Laporan Tugas Individu

Eksplorasi Hyperparameter CNN dan Neural Network

Mata Kuliah : EI7007 Pembelajaran Mesin Lanjut

Nama : Arief Sartono

NIM : 33221018

Bagian 1 - Tugas Pertama

Telah dilakukan ekplorasi CNN untuk persoalan klasifikasi. Dasar kode program menggunakan contoh program *digit recognition* sebagai kode program. Kemudian *dataset* menggunakan yang telah disediakan oleh modul tf.keras.datasets (https://keras.io/api/datasets/). Kinerja model ditentukan dengan menggunakan *testing dataset*. Berikut profil dari model *deep learning* yang akan dibangun:

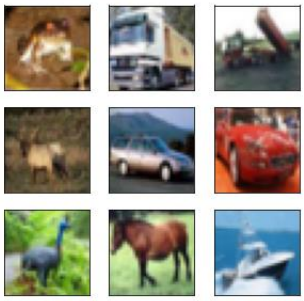
- ✓ Persoalan: Klasifikasi
- ✓ Model: *Convolutional Neural Network (CNN)*
- ✓ Dataset: CIFAR10 dari modul tf.keras.datasets
- ✓ Perangkat keras: NVIDIA DGX A100-SXM4 (Memory: 40 GB)

Model yang baik, adalah model yang bisa menjelaskan data tanpa terpengaruh oleh *data noise*. Model tidak akan fit terhadap tiap data, namun mampu menjelaskan *trend* atau kelompok data. Model yang baik akan memiliki *loss* rendah dan akurasi *(acc)* tinggi. Namun demikian, dalam aplikasinya di CNN, *overfitting* biasa terjadi, oleh karena itu terdapat beberapa *fine-tuning* atau penyesuaian yang dapat dilakukan, salah satunya yaitu penyesuaian *hyperparameter*.

Dataset CIFAR-10 adalah set data standar yang biasa digunakan dalam *computer vision* dan komunitas *deep learning*. Dataset CIFAR10 berisi gambar pesawat terbang, mobil, burung, kucing, rusa, anjing, katak, kuda, kapal laut, dan truk. Dataset ini terdiri dari 60.000 gambar berwarna dengan dimensi 32x32 *pixel* dengan 3 *channel* warna (RGB, channel warna dalam CIFAR10 memiliki rentang nilai antara 0 s.d. 255) yang terklasifikasi dalam 10 kelas, dengan 6000 gambar per kelas. Total terdapat 50.000 gambar dalam *folder training* dan 10.000 gambar dalam *folder testing*. Dengan menggunakan pyplot dari *library* matplotlib, Gambar 1 menunjukkan tampilan dari beberapa image yang diambil dari dataset CIFAR10 tersebut.

```
In [5]: # Visualisasi dataset
        import matplotlib.pyplot as plt

        plt.figure(figsize=(5,5))
        for i in range(9):
            plt.subplot(3,3,i+1)
            plt.xticks([])
            plt.yticks([])
            plt.grid(False)
            plt.imshow(train_images[i], cmap=plt.cm.binary)
        plt.show()
```

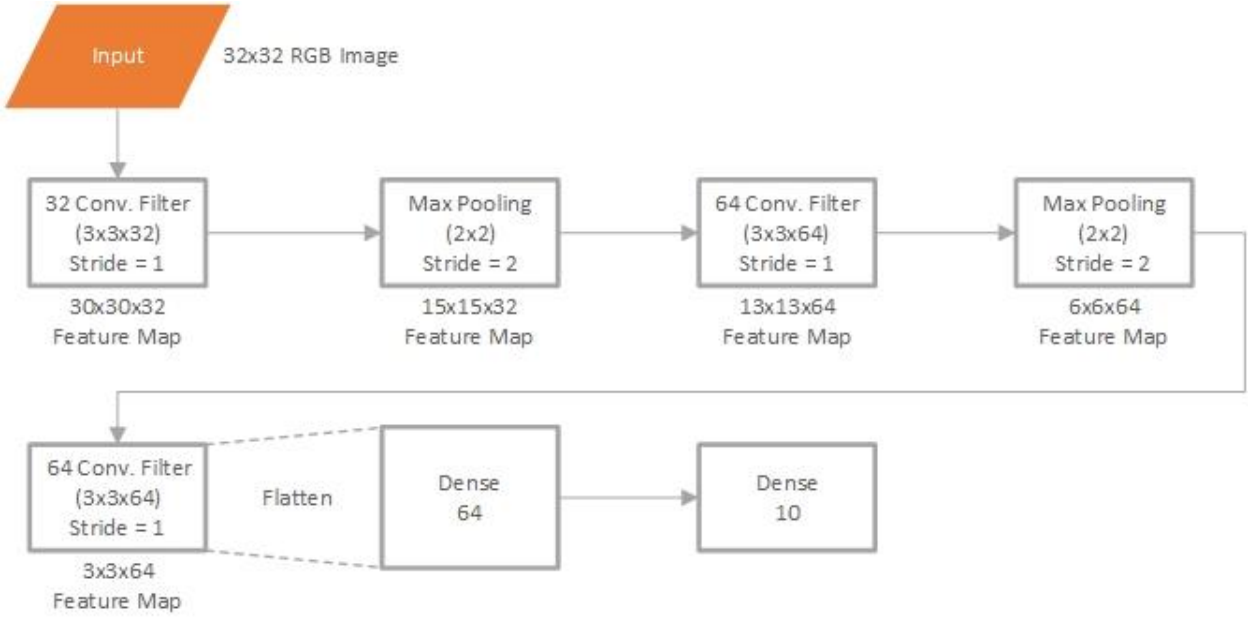


Gambar 1.1. Visualisasi Dataset CIFAR10

Untuk mengetahui nilai *hyperparameter* yang paling optimal, telah dilakukan percobaan dengan membuat variasi nilai dari *hyperparameter* 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 tersebut pada eksplorasi berikutnya Nilai optimal diambil dari percobaan yang menghasilkan kinerja terbaik. Eksplorasi yang telah dilakukan mencakup:

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

Arsitektur dari model CNN yang akan dibuat sebagaimana diilustrasikan pada Gambar 2, akan dioptimasi dengan melakukan penyesuaian hyperparameter sesuai dengan konten eksplorasi di atas.



Gambar 1.2. Arsitektur Model CNN (Sebelum Penyesuaian *Hyperparameter*)

Jawaban dari pertanyaan-pertanyaan eksplorasi di atas akan dijelaskan pada paragraf-paragraf berikut. Pada penjelasan setiap poin akan disertakan tabel berisikan data-data hasil eksperimen, Berikut penjelasan terkait nilai *hyperparameter* optimal yang didapat pada masing-masing eksperimen tersebut, dengan memperhatikan nilai tetap *(fixed)* untuk *hyperparameter* yang telah didapat pada eksperimen sebelumnya.

1.1. Jumlah *convolution layer*

Untuk mengkondisikan banyaknya *layer* yang terdapat pada file main_optimized.ipynb, pada *step* 4 (pembuatan model CNN) dipergunakan *coding* seperti berikut:

```
model.add(layers.Conv2D(64, (3, 3), padding='valid', activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
```

2 (dua) baris coding tersebut direplikasi sesuai dengan jumlah banyaknya *convolution layer* yang ingin ditentukan, dimana untuk layer *convolution* 2D:

- ✓ 64 menandakan jumlah kernel atau *filter* yang digunakan adalah 64.

- ✓ (3, 3) menandakan ukuran kernal atau *filter* yang digunakan adalah 3 x 3.
- ✓ padding='valid' berarti tidak menggunakan *padding* yang nantinya akan menyebabkan pengurangan ukuran *output* menjadi 26 x 26.
- ✓ *input_shape*=(32, 32, 3) berarti *input layer* tersebut adalah 3D *tensor(height, width, channels)*. *height* = 32, *width* = 32, dan *channels* = 3 karena CIFAR10 adalah gambar RGB.

Dan untuk layer MaxPooling:

- ✓ Melakukan "pooling" untuk mencari nilai maksimal dari hasil *convolution* (proses *downsampling* untuk mengurasi dimensi data).
- ✓ MaxPooling akan melakukan scanning gambar dengan dimensi tertentu dan memilih data dengan nilai maksimal.
- ✓ Parameter (2, 2) akan membuat sebuah kernel berukuran 2 dan kernel tersebut akan bergeser sepanjang 2.

Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 1. Nilai ACC dan LOSS yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 1.1. Nilai Accuracy (ACC) dan Loss (LOSS) yang dihasilkan dari eksperimen jumlah *convolution layer*

#	**Jumlah *Convolution Layer***	**ACC**	**LOSS**
1	1	0,599	1,169
2	2	0,665	0,971
3	3	0,642	1,042

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa banyaknya *convolution layer* yang optimal adalah 2 (dua). Untuk selanjutnya nilai *hyperparameter* jumlah *convolution layer* ini akan dipertahankan *(fixed)* pada 3 (tiga) eksperimen selanjutnya.

1.2. Ukuran filter untuk setiap *convolution layer*

Untuk mengkondisikan ukuran *filter* yang terdapat pada file main_optimized.ipynb, pada *step* 4 (pembuatan model CNN) dipergunakan *coding* seperti berikut:

```
model.add(layers.Conv2D(64, (3, 3), padding='valid', activation='relu', input_shape=(32, 32, 3)))
```

dimana untuk *layer convolution* 2D, (3,3) menandakan ukuran *kernel* atau *filter* yang digunakan untuk setiap *convolution layer*. Dalam hal ini adalah 3x3. Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2. Nilai ACC dan LOSS yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 1.2. Nilai Accuracy (ACC) dan Loss (LOSS) yang dihasilkan dari eksperimen ukuran *filter* untuk setiap *convolution layer*

#	Ukuran *Filter*	ACC	LOSS
1	2x2	0,656	1,006
2	3x3	0,665	0,971
3	4x4	0,973	1,241
4	5x5	0,645	1,029
5	6x6	0,593	1,204

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa ukuran filter untuk setiap *convolution layer* yang optimal adalah 3x3. Untuk selanjutnya nilai *hyperparameter* ukuran *filter* ini akan dipertahankan *(fixed)* pada 2 (dua) eksperimen selanjutnya.

1.3. Banyaknya filter untuk setiap *convolution layer*

Untuk mengkondisikan ukuran *filter* yang terdapat pada file main_optimized.ipynb, pada *step* 4 (pembuatan model CNN) dipergunakan *coding* seperti berikut:

```
model.add(layers.Conv2D(64, (3, 3), padding='valid', activation='relu', input_shape=(32, 32, 3)))
```

dimana untuk *layer convolution* 2D, 64 menandakan jumlah *kernel* atau *filter* yang digunakan untuk setiap *convolution layer*. Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 3. Nilai ACC dan LOSS yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 1.3. Nilai Accuracy (ACC) dan Loss (LOSS) yang dihasilkan dari eksperimen jumlah *filter* untuk setiap *convolution layer*

#	**Jumlah filter pada layer Conv 1**	**Jumlah filter pada layer Conv 2**	**ACC**	**LOSS**
1	32	32	0,626	1,083
2	32	64	0,584	1,270
3	64	64	0,675	0,966
4	64	128	0,691	0,924
5	128	128	0,613	1,258

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa jumlah filter untuk *convolution layer 1* yang optimal adalah 64, dan bahwa jumlah filter untuk *convolution layer 2* yang optimal adalah 128. Untuk selanjutnya nilai *hyperparameter* jumlah *filter* ini akan dipertahankan *(fixed)* pada 1 (satu) eksperimen selanjutnya.

1.4. Banyaknya *hidden unit* pada bagian *fully connected network*

Untuk mengkondisikan jumlah *hidden unit* pada bagian *fully connected network* yang terdapat pada file main_optimized.ipynb, pada *step* 4 (pembuatan model CNN) dipergunakan *coding* seperti berikut:

```
model.add(layers.Dense(64, activation='relu'))
```

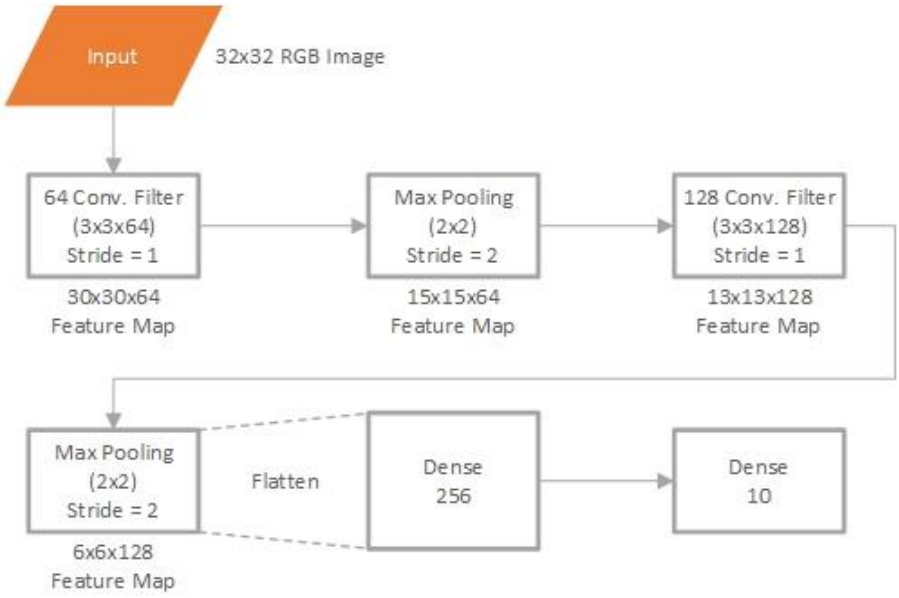
dimana untuk *layer convolution* 2D, 64 menandakan jumlah *hidden unit* yang digunakan pada bagian *fully connected network*. Nilai performa yang didapat hasil eksperimen sebagaimana

tercantum pada Tabel 4. Nilai ACC dan LOSS yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 1.4. Nilai Accuracy (ACC) dan Loss (LOSS) yang dihasilkan dari eksperimen jumlah *hidden unit* pada bagian *fully connected network*

#	**Jumlah *Hidden Unit***	**ACC**	**LOSS**
1	512	0,703	0,937
2	256	0,717	0,857
3	128	0,719	0,867
4	64	0,691	0,924
5	32	0,687	0,913
6	16	0,674	0,955

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen jumlah *hidden unit* pada bagian *fully connected network* yang optimal adalah 256.



Gambar 3. Arsitektur Model CNN (Setelah Penyesuaian *Hyperparameter*)

Arsitektur model yang telah dioptimasi sebagaimana diilustrasikan pada Gambar 3. *Feature map* yang berhasil di-*extract* dari *input* berukuran 6x6 sebanyak 128. Selanjutnya terdapat *Flatten layer* yang merubah *feature map* tersebut menjadi 1-D *vector* yang akan digunakan pada FC Layer.

Kemudian dilanjutkan dengan eksplorasi yang mencakup:

- Dari semua pilihan yang disediakan oleh *Keras Optimizer*, mana yang menghasilkan kinerja paling baik (pada nilai parameter *default*)?
- Dari *Keras Optimizer* yang optimal (pada nilai parameter *default*), lakukan eksplorasi lebih lanjut, apakah ada *learning rate schedule* yang menghasilkan kinerja yang lebih baik lagi?
- Dari semua pilihan yang disedaikan oleh *Keras (Probabilistic) Losses*, mana yang menghasilkan kinerja paling baik?

Jawaban dari pertanyaan-pertanyaan di atas akan dijelaskan pada paragraf-paragraf berikut. Pada penjelasan setiap poin akan disertakan tabel berisikan data-data hasil eksperimen,

1.5. *Keras Optimizer*

Optimizer dipergunakan untuk mengoptimasi suatu parameter, dimana mengoptimasi bisa membuat nilai sebuah parameter menjadi maksimum ataupun minimum. Sebagaimana dijelaskan dalam Keras API reference (https://keras.io/api/optimizers/), terdapat 8 (delapan) macam optimizer, yaitu SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, dan Ftrl.

Untuk mengetahui *optimizer* mana yang menghasilkan kinerja paling baik, pada file main_optimized.ipynb, pada *step* 5 (kompilasi model) dipergunakan *coding* seperti berikut:

```
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

dimana *string* yang dientrikan pada variabel *optimizer* adalah *optimizer* yang dipergunakan pada proses berjalan. Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 5. Nilai ACC dan LOSS yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 1.5. Nilai Accuracy (ACC) dan Loss (LOSS) yang dihasilkan dari eksperimen *Keras Optimizer*

#	Nama Optimizer	ACC	LOSS
1	SGD	0,471	1,525
2	RMSprop	0,717	0,857
3	Adam	0,709	0,840
4	Adadelta	0,217	2,230
5	Adagrad	0,386	1,772
6	Adamax	0,677	0,959
7	Nadam	0,704	0,885
8	Ftrl	0,100	2,303

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen terkait *optimizer* yang menghasilkan kinerja paling baik (pada nilai parameter *default*) adalah Adam. Adapun nilai *default learning rate* untuk *optimizer* Adam adalah 0,001.

1.6. *Keras Optimizer* dan *Learning Rate Schedule*

Poin 5 membahas tentang learning rate dengan nilai yang bersifat statik. Ada beberapa mekanisme yang bisa dipakai untuk mengatur nilai learning rate agar bisa berubah seiring berjalannya waktu dalam upaya Gradien Descent (GD) menemukan titik Global Minimum. GD menggunakan nilai *learning rate* yang kecil. GD akan melakukan *update weights* setelah semua data *training* harus selesai diproses, atau setiap 1 *epoch* (iterasi). Karena *learning rate* yang kecil maka untuk mencapai minimum dibutuhkan proses yang lama.

Learning rate schedule dapat dipakai untuk memodulasi (mengatur) bagaimana *learning rate* dari *optimizer* yang sedang dipergunakan berubah dari waktu ke waktu. Penjadwalan yang dimaksud

di sini adalah persamaan yang menghasilkan nilai *learning rate* (kecepatan pembelajaran) yang semakin menurun ketika melalui *optimizer step* yang sedang berlangsung. Dipakai untuk mengubah nilai kecepatan *learning rate* pada fungsi *optimizer*. Proses perhitungannya adalah sebagai berikut:

```
def decayed_learning_rate(step):
  return initial_learning_rate * decay_rate ^ (step / decay_steps)
```

Learning rate schedule ini dapat diberikan secara langsung ke tf.keras.optimizers.Optimizer, sebagai *learning rate*. Berikut contoh coding untuk melakukan *learning rate schedule*, yang menggunakan fungsi API ExponentialDecay. Ketika melakukan penyesuaian pada model Keras model, lakukan decay setiap 100.000 steps dengan basis 0.96:

```
initial_learning_rate = 0.1
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(data, labels, epochs=5)
```

Terdapat beberapa API untuk melakukan *learning rate schedule*, yaitu:

- ExponentialDecay,
- PiecewiseConstantDecay,
- PolynomialDecay,
- InverseTimeDecay,

- CosineDecay, dan
- CosineDecayRestarts.

Yang dibahas pada tugas kali ini adalah ExponentialDecay, yang merupakan persamaan yang menghasilkan nilai *learning rate* yang semakin menurun secara eksponensial. Biasanya fungsi ini dipergunakan dalam 2 (dua) tahap. Tahap 1, menentukan *learning rate* yang cukup besar, misal 0,1. Setelah itu, *learning rate* akan diupdate setiap *epoch* dengan rule sebagai berikut:

$$\eta = \eta_0 e^{-n/c}$$

dimana c *(decay coefficient)* adalah *hyperparameter* yang dapat kita atur. Tidak ada aturan baku untuk menentukan nilai c.

1.7. *Keras (Probabilistic) Losses*

Tujuan dari fungsi loss dari suatu model adalah untuk meminimalkan nilai loss selama pelatihan. Semua loss dapat di-*handle* baik melalui *class* maupun fungsi. *Class* memungkinkan kita untuk meneruskan argumen konfigurasi ke konstruktor (misalnya loss_fn = CategoricalCrossentropy(from_logits=True)), dan mereka akan melakukan reduksi secara default saat digunakan secara *stand alone*. Adapun class dan fungsi untuk probabilistic loss adalah sebagai berikut:

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Yang dibahas pada tugas kali ini adalah *class* dan fungsi CategoricalCrossentropy. Untuk *class* CategoricalCrossentropy, yang dihitung adalah crossentropy loss antara label dan prediksi. Berikut contoh codingnya.

```
CategoricalCrossentropy class
tf.keras.losses.CategoricalCrossentropy(
    from_logits=False,
    label_smoothing=0.0,
    axis=-1,
    reduction="auto",
    name="categorical_crossentropy",
)
```

Fungsi loss crossentropy ini dipergunakan ketika ada dua atau lebih *class* label. Label disediakan dalam bentuk representasi one_hot. Jika ngin memberikan label sebagai bilangan bulat, gunakan fungsi loss SparseCategoricalCrossentropy. Harus ada # class untuk nilai floating point per fitur.

Sedangkan untuk fungsi categorical_crossentropy, yang dihitung adalah *categorical crossentropy loss*. Berikut contoh codingnya.

```
tf.keras.losses.categorical_crossentropy(
    y_true, y_pred, from_logits=False, label_smoothing=0.0, axis=-1
)
```

SUMMARY

Berikut adalah *summary* dari nilai *hyperparameter* yang dipergunakan untuk menghasilkan model klasifikasi yang optimal pada tugas pertama:

- ✓ jumlah *convolution layer*: 2
- ✓ ukuran *filter* untuk setiap *convolution layer:* 3x3
- ✓ jumlah *filter* untuk setiap *convolution layer*: 64 (untuk *convolution layer* pertama) & 128 (untuk *convolution layer* kedua)
- ✓ jumlah *hidden unit* pada *fully connected network*: 256
- ✓ Keras *optimizer*: Adam

Bagian 2 - Tugas Kedua

Telah dilakukan ekplorasi *Fully Connected Neural Network* untuk persoalan regresi. Dasar kode program menggunakan contoh program *digit recognition* sebagai kode program. Kemudian *dataset* menggunakan yang telah disediakan oleh modul tf.keras.datasets (https://keras.io/api/datasets/). Kinerja model ditentukan dengan menggunakan *testing dataset*. Berikut profil dari model *deep learning* yang akan dibangun:

- ✓ Persoalan: Regresi
- ✓ Model: *Fully Connected Neural Network*
- ✓ Dataset: *Boston Housing Price* dari modul tf.keras.datasets
- ✓ Perangkat keras: NVIDIA DGX A100-SXM4 (Memory: 40 GB)

Dataset *Boston Housing Price* terdiri dari harga rumah di berbagai tempat di Boston. Setiap catatan dalam database menggambarkan pinggiran kota atau kota Boston. Data diambil dari Boston Standard Metropolitan Statistical Area (SMSA) pada tahun 1970. Selain harga, dataset tersebut juga memberikan informasi seperti Crime (CRIM), wilayah usaha *non-retail* di dalam kota (INDUS), umur pemilik rumah (AGE), dan masih banyak atribut lain yang tersedia di https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names. Dataset itu sendiri tersedia https://archive.ics.uci.edu/ml/datasets/Housing. Sebagai alternatif, dapat juga diimpor langsung dari scikit-learn bila menggunakan *library* scikit-learn. Adapun deskripsi dari fitur-fitur yang terdapat pada dataset *Boston Housing Price* tersebut adalah:

1. CRIM: Per capita crime rate by town
2. ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
3. INDUS: Proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: Nitric oxide concentration (parts per 10 million)
6. RM: Average number of rooms per dwelling
7. AGE: Proportion of owner-occupied units built prior to 1940
8. DIS: Weighted distances to five Boston employment centers
9. RAD: Index of accessibility to radial highways

10. TAX: Full-value property tax rate per $10,000
11. PTRATIO: Pupil-teacher ratio by town
12. B: 1000(Bk — 0.63)², where Bk is the proportion of [people of African American descent] by town
13. LSTAT: Percentage of lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000s

Dengan menggunakan *dataframe* dari *library* pandas, Gambar 4 menunjukkan tampilan dari beberapa *record* yang diambil dari dataset *Boston Housing Price* tersebut.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Gambar 4. Visualisasi Dataset Boston Housing Price

Untuk mengetahui nilai *hyperparameter* yang paling optimal, telah dilakukan percobaan dengan membuat variasi nilai dari *hyperparameter* 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 tersebut pada eksplorasi berikutnya Nilai optimal diambil dari percobaan yang menghasilkan kinerja terbaik. Eksplorasi yang telah dilakukan mencakup:

- Berapa banyaknya *hidden layer* yang optimal?
- Berapa banykanya *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?

Arsitektur dari model *Fully Connected Neural Network* yang akan dibuat sebagaimana diilustrasikan pada Gambar 2.1, akan dioptimasi dengan melakukan penyesuaian *hyperparameter* sesuai dengan konten eksplorasi di atas.

Gambar 2.1. Arsitektur Model *Fully Connected Neural Network* (Sebelum Penyesuaian *Hyperparameter*)

Jawaban dari pertanyaan-pertanyaan eksplorasi di atas akan dijelaskan pada paragraf-paragraf berikut. Pada penjelasan setiap poin akan disertakan tabel berisikan data-data hasil eksperimen, Berikut penjelasan terkait nilai *hyperparameter* optimal yang didapat pada masing-masing eksperimen tersebut, dengan memperhatikan nilai tetap *(fixed)* untuk *hyperparameter* yang telah didapat pada eksperimen sebelumnya.

Untuk evaluasi model, dipergunakan *Mean Absolute Error (MAE)* dan fungsi *loss*. MAE menyatakan kesalahan prediksi model rata-rata dalam unit variabel yang diminati. Nilainya dapat berkisar dari 0 hingga ∞ dan tidak berbeda dengan arah kesalahan. Sedangkan fungsi loss secara default menggunakan *Mean Squared Error (MSE)*, yang mana bisa diubah sesuai kebutuhan pada poin 2.5. Baik nilai LOSS maupun nilai MAE berorientasi negatif, yang berarti nilai yang lebih rendah lebih baik.

2.1. Jumlah *hidden layer*

Untuk mengkondisikan banyaknya *hidden layer* yang terdapat pada file main_optimized.ipynb, pada *step* 5 (pembuatan model regresi) dipergunakan *coding* seperti berikut:

```
model.add(layers.Dense(8, activation='relu', input_shape=[X_train.shape[1]]))
model.add(layers.Dense(16, activation='relu'))
```

Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2.1. Nilai LOSS dan MAE yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol). Jumlah hidden unit *(dense)* pada setiap *hidden layer* ditetapkan sembarang. Dalam hal ini adalah 8-16-32-16-8 untuk masing-masing *hidden layer* ke 1-2-3-4-5.

Tabel 2.1. Nilai fungsi loss (LOSS) dan MAE yang dihasilkan dari eksperimen jumlah *hidden layer*

#	**Jumlah *Hidden Layer (Dense)***	**Jumlah *Hidden Unit* pada *Hidden Layer***	**LOSS**	**MAE**
1	5	8-16-32-16-8	24,382	2,988
2	4	8-16-32-16	23,457	3,325
3	3	8-16-32	28,442	3,525
4	2	8-16	28,006	3,649
5	1	8	132,488	9,058

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen jumlah *hidden layer* yang optimal adalah 4. Untuk selanjutnya nilai *hyperparameter* jumlah *hidden layer* ini akan dipertahankan *(fixed)* pada eksperimen-eksperimen selanjutnya.

2.2. Jumlah *hidden unit* pada setiap *hidden layer*

Untuk mengkondisikan banyaknya *hidden unit pada* setiap *hiddel layer* yang terdapat pada file main_optimized.ipynb, pada *step* 5 (pembuatan model regresi) dipergunakan *coding* seperti berikut:

```
model.add(layers.Dense(8, activation='relu', input_shape=[X_train.shape[1]]))
model.add(layers.Dense(16, activation='relu'))
```

Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2.2. Nilai LOSS dan MAE yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 2.2. Nilai fungsi loss (LOSS) dan MAE yang dihasilkan dari eksperimen jumlah *hidden unit* pada setiap *hidden layer*

#	**Jumlah *Hidden Unit* pada setiap *Hidden Layer***	**LOSS**	**MAE**
1	8	20,303	3,125
2	16	27,176	3,071
3	32	33,168	3,781
4	64	25,214	3,089
5	128	17,545	2,738

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen jumlah *hidden layer* yang optimal adalah 4. Untuk selanjutnya nilai *hyperparameter* jumlah *hidden layer* ini akan dipertahankan *(fixed)* pada eksperimen-eksperimen selanjutnya.

2.3. Fungsi aktivasi

Untuk menentukan fungsi aktivasi apa di setiap *layer* agar bisa mendapatkan hasil yang optimal, pada file main_optimized.ipynb, pada *step* 5 (pembuatan model regresi) dipergunakan *coding* seperti berikut:

```
model.add(layers.Dense(8, activation='relu', input_shape=[X_train.shape[1]]))
model.add(layers.Dense(16, activation='relu'))
```

Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2.3. Nilai LOSS dan MAE yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 2.3. Nilai fungsi loss (LOSS) dan MAE yang dihasilkan dari eksperimen fungsi aktivasi

#	Fungsi Aktivasi	LOSS	MAE
1	ReLU	17,545	2,738
2	Sigmoid	29,877	3,932
3	Softmax	512,111	20,709
4	Softplus	34,292	4,063
5	Softsign	25,061	3,057
6	Tanh	22,076	2,899
7	Selu	17,405	2,841
8	Elu	22,022	3,431
9	Exponential	nan	nan

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen fungsi aktivasi yang optimal adalah ReLU. Untuk selanjutnya nilai *hyperparameter* fungsi aktivasi ini akan dipertahankan *(fixed)* pada eksperimen-eksperimen selanjutnya.

2.4. *Optimizer*

Untuk menentukan optimizer apa agar bisa mendapatkan hasil yang optimal, pada file main_optimized.ipynb, pada *step* 5 (pembuatan model regresi) dipergunakan *coding* seperti berikut:

```
model.compile(optimizer='rmsprop', loss='mse', metrics='mae')
```

Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2.4. Nilai LOSS dan MAE yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol).

Tabel 2.4. Nilai fungsi loss (LOSS) dan MAE yang dihasilkan dari eksperimen *Keras Optimizer*

#	Nama Optimizer	LOSS	MAE
1	SGD	nan	nan

2	RMSprop	17,545	2,738
3	Adam	15,439	2,565
4	Adadelta	608,572	22,916
5	Adagrad	22,354	3,471
6	Adamax	28,393	3,216
7	Nadam	47,691	5,069
8	Ftrl	23,748	3,685

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen Keras *optimizer* yang optimal adalah ReLU. Untuk selanjutnya nilai *hyperparameter* Keras *optimizer* ini akan dipertahankan *(fixed)* pada eksperimen selanjutnya.

2.5. Fungsi *loss*

Untuk menentukan fungsi *loss* apa agar bisa mendapatkan hasil yang optimal, pada file main_optimized.ipynb, pada *step* 5 (pembuatan model regresi) dipergunakan *coding* seperti berikut:

```
model.compile(optimizer='rmsprop', loss='mse', metrics='mae')
```

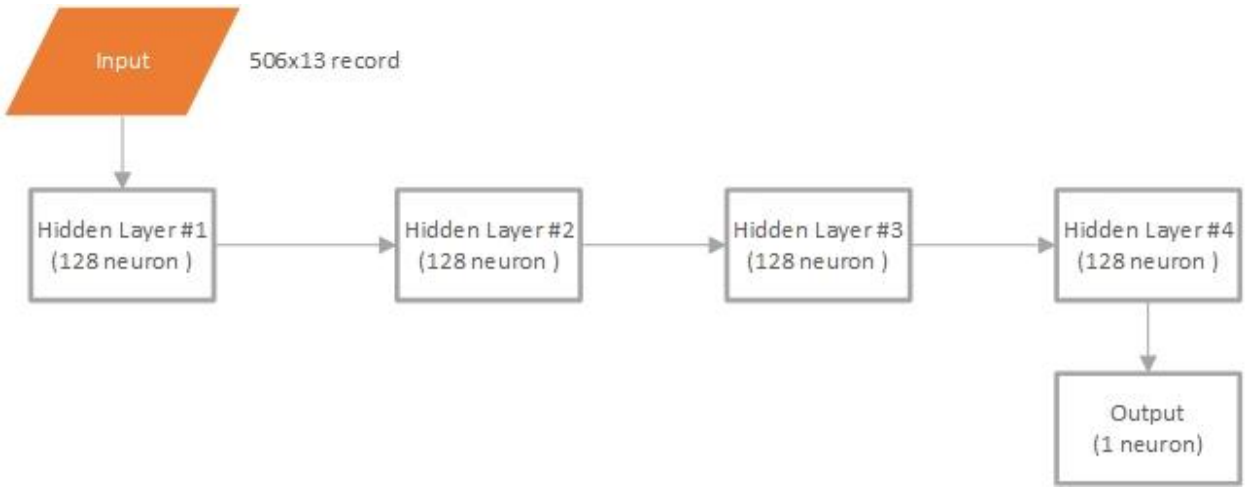
Nilai performa yang didapat hasil eksperimen sebagaimana tercantum pada Tabel 2.5. Nilai MAE yang disertakan dibulatkan sampai 3 dijit dibelakang angka 0 (nol). Nilai LOSS tidak dapat disertakan karena fungsi loss itu sendiri merupakan hyperparameter yang dijadikan objek eksperimen kali ini.

Tabel 2.5. Nilai MAE yang dihasilkan dari eksperimen fungsi *loss*

#	**Nama Fungsi Loss**	**MAE**
1	Mean_squared_error	2,565
2	Mean_absolute_error	3,001
3	Mean_absolute_percentage_error	2,988

4	Mean_squared_logarithmic_error	2,956
5	Cosine_similarity	21,399
6	Huber	2,982
7	Huber_loss	2,994
8	Log_cosh	2,945

Dari data eksperimen yang didapat, bisa ditarik kesimpulan bahwa eksperimen fungsi *loss* yang optimal adalah mean_squared_error (MSE).



Gambar 2.2. Arsitektur Model *Fully Connected Neural Network* (Sesudah Penyesuaian *Hyperparameter*)

Arsitektur model yang telah dioptimasi sebagaimana diilustrasikan pada Gambar 2.2. Jumlah hidden layer yang menghasilkan model yang optimal adalah 4 layer dengan masing-masing memiliki 128 neuron.

SUMMARY

Berikut adalah *summary* dari nilai *hyperparameter* yang dipergunakan untuk menghasilkan model regresi yang optimal pada tugas kedua:

- ✓ jumlah *hidden layer*: 4
- ✓ jumlah *hidden unit* untuk setiap *hidden layer*: 128
- ✓ *activation function:* ReLU
- ✓ Keras *optimizer*: Adam
- ✓ *loss function:* Mean Squared Error (MSE)