Tugas Mata Kuliah

Pembelajaran Mesin Lanjut (EI7007)

Oleh,

Nama : Riyanto
NIM : 33221044
Tanggal dikumpulkan : 28 Maret 2022

Eksplorasi Hyperparameter MLP dan CNN

- **Bagian I : Multi Layer Perceptron (MLP)**

  Studi Kasus Pembelajaran Mendalam untuk Masalah Regresi: Prediksi Harga Perumahan Boston.

- **Bagian II: Convolutional Neural Network (CNN)**

  Studi Kasus Pembelajaran Mendalam untuk Masalah Klasifikasi: Pengenalan Tulisan Tangan Digit.

Bagian I

Multi Layer Perceptron (MLP)

Studi Kasus Pembelajaran Mendalam untuk Masalah Regresi: Prediksi Harga Perumahan Boston

Dalam tugas ini kita akan menemukan bagaimana mengembangkan dan mengevaluasi model jaringan saraf menggunakan Keras untuk masalah regresi. Tugas ini melakukan eksplorasi untuk menjawab pertanyaan-pertanyaan berikut:

- Berapa banyaknya hidden layer yang optimal?
- Berapa banyaknya hidden unit yang optimal di setiap hidden layer?
- Apa activation function di setiap layer sehingga hasilnya optimal?
- Dari semua pilihan optimizer, apa optimizer yang hasilnya optimal?
- Dari semua pilihan loss function, apa yang hasilnya optimal?

Dataset yang digunakan pada model prediktif ini sumbernya StatLib library tentang nilai perumahan di pinggiran kota Boston.

Berikut adalah fitur yang dipakai untuk prediksi harga rumah:

1. CRIM : tingkat kejahatan per kapita menurut kota
2. ZN : proporsi tanah perumahan yang dikategorikan untuk kavling lebih dari 25.000 sq.ft.(sekitar 2.300 meter persegi)
3. INDUS : proporsi acre bisnis non-ritel per kota
4. CHAS : Variabel dummy Sungai Charles (= 1 jika saluran membatasi sungai; 0 sebaliknya)
5. NOX : konsentrasi oksida nitrat (bagian per 10 juta)
6. RM : rata-rata jumlah kamar per hunian
7. USIA : proporsi unit yang ditempati oleh pemilik yang dibangun sebelum tahun 1940
8. DIS : jarak tertimbang (weighted distance) ke lima pusat pekerjaan Boston
9. RAD : indeks aksesibilitas ke jalan raya radial
10. PAJAK : tarif pajak properti nilai penuh per $10.000
11. PTRATIO : rasio murid-guru menurut kota
12. B : 1000(Bk - 0.63)^2 dimana Bk adalah proporsi kulit hitam menurut kota
13. LSTAT : % status penduduk yang lebih rendah

LOAD DATASET

```
# Load libraries
import numpy as np
import pandas as pd

# keras library
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor

# scikit functions
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# load dataset
dataframe = pd.read_csv("housing.csv", delim_whitespace=True, header=None)
dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:,0:13]
Y = dataset[:,13]
dataframe.head()     # preview the first 5 rows
```

Berikut adalah preview dari data yang digunakan sebagai input: (5 baris pertama saja)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Kolom 0 sampai 12 menunjukkan fitur yang dipakai untuk prediksi harga rumah seperti yang sudah dijelaskan di awal paragraph. Kolom ke 13 menunjukkan harga rumah yang digunakan untuk training dan validasi/testing.

Di bawah ini, kami mendefinisikan fungsi untuk membuat model dasar yang akan dievaluasi. Model dasar ini adalah model sederhana yang memiliki satu lapisan tersembunyi yang terhubung penuh, dengan jumlah neuron yang sama dengan atribut input (13). Jaringan menggunakan fungsi aktivasi penyearah untuk lapisan tersembunyi. Tidak ada fungsi aktivasi yang digunakan untuk lapisan keluaran karena ini merupakan masalah regresi, dan kami tertarik untuk memprediksi nilai numerik secara langsung tanpa transformasi.

Algoritma optimasi ADAM yang efisien digunakan dan *loss function mean square error* (mse) dioptimalkan. Ini akan menjadi metrik yang sama yang akan kita gunakan untuk mengevaluasi kinerja model. Ini adalah metrik yang diinginkan karena dengan mengambil akar kuadrat dari nilai kesalahan, itu memberi kita hasil yang dapat kita pahami secara langsung, dalam konteks masalah harga rumah (dalam satuan ribuan dolar).

BASELINE MODEL

```
# define base mode
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimator = KerasRegressor(build_fn=baseline_model, epochs=50, batch_size=
5, verbose=0)

kfold = KFold(n_splits=10)
results = cross_val_score(estimator, X, Y, cv=kfold)
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Baseline: -39.53 (26.27) MSE
```

Menstandardkan dataset untuk menaikkan performa

Pada dataset harga rumah Boston semua atribut input bervariasi dalam skalanya, karena mereka mengukur kuantitas yang berbeda.

BASELINE MODEL DENGAN DATASET STANDARDIZED

```
#evaluate baseline model with standardized dataset
np.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=baseline_model, epochs=5
0, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Standardized: -30.47 (29.83) MSE
```

Setelah dataset distandardkan hasilnya memberikan kinerja yang lebih baik dibandingkan model dasar tanpa data standar, yaitu mengurangi kesalahan sekitar 10 ribu dolar kuadrat. Nilai MSE sebelum data distandarkan: -38.32 Nilai MSE setelah data distandarkan: -27.17.

Eksplorasi Topologi Jaringan yang Lebih Dalam

Berapa banyaknya hidden layer yang optimal?

Salah satu cara untuk meningkatkan kinerja jaringan saraf adalah dengan menambahkan lebih banyak lapisan. Ini memungkinkan model untuk mengekstrak dan menggabungkan kembali fitur tingkat tinggi yang disematkan dalam data. Di bagian ini, kita akan mengevaluasi efek penambahan satu lapisan tersembunyi lagi ke model. Ini semudah mendefinisikan fungsi baru yang akan membuat model yang lebih dalam ini, disalin dari model dasar kami di atas. Kita kemudian dapat menyisipkan baris baru setelah lapisan tersembunyi pertama. Dalam hal ini dengan sekitar setengah jumlah neuron. Topologi jaringan kami sekarang terlihat seperti:

13 masukan -> [13 -> 6] -> 1 keluaran

Kita dapat mengevaluasi topologi jaringan ini dengan cara yang sama seperti di atas, sementara juga menggunakan standarisasi dataset yang ditunjukkan di atas untuk meningkatkan kinerja.

LARGE MODEL: 1 HIDDEN LAYER

```
# define the model
def larger_model_satu_hidden():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn = larger_model_satu_hidd
en, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Larger: -22.13 (23.89) MSE
```

LARGE MODEL: 2 HIDDEN LAYER

```
# define the model
def larger_model_dua_hidden():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

```
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn = larger_model_dua_hidde
n, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Larger: -21.51 (25.09) MSE
```

LARGE MODEL: 3 HIDDEN LAYER

```
# define the model
def larger_model_tiga_hidden():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn = larger_model_tiga_hidd
en, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
```

```
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Larger: -21.95 (23.79) MSE
```

LARGE MODEL: 4 HIDDEN LAYER

```
# define the model
def larger_model_empat_hidden():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn = larger_model_empat_hid
den, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Larger: -36.87 (56.45) MSE
```

Percobaan diperbanyak untuk mengamati pengaruh jumlah hidden layer terhadap kinerjanya. Berikut hasil penambahan jumlah hidden layer:

- 1 hideen layer -> MSE = -23.63 dengan nilai varian 26.74
- 2 hidden layer -> MSE = -21.69 dengan nilai varian 25.29
- 3 hidden layer -> MSE = -21.95 dengan nilai varian 23.79
- 4 hidden layer -> MSE = -36.87 dengan nilai varian 56.45

Dari hasil percobaan menunjukan bahwa 2 hidden layer menghasilkan nilai yang optimal. Penambahan jumlah hidden layer menunjukkan bahwa jaringan yang lebih dalam, yang juga lebih mahal secara komputasi tidak selalu lebih baik dalam hal kinerja dan akurasi.

Berapa banyaknya hidden unit yang optimal di setiap hidden layer?

Untuk mencari nilai yang optimal untuk jumlah unit dalam hidden layer maka perlu dilakukan percobaan variasi jumlah unit dalam hidden layer. Berikut variasi yang dilakukan:Jumlah unit 13, 16, 20

HIDDEN UNIT: 13,10,6

```
# define wider model
def wider_model_a():
    # create model
    model = Sequential()
    model.add(Dense(13, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(10, activation='relu', kernel_initializer='normal'))
    model.add(Dense(6, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_a, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
```

```
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -21.65 (27.47) MSE
```

HIDDEN UNIT: 16,12,8

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -20.04 (24.38) MSE
```

HIDDEN UNIT: 20,12,8

```
# define wider model
def wider_model_c():
    # create model
```

```
    model = Sequential()
    model.add(Dense(20, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_c, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -21.31 (26.11) MSE
```

Ketiga model tersebut menghasilkan nilai sebagai berikut: model_wider_a dengan jumlah unit 13,10,6 -> MSE: -21.65 model_wider_b dengan jumlah unit 16,12,8 -> MSE: -20.04 model_wider_c dengan jumlah unit 20,12,8 -> MSE: -21.31

Hasil tersebut menunjukkan bahwa variasi jumlah unit pada hidden layer tidak meningkatkan performa secara signifikan. Model_wider_b menghasilkan performa yang sedikit membaik.

Apa activation function di setiap layer sehingga hasilnya optimal?

Fungsi aktivasi yang sesuai untuk permasalahan ini dapat dicari dengan mencoba ReLu , Sigmoid dan Tanh

ACTIVATION FUNCTION: SIGMOID

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='sigmoid', kernel_initial
izer='normal'))
    model.add(Dense(12, activation='sigmoid', kernel_initializer='normal')
)
    model.add(Dense(8, activation='sigmoid', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -61.58 (53.77) MSE
```

ACTIVATION FUNCTION: TANH

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='tanh', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='tanh', kernel_initializer='normal'))
    model.add(Dense(8, activation='tanh', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -61.58 (53.77) MSE
```

Hasilnya menunjukkan bahwa fungsi aktivasi ReLu memberikan hasil yang terbaik. Activation function ReLu -> MSE = -20.04 Activation function Sigmoid -> MSE = -61.58 Activation function Tanh -> MSE = -79.28

Dari semua pilihan optimizer, apa optimizer yang hasilnya optimal?

OPTIMIZER: ADAM

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -20.62 (25.24) MSE
```

OPTIMIZER: SGD

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='SGD')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -83.83 (60.57) MSE
```

OPTIMIZER: ADAGARD

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='ADAGRAD')
```

```
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output:

```
Wider: -349.86 (194.00) MS
```

Hasil dari beberapa pilihan optimizer yang digunakan adalah sebagai berikut:

- SGD: MSE -83.83 (varian 60.57)
- ADAGRAD: MSE -349.86 (varian 194.00)
- RMSPROP: MSE -21.26 (varian 25.73)
- ADAM: MSE -20.62 (Varian 25.24)

Dari hasil tersebut nampak optimizer yang memberikan hasil terbaik adalah ADAM dan RMSPROP.

Dari semua pilihan loss function, apa yang hasilnya optimal?

Pilihan loss function dapat berupa:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber Loss

LOSS FUCTION: MEAN ABSOLUTE ERROR (MAE)

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_absolute_error', optimizer='ADAM')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MAE" % (results.mean(), results.std()))
```

Output:

```
Wider: -3.01 (1.30) MAE
```

LOSS FUCTION: HUBER LOSS

```
# define wider model
def wider_model_b():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, activation='relu', kernel_initialize
r='normal'))
    model.add(Dense(12, activation='relu', kernel_initializer='normal'))
    model.add(Dense(8, activation='relu', kernel_initializer='normal'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='huber_loss', optimizer='ADAM')
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model_b, epochs=50
, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) huber_loss" % (results.mean(), results.std()))
```

Output:

```
Wider: -2.51 (1.50) MAE
```

Hasil dari beberapa pilihan fungsi loss sebagai berikut:

- Mean Squared Error (MSE) : -20.62 (varian 25.24)
- Mean Absolute Error (MAE) : -3.01 (varian 1.30)
- Huber loss : -2.51 (varian 1.50)

Bagian II

Convolutional Neural Network (CNN)

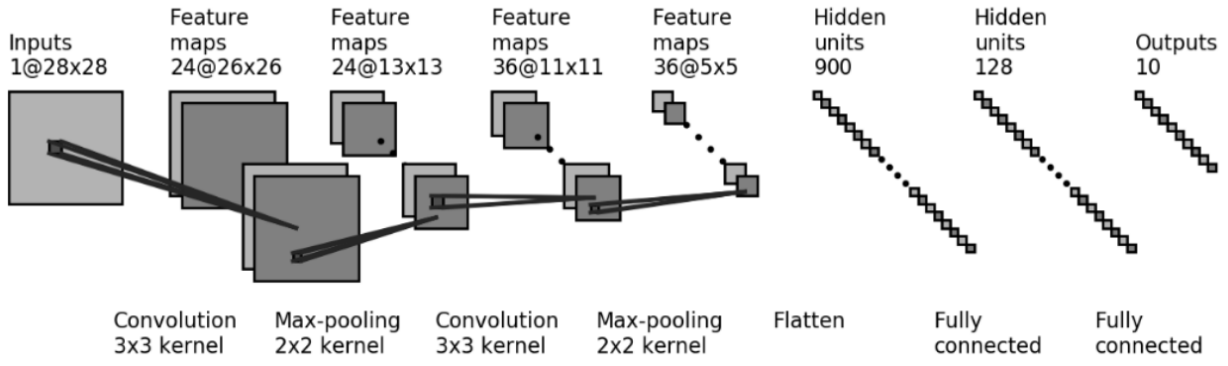
Studi Kasus Pembelajaran Mendalam untuk Masalah Klasifikasi: Pengenalan Tulisan Tangan Digit

Tugas ini melakukan eksplorasi model CNN untuk masalah Pengenalan Tulisan Tangan Digit (Digit Recognition). Dataset yang digunakan menggunakan dataset MNIST. Eksplorasi dilakukan untuk menjawab pertanyaan-pertanyaan berikut:

- Berapa banyaknya convolution layar yang optimal?
- Berapa ukuran filter yang optimal untuk setiap convolution layar?
- Berapa banyaknya filter yang optimal untuk setiap convolution layar?
- Berapa banyaknya hidden unit yang optimal pada bagian fully connected network?

Untuk mengetahui nilai yang paling optimal, harus dilakukan percobaan dengan membuat variasi nilai dari hypermarater yang sedang dieksplorasi dengan nilai hyperparameter lainnya dibuat tetap (fixed). Jika ada nilai hyperparameter lainnya yang sudah ditemukan pada eksplorasi sebelumnya, gunakan nilai hyperparameter optimal tsb pada eksplorasi berikutnya Nilai optimal diambil dari percobaan yang menghasilkan kinerja terbaik.

Kami akan menggunakan set data pelatihan dan pengujian yang sama seperti sebelumnya, dan melanjutkan dengan cara yang sama seperti jaringan kami yang sepenuhnya terhubung untuk menentukan dan melatih model CNN baru kami. Untuk melakukan ini, kita akan menjelajahi dua layer yang belum pernah kita temui sebelumnya: Anda dapat menggunakan keras.layers.Conv2D untuk mendefinisikan layer convolutional dan keras.layers.MaxPool2D untuk mendefinisikan layer pooling. Gunakan parameter yang ditunjukkan dalam arsitektur jaringan di atas untuk menentukan lapisan ini dan membangun model CNN.



MNIST DATASET

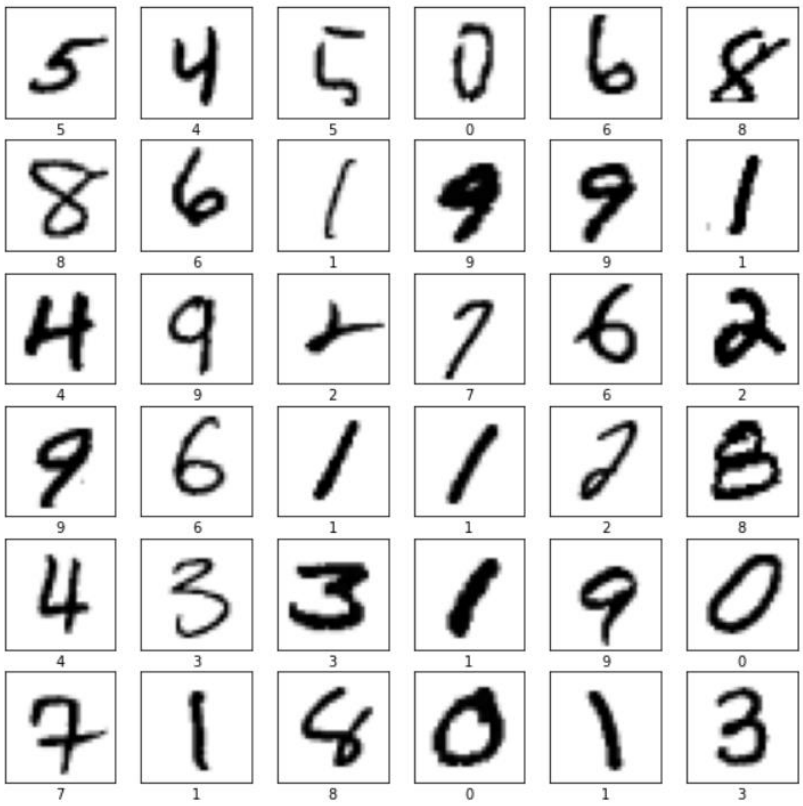
```
# Import Tensorflow 2.0
import tensorflow as tf
!pip install mitdeeplearning
import mitdeeplearning as mdl
import matplotlib.pyplot as plt
import numpy as np
import random
from tqdm import tqdm

# Check that we are using a GPU, if not switch runtimes
#   using Runtime > Change Runtime Type > GPU
assert len(tf.config.list_physical_devices('GPU')) > 0

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data
()
train_images = (np.expand_dims(train_images, axis=-
1)/255.).astype(np.float32)
train_labels = (train_labels).astype(np.int64)
test_images = (np.expand_dims(test_images, axis=-
1)/255.).astype(np.float32)
test_labels = (test_labels).astype(np.int64)

plt.figure(figsize=(10,10))
random_inds = np.random.choice(60000,36)
for i in range(36):
    plt.subplot(6,6,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    image_ind = random_inds[i]
    plt.imshow(np.squeeze(train_images[image_ind]), cmap=plt.cm.binary)
    plt.xlabel(train_labels[image_ind])
```

Berikut adalah contoh gambar dataset MNIST



MODEL DASAR CNN

```
def build_cnn_model():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=24, kernel_size=(3,3), activation=t
f.nn.relu),
        # tf.keras.layers.Conv2D('''TODO''')

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),
        # tf.keras.layers.MaxPool2D('''TODO''')

        # Define the second convolutional layer
        tf.keras.layers.Conv2D(filters=36, kernel_size=(3,3), activation=t
f.nn.relu),
        # tf.keras.layers.Conv2D('''TODO''')

        # Define the second max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),
        # tf.keras.layers.MaxPool2D('''TODO''')
```

```
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model()
# Initialize the model by passing some data through
cnn_model.predict(train_images[[0]])
# Print the summary of the layers in the model.
print(cnn_model.summary())
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 24)        240

 max_pooling2d (MaxPooling2D  (None, 13, 13, 24)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 36)        7812

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 36)         0
 2D)

 flatten (Flatten)           (None, 900)               0

 dense (Dense)               (None, 128)               115328

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 124,670
Trainable params: 124,670
Non-trainable params: 0
_________________________________________________________________
None
```

Pelatihan dan Pengujian model CNN

Kita dapat mendefinisikan fungsi loss, optimizer, dan metrik melalui metode kompilasi. Kompilasi model CNN dengan pengoptimal dan tingkat pembelajaran pilihan. Untuk melatih CNN kami menggunakan metode fit melalui Keras API. Selanjutnya melatih model, untuk mengevaluasi pada dataset uji menggunakan metode evaluasi:

PELATIHAN DAN PENGUJIAN

```
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Test accuracy: 0.9901999831199646
```

Eksplorasi CNN yang Lebih Dalam

Berapa banyaknya convolution layar yang optimal?

Dalam percobaan ini digunakan satu, dua dan tiga convolutionl layer untuk menunjukkan pengaruh jumlah konvolusional terhadap akurasi testing.

MODEL CNN: SATU KONVOLUSI

```
def build_cnn_model_satu_conv():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=24, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),
```

```
        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_satu_conv()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 6s 6ms/step - loss: 0.1858 -
accuracy: 0.9449
Epoch 2/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0520 -
accuracy: 0.9839
Epoch 3/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0359 -
accuracy: 0.9886
Epoch 4/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0279 -
accuracy: 0.9913
Epoch 5/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0218 -
accuracy: 0.9931
313/313 [==============================] - 1s 3ms/step - loss: 0.0318 -
accuracy: 0.9899
Test accuracy: 0.9898999929428101
```

MODEL CNN: DUA KONVOLUSI

```
def build_cnn_model_dua_conv():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=24, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        # Define the second convolutional layer
        tf.keras.layers.Conv2D(filters=36, kernel_size=(3,3), activation=t
f.nn.relu),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_dua_conv()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 6s 6ms/step - loss: 0.1858 -
accuracy: 0.9449
Epoch 2/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0520 -
accuracy: 0.9839
Epoch 3/5
```

```
938/938 [==============================] - 6s 6ms/step - loss: 0.0359 -
accuracy: 0.9886
Epoch 4/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0279 -
accuracy: 0.9913
Epoch 5/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0218 -
accuracy: 0.9931
313/313 [==============================] - 1s 3ms/step - loss: 0.0318 -
accuracy: 0.9899
Test accuracy: 0.9898999929428101
```

MODEL CNN: TIGA KONVOLUSI

```
def build_cnn_model_tiga_conv():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=24, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        # Define the second convolutional layer
        tf.keras.layers.Conv2D(filters=36, kernel_size=(3,3), activation=t
f.nn.relu),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_tiga_conv()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
```

```
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 6s 6ms/step - loss: 0.1858 -
accuracy: 0.9449
Epoch 2/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0520 -
accuracy: 0.9839
Epoch 3/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0359 -
accuracy: 0.9886
Epoch 4/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0279 -
accuracy: 0.9913
Epoch 5/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0218 -
accuracy: 0.9931
313/313 [==============================] - 1s 3ms/step - loss: 0.0318 -
accuracy: 0.9899
Test accuracy: 0.9898999929428101
```

Dari tiga percobaan diatas diperoleh hasil:

- akurasi tes satu konvolusi layer : 0.9861999750137329
- akurasi tes dua konvolusi layer : 0.9889000058174133
- akurasi tes tiga konvolusi layer : 0.9818999767303467

Hasil tersebut menunjukkan bahwa dengan satu konvolusi layer sudah cukup baik dalam hal akurasi.

Berapa banyaknya filter yang optimal untuk setiap convolution layar?

Untuk mengetahui pengaruh jumlah filter rerhadap akurasidilakukan beberapa percobaan berikut:

MODEL CNN: 16 FILTER

```
def build_cnn_model_16_filter():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_16_filter()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1918 -
accuracy: 0.9447
Epoch 2/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0654 -
```

```
accuracy: 0.9804
Epoch 3/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0438 -
accuracy: 0.9866
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0322 -
accuracy: 0.9904
Epoch 5/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0238 -
accuracy: 0.9926
313/313 [==============================] - 1s 3ms/step - loss: 0.0363 -
accuracy: 0.9875
Test accuracy: 0.987500011920929
```

MODEL CNN: 32 FILTER

```
def build_cnn_model_32_filter():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_32_filter()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
```

```
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1810 -
accuracy: 0.9472
Epoch 2/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0592 -
accuracy: 0.9823
Epoch 3/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0396 -
accuracy: 0.9879
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0279 -
accuracy: 0.9909
Epoch 5/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0205 -
accuracy: 0.9937
313/313 [==============================] - 1s 3ms/step - loss: 0.0376 -
accuracy: 0.9873
Test accuracy: 0.9872999787330627
```

MODEL CNN: 40 FILTER

```
def build_cnn_model_40_filter():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_40_filter()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
```

```
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 6s 6ms/step - loss: 0.1694 -
accuracy: 0.9503
Epoch 2/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0572 -
accuracy: 0.9826
Epoch 3/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0385 -
accuracy: 0.9882
Epoch 4/5
938/938 [==============================] - 7s 8ms/step - loss: 0.0266 -
accuracy: 0.9918
Epoch 5/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0190 -
accuracy: 0.9939
313/313 [==============================] - 1s 4ms/step - loss: 0.0459 -
accuracy: 0.9859
Test accuracy: 0.9858999848365784
```

Dari variasi jumlah filter diperoleh hasil sebagai berikut:

- Jumlah filter 16: akurasi = 0.987500011920929
- Jumlah filter 32: akurasi = 0.9872999787330627
- Jumlah filter 40: akurasi = 0.9858999848365784

Dari hasil tersebut jumlah filter 16 sudah memberikan akurasi yang cukup baik.

Berapa ukuran filter yang optimal untuk setiap convolution layar?

Ukuran filter dalam CNN digunakan untuk ekstraksi ciri sehingga perlu dilakukan optimasi ukuran filter yang sesuai. Beberapa ukuran filter yang digunakan berupa 3x3, 5x5, dan 7x7

MODEL CNN: 3x3 FILTER

```
def build_cnn_model_filter_3x3():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_filter_3x3()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1928 -
accuracy: 0.9453
Epoch 2/5
```

```
938/938 [==============================] - 4s 5ms/step - loss: 0.0620 -
accuracy: 0.9811
Epoch 3/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0427 -
accuracy: 0.9866
Epoch 4/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0296 -
accuracy: 0.9909
Epoch 5/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0226 -
accuracy: 0.9930
313/313 [==============================] - 1s 3ms/step - loss: 0.0398 -
accuracy: 0.9868
Test accuracy: 0.9868000149726868
```

MODEL CNN: 5x5 FILTER

```
def build_cnn_model_filter_5x5():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_filter_5x5()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
```

```
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1846 -
accuracy: 0.9457
Epoch 2/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0577 -
accuracy: 0.9828
Epoch 3/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0381 -
accuracy: 0.9887
Epoch 4/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0287 -
accuracy: 0.9907
Epoch 5/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0221 -
accuracy: 0.9931
313/313 [==============================] - 1s 3ms/step - loss: 0.0342 -
accuracy: 0.9887
Test accuracy: 0.9886999726295471
```

MODEL CNN: 7x7 FILTER

```
def build_cnn_model_filter_7x7():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_filter_7x7()
```

```
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1684 -
accuracy: 0.9503
Epoch 2/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0538 -
accuracy: 0.9834
Epoch 3/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0376 -
accuracy: 0.9881
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0272 -
accuracy: 0.9911
Epoch 5/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0209 -
accuracy: 0.9933
313/313 [==============================] - 1s 4ms/step - loss: 0.0382 -
accuracy: 0.9879
Test accuracy: 0.9879000186920166
```

Variasi ukuran filter diperoleh hasil sebagai berikut:

- Ukuran filter 3x3: akurasi = 0.9894000291824341
- Ukuran filter 5x5: akurasi = 0.9886999726295471
- Ukuran filter 7x7: akurasi = 0.9879000186920166

Dari hasil tersebut ukuran filter 3x3 sudah memberikan akurasi yang cukup baik.

Berapa banyaknya hidden unit yang optimal pada bagian fully connected network?

Optimasi CNN dapat juga dilakukan pada bagian fully connected network. Berikut percobaan untuk mengamaati pengaruh hidden unit terhadap akurasi:

MODEL CNN: 128 HIDDEN

```
def build_cnn_model_128_hidden ():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_128_hidden()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1895 -
accuracy: 0.9462
Epoch 2/5
```

```
938/938 [==============================] - 4s 5ms/step - loss: 0.0605 -
accuracy: 0.9818
Epoch 3/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0407 -
accuracy: 0.9875
Epoch 4/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0295 -
accuracy: 0.9911
Epoch 5/5
938/938 [==============================] - 4s 5ms/step - loss: 0.0207 -
accuracy: 0.9937
313/313 [==============================] - 1s 3ms/step - loss: 0.0440 -
accuracy: 0.9843
Test accuracy: 0.9843000173568726
```

MODEL CNN: 256 HIDDEN

```
def build_cnn_model_256_hidden ():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_256_hidden()
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
```

```
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1726 -
accuracy: 0.9492
Epoch 2/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0572 -
accuracy: 0.9828
Epoch 3/5
938/938 [==============================] - 5s 6ms/step - loss: 0.0362 -
accuracy: 0.9889
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0253 -
accuracy: 0.9922
Epoch 5/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0180 -
accuracy: 0.9943
313/313 [==============================] - 1s 3ms/step - loss: 0.0391 -
accuracy: 0.9862
Test accuracy: 0.9861999750137329
```

MODEL CNN: 512 HIDDEN

```
def build_cnn_model_512_hidden ():
    cnn_model = tf.keras.Sequential([

        # Define the first convolutional layer
        tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation=t
f.nn.relu),

        #  Define the first max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Define the last Dense layer to output the classification
        # probabilities. Pay attention to the activation needed a probabil
ity
        # output
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    return cnn_model

cnn_model = build_cnn_model_512_hidden()
```

```
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
BATCH_SIZE = 64
EPOCHS = 5
cnn_model.fit(train_images, train_labels, batch_size=BATCH_SIZE, epochs=EP
OCHS)
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Output:

```
Epoch 1/5
938/938 [==============================] - 5s 5ms/step - loss: 0.1512 -
accuracy: 0.9550
Epoch 2/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0472 -
accuracy: 0.9849
Epoch 3/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0306 -
accuracy: 0.9901
Epoch 4/5
938/938 [==============================] - 5s 5ms/step - loss: 0.0216 -
accuracy: 0.9929
Epoch 5/5
938/938 [==============================] - 6s 6ms/step - loss: 0.0141 -
accuracy: 0.9956
313/313 [==============================] - 1s 4ms/step - loss: 0.0405 -
accuracy: 0.9879
Test accuracy: 0.9879000186920166
```

Berikut hasil akurasi terhdap variasi jumlah unit dalam hidden layer:

- Hidden layer 128 unit, akurasi = 0.9843000173568726
- Hidden layer 256 unit, akurasi = 0.9861999750137329
- Hidden layer 512 unit, akurasi = 0.9879000186920166

Hidden unit sebanyak 128 memberikan akurasi yang hampir sama dengan 256 unit bahkan 512 unit.