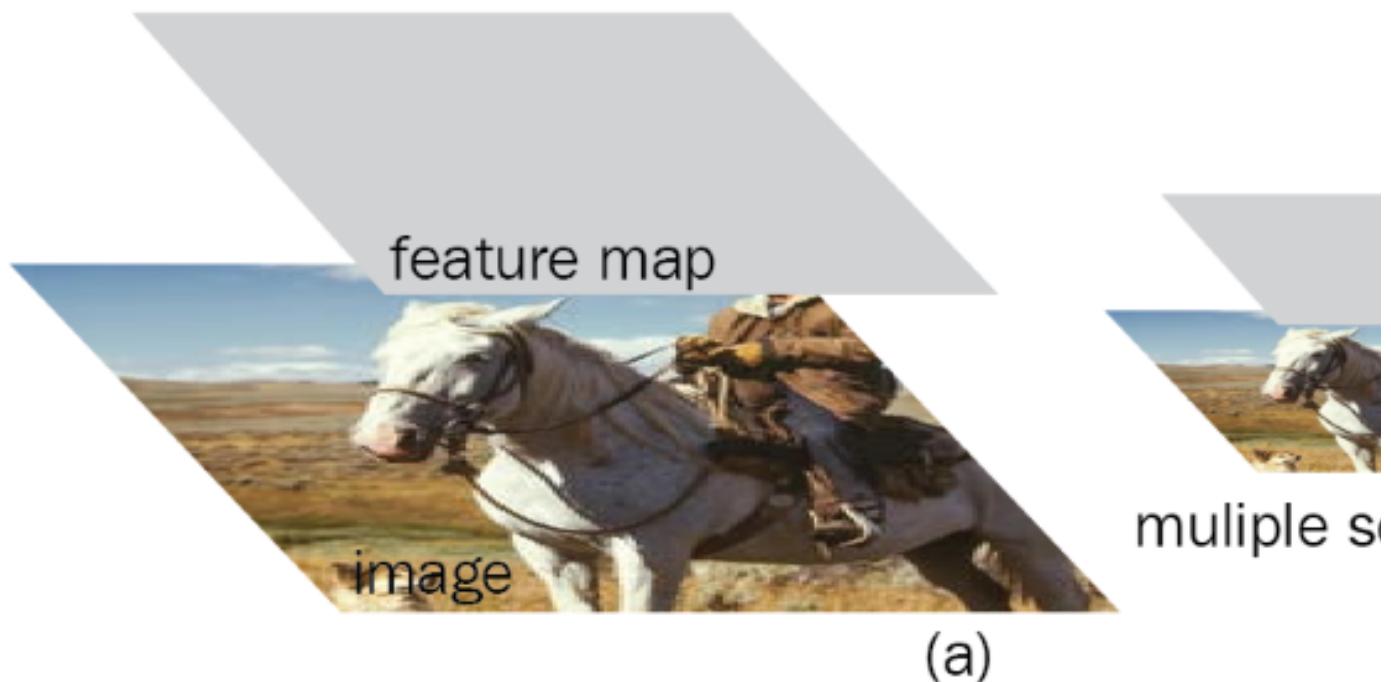


R-CNN Lebih Cepat: Bekerja - Jaringan Proposal Wilayah bertindak sebagai Mekanisme Perhatian

Untuk pemahaman yang lebih baik tentang Mekanisme Perhatian, lihat bab tentang Mekanisme Perhatian untuk CNN dalam buku ini.

RPN bekerja dalam mekanisme jendela geser, di mana jendela bergeser (seperti filter CNN) melintasi peta konvolusi terakhir dari lapisan konvolusional bersama. Dengan setiap slide, jendela geser menghasilkan k ($k = N_{Scale} \times N_{Size}$) jumlah Anchor Boxes (mirip dengan Candidate Boxes), dimana N_{Scale} adalah jumlah (seperti piramida) per ukuran dari ukuran N_{Size} (aspek rasio) kotak diekstrak dari tengah jendela geser, seperti gambar berikut.

RPN mengarah ke lapisan FC yang diratakan. Ini, pada gilirannya, mengarah ke dua jaringan, satu untuk memprediksi empat angka untuk masing-masing kotak k (menentukan koordinat, panjang dan lebar kotak seperti dalam Fast R-CNN), dan satu lagi ke dalam model klasifikasi binomial yang menentukan objek atau probabilitas menemukan salah satu objek yang diberikan dalam kotak itu. Output dari RPN mengarah ke jaringan deteksi, yang mendeteksi kelas objek tertentu yang berada di masing-masing kotak k dengan mengetahui posisi kotak dan objeknya.



Faster R-CNN: Bekerja - mengekstraksi skala dan ukuran yang berbeda

Salah satu masalah dalam arsitektur ini adalah adanya pelatihan terhadap dua jaringan yaitu Region Proposal dan jaringan deteksi. Kami mempelajari bahwa CNN dilatih menggunakan backpropagating di semua lapisan sambil mengurangi lapisan kerugian

dengan setiap iterasi. Tetapi karena terpecah menjadi dua jaringan yang berbeda, kami dapat melakukan propagasi mundur pada satu waktu hanya melalui satu jaringan. Untuk mengatasi masalah ini, pelatihan dilakukan secara berulang di setiap jaringan, sambil menjaga bobot jaringan lain tetap konstan. Ini membantu dalam menyatukan kedua jaringan dengan cepat.

Fitur penting dari arsitektur RPN adalah fitur ini invariansi terjemahan sehubungan dengan kedua fungsi tersebut, yang menghasilkan jangkar, dan yang lainnya menghasilkan atribut (koordinat dan objektivitasnya) untuk jangkar. Karena invariansi translasi, operasi kebalikannya, atau menghasilkan bagian gambar yang diberikan peta vektor dari peta jangkar dimungkinkan.

Berkat Translational Invariance, kita dapat bergerak ke salah satu arah dalam CNN, yaitu dari proposal gambar ke (wilayah), dan dari proposal ke bagian gambar yang sesuai.

Mask R-CNN - Segmentasi instance dengan CNN

R-CNN yang lebih cepat adalah alat canggih dalam deteksi objek hari ini. Tetapi ada masalah yang tumpang tindih dengan area deteksi objek yang tidak dapat diselesaikan secara efektif oleh Faster R-CNN, di mana Mask R-CNN, sebuah evolusi dari Faster R-CNN dapat membantu.

Bagian ini memperkenalkan konsep segmentasi contoh, yang merupakan kombinasi dari masalah deteksi objek standar seperti yang dijelaskan dalam bab ini, dan tantangan segmentasi emantik. .

Dalam segmentasi semantik, seperti yang diterapkan pada gambar, tujuannya adalah untuk mengklasifikasikan setiap piksel ke dalam kumpulan kategori tetap tanpa membedakan contoh objek.

Ingat contoh kami menghitung jumlah anjing pada gambar di bagian intuisi? Kami dapat menghitung jumlah anjing dengan mudah, karena mereka sangat berjauhan, tanpa tumpang tindih, jadi pada dasarnya hanya menghitung jumlah objek sudah cukup. Sekarang, ambil gambar berikut, misalnya, dan hitung jumlah tomat menggunakan deteksi objek. Ini akan menjadi tugas yang menakutkan karena Bounding Box akan memiliki banyak tumpang tindih sehingga akan sulit untuk membedakan Contoh tomat dari kotaknya.

Jadi, pada dasarnya, kita perlu melangkah lebih jauh, melampaui kotak pembatas dan ke piksel untuk mendapatkan pemisahan dan identifikasi level itu. Seperti yang kami gunakan untuk mengklasifikasikan kotak pembatas dengan nama objek dalam deteksi

objek, di Segmen Instance, kami menyegmentasikan / mengklasifikasikan, setiap piksel tidak hanya dengan nama objek tertentu tetapi juga objek-instance.

Deteksi objek dan Segmentasi Instance dapat diperlakukan sebagai dua tugas yang berbeda, satu secara logis mengarah ke yang lain, sama seperti kami menemukan tugas untuk menemukan Proposal Wilayah dan Klasifikasi dalam kasus deteksi objek. Tetapi seperti dalam kasus deteksi objek, dan terutama dengan teknik seperti Fast / Faster R-CNN, kami menemukan bahwa akan jauh lebih efektif jika kami memiliki mekanisme untuk melakukannya secara bersamaan, sementara juga memanfaatkan banyak komputasi dan jaringan untuk melakukannya. jadi, untuk mempermudah tugas.



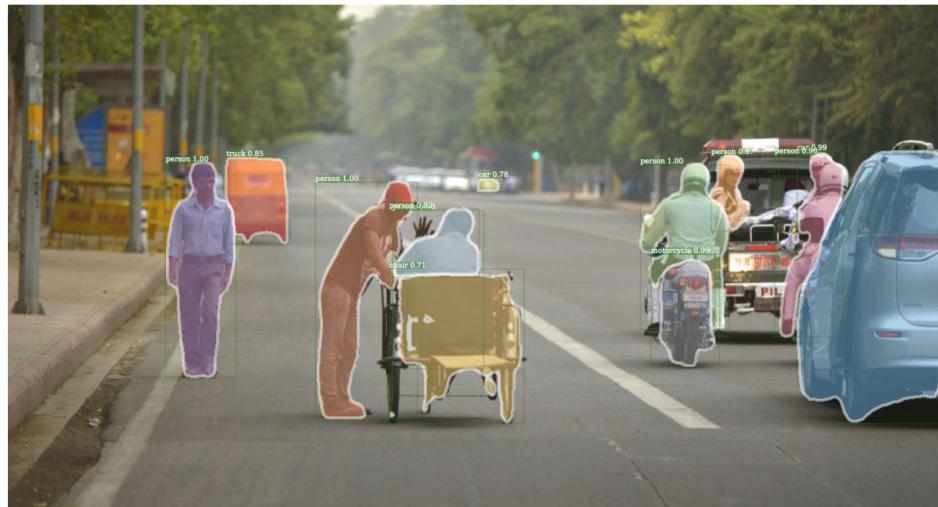
Segmentasi Instance - Intuisi

Mask R-CNN adalah perpanjangan dari Faster R-CNN yang tercakup dalam jaringan sebelumnya, dan menggunakan semua teknik yang digunakan dalam Faster R-CNN, dengan satu tambahan — jalur tambahan dalam jaringan untuk menghasilkan Segmentation Mask (atau Object Mask) untuk setiap Contoh Objek yang terdeteksi secara paralel. Selain itu, karena pendekatan ini menggunakan sebagian besar jaringan yang ada, ia hanya menambahkan overhead minimal ke seluruh pemrosesan dan memiliki waktu penilaian (pengujian) yang hampir setara dengan Faster R-CNN. Ini memiliki salah satu akurasi terbaik di semua solusi model tunggal seperti yang diterapkan pada tantangan COCO2016 (menggunakan kumpulan data COCO2015).

Seperti, PASCAL VOC, COCO lain adalah skala besar standar (seri) dataset (dari Microsoft). Selain deteksi objek, COCO juga digunakan untuk segmentasi dan captioning. COCO lebih luas daripada banyak kumpulan data lain dan banyak perbandingan terbaru pada deteksi objek dilakukan untuk tujuan perbandingan. Dataset COCO hadir dalam tiga varian yaitu COCO 2014, COCO 2015, dan COCO 2017.

Di Mask R-CNN, selain memiliki dua cabang yang menghasilkan objektivitas dan lokalisasi untuk setiap kotak jangkar atau ROI, ada juga FCN ketiga yang mengambil ROI dan memprediksi topeng segmentasi dengan cara piksel-ke-piksel untuk diberi kotak jangkar.

Namun masih ada beberapa tantangan. Meskipun Faster R-CNN mendemonstrasikan invariansi transformasional (yaitu, kita dapat melacak dari peta konvolusional RPN ke peta piksel dari gambar aktual), peta konvolusional memiliki struktur yang berbeda dari yang ada pada piksel gambar aktual. Jadi, tidak ada penyelarasan piksel-ke-piksel antara input dan output jaringan, yang penting untuk tujuan kami menyediakan masking piksel-ke-piksel menggunakan jaringan ini. Untuk mengatasi tantangan ini, Mask R-CNN menggunakan lapisan bebas kuantisasi (dinamai RoIAlign di kertas asli) yang membantu menyelaraskan lokasi spasial yang tepat. Lapisan ini tidak hanya memberikan penyelarasan yang tepat tetapi juga membantu dalam meningkatkan akurasi sebagian besar, karena itu Mask R-CNN mampu mengungguli banyak jaringan lain:



Mask R-CNN - Masker Segmentasi Instance (keluaran ilustrasi)

Konsep segmentasi instance sangat kuat dan dapat menghasilkan banyak kasus penggunaan yang sangat berdampak yang tidak mungkin dilakukan hanya dengan deteksi objek.

Kami bahkan dapat menggunakan segmentasi contoh untuk memperkirakan pose manusia dalam kerangka yang sama dan menghilangkannya.

Segmentasi instance dalam kode

Sekarang saatnya untuk mempraktikkan hal-hal yang telah kita pelajari. Kami akan menggunakan kumpulan data COCO dan API-nya untuk data, dan menggunakan proyek Detectron Facebook Research (tautan dalam Referensi), yang menyediakan implementasi Python dari banyak teknik yang telah dibahas sebelumnya di bawah lisensi Apache 2.0. Kode bekerja dengan Python2 dan Caffe2, jadi kita membutuhkan lingkungan virtual dengan konfigurasi yang diberikan.

Menciptakan lingkungan

Lingkungan virtual, dengan penginstalan Caffe2, dapat dibuat sesuai petunjuk penginstalan di link repositori Caffe2 di Bagian *Referensi*. Selanjutnya, kami akan menginstal dependensi.

Menginstal dependensi Python (lingkungan Python2)

Kita dapat menginstal dependensi Python seperti yang ditunjukkan pada blok kode berikut:

Python 2X dan Python 3X adalah dua rasa Python yang berbeda (atau lebih tepatnya CPython), dan bukan peningkatan versi konvensional, oleh karena itu pustaka untuk satu varian mungkin tidak kompatibel dengan yang lain. Gunakan Python 2X untuk bagian ini.

Ketika kita mengacu pada bahasa pemrograman (ditafsirkan) Python, kita perlu merujuknya dengan penerjemah khusus (karena ini adalah bahasa yang ditafsirkan sebagai lawan dari bahasa yang dikompilasi seperti Java). Penerjemah yang secara implisit kami sebut sebagai juru bahasa Python (seperti yang Anda unduh dari Python.org atau yang disertakan bersama Anaconda) secara teknis disebut CPython, yang merupakan penerjemah kode byte default dari Python, yang tertulis di C. Tapi ada interpreter Python lain juga seperti Jython (dibangun di Java), PyPy (ditulis dengan Python itu sendiri - tidak begitu intuitif, kan?), IronPython (implementasi .NET Python).

```
| pip install numpy> = 1.13 pyyaml> = 3.12 matplotlib opencv-python> = 3.2 setuptools Cython mock  
| scipy
```

Mengunduh dan menginstal COCO API dan pustaka detektor (perintah shell OS)

Kami kemudian akan mengunduh dan menginstal dependensi Python seperti yang ditunjukkan pada blok kode berikut:

```
# Unduh dan instal COCO API  
# COCOAPI = / path / to / clone / cocoapi  
git clone https://github.com/cocodataset/cocoapi.git $ COCOAPI  
cd $ COCOAPI / PythonAPI  
buat instal
```

```
# Unduh dan instal perpustakaan detectron
# DETECTRON = / path / ke / clone / detectron
git clone https://github.com/facebookresearch/detectron $ DETECTRON
cd $ DETECTRON / lib && make
```

Alternatifnya, kita dapat mengunduh dan menggunakan image Docker dari lingkungan (memerlukan dukungan GPU Nvidia):

```
# DOCKER image build
cd $ DETECTRON / docker docker build -t detectron: c2-cuda9-cudnn7.
nvidia-docker run --rm -it detectron: c2-cuda9-cudnn7 python2 tes / test_batch_permutation_op.py
```

Mempersiapkan struktur folder dataset COCO

Sekarang kita akan melihat kode untuk menyiapkan struktur folder dataset COCO sebagai berikut:

```
# Kita memerlukan struktur Folder berikut: coco [coco_train2014, coco_val2014, anotasi]
mkdir -p $ DETECTRON / lib / dataset / data / coco
ln -s / path / to / coco_train2014 $ DETECTRON / lib / dataset / data / coco /
ln -s / path / to / coco_val2014 $ DETECTRON / lib / dataset / data / coco /
ln -s / path / to / json / annotations $ DETECTRON / lib / dataset / data / coco / annotations
```

Menjalankan model terlatih pada set data COCO

Sekarang kita dapat menerapkan model terlatih pada set data COCO seperti yang ditunjukkan dalam cuplikan kode berikut:

```
python2 tools / test_net.py \
--cfg configs / 12_2017_baselines / e2e_mask_rcnn_R-101-FPN_2x.yaml \
TEST.WEIGHTS https://s3-us-west-
2.amazonaws.com/detectron/35861858/12_2017_baselines/e2e_mask_rcnn_R-101-
FPN_2x.yaml.02_32_51.SgT4y1c0 / output / train / coco_2014_train: coco_2014_valized \
NUM_GPUS 1
```

Referensi

1. Paul Viola dan Michael Jones, Deteksi objek cepat menggunakan rangkaian fitur sederhana yang ditingkatkan, *Conference on Computer Vision and Pattern Recognition* , 2001.
2. Paul Viola dan Michael Jones, Robust Real-time object detection, *International Journal of Computer Vision* , 2001.
3. Itseez2015opencv, OpenCV, *Open Source Computer Vision Library* , Itseez, 2015.

4. Ross B.Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Hierarki fitur kaya untuk deteksi objek yang akurat dan segmentasi semantik* , CoRR, arXiv: 1311.2524, 2013.
5. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Hierarki fitur yang kaya untuk deteksi objek yang akurat dan segmentasi semantik* , Computer Vision and Pattern Recognition, 2014.
6. M. Everingham, L. VanGool, CKI Williams, J. Winn, A. Zisserman, *Tantangan Kelas Objek Visual PASCAL 2012* , VOC2012, Hasil.
7. D. Lowe. *Fitur gambar yang khas dari titik kunci invarian-skala* , IJCV, 2004.
8. N. Dalal dan B. Triggs. *Histogram gradien berorientasi untuk deteksi manusia* . Dalam CVPR, 2005.
9. Ross B.Girshick, Fast R-CNN, CoRR, arXiv: 1504.08083, 2015.
10. Rbgirshick, fast-rcnn, GitHub, <https://github.com/rbgirshick/fast-rcnn> , Feb-2018.
11. Shaoqing Ren, Kaiming He, Ross B.Girshick, Jian Sun, Faster R-CNN: *Towards Real-Time Object Detection with Region Proposal Networks* , CoRR, arXiv: 1506.01497, 2015.
12. Shaoqing Ren dan Kaiming He dan Ross Girshick dan Jian Sun, Faster R-CNN: *Menuju Deteksi Objek Real-Time dengan Region Proposal Networks* , Maju dalam **Neural Information Processing Systems (NIPS)**, 2015.
13. Rbgirshick, py-lebih cepat-rcnn, GitHub, <https://github.com/rbgirshick/py-faster-rcnn> , Feb-2018.
14. Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollar, Kaiming He, Detectron, GitHub, <https://github.com/facebookresearch/Detectron> , Feb-2018.
15. Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B.Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, C. Lawrence Zitnick, *Microsoft COCO: Common Objects in Context* , CoRR, arXiv: 1405.0312, 2014.
16. Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross B.Girshick, Mask R-CNN, CoRR, arXiv: 1703.06870, 2017.
17. Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, Hartwig Adam, MaskLab: *Segmentasi Instance dengan Menyempurnakan Deteksi Objek dengan Fitur Semantik dan Arah* , CoRR, arXiv: 1712.04837, 2017.

18. Anurag Arnab, Philip HS Torr, *Segmentasi Instans Pixelwise dengan Jaringan Instansi Dinamis* , CoRR, arXiv: 1704.02386, 2017.
19. Matterport, Mask_RCNN, GitHub, https://github.com/matterport/Mask_RCNN , Feb-2018.
20. CharlesShang, FastMaskRCNN, GitHub, <https://github.com/CharlesShang/FastMaskRCNN> , Feb-2018.
21. Caffe2, Caffe2, GitHub, <https://github.com/caffe2/caffe2> , Feb-2018.

Ringkasan

Dalam bab ini, kami memulai dari intuisi yang sangat sederhana di balik tugas deteksi objek dan kemudian berlanjut ke konsep yang sangat canggih, seperti Segmentasi Instance, yang merupakan area penelitian kontemporer. Deteksi objek merupakan inti dari banyak inovasi di bidang Ritel, Media, Media Sosial, Mobilitas, dan Keamanan; Ada banyak potensi untuk menggunakan teknologi ini untuk menciptakan fitur yang sangat berdampak dan menguntungkan untuk konsumsi perusahaan dan sosial.

Dari perspektif Algoritma, bab ini dimulai dengan algoritma Viola-Jones yang legendaris dan mekanisme yang mendasarinya, seperti Haar Features dan Cascading Classifiers. Dengan menggunakan intuisi tersebut, kami mulai menjelajahi dunia CNN untuk deteksi objek dengan algoritme, seperti R-CNN, Fast R-CNN, hingga Faster R-CNN yang sangat canggih.

Dalam bab ini, kami juga meletakkan dasar dan memperkenalkan bidang penelitian terbaru dan berdampak yang disebut **segmentasi contoh** . Kami juga membahas beberapa Deep CNN yang canggih berdasarkan metode, seperti Mask R-CNN, untuk implementasi segmentasi instans yang mudah dan berkinerja baik.

GAN: Menghasilkan Gambar Baru dengan CNN

Umumnya, jaringan saraf memerlukan contoh berlabel untuk belajar secara efektif. Pendekatan pembelajaran tanpa pengawasan untuk belajar dari data tidak berlabel tidak bekerja dengan baik. Sebuah **jaringan adversarial generatif** , atau hanya **GAN**, adalah bagian dari pendekatan pembelajaran tanpa pengawasan tetapi berdasarkan jaringan generator yang dapat dibedakan. GAN pertama kali ditemukan oleh Ian Goodfellow dan lainnya pada tahun 2014. Sejak saat itu, GAN menjadi sangat populer. Ini didasarkan pada teori permainan dan memiliki dua pemain atau jaringan: jaringan generator dan b) jaringan diskriminasi, keduanya bersaing satu sama lain.

Pendekatan berbasis teori permainan jaringan ganda ini sangat meningkatkan proses pembelajaran dari data yang tidak berlabel. Jaringan generator menghasilkan data palsu dan meneruskannya ke diskriminator. Jaringan diskriminator juga melihat data nyata dan memprediksi apakah data yang diterimanya itu palsu atau nyata. Jadi generator dilatih agar dapat dengan mudah menghasilkan data yang sangat dekat dengan data nyata untuk mengelabui jaringan diskriminator. Diskriminasi Jaringan ator dilatih untuk mengklasifikasikan data mana yang nyata dan mana yang palsu. Jadi, pada akhirnya, jaringan generator belajar menghasilkan data yang sangat, sangat mirip dengan data nyata. GAN akan menjadi sangat populer di bidang musik dan seni.

Menurut Goodfellow, " Anda dapat menganggap model generatif seperti memberi Kecerdasan Buatan suatu bentuk imajinasi . "

Berikut adalah beberapa contoh GAN:

- Pix2pix
- CycleGAN

Pix2pix - Terjemahan Gambar-ke-Gambar GAN

Jaringan ini menggunakan jaringan **adversarial generatif bersyarat (cGAN)** untuk mempelajari pemetaan dari masukan dan keluaran gambar. Beberapa contoh yang dapat dilakukan dari kertas asli adalah sebagai berikut:



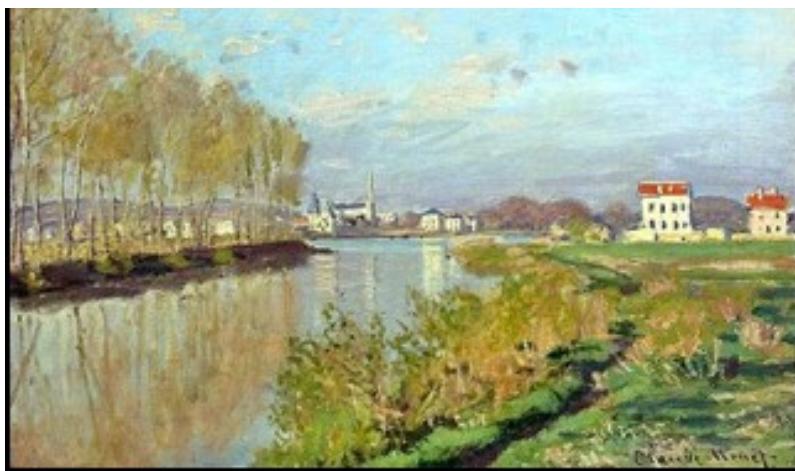
Dalam contoh tas tangan, jaringan belajar bagaimana mewarnai gambar hitam putih. Di sini, dataset pelatihan memiliki citra masukan hitam-putih dan citra target adalah versi warna.

CycleGAN

CycleGAN juga merupakan penerjemah gambar-ke-gambar tetapi tanpa pasangan input / output. Misalnya, untuk menghasilkan foto dari lukisan, ubah gambar kuda menjadi gambar zebra:

Monet  Photos

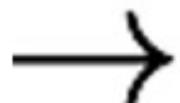




Monet → photo



photo → Monet

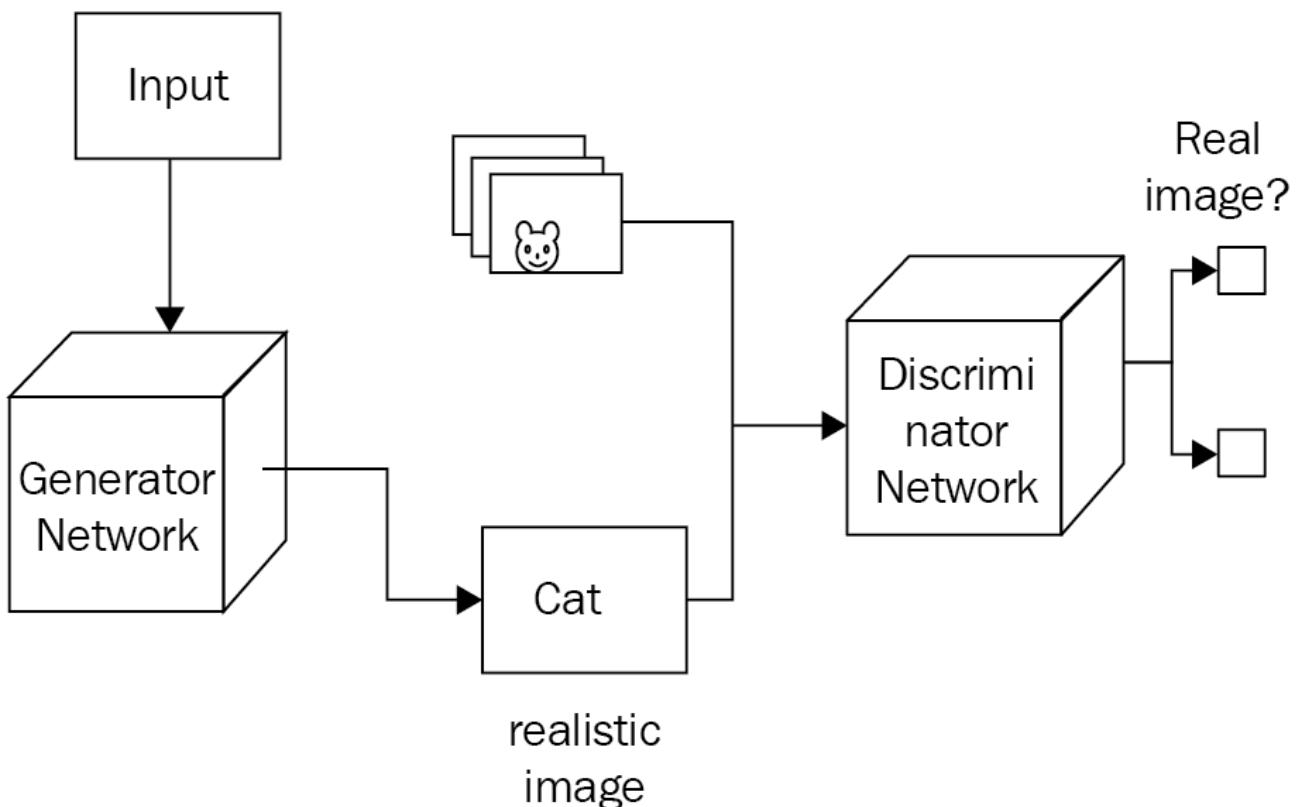


Photograph

Dalam jaringan diskriminasi, penggunaan putus sekolah menjadi penting. Jika tidak, itu mungkin menghasilkan hasil yang buruk.

Jaringan generator mengambil noise acak sebagai input dan menghasilkan gambar realistik sebagai output. Menjalankan jaringan generator untuk berbagai jenis gangguan acak menghasilkan berbagai jenis gambar realistik. Jaringan kedua, yang dikenal sebagai **jaringan diskriminasi**, sangat mirip dengan pengklasifikasi jaringan neural biasa. Jaringan ini dilatih pada gambar nyata, meskipun melatih GAN sangat berbeda dari metode pelatihan yang diawasi. Dalam pelatihan yang diawasi, setiap gambar diberi label terlebih dahulu sebelum ditampilkan ke model. Misalnya, jika inputnya adalah gambar anjing, kami memberi tahu modelnya bahwa ini adalah seekor anjing. Dalam kasus model generatif, kami menunjukkan model itu banyak gambar dan memintanya untuk membuat lebih banyak gambar serupa dari distribusi probabilitas yang sama. Sebenarnya, jaringan diskriminasi membantu jaringan generator untuk mencapai hal ini.

Diskriminator mengeluarkan probabilitas bahwa gambar itu nyata atau palsu dari jaringan generator. Dengan kata lain, ia mencoba menetapkan probabilitas mendekati 1 untuk citra nyata dan probabilitas mendekati 0 untuk citra palsu. Sementara itu, generator melakukan yang sebaliknya. Ini dilatih untuk mengeluarkan gambar yang akan memiliki probabilitas mendekati 1 oleh diskriminator. Seiring waktu, generator menghasilkan gambar yang lebih realistik dan mengelabui diskriminator:



Melatih model GAN

Sebagian besar model pembelajaran mesin yang dijelaskan di bab sebelumnya didasarkan pada pengoptimalan, yaitu, kami meminimalkan fungsi biaya melalui ruang parameternya. GAN berbeda karena dua jaringan: generator G dan diskriminator D. Masing-masing memiliki biayanya sendiri. Cara mudah untuk memvisualisasikan GAN adalah biaya diskriminasi adalah negatif dari biaya generator. Dalam GAN, kita dapat mendefinisikan fungsi nilai yang harus diminimalkan oleh generator dan harus dimaksimalkan oleh diskriminasi. Proses pelatihan untuk model generatif sangat berbeda dengan metode pelatihan terbimbing. GAN sensitif terhadap bobot awal. Jadi kita perlu menggunakan normalisasi batch. Batch normalisasi membuat model stabil, selain meningkatkan performa. Di sini, kami melatih dua model, model generatif dan model diskriminatif secara bersamaan. Model generatif G menangkap distribusi data dan model diskriminatif D memperkirakan probabilitas sampel yang berasal dari data pelatihan daripada G.

GAN - contoh kode

Dalam contoh berikut, kami membuat dan melatih model GAN menggunakan kumpulan data MNIST dan menggunakan TensorFlow. Di sini, kami akan menggunakan versi khusus dari fungsi aktivasi ULT yang dikenal sebagai **Leaky ReLU**. Keluarannya adalah jenis digit tulisan tangan baru:

*Leaky ReLU adalah variasi dari fungsi aktivasi ULT yang diberikan dengan rumus $f(x) = \max(\alpha * x, x)$. Jadi keluaran untuk nilai negatif untuk x adalah $\alpha * x$ dan keluaran untuk x positif adalah x .*

```
#import semua pustaka yang diperlukan dan muat kumpulan data
%matplotlib inline

import pickle sebagai pkl
import numpy as np
import tensorflow sebagai tf
import matplotlib.pyplot sebagai plt

dari tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data')
```

Untuk membangun jaringan ini, kita membutuhkan dua input, satu untuk generator dan satu lagi untuk diskriminasi. Dalam kode berikut, kami membuat placeholder untuk `real_input` untuk diskriminasi dan `z_input` untuk generator, dengan ukuran input masing-masing sebagai `dim_real` dan `dim_z`:

```
#place holder untuk input model
def model_inputs(dim_real, dim_z):
    real_input = tf.placeholder(tf.float32, name = 'dim_real')
    z_input = tf.placeholder(tf.float32, name = 'dim_z')

    return real_input, z_input
```

Di sini, input `z` adalah vektor acak ke generator yang mengubah vektor ini menjadi gambar. Kemudian kami menambahkan lapisan tersembunyi, yang merupakan lapisan ULT bocor, untuk memungkinkan gradien mengalir mundur. Leaky ULT sama seperti

ULT normal (untuk nilai negatif yang memancarkan nol) kecuali bahwa ada keluaran kecil bukan nol untuk nilai masukan negatif. Generator bekerja lebih baik dengan `tanh` fungsi tersebut. Keluaran generator adalah `tanh` keluaran. Jadi, kita harus mengubah skala gambar MNIST menjadi antara -1 dan 1, bukan 0 dan 1. Dengan pengetahuan ini, kita dapat membangun jaringan generator:

```
#Kode berikut membangun generator
def Jaringan_Generator (z, out_dim, n_units = 128, reuse = False, alpha = 0.01):
    '''Bangun jaringan generator.

    Argumen
    -----
    z: Tensor masukan untuk generator
    out_dim: Bentuk keluaran generator
    n_units: Jumlah unit dalam lapisan tersembunyi yang digunakan kembali: Gunakan kembali variabel dengan tf.variable_scope
    alpha: parameter kebocoran untuk ReLU yang bocor

    kembali
    -----
    out:
    ...
    dengan tf.variable_scope (' generator ', reuse = reuse) sebagai generator_scope: # selesaikan ini
        # Hidden layer
        h1 = tf.layers.dense (z, n_units, activation = None)
        # Leaky ReLU
        h1 = tf.nn.leaky_relu (h1, alpha = alpha, name = 'leaky_generator')

        # Log dan
        logit keluaran tanh = tf.layers.dense (h1, out_dim, activation = None)
        out = tf.tanh ( logits )

        kembali
    ...
```

Jaringan diskriminasi sama dengan generator kecuali bahwa lapisan keluaran adalah `sigmoid` fungsi:

```
def diskriminator (x, n_units = 128, reuse = False, alpha = 0.01):
    '''Bangun jaringan diskriminasi.

    Argumen
    -----
    x: tensor Masukan untuk diskriminasi
    n_units: Jumlah unit pada lapisan tersembunyi
    reuse: Reuse variabel dengan tf.variable_scope
    alpha: parameter kebocoran untuk bocor ReLU

    Pengembalian
    -----
    keluar , logits:
    ...
    dengan tf.variable_scope (' diskriminator ', reuse = reuse) sebagai diskriminator_scope: # selesaikan ini
        # Lapisan tersembunyi
        h1 = tf.layers.dense (x, n_units, activation = None)
        # Leaky ReLU
        h1 = tf.nn.leaky_relu (h1, alpha = alpha, name = 'leaky_discriminator')

        logits = tf.layers.dense (h1, 1, activation = None)
        out = tf.sigmoid (logits)

        kembali keluar, logits
```

Untuk membangun jaringan, gunakan kode berikut:

```
#Hyperparameters
# Ukuran gambar input ke diskriminasi
input_size = 784 # 28x28 Gambar MNIST diratakan
# Ukuran vektor laten ke generator
```

```

z_size = 100
# Ukuran lapisan tersembunyi di generator dan diskriminat
g_hidden_size = 128
d_hidden_size = 128
# Faktor kebocoran untuk kebocoran ReLU
alpha = 0.01
# Label smoothing
smooth = 0.1

```

Kami ingin berbagi bobot antara data asli dan palsu, jadi kami perlu menggunakan kembali variabel:

```

#Bangun jaringan
tf.reset_default_graph ()
# Buat placeholder masukan kita
input_real, input_z = model_inputs (input_size, z_size)

# Bangun model
g_model = generator (input_z, input_size, n_units = g_hidden_size, alpha = alpha)
# g_model adalah keluaran generator

d_model_real, d_logits_real = diskriminat (input_real, n_units = d_hidden_size, alpha = alpha)
d_model_fake, d_logits_fake = diskriminat (g_model, reuse = True, n_units = d_hidden_size, alpha = alpha)

```

Menghitung kerugian

Bagi diskriminat, kerugian total adalah jumlah kerugian gambar asli dan palsu. Kerugiannya akan menjadi cross-entropyies sigmoid, yang bisa kita peroleh menggunakan TensorFlow `tf.nn.sigmoid_cross_entropy_with_logits`. Kemudian kami menghitung mean untuk semua gambar dalam batch. Jadi kerugiannya akan terlihat seperti ini:

```
| tf.reduce_mean (tf.nn.sigmoid_cross_entropy_with_logits (logits = logits, label = label))
```

Untuk membantu diskriminat menggeneralisasi dengan lebih baik, nilai `label`s dapat dikurangi sedikit dari 1,0 menjadi 0,9, misalnya dengan menggunakan parameter `smooth`. Ini dikenal sebagai **penghalusan label**, dan biasanya digunakan dengan pengklasifikasi untuk meningkatkan kinerja. Kerugian diskriminat untuk data palsu serupa. The `logits` adalah `d_logits_fake`, yang kami dapatkan dari melewati output generator untuk discriminator. Palsu `logits` ini digunakan dengan `label`s semua nol. Ingatlah bahwa kita ingin diskriminat mengeluarkan 1 untuk gambar asli dan 0 untuk gambar palsu, jadi kita perlu menyiapkan kerugian untuk mencerminkannya.

Akhirnya, kerugian generator menggunakan `d_logits_fake`, gambar palsu `logits`. Tapi sekarang `label`s semuanya 1s. Generator mencoba mengelabui diskriminat, jadi ia ingin diskriminat mengeluarkan gambar palsu:

```

Kerugian # Hitung
d_loss_real = tf.reduce_mean (
    tf.nn.sigmoid_cross_entropy_with_logits (logits = d_logits_real,
                                             label = tf.ones_like (d_logits_real) * (1
- halus)))
d_loss_fake = tf.reduce_mean (
    tf.nn.sigmoid_cross_entropy_with_logits (logits = d_logits_fake ,
                                             label = tf.zeros_like (d_logits_real)))
d_loss = d_loss_real + d_loss_fake
g_loss = tf.reduce_mean (

```

```
tf.nn.sigmoid_cross_entropy_with_logits (logits = d_logits_fake,
                                         labels = tf.ones_like (dfake))
```

Menambahkan pengoptimal

Kita perlu memperbarui variabel generator dan diskriminator secara terpisah. Jadi, pertama-tama dapatkan semua variabel grafik dan kemudian, seperti yang telah kita jelaskan sebelumnya, kita hanya bisa mendapatkan variabel generator dari lingkup generator dan, demikian pula, variabel diskriminator dari lingkup diskriminator:

```
# Pengoptimal
learning_rate = 0.002

# Dapatkan trainable_variables, bagi menjadi bagian G dan D
t_vars = tf.trainable_variables ()
g_vars = [var for var in t_vars if var.name.startswith ('generator')]
d_vars = [var for var in t_vars if var.name.startswith ('diskriminator')]

d_train_opt = tf.train.AdamOptimizer (learning_rate). minimalkan (d_loss, var_list = d_vars)
g_train_opt = tf.train.AdamOptimizer (learning_rate). )
```

Untuk melatih jaringan, gunakan:

```
batch_size = 100
periode = 100
sampel = []
kerugian = []
# Hanya simpan variabel generator
saver = tf.train.Saver (var_list = g_vars)
dengan tf.Session () sebagai sess:
    sess.run (tf.global_variables_initializer ())
    untuk e dalam range (epochs):
        untuk ii dalam range (mnist.train.num_examples // batch_size):
            batch = mnist.train.next_batch (batch_size)

            # Dapatkan gambar, bentuk ulang dan skala ulang untuk diteruskan ke D
            batch_images = batch [0] .reshape ((batch_size, 784))
            batch_images = batch_images * 2 - 1

            # Contoh derau acak untuk G
            batch_z = np.random.uniform (-1, 1, size = (batch_size, z_size))

            # Jalankan pengoptimal
            _ = sess.run (d_train_opt, feed_dict = {input_real: batch_images, input_z: batch_z})
            _ = sess.run (g_train_opt, feed_dict = {input_z: batch_z})

            # Di akhir setiap epoch, dapatkan kerugian dan cetak
            train_loss_d = sess.run (d_loss, {input_z: batch_z, input_real: batch_images})
            train_loss_g = g_loss.eval ({input_z: batch_z})

            print ("Epoch {} / {} ...". format (e + 1, epochs),
                  "Discriminator Loss: {:.4f} ...". format (train_loss_d),
                  "Generator Loss: {:.4f}...". format (train_loss_g))
            # Simpan kerugian untuk dilihat setelah
            kerugian pelatihan .append ((kerugian_kereta_kerja, kerugian_kerja_g))

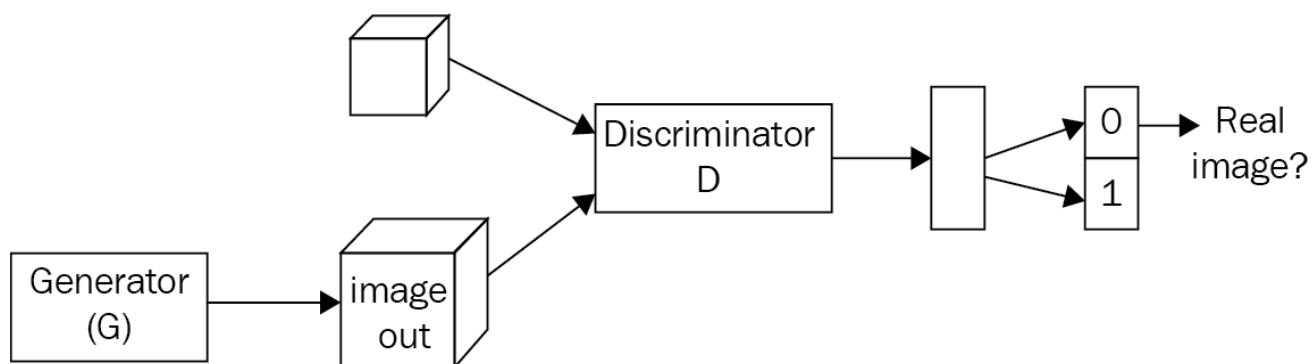
            # Sampel dari generator saat kita berlatih untuk melihat setelahnya
            sample_z = np.random.uniform (-1, 1, size = (16, z_size))
            gen_samples = sess.run (
                generator (input_z, input_size, n_units = g_hidden_size, reuse = True, alpha
= alpha),
                feed_dict = {input_z: sample_z})
            samples.append (gen_samples)
            saver.save (sess, './checkpoints/generator.ckpt')

            # Simpan sampel generator pelatihan
            dengan open ('train_samples.pkl ','wb ') sebagai f:
                pkl.dump (sampel, f)
```

Setelah model dilatih dan disimpan, Anda dapat memvisualisasikan digit yang dihasilkan (kode tidak ada di sini, tetapi dapat diunduh).

Pembelajaran semi-supervisi dan GAN

Untuk itu, kita telah melihat bagaimana GAN dapat digunakan untuk menghasilkan gambar yang realistik. Di bagian ini, kita akan melihat bagaimana GAN dapat digunakan untuk tugas klasifikasi di mana kita memiliki lebih sedikit data berlabel tetapi masih ingin meningkatkan akurasi pengklasifikasi. Di sini kami juga akan menggunakan **Nomor Rumah Street View** atau **kumpulan data SVHN yang sama** untuk mengklasifikasikan gambar. Seperti sebelumnya, disini kita juga memiliki dua jaringan yaitu generator G dan diskriminator D. Dalam hal ini diskriminator dilatih untuk menjadi pengklasifikasi. Perubahan lainnya adalah bahwa keluaran diskriminator beralih ke fungsi softmax alih-alih sigmoid fungsi, seperti yang terlihat sebelumnya. Fungsi softmax mengembalikan distribusi probabilitas atas label:



GAN: Image classification

Sekarang kami memodelkan jaringan sebagai:

$$\text{biaya total} = \text{biaya data berlabel} + \text{biaya data tak berlabel}$$

Untuk mendapatkan biaya data berlabel, kita dapat menggunakan `cross_entropy` fungsi:

```
| biaya data berlabel = cross_entropy (logits, label)
| biaya data tidak berlabel = cross_entropy (logits, real)
```

Kemudian kita dapat menghitung jumlah semua kelas:

```
| real_prob = jumlah (softmax (real_classes))
```

Pengklasifikasi normal bekerja pada data berlabel. Namun, pengklasifikasi berbasis GAN semi-supervised berfungsi pada data berlabel, data nyata tanpa label, dan gambar palsu. Ini bekerja dengan sangat baik, yaitu, kesalahan klasifikasi lebih sedikit meskipun kami memiliki lebih sedikit data berlabel dalam proses pelatihan.

Pencocokan fitur

Ide pencocokan fitur adalah menambahkan variabel tambahan ke fungsi biaya generator untuk menghukum perbedaan antara kesalahan absolut dalam data pengujian dan data pelatihan.

Klasifikasi semi-terbimbing menggunakan contoh GAN

Pada bagian ini, kami menjelaskan cara menggunakan GAN untuk membangun pengklasifikasi dengan pendekatan pembelajaran semi-supervised.

Dalam pembelajaran yang diawasi, kami memiliki satu set pelatihan input dan label kelas . Kami melatih model yang menerima masukan dan memberikan sebagai keluaran. $x \ y \ x \ y$

Dalam pembelajaran semi-supervised, tujuan kami masih melatih model yang mengambil input dan menghasilkan output. Namun, tidak semua contoh pelatihan kami memiliki label . $x \ y \ y$

Kami menggunakan dataset SVHN. Kami akan mengubah diskriminator GAN menjadi 11 kelas diskriminator (0 hingga 9 dan satu label untuk gambar palsu). Ini akan mengenali 10 kelas berbeda dari digit SVHN nyata, serta kelas kesebelas gambar palsu yang berasal dari generator. Diskriminator akan mendapatkan pelatihan tentang gambar berlabel nyata, gambar nyata tanpa label, dan gambar palsu. Dengan menggunakan tiga sumber data, bukan hanya satu, ini akan menggeneralisasi set pengujian jauh lebih baik daripada pengklasifikasi tradisional yang dilatih hanya pada satu sumber data:

```
def model_inputs (real_dim, z_dim):
    input_real = tf.placeholder (tf.float32, (None, * real_dim), name = 'input_real')
    input_z = tf.placeholder (tf.float32, (None, z_dim), name = ' input_z ')
    y = tf.placeholder (tf.int32, (None), name = ' y ')
    label_mask = tf.placeholder (tf.int32, (None), name = ' label_mask ')

    return input_real, input_z, y, label_mask
```

Tambahkan generator:

```
def generator (z, output_dim, reuse = False, alpha = 0.2, training = True, size_mult = 128):
    dengan tf.variable_scope ('generator', reuse = reuse):
        # Lapisan pertama yang terhubung sepenuhnya
        x1 = tf.layers.dense (z, 4 * 4 * size_mult * 4)
        # Bentuk kembali untuk memulai susunan konvolusional
        x1 = tf.reshape (x1, (-1, 4, 4, size_mult * 4))
        x1 = tf.layers.batch_normalization (x1, training = training)
        x1 = tf.maximum (alpha * x1, x1)

        x2 = tf.layers.conv2d_transpose (x1, size_mult * 2, 5, step = 2, padding = 'same')
        x2 = tf.layers.batch_normalization ( x2, pelatihan = pelatihan)
        x2 = tf.maksimal (alfa * x2, x2)
```

```

x3 = tf.layers.conv2d_transpose (x2, size_mult, 5, langkah = 2, padding = 'same')
x3 = tf.layers.batch_normalization (x3, training = training)
x3 = tf.maximum (alpha * x3, x3)

# Output layer
logits = tf.layers.conv2d_transpose (x3, output_dim, 5, strides = 2, padding = 'same')

out = tf.tanh (logits)

kembali keluar

```

Tambahkan diskriminator:

```

def diskriminator (x, reuse = False, alpha = 0.2, drop_rate = 0., num_classes = 10, size_mult =
64):
    dengan tf.variable_scope ('diskriminator', reuse = reuse):
        x = tf.layers.dropout (x , rate = drop_rate / 2.5)

        # Input layer adalah 32x32x3
        x1 = tf.layers.conv2d (x, size_mult, 3, strides = 2, padding = 'same')
        relu1 = tf.maximum (alpha * x1, x1)
        relu1 = tf.layers.dropout (relu1, rate = drop_rate)

        x2 = tf.layers.conv2d (relu1, size_mult, 3, strides = 2, padding = 'same')
        bn2 = tf.layers.batch_normalization (x2, training = True)
        relu2 = tf.maximum (alpha * x2, x2)

        x3 = tf.layers.conv2d (relu2, size_mult, 3, langkah = 2, padding = 'sama')
        bn3 = tf.layers.batch_normalization (x3, training = True)
        relu3 = tf.maximum (alpha * bn3, bn3)
        relu3 = tf.layers.dropout (relu3, rate = drop_rate)

        x4 = tf.layers.conv2d (relu3, 2 * size_mult, 3, strides = 1, padding = 'same')
        bn4 = tf.layers.batch_normalization (x4, training = True)
        relu4 = tf.maximum (alpha * bn4, bn4)

        x5 = tf.layers.conv2d ( relu4, 2 * size_mult, 3, strides = 1, padding = 'same')
        bn5 = tf.layers.batch_normalization (x5, training = True)
        relu5 = tf.maximum (alpha * bn5, bn5)

        x6 = tf.layers.conv2d (relu5, 2 * size_mult, 3, langkah = 2, padding = 'same')
        bn6 = tf.layers.batch_normalization (x6, training = True)
        relu6 = tf.maximum (alpha * bn6, bn6)
        relu6 = tf.layers.dropout (relu6, rate = drop_rate)

        x7 = tf.layers.conv2d (relu6, 2 * size_mult, 3, strides = 1, padding = 'valid ')
        # Jangan gunakan bn pada layer ini, karena bn akan menyetel mean dari setiap fitur
        # ke parameter bn mu.
        # Lapisan ini digunakan untuk kehilangan pencocokan fitur, yang hanya berfungsi jika
        # artinya bisa berbeda ketika diskriminator dijalankan pada data daripada
        # saat diskriminator dijalankan pada sampel generator.
        relu7 = tf.maximum (alpha * x7, x7)

        # Ratakan dengan
        fitur penggabungan rata-rata global = naikkan NotImplementedError ()

        # Setel class_logits menjadi input ke distribusi softmax di kelas yang berbeda
        naikkan NotImplementedError ()

        # Set gan_logits sedemikian rupa sehingga P (input real | input) = sigmoid (gan_logits).
        # Ingatlah bahwa class_logits memberi Anda distribusi probabilitas atas semua
        # kelas # nyata dan kelas palsu. Anda perlu mengetahui cara mengubah
        # distribusi multikelas softmax # ini menjadi keputusan biner nyata vs palsu yang dapat
        # dijelaskan dengan sigmoid.
        # Stabilitas numerik sangat penting.
        # Anda mungkin perlu menggunakan trik stabilitas numerik ini:
        # log sum_i exp a_i = m + log sum_i exp (a_i - m).
        # Ini stabil secara numerik ketika m = max_i a_i.
        # (Ini membantu untuk memikirkan apa yang salah ketika ...)
        # 1. Satu nilai a_i sangat besar
        # 2. Semua nilai a_i sangat negatif
        # Trik ini dan nilai m ini memperbaiki kedua kasus tersebut, tetapi implementasi naif dan
        # nilai lain m menghadapi berbagai masalah)
        meningkatkan NotImplementedError ()

return out, class_logits, gan_logits, features

```

Hitung kerugiannya:

```
def model_loss (input_real, input_z, output_dim, y, num_classes, label_mask, alpha = 0.2, drop_rate = 0.):
    """
        Dapatkan kerugian untuk diskriminator dan generator
        : param input_real: Gambar dari set data nyata : param input_z: Z input
        : param output_dim: Jumlah saluran pada gambar keluaran
        : param y: Label kelas
        bilangan bulat: param num_classes: Jumlah kelas
        : param alpha: Kemiringan separuh kiri aktivasi ULT yang bocor
        : param drop_rate: Probabilitas penurunan unit tersembunyi
        : return: Sebuah tuple (kerugian diskriminator, kerugian generator)
    """

    # Angka-angka ini mengalikan ukuran setiap lapisan generator dan diskriminator,
    # masing-masing. Anda dapat menguranginya untuk menjalankan kode Anda lebih cepat untuk tujuan
    debugging.
    g_size_mult = 32
    d_size_mult = 64

    # Di sini kita menjalankan generator dan diskriminator
    g_model = Generator (input_z, output_dim, alpha = alpha, size_mult = g_size_mult)
    d_on_data = discriminator (input_real, alpha = alpha, drop_rate = drop_rate, size_mult = d_size_mult)
    d_model_real , class_logits_on_data, gan_logits_on_data, data_features = d_on_data
    d_on_samples = discriminator (g_model, reuse = True, alpha = alpha, drop_rate = drop_rate,
size_mult = d_size_mult)
    d_model_fake, class_logits_on_samples, gan_logits_on_samples, sample_features = d_on_samples

    # Di sini kita menghitung `d_loss`, kerugian untuk diskriminator.
    # Ini harus menggabungkan dua kerugian yang berbeda:
    # 1. Kerugian untuk masalah GAN, di mana kita meminimalkan cross-entropy untuk
    masalah klasifikasi biner # real-vs-palsu.
    # 2. Kerugian untuk masalah klasifikasi digit SVHN, dimana kita meminimalkan cross-entropy
    # untuk softmax multi-kelas. Untuk yang satu ini kami menggunakan label. Jangan lupa untuk
    mengabaikan
    # use `label_mask` untuk mengabaikan contoh yang kita anggap tidak berlabel untuk
    # masalah pembelajaran semi-supervised.
    menaikkan NotImplementedError ()

    # Di sini kita menyetel `g_loss` ke kerugian" pencocokan fitur "yang ditemukan oleh Tim
    Salimans di OpenAI.
    # Kerugian ini terdiri dari meminimalkan perbedaan mutlak antara fitur yang diharapkan
    # pada data dan fitur yang diharapkan pada sampel yang dihasilkan.
    # Kerugian ini bekerja lebih baik untuk pembelajaran semi-supervisi daripada kerugian GAN
    tradisi.
    menaikkan NotImplementedError ()

    pred_class = tf.cast (tf.argmax (class_logits_on_data, 1), tf.int32)
    eq = tf.equal (tf.squeeze (y), pred_class)
    benar = tf.reduce_sum (tf.to_float (eq) )
    masked_correct = tf.reduce_sum (label_mask * tf.to_float (eq))

    kembalikan d_loss, g_loss, benar, masked_correct, g_model
```

Tambahkan pengoptimal:

```
def model_opt (d_loss, g_loss, learning_rate, beta1):
    """
        Dapatkan operasi pengoptimalan
        : param d_loss: Kerugian diskriminator Tensor
        : param g_loss: Generator loss Tensor
        : param learning_rate: Learning Rate Placeholder
        : param beta1: Tingkat peluruhan eksponensial untuk yang pertama Momen dalam pengoptimal
        : return: Sebuah tupel (operasi pelatihan diskriminator, operasi pelatihan generator)
    """

    # Dapatkan bobot dan bias untuk diperbarui. Dapatkan mereka secara terpisah untuk diskriminator
    dan generator
    menaikkan NotImplementedError ()

    # Minimalkan biaya kedua pemain secara bersamaan
    naikkan NotImplementedError ()
    shrink_lr = tf.assign (learning_rate, learning_rate * 0.
```

```
kembalikan d_train_opt, g_train_opt, shrink_lr
```

Bangun model jaringan:

```
class GAN:  
    """  
    Model GAN  
    .: param real_size: Bentuk data nyata  
    .: param z_size: Jumlah entri dalam vektor kode z  
    .: param learnin_rate: Kecepatan pembelajaran yang digunakan untuk Adam  
    .: param num_classes : Jumlah kelas yang harus dikenali  
    .: Param alpha: Kemiringan paruh kiri aktivasi ULT yang bocor  
    : param beta1: Parameter beta1 untuk Adam.  
    """  
    Def __init __ (self, real_size, z_size, learning_rate, num_classes = 10 , alpha = 0.2, beta1 = 0.5):  
        tf.reset_default_graph ()  
  
        self.learning_rate = tf. Variabel (learning_rate, trainable = False)  
        input = model_inputs (real_size, z_size)  
        self.input_real, self.input_z, self.y, self.label_mask = input  
        self.drop_rate = tf.placeholder_with_default (.5, (), "drop_rate")  
  
        loss_results = model_loss (self.input_real, self.input_z,  
                                real_size [2 ], self.y, num_classes,  
                                label_mask = self.label_mask,  
                                alpha = 0.2,  
                                drop_rate = self.drop_rate)  
        self.d_loss, self.g_loss, self.correct, self.masked_correct, self.samples = loss_results  
  
        self.d_opt, self.g_opt, self.shrink_lr = model_opt (self.d_loss, self.g_loss,  
self.learning_rate, beta1)
```

Latih dan pertahankan model:

```
def train (net, dataset, epochs, batch_size, figsize = (5,5)):  
  
    saver = tf.train.Saver ()  
    sample_z = np.random.normal (0, 1, size = (50, z_size))  
  
    sampel , train_accuracies, test_accuracies = [], [], []  
    steps = 0  
  
    dengan tf.Session () as sess:  
        sess.run (tf.global_variables_initializer ())  
        untuk e dalam range (epochs):  
            print ("Epoch", e )  
  
            t1e = time.time ()  
            num_examples = 0  
            num_correct = 0  
            untuk x, y, label_mask dalam dataset.batches (batch_size):  
                tegaskan 'int' dalam langkah str (y.dtype)  
                + = 1  
                num_examples + = label_mask.sum ( )  
  
                # Contoh gangguan acak untuk G  
                batch_z = np.random.normal (0, 1, size = (batch_size, z_size))  
  
                # Jalankan pengoptimal  
                t1 = time.time ()  
                _, _, correct = sess.run ([net.d_opt , net.g_opt, net.masked_correct],  
                                         feed_dict = {net.input_real: x, net.input_z: batch_z,  
                                         net.y: y, net.label_mask: label_mask})  
                t2 = time.time ()  
                num_correct + = true  
  
                sess .run ([net.shrink_lr])  
  
                train_accuracy = num_correct / float (num_examples)  
                print ("\ t \ tAkurasi kereta Classifier:", train_accuracy)  
                num_examples = 0
```

```

num_correct = 0
untuk x, y dalam dataset.batches (batch_size, which_set = "test"):
    tegaskan 'int' dalam str (y.dtype)
    num_examples += x.shape [0]

    benar, = sess.run ([net.correct], feed_dict = {net.input_real: x,
                                                    net.y: y,
                                                    net.drop_rate: 0.})
    num_correct += benar

test_accuracy = num_correct / float (num_examples)
print ("\t\t\tClassifier test akurasi ", akurasi_ uji)
cetak ("\t\t\tWaktu langkah: ", t2 - t1)
t2e = waktu.waktu ()
print ("\t\t\tEpoch time:", t2e - t1e)

gen_samples = sess.run (
    net.samples,
    feed_dict = {net.input_z: sample_z})
samples.append (gen_samples)
_= view_samples (-1, sampel, 5, 10, figsize = figsize)
plt.show ()

# Simpan riwayat akurasi untuk dilihat setelah pelatihan
train_accuracies.append (train_accuracy)
test_accuracies.append (test_accuracy)

saver.save (sess, './checkpoints/generator.ckpt')

dengan open ('samples.pkl', 'wb') sebagai f:
pkl.dump (sample, f)

return train_accuracies, test_accuracies, sample

```

GAN konvolusional yang dalam

GAN konvolusional dalam, juga disebut **DCGAN**, digunakan untuk menghasilkan gambar berwarna. Di sini kami menggunakan lapisan konvolusional di generator dan diskriminator. Kita juga perlu menggunakan normalisasi batch agar GAN dilatih dengan benar. Kami akan membahas normalisasi batch secara rinci dalam bab peningkatan kinerja jaringan neural dalam. Kami akan melatih GAN pada set data SVHN; contoh kecil ditunjukkan pada gambar berikut. Setelah pelatihan, generator akan dapat membuat gambar yang hampir identik dengan gambar ini. Anda dapat mengunduh kode untuk contoh ini:



Tampilan nomor rumah Google Street View

Normalisasi batch

Normalisasi batch adalah teknik untuk meningkatkan kinerja dan stabilitas jaringan saraf. Idenya adalah untuk menormalkan input lapisan sehingga memiliki rata-rata nol dan varians 1. Normalisasi batch diperkenalkan dalam makalah 2015 Sergey Ioffe dan Christian Szegedy, *Normalisasi Batch Diperlukan untuk Membuat DCGAN Bekerja*. Idenya adalah bahwa alih-alih hanya menormalkan input ke jaringan, kami menormalkan input ke lapisan dalam jaringan. Ini disebut **normalisasi batch** karena selama pelatihan, kita menormalkan input setiap layer dengan menggunakan mean dan varians dari nilai-nilai dalam mini-batch saat ini.

Ringkasan

Dalam bab ini, kita telah melihat bagaimana model GAN benar-benar menampilkan kekuatan CNN. Kami belajar bagaimana melatih model generatif kami sendiri dan melihat contoh praktis GAN yang dapat menghasilkan foto dari lukisan dan mengubah kuda menjadi zebra .

Kami memahami bagaimana GAN berbeda dari model diskriminatif lainnya dan mempelajari mengapa model generatif lebih disukai.

Pada bab berikutnya, kita akan belajar tentang perbandingan perangkat lunak pembelajaran mendalam dari awal.

Mekanisme Perhatian untuk CNN dan Model Visual

Tidak semua dalam gambar atau teks — atau secara umum, data apa pun — sama-sama relevan dari perspektif wawasan yang perlu kita tarik darinya. Misalnya, pertimbangkan tugas di mana kami mencoba memprediksi kata berikutnya dalam urutan pernyataan verbose seperti *Alice dan Alya adalah teman. Alice tinggal di Prancis dan bekerja di Paris. Alya adalah orang Inggris dan bekerja di London. Alice lebih suka membeli buku yang ditulis dalam bahasa Prancis, sedangkan Alya lebih memilih buku dalam _____.*

Ketika contoh ini diberikan kepada manusia, bahkan seorang anak dengan kemampuan bahasa yang baik dapat memprediksi dengan baik kata berikutnya kemungkinan besar adalah *bahasa Inggris*. Secara matematis, dan dalam konteks pembelajaran yang mendalam, hal ini dapat dipastikan dengan cara yang sama dengan membuat penyematan vektor dari kata-kata ini dan kemudian menghitung hasilnya menggunakan matematika vektor, sebagai berikut:

$$V(\overrightarrow{Word}) = V(\overrightarrow{French}) - V(\overrightarrow{Paris}) + V(\overrightarrow{London})$$

Di sini, $V(Word)$ adalah vektor yang disematkan untuk kata yang diperlukan; demikian pula, $V(Prancis)$, $V(Paris)$, dan $V(London)$ adalah embeddings vektor yang diperlukan untuk kata *Prancis*, *Paris*, dan *London*.

Embeddings adalah (sering) representasi vektor berdimensi lebih rendah dan padat (numerik) dari input atau indeks input (untuk data non-numerik); dalam hal ini, teks.

Algoritme seperti word2vec dan glove dapat digunakan untuk mendapatkan embeddings kata. Varian yang telah dilatih sebelumnya dari model ini untuk teks umum tersedia di library NLP berbasis Python yang populer, seperti SpaCy, Gensim, dan lainnya juga dapat dilatih menggunakan sebagian besar library deep learning, seperti Keras, TensorFlow, dan sebagainya.

Konsep embeddings sama relevannya dengan visi dan gambar, maupun teks.

Mungkin tidak ada vektor yang sama persis dengan vektor yang baru saja kita peroleh dalam bentuk $V(\overrightarrow{Word})$; tetapi jika kita mencoba menemukan yang paling dekat dengan yang diperoleh $V(\overrightarrow{Word})$ yang ada dan menemukan kata yang mewakili menggunakan

pengindeksan terbalik, kata itu kemungkinan besar akan sama dengan yang kita pikirkan sebelumnya, yaitu, *bahasa Inggris* .

Algoritme seperti cosine similarity dapat digunakan untuk mendapatkan vektor yang paling mendekati vektor yang dihitung.

*Untuk implementasi, cara komputasi yang lebih efisien untuk menemukan vektor terdekat adalah **perkiraan tetangga terdekat (ANN)**, seperti yang tersedia di `annoy` pustaka Python .*

Meskipun kami telah membantu mendapatkan hasil yang sama, baik secara kognitif maupun melalui pendekatan pembelajaran mendalam, masukan dalam kedua kasus tersebut tidak sama. Kepada manusia, kami telah memberikan kalimat yang tepat untuk komputer, tetapi untuk aplikasi pembelajaran yang mendalam, kami telah memilih kata-kata yang benar (*Prancis* , *Paris* , dan *London*) dengan hati-hati dan posisi yang tepat dalam persamaan untuk mendapatkan hasil. Bayangkan bagaimana kita dapat dengan mudah menyadari kata-kata yang tepat untuk diperhatikan untuk memahami konteks yang benar, dan karenanya kita mendapatkan hasilnya; tetapi dalam bentuk saat ini, pendekatan deep learning kami tidak dapat melakukan hal yang sama.

Sekarang ada algoritma yang cukup canggih dalam pemodelan bahasa yang menggunakan varian dan arsitektur RNN yang berbeda, seperti LSTM dan Seq2Seq. Ini bisa menyelesaikan masalah ini dan mendapatkan solusi yang tepat, tetapi cara ini paling efektif dalam kalimat yang lebih pendek dan langsung, seperti *Paris ke Prancis dan London untuk _____* . Untuk memahami kalimat panjang dengan benar dan menghasilkan hasil yang benar, penting untuk memiliki mekanisme untuk mengajarkan arsitektur di mana kata-kata tertentu perlu lebih diperhatikan dalam urutan kata yang panjang. Ini disebut **mekanisme perhatian** dalam pembelajaran mendalam, dan ini berlaku untuk banyak jenis aplikasi pembelajaran dalam tetapi dengan cara yang sedikit berbeda.

*RNN adalah singkatan dari **jaringan saraf berulang** dan digunakan untuk menggambarkan urutan data temporal dalam pembelajaran mendalam. Karena masalah gradien menghilang, RNN jarang digunakan secara langsung; sebaliknya, varianya, seperti **LSTM (Long-Short Term Memory)** dan **GRU (Gated Recurrent Unit)** lebih populer dalam implementasi aktual.*

*Seq2Seq adalah singkatan dari **Sequence-to-Sequence** model dan terdiri dari dua jaringan RNN (atau varian) (oleh karena itu disebut **Seq2Seq** , di mana setiap jaringan RNN mewakili sebuah urutan); satu bertindak sebagai encoder dan yang lainnya sebagai decoder. Dua jaringan RNN dapat berupa jaringan RNN berlapis atau berlapis, dan keduanya terhubung melalui vektor pemikiran atau konteks. Selain itu, model Seq2Seq dapat menggunakan mekanisme*

perhatian untuk meningkatkan kinerja, terutama untuk urutan yang lebih panjang.

Padahal, untuk lebih tepatnya, kita pun harus mengolah informasi sebelumnya secara berlapis-lapis, pertama memahami bahwa kalimat terakhir adalah tentang Alya. Lalu kita bisa mengidentifikasi dan mengekstrak kota Alya, lalu kota Alice, dan seterusnya. Cara berpikir manusia yang berlapis seperti itu dianalogikan dengan penumpukan dalam pembelajaran yang mendalam, dan karenanya dalam aplikasi serupa, arsitektur bertumpuk cukup umum.

Untuk mengetahui lebih lanjut tentang cara kerja penumpukan dalam pembelajaran mendalam, terutama dengan arsitektur berbasis urutan, jelajahi topik seperti RNN bertumpuk dan jaringan perhatian bertumpuk.

Dalam bab ini, kami akan membahas topik-topik berikut:

- Mekanisme perhatian untuk pembuatan teks gambar
- Jenis perhatian (Perhatian Keras dan Lembut)
- Menggunakan perhatian untuk meningkatkan model visual
 - Model perhatian visual yang berulang

Mekanisme perhatian untuk pembuatan teks gambar

Dari pendahuluan, sejauh ini, harus jelas bagi Anda bahwa mekanisme perhatian bekerja pada urutan objek, menetapkan bobot setiap elemen dalam urutan untuk iterasi tertentu dari output yang diperlukan. Dengan setiap langkah berikutnya, tidak hanya urutan tetapi juga bobot dalam mekanisme perhatian dapat berubah . Jadi, arsitektur berbasis perhatian pada dasarnya adalah jaringan urutan, paling baik diimplementasikan dalam pembelajaran mendalam menggunakan RNN (atau variannya).

Pertanyaannya sekarang adalah: bagaimana kita menerapkan perhatian berbasis urutan pada gambar statis, terutama yang direpresentasikan dalam **jaringan saraf konvolisional (CNN)**? Baiklah, mari kita ambil contoh yang berada tepat di antara teks dan gambar untuk memahami ini. Asumsikan bahwa kita perlu memberi caption pada gambar sehubungan dengan isinya.

Kami memiliki beberapa gambar dengan teks yang disediakan oleh manusia sebagai data pelatihan dan menggunakan ini, kami perlu membuat sistem yang dapat memberikan teks yang layak untuk gambar baru yang tidak terlihat sebelumnya oleh model. Seperti yang terlihat sebelumnya, mari kita ambil contoh dan lihat bagaimana kita, sebagai manusia, akan memandang tugas ini dan proses serupa yang perlu

diterapkan dalam pembelajaran mendalam dan CNN. Mari pertimbangkan gambar berikut dan buat beberapa keterangan yang masuk akal untuk itu. Kami juga akan memeringkat mereka secara heuristik menggunakan penilaian manusia:

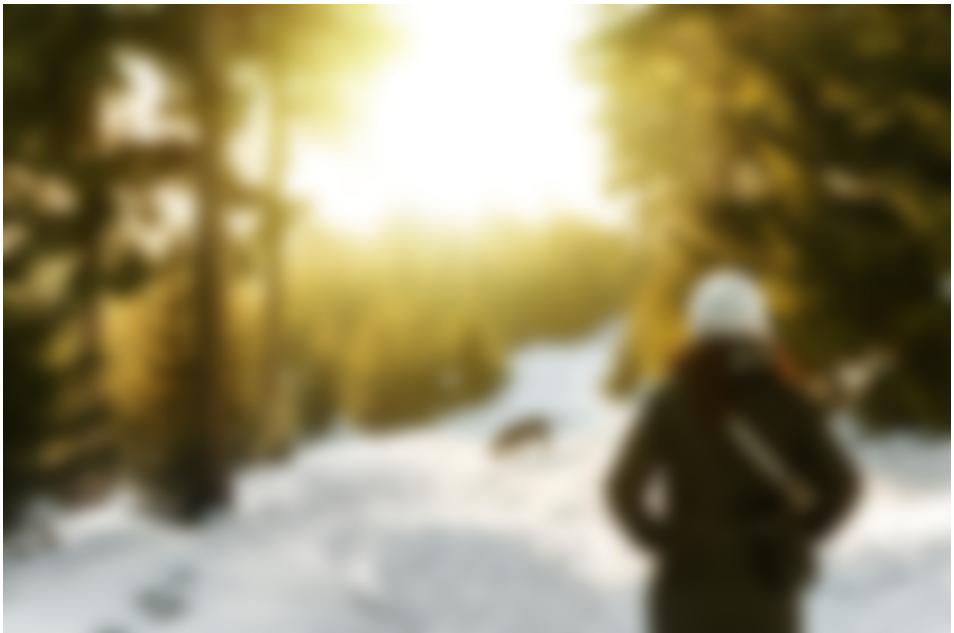


Beberapa teks yang mungkin (dalam urutan yang paling mungkin paling kecil kemungkinannya) adalah:

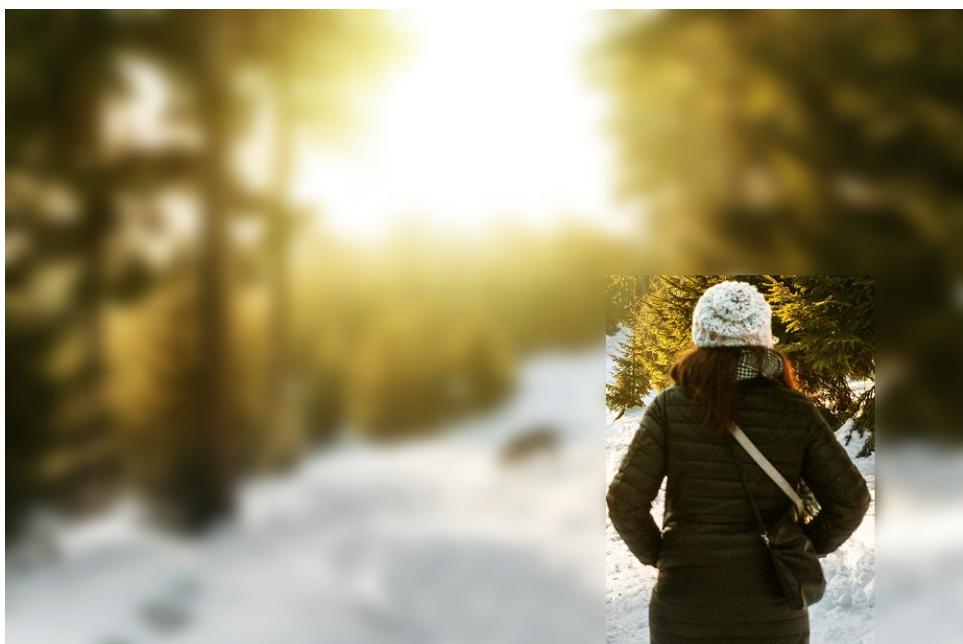
- Wanita melihat anjing di hutan salju
- Anjing coklat di salju
- Seseorang yang memakai topi di hutan dan tanah putih
- Anjing, pohon, salju, orang, dan sinar matahari

Hal penting yang perlu diperhatikan di sini adalah, terlepas dari kenyataan bahwa wanita adalah pusat gambar dan anjing bukanlah objek terbesar dalam gambar, teks yang kami cari kemungkinan besar berfokus pada mereka dan kemudian lingkungan mereka di sini. Ini karena kami menganggap mereka sebagai entitas penting di sini (tanpa konteks sebelumnya). Jadi sebagai manusia, cara kami mencapai kesimpulan ini adalah sebagai berikut: pertama-tama kami melihat sekilas keseluruhan gambar, lalu kami fokus pada wanita, dalam resolusi tinggi, sambil meletakkan semuanya di latar belakang (asumsikan efek **Bokeh** di ponsel kamera ganda). Kami mengidentifikasi bagian teks untuk itu, dan kemudian anjing dalam resolusi tinggi sambil meletakkan yang lainnya dalam resolusi rendah; dan kami menambahkan bagian caption. Akhirnya, kami melakukan hal yang sama untuk lingkungan sekitar dan bagian teks untuk itu.

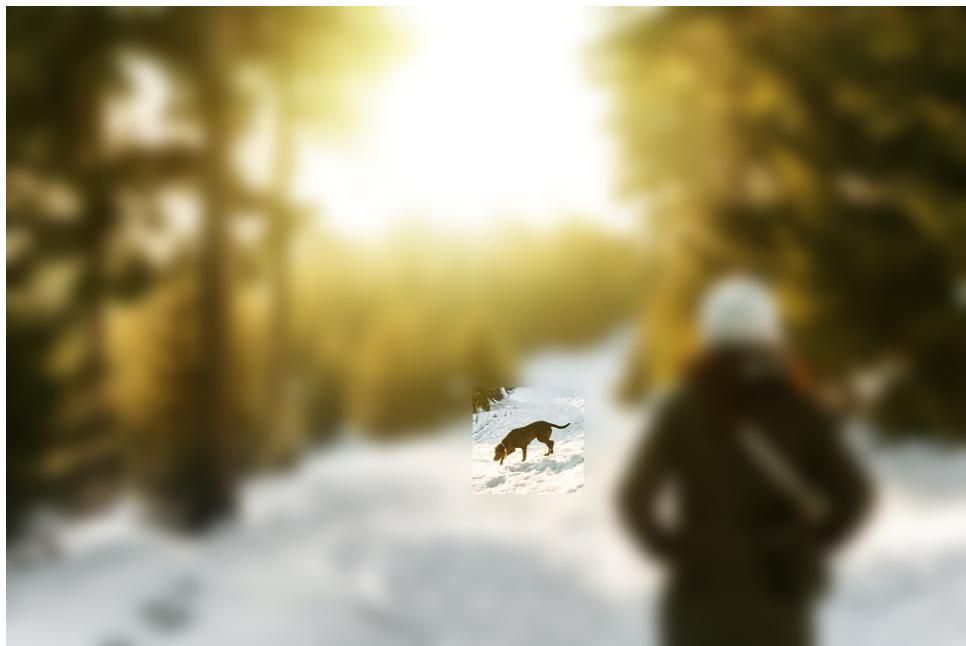
Jadi intinya, kami melihatnya dalam urutan ini untuk mencapai teks pertama:



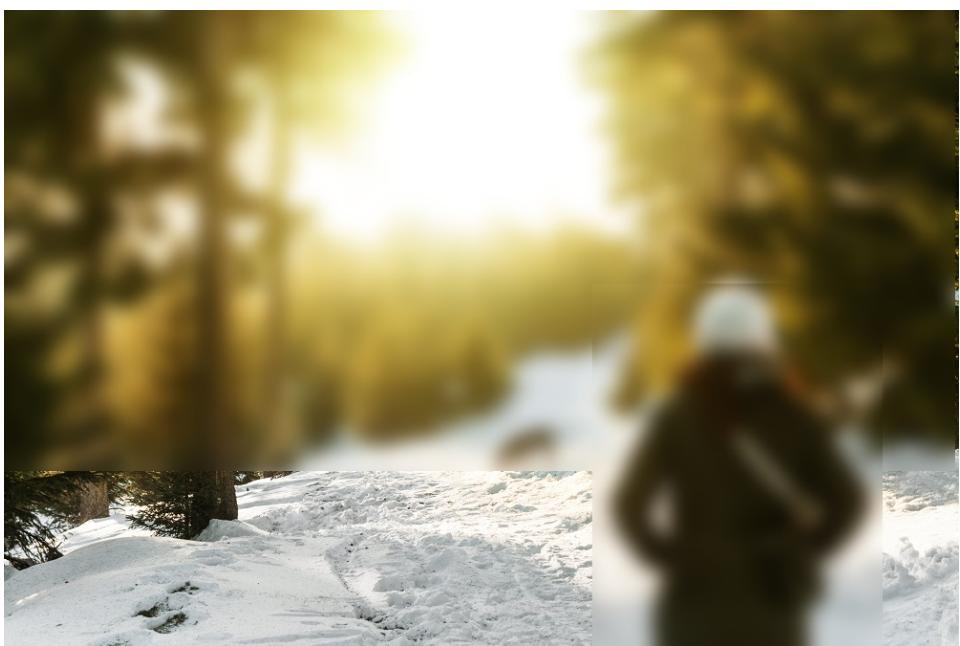
Gambar 1: Lihat sekilas gambarnya terlebih dahulu



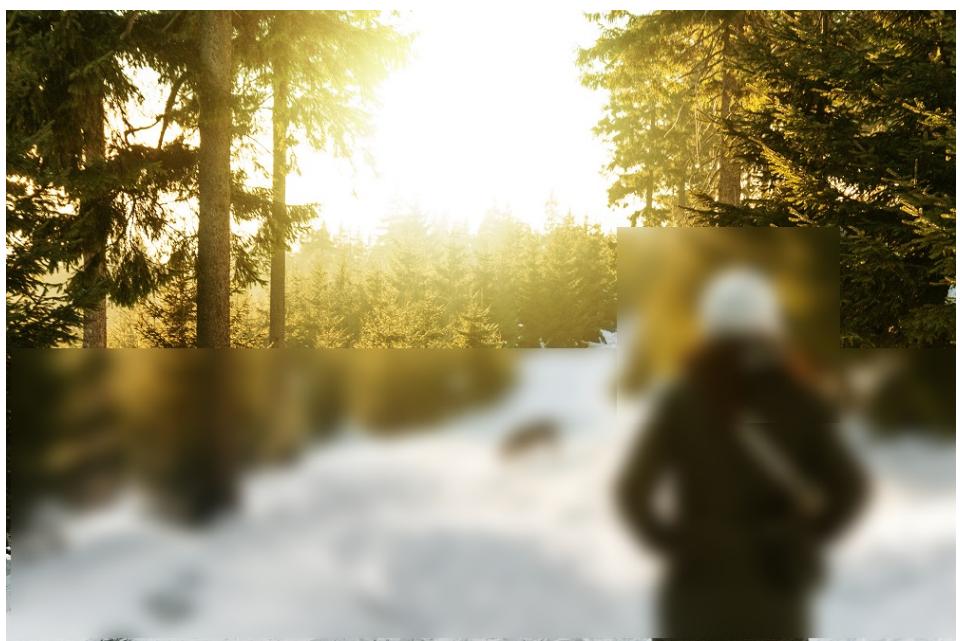
Gambar 2: Fokus pada wanita



Gambar 3: Fokus pada anjing



Gambar 4: Fokus pada salju



Dalam hal bobot perhatian atau fokus, setelah melirik gambar, kami fokus pada objek terpenting pertama: wanita di sini. Ini analog dengan membuat bingkai mental di mana kami meletakkan bagian gambar dengan wanita dalam resolusi tinggi dan bagian gambar lainnya dalam resolusi rendah.

Dalam referensi pembelajaran yang mendalam, urutan perhatian akan memiliki bobot tertinggi untuk vektor (embedding) yang mewakili konsep wanita untuk bagian urutan ini. Pada langkah selanjutnya dari keluaran / urutan, bobot akan bergeser lebih ke arah representasi vektor untuk anjing dan seterusnya.

Untuk memahami ini secara intuitif, kami mengubah gambar yang direpresentasikan dalam bentuk CNN menjadi vektor yang diratakan atau beberapa struktur serupa lainnya; kemudian kami membuat sambungan gambar atau urutan yang berbeda dengan bagian yang berbeda dalam resolusi yang berbeda-beda. Juga, seperti yang kita pahami sekarang dari diskusi kita di [Bab 7](#), *Deteksi Objek & Segmentasi Instance dengan CNN*, kita harus memiliki bagian yang relevan yang perlu kita deteksi dalam berbagai skala juga untuk deteksi yang efektif. Konsep yang sama berlaku di sini juga, dan selain resolusi, kami juga memvariasikan skalanya; tetapi untuk saat ini, kami akan membuatnya tetap sederhana dan mengabaikan bagian skala untuk pemahaman intuitif.

Sambungan atau urutan gambar ini sekarang bertindak sebagai urutan kata, seperti pada contoh sebelumnya, dan karenanya dapat diperlakukan di dalam RNN / LSTM atau arsitektur berbasis urutan serupa untuk tujuan perhatian. Hal ini dilakukan untuk mendapatkan kata yang paling cocok sebagai keluaran pada setiap iterasi. Jadi iterasi pertama dari urutan tersebut mengarah ke wanita (dari bobot urutan yang mewakili objek yang direpresentasikan sebagai *Wanita* dalam *Gambar 2*) → lalu iterasi berikutnya sebagai → *melihat* (dari urutan yang mengidentifikasi bagian belakang *Wanita* seperti pada *Gambar 2*) → *Dog* (urutan seperti pada *Gambar 3*) → *in* (dari urutan di mana semuanya dibuat kabur *filler* words bertransisi dari entitas ke lingkungan sekitar) → *Snow* (urutan seperti pada *Gambar 4*) → *Forest* (urutan seperti pada *Gambar 5*).

Kata pengisi seperti kata *dalam* dan tindakan seperti *melihat* juga dapat dipelajari secara otomatis saat pemetaan sambungan / urutan gambar terbaik ke teks buatan manusia dilakukan di beberapa gambar. Namun untuk versi yang lebih sederhana, caption seperti *Woman*, *Dog*, *Snow*, dan *Forest* juga bisa menjadi penggambaran entitas dan lingkungan yang baik pada gambar.

Jenis Perhatian

Ada dua jenis mekanisme perhatian. Mereka adalah sebagai berikut:

- Perhatian yang keras

- Perhatian lembut

Sekarang mari kita lihat masing-masing secara mendetail di bagian berikut.

Perhatian Keras

Pada kenyataannya, dalam contoh keterangan gambar kami baru-baru ini, beberapa gambar lagi akan dipilih, tetapi karena pelatihan kami dengan teks tulisan tangan, gambar itu tidak akan diberi bobot lebih tinggi. Namun, hal penting yang harus dipahami adalah bagaimana sistem akan memahami semua piksel (atau lebih tepatnya, representasi CNN dari mereka) yang menjadi fokus sistem untuk menggambar gambar resolusi tinggi ini dari berbagai aspek dan kemudian bagaimana memilih piksel berikutnya untuk ulangi prosesnya.

Dalam contoh sebelumnya, titik-titik dipilih secara acak dari sebuah distribusi dan prosesnya diulang. Juga, piksel mana di sekitar titik ini yang mendapatkan resolusi lebih tinggi ditentukan di dalam jaringan perhatian. Jenis perhatian ini dikenal sebagai **perhatian keras**.

Perhatian yang keras memiliki sesuatu yang disebut **masalah diferensiabilitas**. Mari luangkan waktu untuk memahami ini. Kami tahu bahwa dalam pembelajaran mendalam, jaringan harus dilatih dan untuk melatihnya, kami melakukan iterasi di seluruh batch pelatihan untuk meminimalkan fungsi kerugian. Kita dapat meminimalkan fungsi kerugian dengan mengubah bobot ke arah gradien minimum, yang pada gilirannya diperoleh setelah fungsi kerugian diturunkan.

Proses meminimalkan kerugian di seluruh lapisan jaringan dalam, mulai dari lapisan terakhir hingga lapisan pertama, dikenal sebagai propagasi balik .

Contoh beberapa fungsi kerugian yang dapat dibedakan yang digunakan dalam pembelajaran mendalam dan pembelajaran mesin adalah fungsi kerugian kemungkinan log, fungsi kerugian kesalahan kuadrat, entropi silang binomial dan multinomial, dan sebagainya.

Namun, karena titik dipilih secara acak di setiap iterasi dengan perhatian khusus — dan karena mekanisme pemilihan piksel acak seperti itu bukanlah fungsi yang dapat dibedakan - pada dasarnya kita tidak dapat melatih mekanisme perhatian ini, seperti yang dijelaskan. Masalah ini diatasi baik dengan menggunakan **Reinforcement Learning (RL)** atau dengan beralih ke soft attention.

*RL melibatkan mekanisme penyelesaian dua masalah, baik secara terpisah maupun kombinasi. Yang pertama disebut **masalah kontrol**, yang menentukan tindakan yang optimal paling bahwa agen harus mengambil dalam setiap langkah diberikan keadaan, dan yang kedua adalah **masalah prediksi**, yang menentukan optimal nilai negara.*

Perhatian Lembut

Seperti yang diperkenalkan dalam sub-bagian sebelumnya pada perhatian keras, perhatian lembut menggunakan RL untuk kereta progresif dan menentukan di mana untuk mencari berikutnya (con masalah trol) .

Ada dua masalah utama dengan menggunakan kombinasi perhatian keras dan RL untuk mencapai tujuan yang diperlukan:

- Menjadi sedikit rumit untuk melibatkan RL dan melatih agen RL dan RNN / jaringan dalam berdasarkan padanya secara terpisah.
- Varians dalam gradien fungsi kebijakan tidak hanya tinggi (seperti pada model A3C), tetapi juga memiliki kompleksitas komputasi $O(N)$, dimana N adalah jumlah unit dalam jaringan. Ini meningkatkan beban komputasi untuk pendekatan semacam itu secara masif. Juga, mengingat bahwa mekanisme perhatian menambah nilai lebih dalam urutan yang terlalu panjang (dari kata-kata atau sambungan penyematan gambar) —dan untuk melatih jaringan yang melibatkan urutan yang lebih panjang membutuhkan memori yang lebih besar, dan karenanya jaringan yang jauh lebih dalam - pendekatan ini secara komputasi tidak terlalu efisien.

The Fungsi Kebijakan di RL, ditentukan sebagai $Q(a, s)$, adalah fungsi yang digunakan untuk menentukan kebijakan yang optimal atau tindakan (a) yang harus diambil dalam keadaan tertentu (s) untuk memaksimalkan imbalan.

Jadi apa alternatifnya? Seperti yang kita diskusikan, masalah muncul karena mekanisme yang kita pilih untuk diperhatikan mengarah pada fungsi yang tidak dapat dibedakan, karena itu kita harus menggunakan RL. Jadi mari kita ambil pendekatan berbeda di sini. Mengambil analogi dari contoh masalah pemodelan bahasa kami (seperti pada bagian *Mekanisme Perhatian - Intuisi*) sebelumnya, kami berasumsi bahwa kami memiliki vektor token untuk objek / kata yang ada di jaringan perhatian. Juga, dalam ruang vektor yang sama (katakanlah di hyperspace embedding) kami membawa token untuk objek / kata-kata dalam kueri yang diperlukan dari langkah urutan tertentu. Dalam mengambil pendekatan ini, menemukan bobot perhatian yang tepat untuk token di jaringan perhatian sehubungan dengan token di ruang kueri semudah menghitung kesamaan vektor di antara mereka; misalnya jarak cosinus. Untungnya, sebagian besar fungsi jarak dan kesamaan vektor dapat dibedakan; maka fungsi kerugian yang diperoleh dengan menggunakan fungsi jarak / kesamaan vektor dalam ruang tersebut juga dapat dibedakan, dan propagasi balik kami dapat bekerja dalam skenario ini.

Jarak kosinus antara dua vektor, katakanlah $A(\vec{a_1}, \vec{a_2}, \vec{a_3})$, dan $B(\vec{b_1}, \vec{b_2}, \vec{b_3})$, dalam ruang vektor multi-dimensi (tiga dalam contoh ini) diberikan

$$sebagai: \quad \text{Cos}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\text{Abs}(\vec{A}) * \text{Abs}(\vec{B})} = \frac{a1b1 + a2b2 + a3b3}{\sqrt{a1^2 + a2^2 + a3^2} * \sqrt{b1^2 + b2^2 + b3^2}}$$

Pendekatan penggunaan fungsi kerugian yang dapat dibedakan untuk melatih jaringan perhatian ini dikenal sebagai **perhatian lunak** .

Menggunakan perhatian untuk meningkatkan model visual

Seperti yang kami temukan dalam contoh NLP yang tercakup di bagian sebelumnya tentang Mekanisme Perhatian - Intuisi, Perhatian sangat membantu kami dalam mencapai kasus penggunaan baru, tidak layak secara optimal dengan NLP konvensional, dan sangat meningkatkan kinerja mekanisme NLP yang ada. Serupa dengan penggunaan Attention di CNN dan Model Visual juga

Di bab sebelumnya [Bab 7](#) , *Deteksi Objek & Segmentasi Instans dengan CNN* , kami menemukan bagaimana mekanisme Perhatian (suka) digunakan sebagai Jaringan Proposal Wilayah untuk jaringan seperti Faster R-CNN dan Mask R-CNN, untuk meningkatkan dan mengoptimalkan wilayah yang diusulkan, dan memungkinkan pembuatan topeng segmen. Ini sesuai dengan bagian pertama diskusi. Pada bagian ini, kami akan membahas bagian kedua dari diskusi, di mana kami akan menggunakan mekanisme 'Perhatian' untuk meningkatkan kinerja CNN kami, bahkan dalam kondisi ekstrim.

Alasan untuk performa model CNN visual yang kurang optimal

Kinerja jaringan CNN dapat ditingkatkan sampai batas tertentu dengan mengadopsi mekanisme penyetelan dan persiapan yang tepat seperti: pra-pemrosesan data, normalisasi batch , pra-inisialisasi bobot yang optimal; memilih fungsi aktivasi yang benar; menggunakan teknik seperti regularisasi untuk menghindari overfitting; menggunakan fungsi pengoptimalan yang optimal; dan pelatihan dengan banyak data (berkualitas).

Di luar pelatihan dan keputusan terkait arsitektur ini, ada nuansa terkait gambar yang karenanya kinerja model visual dapat terpengaruh. Bahkan setelah mengontrol pelatihan dan faktor arsitektural yang disebutkan di atas, pengklasifikasi gambar berbasis CNN konvensional tidak berfungsi dengan baik pada beberapa kondisi berikut yang terkait dengan gambar yang mendasarinya:

- Gambar yang sangat besar
- Gambar sangat berantakan dengan sejumlah entitas klasifikasi

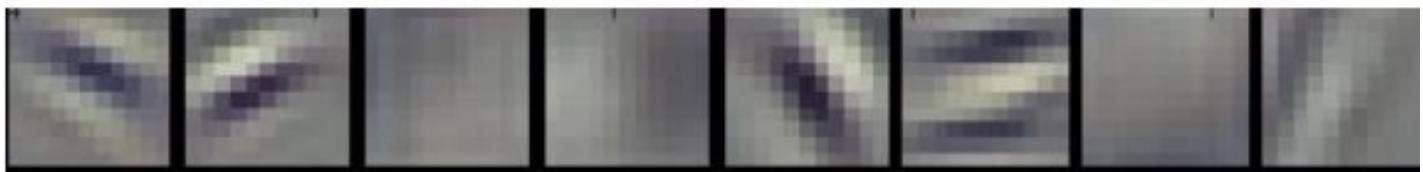
- Gambar yang sangat bising

Mari kita coba memahami alasan di balik kinerja sub-optimal dalam kondisi ini, dan kemudian secara logis kita akan memahami apa yang mungkin memperbaiki masalah tersebut.

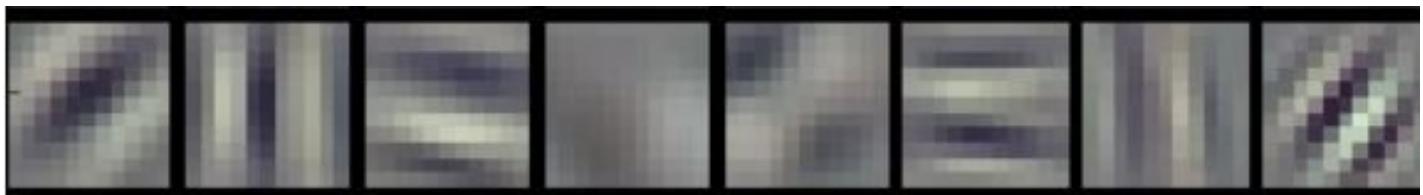
Dalam model konvensional berbasis CNN, bahkan setelah perampingan lintas lapisan, kompleksitas komputasinya cukup tinggi. Faktanya, kompleksitas adalah urutan $O(L * WB * PPI^2)$, di mana L dan W adalah panjang dan lebar gambar dalam inci, dan PPI adalah piksel per inci (kerapatan piksel). Ini diterjemahkan ke dalam kompleksitas linier sehubungan dengan jumlah total piksel (P) pada gambar, atau $O(P)$. Ini secara langsung menjawab poin pertama dari tantangan; untuk L , W , atau PPI yang lebih tinggi, kita membutuhkan daya komputasi dan waktu yang jauh lebih tinggi untuk melatih jaringan.

Operasi seperti penggabungan maksimal, penggabungan rata-rata, dan sebagainya membantu mengurangi beban komputasi secara drastis terhadap semua penghitungan di semua lapisan yang dilakukan pada gambar aktual.

Jika kita memvisualisasikan pola yang terbentuk di setiap lapisan CNN kita, kita akan memahami intuisi di balik kerja CNN dan mengapa perlu mendalam. Di setiap lapisan berikutnya, CNN melatih fitur konseptual yang lebih tinggi, yang secara progresif dapat membantu memahami objek dengan lebih baik pada gambar lapisan demi lapisan. Jadi, dalam kasus MNIST, lapisan pertama hanya dapat mengidentifikasi batas, yang kedua adalah diagonal dan bentuk batas berbasis garis lurus, dan seterusnya:



Fitur konseptual ilustratif yang dibentuk di berbagai lapisan (awal) CNN untuk MNIST



MNIST adalah kumpulan data sederhana, sedangkan gambar kehidupan nyata cukup kompleks; ini membutuhkan fitur konseptual yang lebih tinggi untuk membedakannya, dan karenanya jaringan yang lebih kompleks dan jauh lebih dalam. Selain itu, di MNIST, kami mencoba untuk membedakan antara jenis objek yang serupa (semua angka tulisan tangan). Sedangkan dalam kehidupan nyata, objek mungkin sangat berbeda, dan karenanya berbagai jenis fitur yang mungkin diperlukan untuk memodelkan semua objek tersebut akan sangat tinggi:



Ini membawa kita ke tantangan kedua. Gambar yang berantakan dengan terlalu banyak objek akan membutuhkan jaringan yang sangat kompleks untuk memodelkan semua objek ini. Selain itu, karena ada terlalu banyak objek untuk diidentifikasi, resolusi gambar harus baik untuk mengekstrak dan memetakan fitur untuk setiap objek dengan benar, yang pada gilirannya berarti bahwa ukuran gambar dan jumlah piksel harus tinggi untuk klasifikasi yang efektif. Hal ini, pada gilirannya, meningkatkan kompleksitas secara eksponensial dengan menggabungkan dua tantangan pertama.

Jumlah lapisan, dan kompleksitas arsitektur CNN populer yang digunakan dalam tantangan ImageNet, telah meningkat selama bertahun-tahun. Beberapa contohnya adalah VGG16 - Oxford (2014) dengan 16 lapisan , GoogLeNet (2014) dengan 19 lapisan , dan ResNet (2015) dengan 152 lapisan.

Tidak semua gambar memiliki kualitas SLR yang sempurna. Seringkali, karena cahaya redup, pemrosesan gambar, resolusi rendah, kurangnya stabilisasi, dan sebagainya, mungkin ada banyak noise yang masuk pada gambar. Ini hanyalah salah satu bentuk kebingungan, yang lebih mudah dipahami. Dari perspektif CNN, bentuk lain dari noise dapat berupa transisi gambar, rotasi, atau transformasi:



Gambar tanpa noise



Gambar yang sama dengan noise tambahan

Pada gambar sebelumnya, coba baca judul koran *Business* pada gambar tanpa dan dengan noise, atau identifikasi ponsel di kedua gambar tersebut. Sulit melakukannya pada gambar dengan noise, bukan? Serupa dengan tantangan deteksi / klasifikasi dengan CNN kami dalam kasus gambar yang berisik.

Bahkan dengan pelatihan yang melelahkan, penyesuaian hyperparameter yang sempurna, dan teknik seperti putus sekolah dan lainnya, tantangan kehidupan nyata ini terus mengurangi keakuratan pengenalan citra jaringan CNN. Sekarang setelah kita memahami penyebab dan intuisi di balik kurangnya akurasi dan kinerja di CNN kita, mari kita jelajahi beberapa cara dan arsitektur untuk mengatasi tantangan ini menggunakan perhatian visual.

Model perhatian visual yang berulang

Model perhatian visual yang berulang dapat digunakan untuk menjawab beberapa tantangan yang kita bahas di bagian sebelumnya. Model ini menggunakan metode hard attention, seperti yang dibahas di bagian (*Jenis perhatian*) sebelumnya. Di sini kami menggunakan salah satu varian populer dari model perhatian visual **berulang**, **Recurrent Attention Model (RAM)**.

Seperti yang dibahas sebelumnya, masalah perhatian keras tidak dapat dibedakan dan harus menggunakan RL untuk masalah kontrol. RAM dengan demikian menggunakan RL untuk pengoptimalan ini.

Model perhatian visual berulang tidak memproses seluruh gambar, atau bahkan kotak pembatas berbasis jendela geser, sekaligus. Ini meniru mata manusia dan bekerja pada konsep *Fiksasi* dari *Gaze* di lokasi yang berbeda dari suatu gambar; dengan setiap *Fiksasi*, *Fiksasi* secara bertahap menggabungkan informasi penting untuk secara dinamis membangun representasi internal adegan dalam gambar. Ini menggunakan RNN untuk melakukan ini secara berurutan.

Model memilih lokasi berikutnya untuk diperbaiki berdasarkan kebijakan kontrol agen RL untuk memaksimalkan hadiah berdasarkan keadaan saat ini. Keadaan saat ini, pada gilirannya, merupakan fungsi dari semua informasi masa lalu dan tuntutan tugas. Dengan demikian, ia menemukan koordinat berikutnya untuk fiksasi sehingga dapat memaksimalkan reward (tuntutan tugas), mengingat informasi yang dikumpulkan hingga saat ini di seluruh tatapan sebelumnya dalam snapshot memori RNN dan koordinat yang dikunjungi sebelumnya.

Sebagian besar mekanisme RL menggunakan Markov Decision Process (MDP), di mana tindakan selanjutnya hanya ditentukan oleh keadaan saat ini, terlepas dari negara bagian yang dikunjungi sebelumnya. Dengan menggunakan RNN di sini, informasi penting dari Fiksasi sebelumnya dapat digabungkan dalam kondisi saat ini.

Mekanisme sebelumnya memecahkan dua masalah terakhir yang disorot di CNN di bagian sebelumnya. Selain itu, dalam RAM, jumlah parameter dan jumlah komputasi

yang dilakukannya dapat dikontrol secara independen dari ukuran gambar input, sehingga memecahkan masalah pertama juga.

Menerapkan RAM pada sampel MNIST yang berisik

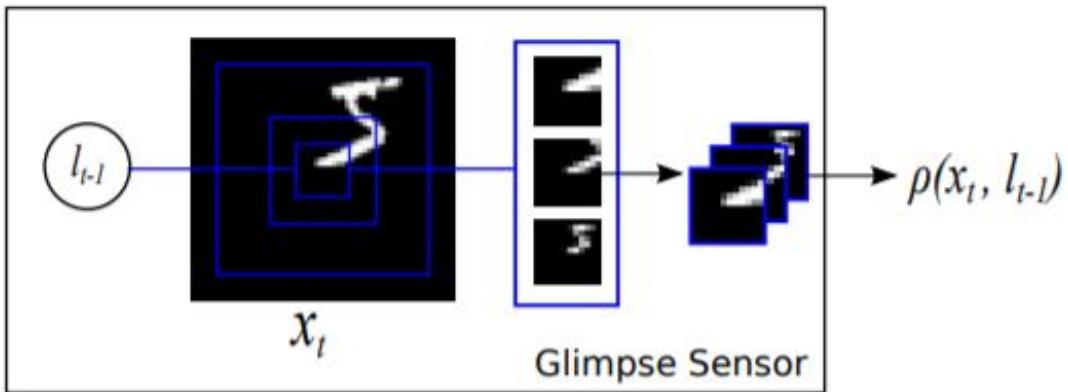
Untuk memahami cara kerja RAM secara lebih rinci, mari kita coba membuat sampel MNIST yang menggabungkan beberapa masalah seperti yang disorot di bagian sebelumnya:



Gambar lebih besar dari MNIST berisik dan terdistorsi

Gambar sebelumnya mewakili gambar / kolase yang lebih besar menggunakan sampel gambar MNIST yang sebenarnya dan sedikit berisik (dari nomor 2), dan banyak distorsi dan cuplikan lain dari sampel parsial lainnya. Juga, angka 2 sebenarnya tidak berada di tengah. Contoh ini mewakili semua masalah yang disebutkan sebelumnya, namun cukup sederhana untuk memahami cara kerja RAM.

RAM menggunakan konsep **Glimpse Sensor**. Agen RL memperbaiki pandangannya pada koordinat tertentu (l) dan waktu tertentu ($t-1$). Koordinat pada waktu $t-1$, l_{t-1} gambar x_t dan menggunakan **Glimpse Sensor** untuk mengekstrak patch beberapa resolusi mirip retina dengan l_{t-1} sebagai pusatnya. Representasi ini, diekstraksi pada waktu $t-1$, secara kolektif disebut $p(x_t, l_{t-1})$:

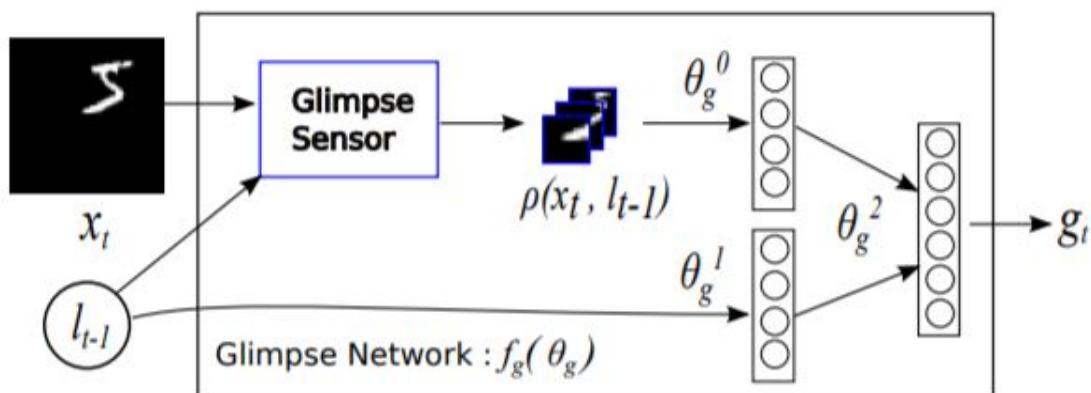


Konsep Sensor Sekilas



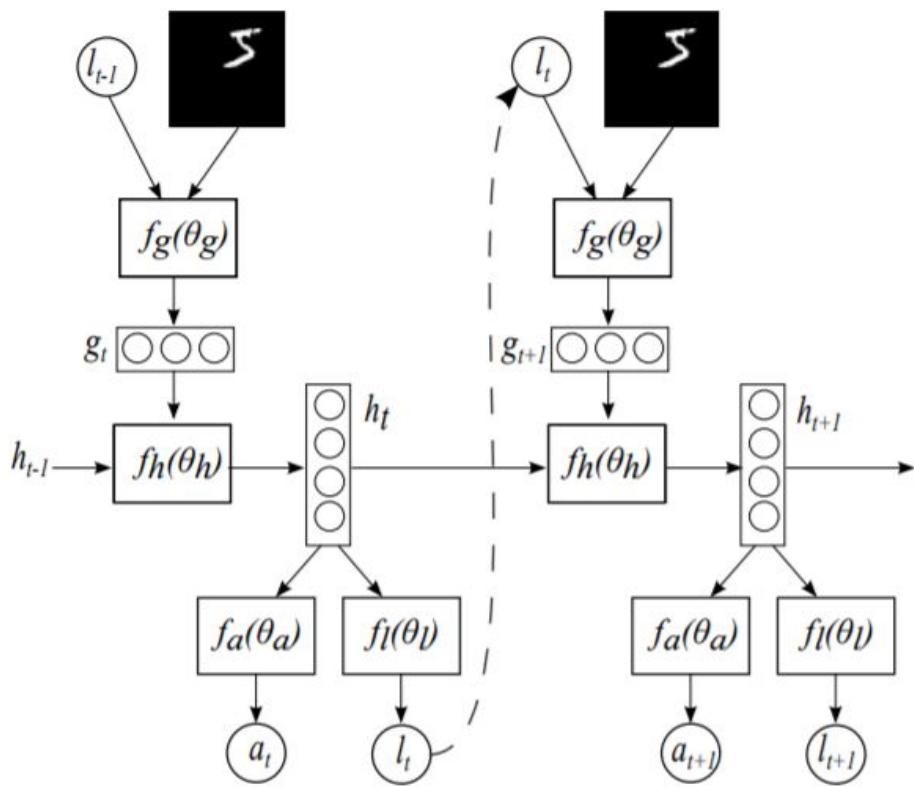
Gambar ini menunjukkan representasi gambar kita di dua fiksasi menggunakan **Glimpse Sensor**.

Representasi yang diperoleh dari **Glimpse Sensor** dilewatkan melalui 'Glimpse Network', yang meratakan representasi pada dua tahap. Pada tahap pertama, representasi dari **Glimpse Sensor** dan **Jaringan Sekilas** diratakan secara terpisah (θ_g^0, θ_g^1), dan kemudian mereka digabungkan ke dalam lapisan pipih tunggal (θ_g^2) untuk menghasilkan output representasi g_t untuk waktu t :



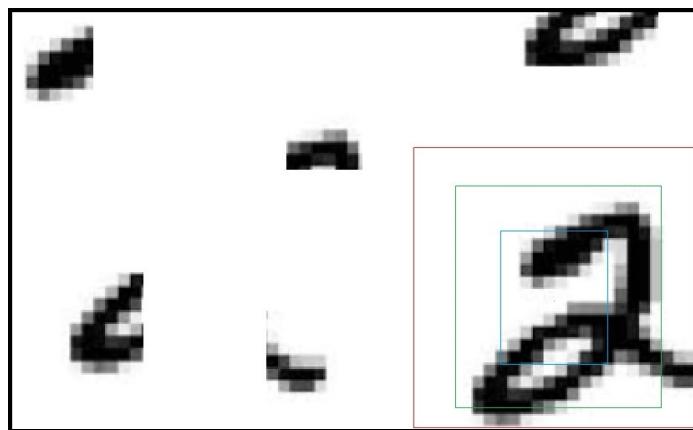
Konsep Jaringan Sekilas

Representasi keluaran ini kemudian diteruskan melalui arsitektur model RNN. Fiksasi untuk langkah berikutnya dalam iterasi ditentukan oleh agen RL untuk memaksimalkan reward dari arsitektur ini:



Arsitektur model (RNN)

Seperti yang dapat dipahami secara intuitif, Glimpse Sensor menangkap informasi penting di seluruh fiksasi, yang dapat membantu mengidentifikasi konsep penting. Misalnya, representasi beberapa resolusi (di sini 3) pada Fiksasi yang diwakili oleh gambar sampel kedua kami memiliki tiga resolusi yang ditandai (merah, hijau, dan biru dalam urutan penurunan resolusi). Seperti yang dapat dilihat, meskipun ini digunakan secara langsung, kami memiliki kemampuan yang bervariasi untuk mendeteksi digit yang tepat yang diwakili oleh kolase yang berisik ini:



| Digit | Probability | Probability | Probability |
|-------|-------------|-------------|-------------|
| 0 | 0.01 | 0.15 | 0.06 |
| 1 | 0.03 | 0.12 | 0.08 |
| 2 | 0.72 | 0.26 | 0.01 |
| 3 | 0.04 | 0.13 | 0.05 |
| 4 | 0.02 | 0.01 | 0.06 |
| 5 | 0.05 | 0.12 | 0.1 |
| 6 | 0.03 | 0.13 | 0.12 |
| 7 | 0.01 | 0.04 | 0.12 |
| 8 | 0.04 | 0.01 | 0.06 |
| 9 | 0.05 | 0.03 | 0.05 |

Sekilas Sensor dalam kode

Seperti yang dibahas di bagian sebelumnya, Sensor Sekilas adalah konsep yang hebat. Dikombinasikan dengan konsep lain, seperti RNN dan RL, seperti yang dibahas sebelumnya, ini merupakan inti dari peningkatan performa model visual.

Mari kita lihat lebih detail di sini. Kode dikomentari di setiap baris untuk memudahkan pemahaman dan cukup jelas:

```

import tensorflow as tf
# kode dalam tensorflow
import numpy as np

def glimpseSensor (image, fixationLocation):
    """
        Glimpse Sensor for Recurrent Attention Model (RAM)
    : param image: image xt : type image: numpy vector : param fixationLocation : cordinates 1 untuk
    pusat fiksasi : tipe fiksasiLokasi: tuple : return : Representasi Multi Resolusi dari Sensor
    Sekilas : rtype :      ''' img_size = np.asarray (image) .shape [: 2 ]
    """

    # ini dapat disetel sebagai default dari ukuran gambar dalam dataset kami, meninggalkan dimensi
    'saluran' ketiga jika ada
    saluran = 1 # pengaturan saluran sebagai 1 secara default if (np.asarray (img_size) .shape [ 0 ]
    == 3 ):           channels = np.asarray (image) .shape [- 1 ] # mengatur ulang ukuran saluran jika
    saluran ada batch_size = 32 # mengatur ukuran batch loc = tf.round (((fixationLocation + 1 ) / 2.0
    ) * img_size) # fixationLocation koordinat dinormalisasi antara -1 dan 1 pusat gambar wrt sebagai
    0,0 loc = tf.cast (loc, tf.int32)

```

```

# mengonversi format angka yang kompatibel dengan tf
image = tf.reshape (image, (batch_size, img_size [ 0 ], img_size [ 1 ], channels)) # mengubah
bentuk vektor img agar sesuai dengan representasi tf = [] # representasi gambar glimpse_images = [
] # untuk ditampilkan di jendela minRadius = img_size [ 0 ] / 10 # mengatur ukuran sisi resolusi
terkecil gambar max_radius = minRadius * 2 offset = 2 * max_radius # mengatur sisi maks dan offset
untuk menggambar representasi kedalaman = 3 # jumlah representasi per fiksasi sensorBandwidth = 8

# bandwidth sensor untuk sensor sekilas
# memproses setiap gambar secara individual untuk k dalam rentang (batch_size):
imageRepresentations = [] one_img = image [k,:,:,:]
# memilih gambar yang diperlukan untuk membentuk batch one_img = tf.image.pad_to_bounding_box (one_img, offset, offset, max_radius * 4 + img_size, max_radius * 4 + img_size) # pad gambar dengan nol untuk digunakan di tf karena kami memerlukan ukuran yang konsisten untuk i dalam rentang (kedalaman): r = int (minRadius * ( 2 ** (i))) # radius gambar d_raw =
2 * r
# diameter
d = tf.Konstanta (d_raw, bentuk = [ 1 ]) # tf konstan untuk dia d = tf.Ubin (d, [ 2 ])
loc_k = loc [k ,:] adjusted_loc = offset + loc_k - r # lokasi gambar wrt disesuaikan
transformasi gambar wrt dan pad one_img2 = tf.reshape (one_img, (one_img.get_shape () [ 0 ].value,
one_img.get_shape () [ 1 ].value)) # membentuk kembali gambar untuk representasi tf = tf.slice
(one_img2, adjusted_loc, d) # crop gambar ke (dxd) untuk representasi representasi =
tf.image.resize_bilinear (tf.reshape (representasi, ( 1
, d_raw, d_raw, 1 )), (sensorBandwidth, sensorBandwidth))
# ubah ukuran gambar yang dipotong menjadi (sensorBandwidth x sensorBandwidth)
representasi = tf.reshape (representasi, (sensorBandwidth, sensorBandwidth)) # reshape untuk tf
imageRepresentations.append (representasi) # menambahkan representasi saat ini ke himpunan
representasi untuk representasi_gambar.append (tf.stack (imageRepresentations)) representations
= tf.stack (representasi) glimpse_images.append (representasi) # return glimpse sensor output
return representasi

```

Referensi

1. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S.Zemel, Yoshua Bengio, Show, Attend and Tell: *Neural Image Caption Generation with Visual Attention* , CoRR, arXiv: 1502.03044, 2015.
2. Karl Moritz Hermann, Tom's Kocisk, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, Phil Blunsom, *Teaching Machines to Read and Comprehend* , CoRR, arXiv: 1506.03340, 2015.
3. Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu, *Recurrent Model of Visual Attention* , CoRR, arXiv: 1406.6247, 2014.
4. Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Tat-Seng Chua, SCA-CNN: *Perhatian Spasial dan Channel-Wise di Convolutional Networks for Image Captioning* , CoRR, arXiv: 1611.05594, 2016.
5. Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, Ram Nevatia, ABC-CNN: *Jaringan Saraf Konvolusional Berbasis Perhatian untuk Menjawab Pertanyaan Visual* , CoRR, arXiv: 1511.05960, 2015.
6. Wenpeng Yin, Sebastian Ebert, Hinrich Schutze, *Jaringan Neural Konvolusional Berbasis Perhatian untuk Pemahaman Mesin* , CoRR, arXiv: 1602.04341, 2016.
7. Wenpeng Yin, Hinrich Schutze, Bing Xiang, Bowen Zhou, ABCNN: *Jaringan Neural Konvolusional Berbasis Perhatian untuk Modeling Sentence Pairs* , CoRR, arXiv: 1512.05193, 2015.
8. Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, Alexander J. Smola, *Jaringan Perhatian Bertumpuk untuk Penjawab Pertanyaan Gambar* , CoRR, arXiv: 1511.02274, 2015.
9. Y. Chen, D. Zhao, L. Lv dan C. Li, *Jaringan saraf konvolusional berbasis perhatian visual untuk klasifikasi gambar* , Kongres Dunia ke-12 untuk Kontrol dan Otomasi Cerdas (WCICA) 2016, Guilin, 2016, hlm.764-769.
10. H. Zheng, J. Fu, T. Mei dan J. Luo, *Learning Multi-attention Convolutional Neural Network for Fine-Grained Image Recognition* , 2017 IEEE International Conference on Computer Vision (ICCV) , Venice, 2017, hlm. 5219-5227 .
11. Tianjun Xiao, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, Zheng Zhang, *Penerapan Model Perhatian Dua Tingkat di Jaringan Saraf Konvolusional Dalam untuk Klasifikasi Gambar Berbutir Halus*, CoRR, arXiv: 1411.6447, 2014.
12. Jlindsey15, *Implementasi TensorFlow dari model perhatian berulang* , GitHub, <https://github.com/jlindsey15/RAM> , Feb 2018.
13. QihongL, *Implementasi TensorFlow dari model perhatian berulang* , GitHub, <https://github.com/QihongL/RAM> , Feb 2018.

Ringkasan

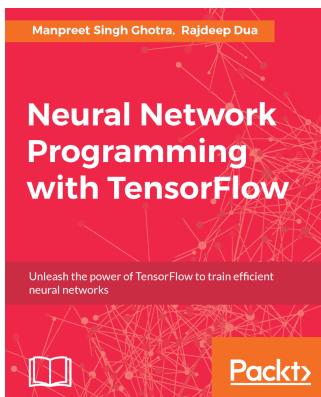
Mekanisme perhatian adalah topik terpanas dalam pembelajaran mendalam saat ini dan dianggap sebagai pusat dari sebagian besar algoritme mutakhir dalam penelitian saat ini, dan dalam kemungkinan penerapan di masa mendatang. Masalah-masalah seperti pembuatan teks gambar , jawaban pertanyaan visual, dan banyak lagi mendapatkan solusi hebat dengan menggunakan pendekatan ini. Faktanya, perhatian tidak terbatas pada tugas visual dan telah dipahami sebelumnya untuk masalah seperti terjemahan mesin saraf dan masalah NLP canggih lainnya. Dengan demikian, memahami mekanisme perhatian sangat penting untuk menguasai banyak teknik pembelajaran mendalam tingkat lanjut.

CNN digunakan tidak hanya untuk penglihatan tetapi juga untuk banyak aplikasi bagus dengan perhatian untuk memecahkan masalah NLP yang kompleks, seperti **pasangan kalimat pemodelan dan terjemahan mesin** . Bab ini membahas mekanisme perhatian dan penerapannya pada beberapa masalah NLP, bersama dengan pembuatan teks gambar dan model penglihatan berulang. Di RAM, kami tidak menggunakan CNN; sebagai gantinya, kami menerapkan RNN dan perhatian pada representasi gambar dengan ukuran yang diperkecil dari Sensor Sekilas. Namun ada juga karya terbaru yang menerapkan perhatian pada model visual berbasis CNN juga.

Pembaca sangat dianjurkan untuk membaca makalah asli dalam referensi dan juga mengeksplorasi konsep lanjutan dalam menggunakan perhatian, seperti perhatian multi-level, model perhatian bertumpuk, dan penggunaan model RL (seperti **Asynchronous Advantage Actor-Critic (A3C)**) model untuk masalah kontrol perhatian keras).

Buku Lain yang Mungkin Anda Nikmati

Jika Anda menyukai buku ini, Anda mungkin tertarik dengan buku-buku lain karya Packt ini:

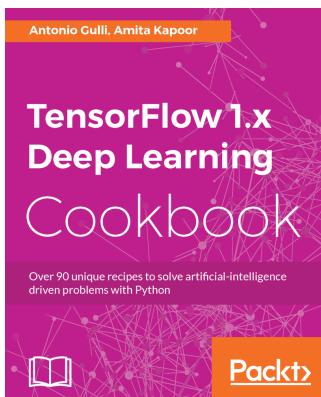


Pemrograman Jaringan Neural dengan Tensorflow

Rajdeep Dua, Manpreet Singh Ghotra

ISBN: 978-1-78839-039-2

- Pelajari Aljabar Linear dan matematika di balik jaringan saraf.
- Selami jauh ke dalam jaringan Neural dari konsep dasar hingga lanjutan seperti CNN, RNN Deep Belief Networks, Deep Feedforward Networks.
- Jelajahi teknik Pengoptimalan untuk memecahkan masalah seperti Minima lokal, Minimum global, poin Saddle
- Pelajari melalui contoh dunia nyata seperti Analisis Sentimen.
- Latih berbagai jenis model generatif dan jelajahi pembuat kode otomatis.
- Jelajahi TensorFlow sebagai contoh implementasi deep learning.



TensorFlow 1.x Buku Resep Deep Learning

Antonio Gulli, Amita Kapoor

ISBN: 978-1-78829-359-4

- Instal TensorFlow dan gunakan untuk operasi CPU dan GPU
- Terapkan DNN dan terapkan untuk menyelesaikan berbagai masalah yang didorong oleh AI.
- Manfaatkan kumpulan data yang berbeda seperti MNIST, CIFAR-10, dan Youtube8m dengan TensorFlow serta pelajari cara mengakses dan menggunakan dalam kode Anda.
- Gunakan TensorBoard untuk memahami arsitektur jaringan neural, mengoptimalkan proses pembelajaran, dan mengintip ke dalam kotak hitam jaringan neural.

- Gunakan teknik regresi yang berbeda untuk masalah prediksi dan klasifikasi
- Buat perceptron tunggal dan multilayer di TensorFlow
- Implementasikan CNN dan RNN di TensorFlow, dan gunakan untuk menyelesaikan kasus penggunaan dunia nyata.
- Pelajari bagaimana Mesin Boltzmann terbatas dapat digunakan untuk merekomendasikan film.
- Pahami penerapan Autoencoders dan jaringan keyakinan mendalam, dan gunakan untuk mendeteksi emosi.
- Kuasai berbagai metode pembelajaran penguatan untuk menerapkan agen permainan game.
- GAN dan implementasinya menggunakan TensorFlow.

Tinggalkan ulasan - beri tahu pembaca lain apa yang Anda pikirkan

Bagikan pendapat Anda tentang buku ini dengan orang lain dengan meninggalkan ulasan di situs tempat Anda membelinya. Jika Anda membeli buku itu dari Amazon, berikan ulasan yang jujur kepada kami di halaman Amazon buku ini. Hal ini penting agar pembaca potensial lainnya dapat melihat dan menggunakan opini Anda yang tidak bias untuk membuat keputusan pembelian, kami dapat memahami apa yang pelanggan pikirkan tentang produk kami, dan penulis kami dapat melihat tanggapan Anda tentang judul yang mereka buat dengan Packt. Hanya perlu beberapa menit dari waktu Anda, tetapi bermanfaat bagi calon pelanggan lainnya, penulis kami, dan Packt. Terima kasih!