Penerapan *Singular Value Decomposition* untuk Optimasi *Neural Networks* melalui *Low-Rank Approximation*

13523077 Albertus Christian Poandy[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523077@std.stei.ib.ac.id , [2]achrisp05@gmail.com

***Abstrak*—Makalah ini membahas tentang penerapan *Singular Value Decomposition* (SVD) untuk mengoptimalkan *neural networks* melalui teknik kompresi low-rank approximation. Teknik ini bertujuan mengurangi parameter dari model *neural network* untuk mengefisienkan sumber daya komputasi, mulai dari waktu hingga memori yang diperlukan, tanpa mengorbankan akurasi yang signifikan. Pendekatan ini memberikan solusi sederhana dan praktis untuk meningkatkan efisiensi model *neural network* untuk proses *deployment*, terutama pada perangkat dengan sumber daya yang terbatas.**

***Kata kunci*—Low-Rank Approximation, Neural Networks, Optimasi model, Singular Value Decomposition**

I. Pendahuluan

Artificial intelligence (*AI*) adalah hal yang tak lagi asing bagi banyak orang. Penggunaannya telah diaplikasikan di berbagai bidang dan memengaruhi berbagai aspek kehidupan, mulai dari industri hingga kehidupan sehari-hari, karena banyaknya manfaat yang ditawarkan. Beberapa teknologi AI yang mencolok dan sering ditemukan adalah pengenalan gambar, *Natural Language Processing (NLP),* pengenalan suara, dan bahkan prediksi untuk pengambilan keputusan. Di balik seluruh teknologi tersebut, *neural network* adalah salah satu alat yang memainkan peran penting .

Neural network telah menjadi tulang punggung dari berbagai teknologi kecerdasan buatan yang sering ditemukan pada kehidupan sehari-hari. Hal ini dapat dicapai dengan kemampuan *neural network* untuk mempelajari berbagai representasi data kompleks dengan strukturnya dan non-linear. Namun, untuk mencapai hasil yang baik dalam data yang kompleks, terkadang dibutuhkan struktur dengan biaya komputasi yang cukup tinggi. Hal ini tentu dapat menjadi masalah jika efisiensi komputasi sangat dibutuhkan dan terdapat keterbatasan sumber daya pada perangkat.

Untuk mengatasi masalah tersebut, dapat diterapkan berbagai teknik optimasi, salah satunya adalah kompresi data. *Singular Value Decomposition* (SVD) adalah contoh teknik yang populer dalam konteks kompresi data, yaitu dengan memanfaatkan bentuk reduksi dari hasil dekomposisi matriks oleh SVD. Pengaproksimasian matriks dengan mengambil beberapa rank terbawah dari hasil dekomposisi SVD disebut dengan *low-rank approximation*.

Untuk itu, makalah ini bertujuan mengaplikasikan teknik kompresi data *low-rank approximation* dalam usaha melakukan optimasi pada *neural network* untuk mengurangi biaya komputasi yang diperlukan. Dengan pengambilan jumlah *rank* yang tepat, biaya komputasi yang dibutuhkan dapat menurun tanpa adanya informasi yang hilang secara signifikan.

Makalah ini terbagi menjadi 5 bagian utama, yaitu: Bagian 1 adalah pendahuluan. Bagian 2 adalah landasan teori yang menyajikan dasar-dasar teori dan rumus. Bagian 3 akan menjelaskan metodologi yang digunakan. Bagian 4 akan membahas hasil yang didapatkan dan analisisnya. Terakhir, bagian 5 akan memberikan kesimpulan yang didapatkan.

II. Landasan Teori

A. Singular Value Decomposition (SVD)

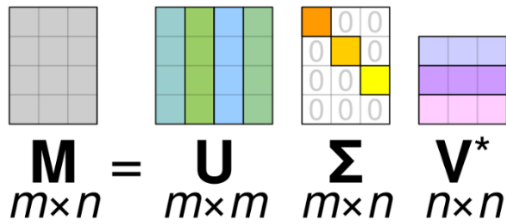
Proses dekomposisi matriks berarti melakukan pemfaktoran pada sebuah matriks, misalnya A, menjadi hasil perkalian matriks dari sejumlah matriks lainnya, $P_1, P_2, \dots, P_k$. Metode dekomposisi nilai singular (*Singular Value Decomposition — SVD*) adalah teknik dekomposisi yang memfaktorkan sebuah matriks menjadi hasil perkalian dari tiga matriks lainnya. Misal A adalah sebuah matriks berukuran $m \times n$, hasil dekomposisi dengan metode SVD dari A didefinisikan sebagai berikut:

$$A = U\Sigma V^{T} \quad (1)$$

Di mana:

1. $\boldsymbol{U}$: Matriks ortogonal berukuran $m \times m$. Kolom-kolomnya disebut dengan vektor-vektor singular kiri dari A.
2. $\boldsymbol{\Sigma}$: Matriks diagonal berukuran $m \times n$. Elemen-elemen diagonalnya disebut dengan nilai-nilai singular (*singular values*) dari A.

3. $\boldsymbol{V}$: Matriks ortogonal berukuran $n \times n$. Kolom-kolomnya disebut dengan vektor-vektor singular kanan dari A.



Gambar 1. Ilustrasi Matriks Hasil Dekomposisi Dengan Metode SVD
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf

B. Low-Rank Approximation

Salah satu hal menarik dari metode SVD hasil dekomposisinya dapat direduksi menjadi bentuk yang lebih sederhana. SVD tereduksi adalah versi yang lebih sederhana dari hasil dekomposisi SVD yang hanya mempertahankan beberapa nilai singular dan vektor singular utama. Hal ini dapat dimanfaatkan unuk melakukan *low-rank approximation*.

Low-rank approximation adalah bentuk aproksimasi sebuah matriks dengan merepresentasikan hasil dekomposisi matriks besar dengan dimensi yang lebih kecil tanpa kehilangan banyak informasi. *Low-rank approximation* dilakukan dengan memilih rank rendah k, yang berarti hanya mempertahankan k nilai singular terbesar dari hasil dekomposisi SVD. Misal matriks A didekomposisi menjadi bentuk (1), maka hasil dekomposisi dipotong sesuai k sedemikian sehingga:

1. U_K adalah matriks $m \times k$
2. Σ_K adalah matriks $k \times k$
3. V_K adalah matriks $k \times n$

Dengan demikian, matriks A dapat diaproksimasi menjadi:

$$A_K = U_K \Sigma_K V_K^T \qquad (2)$$

Di mana A_K adalah aproksimasi rank-k dari A.

C. Neural Network

C.1. Definisi Umum

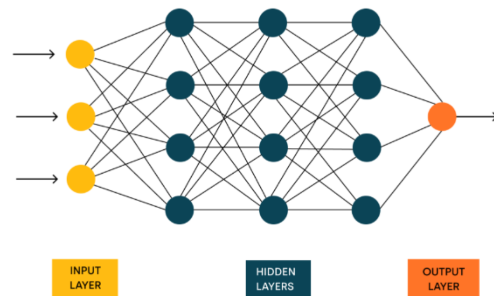
Neural network atau jaringan saraf tiruan adalah model model komputasi yang terinspirasi dari sistem jaringan saraf biologis. *Neural network* terdiri dari lapisan-lapisan (*layers*), dengan tiap lapisan terdiri dari neuron neuron, yang saling terhubung dan tersusun secara berurutan untuk membentuk suatu struktur tertentu. Neuron pada suatu lapisan terhubung dengan seluruh neuron pada lapisan sebelumnya.

Tiap neuron yang berada pada *neural network* akan menerima suatu input, memprosesnya, lalu menghasilkan output yang akan diproses lagi oleh neuron selanjutnya. Seluruh proses dalam *neural network* dapat dianggap sebagai suatu fungsi, yang menerima input berupa data dengan dimensi tertentu dan akan menghasilkan suatu output sesuai dengan dimensi yang diinginkan.

Setiap neuron akan memproses data input dan memberikan output melalui bobot (*weights*), *bias,* dan fungsi aktivasi (*activation function*). Nilai dari bobot dan *bias* yang paling optimal dicari melalui proses *training*, yaitu proses meng-*adjust* bobot dan *bias* secara iteratif hingga mendapatkan output dengan akurasi yang paling tinggi. Hasil output dari suatu *neural network* dapat diaplikasikan pada berbagai hal, seperti tugas prediksi, klasifikasi, ataupun regresi.

Terdapat tiga bagian lapisan pada suatu sistem *neural network*, yaitu *input layer, hidden layer,* dan *output layer*.

1. *Input layer* adalah lapisan paling awal dari suatu *neural network* yang bertugas menerima data masukan, sehingga jumlah neuronnya akan sama dengan dimensi data masukan.
2. *Hidden layer* adalah lapisan yang bertugas memproses segala informasi dari lapisan input. Jumlah *hidden layer* dan jumlah neuron yang berada pada suatu *hidden layer* biasanya disesuaikan pada target yang ingin dihasilkan ataupun input yang diberikan. *Neural network* yang lebar/*wide* (memiliki jumlah neuron yang banyak dengan jumlah *layer* sedikit) akan memiliki kecepatan yang lebih tinggi dalam proses *training* karena memiliki struktur yang lebih sederhana, sedangkan *neural network* yang dalam/*deep* (memiliki banyak *layer*) akan membutuhkan sumber daya komputasi yang lebih besar, tetapi dapat menangkap pola-pola kompleks dengan lebih baik.
3. *Output layer* adalah lapisan yang bertugas menghasilkan prediksi, yaitu output terakhir dari sistem *neural network*. Jumlah neuron pada lapisan ini bergantung pada dimensi output yang diinginkan.



Gambar 2. Ilustrasi Struktur *neural network*
Sumber: https://www.qtravel.ai/blog/what-are-neural-networks-and-what-are-their-applications/

C.2. Fungsi Aktivasi (Activation Function)

Fungsi aktivasi adalah fungsi matematis yang memberikan aspek non-linearitas kepada model, yang memungkinkan *neural network* dapat mempelajari dan mengenali hubungan yang lebih kompleks dalam data yang diberikan. Tanpa fungsi aktivasi yang memberikan aspek non-linearitas, neural network hanya akan menjadi suatu kombinasi linear dari input, artinya seluruh proses yang dijalankan melalui berbagai lapisan-lapisan berbeda sebenarnya dapat dihasilkan hanya dengan satu lapisan.

Hal ini akan membatasi kemampuan neural network dalam menangkap pola-pola yang rumit dan kompleks. Terdapat banyak fungsi aktivasi umum yang dapat digunakan, tetapi pada makalah ini hanya akan ada dua yang digunakan:

1. ReLU (*Rectified Linear Unit*): Fungsi ReLU adalah salah satu fungsi aktivasi yang paling populer dan sering digunakan dalam membangun *neural network*. Fungsi aktivasi ReLU didefinisikan sebagai berikut:

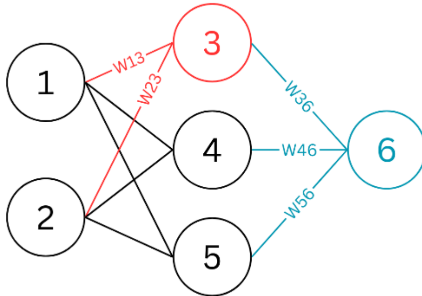
$$f(x) = max(0, x) \tag{3}$$

2. Sigmoid: Fungsi Sigmoid adalah fungsi aktivasi yang menghasilkan output berupa nilai yang berada pada rentang 0 dan 1. Fungsi ini biasanya diterapkan pada lapisan *output* pada tugas *binary classification* untuk mendapatkan probabilitas di antara 0 dan 1. Fungsi Sigmoid didefinisikan sebagai berikut:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

C.3. Forward Propagation

Forward Propagation pada dasarnya adalah proses menghitung output dari suatu *neural network*. Output dari suatu neuron didapatkan dengan menjumlahkan seluruh hasil perkalian bobot yang berkorespondensi dengan neuron tersebut dengan nilai neuron yang berada pada lapisan sebelumnya. Sebagai contoh, perhatikan *neural network* sederhana berikut:



Gambar 3. Contoh *neural network* sederhana
Sumber: arsip pengguna

Misal, nilai dari neuron i adalah z_i, *bias*-nya adalah b_i, bobot yang berkorespondensi dengan neuron i dan j adalah w_{ij}, dan hasil outputnya adalah a_i. Maka, nilai dari neuron:

$$z_3 = w_{13}z_1 + w_{23}z_2 + b_3$$

$$z_6 = w_{36}z_3 + w_{46}z_4 + w_{56}z_5 + b_6$$

Kemudian, diterapkan fungsi aktivasi, sehingga output dari neuron:

$$a_3 = f(z_3)$$

$$a_6 = f(z_6)$$

Di mana f(x) adalah fungsi aktivasi yang diterapkan pada neuron. Berdasarkan pola, perhitungan tiap lapisan dapat ditulis:

$$\begin{bmatrix} z_3 \\ z_4 \\ z_5 \end{bmatrix}^T = \begin{bmatrix} z_1 & z_2 \end{bmatrix} \begin{bmatrix} w_{13} & w_{14} & w_{14} \\ w_{23} & w_{24} & w_{25} \end{bmatrix} + \begin{bmatrix} b_3 \\ b_4 \\ b_5 \end{bmatrix}^T$$

Tulis ulang persamaan di atas menjadi:

$$z = XW + b \tag{5}$$

Di mana:

- $\boldsymbol{W}$: Matriks bobot
- $\boldsymbol{X}$: Lapisan input
- $\boldsymbol{b}$: Bias

Kemudian, terapkan fungsi aktivasi:

$$a = f(z)$$

Maka, untuk mendapatkan output dari suatu neural network, perhitungan akan dimulai dari lapisan ke-dua, ke-tiga, dan seterusnya hingga mencapai lapisan output (nilai neuron-neuron pada lapisan pertama adalah nilai data input).

C.4. Backpropagation

Backpropagation adalah cara *neural network* belajar dan meningkatkan akurasinya. Dalam proses *training*, setelah selesai dilakukannya *formard propagation* dan output dari *neural network* telah didapatkan, hasil prediksi (output) akan dibandingkan dengan nilai yang sebenarnya. Dengan membandingkan output dengan nilai sebenarnya, didapatkan nilai *error*, yaitu seberapa jauh kesalahan prediksi dengan nilai yang seharusnya. Dengan mengetahui nilai *error* ini, bobot pada lapisan output pun di-*adjust* untuk menurunkan nilai *error*.

Bobot pada suatu lapisan di-*adjust* dengan cara menghitung gradien dari *error* (*loss function*) terhadap bobot dengan menggunakan aturan rantai (*chain rule*). Misal, suatu *neural network* dengan lapisan output L memiliki *error* C_0, dengan C_0 adalah *Mean Squared Error (MSE)* yang didefiniskan:

$$C_0 = (a_L - y_{true})^2 \tag{6}$$

Ingat kembali bahwa:

$$z_L = a_{L-1}w_L + b_L$$

$$a_L = f(z_L)$$

Maka, perubahan pada bobot yang perlu dilakukan adalah:

$$\Delta W = \frac{\partial C_0}{\partial w_L} = \frac{\partial z_L}{\partial w_L} \frac{\partial a_L}{\partial z_L} \frac{\partial C_0}{\partial a_L} \tag{7}$$

Di mana:

$$\frac{\partial z_L}{\partial w_L} = a_{L-1}$$

$$\frac{\partial a_L}{\partial z_L} = f'(z_L)$$

$$\frac{\partial C_0}{\partial a_L} = 2(a_L - y_{true})$$

Akhirnya, bobot diperbarui:

$$W_{new} = W_{old} - \eta \cdot \Delta W \quad (8)$$

Di mana η adalah *learning rate*, yaitu parameter yang diatur secara manual untuk menentukan seberapa signifikan perubahan bobot yang diinginkan untuk tiap iterasinya.

Proses *training neural network* pada dasarnya adalah melakukan proses *forward propagation* untuk mendapatkan output dan *backpropagation* untuk meng-*adjust* bobot yang dilakukan secara iteratif dan dibagi menjadi *batches-batches* kecil. Banyaknya iterasi yang dilakukan untuk melatih *neural network* disebut dengan *epoch*, dan jumlah dari suatu *batch* disebut dengan *batch size*.

C.5. Kompresi Bobot dengan Low-Rank Approximation

Setelah proses *training*, model *neural network* telah memiliki bobot dan bias yang sesuai pada tiap lapisannya untuk melakukan prediksi dengan hasil yang akurat. Selanjutnya, untuk menghasilkan prediksi, hanya diperlukan proses *forward propagation* untuk menghasilkan output dari data input yang belum pernah diliat model sebelumnya. Ingat kembali bahwa sesuai dengan persamaan (5), proses *forward propagation* pada dasarnya adalah melakukan operasi perkalian matriks pada tiap lapisan untuk mendapatkan output pada lapisan akhir.

Misal W adalah matriks dengan ukuran $m \times n$ dan X adalah matriks dengan ukuran $b \times m$ dengan b adalah *batch size*. Operasi perkalian matriks antara X dan W pada suatu lapisan memiliki kompleksitas waktu:

$$O(m \cdot n \cdot b)$$

Untuk matriks bobot dengan dimensi atau *batch size* yang berukuran besar, proses komputasi ini tentu saja tidak optimal, terutama jika model yang akan di-*deploy* membutuhkan tingkat efisiensi yang tinggi. Maka, diperlukan optimasi pada proses komputasi ini.

Salah satu optimasi yang dapat dilakukan adalah kompresi matriks menggunakan *low-rank approximation*. Dengan *low-rank approximation*, matriks bobot W didekomposisi menjadi tiga komponen, U_r, S_r, dan V_r^T, di mana:

- U_r adalah matriks berukuran $m \times r$
- Σ_r adalah matriks berukuran $r \times r$
- V_R^T adalah matriks berukuran $r \times n$

Operasi perkalian matriks antara X dan W digantikan dengan tiga langkah:

1. Kalikan X dengan $U_r\Sigma_r$. Kompleksitas waktu:

$$O(b \cdot m \cdot r + b \cdot r^2)$$

2. Kalikan hasil perkalian sebelumnya dengan V_r^T. Kompleksitas waktu:

$$O(b \cdot r \cdot n)$$

Maka, total kompleksitas waktu untuk perkalian matriks dengan *low-rank approximation* adalah:

$$O(b \cdot m \cdot r + b \cdot r^2 + b \cdot r \cdot n)$$

Sebagai contoh, misalkan dimensi matriks W adalah $m \times n$, dengan $m = 512, n = 256$. Rank yang dipilih adalah $r = 50$, dan *batch size* yang digunakan adalah $b = 32$.

- Pada perkalian matriks biasa:

$$O(b \cdot m \cdot n) = O(32 \cdot 512 \cdot 256) = O(4{,}194{,}304)$$

- Pada perkalian matriks kompresi:

$$O(b \cdot m \cdot r + b \cdot r^2 + b \cdot r \cdot n)$$

$$= O(32 \cdot 512 \cdot 50 + 32 \cdot 50\text{^}2 + 32 \cdot 50 \cdot 256$$

$$= O(819{,}200 + 80{,}000 + 409{,}600)$$

$$= O(1{,}308{,}800)$$

Maka, persentase reduksi berdasarkan pengurangan waktu:

$$\%Reduksi = \frac{Biasa - Kompresi}{Biasa} \times 100\%$$

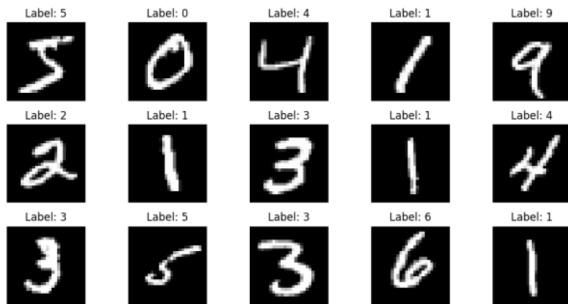
$$= \frac{4{,}194{,}304 - 1{,}308{,}800}{4{,}194{,}304} \times 100\% \approx \mathbf{68.8}\%$$

Walapun pemilihan rank r yang semakin kecil akan meningkatkan efisiensi dan membuat proses komputasi menjadi jauh lebih cepat, pemilihan rank r yang semakin kecil juga akan membuat semakin banyak informasi dari matriks yang hilang, yang akan menurunkan performa model. Maka, kunci dari optimasi adalah untuk memilih jumlah rank r yang tepat agar proses komputasi berjalan lebih cepat tanpa adanya informasi yang hilang secara signifikan.

III. METODOLOGI

A. Dataset

Untuk membandingkan dan mengevaluasi pengaruh kompresi matriks dengan *low-rank approximation* pada *neural network*, akan dibangun model *neural network* yang akan dilatih untuk memprediksi dan mengklasifikasikan gambar berupa digit (angka) yang ditulis tangan, yaitu pada MNIST dataset.



Gambar 4. Beberapa contoh gambar pada MNIST Dataset
Sumber: https://yann.lecun.com/exdb/mnist/

MNIST dataset menyediakan 70.000 gambar dengan ukuran 28x28 pixel yang berisi digit hasil tulisan tangan yang telah diberi label. Data yang akan digunakan dalam proses *training* adalah sebanyak 60.000 gambar, dan data yang digunakan dalam proses *testing* adalah sebanyak 10.000 gambar.

B. Model Neural Network

Model *neural network* yang akan digunakan adalah model *Sequential* yang dikembangkan menggunakan framework *Tensorflow* dan *Keras*. Model dibangun hanya menggunakan lapisan *Dense*, yaitu lapisan *fully connected*, di mana setiap neuron yang berada pada lapisan tersebut terhubung langsung dengan setiap neuron pada lapisan sebelumnya.

Model akan dilatih dengan *batch* size 1 dan 5 *epoch*, mengingat dataset yang digunakan cukup sederhana. Diterapkan juga fungsi aktivasi ReLU untuk ketiga *hidden layers*, serta fungsi aktivasi Sigmoid untuk lapisan output. Lebih jelasnya, arsitektur dari model yang akan digunakan dalam melakukan *training* adalah:

```
original_model = Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(1028, activation="relu"),
    layers.Dense(512, activation="relu"),
    layers.Dense(256, activation="relu"),
    layers.Dense(10, activation="sigmoid")
])
```

Gambar 5. Potongan kode arsitektur model *neural network*
Sumber: arsip pengguna

Tabel 1. Arsitektur model *neural network*

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 1028)	806,690
dense_2 (Dense)	(None, 512)	526,848
dense_3 (Dense)	(None, 256)	131,328
dense_4 (Dense)	(None, 10)	2,570
Total Params		**1,467,726** (5.60 MB)

Setelah melakukan proses *training* untuk mendapatkan bobot dan bias yang optimal untuk melakukan klasifikasi, matriks bobot dari tiap lapisan pun dapat dikompresi menggunakan *low-rank approximation*. Kompresi dilakukan dengan cara mengganti satu lapisan menjadi 2 lapisan dense, dengan bobot pada lapisan pengganti pertama adalah matriks $U_r \Sigma_r$, dan bobot pada lapisan pengganti ke-dua adalah matriks V_r^T.

Sebagai contoh, perhatikan lapisan *dense* pertama pada model yang memiliki 1028 neuron. Terdapat 784 neuron sebagai input pada lapisan pertama, maka matriks bobot yang diperlukan untuk melakukan komputasi output pada lapisan dense pertama akan memiliki dimensi 784×1028. Matriks input yang berdimensi $b \times 784$ (b adalah *batch size*, $b = 1$) untuk menghasilkan output dengan dimensi $b \times 1028$. Untuk melakukan *low-rank approximation*, matriks bobot yang berukuran 784×1028 akan didekomposisi dan direduksi menjadi:

- U_r : matriks berukuran $784 \times r$
- Σ_r : matriks berukuran $r \times r$
- V_r^T : matriks berukuran $r \times 1028$

Maka, lapisan *dense* akan dipecah menjadi dua lapisan:

1. Lapisan pecahan pertama memiliki r neuron, dengan matriks bobot berukuran $784 \times r$, yaitu hasil perkalian matriks U_r dan Σ_r.
2. Lapisan pecahan ke-dua memiliki 1028 neuron, dengan matriks bobot berukuran $r \times 1028$, yaitu matriks V_r^T.

Proses ini akan dilakukan pada setiap lapisan *dense*, sehingga jumlah lapisan dalam model *neural network* yang digunakan pada makalah ini akan berubah menjadi dua kali lipat, yang sebelumnya hanya memiliki 4 lapisan *dense*, berubah menjadi 8 lapisan *dense*.

```
def apply_low_rank_approximation(layer, rank):
    weights, biases = layer.get_weights()

    # lakukan dekomposisi SVD
    U, S, Vt = np.linalg.svd(weights, full_matrices=False)

    # pastikan rank yang diambil tidak melebihi rank dari SVD
    rank = min(rank, len(S))
    # ambil r komponen
    U_r = U[:, :rank]
    S_r = np.diag(S[:rank])
    Vt_r = Vt[:rank, :]

    # ganti satu layer dengan dua layer baru
    dense1 = layers.Dense(rank,
                          activation=None,
                          use_bias=False)
    dense2 = layers.Dense(layer.units,
                          activation=layer.activation,
                          use_bias=True)
    return [dense1, dense2], [U_r @ S_r, Vt_r, biases]
```

Gambar 6. Potongan kode fungsi memecah lapisan untuk *low-rank approximation*
Sumber: arsip pengguna

```python
# Create a low-rank model
def create_low_rank_model(original_model, rank):
    low_rank_model = tf.keras.Sequential()
    low_rank_model.add(layers.Input(shape=(784,)))

    for layer in original_model.layers:

        # Untuk simplisitas,
        # terapkan low-rank approximation hanya pada lapisan Dense
        if isinstance(layer, tf.keras.layers.Dense):
            new_layers, new_weights = apply_low_rank_approximation(
                                        layer=layer,
                                        rank=rank
                                    )

            # tambahkan lapisan baru ke model
            for new_layer in new_layers:
                low_rank_model.add(new_layer)

            # tetapkan bobot baru ke lapisan baru
            low_rank_model.layers[-2].set_weights([new_weights[0]])
            low_rank_model.layers[-1].set_weights([new_weights[1],
                                                   new_weights[2]])

        else:
            low_rank_model.add(layer)

    return low_rank_model
```

Gambar 7. Potongan kode fungsi membuat *low-rank* model
Sumber: arsip pengguna

Tabel 2. Arsitektur model setelah *low-rank approximation*

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, r)	?
dense_2 (Dense)	(None, 1028)	?
dense_3 (Dense)	(None, r)	?
dense_4 (Dense)	(None, 512)	?
dense_5 (Dense)	(None, r)	?
dense_6 (Dense)	(None, 256)	?
dense_7 (Dense)	(None, r)	?
dense_8 (Dense)	(None, 10)	?
Total Params		?

Perhatikan bahwa walaupun jumlah lapisan pada model *neural network* bertambah, belum tentu bahwa jumlah parameter dalam model ikut bertambah, tetapi tentu saja bisa berkurang. Selanjutnya perlu membandingkan dampak *low-rank approximation* terhadap hasil akurasi yang didapatkan.

C. Evaluasi Performa

Perbandingan performansi akan dilakukan antara model *neural network* biasa dengan model hasil pengaplikasian *low-rank approximation*. Jumlah rank r yang diambil untuk *low-rank approximation* juga akan divariasikan untuk membandingkan pengaruhnya terhadap hasil. Perbandingan akan dilakukan dengan menggunakan beberapa metrik evaluasi pada dataset *testing*, yaitu akurasi prediksi, efisiensi waktu, dan penggunaan memori.

Metrik akurasi akan dihitung sebagai proporsi prediksi yang benar dibandingkan dengan total gambar dalam dataset *testing*. Karena terdapat 10.000 total gambar pada dataset *testing*, metrik akurasi didefinisikan sebagai:

$$Akurasi = \frac{Jumlah\ Prediksi\ Benar}{Total\ Gambar} \tag{9}$$

$$Akurasi = \frac{Jumlah\ Prediksi\ Benar}{10000}$$

Perbandingan metrik akurasi ini dilakukan untuk membandingkan seberapa signifikan penurunan akurasi yang terjadi dibandingkan dengan model biasa.

Metrik efisiensi waktu akan dihitung sebagai waktu rata-rata yang dibutuhkan model untuk melakukan prediksi pada dataset *testing*. Waktu rata-rata didapatkan dengan melakukan proses prediksi sebanyak n iterasi, kemudian membagi total waktu dengan n.

```python
def measure_inference_time(model, dataset, iterations=10):
    import time
    start_time = time.time()
    for _ in range(iterations):
        _ = model.predict(dataset, verbose=0)
    end_time = time.time()
    return (end_time - start_time) / iterations
```

Gambar 8. Potongan kode fungsi perhitungan efisiensi waktu
Sumber: arsip pengguna

Perbandingan metrik efisiensi waktu ini dilakukan untuk membandingkan seberapa baik model dengan *low-rank approximation* dapat mempercepat waktu prediksi.

Metrik penggunaan memori akan dihitung sebagai ukuran total parameter model *neural network* yang memengaruhi konsumsi memori. Perbandingan metrik penggunaan memori dilakukan untuk membandingkan seberapa efektif kompresi data yang dilakukan pada model dengan *low-rank approximation*.

IV. Hasil dan Pembahasan

A. Hasil

Perbandingan hasil dilakukan antara model *neural network* biasa dengan model *low-rank approximation* dengan rank: $1, 5, 10, 20, 25, 50, 100$. Dengan menggunakan fungsi-fungsi dan metrik evaluasi yang telah didefinisikan pada bagian metodologi, hasil yang didapatkan adalah sebagai berikut:

Tabel 3. Tabel Perbandingan Hasil Evaluasi Performa (1)

Rank	Accuracy	Time Efficiency (s)	Total Parameter
Original	0.9727	1.031	1,467,726
1	0.0974	0.5066	6,192
5	0.2154	0.5136	23,736
10	0.6888	0.5241	45,666
15	0.8707	0.6032	66,266
20	0.9502	0.6917	86,866
25	0.9651	0.6965	107,466
50	0.9710	0.7572	210,466
100	0.9716	0.8990	416,466

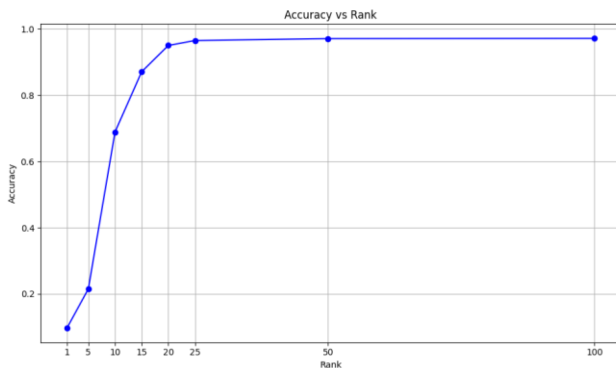
Untuk pemahaman yang lebih baik terhadap data yang ada, berikut adalah tabel yang lebih menggambarkan pengaruh jumlah rank pada model dengan *low-rank approximation* terhadap metrik evaluasi jika dibandingkan dengan model *original* (biasa).

Tabel 4. Tabel Perbandingan Hasil Evaluasi Performa (2)

Rank	Accuracy	Speedup	Memory	Memory Saving (%)
Original	0.9727	1.00x	5.60 MB	0.00
1	0.0974	2.04x	0.02 MB	99.58
5	0.2154	2.01x	0.09 MB	98.38
10	0.6888	1.97x	0.17 MB	96.89
15	0.8707	1.71x	0.25 MB	95.49
20	0.9502	1.49x	0.33 MB	94.08
25	0.9651	1.48x	0.41 MB	92.68
50	0.9710	1.36x	0.80 MB	85.66
100	0.9716	1.15x	1.59 MB	71.63

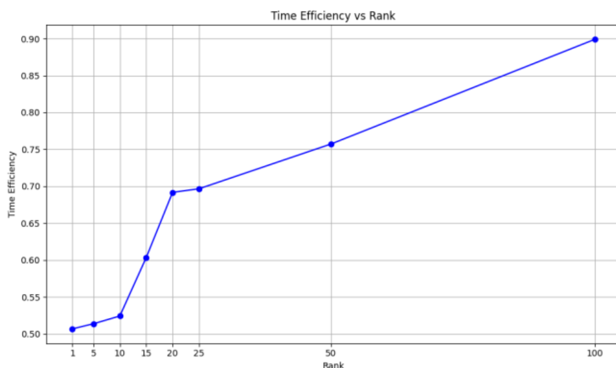
B. Pembahasan

Berdasarkan tabel hasil, dapat dilihat bahwa teknik kompresi menggunakan *low-rank approximation* berhasil menurunkan sumber daya komputasi, mulai dari waktu untuk melakukan prediksi hingga besar memori yang dibutuhkan. Untuk membandingkan pengaruh jumlah rank yang diambil, tabel akan divisualisasikan menggunakan grafik pada tiap metrik evaluasi.



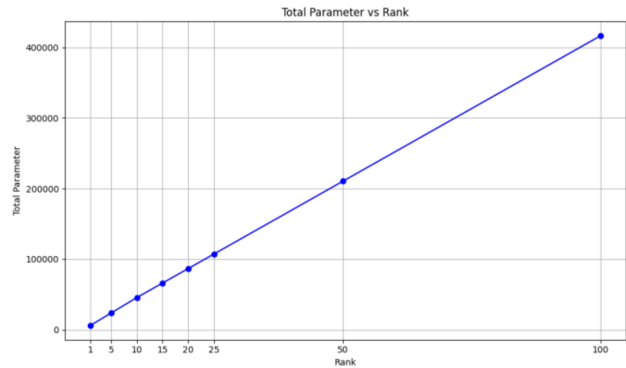
Gambar 9. Grafik Perbandingan Akurasi Terhadap Rank
Sumber: arsip pengguna

Berdasarkan Tabel 3. dan grafik pada Gambar 9., dapat dilihat bahwa semakin tinggi nilai rank r yang diambil untul *low-rank approximation*, semakin dekat nilai akurasi yang didapatkan dengan nilai akurasi pada model biasa. Namun, pertumbuhan terjadi secara logaritmik, sehingga nilai akurasi pada rank besar tidak akan berbeda terlalu jauh antara satu dan lainnya.



Gambar 10. Grafik Perbandingan Efisiensi Waktu Terhadap Rank
Sumber: arsip pengguna

Berdasarkan Tabel 3. dan grafik pada Gambar 10., sama seperti grafik sebelumnya, semakin tinggi rank yang diambil, semakin besar juga waktu komputasi yang dibutuhkan (semakin mendekati waktu komputasi model *original*. Grafik terlihat tidak memiliki suatu pola pertumbuhan tertentu, dikarenakan terdapat banyak faktor eksternal yang dapat memengaruhi waktu komputasi, contohnya tugas dari aplikasi lain yang sedang berjalan di latar belakang secara bersamaan. Karena faktor eksternal ini, waktu komputasi dapat menjadi cukup fluktuatif. Pada dataset sederhana seperti yang digunakan sekarang, tingkat fluktuatifitas ini terlihat cukup besar, karena perbedaan waktu komputasi ~0.1 detik akan terlihat cukup signifikan. Namun, pada dataset yang lebih besar dan kompleks, tingkat fluktuasi waktu ini seharusnya tidak akan memberikan dampak yang besar.



Gambar 11. Grafik Perbandingan Total Parameter Terhadap Rank
Sumber: arsip pengguna

Berdasarkan Tabel 3. dan grafik pada Gambar 11., terlihat jelas bahwa semakin tinggi nilai rank r yang diambil, semakin tinggi juga memori/total parameter yang dibutuhkan. Pertumbuhan ini terjadi secara linear, berbeda dari dua grafik sebelumnya.

Maka, secara intuitif, terlihat bahwa nilai rank r yang paling optimal untuk dipiliah adalah 25. Hal ini karena pemilihan rank 25 akan memiliki nilai akurasi yang sangat dekat dengan model *original*, dengan memori yang jauh lebih kecil dan perbedaan waktu yang cukup signifikan, yaitu sekitar 1.5 kali lebih cepat.

V. KESIMPULAN

Teknik kompresi data *low-rank approximation* dengan menggunakan SVD dapat mengoptimasi proses *forward propagation* pada model *neural network* yang telah dilatih. Hal yang penting untuk diperhatikan adalah pemilihan jumlah rank r yang akan diambil untuk menerapkan *low-rank approximation* pada matriks bobot. Nilai rank r perlu dipilih dengan tepat, agar hasil yang didapatkan optimal, yaitu dapat menurunkan sumber daya komputasi secara signifikan, tanpa kehilangan banyak informasi yang menyebabkan turunnya akurasi.

Pada MNIST Dataset yang digunakan pada makalah ini, didapatkan pilihan rank yang tepat pada model yang dibangun adalah $r = 25$, dengan akurasi yang tidak terlalu jauh dari model *original,* yaitu 96.51%, dengan waktu komputasi 1.5x lebih cepat daripada model *original* dan

penghematan memori hingga 92.68%. Perbedaan waktu ~0.3 detik dan perbedaan memori ~4 MB mungkin tidak terlihat signifikan pada dataset sederhana ini, tetapi pada dataset yang lebih besar dan lebih kompleks, waktu komputasi 1.5x lebih cepat dan kompresi memori 92% akan memberikan pengaruh yang signifikan.

VI. LAMPIRAN

Source code program implementasi dapat diakses pada tahutan berikut: https://github.com/albertchriss/nn-lowrank-approximation.git

VII. UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya, makalah yang berjudul "Penerapan *Singular Value Decomposition* (SVD) untuk Optimasi *Neural Networks* melalui *Low-Rank Approximation*" dapat diselesaikan dengan lancar tanpa adanya hambatan. Terima kasih juga kepada dosen-dosen pengampu mata kuliah IF2123 Aljabar Linier dan Geometri, terutama kepada Bapak Rila Mandala karena telah membagikan ilmu yang mendukung pembuatan makalah ini.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org. [Accessed: Dec. 28, 2024].

[2] R. Munir, "Singular Value Decomposition (Bagian 1)," Institut Teknologi Bandung (ITB), 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf. [Accessed: Dec. 29, 2024].

[3] R. Munir, "Singular Value Decomposition (Bagian 2)," Institut Teknologi Bandung (ITB), 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf. [Accessed: Dec. 29, 2024].

[4] Stanford University, "Lecture 9: Singular Value Decomposition (SVD)," Stanford University, [Online]. Available: https://web.stanford.edu/class/cs168/l/l9.pdf. [Accessed: Dec. 29, 2024].

[5] T.J. Machine Learning, "Neural networks 2." [Online]. Available: https://tjmachinelearning.com/lectures/1718/nn2/nn2.pdf.. [Accessed: Dec. 28, 2024].

[6] Y. LeCun, "The MNIST Database of Handwritten Digits," [Online]. Available: https://yann.lecun.com/exdb/mnist/. [Accessed: Dec. 28, 2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 Desember 2024

Albertus Christian Poandy 13523077