BAB 3

Klasifikasi Gambar

Menggunakan LeNet

Perjalanan seribu mil dimulai dengan satu langkah.

—Lao Tzu

Dan Anda telah mengambil langkah luar biasa dalam mempelajari Pembelajaran Mendalam. Pembelajaran Mendalam merupakan bidang yang terus berkembang. Dari Jaringan Syaraf dasar, kini kita telah berevolusi ke arsitektur kompleks yang memecahkan banyak masalah bisnis. Kemampuan Pemrosesan Gambar dan visi komputer yang didukung Pembelajaran Mendalam memungkinkan kita menciptakan solusi deteksi kanker yang lebih baik, mengurangi tingkat polusi, menerapkan sistem pengawasan, dan meningkatkan pengalaman konsumen. Terkadang, masalah bisnis menuntut pendekatan yang disesuaikan. Kita dapat merancang jaringan kita sendiri agar sesuai dengan masalah bisnis yang dihadapi dan berdasarkan kualitas gambar yang tersedia bagi kita.

Desain jaringan juga akan mempertimbangkan daya komputasi yang tersedia untuk melatih dan menjalankan jaringan. Para peneliti di berbagai organisasi dan universitas menghabiskan banyak waktu untuk mengumpulkan dan mengkurasi kumpulan data, membersihkan dan menganalisisnya, merancang arsitektur, melatih dan mengujinya, serta mengulanginya untuk meningkatkan kinerja. Diperlukan banyak waktu dan kesabaran untuk membuat solusi yang inovatif.

Dalam dua bab pertama, kita telah membahas dasar-dasar Jaringan Neural dan membuat solusi Pembelajaran Mendalam menggunakan Keras dan Python. Dari bab ini dan seterusnya, kita akan mulai membahas Jaringan Neural yang kompleks.

Verdhan, *Visi Komputer Menggunakan Pembelajaran Mendalam,*
https://doi.org/10.1007/978-1-4842-6616-8_3

Arsitektur jaringan. Pertama-tama, kami akan memperkenalkan arsitektur LeNet.

Kita akan membahas desain jaringan, berbagai lapisan, fungsi aktivasi, dan sebagainya.

Kemudian, kita akan mengembangkan model untuk kasus penggunaan klasifikasi gambar.

Secara khusus, kami akan membahas topik-topik berikut dalam bab ini:

1. Arsitektur LeNet dan variannya
2. Desain Arsitektur LeNet
3. Klasifikasi digit MNIST
4. Klasifikasi rambu lalu lintas Jerman
5. Ringkasan

Selamat datang di bab ketiga dan semoga sukses!

3.1 Persyaratan teknis

Kode dan kumpulan data untuk bab ini diunggah di tautan GitHub https://github.com/Apress/computer-vision-using-deep-learning/pohon/utama/Bab3 untuk buku ini. Kami akan menggunakan Jupyter Notebook. Untuk bab ini, CPU sudah cukup baik untuk menjalankan kode, tetapi jika diperlukan Anda dapat menggunakan Google Colaboratory. Anda dapat merujuk ke referensi buku untuk Google Colab.

Mari lanjutkan dengan arsitektur Pembelajaran Mendalam di bagian berikutnya.

3.2 Arsitektur Pembelajaran Mendalam

Ketika kita membahas jaringan Pembelajaran Mendalam, ada beberapa komponen yang langsung terlintas di pikiran kita – jumlah neuron, jumlah lapisan, fungsi aktivasi yang digunakan, kerugian, dan sebagainya. Semua parameter ini memainkan peran penting dalam desain jaringan dan kinerjanya. Ketika kita merujuk pada kedalaman dalam Jaringan Neural, yang dimaksud adalah jumlah lapisan tersembunyi dalam jaringan. Dengan peningkatan daya komputasi, jaringan menjadi lebih dalam dan permintaan daya komputasi juga meningkat.

Info Meskipun Anda mungkin berpikir bahwa menambah jumlah lapisan dalam jaringan akan menghasilkan peningkatan akurasi, hal itu tidak selalu terjadi. Inilah yang menghasilkan jaringan baru yang disebut ResNet.

Dengan menggunakan komponen dasar ini, kita dapat merancang jaringan kita sendiri. Para peneliti dan ilmuwan di seluruh dunia telah menghabiskan banyak waktu dan upaya untuk menghasilkan berbagai arsitektur Jaringan Saraf.
Arsitektur yang paling populer adalah *LeNet-5, AlexNet, VGGNet, GoogLeNet, ResNet, R-CNN (Region-based CNN), YOLO (You Only Look Once), SqueezeNet, SegNet, GAN (Generative Adversarial Network),* dan sebagainya. Jaringan ini menggunakan jumlah lapisan tersembunyi, neuron, fungsi aktivasi, metode pengoptimalan, dan sebagainya yang berbeda. Beberapa arsitektur lebih cocok daripada yang lain berdasarkan masalah bisnis yang dihadapi.

Dalam bab ini, kita akan membahas arsitektur LeNet secara terperinci dan kemudian mengembangkan beberapa kasus penggunaan. Kita mulai dengan LeNet karena merupakan salah satu kerangka kerja yang lebih mudah dipahami dan salah satu pelopor dalam arsitektur Pembelajaran Mendalam. Kita juga akan memeriksa dampaknya terhadap kinerja model kita dengan perubahan dalam berbagai hiperparameter.

3.3 Arsitektur LeNet

Seperti yang telah kita bahas di bagian sebelumnya, LeNet adalah arsitektur pertama yang kita bahas dalam buku ini. Ini adalah salah satu arsitektur CNN yang paling sederhana. Arsitektur ini menjadi penting karena sebelum ditemukan, pengenalan karakter merupakan proses yang rumit dan memakan waktu. Arsitektur LeNet diperkenalkan pada tahun 1998, dan menjadi populer saat digunakan untuk mengklasifikasikan angka tulisan tangan pada cek bank di Amerika Serikat.

Arsitektur LeNet memiliki beberapa bentuk – *LeNet-1, LeNet-4,* dan *LeNet-5,* yang merupakan yang paling banyak dikutip dan dirayakan. LeNet-5 dikembangkan oleh Yann LeCun selama kurun waktu tertentu. Demi menghemat ruang, kami meneliti LeNet-5 secara terperinci, dan arsitektur lainnya dapat dipahami menggunakan metodologi yang sama.

Kami memulai dengan arsitektur dasar LeNet-1 di bagian berikutnya.

3.4 Arsitektur LeNet-1

Arsitektur LeNet-1 mudah dipahami. Mari kita lihat dimensi lapisannya.

Lapisan pertama: gambar masukan 28x28

Lapisan kedua: Empat Lapisan Konvolusional 24x24 (ukuran 5x5)

Lapisan ketiga: Lapisan pengumpulan rata-rata (ukuran 2×2)

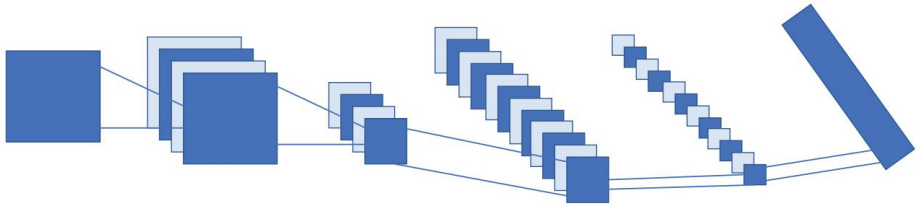
Lapisan keempat: Delapan lapisan konvolusional 12×12 (ukuran 5×5)

Lapisan kelima: Lapisan pengumpulan rata-rata (ukuran 2×2)

Dan akhirnya, kita memiliki lapisan keluaran.

Info Ketika LeNet diperkenalkan, para peneliti tidak mengusulkan penggabungan maksimum; sebagai gantinya, penggabungan rata-rata digunakan. Anda disarankan untuk menguji solusi dengan menggunakan penggabungan rata-rata dan maksimum.

Arsitektur LeNet-1 ditunjukkan pada Gambar 3-1.



Gambar 3-1. *Arsitektur LeNet-1 merupakan LeNet pertama yang dikonseptualisasikan. Kita harus perhatikan bagaimana kita memiliki lapisan pertama sebagai lapisan konvolusional, diikuti oleh penggabungan, lapisan konvolusional lain, dan lapisan penggabungan. Dan akhirnya, kita memiliki lapisan keluaran di bagian akhir*

Anda dapat melihat di sini bahwa kita memiliki lapisan input, diikuti oleh Lapisan Konvolusional, lalu Lapisan Penggabungan, diikuti oleh Lapisan Konvolusional, lalu Lapisan Penggabungan, dan terakhir keluaran. Gambar-gambar tersebut mengalami transformasi selama seluruh jaringan sesuai konfigurasinya.

Kami telah menjelaskan secara rinci fungsi semua lapisan dan keluaran masing-masing saat kami membahas LeNet-5 di bagian berikutnya.

Arsitektur 3.5 LeNet-4

Arsitektur LeNet-4 sedikit lebih baik daripada LeNet-1. Ada satu lagi Fully Connected Layer dan lebih banyak peta fitur.

Lapisan pertama: gambar masukan 32x32

Lapisan kedua: Empat Lapisan Konvolusional 24x24 (ukuran 5x5)

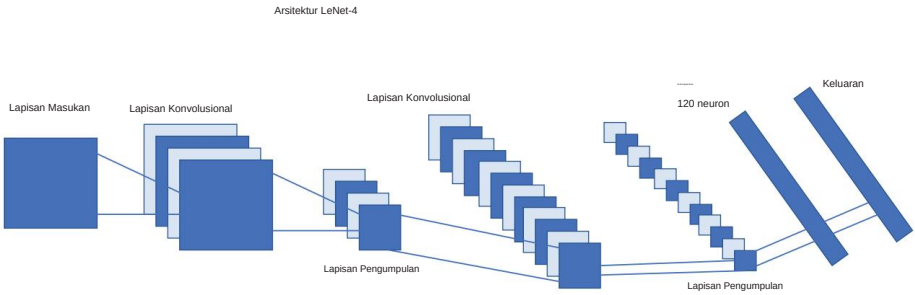
Lapisan ketiga: Lapisan pengumpulan rata-rata (ukuran 2×2)

Lapisan keempat: Enam belas lapisan konvolusional 12×12 (ukuran 5×5)

Lapisan kelima: Lapisan pengumpulan rata-rata (ukuran 2×2)

Output terhubung sepenuhnya ke 120 neuron, yang sepenuhnya terhubung lebih lanjut ke 10 neuron sebagai keluaran akhir.

Arsitektur LeNet-4 ditunjukkan pada Gambar 3-2.



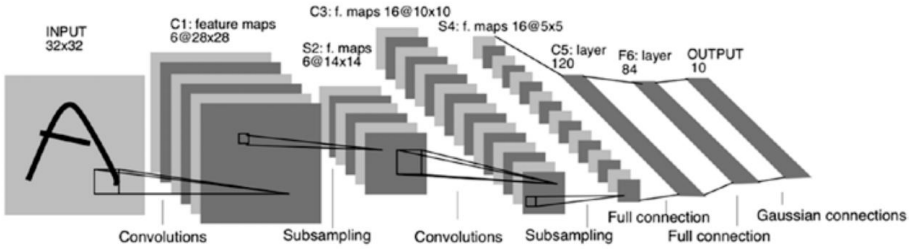
Gambar 3-2. *LeNet-4 merupakan penyempurnaan dari arsitektur LeNet-1. Satu lapisan yang terhubung sepenuhnya telah diperkenalkan di dalamnya*

Untuk pemahaman lebih rinci tentang fungsi semua lapisan, lihat bagian berikutnya di mana kami membahas LeNet-5.

3.6 Arsitektur LeNet-5

Di antara ketiga arsitektur LeNet, arsitektur yang paling banyak dikutip adalah LeNet-5, dan merupakan arsitektur yang umumnya digunakan saat memecahkan masalah bisnis.

Arsitektur LeNet-5 ditunjukkan pada Gambar 3-3. Arsitektur ini awalnya dibahas dalam makalah “Gradient-Based Learning Applied to Document Recognition,” Y. LeCun et al. Makalah ini dapat diakses di http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.



Gambar 3-3. *Arsitektur LeNet – gambar diambil dari http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf*

Mari kita bahas lapisan-lapisannya secara detail karena ini adalah lapisan yang paling umum digunakan arsitektur:

1. Lapisan pertama: Lapisan pertama LeNet-5 adalah 32x32 lapisan gambar input. Ini adalah gambar skala abu-abu yang melewati blok konvolusional dengan enam filter berukuran 5x5. Dimensi yang dihasilkan adalah 28x28x1 dari 32x32x1. Di sini, 1 mewakili saluran; nilainya 1 karena ini adalah gambar skala abu-abu. Jika itu RGB, akan ada tiga saluran.

2. Lapisan kedua: Lapisan pengumpulan juga disebut *Lapisan subsampling* memiliki ukuran filter 2x2 dan langkah 2. Dimensi gambar diperkecil menjadi 14x14x6.

3. Lapisan ketiga: Ini lagi-lagi adalah Lapisan Konvolusional dengan 16 peta fitur, ukuran 5x5, dan langkah 1. Perhatikan bahwa di lapisan ini, hanya 10 dari 16 peta fitur yang terhubung ke 6 peta fitur dari lapisan sebelumnya. Ini memberi kita beberapa keuntungan yang jelas:

a. Biaya komputasi lebih rendah. Hal ini dikarenakan jumlah koneksi berkurang dari 240.000 menjadi 151.600.

b. Jumlah total parameter pelatihan untuk lapisan ini adalah 1516, bukan 2400.

c. Hal ini juga merusak simetri arsitektur, sehingga pembelajaran dalam jaringan menjadi lebih baik.

4. Lapisan keempat: Ini adalah lapisan pengumpulan dengan ukuran filter 2x2 dan langkah 2 dengan keluaran 5x5x16.

5. Lapisan kelima: Ini adalah Konvolusional yang Terhubung Penuh lapisan dengan 120 peta fitur dan ukuran masing-masing 1x1. Masing-masing dari 120 unit terhubung ke 400 node di lapisan sebelumnya.

6. Lapisan keenam: Ini adalah lapisan Fully Connected dengan 84 satuan.

7. Terakhir, lapisan keluaran adalah lapisan softmax dengan sepuluh nilai yang mungkin sesuai dengan setiap digit.

Ringkasan arsitektur LeNet-5 ditunjukkan pada Tabel 3-1.

Tabel 3-1. *Ringkasan seluruh arsitektur LeNet*

Layer	Operation	Feature Map	Input Size	Kernel Size	Stride	Activation function
Input		1	32x32			
1	Convolution	6	28x28	5x2	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x2	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x2	1	tanh
6	Fully Connected	-	84			tanh
Output	Fully Connected	-	10	-		softmax

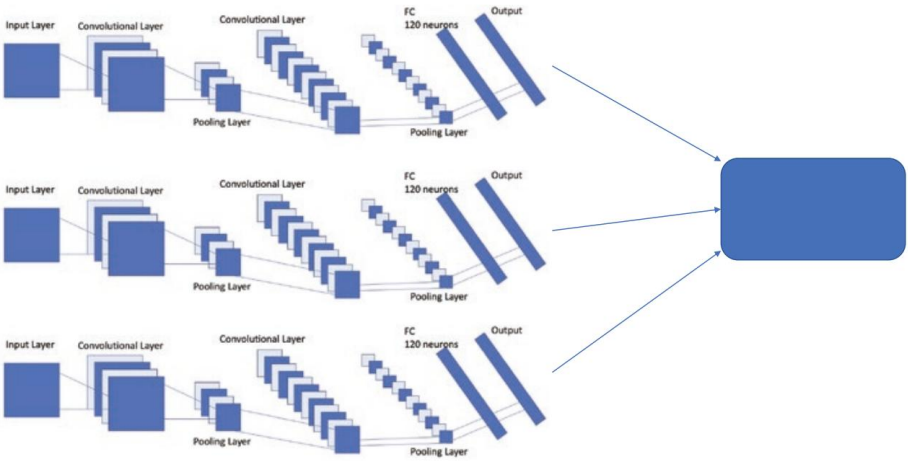
LeNet merupakan arsitektur yang kecil dan sangat mudah dipahami. Namun, arsitektur ini cukup besar dan matang untuk menghasilkan hasil yang baik. Arsitektur ini juga dapat dijalankan pada CPU. Pada saat yang sama, sebaiknya Anda menguji solusi dengan arsitektur yang berbeda dan menguji akurasinya untuk memilih yang terbaik.

Kami memiliki arsitektur LeNet yang ditingkatkan yang akan dibahas berikutnya.

3.7 Arsitektur LeNet-4 yang ditingkatkan

Boosting adalah teknik ensemble yang menggabungkan pembelajar yang lemah menjadi pembelajar yang kuat dengan terus meningkatkan kemampuan dari iterasi sebelumnya. Dalam Boosted LeNet-4, output dari arsitektur ditambahkan, dan yang memiliki nilai tertinggi adalah kelas yang diprediksi.

Arsitektur LeNet-4 yang ditingkatkan ditunjukkan pada Gambar 3-4.



Gambar 3-4. *Arsitektur LeNet-4 yang ditingkatkan menggabungkan pembelajar yang lemah untuk melakukan perbaikan. Hasil akhir jauh lebih akurat dan tangguh*

Pada arsitektur jaringan yang digambarkan sebelumnya, kita dapat melihat bagaimana kelemahan pelajar digabungkan untuk membuat solusi prediktif yang kuat.

LeNet adalah yang pertama dari beberapa arsitektur yang menjadi populer dan digunakan untuk memecahkan masalah Deep Learning. Dengan kemajuan dan penelitian yang lebih lanjut, kami telah mengembangkan algoritma yang canggih, tetapi LeNet masih memiliki tempat yang istimewa.

Sekarang saatnya bagi kita untuk membuat solusi pertama menggunakan arsitektur LeNet.

3.8 Membuat model klasifikasi gambar menggunakan LeNet

Setelah kita memahami arsitektur LeNet, sekarang saatnya mengembangkan kasus penggunaan yang sebenarnya. Kita akan mengembangkan dua kasus penggunaan menggunakan arsitektur LeNet. LeNet adalah arsitektur yang sederhana sehingga kode dapat dikompilasi pada CPU Anda sendiri. Kita membuat sedikit penyesuaian pada arsitektur dalam hal fungsi aktivasi menggunakan fungsi Max Pooling alih-alih Average Pooling dan seterusnya.

Penggabungan **Info** Max mengekstraksi fitur-fitur penting dibandingkan dengan penggabungan rata-rata. Penggabungan rata-rata menghaluskan gambar, dan karenanya fitur-fitur yang tajam mungkin tidak teridentifikasi.

Sebelum kita mulai dengan kode, ada pengaturan kecil yang harus kita ketahui. Posisi saluran dalam tensor menentukan bagaimana kita harus membentuk ulang data kita. Hal ini dapat diamati pada langkah 8 dari studi kasus berikut.

Setiap gambar dapat direpresentasikan berdasarkan tinggi, lebar, dan jumlah saluran atau jumlah saluran, tinggi, dan lebar. Jika saluran berada di posisi pertama dari array input, kami membentuk ulang menggunakan kondisi channels_first. Artinya saluran berada di posisi pertama dalam tensor (array n-dimensi). Dan sebaliknya berlaku untuk channels_last.

3.9 Klasifikasi MNIST menggunakan LeNet

Kasus penggunaan ini merupakan kelanjutan dari dataset MNIST yang kami gunakan di bab sebelumnya. Kode tersebut diperiksa di tautan GitHub yang diberikan di awal bab

1. Pertama, impor semua pustaka yang diperlukan.

```
impor keras
dari keras.optimizers impor SGD
dari sklearn.preprocessing impor
LabelBinarizer dari sklearn.model_selection impor
train_test_split dari sklearn.metrics impor classification_report
dari sklearn impor dataset

dari keras impor backend sebagai K
impor matplotlib.pyplot sebagai plt
impor numpy sebagai np
```

2. Selanjutnya kita import datasetnya lalu kita import serangkaian lapisan dari Keras.

```
dari keras.datasets import mnist ## Kumpulan data diimpor di
sini
dari keras.models impor Sequential
dari keras.layers.convolutional impor Conv2D
dari keras.layers.convolutional impor MaxPooling2D

dari keras.layers.core impor Aktivasi
dari keras.layers.core impor Ratakan
dari keras.layers.core impor Padat
dari keras impor backend sebagai K
```

3. Tentukan hiperparameter. Langkah ini mirip dengan yang ada di bab sebelumnya di mana kita mengembangkan MNIST dan klasifikasi anjing/kucing.

```
baris_gambar, kolom_gambar = 28, 28
ukuran_batch = 256
jumlah_kelas = 10
zaman = 10
```

4. Muat dataset sekarang. MNIST adalah dataset yang ditambahkan secara default di perpustakaan.

```
(kereta_x, kereta_y), (uji_x, uji_y) = mnist.muat_data()
```

5. Ubah data gambar menjadi float dan kemudian normalisasikan.

```
x_train = x_train.astype('float32') x_uji =
x_uji.astype('float32') x_train /= 255
x_uji /= 255
```

6. Mari cetak bentuk data kereta dan data uji kita.

```
cetak('bentuk kereta x:', bentuk_kereta x)
cetak(bentuk_kereta x[0], 'sampel kereta')
cetak(x_test.shape[0], 'sampel uji')
```

7. Pada blok kode berikutnya, kita mengonversi variabel kita menjadi pengkodean one-hot. Kami menggunakan Keras untuk_ metode kategoris untuk itu.

```
y_train = keras.utils.to_categorical(y_train, jumlah_kelas)

y_test = keras.utils.to_categorical(y_test, jumlah_kelas)
```

Tip Kami menggunakan pernyataan cetak untuk menganalisis output pada setiap langkah. Pernyataan ini memungkinkan kami untuk melakukan debug pada tahap selanjutnya jika diperlukan.

8. Mari kita bentuk ulang data kita sesuai dengan itu.

```
jika K.image_data_format() == 'channels_first':
    x_train = x_train.bentuk_ulang(x_train.bentuk[0], 1, baris_gambar,
    kolom_gambar)
    x_test = x_test.reshape(x_test.shape[0], 1, baris_gambar,
    kolom_gambar)
    input_shape = (1, baris_gambar, kolom_gambar)
kalau tidak:
    x_train = x_train.bentuk_ulang(x_train.bentuk[0], baris_gambar,
    kolom_gambar, 1)
    x_test = x_test.reshape(x_test.shape[0], baris_gambar,
    kolom_gambar, 1)
    input_shape = (baris_gambar, kolom_gambar, 1)
```

Sekarang saatnya membuat model kita!

9. Mulailah dengan menambahkan lapisan sekuensial, diikuti oleh Lapisan Konvolusional dan lapisan Pengumpulan Maksimum.

```
model = Berurut()
model.tambahkan(Conv2D(20, (5, 5),
padding="sama",bentuk_input=bentuk_input))
model.tambahkan(Aktivasi("relu")) model.
tambahkan(MaxPooling2D(ukuran_pool=(2, 2),
langkah=(2, 2)))
```

10. Sekarang kita akan menambahkan beberapa lapisan
    Lapisan konvolusional, lapisan Max Pooling, dan lapisan
    untuk meratakan data.

```
model.add(Conv2D(50, (5, 5), padding="sama"))
model.add(Activation("relu")) model.
tambahkan(MaxPooling2D(ukuran_pool=(2, 2),
langkah=(2, 2)))
model.tambahkan(Ratakan()) model.tambahkan(Padat(500))
model.tambahkan(Aktivasi("relu"))
```

11. Kita tambahkan Lapisan Padat diikuti oleh lapisan softmax.
    Softmax digunakan untuk model klasifikasi. Setelah itu, kami
    mengkompilasi model kami.

```
model.tambahkan(Dense(num_classes)) model.
tambahkan(Aktivasi("softmax"))
model.compile(kerugian=keras.kerugian.kategori_
crossentropy, pengoptimal=keras.optimizers.
Adadelta(), metrik=['akurasi'])
```

    Model siap untuk dilatih dan disesuaikan.

12. Kita dapat melihat dampak per epoch pada akurasi
    dan kerugian. Kami menganjurkan Anda untuk mencoba
    berbagai variasi hiperparameter seperti epoch dan
    ukuran batch. Bergantung pada hiperparameter, jaringan akan
    memerlukan waktu untuk memproses.

```
theLeNetModel = model.fit(kereta_x, kereta_y,
ukuran_batch=ukuran_batch,
zaman=zaman,
verbose=1, validasi_data=(uji_x, uji_y))
```

Di sini, kita dapat menganalisis bagaimana kerugian dan akurasi bervariasi pada setiap zaman.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 36s 607us/step - loss: 0.2736 - acc: 0.9127 - val_loss: 0.1051 - val_a
cc: 0.9649
Epoch 2/10
60000/60000 [==============================] - 37s 622us/step - loss: 0.0590 - acc: 0.9813 - val_loss: 0.0490 - val_a
cc: 0.9835
Epoch 3/10
60000/60000 [==============================] - 37s 614us/step - loss: 0.0387 - acc: 0.9879 - val_loss: 0.0939 - val_a
cc: 0.9671
Epoch 4/10
60000/60000 [==============================] - 37s 625us/step - loss: 0.0285 - acc: 0.9910 - val_loss: 0.0267 - val_a
cc: 0.9905
Epoch 5/10
60000/60000 [==============================] - 37s 615us/step - loss: 0.0215 - acc: 0.9933 - val_loss: 0.0305 - val_a
cc: 0.9896
Epoch 6/10
60000/60000 [==============================] - 37s 614us/step - loss: 0.0164 - acc: 0.9949 - val_loss: 0.0228 - val_a
cc: 0.9920
Epoch 7/10
60000/60000 [==============================] - 37s 614us/step - loss: 0.0136 - acc: 0.9955 - val_loss: 0.0236 - val_a
cc: 0.9918
Epoch 8/10
60000/60000 [==============================] - 37s 616us/step - loss: 0.0106 - acc: 0.9969 - val_loss: 0.0279 - val_a
cc: 0.9909
Epoch 9/10
60000/60000 [==============================] - 37s 617us/step - loss: 0.0082 - acc: 0.9976 - val_loss: 0.0246 - val_a
cc: 0.9917
Epoch 10/10
60000/60000 [==============================] - 37s 620us/step - loss: 0.0062 - acc: 0.9983 - val_loss: 0.0316 - val_a
cc: 0.9907
```

Setelah sepuluh kali periode, akurasi validasinya adalah 99,07%.

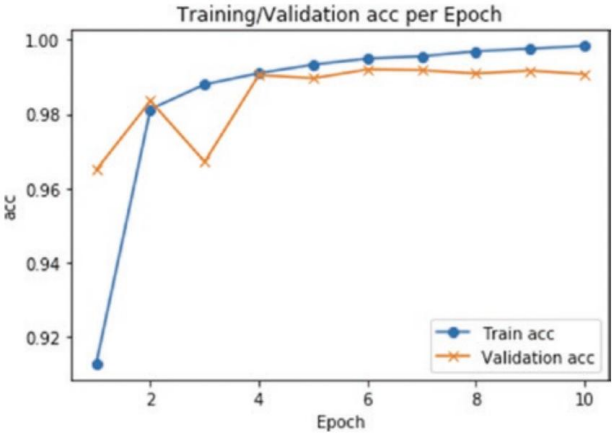
Sekarang, mari kita visualisasikan hasilnya.

13. Kita akan memplot akurasi pelatihan dan pengujian di blok kode berikutnya.

```
impor matplotlib.pyplot sebagai plt
f, ax = plt.subplot()
ax.plot([None] + theLeNetModel.history['acc'], 'o-') ax.plot([None]
+ theLeNetModel.history['val_acc'], 'x-') ax.legend(['Akun kereta',
'Akun validasi'], loc = 0) ax.set_title('Akun pelatihan/validasi
per Epoch') ax.set_xlabel('Epoch')

ax.set_ylabel('akun')
```

Pada grafik di Gambar 3-5, kita dapat melihat bahwa dengan setiap epoch berikutnya, parameter akurasi masing-masing untuk pelatihan dan validasi terus meningkat. Setelah epoch 7/8, akurasi menjadi stabil. Kita dapat mengujinya dengan nilai yang berbeda hiperparameter.



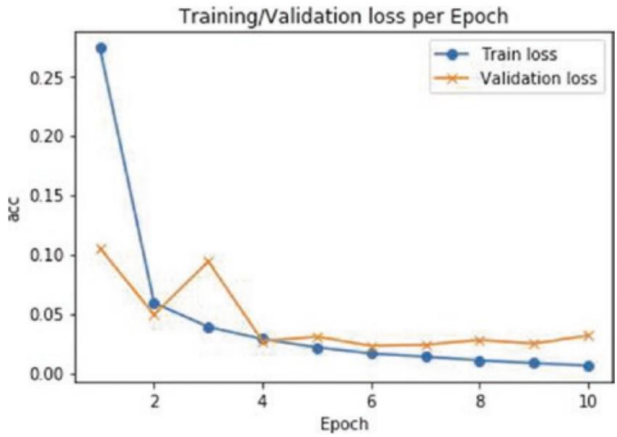
Gambar 3-5. *Akurasi pelatihan dan validasi ditunjukkan di sini. Setelah epoch 7/8, akurasinya telah stabil*

14. Mari kita menganalisis kerugiannya:

```
impor matplotlib.pyplot sebagai plt f,
kapak = plt.subplot()
ax.plot([None] + theLeNetModel.history['loss'], 'o-') ax.plot([None]
+ theLeNetModel.history['val_loss'], 'x-') ax.legend(['Kerugian
pelatihan', 'Kerugian validasi'], loc = 0) ax.set_title('Kerugian
pelatihan/validasi per Epoch') ax.set_xlabel('Epoch')

ax.set_ylabel('akun')
```

Pada grafik di Gambar 3-6, kita dapat melihat bahwa pada setiap periode berikutnya, masing-masing ukuran kerugian untuk pelatihan dan validasi terus menurun. Setelah epoch 7/8, akurasinya stabil. Kita dapat mengujinya dengan nilai-nilai hiperparameter yang berbeda.



Gambar 3-6. *Kerugian pelatihan dan validasi ditunjukkan di sini. Setelah 7/8 epoch, kerugian telah stabil dan tidak banyak pengurangan yang diamati.*

Bagus! Sekarang kita memiliki model LeNet yang berfungsi untuk mengklasifikasikan gambar.

Dalam latihan ini, kami melatih model klasifikasi gambar menggunakan Arsitektur LeNet-5.

Info Gunakan Google Colaboratory jika Anda menghadapi tantangan dengan efisiensi komputasi.

3.10 Identifikasi rambu lalu lintas Jerman menggunakan LeNet

Kasus penggunaan kedua adalah identifikasi rambu lalu lintas Jerman. Ini dapat digunakan dalam solusi mengemudi otomatis.

Dalam kasus penggunaan ini, kita akan membangun model Pembelajaran Mendalam menggunakan arsitektur LeNet-5.

1. Kami akan mengikuti proses serupa di seluruh buku, yaitu mengimpor pustaka terlebih dahulu.

```
impor keras
dari keras.optimizers impor SGD
dari sklearn.preprocessing impor LabelBinarizer

dari sklearn.model_selection impor train_test_split
dari sklearn.metrics impor laporan klasifikasi
dari sklearn impor kumpulan data
dari keras impor backend sebagai K
impor matplotlib.pyplot sebagai plt
impor numpy sebagai np
```

2. Impor pustaka Keras beserta semua paket yang diperlukan untuk membuat plot.

```
dari keras.models impor Sequential
dari keras.layers.convolutional impor Conv2D
dari keras.layers.convolutional impor
MaxPooling2D
dari keras.layers.core impor Aktivasi
dari keras.layers.core impor Ratakan
dari keras.layers.core impor Padat
dari keras impor backend sebagai K
```

3. Kemudian impor pustaka umum seperti numpy, matplotlib, os, OpenCV, dan seterusnya.

```
impor glob
impor pandas sebagai pd
impor matplotlib impor
matplotlib.pyplot sebagai plt impor acak
impor
matplotlib.image sebagai mpimg impor cv2
impor os dari

sklearn.model_selection impor train_test_split dari sklearn.metrics
impor
chaotic_matrix dari sklearn.utils impor shuffle impor
peringatan dari skimage impor exposure #
Muat data yang
diawetkan impor pickle %matplotlib
inline

matplotlib.style.use('ggplot') %config

InlineBackend.figure_format = 'retina'
```

4. Kumpulan data disediakan sebagai file pickle dan disimpan sebagai file train.p dan test.p. Kumpulan data dapat diunduh dari Kaggle di www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign .

```
file_pelatihan = "train.p"
file_pengujian = "test.p"
```

5. Buka file dan simpan data di dalam kereta dan variabel uji.

```
dengan open(training_file, mode='rb') sebagai f: train =
pickle.load(f)
dengan open(testing_file, mode='rb') sebagai f: test =
pickle.load(f)
```

6. Membagi dataset menjadi data uji dan data latih. Kami telah mengambil Ukuran tesnya 4000 di sini, tetapi Anda bebas mencoba ukuran tes yang berbeda.

```
X, y = kereta['fitur'], kereta['label']
x_kereta, x_sah, y_kereta, y_sah = kereta_
test_split(X, y, stratifikasi=y,
ukuran_uji=4000, status_acak=0)
x_test,y_test=tes['fitur'],tes['label']
```

7. Mari kita lihat beberapa contoh file gambar di sini. Kami telah menggunakan fungsi acak untuk memilih gambar acak; jadi, jangan khawatir jika hasil keluaran Anda tidak sama dengan hasil keluaran kami.

```
gambar, sumbu = plt.subplots(2,5, ukuran gambar=(15, 4))
gambar.subplot_sesuaikan(hspace = .2, wspace = .001)
sumbu = sumbu.ravel()
untuk i dalam rentang
    (10): indeks = acak.randint (0, len (x_train))
    gambar = x_train [indeks]
    axiss [i]. axis ('off') axiss
    [i]. imshow (gambar)
    sumbu[i].set_judul( y_train[indeks])
```

Berikut output seperti yang ditunjukkan pada Gambar 3-7.

Gambar 3-7. *Beberapa contoh kumpulan data klasifikasi rambu lalu lintas Jerman ditunjukkan di sini*

8. Selanjutnya, mari kita pilih hiperparameter kita. Jumlah kelas yang berbeda adalah 43. Kita telah mengambil sepuluh epoch untuk memulai, tetapi kami menganjurkan Anda untuk memeriksa kinerja dengan nilai epoch yang berbeda. Hal yang sama juga berlaku untuk ukuran batch.

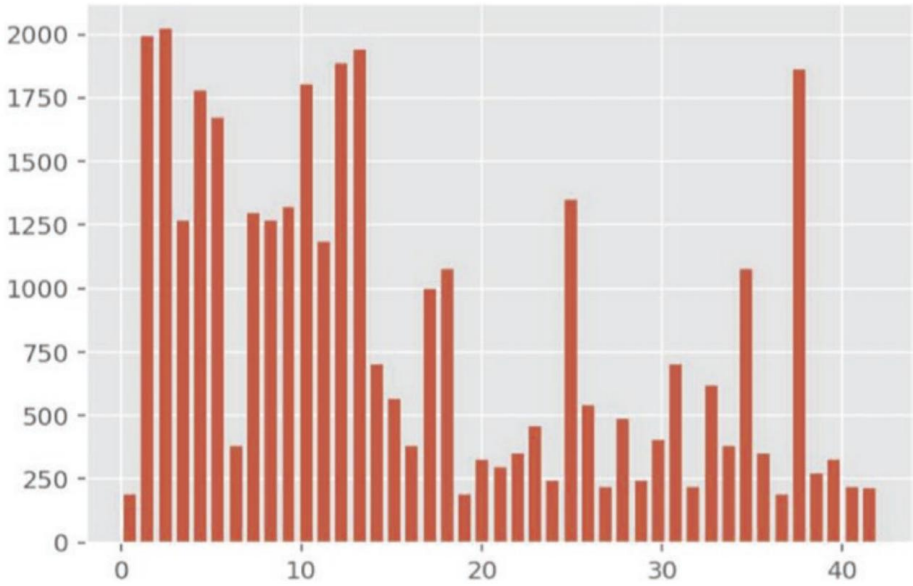
```
baris_gambar, kolom_gambar = 32, 32
ukuran_batch = 256
jumlah_kelas = 43
zaman = 10
```

9. Sekarang, kita akan melakukan beberapa analisis data eksploratif. Hal ini dilakukan untuk melihat bagaimana tampilan kumpulan gambar kita dan bagaimana distribusi frekuensi berbagai kelas dalam histogram.

```
histogram, bins = np.histogram(y_train, bins=num_classes)
lebar_ruang = 0.7 * (
tempat sampah[1] - tempat_sampah[0])
```

```
tengah = (tempat_sampah[:-1] + tempat_sampah[1:]) / 2
plt.bar(tengah, histogram, align='tengah',
lebar=lebar_yang_diinginkan) plt.show()
```

Outputnya ditunjukkan pada Gambar 3-8. Kita dapat mengamati bahwa ada perbedaan dalam jumlah contoh kelas. Beberapa kelas terwakili dengan sangat baik, sementara beberapa lainnya tidak. Dalam solusi dunia nyata, kita ingin memiliki kumpulan data yang seimbang. Kami membahasnya lebih lanjut di Bab 8 buku ini.



Gambar 3-8. *Distribusi frekuensi berbagai kelas. Beberapa kelas memiliki lebih banyak contoh, sementara beberapa kelas tidak memiliki banyak representasi. Idealnya, kita harus mengumpulkan lebih banyak data untuk kelas yang kurang terwakili*

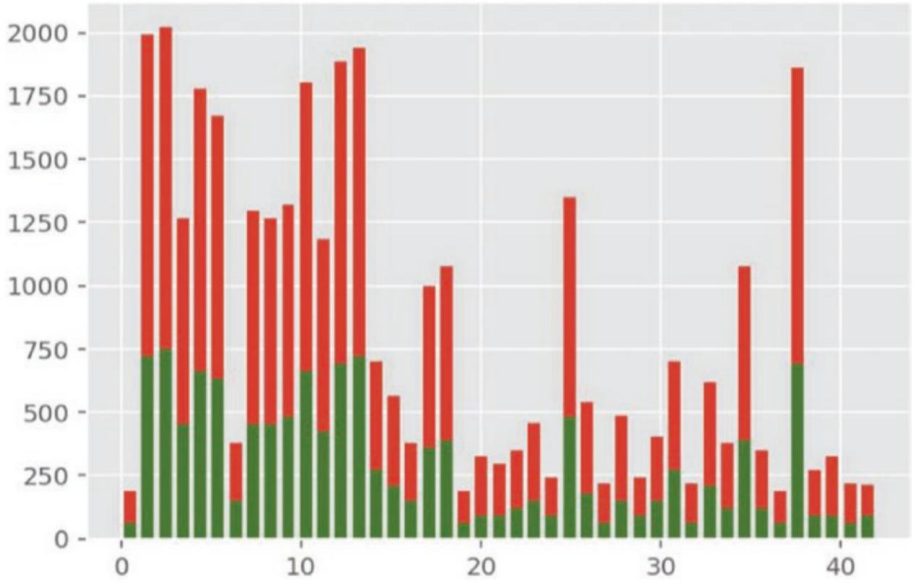
10. Sekarang, mari kita periksa bagaimana distribusinya di antara berbagai kelas. Ini adalah fungsi histogram reguler dari pustaka NumPy.

```
train_hist, train_bins = np.histogram(kereta_y,
bins=jumlah_kelas)
test_hist, test_bins = np.histogram(y_test,
bins=num_classes)
lebar_kereta = 0,7 * (kereta_bins[1] - kereta_bins[0]) pusat_kereta
= (kereta_bins[:-1] + kereta_bins[1:]) / 2 lebar_uji = 0,7 *
(kereta_bins[1] - kereta_bins[0]) pusat_uji = (kereta_bins[:-1]
+ kereta_bins[1:]) / 2
```

11. Sekarang, buat histogramnya; warnanya diatur menjadi merah dan hijau untuk dataset latih dan uji.

```
plt.bar(pusat_kereta, riwayat_kereta,
align='pusat', color='merah', width=lebar_kereta)
plt.bar(pusat_uji, riwayat_uji, align='pusat', warna='hijau',
lebar=lebar_uji)
plt.tampilkan()
```

Berikut outputnya seperti yang ditunjukkan pada Gambar 3-9.



Gambar 3-9. *Distribusi frekuensi berbagai kelas dan didistribusikan antara dataset train dan test. Train ditunjukkan dengan warna merah, sedangkan test digambarkan dengan warna hijau.*

Mari kita menganalisis distribusinya di sini; lihat perbedaan dalam proporsi kereta vs. pengujian pada histogram sebelumnya.

12. Ubah data gambar menjadi float dan kemudian normalisasikan.

```
x_train = x_train.astype('float32') x_uji =
x_uji.astype('float32') x_train /= 255

x_uji /= 255
cetak('bentuk kereta x:', bentuk_kereta x)
cetak(bentuk_kereta x[0], 'sampel kereta')
cetak(bentuk_uji_x[0], 'sampel uji')
```

Jadi, kita memiliki 35.209 titik data pelatihan dan 12.630 titik data pengujian. langkah selanjutnya, mengonversi vektor kelas ke matriks kelas biner. Ini mirip dengan langkah-langkah pada contoh terakhir saat kami mengembangkan klasifikasi MNIST.

```
y_train = keras.utils.to_categorical(y_train, jumlah_kelas) y_test = keras.utils.to_categorical(y_test, jumlah_kelas)
```

Blok kode berikut sama dengan yang dijelaskan dalam klasifikasi MNIST yang dikembangkan sebelumnya. Di sini, channels_first berarti bahwa saluran berada pada posisi pertama dalam array. Dan kami mengubah input_shape tergantung pada posisi channels_first.

```
jika K.image_data_format() == 'channels_first': input_shape = (1,
    baris_gambar, kolom_gambar) yang lain:

    input_shape = (baris_gambar, kolom_gambar, 1)
```

Sekarang mari kita mulai membuat Arsitektur Jaringan Syaraf Tiruan. Langkah-langkahnya serupa dengan kasus penggunaan sebelumnya.

13. Tambahkan lapisan sekuensial diikuti oleh lapisan Konvolusional lapisan.

```
model = Sequential() model.tambahkan(Conv2D(16,(3,3),
bentuk_input=(32,32,3)))
```

14. Tambahkan lapisan Pooling diikuti oleh lapisan Convolutional dan seterusnya.

```
model.tambahkan(Aktivasi("relu")) model.
tambahkan(MaxPooling2D(ukuran_pool=(2, 2),
langkah=(2, 2)))
```

```
model.add(Conv2D(50, (5, 5), padding="sama"))
model.add(Activation("relu")) model.
tambahkan(MaxPooling2D(ukuran_pool=(2,
2), langkah=(2, 2)))
model.tambahkan(Ratakan()) model.tambahkan(Padat(500))
model.tambahkan(Aktivasi("relu"))
model.tambahkan(Dense(num_classes)) model.
tambahkan(Aktivasi("softmax"))
```

15. Mari kita cetak ringkasan modelnya.

```
model.ringkasan()
```

Berikut outputnya.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 30, 30, 16)        448
_________________________________________________________________
activation_13 (Activation)   (None, 30, 30, 16)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 15, 15, 16)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 15, 15, 50)        20050
_________________________________________________________________
activation_14 (Activation)   (None, 15, 15, 50)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 50)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2450)              0
_________________________________________________________________
dense_7 (Dense)              (None, 500)               1225500
_________________________________________________________________
activation_15 (Activation)   (None, 500)               0
_________________________________________________________________
dense_8 (Dense)              (None, 43)                21543
_________________________________________________________________
activation_16 (Activation)   (None, 43)                0
=================================================================
Total params: 1,267,541
Trainable params: 1,267,541
Non-trainable params: 0
```

16. Model siap dikompilasi; mari melatihnya.

```
model.compile(kerugian=keras.losses.categorical_
crossentropy, optimizer=keras.optimizers.
Adadelta(), metrik=['akurasi']) theLeNetModel
= model.fit(kereta_x, kereta_y, ukuran_batch=ukuran_batch,
epoch=periode, verbose=1,
data_validasi=(uji_x,
uji_y))
```

Berikut outputnya seperti yang ditunjukkan pada Gambar 3-10.

```
WARNING:tensorflow:From /Users/vaibhavverdhan/anaconda3/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.p
y:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 35209 samples, validate on 12630 samples
Epoch 1/10
35209/35209 [==============================] - 22s 632us/step - loss: 2.2025 - acc: 0.3971 - val_loss: 1.4474 - val_a
cc: 0.5604
Epoch 2/10
35209/35209 [==============================] - 22s 631us/step - loss: 0.5801 - acc: 0.8291 - val_loss: 0.6247 - val_a
cc: 0.8179
Epoch 3/10
35209/35209 [==============================] - 22s 629us/step - loss: 0.1937 - acc: 0.9501 - val_loss: 0.4696 - val_a
cc: 0.8833
Epoch 4/10
35209/35209 [==============================] - 22s 623us/step - loss: 0.0956 - acc: 0.9770 - val_loss: 0.4750 - val_a
cc: 0.8850
Epoch 5/10
35209/35209 [==============================] - 23s 651us/step - loss: 0.0564 - acc: 0.9876 - val_loss: 0.5317 - val_a
cc: 0.8864
Epoch 6/10
35209/35209 [==============================] - 23s 642us/step - loss: 0.0364 - acc: 0.9925 - val_loss: 0.4336 - val_a
cc: 0.9140
Epoch 7/10
35209/35209 [==============================] - 22s 630us/step - loss: 0.0251 - acc: 0.9953 - val_loss: 0.4621 - val_a
cc: 0.9138
Epoch 8/10
35209/35209 [==============================] - 22s 631us/step - loss: 0.0186 - acc: 0.9964 - val_loss: 0.4819 - val_a
cc: 0.9117
Epoch 9/10
35209/35209 [==============================] - 22s 628us/step - loss: 0.0121 - acc: 0.9981 - val_loss: 0.5061 - val_a
cc: 0.9124
Epoch 10/10
35209/35209 [==============================] - 22s 619us/step - loss: 0.0112 - acc: 0.9983 - val_loss: 0.5421 - val_a
cc: 0.9116
```

Gambar 3-10. *Pergerakan akurasi dan kerugian terhadap setiap epoch. Kita harus memperhatikan bagaimana akurasi meningkat dari epoch pertama ke epoch terakhir.*

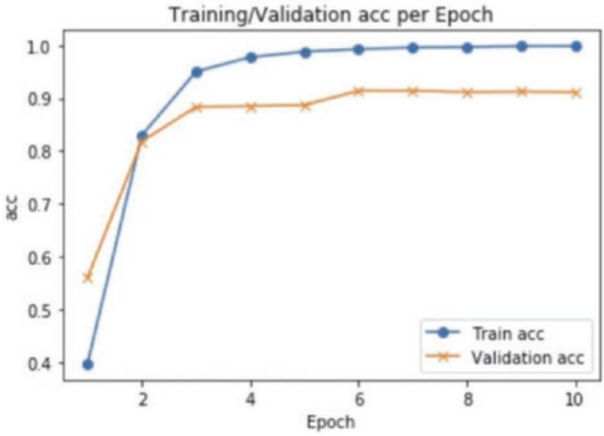
Setelah sepuluh periode, akurasi validasinya adalah 91,16%.

Mari kita visualisasikan hasilnya sekarang.

17. Pertama kita akan memplot akurasi pelatihan dan pengujian untuk jaringan.

```
impor matplotlib.pyplot sebagai plt
f, ax = plt.subplot()
ax.plot([None] + theLeNetModel.history['acc'], 'o-') ax.plot([None] +
theLeNetModel.history['val_acc'], 'x-') ax.legend(['Akun kereta', 'Akun
validasi'], loc = 0) ax.set_ title('Akun pelatihan/validasi per Epoch')
ax.set_xlabel('Epoch') ax.set_ylabel('acc')
```

Berikut hasil plotnya seperti ditunjukkan pada Gambar 3-11.



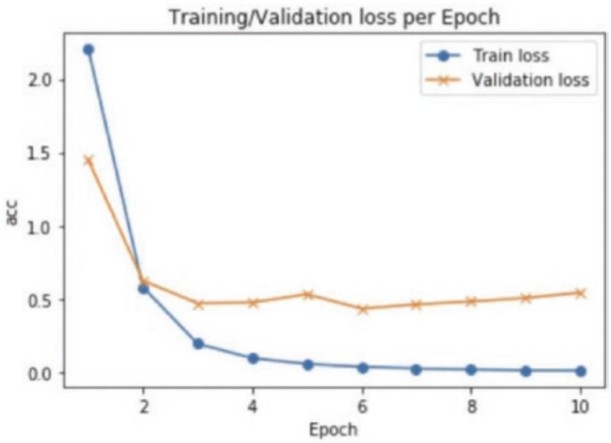
Gambar 3-11. *Akurasi pelatihan dan validasi ditunjukkan di sini. Setelah epoch 5/6, akurasinya telah stabil*

18. Mari kita plot kerugian untuk data pelatihan dan pengujian.

```
impor matplotlib.pyplot sebagai plt
f, ax = plt.subplot()
ax.plot([None] + theLeNetModel.history['loss'], 'o-') ax.plot([None]
+ theLeNetModel.history['val_loss'], 'x-')
ax.legend(['Kerugian pelatihan', 'Kerugian validasi'], loc = 0) ax.set_
title('Kerugian pelatihan/validasi per Epoch') ax.set_xlabel('Epoch')

ax.set_ylabel('akun')
```

Plot yang dihasilkan dapat dilihat pada Gambar 3-12.



Gambar 3-12. *Kerugian pelatihan dan validasi ditunjukkan di sini. Setelah 5/6 periode, kerugian telah stabil dan tidak banyak pengurangan yang diamati.*

Plot akurasi dan fungsi kerugian dibuat untuk mengukur kinerja model. Plot tersebut mirip dengan plot yang dikembangkan dalam model klasifikasi MNIST.

Info Semua parameter kinerja model ada di dalam LeNetModel.model atau LeNetModel.model.metrics.

Dalam contoh ini, kita mengambil satu langkah tambahan dan membuat Matriks Kebingungan untuk prediksi juga. Untuk ini, pertama-tama kita harus membuat prediksi atas set pengujian dan kemudian membandingkan prediksi dengan label gambar yang sebenarnya.

19. Buat prediksi menggunakan fungsi prediksi.

```
prediksi = theLeNetModel.model.predict (x_test)
```

20. Sekarang, mari kita buat Matriks Kebingungan. Ini adalah tersedia di pustaka scikit-learn.

```
dari sklearn.metrics impor kebingungan_matriks impor numpy
sebagai np
kebingungan = matriks_kebingungan(uji_y,
np.argmax(prediksi,sumbu=1))
```

21. Sekarang mari kita buat sebuah variabel yang disebut cm yang tidak ada apa-apa selain matriks kebingungan.

Jangan ragu untuk mencetaknya dan menganalisis hasilnya.

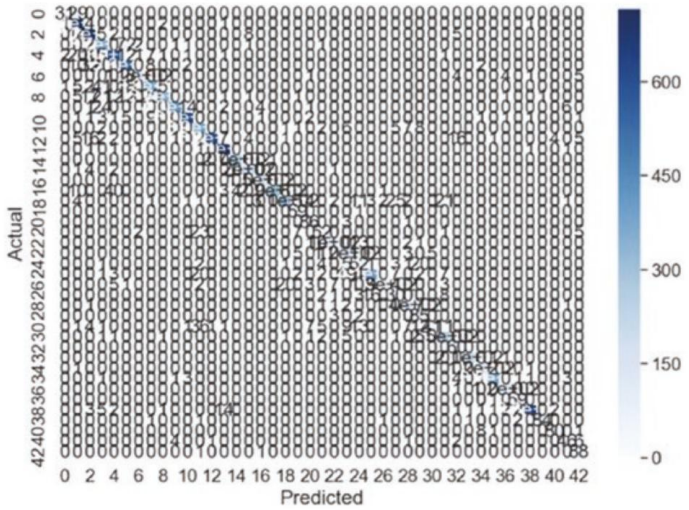
```
cm = matriks_kebingungan(uji_y,
np.argmax(prediksi,sumbu=1))
```

22. Sekarang mari kita mulai visualisasi untuk Kebingungan Matrix. Seaborn adalah pustaka lain bersama dengan matplotlib yang digunakan untuk visualisasi.

```
    impor seaborn sebagai sn
df_cm = pd.DataFrame(cm, kolom=np.unik (uji_y), indeks =
np.unik (uji_y))
```

```
df_cm.index.name = 'Aktual'
df_cm.columns.name = 'Diprediksi'
plt.gambar(ukuranfig = (10,7))
sn.set(font_scale=1.4)#untuk ukuran label
sn.heatmap(df_cm, cmap="Blues",
annot=Benar,annot_kws={"ukuran": 16})# ukuran font
```

Outputnya ditunjukkan sebagai berikut. Karena banyaknya dimensi, Confusion Matrix tidak terlihat jelas pada Gambar 3-13, jadi mari kita buat sedikit lebih baik pada blok kode berikutnya.



Gambar 3-13. *Matriks kebingungan ditampilkan, tetapi karena banyaknya dimensi, outputnya tidak terlalu jelas yang akan kami perbaiki pada gambar berikutnya.*

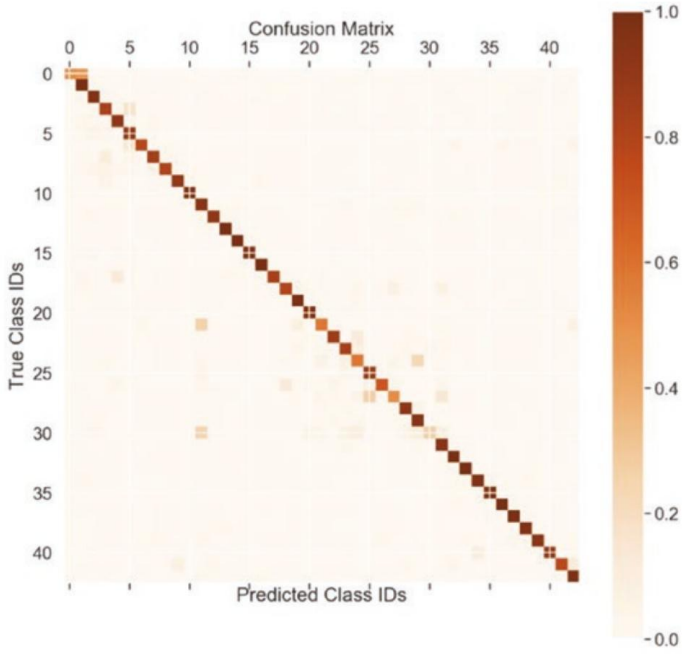
23. Di sini, kita memetakan Matriks Kebingungan lagi.

Harap dicatat bahwa kami telah mendefinisikan fungsi plot_ Confusion_matrix yang mengambil matriks kebingungan sebagai parameter input. Kemudian kita menggunakan pustaka matplotlib biasa dan fungsinya untuk memplot Matriks Kebingungan. Anda dapat menggunakan fungsi ini untuk solusi lainnya juga.

```
def plot_kebingungan_matriks(cm):
cm = [baris/jumlah(baris) untuk baris dalam cm]
gambar = plt.gambar(ukuran gambar=(10, 10))
ax = fig.tambahkan_subplot(111)
cax = ax.matshow(cm, cmap=plt.cm.Jeruk) gbr.
bilah warna (cax)
plt.title('Matriks Kebingungan') plt.
xlabel('ID Kelas yang Diprediksi') plt.ylabel('ID Kelas Sebenarnya')

plt.tampilkan()
    plot_matriks_kebingungan(cm)
```

Berikut grafik yang menunjukkan matriks kebingungan (Gambar 3-14).



Gambar 3-14. *Matriks kebingungan dihasilkan untuk semua kelas. Untuk beberapa kelas, hasilnya tidak terlalu bagus. Sebaiknya cari kesalahan klasifikasi dan analisis penyebabnya.*

Kita dapat melihat di sini bahwa untuk beberapa kelas kita memperoleh hasil yang bagus. Anda disarankan untuk menganalisis hasil dan mengulanginya dengan hiperparameter untuk melihat dampaknya. Pengamatan yang memiliki kesalahan klasifikasi harus dianalisis untuk mengetahui alasannya. Misalnya, dalam kasus klasifikasi digit, suatu algoritme mungkin menjadi bingung antara 1 dan 7. Oleh karena itu, setelah menguji model, kita harus mencari kesalahan klasifikasi yang dilakukan dan mencari tahu alasannya. Solusi yang mungkin adalah menghapus gambar yang membingungkan dari set data pelatihan. Meningkatkan kualitas gambar dan menambah kuantitas kelas yang salah diklasifikasikan juga dapat membantu mengatasi masalah tersebut.

Dengan ini, kita telah menyelesaikan dua studi kasus tentang klasifikasi gambar menggunakan LeNet. Kita telah sampai pada akhir bab ini. Anda dapat melanjutkan ke ringkasan sekarang.

3.11 Ringkasan

Arsitektur Jaringan Neural merupakan solusi yang cukup menarik dan ampuh untuk masalah visi komputer. Arsitektur ini memungkinkan kita untuk berlatih pada kumpulan data yang sangat besar dan berguna untuk mengidentifikasi gambar dengan benar. Kemampuan ini dapat digunakan untuk berbagai macam masalah di seluruh domain. Pada saat yang sama, penting untuk dicatat bahwa kualitas solusi sangat bergantung pada kualitas kumpulan data pelatihan. Ingat pepatah terkenal, sampah masuk, sampah keluar.

Kami mempelajari konsep konvolusional, pengumpulan maksimum, padding, dan sebagainya di bab terakhir dan mengembangkan solusi menggunakan CNN. Bab ini menandai dimulainya arsitektur Jaringan Neural yang disesuaikan. Arsitektur ini berbeda satu sama lain berdasarkan desainnya, yaitu jumlah lapisan, fungsi aktivasi, langkah, ukuran kernel, dan sebagainya. Lebih sering, kami menguji tiga dari empat arsitektur berbeda untuk membandingkan akurasinya.

Dalam bab ini, kami membahas arsitektur LeNet dan fokus pada LeNet-5. Kami mengembangkan dua kasus penggunaan dengan implementasi menyeluruh mulai dari pemuatan data hingga perancangan jaringan dan pengujian ketepatan.

Pada bab berikutnya, kita akan mempelajari arsitektur populer lainnya yang disebut VGGNet.

Sekarang, Anda seharusnya dapat menjawab pertanyaan dalam latihan ini!

LATIHAN ULASAN

1. Apa perbedaan antara berbagai versi LeNet?
2. Bagaimana kita mengukur distribusi akurasi dengan epoch?
3. Kami telah membahas dua kasus penggunaan dalam bab ini. Ulangi solusi yang sama dengan nilai hiperparameter yang berbeda. Buat fungsi kerugian dan distribusi akurasi untuk kasus penggunaan yang dilakukan di bab terakhir.

4. Ambil dataset yang digunakan pada bab terakhir dan uji dengan LeNet-5 untuk membandingkan hasilnya.

5. Unduh dataset klasifikasi Image Scene dari www.kaggle.com/puneet6060/intel-image- classification/version/2. Jalankan kode yang digunakan dalam Dataset klasifikasi lalu lintas Jerman untuk dataset ini.

6. Unduh dataset Linnaeus 5 dari http://chaladze.com/l5/. Berisi lima kelas – beri, burung, anjing, bunga, dan lainnya. Gunakan kumpulan data ini untuk membuat solusi berbasis CNN.

3.11.1 Bacaan lebih lanjut

1. Baca makalah “Convolutional Neural Network” Jaringan untuk pengenalan objek 3D menggunakan representasi volumetrik” di https://drive.google.com/drive/folder/1-5V1vj88-ANdRQJ5PpcAQkErvG7 lqzxs.

2. Baca makalah tentang penyakit Alzheimer klasifikasi menggunakan CNN di https://arxiv.org/abs/1603.08631.