

**PENERAPAN METODE CONVOLUTIONAL NEURAL
NETWORK TERHADAP CITRA GRayscale DAN
PSEUDOCOLOR RGB UNTUK KLASIFIKASI MALWARE**

TUGAS AKHIR

Chris Christian

1118041



INSTITUT
TEKNOLOGI
HARAPAN
BANGSA

Veritas vos liberabit

**PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2022**

**PENERAPAN METODE CONVOLUTIONAL NEURAL
NETWORK TERHADAP CITRA GRayscale DAN
PSEUDOCOLOR RGB UNTUK KLASIFIKASI MALWARE**

TUGAS AKHIR

**Diajukan sebagai salah satu syarat untuk memperoleh
gelar sarjana dalam bidang Informatika**

Chris Christian

1118041



INSTITUT
TEKNOLOGI
HARAPAN
BANGSA

Veritas vos liberabit

**PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2022**

DAFTAR ISI

ABSTRAK	i
ABSTRACT	i
KATA PENGANTAR	i
DAFTAR ISI	i
DAFTAR TABEL	iii
DAFTAR GAMBAR	iv
BAB 1 PENDAHULUAN	1-1
1.1 Latar Belakang	1-1
1.2 Rumusan Masalah	1-2
1.3 Tujuan Penelitian	1-3
1.4 Batasan Masalah	1-3
1.5 Kontribusi Penelitian	1-3
1.6 Metodologi Penelitian	1-4
1.7 Sistematika Pembahasan	1-4
BAB 2 LANDASAN TEORI	2-1
2.1 Tinjauan Pustaka	2-1
2.1.1 <i>Malware Classification</i>	2-1
2.1.2 Visualisasi <i>Malware</i>	2-3
2.1.3 Principal Component Analysis (PCA)	2-4
2.1.4 Model Warna	2-7
2.1.4.1 RGB	2-8
2.1.4.2 <i>Grayscale</i>	2-8
2.1.4.3 <i>Pseudocolor</i>	2-9
2.1.5 Convolutional Neural Network	2-9
2.1.5.1 <i>Convolution Layer</i>	2-11
2.1.5.2 <i>Pooling Layer</i>	2-13
2.1.5.3 <i>Fully Connected Layer</i>	2-14
2.1.5.4 <i>Dropout Layer</i>	2-16
2.1.6 Fungsi Aktivasi	2-17

2.1.6.1	<i>Rectified Linear Unit (ReLU)</i>	2-17
2.1.6.2	<i>Softmax</i>	2-18
2.1.7	<i>Loss Function</i>	2-18
2.1.8	Regularisasi	2-19
2.1.8.1	Regularisasi L2	2-19
2.1.9	<i>Confusion Matrix</i>	2-20
2.2	Pustaka Python	2-22
2.2.1	Numpy	2-22
2.2.2	Pandas	2-22
2.2.3	OpenCV	2-23
2.2.4	Matplotlib	2-23
2.2.5	Scikit-learn	2-24
2.2.6	Keras	2-25
2.3	Tinjauan Studi	2-26
2.3.1	<i>State of The Art</i>	2-26
2.3.2	Pembahasan Penelitian Terkait	2-30
2.4	Tinjauan Objek	2-32
2.4.1	<i>Malware Family</i>	2-32
2.4.2	<i>Dataset Citra Grayscale</i>	2-32
BAB 3 ANALISIS DAN PERANCANGAN SISTEM		3-1
3.1	Analisis Masalah	3-1
3.2	Kerangka Pemikiran	3-2
3.3	Urutan Proses Global	3-4
3.3.1	Proses Pelatihan (<i>Training</i>)	3-5
3.3.2	Proses Pengujian (<i>Testing</i>)	3-6
3.4	Analisis Manual	3-7
3.4.1	Data Sampel	3-7
3.4.2	<i>Preprocessing</i>	3-11
3.4.2.1	Principal Component Analysis	3-11
3.4.2.2	<i>Pseudocolor</i>	3-12
3.4.3	Perhitungan Convolutional Neural Network	3-14

DAFTAR TABEL

2.1	Confusion Matrix	2-21
2.2	Daftar <i>Method</i> yang Digunakan dari Pustaka Numpy	2-22
2.3	Daftar <i>Method</i> yang Digunakan dari Pustaka Pandas	2-23
2.4	Daftar <i>Method</i> yang Digunakan dari Pustaka OpenCV	2-23
2.5	Daftar <i>Method</i> yang Digunakan dari Pustaka Matplotlib	2-24
2.6	Daftar <i>Method</i> yang Digunakan dari Pustaka Scikit-learn	2-25
2.7	Daftar <i>Method</i> yang Digunakan dari Pustaka Keras	2-26
2.8	<i>State of The Art</i>	2-27
2.9	<i>Family</i> dan Jenis <i>Malware</i>	2-33
3.1	Contoh Matriks Citra Masukan	3-14
3.2	Contoh Matriks Citra Setelah Diberi <i>Padding</i>	3-15
3.3	Contoh <i>Kernel</i> 3×3	3-15
3.4	Contoh Matriks <i>Feature Map</i> Hasil Konvolusi	3-16
3.5	Contoh Matriks <i>Feature Map</i> Setelah Perhitungan Fungsi Aktivasi ReLu	3-17
3.6	Contoh Hasil Perhitungan <i>Max Pooling</i>	3-17
3.7	Contoh Inisialisasi Random Number Proses <i>Dropout</i>	3-18
3.8	Contoh Hasil Perhitungan dalam <i>Dropout Layer</i>	3-18
3.9	Contoh <i>Kernel</i> pada <i>Dense Layer</i>	3-19
3.10	Prediksi Kelas Klasifikasi dengan <i>Softmax</i>	3-20

DAFTAR GAMBAR

2.1	<i>Malware</i> yang Dikemas dengan Teknik Enkripsi [11]	2-3
2.2	Proses Visualisasi Malware [1]	2-4
2.3	Contoh Citra RGB [14]	2-8
2.4	Contoh Citra <i>Grayscale</i>	2-9
2.5	Proses transformasi warna pada nilai intensitas keabuan [14]	2-9
2.6	Arsitektur Convolutional Neural Network [8]	2-10
2.7	<i>Convolutional Layer</i> dengan <i>Filter 3x3</i> [8]	2-12
2.8	<i>Max Pooling 2×2</i> [8]	2-14
2.9	<i>Fully Connected Layer</i> [20]	2-15
2.10	<i>Dropout Layer</i> [20]	2-16
2.11	Kurva Fungsi ReLu	2-17
2.12	Bagian dari Citra <i>Grayscale Malware</i> [12]	2-33
3.1	Kerangka Pemikiran	3-2
3.2	<i>Flowchart</i> Urutan Proses Global	3-4
3.3	<i>Flowchart</i> Proses Pelatihan	3-5
3.4	<i>Flowchart</i> Proses Pengujian	3-6
3.5	<i>Family Adialer.C</i>	3-10
3.6	<i>Family Dialplatform.B</i>	3-10
3.7	<i>Family Dontovo.A</i>	3-11
3.8	Kode Python untuk Principal Component Analysis	3-11
3.9	Citra <i>Grayscale Malware</i> setelah PCA dengan Berbagai Nilai <i>n_component</i>	3-12
3.10	Kode Python untuk <i>Pseudocolor</i>	3-13
3.11	Skala Warna COLORMAP_JET	3-13
3.12	Perbandingan Citra <i>Grayscale</i> dan Citra RGB <i>Malware Family Adialer.C</i>	3-14

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Malware (*malicious software*) adalah perangkat lunak yang dibuat untuk merusak atau menjalankan proses yang dapat membahayakan sistem komputer, contohnya *virus*, *worms*, *trojan*, dan *spyware* [1]. Dengan berkembangnya internet, banyak variasi *malware* baru bermunculan yang seringkali digunakan untuk melakukan penyerangan dan tindakan kejahatan. Sebagai contoh, menurut laporan dari AV Test, dalam periode 2011 – November 2020, melaporkan sebanyak total jumlah *malware* yang terdeteksi sebanyak 1113.73 juta dan sebanyak 267.23 juta varian *malware* baru muncul dalam kurun 2 tahun terakhir [2]. Oleh karena itu, diperlukan sistem deteksi dan klasifikasi *malware* yang efektif untuk mencegah dan mengurangi terjadinya serangan *malware* pada sistem komputer.

Pada saat ini, pendekatan berbasiskan visualisasi citra *grayscale malware* banyak dikembangkan untuk melakukan klasifikasi *malware* [1] [3] [4] [5]. Citra *grayscale malware* dinilai lebih efisien dalam masalah klasifikasi *malware* karena keseluruhan struktur antar *malware* dapat diukur melalui kesamaan dan perbedaan fitur dan tekstur secara visual pada citra *grayscale*. Pada penelitian [3] menerapkan algoritme transformasi *wavelet* yang digunakan untuk metode ekstraksi fitur, sedangkan model *machine learning* yang dikembangkan adalah K-Nearest Neighbor (KNN) dan Support Vector Machine (SVM) untuk klasifikasi *malware*. Dalam penelitian ini, *dataset* yang digunakan adalah Malware Image (Malimg) khususnya *malware* yang berjenis *Trojan*. Akurasi terbaik yang dihasilkan model KNN adalah sebesar 89.11%, serta model SVM menghasilkan akurasi sebesar 73.55%. Kelemahan pada penelitian [3] adalah klasifikasi *malware* yang dilakukan belum memakai keseluruhan data di dalam *dataset* Malimg.

Pada penelitian [4], model klasifikasi *machine learning* yang dikembangkan untuk klasifikasi *malware* adalah Random Forest dengan akurasi 97.47%, K-Nearest Neighbor (KNN) dengan akurasi 96.23%, dan Support Vector Machine (SVM) dengan akurasi 95.23%. Metode *machine learning* konvensional, seperti Random Forest, KNN, dan SVM memiliki kelemahan, yaitu peningkatan performa akan melambat seiring berkembangnya jumlah data masukan sehingga kurang cocok untuk pelatihan dengan volume data yang banyak.

Metode berbasiskan *deep learning* juga banyak dimanfaatkan untuk pengolahan citra *grayscale* untuk melakukan deteksi dan klasifikasi *malware* [1]

[5]. Penelitian [1] menerapkan metode Convolution Neural Network (CNN) untuk klasifikasi *malware* terhadap citra *grayscale malware* menghasilkan akurasi 98.52% pada *dataset* Malimg dan 98.99% pada *dataset* Microsoft. Selain CNN, penelitian [5] mengembangkan metode *deep learning* lainnya, yaitu Extreme Learning Machine (ELM) untuk klasifikasi *malware* dengan akurasi 93.9%. Berdasarkan penelitian [1], [3], [4], [5] yang sudah dibahas, metode CNN dinilai mampu menganalisis pola tersembunyi pada data citra *grayscale* untuk klasifikasi *malware* sehingga dapat menghasilkan akurasi yang lebih tinggi dibandingkan metode lainnya.

Penelitian [7] melakukan eksperimen klasifikasi *malware* dengan model CNN menggunakan citra RGB yang didapat dari *file malware* pada *platform* Android. Klasifikasi *malware* menggunakan citra RGB dinilai efektif dengan akurasi sebesar 98.77%.

Oleh karena itu, penelitian ini akan menerapkan metode CNN untuk melakukan klasifikasi *malware* menggunakan visualisasi citra *malware* dengan menggunakan teknik *pseudocolor*, serta Principal Component Analysis (PCA) untuk ekstraksi fitur. PCA dapat melakukan *dimension reduction* atau mengurangi dimensi pada masukan dengan menyeleksi jumlah komponen data yang optimal untuk mengekstrak fitur dengan informasi varians yang tinggi [6]. Kemudian, pada penelitian ini teknik *pseudocolor* dilakukan pada citra *grayscale malware* dari *dataset* untuk menghasilkan citra berwarna RGB, tujuannya adalah citra RGB memuat 3 *channel* warna, yaitu merah (*red*), hijau (*green*), dan biru (*blue*) sehingga memiliki informasi fitur yang lebih banyak dibandingkan citra *grayscale* yang hanya berisi 1 *channel* warna saja. Penelitian dibuat untuk membandingkan besar nilai akurasi pada klasifikasi terhadap citra *grayscale* dan citra RGB. Pengujian kinerja model klasifikasi CNN menggunakan *confusion matrix* untuk mengukur nilai akurasi, *recall*, dan *F-measure*.

1.2 Rumusan Masalah

Berikut ini adalah rumusan masalah yang menyangkut ide pokok dari setiap masalah yang akan dibahas dan dipecahkan melalui penelitian ini:

1. Berapa nilai akurasi dan *recall* dari metode Convolutional Neural Network untuk klasifikasi *malware* terhadap data citra *grayscale* maupun citra RGB ?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah di atas, berikut adalah tujuan dari penelitian tugas akhir ini:

1. Mengimplementasikan metode CNN dengan menerapkan PCA untuk melakukan klasifikasi *malware* menggunakan data visualisasi citra *malware*.
2. Mengimplementasikan dan mencari nilai *hyperparameter* terbaik untuk jumlah *convolution layer*, jumlah *filter*, *epoch*, dan *learning rate* terhadap model CNN untuk klasifikasi *malware*.
3. Melihat pengaruh citra RGB dibandingkan citra *grayscale* terhadap performa akurasi dan *recall* pada metode CNN untuk klasifikasi *malware* ?

1.4 Batasan Masalah

Agar penelitian ini lebih fokus dan terarah, maka peneliti akan memberikan beberapa batasan masalah sebagai berikut:

1. *Dataset* yang digunakan adalah Malware Image (Malimg) Dataset, berisi data citra *grayscale* yang terdiri dari 9339 citra dan 25 *malware family*.

1.5 Kontribusi Penelitian

Terdapat penelitian tugas akhir yang pernah dikembangkan terkait klasifikasi *malware* dengan mengimplementasikan metode Random Forest, tugas akhir tersebut berjudul "Penerapan Metode Random Forest Untuk Klasifikasi Malware Dengan Sumber Data Berbasis *Gray-Scale Image*". Sedangkan, pada penelitian ini akan mengambil pendekatan dengan metode yang berbeda untuk sistem klasifikasi *malware*, yaitu mengimplementasikan metode CNN dan PCA untuk ekstraksi fitur, serta membandingkan akurasi antara klasifikasi citra *grayscale* dan citra RGB. Berikut ini merupakan kontribusi penelitian dalam penelitian ini:

1. Menerapkan metode CNN sebagai metode klasifikasi dengan menggunakan teknik *pseudocolor* dan PCA untuk melakukan klasifikasi *malware* berdasarkan data visualisasi citra *malware*.
2. Membandingkan dan melakukan analisis pengaruh antara citra *grayscale* dan citra RGB terhadap performa akurasi dan *recall* pada sistem klasifikasi *malware* dengan metode CNN.

1.6 Metodologi Penelitian

Berikut ini merupakan metodologi penelitian yang dilakukan dalam penelitian ini:

1. Studi Literatur

Penulisan penelitian diawali dengan melakukan studi kepustakaan yang bersumber dari jurnal penelitian terkait topik yang sudah ada.

2. Data Sampling

Data sampling yang digunakan untuk penelitian berupa citra *grayscale malware*.

3. Analisis Masalah

Pada tahap ini, dilakukan analisis masalah berdasarkan batasan masalah yang ada.

4. Perancangan dan Implementasi

Pada tahap ini dilakukan pembangunan model pembelajaran mesin dengan metode CNN, serta pelatihan dengan kedua jenis data citra *grayscale* dan *RGB malware*.

5. Pengujian

Pada tahap ini dilakukan pengujian terhadap performa metode CNN dan mengukur keakuratan dalam melakukan klasifikasi *malware*.

6. Dokumentasi

Di tahap ini akan dilakukan dokumentasi hasil analisis dan implementasi secara tertulis dalam bentuk laporan metode penelitian.

1.7 Sistematika Pembahasan

Penelitian ini disusun berdasarkan sistematika penulisan sebagai berikut:

Bab I Pendahuluan

Bab ini menjelaskan latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, kontribusi penelitian, metodologi penelitian serta sistematika pembahasan.

Bab II Landasan Teori

Bab ini menjelaskan teori mengenai tinjauan pustaka, tinjauan studi, dan tinjauan objek yang mendukung implementasi penelitian ini.

Bab III Analisis dan Perancangan

Bab ini menjelaskan analisis terhadap masalah dan perancangan metode yang akan digunakan.

Bab IV Implementasi dan Pengujian

Bab ini menjelaskan implementasi dan pengujian pada sistem yang dikembangkan serta pengukuran evaluasi sistem berdasarkan nilai akurasi.

Bab V Kesimpulan dan Saran

Bab ini menjelaskan kesimpulan berdasarkan hasil sistem klasifikasi yang dibuat dan saran untuk pengembangan lebih lanjut di masa mendatang.

BAB 2 LANDASAN TEORI

2.1 Tinjauan Pustaka

Bagian ini menjelaskan dasar teori terkait sistem klasifikasi *malware* dalam penelitian ini.

2.1.1 *Malware Classification*

Malware atau *malicious software* merupakan *file* atau kode yang sengaja dibuat untuk merusak dan menyerang sistem komputer sehingga merusak aspek integritas, kerahasiaan, dan fungsionalitas sistem. Dengan berkembangnya penggunaan internet yang masif, menyebabkan meningkatnya ancaman keamanan akibat serangan *malware* yang mungkin terjadi pada organisasi perusahaan ataupun pengguna umum. Selain dapat menginfeksi sistem komputer, *malware* juga dapat menyebabkan pencurian data berharga milik pengguna, merusak sistem komputer, serta mengambil akses dan kontrol terhadap perangkat komputer [8].

Malware yang bermunculan saat ini ada banyak jenisnya yang dikelompokkan berdasarkan dengan sifat dan cara menginfeksinya, seperti *virus*, *worm*, *trojan*, *ransomware*, dan *spyware*. Jenis atau kelas *malware* lebih jauh dibagi lagi ke dalam beberapa varian atau disebut *malware family*. Salah satu penyebab tingginya volume sampel *malware* adalah *file malware* yang berbahaya secara terus-menerus dimodifikasi dengan teknik pemrograman, seperti transformasi *polymorphic* dan *metamorphic* dengan tujuan untuk menyamarkan dan menghindari deteksi *malware* [1].

Terdapat 2 jenis analisis *malware* yang umum digunakan pada metode deteksi dan klasifikasi *malware* yaitu, analisis statis dan analisis dinamis. Analisis statis adalah analisis yang dilakukan dengan menganalisis struktur *source code* di dalam program tanpa menjalankan atau mengeksekusi program tersebut untuk mendeteksi *malware*. Sedangkan, pendekatan analisis dinamis dapat dilakukan dengan menjalankan *file* atau program *malware* untuk menganalisis dampaknya terhadap sistem. Analisis statis lebih banyak dipilih karena lebih cepat dan mudah dalam pengumpulan data yang diperlukan, serta beban komputasi lebih rendah dibandingkan analisis dinamis karena tidak perlu menjalankan sampel *malware* [8].

Klasifikasi *malware* atau *malware classification* adalah proses untuk menetapkan atau melabeli suatu sampel *malware* ke dalam suatu *malware family*

yang spesifik. Metode klasifikasi dan deteksi *malware* terdiri dari *signature-based*, *behaviour-based*, dan *heuristic-based* [9].

Signature-based merupakan metode analisis statis yang digunakan untuk mendeteksi *malware* menggunakan *signature* milik *malware* yang dapat diidentifikasi secara unik. *Signature* merupakan pola *byte* yang unik pada struktur program di dalam suatu *malware*. Deteksi suatu sampel *file* dianggap *file* berbahaya atau tidak dengan melakukan perbandingan antara *signature* dari sampel *file* tersebut dengan *signature* *malware* yang sudah dikumpulkan. Kebanyakan aplikasi *anti-virus* yang beredar biasanya menggunakan metode *signature-based* dan mempunyai *database* tersendiri untuk menyimpan *malware signature*. Pendekatan ini cepat dan efektif untuk mendeteksi *malware family* yang sudah diketahui sebelumnya, tetapi tidak bisa mendeteksi varian atau *malware family* baru, serta masih memerlukan keterlibatan manusia [9].

Behaviour-based merupakan pendekatan mendeteksi *malware* dengan melakukan observasi pada perubahan-perubahan yang terjadi pada sistem akibat *malware*. Perubahan pada sistem dapat dilihat dari pemanggilan API, perubahan *registry*, dan monitoring pada jaringan, apakah perubahan-perubahan tersebut menyebabkan anomali atau kerusakan pada sistem untuk menentukan sampel *file* tersebut merupakan berbahaya atau tidak. Metode *behaviour-based* ini tergolong ke dalam analisis dinamis dan dijalankan pada *sandbox environment*, seperti *virtual machine*. Kelemahan pada metode ini adalah dampak *malware* pada *virtual machine* bisa saja berbeda dengan *environment* yang sebenarnya.

Heuristic-based merupakan metode pendekatan mendeteksi *malware* dengan sekumpulan *rules* atau aturan untuk menentukan apakah suatu sampel *file* merupakan *malware* atau tidak. Aturan atau *rules* yang dipakai dalam pendekatan *heuristic* ini didapatkan dari hasil analisis baik dalam analisis *source code* dan *behaviour malware*. Setelah aturan ditentukan, sampel *file* akan diuji seberapa besar persentase kesesuaian sample *file* dengan aturan yang sudah dibuat, jika suatu sampel memenuhi persentase tertentu bisa dianggap sebagai *malware*. Metode *heuristic-based* bisa digunakan untuk menentukan varian *malware* yang belum diketahui sebelumnya.

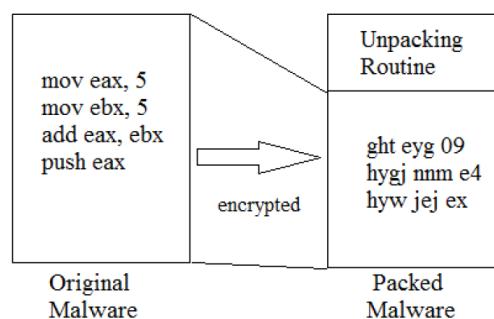
Klasifikasi *malware* berdasarkan citra *grayscale* dalam penelitian ini merupakan pengembangan metode analisis statis dengan menggunakan citra *grayscale*. Citra *grayscale* didapatkan dari *bytes* di dalam *file malware*, lalu dilakukan pemetaan ke dalam *pixel* sehingga membentuk citra *grayscale*. Dengan

citra *grayscale*, setiap perubahan struktur pada *file* atau program dapat lebih mudah terdeteksi sehingga dapat mengatasi transformasi *polymorphic* dan *metamorphic* menjadikan metode ini efektif pada deteksi dan klasifikasi *malware*.

2.1.2 Visualisasi *Malware*

Dalam konteks keamanan siber, *malware* terdiri dari kode-kode yang memuat pola-pola berbahaya untuk melakukan akses yang tidak terautentikasi dan penyerangan pada sistem komputer. Pendekatan analisis statis konvensional, seperti deteksi *signature*, bekerja dengan memeriksa kode dan *signature* untuk mengenali pola-pola mencurigakan yang terdapat dalam *malware*. Analisis statis membutuhkan analisis kode secara menyeluruh untuk mendeteksi suatu *malware*, tetapi salah satu tantangan terbesar pada analisis kode adalah *code obfuscation*.

Code obfuscation digunakan untuk kompresi dan enkripsi pada kode *malware* yang sebenarnya menjadi kode yang berbeda, tetapi memiliki kesamaan secara semantik untuk menghindari deteksi *malware* [10]. Pada saat *malware* dieksekusi oleh sistem operasi, kode *malware* yang asli akan dikembalikan saat dimuat dalam RAM sehingga *malware* bisa menjalankan kode-kode berbahaya di dalam sistem. Gambar 2.1 merupakan salah satu contoh *code obfuscation malware* yang dikemas dengan teknik enkripsi.



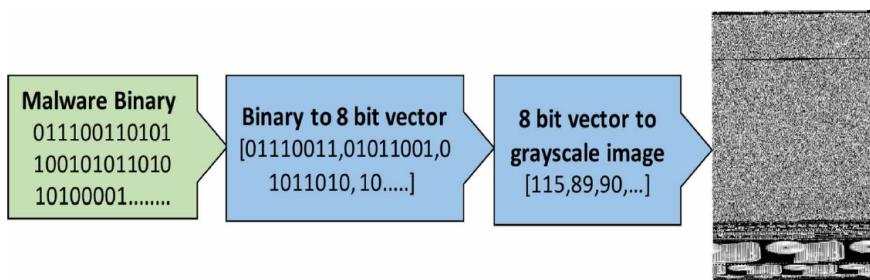
Gambar 2.1 *Malware* yang Dikemas dengan Teknik Enkripsi [11]

Code obfuscation dibagi menjadi 2 jenis meliputi, teknik *polymorphic* dan *metamorphic*. Teknik *polymorphic* merupakan metode enkripsi untuk memodifikasi kode *binary* di dalam *malware*. Teknik *polymorphic* akan terus menghasilkan kode *malware* yang berbeda seiring berubahnya *encryption key*. Sedangkan, teknik *metamorphic* adalah perubahan kode *malware* tanpa penggunaan teknik enkripsi. Teknik *metamorphic* yang dapat dilakukan adalah penyisipan kode yang berulang dan *dead code* yang tidak memengaruhi terhadap fungsi *malware*, serta digunakan untuk meningkatkan jumlah kode yang dianalisis.

Teknik *metamorphic* lainnya adalah subsitusi instruksi untuk mengubah instruksi atau fungsi dengan nama yang berbeda tetapi menghasilkan keluaran yang sama [11].

Visualisasi *malware* adalah proses visualisasi dari *malware binary* menjadi citra [8], proses visualisasi *malware* dapat dilihat pada Gambar 2.2. *Malware binary* merupakan program *executable* yang bisa langsung dijalankan atau biasanya disebut file biner atau *binary file* dengan ekstensi .bin atau .exe, serta bisa direpresentasikan sebagai string yang berisi 1 dan 0 [8]. Pendekatan visualisasi *malware* merupakan salah satu solusi untuk menangani masalah *polymorphic* dan *metamorphic* karena perubahan-perubahan *malware* dapat terlihat secara visual dalam fitur dan tekstur dalam citra *malware*.

Setiap string *binary* akan diambil sebagai suatu vektor *8-bit unsigned integer* dan diubah menjadi *array* 2 dimensi. Vektor yang berisi *8-bit unsigned integer* digunakan untuk menentukan nilai keabuan pada setiap *pixel* membentuk citra *grayscale*. Setiap *pixel* memiliki nilai keabuan dengan rentang nilai antara 0-255, nilai 0 merupakan nilai keabuan terendah yang berwarna hitam, sedangkan nilai 255 merupakan nilai keabuan tertinggi yang berwarna putih [12].



Gambar 2.2 Proses Visualisasi Malware [1]

Visualisasi *malware* dapat menggambarkan keseluruhan struktur kode *malware* sehingga dapat mengidentifikasi perubahan-perubahan kecil secara visual melalui citra *grayscale*. Perbedaan dan kesamaan tekstur atau fitur dalam citra *grayscale* dapat membantu analisis dalam klasifikasi satu *malware family* dengan *family* lainnya [8].

2.1.3 Principal Component Analysis (PCA)

Seleksi dan ekstraksi fitur memiliki peranan penting dalam representasi data sebelum diolah. Sasarannya adalah mendapatkan fitur yang signifikan yang berpotensi untuk meningkatkan akurasi dari model klasifikasi. Karena meroketnya ukuran *dataset* maka diperlukan teknik *dimension reduction* atau reduksi dimensi

data sehingga dapat meningkatkan performa sistem klasifikasi.

Principal Component Analysis merupakan salah satu metode statistik multivariat yang berfungsi untuk ekstraksi fitur dari *dataset* asli menjadi sekumpulan fitur atau variabel ortogonal yang disebut *principal component* [13]. *Principal component* merupakan ringkasan fitur-fitur penting yang mempunyai varians yang tinggi dari data dalam *dataset*, setiap *principal component* mempunyai tingkat varians yang berbeda. Jumlah informasi yang dipertahankan bergantung pada nilai *threshold* yang ditentukan. Semakin tinggi nilai *threshold*, maka semakin tinggi jumlah *principal component* yang dipertahankan, sedangkan semakin rendah nilai *threshold*, maka semakin rendah *principal component* yang dipertahankan atau semakin tinggi *principal component* yang akan diseleksi. Oleh karena itu, diperlukan pemilihan jumlah *principal component* yang tepat untuk untuk mengurangi dimensi atau ukuran data, sambil mempertahankan fitur dengan informasi atau varians yang signifikan.

Berikut Algoritme 2.1 menjelaskan tahapan metode PCA untuk ekstraksi fitur.

Algorithm 2.1 Algoritme Principal Component Analysis

- 1: Normalisasi atau standarisasi data dengan fungsi normalisasi *Z-score* dengan Persamaan 2.1.

Normalisasi dilakukan agar *range* (skala) suatu variabel terdistribusi sama dengan *range* variabel lainnya sehingga tidak ada perbedaan *range* yang besar untuk menghindari terjadinya hasil yang bias. *Range* data yang sudah ditransformasikan bernilai antara 0-1 untuk semua variabel.

- 2: Menghitung matriks kovarians.

Matriks kovarians menunjukkan besar hubungan korelasi antar variabel data. Hasil dari matriks kovarians akan menghasilkan matriks $n \times n$ sejumlah dengan banyak n dimensi data. Berikut contoh matriks untuk data 3 dimensi dengan 3 variabel x , y , dan z .

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

- 3: Menghitung *eigenvector* dan *eigenvalue* dari matriks kovarians dengan Persamaan 2.2 dan 2.3.

Eigenvector merupakan vektor yang menunjukkan arah dan nilai seberapa banyak varians atau informasi yang diproyeksikan dari matriks kovarians, vektor ini disebut juga *principal component*. Sedangkan, *eigenvalue* merupakan koefiesien besar dari *eigenvector*, serta menunjukkan proporsi jumlah informasi varians yang dimiliki suatu *eigenvector*.

BAB 2 LANDASAN TEORI

- 4: Memilih jumlah *eigenvector* atau *principal component* yang akan digunakan. *Eigenvector* terlebih dahulu diurutkan dari nilai *eigenvalue* secara *descending* atau dari nilai *eigenvalue* yang terbesar sampai nilai *eigenvalue* terkecil. Pemilihan komponen dapat dilakukan dengan cara *explained variance percentage*, menghitung persentase berapa banyak informasi varians yang dipertahankan terhadap keseluruhan komponen. Cara pemilihan ini adalah mempertahankan komponen yang memiliki *eigenvalue* yang tinggi, serta komponen kurang signifikan yang memiliki nilai *eigenvalue* yang rendah akan dibuang atau diabaikan. Matriks vektor yang terpilih ini disebut *feature vector*. Pemilihan dapat dilihat seperti contoh berikut.

Dataset awal memiliki 10 dimensi atau fitur. Setelah menghitung matriks kovarians, diperoleh data :

Eigenvalues : [14, 11, 8.5, 7.3, 4.2, 1.5, 0.1, 0.025, 0.005, 0.0029].

Feature vector (5 *eigenvalue* tertinggi) : [14, 11, 8.5, 7.3, 4.2]

Total keseluruhan *eigenvalues* dari array ini adalah = 46.6329.

Informasi varians yang dipertahankan : 45 / 46.6329 = 96.50% dari total data.

- 5: Transformasi data berdasarkan dimensi *feature vector* dengan Persamaan 2.4. Transformasi data dilakukan dengan cara mengalikan *transpose* dari matriks data awal yang sudah dinormalisasi dengan *transpose* dari *feature vector*.
-

Z-score dihitung dengan Persamaan 2.1 untuk melakukan normalisasi data.

$$Z-score = \frac{value - \mu}{\delta} \quad (2.1)$$

Keterangan :

value = nilai data

μ = *mean* atau rata-rata data

δ = standar deviasi data

Persamaan 2.2 untuk menghitung *eigenvector* dari matriks kovarians.

$$\det(A - \lambda I) = 0 \quad (2.2)$$

BAB 2 LANDASAN TEORI

Keterangan :

λ = eigenvalue

A = matriks data

I = matriks identitas

Persamaan 2.3 untuk menghitung *eigenvalue* dari matriks kovarians.

$$\det(A - \lambda I)x = 0 \quad (2.3)$$

Keterangan :

x = *eigenvector*

Transformasi data awal berdasarkan *feature vector* dapat dihitung dengan Persamaan 2.4.

$$\text{Transformed Data} = \text{Feature Vector}^T \times \text{Original Data}^T \quad (2.4)$$

Persentase informasi yang dipertahankan pada Langkah 4 dalam Algoritme 2.1 merupakan nilai *threshold* yang dapat ditentukan saat menggunakan PCA. Semakin tinggi nilai *threshold*, maka semakin besar fitur informasi yang dipertahankan. Untuk nilai *threshold* yang umum dipakai adalah 95%.

2.1.4 Model Warna

Fungsi dari model warna atau disebut juga *color space* adalah untuk memfasilitasi spesifikasi warna dalam cara yang standar. Model warna yang umum digunakan dalam pengolahan citra adalah model warna RGB (*red, green, blue*) digunakan untuk warna dalam layar monitor, model warna CMYK (*cyan, magenta, yellow, black*) untuk warna dalam percetakan, dan model HSI (*hue, saturation, intensity*) merupakan model warna yang digunakan manusia untuk mendeskripsikan dan menginterpretasi suatu warna [14]. Dalam penelitian ini, pembahasan terkait model warna akan berfokus pada model warna RGB dan *grayscale*.

2.1.4.1 RGB

Dalam model RGB, setiap warna pada model warna akan terbentuk dari 3 warna utama yaitu, merah, hijau, dan biru. *Pixel* citra yang direpresentasikan dengan model warna RGB akan berisi 3 komponen, setiap komponen untuk masing-masing warna utama, komponen ini disebut juga *pixel depth* [14]. Setiap komponen atau *channel* dalam model RGB dalam suatu *pixel* akan berisi nilai intensitas dengan skala 0-255 atau 8-bit. *Pixel* di dalam citra RGB akan berbentuk matriks 3 dimensi dengan *depth* 24-bit yang terdiri dari 3 *channel* warna merah, hijau, dan biru yang masing-masing memiliki *pixel depth* 8-bit. Total warna yang dapat ditampilkan dalam suatu citra RGB 24-bit adalah 16.777.216 warna. Gambar 2.3 merupakan salah satu contoh citra dengan model warna RGB.



Gambar 2.3 Contoh Citra RGB [14]

Model warna RGB umumnya digunakan untuk menampilkan warna pada layar televisi, komputer, dan kamera digital, serta dimanfaatkan dalam pengolahan citra dengan tujuan untuk manipulasi warna, seperti *editing* foto dan video.

2.1.4.2 Grayscale

Model *grayscale* merupakan model warna yang mendefinisikan warna dengan satu komponen atau *channel* warna, yaitu dengan tingkat kecerahan. Pengukuran nilai intensitas kecerahan menggunakan rentang antara nilai 0 sampai 255 atau 8-bit [15]. Jumlah warna yang dapat dihasilkan dari *citra grayscale* adalah 256 warna yang terdiri dari warna hitam absolut direpresentasikan dengan nilai intensitas 0 (nilai intensitas terendah), warna putih absolut direpresentasikan dengan nilai intensitas 255 (nilai intensitas tertinggi), serta warna keabuan direpresentasikan dengan nilai intensitas antara 1 sampai 254. Model warna *grayscale* biasanya digunakan dalam citra medis, seperti citra *X-ray* untuk observasi dan diagnosis suatu penyakit, serta pengolahan citra untuk segmentasi dan morfologi. Contoh citra *grayscale* dapat dilihat pada Gambar 2.4.

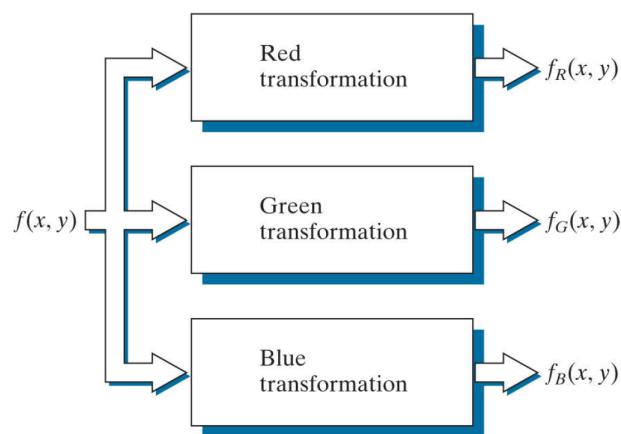


Gambar 2.4 Contoh Citra *Grayscale*

2.1.4.3 *Pseudocolor*

Pseudocolor atau bisa disebut juga *false color* merupakan pengolahan citra untuk pemberian warna pada nilai keabuan di dalam citra *grayscale* berdasarkan kriteria atau aturan tertentu. Karena manusia lebih mudah memahami dan peka terhadap intensitas warna sehingga penerapan *pseudocolor* dapat membantu manusia dalam melakukan visualisasi dan interpretasi pada bagian keabuan pada suatu atau sekumpulan citra [14].

Salah satu pendekatan *pseudocolor* adalah transformasi warna pada nilai intensitas keabuan. Citra *grayscale* biasanya memiliki *channel* warna tunggal yang bersifat akhromatis atau tingkat kecerahan yang menampilkan informasi, seperti temperatur dan tekstur, lalu dilakukan perhitungan fungsi transformasi pada nilai intensitas dari *pixel* masukan untuk mendapatkan masing-masing nilai *channel* merah, hijau, dan biru sehingga menghasilkan citra berwarna dengan model warna RGB [14]. Proses transformasi warna dari nilai intensitas keabuan dapat dilihat pada Gambar 2.5.

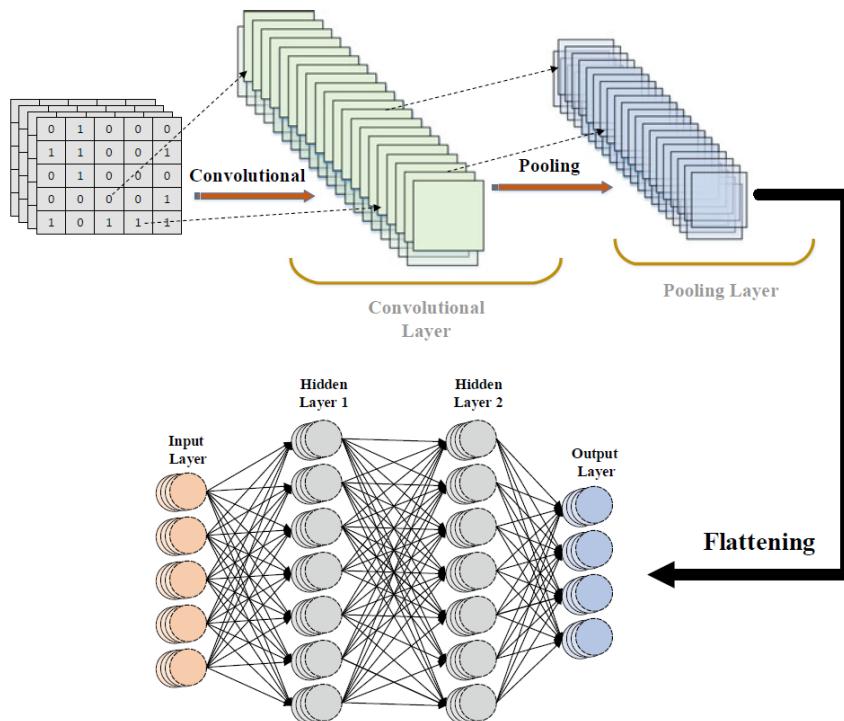


Gambar 2.5 Proses transformasi warna pada nilai intensitas keabuan [14]

2.1.5 Convolutional Neural Network

Convolutional Neural Network (CNN) merupakan pengembangan dari *neural network* untuk memproses data matriks 2 dimensi atau lebih sebagai data

masukan. Konsep dasar CNN terinspirasi dari cara kerja korteks visual pada hewan yang dirancang secara adaptif dapat mengenali fitur spasial dari fitur yang sederhana (*low-level*) sampai kompleks (*high-level*). Arsitektur model CNN secara umum dapat dilihat pada Gambar 2.6, terdiri dari 3 jenis *layer* atau *building block* yaitu, *convolution layer*, *pooling layer*, dan *fully connected layer*. *Convolution* dan *pooling layer* berfungsi untuk melakukan ekstraksi fitur, sedangkan *fully connected layer* digunakan untuk melakukan pemetaan berdasarkan fitur yang diekstrak sebagai keluaran, seperti klasifikasi dan deteksi [16].



Gambar 2.6 Arsitektur Convolutional Neural Network [8]

Proses pelatihan pada CNN terdiri dibagi menjadi 2 proses yaitu, *forward propagation* dan *backpropagation* dengan penjelasan sebagai berikut [17].

1. *Forward propagation* adalah proses perhitungan keluaran dari satu *layer* ke *layer* berikutnya secara berurutan sampai menghasilkan keluaran akhir sebagai hasil prediksi, lalu prediksi *error* dihitung menggunakan *loss function* untuk menunjukkan seberapa besar selisih keluaran prediksi dibandingkan keluaran sebenarnya.
2. *Backpropagation* adalah proses untuk optimisasi arsitektur CNN dengan melakukan modifikasi nilai *parameter* yang diinisialisasi sebelumnya untuk meminimalkan nilai *error* dari keluaran *forward propagation*.

2.1.5.1 Convolution Layer

Convolutional layer merupakan *layer* yang paling penting dalam membangun arsitektur CNN, *layer* ini terdiri dari kumpulan *kernel* atau *filter* yang digunakan untuk melakukan ekstraksi fitur, seperti informasi tepi, warna, dan bentuk pada citra masukan. Fitur hasil ekstraksi dari suatu *kernel* disebut *feature map*. *Kernel* merupakan suatu matriks atau *grid* yang berisi nilai angka, setiap nilai angka dalam *kernel* merupakan *weight* atau bobot. Semua nilai bobot pada suatu *kernel* akan diinisialisasi secara acak. Pada saat proses pelatihan, nilai bobot pada *kernel* akan terus diperbarui atau di-*tuning* sehingga dapat mengenali fitur penting pada citra masukan [18].

Pendekatan yang digunakan untuk inisialisasi bobot adalah inisialisasi acak dengan distribusi normal atau disebut juga distribusi Gaussian. Nilai bobot awal pada *kernel* dari *convolution layer* dan *fully connected layer* akan diinisialisasi secara acak berdasarkan distribusi normal dengan menentukan nilai rata-rata dan standar deviasi. Pendekatan ini digunakan untuk mencegah terjadinya *vanishing gradient*, kondisi di mana nilai bobot yang diperbarui menjadi bernilai 0 disebabkan karena inisialisasi nilai bobot terlalu besar atau kecil [19]. Persamaan distribusi normal yang dipakai untuk inisialisasi acak dapat dilihat pada Persamaan

$$k = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad (2.5)$$

Keterangan :

- k = nilai *kernel*
 - x = variabel acak normal
 - σ = standar deviasi
 - μ = rata-rata atau *mean*
 - π = nilai *pi* (3.14159265359...)
 - e = bilangan *Euler* (2.718281828...)
-

Pada *convolutional layer* dilakukan operasi konvolusi untuk ekstraksi fitur. Operasi konvolusi merupakan proses pergerakan *kernel* secara horizontal dan vertikal pada setiap *pixel* di dalam suatu citra, lalu dilakukan perkalian elemen (dot) antara nilai pada citra dengan nilai bobot pada *kernel*. Semua hasil perkalian tersebut dijumlahkan untuk menghasilkan suatu nilai skalar sebagai *feature map*.

dari suatu *pixel*. Operasi konvolusi akan terus berjalan untuk setiap *pixel* di sepanjang citra. Perhitungan operasi konvolusi dilakukan dengan Persamaan 2.6 berikut ini [8].

$$c_k = \sum_{k=i+j} x_i y_j = \sum_i x_i y_{k-i} \quad (2.6)$$

Keterangan :

c_k = konvolusi elemen ke- k

k = elemen ke- k

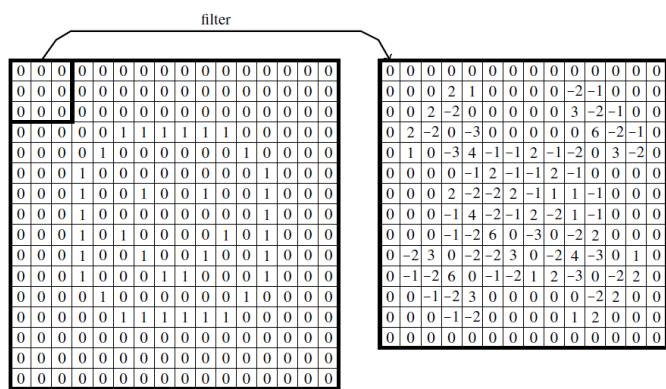
x_i = sekvens konvolusi x iterasi ke- i

y_j = sekuens konvolusi y iterasi ke- j

i = iterasi ke-*i*

j = iterasi ke-*j*

Sebagai contoh, Gambar 2.7 menunjukkan suatu *convolution layer* menggunakan ukuran *kernel* 3×3 . Selain itu, untuk citra RGB diproses dengan *kernel* berbentuk matriks 3 dimensi untuk melakukan operasi konvolusi untuk masing-masing *channel red, green, dan blue*. Karena bobot dari setiap *kernel* diinisialisasi secara acak maka *feature map* yang dihasilkan akan berbeda-beda di setiap *channel* warnanya [8].



Gambar 2.7 Convolutional Layer dengan Filter 3x3 [8]

Convolutional layer memiliki beberapa *hyperparameter* meliputi ukuran *kernel*, jumlah *filter*, *stride*, dan *padding* dengan penjelasan sebagai berikut [18].

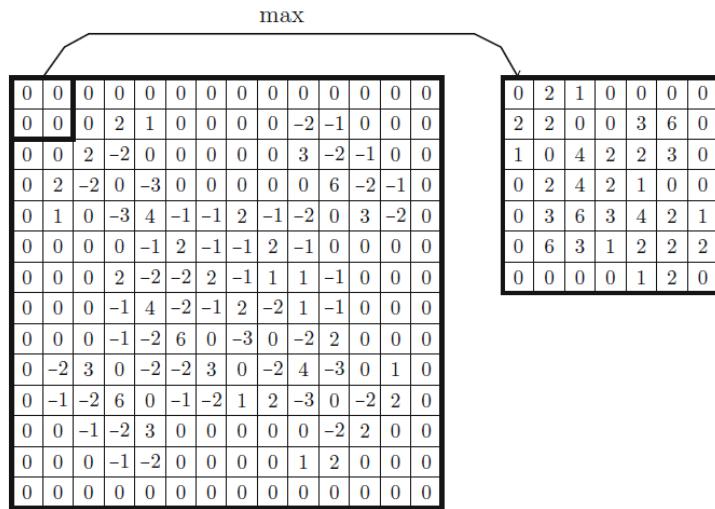
1. Ukuran *kernel* (*kernel size*) : menentukan ukuran matriks dari *kernel* yang digunakan.

2. Jumlah filter (*filter size*) : menentukan jumlah *filter* yang akan melakukan operasi konvolusi sehingga akan menentukan jumlah keluaran *feature map* dalam suatu *convolution layer*.
3. *Stride* : menentukan besar perpindahan suatu *kernel* di sepanjang citra masukan. *Stride* dengan nilai lebih dari 1 biasanya digunakan untuk melakukan *downsampling* pada *feature map*
4. *Padding* : penambahan *pixel* di sekitar tepi atau ujung dalam citra masukan. Teknik *padding* yang umum digunakan adalah *zero padding*, *padding* untuk menambahkan *pixel* bernilai 0 di sekitar tepi citra, tujuannya untuk menghindari penyusutan pada keluaran *feature map*.

Feature map yang dihasilkan dari *convolution layer* akan dimasukkan ke dalam suatu fungsi yang disebut *activation function* atau fungsi aktivasi, fungsi yang digunakan untuk membantu model klasifikasi untuk mempelajari pola yang kompleks pada data. Fungsi aktivasi yang digunakan pada *convolution layer* adalah *Rectified Linear Unit* (ReLU) yang dijabarkan pada Bab 2.1.6.

2.1.5.2 *Pooling Layer*

Pooling layer berfungsi untuk *downsampling* dengan mengurangi dimensi *feature map* yang dihasilkan dari *convolution layer*. Tujuan dari *pooling layer* adalah mereduksi fitur dalam pelatihan dengan mempertahankan fitur yang paling penting atau signifikan dari *feature map*. *Max-pooling* merupakan teknik yang paling umum digunakan dalam *pooling layer*, teknik *pooling* yang menyimpan nilai *pixel* maksimum dalam suatu wilayah atau bagian dalam citra masukan sehingga akan menghasilkan ukuran citra yang lebih kecil, sebagai contoh pada Gambar 2.8 merupakan *pooling layer* menggunakan *max-pooling* dengan *filter* 2×2 .

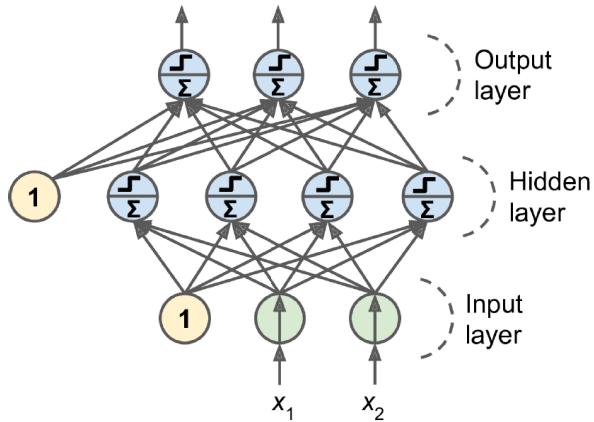


Gambar 2.8 Max Pooling 2×2 [8]

Batasan keluaran *feature map* dari *convolution layer* adalah fitur yang dikenali sensitif terhadap lokasi dalam citra masukan sehingga jika terjadi perubahan kecil pada citra masukan, seperti adanya pergeseran, rotasi, dan pemotongan akan menghasilkan *feature map* yang berbeda. Penerapan *pooling layer* digunakan mengatasi masalah tersebut dengan merangkum fitur-fitur penting dalam *feature map* sehingga fitur yang dikenali bersifat kokoh (*robust*) karena tidak terpengaruh terhadap lokasi ketetanggaan dengan fitur lain [18].

2.1.5.3 Fully Connected Layer

Masukan dari *fully connected layer* adalah keluaran *feature map* dari *convolution* atau *pooling layer* terakhir, lalu dilakukan *flatten*, proses untuk transformasi *feature map* menjadi *array* atau matriks 1 dimensi. Keluaran dari *fully connected layer* adalah probabilitas dari setiap kelas target atau label sebagai hasil klasifikasi. *Fully connected layer* terdiri dari 3 jenis *layer*, yaitu *input layer*, *hidden layer*, dan *output layer* seperti yang bisa dilihat pada Gambar 2.9 berikut ini.



Gambar 2.9 Fully Connected Layer [20]

Setiap *layer* memuat *perceptron* atau *neuron* yang digunakan untuk melakukan perhitungan komputasi di dalam *neural network*, tujuannya adalah menganalisis pola tersembunyi di dalam data masukan. Setiap *neuron* dalam *fully connected layer* terkoneksi dengan setiap *neuron* pada *layer* berikutnya. Setiap koneksi akan menghitung bobot dan bias dari data masukan dalam suatu *neuron* menggunakan Persamaan 2.7 sebagai berikut.

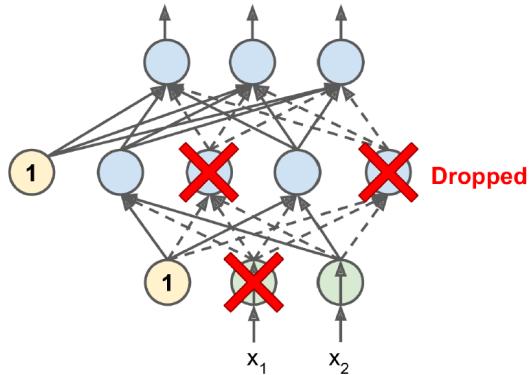
$$f(x) = \sum_{i=0}^{n-1} w_i x_i + b \quad (2.7)$$

Keterangan :

- x_i = data vektor masukan iterasi ke-*i*
 - w_i = nilai bobot iterasi ke-*i*
 - b = nilai bias
 - i = iterasi ke-*i*
-

Hasil perhitungan dari setiap *neuron* akan digunakan untuk data masukan pada *neuron* di dalam *layer* berikutnya. *Hidden Layer* di dalam *fully connected layer* biasanya akan diikuti oleh *dropout layer*, tujuannya untuk melakukan regularisasi pada model supaya mencegah terjadinya *overfitting* [1]. Sedangkan, pada *output layer* menerapkan suatu fungsi aktivasi *softmax* untuk menghasilkan keluaran akhir prediksi kelas klasifikasi.

2.1.5.4 Dropout Layer



Gambar 2.10 Dropout Layer [20]

Dropout layer digunakan untuk melakukan regularisasi sehingga dapat meningkatkan generalisasi pada model dan dapat mencegah *overfitting* yang seringkali terjadi pada model *deep learning* [1]. *Overfitting* terjadi karena model menghasilkan akurasi yang terlalu tinggi pada data belajar sehingga model kesulitan mengenali data baru yang belum dianalisis sebelumnya menyebabkan misklasifikasi dan nilai akurasi yang rendah pada saat pengujian.

Dropout layer bekerja dengan cara menonaktifkan atau mematikan sekumpulan *neuron* pada *fully connected layer* secara acak untuk mencegah terjadinya *overfitting*. Sebelum *neuron* melakukan perhitungan bobot dan bias, variabel acak dengan rentang nilai dari 0 sampai 1 digunakan untuk menentukan apakah *neuron* akan dinonaktifkan atau tidak, perhitungan pada *dropout layer* dapat dilihat pada Persamaan 2.8 sebagai berikut [21].

$$O_{(i)} = \begin{cases} 0 & \text{for } rn > p \\ O_{(i)} \times \frac{1}{p} & \text{for } rn \leq p \end{cases} \quad (2.8)$$

Keterangan :

- $O_{(i)}$ = matriks keluaran posisi ke-*i*
- p = nilai *hyperparameter probability* *p*
- rn = nilai acak dari 0 sampai 1

Pada Persamaan 2.8, jika nilai acak lebih besar dari nilai *probability* *p*, maka akan menghasilkan nilai 0, sedangkan jika nilai acak lebih kecil dari nilai *probability* *p*, maka nilai masukan akan dikalikan dengan $\frac{1}{p}$. Keluaran yang bernilai 0 merupakan *neuron* atau masukan yang akan dinonaktifkan.

2.1.6 Fungsi Aktivasi

Dalam model *deep learning*, data berbobot akan dimasukkan ke dalam suatu fungsi aktivasi. Fungsi aktivasi adalah suatu fungsi untuk melakukan perhitungan transformasi terhadap data berbobot. Fungsi aktivasi terbagi menjadi 2 jenis, yaitu fungsi linear dan fungsi non-linear. Fungsi linear adalah fungsi yang tidak dibatasi dengan rentang nilai apapun. Sedangkan fungsi non-linear, fungsi yang paling banyak digunakan dalam model *deep learning*, fungsi yang memungkinkan suatu model dapat mempelajari pola yang lebih kompleks dalam data [22]. Fungsi aktivasi yang khusus digunakan dalam penelitian ini terdiri dari *Rectified Linear Unit* dan *Softmax*.

2.1.6.1 Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU) merupakan fungsi aktivasi untuk meningkatkan non-linear pada *convolution layer* [22]. Fungsi ReLU dapat dihitung melalui Persamaan 2.9 sebagai berikut.

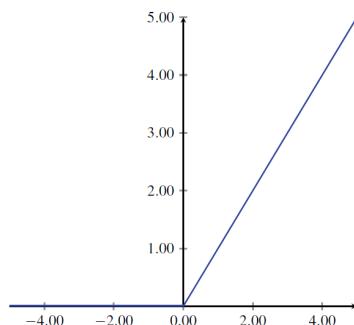
$$f(x) = \max(0, x) \quad (2.9)$$

Keterangan :

$f(x)$ = keluaran fungsi aktivasi ReLu

x = data masukan x

Berdasarkan Persamaan 2.9, fungsi ReLu akan mengembalikan semua bilangan negatif menjadi 0. Jika nilai x adalah not atau bilangan negatif, maka fungsi akan menghasilkan nilai 0. Sebaliknya, jika nilai x bernilai positif maka fungsi ReLu akan menghasilkan nilai x itu sendiri. Dari Persamaan 2.9, dapat disimpulkan bahwa fungsi ReLu memiliki kurva pada Gambar 2.11.



Gambar 2.11 Kurva Fungsi ReLu

2.1.6.2 *Softmax*

Softmax digunakan pada *output layer* atau layer terakhir pada model *neural network* untuk menghitung keluaran akhir suatu klasifikasi. Fungsi *softmax* digunakan untuk masalah klasifikasi *multi-class* di mana kelas klasifikasi yang diprediksi berjumlah lebih dari 2 kelas atau label [22]. Fungsi *softmax* dinyatakan secara matematis dengan Persamaan 2.10 sebagai berikut.

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (2.10)$$

Keterangan :

- | | |
|-------|----------------------------|
| x_i | = data atau nilai x ke-i |
| x_j | = data atau nilai x ke-j |
| i | = iterasi ke-i |
| j | = iterasi ke-j |
-

Keluaran fungsi *softmax* berupa nilai probabilitas pada setiap kelas atau label dengan rentang nilai 0 sampai 1, jika semua nilai probabilitas pada setiap kelas dijumlahkan akan mendapat nilai probabilitas 1 atau 100%. Kelas yang memiliki nilai probabilitas tertinggi yang akan diambil sebagai hasil klasifikasi atau kelas target [8].

2.1.7 *Loss Function*

Loss function merupakan suatu fungsi pengukuran *error* untuk model *neural network* dalam melakukan klasifikasi atau prediksi. *Loss function* ini menghitung seberapa besar perbedaan nilai keluaran yang dihasilkan suatu model dibandingkan nilai target yang sebenarnya sehingga Pendekatan ini digunakan untuk mengukur performa model dalam proses pelatihan, serta digunakan untuk pembaharuan bobot dan bias agar nilai *error* dapat diminimalkan pada iterasi pelatihan berikutnya.

Dalam penelitian ini, fungsi *softmax* digunakan untuk menghitung keluaran kelas klasifikasi dalam model CNN yang dibangun, keluaran berupa nilai probabilitas sebanyak N kelas target klasifikasi yang berarti akan menghasilkan 25 nilai probabilitas untuk setiap kelas *malware family*. *Loss function* yang digunakan untuk menghitung nilai *loss* atau *error* hasil prediksi kelas *malware family* adalah *categorical cross-entropy* [23].

Categorical cross-entropy adalah suatu *loss function* yang biasa digunakan untuk masalah klasifikasi *multiclass*, klasifikasi dengan kelas target berjumlah 2 kelas atau lebih. *Categorical cross-entropy* menggunakan perhitungan logaritma terhadap hasil keluaran fungsi *softmax*, simbol ϕ merujuk pada hasil keluaran fungsi aktivasi *softmax*. Perhitungan *categorical cross-entropy* ditunjukkan pada Persamaan 2.11 berikut ini [23].

$$Loss = -\log(\phi_i) \quad (2.11)$$

Keterangan :

$Loss$ = nilai *loss*

ϕ_i = hasil fungsi *softmax* berupa nilai probabilitas kelas ke-*i*

2.1.8 Regularisasi

Regularisasi adalah teknik untuk menyesuaikan model pembelajaran mesin yang bertujuan untuk mengurangi nilai *loss* atau *error* dan mencegah *overfitting* pada model. Metode regularisasi memberikan nilai penalti pada model untuk mencegah berkembangnya kompleksitas model sehingga dapat menghindari terjadinya *overfitting* pada model, serta membantu model melakukan prediksi yang efisien dan tidak bias. Terdapat banyak metode regularisasi yang dapat dipakai, tetapi pada penelitian ini metode regularisasi yang digunakan adalah regularisasi L2.

2.1.8.1 Regularisasi L2

Regularisasi L2, atau disebut juga Regresi Ridge, digunakan untuk menghitung nilai atau penalti yang akan ditambahkan pada nilai *loss* untuk melakukan penalti atau pengurangan pada bobot dan bias yang bernilai besar. Metode ini bertujuan akan menjaga nilai bobot dan bias sekecil mungkin, tetapi tidak sampai bernilai 0. Nilai bobot yang tinggi mengindikasikan bahwa suatu *neuron* berusaha mempelajari dan mengingat terlalu banyak elemen data. Tetapi, pendekatan yang lebih baik seharusnya model mempunyai lebih banyak *neuron* yang berkontribusi terhadap keluaran model dibandingkan memiliki sedikit *neuron* [24].

Penalti dari regularisasi L2 dihitung berdasarkan total dari nilai bobot dan bias yang dikuadratkan. Dengan pendekatan non-linear cenderung melakukan penalti yang lebih besar terhadap nilai bobot dan bias yang besar dibandingkan

nilai yang kecil. Regularisasi L2 lebih umum dipakai karena tidak memengaruhi *parameter* yang bernilai kecil, serta menjaga nilai bobot tidak bertambah besar karena peneliti yang diberikan relatif besar terhadap nilai bobot yang besar. Untuk konteks regularisasi ini hanya ditambahkan pada *loss function* selama proses pelatihan model [24].

Regularisasi L2 akan memberikan penalti pada bobot dan bias dengan melakukan perhitungan pada Persamaan 2.12 dan 2.13 [24]. Dalam penelitian ini, nilai konstanta regularisasi λ yang digunakan adalah sebesar 0.001.

$$L_{2w} = \lambda \sum_m w_m^2 \quad (2.12)$$

Keterangan :

w = nilai bobot atau *weight*

L_{2w} = hasil regularisasi pada bobot w

λ = konstanta regularisasi

$\sum_m w_m^2$ = penjumlahan kuadrat bobot w dari sekumpulan *neuronm*

$$L_{2b} = \lambda \sum_n b_n^2 \quad (2.13)$$

Keterangan :

b = nilai bias

L_{2b} = hasil regularisasi pada bias b

λ = konstanta regularisasi

$\sum_n b_n^2$ = penjumlahan kuadrat bias b dari sekumpulan *neuronn*

2.1.9 Confusion Matrix

Confusion matrix merupakan salah satu teknik pengukuran performa untuk suatu algoritme klasifikasi, dan bisa diterapkan baik pada kasus klasifikasi biner dan *multiclass*. Untuk memudahkan meringkas hasil kinerja suatu algoritme atau model umumnya *confusion matrix* digambarkan dalam bentuk matriks atau tabel 2×2 .

Confusion matrix menunjukkan jumlah dari nilai hasil prediksi dan nilai sebenarnya. *Confusion matrix* dibagi menjadi 4 nilai, yaitu :

Tabel 2.1 Confusion Matrix

		Predicted	
		Yes	No
Actual	Yes	True Positive	False Negative
	No	False Positive	True Negative

1. *True Positive* (TP) adalah kondisi ketika suatu kelas diprediksi merupakan kelas positif dan terbukti benar kelas tersebut merupakan kelas positif.
2. *True Negative* (TN) adalah kondisi ketika suatu kelas diprediksi merupakan kelas negatif dan terbukti benar kelas tersebut merupakan kelas negatif.
3. *False Positive* (FP) adalah kondisi ketika suatu kelas diprediksi merupakan kelas positif, namun ternyata kelas tersebut merupakan kelas negatif.
4. *False Negative* (FN) adalah kondisi ketika suatu kelas diprediksi merupakan kelas negatif, namun ternyata kelas tersebut merupakan kelas positif.

Untuk menghitung nilai prediksi yang benar atau akurasi digunakan Persamaan 2.14.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.14)$$

Untuk menghitung nilai TP rate atau *precision* dapat menggunakan Persamaan 2.15.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.15)$$

Perhitungan nilai sensitivitas atau *recall* dapat menggunakan Persamaan 2.16.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.16)$$

Nilai *F-measure* atau *F1-score* menunjukkan perbandingan rata-rata antara nilai *precision* dan *recall*, serta dihitung dengan Persamaan 2.17

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.17)$$

2.2 Pustaka Python

Berikut adalah penjelasan dari pustaka atau *library* Python yang digunakan di dalam penelitian.

2.2.1 Numpy

Numpy merupakan pustaka untuk bahasa pemrograman Python untuk melakukan perhitungan operasi matematika dan statistik khususnya pada objek matriks atau *array* multi-dimensi. Tetapi, pada penelitian ini pustaka Numpy digunakan untuk pemrosesan citra dalam bentuk *array* 2 dimensi, serta pemrosesan pada hasil klasifikasi. Tabel 2.2 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka Numpy.

Tabel 2.2 Daftar *Method* yang Digunakan dari Pustaka Numpy

No.	Method	Masukan	Luaran	Keterangan
1.	array	array: array_like	array[]	Mengubah data <i>array</i> menjadi data objek bertipe ndarray yang merupakan tipe data <i>array</i> dari pustaka Numpy.
2.	argmax	array: array_like, axis: int	int	Mengembalikan indeks dari nilai maksimum di dalam suatu <i>array</i> .
3.	unique	array: array_like	unique[]	Mencari nilai-nilai unik di dalam suatu data <i>array</i> .
4.	expand_dims	array: array_like, axis: int	expanded[]	Meningkatkan dimensi dalam suatu <i>array</i> .

2.2.2 Pandas

Pandas merupakan suatu pustaka Python yang berfungsi untuk melakukan manipulasi dan analisis pada data tabel. Tabel 2.3 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka Pandas.

Tabel 2.3 Daftar *Method* yang Digunakan dari Pustaka Pandas

No.	Method	Masukan	Luaran	Keterangan
1.	DataFrame	array: array_like	DataFrame	Mengubah data <i>array</i> menjadi data tabel dengan tipe DataFrame.
2.	DataFrame.plot	figsize: tuple(width, height)	graph image	Visualisi data berupa plot dari DataFrame.

2.2.3 OpenCV

OpenCV adalah pustaka Python yang digunakan untuk keperluan pengolahan citra dan *computer vision*. OpenCV terintegrasi dengan pustaka Numpy, citra yang diolah pada OpenCV akan direpresentasikan menggunakan NumPy *array*. Tetapi, pada penelitian ini pustaka OpenCV digunakan untuk pemrosesan pada data citra dan penerapan *pseudocolor*. Tabel 2.4 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka OpenCV.

Tabel 2.4 Daftar *Method* yang Digunakan dari Pustaka OpenCV

No.	Method	Masukan	Luaran	Keterangan
1.	imread	path: str	image	Memuat <i>file</i> citra dari <i>path</i> direktori yang sudah ditentukan.
2.	cvtColor	image: image, colorspace: cv.cvtColor	converted color space image	Melakukan konversi <i>color space</i> pada citra.
2.	applyColorMap	src: image, colormap: cv.cvtColor	RGB image	Penerapan <i>pseudocolor</i> dengan <i>color-map</i> pada suatu citra untuk menghasilkan citra RGB yang baru.

2.2.4 Matplotlib

Matplotlib adalah pustaka yang berfungsi untuk visualisasi data dan *plotting* untuk bahasa pemrograman Python. Tabel 2.5 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka Matplotlib.

Tabel 2.5 Daftar *Method* yang Digunakan dari Pustaka Matplotlib

No.	Method	Masukan	Luaran	Keterangan
1.	imshow	image: image atau array_like	image	Menampilkan citra yang diinginkan.
2.	show	-	figure	Menampilkan grafik yang sudah didefinisikan sebelumnya.
3.	title	label: str	figure title	Mengatur judul pada grafik atau plot.
4.	bar	data: array_like, height: array_like, width: array_like	bar graph	Membuat visualisasi data dari data masukan berupa grafik batang.
5.	figure	figsize: tuple(width, height)	figure	Membuat figure baru atau mengaktifkan figure yang sudah ada.
6.	axis	option: str atau boolean	graph axis	Mengatur sumbu koordinat pada suatu grafik.
7.	xlabel	label: str	x label	Mengatur label keterangan pada sumbu x.
8.	ylabel	label: str	y label	Mengatur label keterangan pada sumbu y.

2.2.5 Scikit-learn

Scikit-learn adalah pustaka Python yang menyediakan berbagai algoritme machine learning, seperti algoritme regresi, klasifikasi, dan *clustering*. Tetapi, pada penelitian ini pustaka Scikit-learn digunakan untuk proses ekstraksi fitur dengan PCA dan *data split* untuk membagi *dataset* menjadi data belajar dan data uji. Tabel 2.6 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka Scikit-learn.

Tabel 2.6 Daftar *Method* yang Digunakan dari Pustaka Scikit-learn

No.	Method	Masukan	Luaran	Keterangan
1.	PCA	n_component: int atau float	object	Membangun model PCA untuk reduksi dimensi data berdasarkan jumlah komponen yang ditentukan.
1.	fit_transform	input: array_like	output[]	Melakukan proses ekstraksi fitur dengan PCA untuk reduksi dimensi pada data masukan.
2.	train_test_split	data: array_like, test_size: float atau int, train_size: float atau int, random_state: int	data train data test target train target test	Melakukan pembagian pada <i>dataset</i> menjadi data belajar dan data uji secara acak.
3.	class_weight. compute_class_weight	class_weight: str, classes: array_like, target_data: array_like	class_weight[]	Melakukan perhitungan bobot dari setiap kelas target atau label dari <i>dataset</i> .
4.	metrics. confusion_matrix	target_true: array_like, target_pred: array_like	output[]	Menghitung <i>confusion matrix</i> untuk mengevaluasi akurasi klasifikasi.
5.	metrics. classification_report	target_true: array_like, target_pred: array_like	report_dict	Rangkuman bertipe data <i>dictionary</i> yang berisi nilai <i>precision</i> , <i>recall</i> , dan <i>F1 score</i> untuk setiap kelas klasifikasi.

2.2.6 Keras

Keras merupakan pustaka Python yang bersifat *open-source* yang berfungsi untuk membangun *artificial neural network*. Keras merupakan abstraksi

tingkat tinggi dari pustaka Tensorflow untuk memudahkan pembuatan model *neural network*. Tetapi, pada penelitian ini pustaka Keras digunakan untuk proses *resizing* citra *malware* yang terdapat dalam *dataset*. Tabel 2.7 berikut ini merupakan penjelasan mengenai *method* yang digunakan dari pustaka Keras.

Tabel 2.7 Daftar *Method* yang Digunakan dari Pustaka Keras

No.	<i>Method</i>	Masukan	Luaran	Keterangan
1.	ImageDataGenerator().flow_from_directory	directory: str, target_size: tuple(height, width), batch_size: int, color_mode: str	batch image	Melakukan <i>resizing</i> citra pada <i>dataset</i> atau <i>data augmentation</i> .
2.	utils.img_to_array	image: array_like	array[]	Melakukan konversi dari data citra menjadi NumPy <i>array</i> .

2.3 Tinjauan Studi

Pada bagian ini merupakan penjelasan mengenai jurnal penelitian yang dipakai sebagai tinjauan terkait penelitian yang dilakukan.

2.3.1 *State of The Art*

Terdapat beberapa metode di dalam ruang lingkup yang mirip dengan penelitian ini khususnya yang terkait klasifikasi *malware* menggunakan citra *grayscale*. Tabel 2.8 *State of The Art* akan menjelaskan perbedaan-perbedaan metode yang telah dipelajari dari jurnal penelitian.

BAB 2 LANDASAN TEORI

Tabel 2.8 State of The Art

No.	Jurnal	Rumusan Masalah	Metode	Hasil
1	M. Kalash, M. Rochan, N. Mohammed, N. Bruce, Y. Wang, and F. Iqbal, "A Deep Learning Framework for Malware Classification," <i>International Journal of Digital Crime and Forensics (IJDCF)</i> , vol. 12, no.1, pp. 90-108, March 2020. [1]	1. Apakah metode Deep Convolutional Neural Network (CNN) dapat melakukan klasifikasi <i>malware</i> berdasarkan citra <i>grayscale</i> dengan akurat ? 2. Bagaimana performa pada metode Deep CNN dalam klasifikasi citra <i>malware</i> yang tidak utuh ?	Deep Convolutional Neural Network	Metode Deep CNN bisa melakukan klasifikasi secara efektif menghasilkan akurasi tertinggi sebesar 99.08% pada citra <i>grayscale</i> yang utuh, sedangkan pada citra <i>grayscale</i> yang dipotong, performa metode mengalami sedikit penurunan dengan akurasi sebesar 86.36%.

BAB 2 LANDASAN TEORI

Tabel 2.8 State of The Art (Lanjutan)

No.	Jurnal	Rumusan Masalah	Metode	Hasil
2	J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware Visualization for Fine-Grained Classification," <i>IEEE Access</i> , vol. 6, pp. 14510-14523, 2018. [4]	<p>1. Apa teknik ekstraksi fitur yang dapat meningkatkan performa akurasi metode <i>machine learning</i> untuk klasifikasi <i>malware</i> ?</p> <p>2. Model <i>machine learning</i> apa yang paling efektif dalam klasifikasi <i>malware</i> ?</p>	<p>1. Random Forest 2. K-Nearest Neighbor 3. Support Vector Machine 4. Fitur Global dan Fitur Lokal</p>	Melakukan teknik ekstraksi fitur berupa fitur global untuk mengekstrak fitur dari citra dan fitur lokal yang merupakan fitur vektor yang diekstrak dari bagian kode pada <i>file malware</i> , kedua metode tersebut terbukti dapat meningkatkan performa akurasi model <i>machine learning</i> . Model <i>machine learning</i> yang paling efektif dalam mengklasifikasikan <i>malware</i> adalah Random Forest dengan akurasi 97.47% yang telah dilakukan kombinasi ekstraksi fitur global dan lokal.

BAB 2 LANDASAN TEORI

Tabel 2.8 State of The Art (Lanjutan)

No.	Jurnal	Rumusan Masalah	Metode	Hasil
3	M. Jain, W. Andreopoulos, and M. Stamp, "Convolutional neural networks and extreme learning machines for malware classification," <i>Journal of Computer Virology and Hacking Techniques</i> , vol. 16, pp. 229-244, 2020. [5]	Bagaimana perbandingan performa akurasi dan waktu pelatihan pada metode Convolution Neural Network dan Extreme Learning Machine (ELM) untuk klasifikasi citra grayscale malware ?	1. Convolutional Neural Network 2. Extreme Learning Machine	Akurasi tertinggi untuk klasifikasi <i>malware</i> menggunakan metode CNN sebesar 95.67%, akurasi tersebut lebih tinggi dibandingkan akurasi metode ELM sebesar 93.9%, sedangkan waktu pelatihan ELM membutuhkan waktu 74.12 detik, lebih cepat dibandingkan waktu pelatihan CNN yang membutuhkan waktu 334.50 detik. metode CNN cenderung memiliki nilai akurasi yang lebih tinggi, sedangkan waktu pelatihan lebih cepat pada metode ELM.
4	A. Darwaish and F. Naït-Abdesselam, "RGB-based Android Malware Detection and Classification Using Convolutional Neural Network," <i>GLOBECOM 2020 - 2020 IEEE Global Communications Conference</i> , pp. 1-6, 2020. [7]	Berapa akurasi metode Convolutional Neural Network untuk klasifikasi <i>malware</i> pada <i>platform Android</i> berdasarkan citra RGB hasil konversi dari <i>file APK</i> ?	Convolutional Neural Network	Metode CNN dengan konfigurasi 2 <i>convolution layer</i> menghasilkan nilai akurasi tertinggi sebesar 98.77% dengan rata-rata <i>false negative</i> 1.72%, serta rata-rata <i>false positive</i> 0.72% pada klasifikasi <i>malware</i> menggunakan citra RGB.

2.3.2 Pembahasan Penelitian Terkait

Terdapat beberapa metode yang dapat digunakan untuk klasifikasi *malware* menggunakan visualisasi citra *grayscale*. Penelitian *A Deep Learning Framework for Malware Classification* [1] membangun model *Deep Convolutional Neural Network* yang digunakan untuk klasifikasi citra *grayscale malware*. Penelitian ini menggunakan 2 *dataset* sebagai *benchmark*, yaitu *dataset* Malware Image (Malimg) dan Microsoft. Tujuannya adalah untuk menguji generalisasi model apakah model dapat mengklasifikasikan citra *malware* secara generik tanpa bergantung pada *dataset* tertentu saja. Penelitian ini juga melakukan eksperimen klasifikasi dengan seluruh bagian citra dan citra yang dipotong untuk melihat pengaruhnya terhadap model. Model CNN dapat melakukan klasifikasi citra *malware* secara efektif dengan akurasi 98.52%, tetapi mengalami penurunan performa klasifikasi yang signifikan pada citra *malware* yang dipotong dengan akurasi 72.80%. Lalu, model CNN dilakukan modifikasi untuk meningkatkan performa akurasi sehingga nilai akurasi terbaik yang dihasilkan model CNN adalah 99.08% pada citra yang utuh dan 86.36% pada citra yang dipotong. Modifikasi yang dilakukan pada model CNN tidak terlalu dijelaskan di dalam penelitian.

Pada Penelitian *Malware Visualization for Fine-Grained Classification* [4] mengembangkan 3 model klasifikasi, yaitu Random Forest, K-Nearest Neighbor (KNN), dan Support Vector Machine (SVM). Pada penelitian ini juga mengembangkan metode visualisasi *malware* yang baru, dengan menggunakan nilai *entropy*, *byte value*, dan *relative size* untuk mendefinisikan nilai *channel red*, *green*, dan *blue* pada setiap *pixel* sehingga mengonversikan *file malware* menjadi citra berwarna RGB. Citra RGB dinilai dapat memudahkan dalam membedakan *malware family* dibandingkan citra *grayscale*. Lalu, dilakukan ekstraksi fitur dengan mengkombinasikan fitur global dan fitur lokal, fitur global digunakan untuk mendapatkan tekstur dan warna citra dengan algoritme *Gray Level Co-occurrence Matrix* (GLCM), sedangkan fitur lokal untuk mendapatkan fitur vektor *malware* dengan melakukan *hashing* pada kode dalam *file malware*. Akurasi tertinggi dihasilkan dengan menerapkan kombinasi ekstraksi fitur global dan fitur lokal, pada model Random Forest memiliki akurasi sebesar 97.47%, KNN memiliki akurasi sebesar 96.23%, dan SVM memiliki akurasi sebesar 95.23%. Namun, kekurangan dari penelitian ini adalah proses ekstraksi fitur membutuhkan waktu yang lama.

Pada Penelitian *Convolutional Neural Networks and Extreme Learning Machines for Malware Classification* [5], model Convolutional Neural Network

(CNN) dan Extreme Learning Machine (ELM) dikembangkan pada sistem klasifikasi *malware* berdasarkan citra *grayscale*, tujuannya untuk membandingkan performa dan waktu pelatihan antara kedua model tersebut. Metode CNN terbukti mempunyai performa yang tinggi pada kasus klasifikasi citra, sedangkan metode ELM dapat melakukan klasifikasi dengan yang baik juga dengan komputasi yang lebih rendah sehingga waktu pelatihan dapat lebih cepat. Selain itu, penelitian ini melakukan pengujian dan analisis pengaruh dimensi dari data masukan, yaitu citra 2 dimensi biasa dan vektor 1 dimensi terhadap performa akurasi dan waktu pelatihan model. Vektor 1 dimensi dihasilkan dari proses transformasi citra menjadi vektor, dengan menggunakan vektor 1 dimensi dinilai dapat mengurangi jumlah fitur pada saat pelatihan sehingga dapat mengurangi komputasi dan mempercepat waktu pelatihan. Perbandingan akurasi dengan data masukan citra 2 dimensi, yaitu pada model CNN dengan 2 *convolutional layer* memiliki akurasi sebesar 95.67% dengan waktu pelatihan 334.50 detik, sedangkan model ELM memiliki akurasi sebesar 93.9% dengan waktu pelatihan 74.12 detik. Untuk klasifikasi dengan data masukan vektor 1 dimensi hasil akurasi model CNN 96.25% waktu pelatihan 250 detik dan ELM 96.3% waktu pelatihan 153.15 detik. Dapat disimpulkan bahwa model ELM memiliki akurasi yang mendekati model CNN dengan perbedaan nilai akurasi tidak terlalu jauh, serta keuntungan model ELM adalah waktu pelatihan yang lebih cepat dibandingkan model CNN. Data masukan dengan dimensi yang rendah juga turut memengaruhi waktu pelatihan menjadi lebih cepat, tetapi menghasilkan akurasi yang lebih rendah dibandingkan data masukan dengan dimensi yang lebih tinggi.

Pada Penelitian *RGB-based Android Malware Detection and Classification Using Convolutional Neural Network* [7], klasifikasi yang dilakukan pada *malware* yang terdapat di *platform Android* dengan menggunakan *dataset AndroZoo* yang memuat *file APK*. *File dex* dan *manifest* yang terdapat di dalam *file APK* akan dilakukan analisis dan *mapping* sehingga didapat citra RGB. Karena citra RGB terdiri dari 3 *channel* warna, komponen *channel* warna merah didapat dari *mapping* semua pemanggilan API dan *opcode* dalam *file dex*, komponen *channel* warna hijau didapat dari *mapping* semua *permission* dan layanan yang digunakan dalam *file manifest.xml*, dan komponen *channel* biru didapat dari *mapping permission* dan pemanggilan API yang dicurigai. Selanjutnya, untuk mengklasifikasikan citra *malware* RGB mengembangkan model klasifikasi Convolutional Neural Network. Nilai akurasi tertinggi yang dihasilkan model CNN dengan konfigurasi 2 *convolution layer* sebesar sebesar 98.77%, serta rata-rata *false negative* dan *false positive* sebesar 1.72% dan 0.72% Dari penelitian

ini, dapat disimpulkan bahwa citra RGB cukup efektif digunakan dalam klasifikasi *malware*.

Berdasarkan penelitian [1] dan [5], CNN merupakan metode yang efektif dalam melakukan analisis citra, serta memiliki performa yang tinggi untuk masalah klasifikasi citra. Penelitian ini akan berfokus mengembangkan sistem klasifikasi *malware* menggunakan data citra dengan mengimplementasikan Convolutional Neural Network dengan arsitektur yang optimal, serta membandingkan performa klasifikasi *malware* terhadap citra *malware* dan citra berwarna.

2.4 Tinjauan Objek

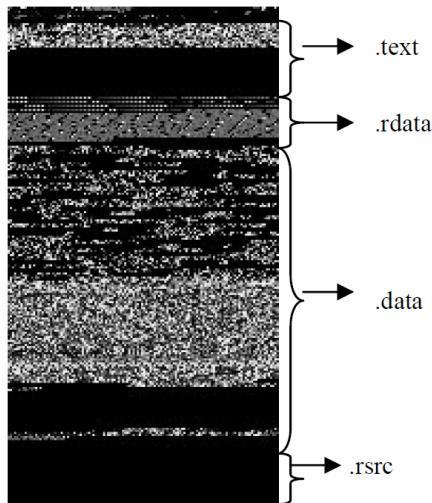
Pada bagian ini akan diulas mengenai objek-objek yang terkait dengan klasifikasi *malware*.

2.4.1 *Malware Family*

Malware family adalah sekelompok *malware* yang mempunyai persamaan kode atau sifat serangan pada sistem. Suatu *malware family* bisa mempunyai beberapa varian dalam suatu *family*, varian berasal dari *code base* yang sama sehingga pola infeksi atau penyerangan memiliki kemiripan dengan *family* yang sama [8]. Dengan mengenali *malware family*, penyerangan yang dilakukan suatu *malware* dapat dikenali serta penanganan dapat ditentukan lebih cepat dan tepat.

2.4.2 *Dataset Citra Grayscale*

Dataset citra *grayscale* yang digunakan adalah Malware Image (Malimg) yang bersumber dari penelitian yang dibuat oleh Faculty of Computer Science, Computer Engineering, and Electrical Engineering, University of California. *Dataset* Malimg ini berisi *malware family* yang sering ditemukan pada *file executable*, dan *dataset* sudah diolah sebelumnya menjadi citra *grayscale*. Selain itu, terdapat 7 jenis *malware* di dalam *dataset*, dan setiap jenisnya terdiri dari beberapa variasi *malware family* dengan jumlah keseluruhan adalah 25 *family*, seperti pada Tabel 2.9. *Malware* dengan *family* yang sama akan memiliki kemiripan tekstur pada citra *grayscale*.



Gambar 2.12 Bagian dari Citra *Grayscale Malware* [12]

Citra *grayscale malware* terdiri dari 4 bagian yaitu .text, .rdata, .data, dan .rsrc, seperti yang terlihat pada Gambar 2.12. Pada bagian .text memuat *executable codes* yang berfungsi mengeksekusi atau menjalankan program *malware* tersebut. Pada .rdata hanya berisi padding yang digunakan untuk menutup atau mengakhiri bagian awal. Bagian .data menjelaskan *uninitialized code* yang ditandai dengan warna hitam dan *initialized code* yang ditandai dengan arsiran hitam dan putih yang merupakan isi dari malware itu. Bagian terakhir adalah .rsrc yang merupakan sumber daya yang dipakai modul, seperti gambar ikon aplikasi. Ukuran dimensi citra *grayscale malware* dapat bervariasi bergantung dari panjangnya *source code* di dalam suatu *malware*.

Malware dibagi menjadi beberapa jenis berdasarkan sifatnya. Pada Tabel 2.9 berikut ini merupakan *family* dan jenis *malware* yang terdapat pada *dataset* Malimg, yaitu:

Tabel 2.9 *Family* dan Jenis *Malware*

No.	Family	Jenis	Sampel
1	Adialer.C	Dialer	122
2	Agent.FYI	Backdoor	116
3	Allapple.A	Worm	2949
4	Allapple.L	Worm	1591
5	Alueron.gen!J	Trojan	198
6	Autorun.K	Worm AutoIT	106
7	C2LOP.P	Trojan	146
14	Lolyada_AA2	PWS	184
15	Lolyada_AA3	PWS	123
16	Lolyada_AT	PWS	159
17	Malex.gen!J	Trojan	136
18	Obfuscator_AD	TDownloader	142
19	RBot!gen	Backdoor	158
20	Skintrim_N	Trojan	80

Tabel 2.9 Family dan Jenis Malware (Lanjutan)

No.	Family	Jenis	Sampel
8	C2LOP.gen!G	Trojan	200
9	Dialplatform.B	Dialer	177
10	Dontoo.A	TDownloader	162
11	Fakerean	Rogue	381
12	Instantaccess	Dialer	431
13	Lolyada-AA1	PWS	213
21	Swizzor.gen!E	TDownloader	128
22	Swizzor.gen!I	TDownloader	132
23	VB.AT	Worm	408
24	Wintrim.BX	TDownloader	97
25	Yuner.A	Worm	800

Dataset yang digunakan berisi citra *grayscale* yang merupakan hasil visualisasi dari *file malware executable*, dengan jumlah 9339 citra yang dibagi ke dalam 25 *family*, serta setiap citra *grayscale* sudah dimasukkan ke dalam folder menurut *malware family* masing-masing.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

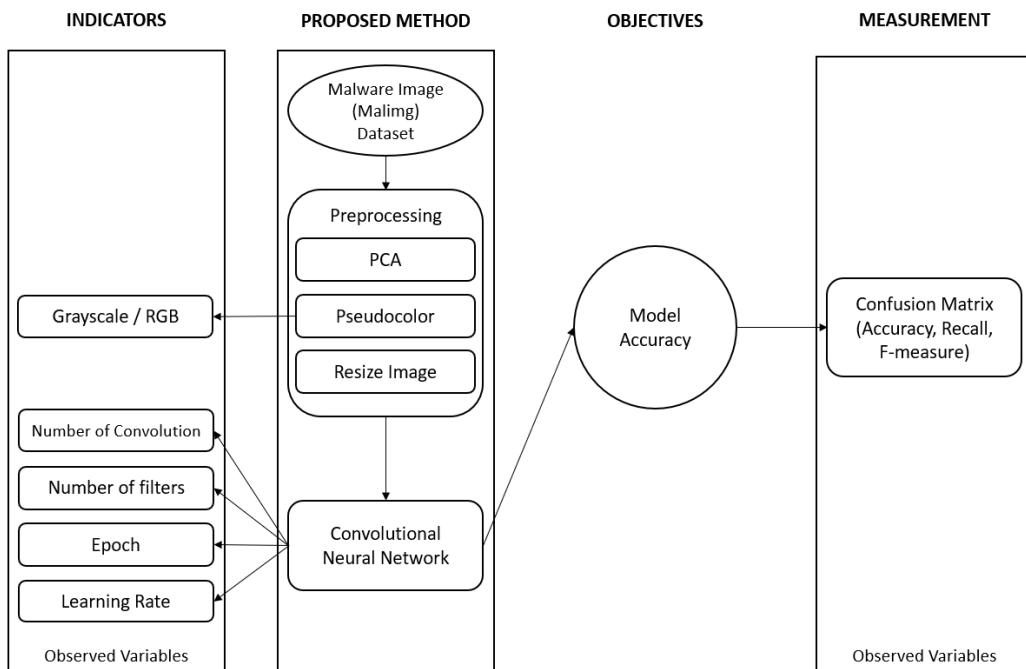
Pada Bab 1 telah dijelaskan bahwa banyak metode *machine learning* yang sudah dikembangkan untuk melakukan klasifikasi *malware* berbasiskan citra *grayscale*. Pada penelitian ini, metode Convolutional Neural Network dipilih karena menghasilkan akurasi yang tinggi dalam mengklasifikasikan citra, serta mampu menangani *dataset* yang besar. Selain itu, ekstraksi fitur pada tahap *preprocessing* akan menerapkan Principal Component Analysis (PCA) untuk menangani masalah *curse of dimensionality* dengan melakukan *dimension reduction* atau mereduksi dimensi data masukan dengan menyeleksi fitur-fitur penting di dalam data, tujuannya untuk meningkatkan akurasi klasifikasi [13]. Kemudian untuk meningkatkan akurasi lagi, klasifikasi *malware* akan menggunakan citra *grayscale* dan berwarna RGB, citra RGB didapat dari hasil teknik *pseudocolor* dengan melakukan pemetaan *channel* warna merah, biru, dan hijau dari citra *grayscale*. Citra RGB digunakan dalam klasifikasi karena mempunyai warna yang lebih kaya dibandingkan citra *grayscale* [7]. Hasil akhir penelitian adalah melakukan perbandingan performa nilai akurasi dan *recall* antara klasifikasi *malware* yang menggunakan citra *grayscale* dan RGB.

Dataset yang digunakan berupa citra *grayscale malware* dengan format citra PNG yang terdiri dari 7 jenis *malware* yang terbagi lagi menjadi *malware family* dengan total 25 *family*. *Dataset* akan dibagi dengan rasio 90:10 yang artinya 90% data belajar (*training*) dan 10% data uji (*testing*). Sebelum melakukan klasifikasi, *preprocessing* yang akan dilakukan meliputi proses ekstraksi fitur dengan PCA dan teknik *pseudocolor* untuk mendapatkan citra RGB dari citra *grayscale malware*. Lalu, proses selanjutnya adalah melakukan *resize image* baik pada citra *malware* dengan model warna *grayscale* dan RGB menjadi ukuran 128×128 .

Masukan untuk model klasifikasi CNN dalam penelitian ini adalah citra *malware* baik dengan citra *grayscale* dan citra RGB yang sudah dilakukan *resizing* untuk memudahkan pemrosesan dalam model CNN. Keluaran atau hasil dari sistem klasifikasi *malware* yang dikembangkan adalah performa model klasifikasi terbaik yang meliputi nilai akurasi, *F-measure*, dan *recall* antara citra *grayscale* dan citra RGB yang membuktikan seberapa signifikan pengaruh warna terhadap performa model klasifikasi.

3.2 Kerangka Pemikiran

Berikut merupakan kerangka pemikiran dari penelitian yang akan dikembangkan untuk klasifikasi *malware* dengan menggunakan metode Convolutional Neural Network.



Gambar 3.1 Kerangka Pemikiran

Berdasarkan Gambar 3.1 menunjukkan beberapa indikator yang akan diuji dalam penelitian ini. Masing-masing indikator dipilih karena dapat memengaruhi hasil dari model klasifikasi yang akan dikembangkan sehingga diharapkan dapat meningkatkan akurasi pada model CNN. Penelitian dimulai dengan data *preprocessing* sebelum data citra diproses dalam model CNN, indikator pada tahap ini berupa citra masukan yang digunakan untuk klasifikasi, citra *grayscale* dan citra RGB. Citra *grayscale malware* merupakan citra asli yang terdapat dari *dataset*, sedangkan citra RGB merupakan keluaran dari teknik *pseudocolor* pada citra *grayscale* sehingga menghasilkan citra berwarna dengan *channel* RGB. Penggunaan citra RGB dalam klasifikasi dikarenakan citra RGB memiliki warna dan fitur yang lebih banyak dibandingkan citra *grayscale* sehingga diharapkan bisa meningkatkan akurasi model klasifikasi. Oleh karena itu, diperlukan pengujian apakah informasi warna yang lebih banyak pada citra RGB dapat memengaruhi terhadap performa akurasi pada klasifikasi model CNN dibandingkan citra *grayscale malware*.

Pada model CNN sendiri terdapat beberapa indikator *hyperparameter* yang akan diobservasi, pemilihan indikator *hyperparameter* akan menentukan

banyaknya kombinasi nilai pengujian yang terbentuk. Berikut ini merupakan penjelasan indikator *hyperparameter* yang akan digunakan pada model CNN:

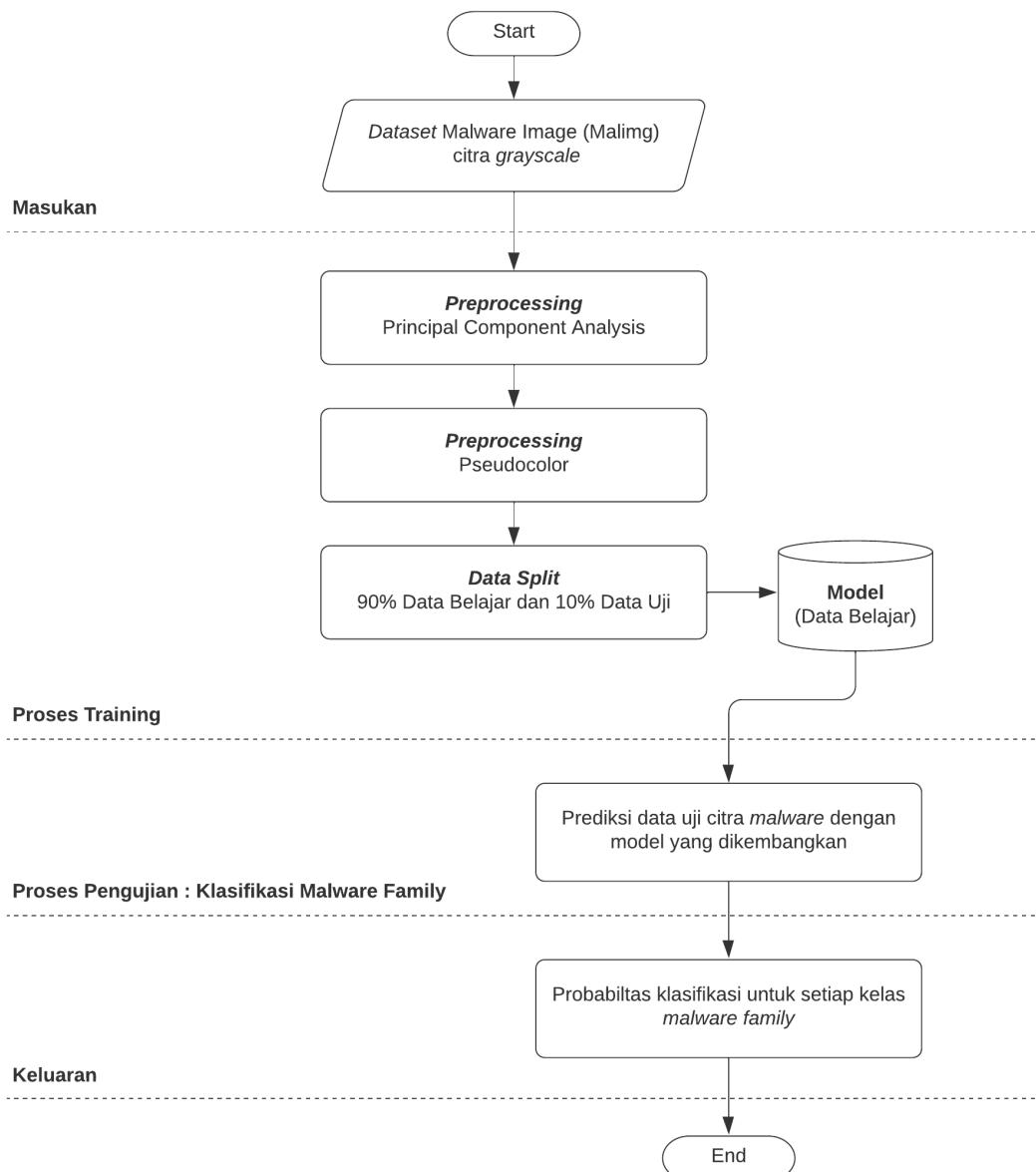
1. *Number of convolution* : menentukan jumlah *convolution layer* yang digunakan dalam arsitektur Convolutional Neural Network. *Convolution layer* berperan penting untuk mengenali fitur yang terdapat dalam citra masukan sehingga secara signifikan memengaruhi kinerja model CNN. Konfigurasi *convolution layer* yang terlalu dalam atau banyak melakukan perhitungan operasi konvolusi dapat menyebabkan *overfitting* karena model klasifikasi terlalu banyak mempelajari fitur pada data belajar sehingga model kesulitan melakukan generalisasi pada data belajar. Oleh karena itu, diperlukan pengujian pada untuk menentukan jumlah operasi konvolusi yang tepat supaya model CNN dapat menghasilkan klasifikasi dengan akurasi yang tinggi.
2. *Number of filter* : merupakan *hyperparameter* pada *convolution layer* untuk menentukan jumlah *filter* atau *kernel* yang digunakan dalam suatu *convolution layer*. Seperti yang dijelaskan pada Bab 2.1.5.1 suatu *convolution layer*, maka semakin banyak fitur atau *feature map* yang diperoleh dari citra masukan. Tetapi, jika jumlah *filter* yang dipakai terlalu banyak maka *feature map* yang dihasilkan akan terlalu banyak menyebabkan model yang dibangun terlalu banyak mengenali fitur sehingga tidak mencapai generalisasi. Oleh karena itu, diperlukan pengujian untuk menentukan jumlah *filter* yang tepat dikombinasikan dengan jumlah *convolution layer* untuk membangun arsitektur model CNN yang dapat melakukan klasifikasi secara efektif.
3. *Epoch* : *hyperparameter* untuk menentukan jumlah iterasi siklus *feed-forward* dan *backpropagation* pada pelatihan model CNN dengan seluruh data belajar, semakin besar nilai *epoch* maka semakin banyak pelatihan yang dilakukan sehingga akan meningkatkan akurasi pada model CNN. Tetapi, jika nilai *epoch* yang terlalu tinggi, maka pelatihan yang dilakukan akan terlalu banyak menyebabkan model rentan mengalami *overfitting*, kondisi di mana model memiliki akurasi yang tinggi pada data belajar, tetapi memiliki akurasi yang rendah pada data uji. Oleh karena itu, diperlukan pengujian untuk menentukan nilai *epoch* agar pelatihan model dapat berjalan efektif.
4. *Learning rate* : *hyperparameter* yang berfungsi untuk mengatur seberapa besar pembaharuan bobot pada saat pelatihan. Jika nilai *learning rate* terlalu rendah maka model membutuhkan pelatihan yang lama untuk model mencapai konvergen, sehingga membutuhkan nilai *epoch* yang relatif besar, sedangkan nilai *learning rate* yang terlalu tinggi menyebabkan titik konvergensi pada model terlewat karena pembaharuan bobot terlalu tinggi. Oleh karena itu,

diperlukan pengujian untuk menentukan nilai *learning rate* yang tepat dapat sehingga menghasilkan model CNN yang konvergen atau *fit*, serta dapat melakukan klasifikasi dengan akurat.

Objektif atau tujuan yang ingin dicapai dari penelitian ini adalah membandingkan nilai akurasi dan *recall* pada klasifikasi *malware* terhadap *citra grayscale* dan citra RGB. Nilai-nilai pengukuran tersebut diperoleh dari *confusion matrix*.

3.3 Urutan Proses Global

Berikut merupakan flowchart yang menggambarkan urutan proses global pada penelitian ini.

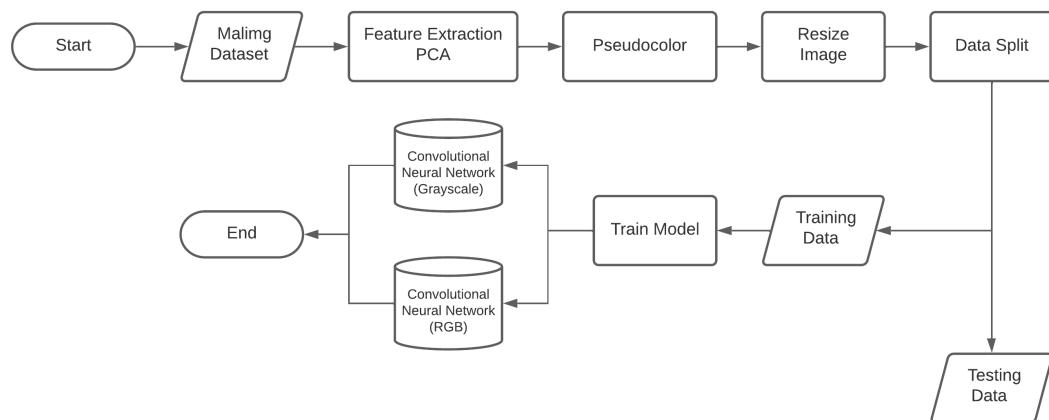


Gambar 3.2 Flowchart Urutan Proses Global

Pada Gambar 3.2 menunjukkan proses-proses penelitian secara global. Proses penelitian dimulai dari data *preprocessing*, lalu citra *malware* akan diklasifikasikan dengan menerapkan metode CNN. Tahap data *preprocessing* dilakukan untuk mempersiapkan data sebelum diolah oleh model klasifikasi. Data *preprocessing* yang dilakukan berupa Principal Component Analysis dan *pseudocolor*.

Sistem klasifikasi *malware family* terbagi menjadi 2 proses, proses pelatihan dan proses pengujian. Proses pelatihan dilakukan supaya model dapat memahami pola tersembunyi di dalam data citra *malware*, sedangkan proses pengujian dilakukan untuk menguji apakah model dapat menentukan kelas *malware family* berdasarkan citra *malware* dengan akurat.

3.3.1 Proses Pelatihan (*Training*)



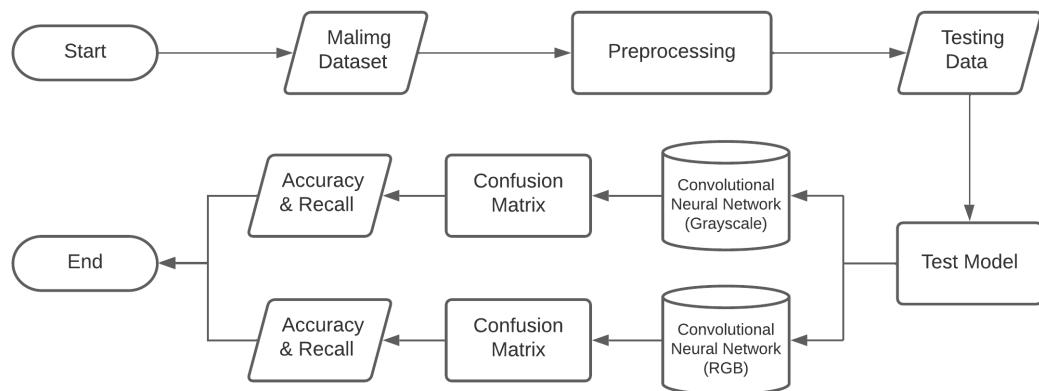
Gambar 3.3 Flowchart Proses Pelatihan

Berikut ini adalah uraian dari *flowchart* proses pelatihan (*training*) pada Gambar 3.3 yang dilakukan pada penelitian ini:

1. *Dataset* yang akan dipakai adalah *Dataset Malware Image* yang memuat citra *grayscale malware* dengan jumlah 9339 citra dan terdiri dari 25 *malware family*.
2. *Feature extraction* dengan PCA untuk melakukan *dimension reduction*, serta menyeleksi fitur-fitur penting untuk membangun model klasifikasi.
3. Citra *grayscale malware* akan diproses dengan teknik *pseudocolor* dengan menggunakan *color-map* atau pemetaan warna sehingga menghasilkan citra berwarna dengan model warna RGB.
4. Setiap citra *malware* baik dengan model warna *grayscale* maupun RGB akan dilakukan *resizing* menjadi ukuran 128×128 untuk panjang dan lebarnya.

5. *Dataset* akan dilakukan *data split* sehingga membagi *dataset* menjadi 90% sebagai data belajar (*training*) dan 10% sebagai data uji (*testing*).
6. Baik citra *grayscale* dan citra RGB dari akan digunakan dalam proses pelatihan dan pengujian pada model CNN sehingga akan menghasilkan 2 model klasifikasi CNN.
7. Data belajar akan dimasukkan ke dalam model untuk melakukan pelatihan model CNN dalam mengenali fitur dan pola yang terdapat pada data belajar. *Hyperparameter* yang dipilih akan diuji juga untuk mengoptimalkan performa model CNN supaya menghasilkan akurasi klasifikasi yang tinggi.
8. Masing-masing model CNN yang selesai dilakukan pelatihan akan disimpan untuk dilakukan proses pengujian.

3.3.2 Proses Pengujian (*Testing*)



Gambar 3.4 Flowchart Proses Pengujian

Berikut ini adalah uraian dari *flowchart* proses pengujian *testing* pada Gambar 3.4 yang dilakukan pada penelitian ini:

1. *Dataset* yang akan dipakai adalah *Dataset Malware Image* yang memuat citra *grayscale malware* dengan jumlah 9339 citra dan terdiri dari 25 *family*.
2. *Dataset* Malimg akan dilakukan tahap *preprocessing* sehingga menghasilkan 10% dari keseluruhan dari *dataset* sebagai data uji
3. Data uji dari citra *grayscale* dan RGB *malware* akan digunakan dalam proses pengujian terhadap masing-masing model CNN yang sudah dilakukan pelatihan sebelumnya.
4. Kinerja model CNN antara klasifikasi *malware* dengan citra *grayscale* dan RGB masing-masing akan diukur menggunakan *confusion matrix* dengan mengobservasi nilai akurasi dan *recall* yang dihasilkan model CNN.

3.4 Analisis Manual

Pada bagian ini, dilakukan analisis tahapan proses yang akan dilakukan dalam sistem dengan contoh perhitungan manual.

3.4.1 Data Sampel

Data sampel yang digunakan penelitian ini adalah Malware Image (Malimg) yang diambil dari penelitian berjudul "*Malware images: visualization and automatic classification*" [12]. Seperti yang dijelaskan pada Bab 2.4.2, *dataset* ini memuat citra *grayscale* hasil visualisasi dari *file executable malware* yang sudah diolah sebelumnya. Jumlah data yang akan digunakan dalam penelitian adalah 9339 citra yang terbagi dalam 25 *malware family*. *Dataset* akan dibagi menjadi data *training* sebagai data belajar pada model CNN dan data *testing* digunakan untuk pengujian setelah pelatihan selesai, dibagi dengan rasio 90:10 sehingga data *training* berjumlah 8394 citra, sedangkan data *testing* berjumlah 945 citra.

Dataset Malimg yang digunakan dalam penelitian ini berisi 7 jenis kelas *malware* dengan penjelasan sebagai berikut [12].

1. *Backdoor* merupakan jenis *malware* yang dapat melewati proses autentikasi sehingga pihak yang tidak berwenang atau *hacker* mendapatkan hak akses terhadap sistem komputer, jaringan, atau aplikasi perangkat lunak. Akibatnya, *hacker* memiliki akses secara jarak jauh terhadap sumber daya, seperti *server* atau sistem berkas untuk menjalankan perintah yang dapat menyerang sistem atau mengambil data konfidensial.
2. *Dialer* merupakan jenis *malware* yang secara diam-diam melakukan sambungan telepon baik ke nomor telepon lokal atau internasional melalui jaringan internet tanpa sepengetahuan pengguna sehingga menyebabkan meningkatnya tagihan telepon secara signifikan.
3. PWS atau *password stealer* merupakan jenis *malware* yang digunakan untuk mencuri informasi pengguna, seperti *username* dan *password*. PWS biasanya menggunakan komponen *keylogger*, suatu komponen atau program yang akan terus aktif di dalam memori untuk memonitor dan merekam setiap masukan dari ketikan *keyboard* yang dilakukan pengguna.
4. *Rogue* merupakan jenis *malware* yang biasanya berupa penipuan di internet yang memberitahukan bahwa terdapat suatu virus di dalam perangkat komputer dan meyakinkan pengguna untuk menggunakan aplikasi tertentu untuk mengamankan komputer yang sebenarnya merupakan *malware* berbahaya.

5. *TDownloader* atau *Trojan downloader* merupakan jenis *malware* yang digunakan untuk mengunduh *malware* bertipe *Trojan* yang dapat membahayakan perangkat komputer.
6. *Trojan* atau *trojan horse* merupakan jenis *malware* yang biasanya menyamar sebagai aplikasi perangkat lunak tetapi sebenarnya *malware* berbahaya yang dapat merusak sistem komputer, mengintai aktivitas pengguna, dan mencuri informasi pribadi. *Trojan* seringkali mengelabui pengguna untuk mengunduh suatu aplikasi tanpa mengetahui aplikasi tersebut merupakan *trojan*.
7. *Worm* merupakan jenis *malware* yang berjalan sendiri tanpa membutuhkan bantuan program lainnya dan mereplikasi dirinya sendiri secara otomatis melalui jaringan untuk menyerang celah keamanan pada sistem. *Worm* biasanya menggunakan banyak memori dan *bandwidth* sehingga dapat menyebabkan malfungsi pada sistem dan jaringan.

Selanjutnya, *dataset* dibagi menjadi 25 *malware family* yang akan diklasifikasikan pada penelitian ini, yaitu [26] [27]:

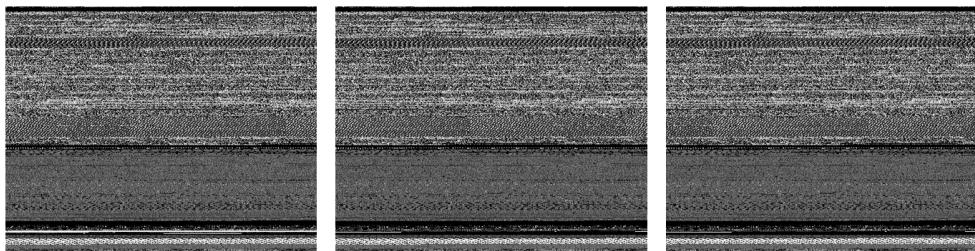
1. Adialer.C merupakan suatu *trojan dialer* yang memanfaatkan suatu komputer untuk melakukan panggilan telepon pada nomor *premium-rate* sehingga dapat menyebabkan tagihan telepon yang mahal. Tipe *malware* ini hanya menyerang komputer dengan *modem* yang terhubung dengan sambungan telepon.
2. Agent.FYI merupakan suatu *backdoor* yang menyerang melalui celah keamanan pada Domain Name System (DNS) Server Service pada Windows Server untuk memasang *malware* lain atau aplikasi tanpa sepengetahuan pengguna.
3. Allapple.A merupakan varian *worm* dalam *family* Allapple yang menginfeksi komputer-komputer yang terhubung dengan suatu *local area network* (LAN) dan melakukan penyerangan *denial-of-service* (DOS) pada situs web.
4. Allapple.L merupakan varian *worm* dalam *family* Allapple yang menyebar pada komputer-komputer dengan cara mereplikasi diri pada *removable driver*, folder jaringan, dan menyebar melalui *email*.
5. Alueron.gen!J merupakan suatu *trojan* yang dapat melakukan aksi mencurigakan pada komputer oleh *hacker* tanpa sepengetahuan pengguna, seperti memodifikasi konfigurasi pengaturan pada komputer.
6. Autorun.K merupakan suatu *worm* yang menyerang sistem operasi Windows dengan mereplikasi diri dari *removable device* ke dalam sistem komputer.
7. C2LOP.P merupakan varian *trojan* dalam *family* C2LOP yang digunakan untuk memodifikasi pengaturan pada *web browser*, *browser bookmark*, dan memunculkan *pop-up* iklan yang tidak diinginkan.

8. C2LOP.gen!G merupakan varian *trojan* dalam *family* C2LOP dengan sifat penyerangan yang sama dengan C2LOP.P.
9. Dialplatform.B merupakan suatu *trojan dialer* untuk melakukan panggilan telepon tanpa sepengetahuan penggunanya, serta bisa digunakan untuk mencuri informasi pribadi pengguna.
10. Dontovo.A merupakan suatu *trojan downloader* yang berfungsi untuk mengunduh dan memasang program *malware* berbahaya yang digunakan pada komputer yang terinfeksi.
11. Fakerean merupakan suatu *rogue* yang berpura-pura berfungsi sebagai aplikasi keamanan atau *antivirus* untuk menipu pengguna membayarkan sejumlah uang untuk membeli produk aplikasi *antivirus* yang palsu.
12. Instantaccess merupakan suatu program *dialer* yang menawarkan iklan layanan premium dari suatu situs web, lalu menghubungkan situs web tersebut dengan melakukan panggilan pada nomor dengan biaya tinggi, serta menaruh *trojan* yang berbahaya dalam sistem komputer.
13. Lolyada_AA1 merupakan varian *password stealer* (PWS) dalam *family* Lolyada yang berfungsi secara diam-diam mencuri informasi kredensial pengguna, seperti *password*, pin, dan informasi konfidensial lainnya.
14. Lolyada_AA2 merupakan varian PWS dalam *family* Lolyada yang berfungsi untuk mencuri informasi kredensial pengguna.
15. Lolyada_AA3 merupakan varian PWS dalam *family* Lolyada yang berfungsi untuk mencuri informasi kredensial pengguna.
16. Lolyada_AT merupakan varian PWS dalam *family* Lolyada yang berfungsi untuk mencuri informasi kredensial pengguna.
17. Malex.gen!J merupakan suatu *trojan* yang digunakan untuk mengunci perangkat komputer yang terinfeksi, serta meminta bayaran sejumlah uang sebagai tembusan untuk membuka perangkat komputer.
18. Obfuscator.AD merupakan suatu *trojan downloader* yang berfungsi untuk mengunduh file *malware* untuk menyerang komputer, serta menerapkan teknik *obfuscation* yang bertujuan untuk menghindari deteksi *malware*.
19. RBot!gen merupakan suatu *backdoor* yang memungkinkan *hacker* untuk mengontrol komputer yang terinfeksi, melakukan *denial-of-service* (DoS), dan mendapatkan informasi sistem dengan mengeksplorasi celah keamanan pada sistem operasi Windows.
20. Skintrim.N merupakan suatu *trojan* yang berfungsi untuk memunculkan banyak *pop-up* iklan yang mengganggu pengguna.
21. Swizzor.gen!E merupakan suatu *trojan downloader* dalam *family* Swizzor yang

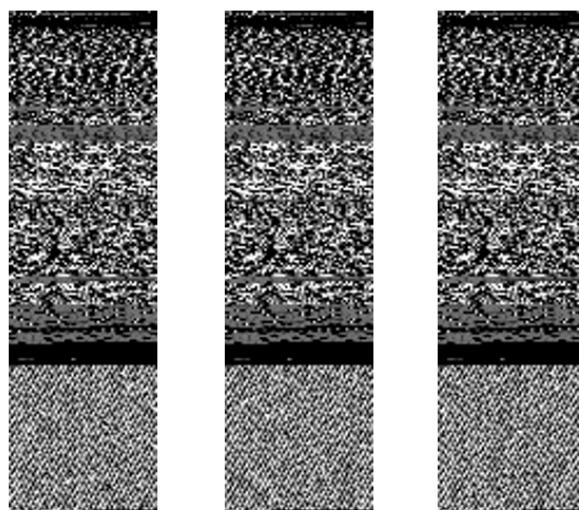
berfungsi untuk mengunduh dan memasang program atau *malware*, serta menambahkan *bookmark* atau *toolbar* pada *browser* tanpa sepengetahuan atau seizin pengguna, virus *trojan* ini bisa disebabkan karena mengunjungi situs berbahaya.

22. Swizzor.gen!I merupakan suatu *trojan downloader* dalam *family* Swizzor yang mempunyai kemiripan sifat penyerangan dengan *family* Swizzor.gen!E.
23. VB.AT merupakan suatu *worm* yang dibuat menggunakan bahasa pemrograman Visual Basic yang menyebar dengan menyisipkan kode pada suatu file atau program dalam komputer atau jaringan sehingga program-program yang terinfeksi tidak bisa berjalan dengan benar.
24. Wintrim.BX merupakan suatu *trojan downloader* yang berfungsi untuk mengunduh dan memasang komponen Wintrim, serta fungsi lainnya adalah memunculkan *pop-up* iklan berdasarkan *browsing history* dan memantau aktivitas *browsing* pengguna.
25. Yuner.A merupakan suatu *worm* yang menghentikan proses yang berhubungan dengan keamanan pada sistem komputer dan mereplikasi diri pada saat suatu *removable drive* yang ditambahkan ke dalam sistem komputer.

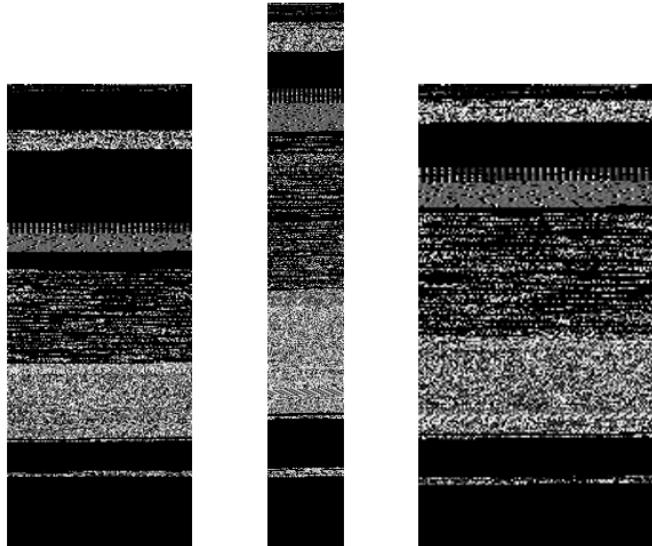
Berikut merupakan contoh *citra grayscale* dari beberapa *malware family* :



Gambar 3.5 *Family* Adialer.C



Gambar 3.6 Family Dialplatform.B



Gambar 3.7 Family Dontovo.A

3.4.2 *Preprocessing*

Pada tahap ini dilakukan *preprocessing* terhadap *dataset* sebelum digunakan untuk proses belajar pada model klasifikasi.

3.4.2.1 Principal Component Analysis

Pada tahap ini dilakukan ekstraksi fitur untuk mereduksi dimensi data untuk kompresi citra dengan mempertahankan fitur-fitur signifikan yang akan digunakan pada proses pembelajaran model klasifikasi. Proses ekstraksi fitur menggunakan Principal Component Analysis (PCA) untuk memproyeksikan data menjadi *principal component*, komponen hasil proyeksi data ke dimensi yang lebih kecil yang berisikan fitur dengan jumlah informasi atau *variance* yang tinggi. Pada penelitian ini untuk melakukan ekstraksi fitur PCA akan menggunakan pustaka Scikit-learn.

```
# Penerapan PCA pada citra grayscale malware
grayscale_img = cv2.imread('/dataset/malimg_paper_dataset_imgs/Swizzor.gen!I/03bb3a23d0aa3bafae88c6c26dd7047d.png')
pca = PCA(n_components=0.95)
compressed_img = pca.fit_transform(grayscale_img)
```

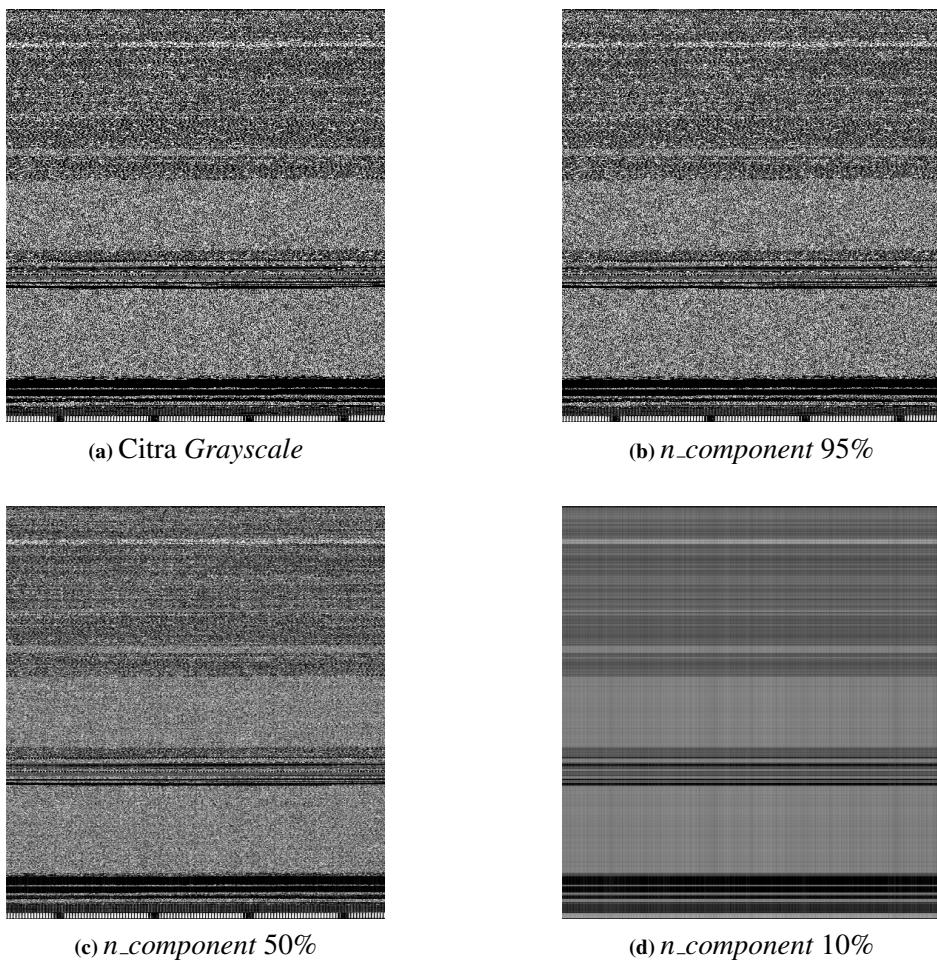
Gambar 3.8 Kode Python untuk Principal Component Analysis

Sebelum melakukan proses ekstraksi fitur, citra *grayscale malware* dilakukan normalisasi terlebih dahulu dengan membagi nilai *pixel* dengan 255 sehingga setiap *pixel* akan memiliki nilai dengan rentang 0-1, 0 merupakan nilai intensitas keabuan terendah, sedangkan 1 merupakan nilai intensitas keabuan

tertinggi. Kemudian, citra *grayscale* akan dilakukan ekstraksi fitur dengan PCA, kode Python untuk PCA dapat dilihat pada Gambar 3.8.

Citra *grayscale* dimuat terlebih dahulu ke dalam variabel *grayscale_img* dengan menggunakan fungsi *imread()* dari pustaka OpenCV, parameter yang digunakan adalah *path* atau direktori *file* citra disimpan. Pada proses ekstraksi fitur, parameter yang digunakan pada fungsi *PCA()* adalah *n_component=0.95* untuk menentukan persentase tingkat *variance* yang ingin dipertahankan saat proses PCA. Kemudian, fungsi *fit_transform* dipanggil untuk melakukan proses PCA terhadap masukan yang sudah ditentukan.

Gambar 3.9 merupakan contoh beberapa citra *grayscale malware family* Swizzor.gen!I setelah dilakukan proses ekstraksi fitur dengan PCA.



Gambar 3.9 Citra *Grayscale Malware* setelah PCA dengan Berbagai Nilai *n_component*

3.4.2.2 *Pseudocolor*

Dataset Malimg yang digunakan berisikan citra *grayscale malware*, citra *grayscale* hanya memiliki 1 *channel* warna saja, yaitu nilai intensitas kecerahan.

Jumlah fitur merupakan salah satu faktor yang memengaruhi model klasifikasi dalam membedakan suatu citra dengan citra lainnya. Oleh karena itu, dengan menerapkan *pseudocolor*, citra *grayscale* akan dikonversi menjadi citra berwarna dengan model warna RGB agar memperbanyak fitur yang terdapat pada citra masukan sehingga diharapkan dapat meningkatkan akurasi model klasifikasi. dikonversi menjadi citra RGB.

```
# Penerapan Pseudocolor pada citra grayscale  
  
grayscale_img = cv2.imread('/malimg_paper_dataset_imgs/Adialer.C/000bde2e9a94ba41c0c111ffd80647c2.png')  
rgb_img = cv2.applyColorMap(grayscale_img, cv2.COLORMAP_JET)
```

Gambar 3.10 Kode Python untuk *Pseudocolor*

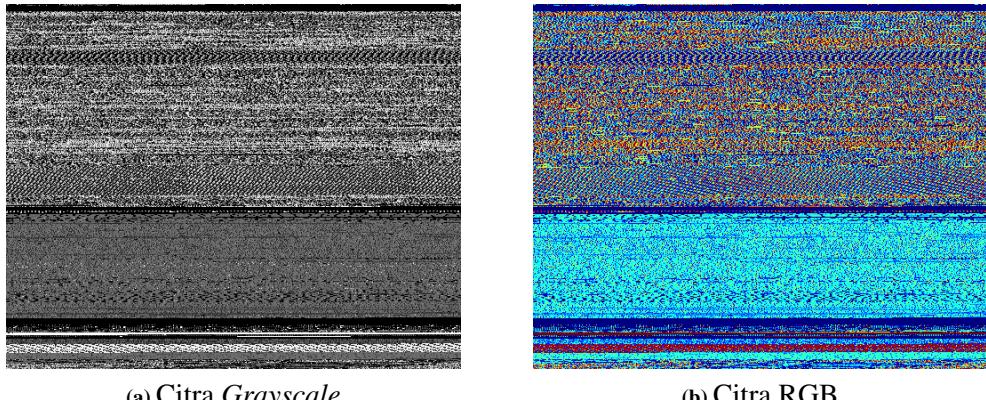
Gambar 3.10 merupakan kode Python untuk penerapan *pseudocolor* dengan memakai pustaka OpenCV. Citra *grayscale* dikonversi menjadi citra RGB dengan memakai fungsi *applyColorMap()* yang dipakai untuk pemetaan warna terhadap citra *grayscale*. Parameter fungsi *applyColorMap()* yang digunakan dalam penelitian yaitu, parameter pertama adalah citra *grayscale* sebagai masukan, sedangkan parameter kedua adalah *color-map* atau palet (skala) warna yang akan digunakan untuk pemetaan pada citra *grayscale*. Banyak jenis *Color-map* dapat digunakan untuk pemetaan warna [25], tetapi pada penelitian ini *color-map* yang digunakan adalah *COLORMAP_JET*.

Skala warna yang terdapat pada *COLORMAP_JET* dapat dilihat pada Gambar 3.11. Semakin rendah nilai intensitas keabuan akan diganti dengan warna yang condong di arah kiri dalam skala warna, sedangkan semakin tinggi nilai intensitas keabuan akan diganti dengan warna yang condong ke arah kanan skala.



Gambar 3.11 Skala Warna *COLORMAP_JET*

Gambar 3.12 merupakan perbandingan antara citra *grayscale* dan citra RGB pada kelas *malware family Adialer.C*.



Gambar 3.12 Perbandingan Citra *Grayscale* dan Citra *RGB* *Malware Family Adialer.C*

3.4.3 Perhitungan Convolutional Neural Network

Visualisasi citra *malware* dimasukkan ke model Convolutional Neural Network dalam bentuk representasi matriks atau *array* 2 dimensi yang berisikan nilai numerik untuk setiap *pixel*. Arsitektur Convolutional Neural Network yang digunakan untuk perhitungan manual di bagian ini adalah sebagai berikut :

1. *Input layer* yang digunakan berupa *convolution layer* dengan *filter size* 32 dan ReLu sebagai fungsi aktivasinya.
2. *Hidden layer* berupa *pooling layer* dengan ukuran 2×2 , serta nilai *stride* 2.
3. *Fully connected layer* yang terdiri dari *dense layer* dan *output layer* yang menggunakan fungsi aktivasi *softmax*.

Contoh nilai matriks citra dapat dilihat pada Tabel 3.1. Operasi konvolusi akan dilakukan pada matriks citra dengan menggunakan *kernel*. Pada penelitian ini, operasi konvolusi berjalan secara satu dimensi ke arah samping, perhitungan dilakukan sampai keseluruhan *pixel* citra berhasil diproses.

Tabel 3.1 Contoh Matriks Citra Masukan

30	30	90	90	60
50	50	100	80	80
40	50	70	70	30
100	60	70	30	30
50	50	80	70	20

Citra masukan akan diberikan *padding*, *padding* digunakan agar menghasilkan keluaran *feature map* dengan ukuran yang sama dengan masukan.

Tabel 3.2 merupakan contoh matriks dari citra masukan yang sudah diberikan *padding* yang bernilai 0.

Tabel 3.2 Contoh Matriks Citra Setelah Diberi *Padding*

0	0	0	0	0	0	0
0	30	30	90	90	60	0
0	50	50	100	80	80	0
0	40	50	70	70	30	0
0	40	50	70	70	30	0
0	100	60	70	30	30	0
0	0	0	0	0	0	0

Kernel atau *filter* yang digunakan dalam *convolution layer* pada perhitungan manual ini berukuran 3×3 . Dengan melakukan perhitungan nilai bobot *kernel* terhadap matriks citra masukan akan menghasilkan *feature map* dalam bentuk matriks dua dimensi. Proses konvolusi bergerak secara satu dimensi, di mana pergerakan *kernel* adalah dari arah samping kanan bergerak sampai *pixel* citra terakhir. Inisialisasi awal nilai bobot yang terdapat dalam *kernel* 3×3 ditentukan secara acak terdistribusi normal dengan metode *Gaussian Normal Distribution* melalui Persamaan 2.5, digunakan nilai *mean* = 0 dan standar deviasi = 0.05. Contoh *kernel* 3×3 yang digunakan dapat dilihat pada Tabel 3.3.

Tabel 3.3 Contoh *Kernel* 3×3

0.04963341	0.02579839	0.07469831
0.01994597	0.00615295	0.02461383
0.00347505	0.04976287	-0.06364544

Operasi konvolusi akan berjalan pada matriks citra dari indeks [0,0], indeks [0,1], dan seterusnya hingga mencapai indeks [x, y] yang berada di ujung kanan bawah matriks citra masukan dengan Persamaan 2.6. Operasi konvolusi akan dihitung dengan perkalian $w \times x$ sehingga menghasilkan fitur citra yang disebut *feature map*. Proses perkalian dalam *convolution layer* menggunakan *stride* yang bernilai 1, sedangkan nilai bias adalah 0 yang ditentukan dengan parameter *bias_initializer* = 0.

Berikut merupakan contoh perhitungan operasi konvolusi indeks [0,0] dari matriks citra masukan dengan *kernel* yang sudah didefinisikan sebelumnya.

$$\begin{aligned}
 c_k &= (w_{k0,0}x_{0,0} + b_{0,0}) + (w_{k0,1}x_{0,1} + b_{0,1}) + (w_{k0,2}x_{0,2} + b_{0,2}) + (w_{k1,0}x_{1,0} + b_{1,0}) \\
 &\quad + (w_{k1,1}x_{1,1} + b_{1,1}) + (w_{k1,2}x_{1,2} + b_{1,2}) + (w_{k2,0}x_{2,0} + b_{2,0}) + (w_{k2,1}x_{2,1} + b_{2,1}) \\
 &\quad + (w_{k2,2}x_{2,2} + b_{2,2}) \\
 &= (0 * 0.04963341 + 0) + (0 * 0.02579839 + 0) + (0 * 0.07469831 + 0) + \\
 &\quad (0 * 0.01994597 + 0) + (30 * 0.00615295 + 0) + (30 * 0.02461383 + 0) + \\
 &\quad (0 * 0.00347505 + 0) + (50 * 0.04976287 + 0) + (50 * -0.06364544 + 0) + \\
 &= 30.05044
 \end{aligned}$$

Feature map yang dihasilkan berukuran 3×3 setelah proses konvolusi dilakukan, *feature map* memiliki ukuran yang sama dengan citra masukan yang dapat dilihat pada Tabel 3.4. Ukuran *feature map* dapat bervariasi bergantung pada ukuran *kernel* yang digunakan, serta nilai *stride* yang mengatur besar pergerakan setiap perhitungan konvolusi pada *convolution layer*. Selain itu, jumlah *feature map* dipengaruhi juga oleh jumlah *filter* (*filter size*) yang digunakan dalam suatu *convolution layer*. Sebagai contoh, jika *filter* yang digunakan berjumlah 512, *feature map* yang dihasilkan akan berjumlah 512 juga. Jumlah *filter* yang digunakan bergantung dari arsitektur yang dipakai.

Tabel 3.4 Contoh Matriks *Feature Map* Hasil Konvolusi

30	29	93	93	66
53	61	113	97	89
47	63	86	85	39
106	70	81	42	36
58	65	90	79	24

Kemudian, setiap nilai dalam *feature map* akan dimasukkan ke dalam fungsi aktivasi ReLU yang dihitung dengan Persamaan 2.9. Nilai positif akan dipertahankan, sedangkan nilai negatif akan diubah menjadi 0. Berikut adalah contoh perhitungan fungsi ReLu untuk nilai dari indeks [0,1].

$$\begin{aligned}
 f(x) &= \max(0, x) \\
 &= \max(0, 29) \\
 &= 29
 \end{aligned}$$

Matriks *feature map* yang dihasilkan setelah fungsi ReLu dapat dilihat pada Tabel 3.5.

Tabel 3.5 Contoh Matriks *Feature Map* Setelah Perhitungan Fungsi Aktivasi ReLu

30	29	93	93	66
53	61	113	97	89
47	63	86	85	39
106	70	81	42	36
58	65	90	79	24

Keluaran *feature map* yang sudah diaktivasi dengan fungsi aktivasi ReLU akan diteruskan ke *pooling layer*. Pada *pooling layer* akan menggunakan *max pooling* dengan ukuran 2×2 dan *stride* bernilai 2. *Feature map* dengan ukuran 5×5 akan dibagi 2 sehingga ukurannya menjadi 3×3 , serta masing-masing daerah *pooling* akan diambil nilai maksimumnya. Berikut adalah contoh perhitungan *max pooling* untuk keluaran terhadap indeks [0,0] :

$$\begin{aligned}
 \max_{0,0} &= \max(x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}) \\
 &= \max(30, 29, 53, 61) \\
 &= 61
 \end{aligned}$$

Hasil perhitungan *max pooling* yang diterapkan pada *feature map* dapat dilihat pada Tabel 3.6.

Tabel 3.6 Contoh Hasil Perhitungan *Max Pooling*

61	113	89
106	86	39
65	90	24

Dari *pooling layer*, proses akan dilanjutkan menuju *dropout layer*. *Dropout layer* digunakan untuk menentukan seberapa banyak nilai fitur yang akan diaktifkan dan dinonaktifkan agar model tidak terlalu banyak mempelajari dari fitur yang dihasilkan untuk menghindari terjadinya *overfitting*. *Dropout layer* memiliki *hyperparameter probability* atau nilai probabilitas, serta nilai probabilitas yang digunakan adalah 0.5.

Nilai *random number* ditentukan terlebih dahulu seperti yang dapat dilihat pada Tabel 3.7.

Tabel 3.7 Contoh Inisialisasi Random Number Proses *Dropout*

0.372	0.296	0.6372	0.861
-------	-------	--------	-------

Perhitungan dalam *dropout layer* menggunakan Persamaan 2.8 dengan contoh perhitungan sebagai berikut.

$$\begin{aligned} O_i &= O_i \times \frac{1}{p} \\ &= 0.372 \times \frac{1}{0.5} \\ &= 0.744 \end{aligned}$$

Proses yang sama dilakukan untuk menghitung nilai keluaran lainnya. Tabel 3.8 menunjukkan hasil keluaran perhitungan dari *dropout layer*.

Tabel 3.8 Contoh Hasil Perhitungan dalam *Dropout Layer*

0.744	0.592	0	0
-------	-------	---	---

Sebelum diteruskan ke *fully connected layer*, *feature map* diproses dengan *flatten* menjadi matriks atau *array* 1 dimensi sehingga ukurannya menjadi 9×1 . Pada *fully connected layer*, terdapat 2 *dense layer* yang berfungsi sebagai *input layer* dan *output layer*. *Input layer* digunakan untuk menerima masukan berupa *feature map* yang berbentuk matriks 1 dimensi. Sementara itu, *output layer* berfungsi untuk menghasilkan keluaran prediksi untuk kelas klasifikasi dengan menggunakan fungsi aktivasi *softmax*.

Dense layer terdiri dari banyak *neuron* yang terkoneksi dengan *neuron* lainnya melalui nilai bobot atau *weight*. Nilai-nilai bobot tersebut terkumpul dalam suatu *kernel* yang diinisialisasi secara acak terdistribusi normal sesuai dengan Persamaan 2.5, serta sejumlah dengan ukuran *array* masukan. Tabel 3.9 merupakan contoh *kernel* pada *dense layer* yang berukuran 9×1 .

Tabel 3.9 Contoh *Kernel* pada *Dense Layer*

-0.04107868	0.02992102	0.02201463	0.04467891	0.03614909	0.04670192	-0.04007071	0.00056198	-0.07777447
-------------	------------	------------	------------	------------	------------	-------------	------------	-------------

Pertama-tama, dilakukan perhitungan *dot product* antara masukan *feature map* dengan *kernel* melalui Persamaan 2.7 dengan contoh perhitungan sebagai berikut.

$$\begin{aligned}
 f(x) &= \sum_{i=0}^{n-1} w_i x_i + b \\
 &= (w_0 x_{0,0} + b) + (w_1 x_{0,1} + b) + \dots + (w_8 x_{2,2} + b) \\
 &= (-0.05034731 * 61 + 0) + (0.04319308 * 113 + 0) + \dots \\
 &\quad +(0.06193049 * 24 + 0) \\
 &= 8.09742
 \end{aligned}$$

Hasil perhitungan pada *output layer* selanjutnya akan menjadi masukan untuk fungsi aktivasi *softmax* dan akan dihitung dengan Persamaan 2.10. Perhitungan ini dilakukan dengan mencari probabilitas setiap kelas target klasifikasi yang telah ditentukan. Dalam penelitian ini, prediksi kelas klasifikasi berjumlah 25 *malware family* sehingga *dense layer* akan menghasilkan 25 probabilitas untuk setiap kelas *malware family* dengan total probabilitas merupakan 1 atau 100%.

Untuk menyederhanakan perhitungan manual ini, contoh perhitungan dengan *dense layer* akan menghitung 5 hasil keluaran menggunakan fungsi aktivasi *softmax* sebagai berikut.

$$\begin{aligned}
 f(x_i) &= \frac{\exp(x_i)}{\sum_j \exp(x_j)} \\
 &= \frac{\exp(8.09742)}{\exp(-21.7009) + \exp(-18.25186) + \dots + \exp(-18.16676)} \\
 &= 0.000789439
 \end{aligned}$$

Hasil akhir perhitungan adalah probabilitas untuk setiap kelas *malware family* dari suatu masukan citra *malware*. Nilai probabilitas tertinggi akan diambil sebagai hasil prediksi klasifikasi. Berdasarkan Tabel 3.10, hasil klasifikasi yang diambil adalah kelas 3, kelas 3 merupakan kelas dengan probabilitas tertinggi dibandingkan kelas lainnya dengan nilai probabilitas 0.999210561.

Tabel 3.10 Prediksi Kelas Klasifikasi dengan *Softmax*

Kelas 0	Kelas 1	Kelas 2	Kelas 3	Kelas 4
0.000789439	9.03805E-17	2.84428E-15	0.999210561	3.0969E-15

Secara matematis, perhitungan fungsi *softmax* memungkinkan menghasilkan prediksi kelas target dengan nilai probabilitas yang sama, yang berarti keanggotaan dari suatu data masukan bisa masuk ke dalam beberapa kelas target sekaligus. Hal yang dapat dilakukan jika hal ini terjadi adalah melakukan penambahan nilai *epoch* untuk menambahkan jumlah pelatihan pada model atau mengganti nilai *hyperparameter* yang digunakan pada suatu model klasifikasi. Tetapi, masalah tersebut kemungkinan kecil terjadi karena perhitungan untuk keluaran probabilitas menghasilkan nilai dengan ketelitian yang tinggi atau perhitungan sampai beberapa angka di belakang koma.

Hasil perhitungan klasifikasi masih mengalami *error*. *Error* atau *loss* merupakan nilai yang menunjukkan seberapa jauh prediksi klasifikasi dengan nilai klasifikasi yang sebenarnya, perhitungan *error* akan menggunakan *loss function* yang terdapat pada Persamaan 2.11. Contoh perhitungan *error* dengan *loss function* sebagai berikut.

$$\begin{aligned}
 Loss &= -\log(\phi_i) \\
 &= -\log(0.000789439) \\
 &= 3.10268
 \end{aligned}$$

Selain itu, nilai *loss* lain akan dihitung dari nilai bobot dan bias pada *convolution layer* dan *dense layer*, perhitungan nilai *loss* menggunakan regularisasi L2 dengan Persamaan 2.12 dan Persamaan 2.13. Karena bias yang digunakan dalam perhitungan manual ini bernilai 0, maka nilai *loss* bias adalah 0.

Contoh perhitungan L2 untuk bobot pada *kernel* di *convolutional layer* adalah sebagai berikut. Konstanta regularisasi λ yang digunakan sebesar 0.001.

$$\begin{aligned}
 L_{2w} &= \lambda \sum_m w_m^2 \\
 &= 0.001 * (0.04963341^2 + 0.02579839^2 + \dots + -0.06364544^2) \\
 &= 0.001 * 0.016290 \\
 &= 0.00001629
 \end{aligned}$$

Contoh perhitungan L2 untuk bobot pada *kernel* di *dense layer* adalah sebagai berikut.

$$\begin{aligned}
 L_{2w} &= \lambda \sum_m w_m^2 \\
 &= 0.001 * (-0.04107868^2 + 0.02992102^2 + \dots + -0.07777447^2) \\
 &= 0.001 * 0.01621 \\
 &= 0.00001621
 \end{aligned}$$

Maka, nilai *loss* akhir didapat dari total penjumlahan dari semua nilai *loss* yang telah dihitung. Nilai *loss* ini akan digunakan pada tahap *backpropagation*.

$$\begin{aligned}
 Loss &= 3.10268 + 0.00001629 + 0.00001621 \\
 &= 3.1027125
 \end{aligned}$$

DAFTAR REFERENSI

- [1] M. Kalash, M. Rochan, N. Mohammed, N. Bruce, Y. Wang, and F. Iqbal, "A Deep Learning Framework for Malware Classification," *International Journal of Digital Crime and Forensics (IJDCF)*, vol. 12, no.1, pp. 90-108, March 2020. [Online], Available: <http://doi.org/10.4018/IJDCF.2020010105>.
- [2] B. Yadav and S. Tokekar, "Recent Innovations and Comparison of Deep Learning Techniques in Malware Classification: A Review," *International Journal Of Information Security Science*, vol. 9, no. 4, pp. 230-247, 2020. [Online], Available: <http://www.ijiss.org/ijiss/index.php/ijiss/article/view/852>.
- [3] A. Makandar and A. Patrot, "Trojan Malware Image Pattern Classification," *Proceedings of International Conference on Cognition and Recognition*, Springer Singapore, 2018. [Online], Available: https://doi.org/10.1007/978-981-10-5146-3_24.
- [4] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware Visualization for Fine-Grained Classification," *IEEE Access*, vol. 6, pp. 14510-14523, 2018. [Online], Available: <https://doi.org/10.1109/ACCESS.2018.2805301>.
- [5] M. Jain, W. Andreopoulos, and M. Stamp, "Convolutional neural networks and extreme learning machines for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 16, pp. 229-244, 2020. [Online], Available: <https://doi.org/10.1007/s11416-020-00354-y>. [Accessed: 27-Sept-2021].
- [6] J. Rasheed, A.A. Hameed, C. Djeddi, A. Jamil, and F. Al-Turjman, "A machine learning-based framework for diagnosis of COVID-19 from chest X-ray images," *Interdiscip Sci Comput Life Sci*, vol. 13, pp. 103-117, 2021. [Online], Available: <https://doi.org/10.1007/s12539-020-00403-6>.
- [7] A. Darwaish and F. Naït-Abdesselam, "RGB-based Android Malware Detection and Classification Using Convolutional Neural Network," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1-6, 2020. [Online], Available: <https://doi.org/10.1109/GLOBECOM42002.2020.9348206>.

DAFTAR REFERENSI

- [8] M. Stamp, M. Alazab, and A. Shalaginov, *Malware Analysis Using Artificial Intelligence and Deep Learning*, Berlin/Heidelberg, Germany: Springer, 2021.
- [9] O. Aslan and A.A.Yilmaz, "A New Malware Classification Framework Based on Deep Learning Algorithms," *IEEE Access*, vol. 9, pp. 87936-87951, 2021. [Online], Available: <https://doi.org/10.1109/ACCESS.2021.3089586>.
- [10] M. Alazab, S. Venkatraman, P. Watters, M. Alazab, and A. Alazab, "Cybercrime: The Case of Obfuscated Malware," *Global security, safety and sustainability & e-Democracy*, pp. 204-211, 2011. [Online], Available: https://doi.org/10.1007/978-3-642-33448-1_28.
- [11] J. Singh, J. Singh, "Challenges of Malware Analysis: Obfuscation Techniques," *International Journal of Information Security Science*, vol. 7, no. 3, pp. 100-110, 2018. [Online], Available: <http://50.87.218.19/ijiss/index.php/ijiss/article/view/327>.
- [12] L. Nataraj, S. Karthikeyan, G. Jacob, and B.S. Manjunath, "Malware images: visualization and automatic classification," *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, article 4, pp. 1-7, 2011. [Online], Available: <https://doi.org/10.1145/2016904.2016908>.
- [13] H. Abdi and L.J. Williams, "Principal component analysis," *WIREs Comp Stat*, vol. 2, no.4m, pp. 433-459, 2010. [Online]. Available: <https://doi.org/10.1002/wics.101>.
- [14] R.C. Gonzalez and R.E.Woods, *Digital Image Processing*, 3rd Edition, New Jersey:Pearson Prentice Hall, 2008.
- [15] Western Sydney University, "Colour modes explained". [Online], Available: https://www.westernsydney.edu.au/tld/home/how_to/how-to_resources/images_and_graphics/colour_modes. [Accessed: 16-Feb-2022].
- [16] R. Yamashita, M. Nishio1, R.K.G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights Imaging*, vol. 9, no. 4, pp. 611-629, 2018. [Online], Available: <https://doi.org/10.1007/s13244-018-0639-9>.
- [17] S. Indolia, A.K. Goswami, S.P. Mishra, and P. Asopa, "Conceptual Understanding of Convolutional Neural Network - A Deep Learning

- Approach,” *Procedia Computer Science*, vol. 132, pp. 679-688, 2018. [Online], Available: <https://doi.org/10.1016/j.procs.2018.05.069>.
- [18] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, ”Fundamental Concepts of Convolutional Neural Network,” *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, vol. 172, Springer, pp. 519-567, 2019. [Online], Available: https://doi.org/10.1007/978-3-030-32644-9_36.
- [19] P. Ligade, ”Why cautiously initializing deep neural networks matters?”, *Towards Data Science*, 18 April 2019. [Online], Available: <https://towardsdatascience.com/what-is-weight-initialization-in-neural-nets-and-why-it-matters-ec45398f99fa>. [Accessed: 21-Feb-2022].
- [20] A. G’eron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd Edition*, O’Reilly Media, Inc, 2019.
- [21] H.H. Aghdam and E.J. Heravi, *Guide to Convolutional Neural Networks*, 1st ed. Switzerland: Springer International Publishing AG, 2017.
- [22] L. Panneerselvam, ”Activation Functions and their Derivatives-A Quick & Complete Guide”, *Analytics Vidhya*, 14 April 2021. [Online], Available: <https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/>. [Accessed: 18-Feb-2022]
- [23] W. Di, A. Bhardwaj, and J. Wei, *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*, Packt Publishing, 2018.
- [24] H. Kinsley and D. Kukieła, *Neural Networks from Scratch in Python*, 1st ed. Harrison Kinsley, 2020.
- [25] OpenCV, ”ColorMaps in OpenCV”. [Online], Available: https://docs.opencv.org/4.x/d3/d50/group__imgproc__colormap.html. [Accessed: 19-Feb-2022]
- [26] Microsoft Security Intelligence, ”Malware Encyclopedia Description”. [Online], Available: <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description>. [Accessed: 5-Mar-2022]
- [27] Trend Micro, ”Threat Encyclopedia”. [Online], Available: <https://www.trendmicro.com/vinfo/us/threat-encyclopedia/>. [Accessed: 5-Mar-2022]