Peran Rust dalam Pengembangan Algoritma Machine Learning yang Efisien

Andriat Ratyanto
Program Studi Magister Teknik Informatika Universitas Pamulang
e-mail: andriat.r@gmail.com

***Abstrak*—Kebutuhan akan efisiensi dalam pengembangan algoritma *machine learning* semakin mendesak, dan dalam konteks ini, Rust, sebuah bahasa pemrograman sistem (*system language programming)* yang dirancang dengan focus pada kinerja tinggi, keamanan dan pemrograman konkurensi. Rust dapat diintegrasikan dengan kerangka kerja *machine learning* seperti TensorFlow, membuka peluang untuk mengembangkan model dengan kinerja tinggi. Potensi Rust dalam pemrosesan data parallel juga membuka kemungkinan percepatan komputasi dalam *data science*.**

***Kata Kunci*— Machine Learning; Rust; Efisiensi; Kinerja Tinggi; Pemrograman Paralel.**

I. PENDAHULUAN

Selama dekade terakhir, kemajuan teknologi infromasi dan komputasi telah mengubah lanskap pemrosesan data, sehingga mendorong kebutuhan akan algoritma *machine learning* yang efisien. Kecepatan, akurasi, dan ketahanan perangkat lunak dalam menghadapi tugas pemrosesan data yang semakin kompleks dan banyak menjadi hal yang penting. Dalam konteks ini, bahasa pemrograman Rust muncul sebagai faktor kunci yang mempengaruhi peran pentingnya dalam mengatasi tantangan tersebut.

Makalah ini bertujuan untuk merinci dan menggali lebih dalam peran Rust dalam mengembangkan algoritma *machine learning* yang efisien. Pertama, kita akan mengeksplorasi algoritma potensi performa Rust, bagaimana bahasa tersebut dapat menghasilkan kode dengan komputasi yang cepat dan efisien, terutama dalam konteks perhitungan matematis yang intensif yang biasa ditemukan dalam *machine learning*. Selanjutnya, kami akan membahas aspek penting dari keamanan perangkat lunak, yang menjadi focus Rust melalui sistem pemrograman ketat (*strict programming system*), yang membantu menghindari kesalahan memori yang sering terjadi pada bahasa pemrograman lain.

Kami juga akan membahas integrasi Rust dengan framework *machine learning* yang ada seperti TensorFlow, sehingga membuka peluang untuk mengembangkan model *machine learning* berperforma tinggi. Potensi Rust dalam pemrosesan data paralel juga akan menjadi fokus, mengingat pentingnya pemrosesan data secara bersamaan dalam lingkungan *data science*.

Dengan demikian, makalah ini akan menggambarkan bagaimana Rust menawarkan solusi menarik dalam mengembangkan algoritma *machine learning* yang efisien, memperkuat peran penting Rust dalam tantangan *data science modern* yang semakin mendesak.

II. METODE PENELITIAN

Untuk menjalankan penelitian yang mendalam tentang peran Rust dalam pengembangan algoritma machine learning yang efisien, Kami menggunakan metode penelitian sebagai berikut:

A. Perbandingan dengan Bahasa Lain

Dalam dunia *machine learning*, bahasa pemrograman yang paling umum digunakan adalah bahasa pemrograman *Python*. *Python* dikembangkan pertama kali oleh Guido van Rossum dan dirilis pada tahun 1991. Versi awal *Python*, yang dikenal sbagai *Python* 0.9.0, muncul pada bulan Februari 1991. *Python* 1.0 dirilis pada bulan Januari 1994 dan merupakan versi pertama yang dikenal sebagai *Python*. *Python* sangat populer dalam machine learning dengan beberapa alasan utama yaitu:

1) *Ekosistem yang Kuat*
   Python memiliki ekosistem yang sangat kuat dan beragam dalam konteks *machine learning*. Ada banyak perpustakaan (*library*), kerangka kerja dan alat yang dirancang khusus untuk pengembangan dan penelitian *machine learning*, seperti *TensorFlow*, *PyTorch*, *scikit-learn*, *Keras*, *Pandas* dan *NumPy*. Hal ini membuat *Python* menjadi bahasa yang sangat cocok untuk eksplorasi data, pengembangan model dan analisis statistik.
2) *Kemudahan penggunaan*
   Python dikenal karena sintaksisnya yang mudah dibaca dan ditulis. Hal ini membuatnya sangat ramah bagi pengembang pemula dan peneliti yang bukan *programmer* sejati. Kemudahan penggunaan *Python* memungkinkan pengembang untuk dengan cepat merancang, menguji dan mengimplementasikan ide-ide dalam *machine learning* tanpa harus memikirkan detail teknis yang rumit.

3) *Komunitas yang besar*
   Python memiliki salah satu komunitas pengembang yang paling besar di dunia. Komunitas ini aktif dan penuh semangat, menghasilkan banyak sumber daya, tutorial dan dukungan. Hal ini membuatnya mudah untuk memecahkan masalah dan mendapatkan bantuan saat menghadapi tantangan dalam *machine learning*.
4) *Cross-Platform*
   Python adalah bahasa pemrograman lintas platform, yang berarti Kita dapat menjalankan kode *Python* di berbagai sistem operasi seperti *Windows*, *macOS*, dan *Linux*. Ini penting dalam pengembangan dan penelitian *machine learning* yang sering melibatkan berbagai lingkungan.
5) *Integrasi dengan bahasa lain*
   Python dapat dengan mudah berintegrasi dengan kode yang ditulis dalam bahasa lain seperti C, C++, dan Java. Hal ini memungkinkan pengembang untuk mengoptimalkan bagian-bagian kritis dari aplikasi mereka dan memanfaatkan perpustakaan eksisting.
6) *Akses ke data*
   Python memiliki alat-alat yang kuat untuk mengakses, memanipulasi, dan menganalisis data. *Pandas*, misalnya, adalah perpustakaan *Python* yang populer untuk manipulasi data tabular.

Sedangkan *Rust* dikembangkan pertama kali oleh *Mozila Research*, dengan proyek dimulai oleh Graydon Hoare pada tahun 2006. Versi awal *Rust* dikembangkan sebagai eksperimen dan prototipe internal *Mozzila*. *Rust* pertama kali diumumkan kepada publik pada tahun 2010. Kemudian, *Rust* melalui serangkaian perubahan desain dan iterasi sebelum mencapai versi stabil pertamanya, *Rust* 1.0, yang dirilis pada Mei 2015. *Rust* sejak itu telah terus berkembang dan mendapatkan popularitas, terutama pengembangan perangkat keras, sistem dan bahasa pemrograman yang aman. *Rust* semakin populer dalam lingkup *machine learning* dikarenakan beberapa hal berikut:

1) *Kinerja tinggi*
   Rust dikenal karena kinerjanya yang sangat tinggi. *Rust* adalah bahasa yang efisien dan sering digunakan dalam pengembangan perangkat sistem, yang menunjukkan potensinya dalam pengolahan data dan komputasi yang membutuhkan kinerja tinggi, seperti dalam *machine learning*.
2) *Ketahanan kesalahan*
   Rust memiliki sistem tipe yang ketat dan aturan peminjaman memori yang aman. Hal ini membuatnya lebih sulit bagi pengembang untuk membuat kesalahan yang umum terjadi, seperti kesalahan memori atau null pointer, yang sering ditemukan dalam bahasa pemrograman lain. Dalam *machine learning*, kesalahan semacam itu dapat memiliki dampak besar.
3) *Interopabilitas*
   Rust dapat berintegrasi dengan kode yang ditulis dalam bahasa lain seperti Python, C, atau C++. Kita dapat memanfaatkan kode yang ada, terutama jika Kita ingin mengoptimalkan bagian-bagian kritis dari model *machine learning* Kita dengan kode *Rust*.
4) *Pengembangan aman dan produktif*
   Meskipun *Rust* memerlukan pemahaman yang lebih dalam tentang manajemen memori, sistem yang aman dan fitur-fitur seperti pemrosesan pinjaman (borrowing) memungkinkan pengembang untuk menulis kode yang lebih aman dan efisien. Ini adalah keuntungan penting dalam mengembangkan model *machine learning* yang kompleks dan kritis.
5) *Komunitas yang berkembang*
   Meskipun komunitas *Rust* belum sebesar *Python*, *Rust* terus berkembang dan semakin banyak perpustakaan yang mendukung *machine learning*, seperti rust-tensorflow. Ini menunjukkan potensi *Rust* dalam pengembangan aplikasi *machine learning* di masa depan.

B. Pengujian Kinerja

Pengujian kinerja adalah metode untuk mengukur sejauh mana sebuah program atau sistem dapat menjalankan tugas tertentu dalam batasan waktu yang diberikan. Dalam konteks ini, Kami menguji kemampuan *Python* dan *Rust* dalam menangani iterasi data dalam jumlah yang sangat besar. Dalam melakukan pengujian kinerja Kami akan menggunakan pendekatan sederhana dengan menghasilkan daftar berisi 100.000.000 (seratus juta) angka 1 (satu). Kemudian kami melakukan iterasi melalu daftar ini dan menjumlahkannya serta mengukur waktu yang diperlukan untuk menyelesaikan penjumlahan tersebut. Kami menggunakan alat komputasi yang sama saat melakukan pengujian ini, yaitu dengan spesifikasi sbb:

1) *Prosesor Apple M1 dengan CPU 8 core (4 performance cores dan 4 efficient cores)*
2) *RAM 8GB LPDDR4X*
3) *SSD 512GB dengan kecepatan baca dan tulis kurang lebih 3000MB/s*
4) *Operating System MacOS Sonoma 14.0*

III. HASIL DAN PEMBAHASAN

Berikut contoh *source code* pada masing-masing bahasa:

1) *Python*

```python
import time

# Generate a list of 10 million one (for illustration purposes)
data = [1] * 100000000

# Measure execution time
start_time = time.time()
sum = 0
for value in data:
    sum += value
    pass
end_time = time.time()

print(f"Total: {sum}")
print(f"Python execution time: {end_time - start_time} seconds")
```

2) *Rust*

```rust
use std::time::Instant;

fn main() {
    // Generate a vector with 10 million one (for illustration purposes)
    let data: Vec<i32> = vec![1; 100_000_000];

    // Measure execution time
    let start_time = Instant::now();
    let mut sum = 0;
    for value in &data {
        sum += value;
    }
    let end_time = start_time.elapsed();

    println!("Total: {}", sum);
    println!("Rust execution time: {} seconds", end_time.as_secs_f64());
}
```

Dari hasil pengujian source code diatas didapatkan hasil sebagai berikut:

Tabel 1.
Hasil Pengujian Source Code

Bahasa Pemrograman	Percobaan ke- (dalam detik)				
	1	2	3	4	5
Phython	5.26879096	4.58498311	5.08577704	5.12177801	4.78628683
Rust	1.27840263	1.26633125	1.2783505	1.27095313	1.28097771
Selisih	3.99038834	3.31865186	3.80742654	3.85082489	3.50530912
Persentase	24%	28%	25%	25%	27%

Bila Kita lihat dari tabel diatas *Rust* mampu melakukan iterasi sebanyak 100 juta data sekitar 25.8% lebih cepat dibandingkan dengan *Python*. Ini adalah hasil yang mengesankan dan menggambarkan salah satu keunggulan utama *Rust* dalam hal kinerja. Efisiensi penggunaan CPU dan memori adalah hal penting karena berdampak pada kinerja dan penggunaan daya. Dengan manajemen memori yang ketat dan optimasi kode yang baik, *Rust* dapat menghasilkan aplikasi yang lebih efisien dalam penggunaan sumber daya. Hal ini menjadi penting dalam aplikasi yang memerlukan kinerja tinggi dan pada perangkat dengan btasan daya khususnya dalam kaitannya dengan *machine learning* yang membutuhkan komputasi yang sangat tinggi.

A. Analisis Keamanan

Dalam analisis keamanan antara Rust dan Python terkait machine learning, kita akan mengevaluasi dan membandingkan bagaimana kedua bahasa pemrograman ini memengaruhi keamanan dalam pengembangan aplikasi machine learning. Keamanan dalam konteks machine learning sangat penting, terutama ketika model-machine learning digunakan untuk mengambil keputusan kritis atau menangani data sensitif. Baik Rust maupun Python memiliki kelebihan dan tantangan unik dalam hal keamanan, dan pemahaman yang mendalam tentang kekuatan dan kelemahan keduanya akan membantu pengembang memilih platform yang sesuai untuk proyek machine learning mereka. Mari kita mulai dengan menyelami aspek keamanan dalam konteks machine learning antara Rust dan Python.

1) Keamanan pada Tingkat Memori

Rust dikenal karena manajemen memori yang ketat, sementara Python memiliki pendekatan manajemen memori yang lebih otomatis. Di bawah ini, Kami akan menjelaskan perbandingan keamanan memori antara keduanya dan memberikan contoh kode untuk mengilustrasikan perbedaan ini.

a) Rust (Keamanan Memori Ketat)

Rust menggunakan konsep *ownership*, yang berarti setiap nilai memiliki satu pemilik. Hal ini memungkinkan *Rust* untuk memastikan bahwa sumber daya dibebaskan secara benar dan tidak ada kesalahan memori seperti *dangling pointers* atau *double frees*. *Rust* juga memiliki *borrow checker* yang memeriksa bagaiman nilai dipinjam dan digunakan untuk menghindari konflik memori yang berpotensi berbahaya. Contoh kode pemrograman *Rust* yang menunjukan keaman Memori:

```
fn main() {
    let mut s = String::from("Halo");
    let r1 = &s; // Peminjaman read-only
    let r2 = &s; // Peminjaman read-only
    // s.push_str(", world!"); // Error, karena s sedang dipinjam
    println!("r1: {}, r2: {}", r1, r2);
    // output code diatas adalah r1: Halo r2: Halo
}
```

Pada baris kode diatas Rust tidak memungkinan untuk merubah suatu variabel yang sedang dipinjam sampai variabel peminjam tersebut digunakan, dapat dilihat dari code dibawah:

```
fn main() {
    let mut s = String::from("Halo");
    let r1 = &s; // Peminjaman read-only
    let r2 = &s; // Peminjaman read-only
    println!("r1: {}, r2: {}", r1, r2);
    // output code diatas adalah r1: Halo r2: Halo
    // variabel r1 dan r2 telah digunakan pada saat print line
    // Peminjaman berakhir di sini

    // setelah peminjaman maka ownershipnya dikembalikan pada s
    s.push_str(", dunia!"); // s dapat digunakan kembali
    println!("s: {}", s);
    // output code diatas adalah s: Halo, dunia!
}
```

Dari contoh kode pemrograman diatas kita dapat mengilustrasikan konsep *ownership* dan *borowing* pada *Rust*. *Ownership* mengacu pada konsep kepemilikan nilai, sedangkan *borrowing* berkaitan dengan cara sementara meminjam referensi ke nilai tersebut. Kedua konsep ini berperan penting dalam mengelola memori dan mencegah kesalahan memori pada program *Rust*, antara lain mencegah masalah umum seperti *null pointer deferencing*, *use after free*, dan *data radce* yang sering terjadi dalam bahasa pemrograman lain.

b) Python (Manajemen Memori yang Otomatis)

Python menggunakan *garbage collection* dan *reference counting* untuk mengelola memori secara otomatis. Hal ini membuatnya lebih mudah digunakan, tetapi juga berarti *Python* bisa rentan terhadap *reference counting errors* dan *circular references*. *Python* juga memungkinkan Kita untuk melakukan hal seperti *aliasing* dan *mutability* dengan lebih sedikit batasan, yang berarti Kita harus lebih waspada terhadap beberapa kesalahan memori. Berikut adalah contoh kode pemrograman *Python* yang menunjukan bagaimana manajemen memori otomatis bekerja:

```
# Membuat objek string
data = "Halo, dunia!"

# Membuat objek lain yang merujuk ke objek yang sama
reference = data

# Objek data dan reference berbagi memori yang sama
# Python akan secara otomatis mengelola siklus referensi dan membebaskan memori ketika tidak ada referensi yang lagi merujuk

# Objek data tidak lagi digunakan
data = None  # Objek data akan dihapus secara otomatis oleh garbage collector

# Garbage collector akan membersihkan objek yang tidak lagi digunakan
```

Python secara otomatis mengelola memori untuk objek string ”Halo dunia!” dan membersihkan objek tersebut ketika tidak ada referensi yang lagi merujuk kepadanya. Kita tidak perlu khawatir tentang alokasi atau pembebasan memori secara manual. *Python* memastikan bahwa objek-objek yang tidak lagi digunakan akan dibersihkan oleh *garbage collector*, sehingga Kita dapat fokus pada logika dari aplikasi yang dibangun. Meskipun *Python* memiliki manajemen memori yang otomatis yang sangat baik, ada beberapa situasi dimana kesalahan masih bisa terjadi. Berikut contoh kesalah yang terkait dengan manajemen memori ototmatis *Python*:

(1) Circular Refences

Hal ini terjadi ketika ada lingkaran referensi antara objek yang masih saling merujuk, sehingga objek tersebut tidak akan pernah dihapus oleh *garbage collector*. Contoh kode pemrograman *Python* yang menyebabakan *circular references* adalah sebagai berikut:

```
class Node:
    def __init__(self):
        self.next = None

a = Node()
b = Node()
a.next = b
b.next = a  # Circular reference
```

(2) References Leak

References leak merupakan situasi dimana Kita lupa atau gagal untuk memutuskan mengatur objek menjadi *None* atau memutuskan referensinya setelah selesai digunakan, hal ini menyebabkan objek tetap berada di dalam memori lebih lama dari yang seharusnya. Contoh kode yang menyebabkan hal ini dapat dilihat pada kode dibawah:

```
data = [1, 2, 3]
reference = data
# Lupa menghapus referensi
```

2) Kerentanan Terhadap Eksekusi Kode Berbahaya

Analisis keamanan terhadap eksekusi kode berbahaya antara *Rust* dan *Python* melibatkan pertimbangan seberapa baik bahasa tersebut melindungi terhadap eksekusi kdoe berbahaya, seperti *injection attacks* atau *arbitrary code execution*. *Rust* dirancang untuk meminimalkan kemungkinan eksekusi kode berbahaya. Untuk mengeksekusi kode berbahaya pada *Rust*, Kita harus menggunakan *keyword unsafe*. Kita harus secara eksplisit menggunakan blok *unsafe* dan kode semacam ini seharusnya dihindari dalam praktik nyata. Berikut contoh penggunaannya:

```
fn main() {
    // Menggunakan blok 'unsafe' untuk mengeksekusi kode berbahaya
    unsafe {
        let pointer = 0x0 as *mut i32;
        *pointer = 42; // Eksekusi kode berbahaya
    }
}
```

Python memiliki tipe data dinamis, yang memungkinkan perubahan tipe data pada saat *runtime*. Meskipun hal ini membuat *Python* sangat fleksibel, juga dapat menjadi resiko jika *input* pengguna tidak difilter dengan benar. Contoh fungsi yang

memungkinkan eksekusi kode dinamis pada *Python* antara lain eval() dan exec(), jika *input* pengguna digunakan dalam fungsi ini tanpa validasi dapat menyebabkan eksekusi kode berbahaya. Berikut contoh kode yang beresiko:

```
user_input = "__import__('os').system('rm -rf /')"
eval(user_input)  # Ini dapat mengeksekusi kode berbahaya
```

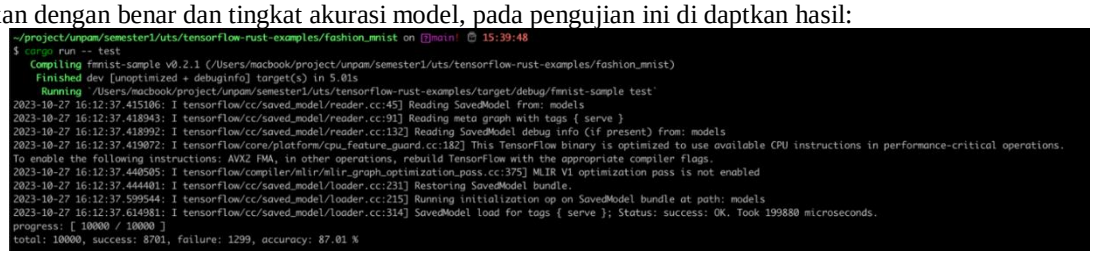
Fungsi diatas memungkinkan pengguna menghapus system *file* pada *folder root*, yang dapat menyebabkan kerusakan pada sistem. Dalam banyak hal, Rust menawarkan lebih banyak alat untuk mencegah eksekusi kode berbahaya dan meminimalkan kerentanan keamanan. Hal Ini disebabkan oleh fitur-fitur unik yang dimiliki oleh Rust yang dirancang untuk menghindari eksekusi kode berbahaya dan mencegah kerentanan keamanan.

B. Integrasi dengan Kerangka Kerja Machine Learning

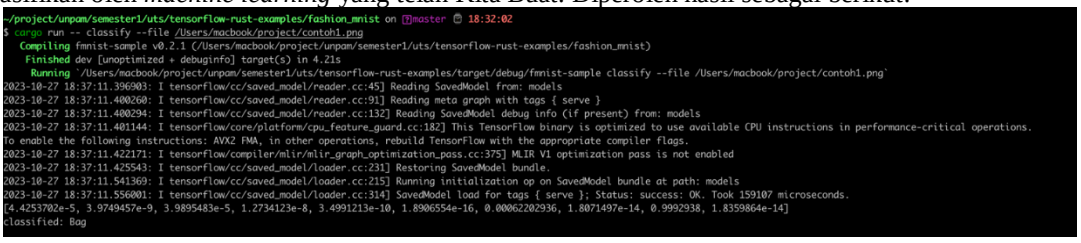
Mesikpun Python adalah bahasa yang paling umum digunakan dalam data science dan machine learning. Rust adalah bahasa perograman yang dapat memberikan keuntungan dalam aspek kecepatan dan keamanan eksekusi. Dalam makalah ini kami akan mencoba menggunakan Rust dalam menjalankan sebuah machine learning dalam mendeteksi gambar, dengan menggunakan TensorFlow untuk mengolah dataset MNIST (Modified Institute of Standards and Technology) yang merupakan dataset standard yang digunakan untuk menguji dan membandingkan berbagai model dan algoritma dalam pengenalan objek. Dalam hal ini Kami persempit kembali dengan menggunakan Fashion MNIST yaitu dataset yang berisi gambar-gambar pakaian dan aksesori fashion. Dataset ini berisi 60.000 (enam puluh ribu) gambar pelatihan dan 10.000 (sepuluh ribu) gambar pengujian. Setiap gambar berukuran 28x28 piksel dan termasul dalam salah satu dari 10 kategori fashion yang berbeda, seperti sepatu, baju, tas, sendal, topi dan lainnya. Tujuan dari pengguanan dataset ini adalah untuk menguji dan melatih model machine learning dan neural network dalam mengenali jenis pakaian dan fashion yang terkandung dalam gambar. Algoritma ini adalah tugas multi kelas dimana model berusaha untuk mengkalisifikasikan gambar kedalam salah satu dari 10 kategori fashion yang berbeda. Berikut adalah contoh pengkodean machine learning TensorFlow dengan bahasa pemrograman Rust pada tautan berikut https://github.com/andriat41/fashion-mnist/tree/master.

Langkah pertama Kita menjalankan generate.py dengan cara eksekusi tugas python3 generate.py. kode ini mengunduh dataset Fashion MNIST yang berisi gambar-gambar pakaian seperti baju, celana, dan sepatu. Data pelatihan dan pengujian dari dataset tersebut dimuat ke dalam variabel yang sesuai. Selain itu, sebuah daftar kelas yang merepresentasikan jenis pakaian (seperti 'Top', 'Trouser', 'Pullover', dll.) juga didefinisikan. Selanjutnya, terdapat beberapa fungsi yang membantu dalam proses persiapan data. Fungsi save_img mengambil data gambar dan menyimpannya sebagai file gambar PNG. Fungsi dump digunakan untuk mengorganisasi dan menyimpan gambar-gambar dari dataset ke dalam folder dengan struktur direktori yang sesuai dengan kelasnya. Data gambar dari dataset kemudian dinormalisasi, yaitu nilai piksel dalam gambar dibagi dengan 255.0, sehingga rentang nilai berada antara 0 hingga 1. Setelah itu, model jaringan saraf tiruan dibangun menggunakan *Keras*. Model ini terdiri dari tiga lapisan: input layer yang meratakan gambar, hidden layer dengan 128 unit dan aktivasi ReLU, serta output layer dengan 10 unit (sesuai dengan jumlah kelas) dan aktivasi softmax untuk menghasilkan probabilitas prediksi kelas. Model ini kemudian dikompilasi dengan pengoptimal 'adam', fungsi loss 'sparse_categorical_crossentropy' (cocok untuk klasifikasi dengan label integer), dan metrik akurasi. Selanjutnya, model dilatih dengan data pelatihan selama 5 epoch. Ini berarti model belajar mengenali dan mengklasifikasikan pakaian-pakaian berdasarkan gambar-gambar pelatihan. Akhirnya, model yang telah dilatih disimpan dalam folder 'models'. Dengan demikian, kode ini lengkap dalam mempersiapkan data, membangun, melatih, dan menyimpan model untuk tugas klasifikasi gambar pakaian menggunakan dataset Fashion MNIST.

Kemudian Kita jalankan perintah ”cargo run -- test” pada skrip di atas akan memicu pengujian akurasi model klasifikasi terhadap gambar-gambar uji yang tersedia dalam direktori "images". Hasil pengujian akan mencakup jumlah gambar yang diklasifikasikan dengan benar dan tingkat akurasi model, pada pengujian ini di daptkan hasil:



Dari 10.000 (sepuluh ribu) data yang di testing, diperoleh data 8701 data yang berhasil dan 1299 data yang gagal, maka didapatkan tingkat akurasinya adalah 87,01%. Kemudian Kita lakukan klasifikasi gambar dan mencetak hasilnya dengan menjalankan ”cargo run -- classify --file path/ke/gambar_anda.png”, pada pengujian kali ini Kami mencoba data tas ”contoh1.png” () untuk diklasifikan oleh *machine learning* yang telah Kita Buat. Diperoleh hasil sebagai berikut:



Setelah gambar tas diujicobakan menggunakan program yang telah disediakan, hasil klasifikasi menunjukkan bahwa gambar tersebut termasuk dalam kategori "Bag" (tas). Hasil ini menunjukkan bahwa model klasifikasi telah mengenali dan mengklasifikasikan gambar tas dengan benar, sesuai dengan label kategori yang telah ditentukan sebelumnya. Ini adalah indikasi bahwa model tersebut berhasil mengenali jenis pakaian yang ada pada gambar, dan klasifikasinya sesuai dengan ekspektasi.

IV. KESIMPULAN

Hasil penelitian ini menegaskan Rust memiliki potensi besar dalam pengembangan algoritma *machine learning* yang efisien. Sebagai bahasa pemrograman sistem yang menekankan pada kinerja yang tinggi, keamanan, dan pemrograman konkurensi, Rust menjunjukan kemampuan untuk menghasilkan kode yang sangat cepat dan efisien dengan keterbatasan mesin komputasi yang digunakan. Keunggulan Rust dalam hal kemanan perangkat lunak juga manjadi poin penting, dengan sistem pemrograman ketat yang mencegah kesalahan memori yang sering terjadi dalam bahasa pemrograman lain. Selain itu, kemampuan Rust untuk terintegrasi dengan kerangka kerja *machine learning* yang populer seperti TensorFlow membuka peluang untuk mengembangkan model *machine learning* dengan kinerja tinggi. Meskipun ada tantangan yang perlu diatasi, seperti *learning curve* yang sangat tinggi, Rust tetap menjadi alternatif menarik dalam ekosistem *machine learning* yang semakin berkembang, terutama untuk aplikasi yang membutuhkan kinerja dan keamanan tinggi.

DAFTAR PUSTAKA

[1] Claus Matzinger, "*Hands-On: Data Structures and Algorithms with Rust*" (Jan 2019)
[2] Youtube Channel Code to the moon (https://www.youtube.com/@codetothemoon), "*Is Rust the New King of Data Science?*" (Nov 23, 2022) https://youtu.be/mlcSpxicx-4?si=GjPfHZMkNt4maYwA
[3] Youtube Channel No BoilerPlate, (https://www.youtube.com/c/NoBoilerplate), "*Rust makes you feel like a Genius"* (Mei 26, 2022) https://www.youtube.com/watch?v=0rJ94rbdteE&ab_channel=NoBoilerplate
[4] Joydeep Bhattacharjee, "Practical Machine Learning with Rust" (2020)
[5] Kyle Gallatin & Chris Albon, "*Machine Learning with Python Cookbook 2nd Editoin" (Jul 2023)*
[6] Denis Rothman, "Artificial Intelligence by Example 2nd Editoin" (Feb 2020)