BAB 3 ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

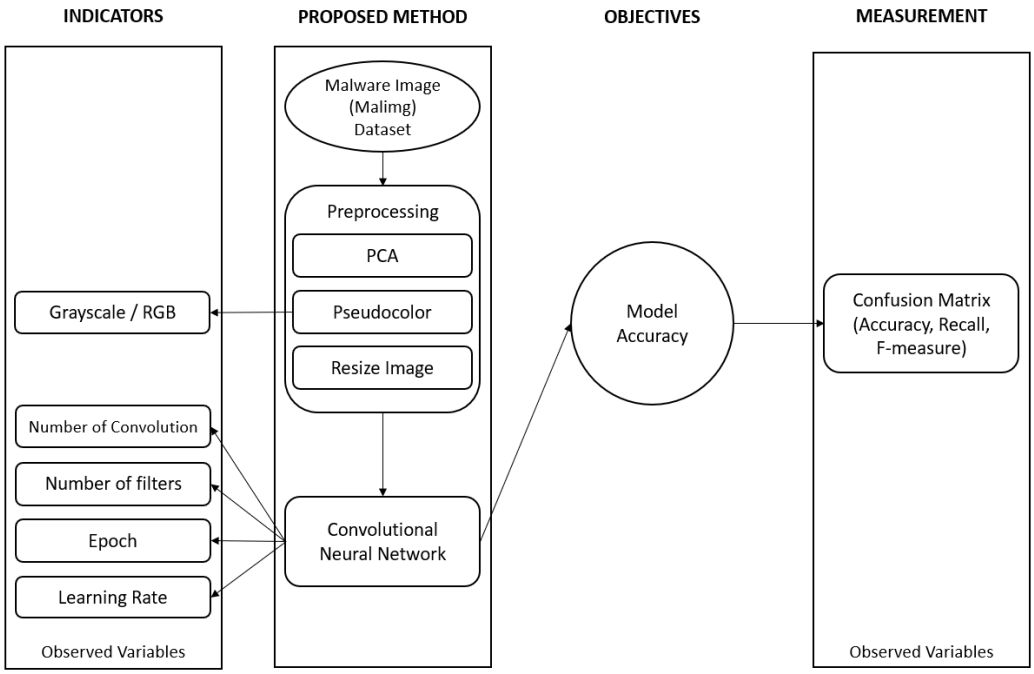
Pada Bab 1 telah dijelaskan bahwa banyak metode *machine learning* yang sudah dikembangkan untuk melakukan klasifikasi *malware* berbasiskan citra *grayscale*. Pada penelitian ini, metode Convolutional Neural Network dipilih karena menghasilkan akurasi yang tinggi dalam mengklasifikasikan citra, serta 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* [6]. 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 80:20 yang artinya 80% data belajar (*training*) dan 20% 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 awal 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 convolutions* : menentukan jumlah *convolution layer* yang digunakan dalam arsitektur Convolutional Neural Network. *Convolution layer* berperans 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 untuk menentukan jumlah operasi konvolusi yang tepat supaya model CNN dapat menghasilkan klasifikasi dengan akurasi yang tinggi.
2. *Number of filters* : *hyperparameter* pada *convolution layer* untuk menentukan jumlah *filter* atau *kernel* yang digunakan dalam suatu *convolution layer*. Seperti yang dijelaskan pada Sub Bab 2.1.5.1, semakin banyak *filter* yang dipakai dalam 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 ketika 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 sehingga menghasilkan model CNN yang konvergen atau *fit*, serta dapat melakukan klasifikasi dengan akurat.

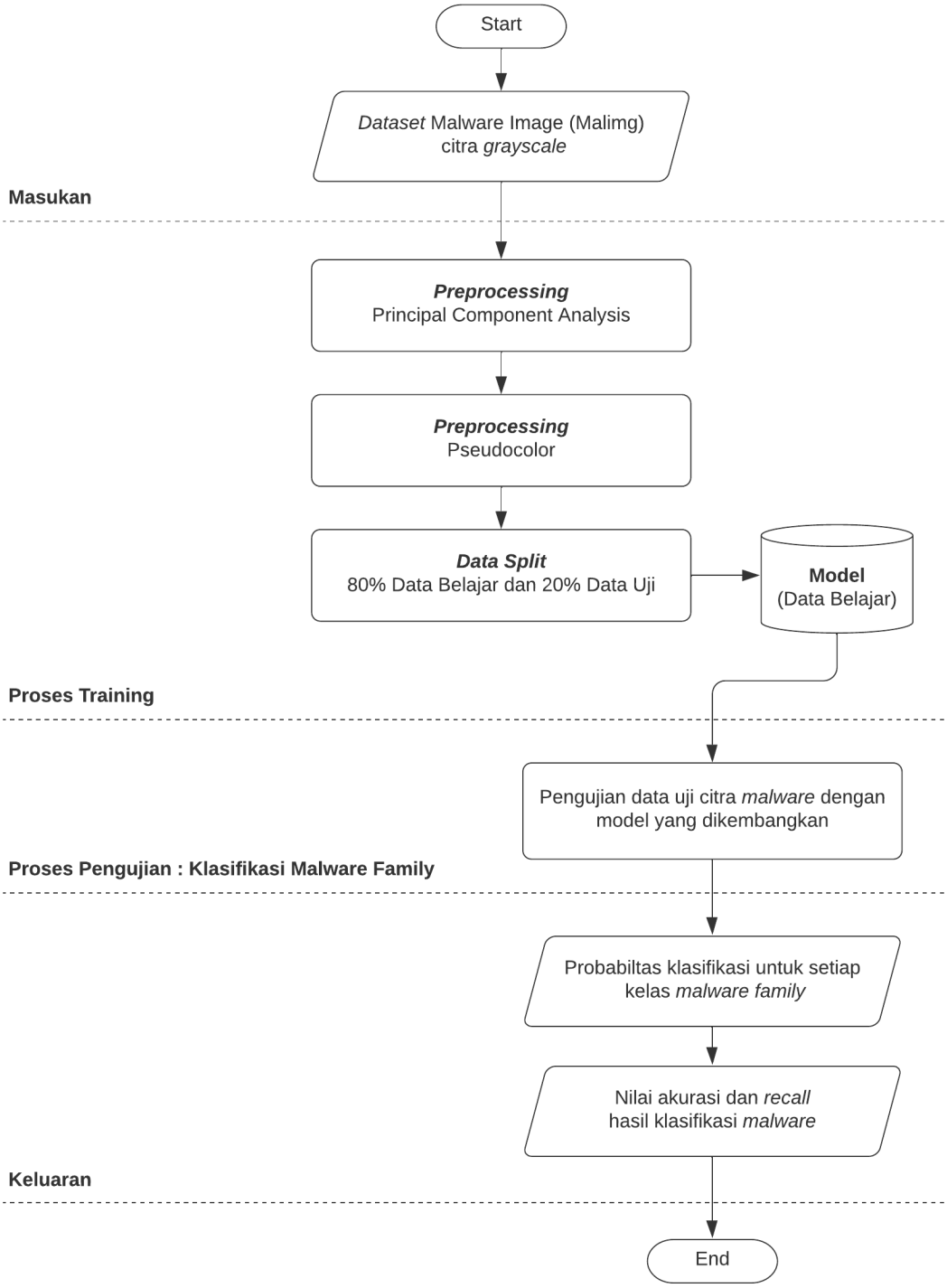
Proposed Method adalah bagian yang menjelaskan proses penelitian dari awal hingga akhir. Penelitian dimulai dari tahap *preprocessing*, yaitu metode Principal Component Analysis (PCA) yang digunakan untuk ekstraksi fitur dan teknik *pseudocolor* untuk memperoleh citra RGB dari citra *grayscale malware*. Lalu, baik citra *grayscale* dan citra RGB *malware* akan dilakukan proses *resizing* untuk menyeragamkan ukuran citra karena citra yang terdapat di dalam *dataset* memiliki dimensi ukuran yang berbeda-beda. Model Convolutional Neural Network kemudian akan dilatih dan diuji untuk mendapatkan hasil prediksi *malware family*.

Objectives (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*.

Measurement adalah proses pengukuran terhadap hal-hal yang terdapat pada bagian *Objectives*. Dalam penelitian ini, digunakan metode *confusion matrix* seperti yang dijelaskan pada Sub Bab 2.1.9 untuk pengukuran nilai akurasi, *recall*, dan *F-measure* untuk melakukan evaluasi terhadap model Convolutional Neural Network dan menunjukkan seberapa akurat model mengklasifikasikan *malware family* berdasarkan visualisasi citra *malware*.

3.3 Urutan Proses Global

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



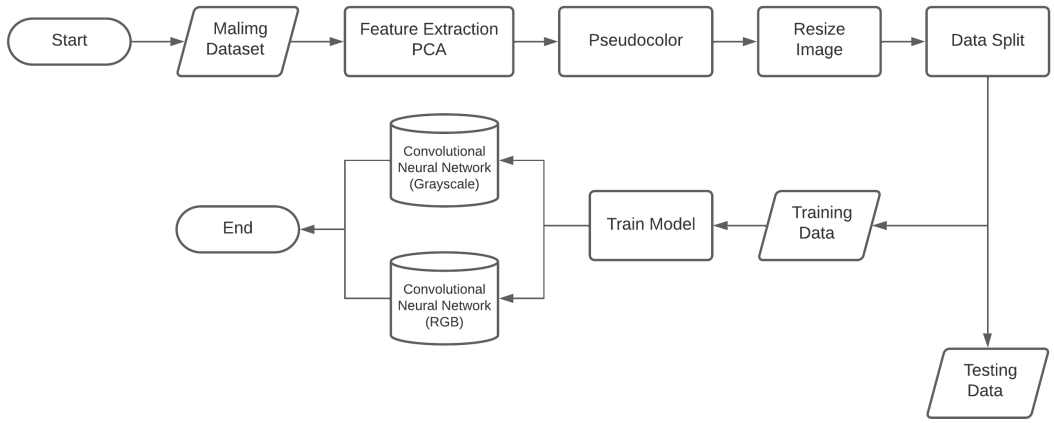
Gambar 3.2 *Flowchart* Urutan Proses Global

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

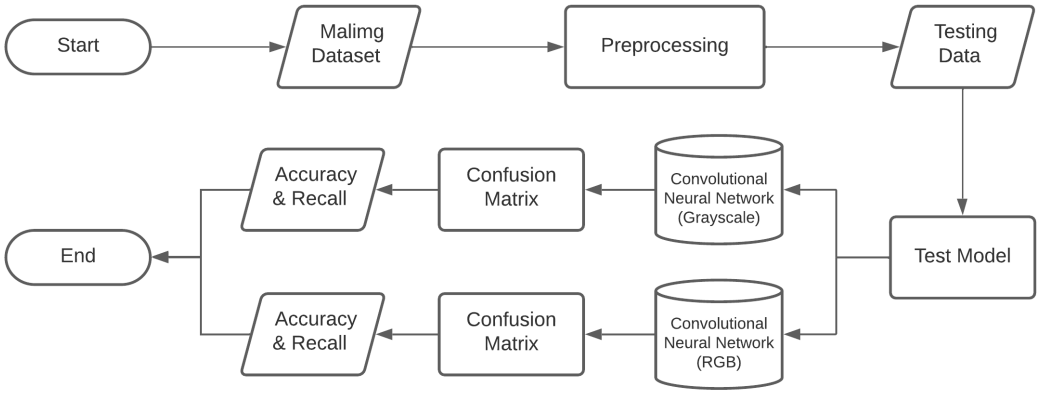
Berikut ini adalah uraian dari *flowchart* proses pelatihan (*training*) pada Gambar 3.3 yang dilakukan pada penelitian ini:

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 80% sebagai data belajar (*training*) dan 20% 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 20% 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 80:20 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. Allaple.A merupakan varian *worm* dalam *family* Allaple yang menginfeksi komputer-komputer yang terhubung dengan suatu *local area network* (LAN) dan melakukan penyerangan *denial-of-service* (DOS) pada situs web.
4. Allaple.L merupakan varian *worm* dalam *family* Allaple 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

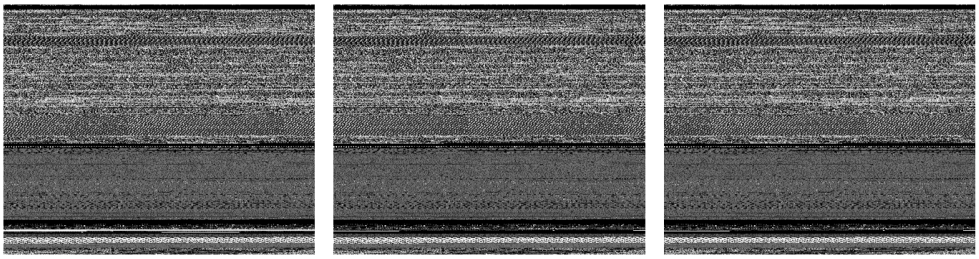
komputer yang terinfeksi.

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 mengeksploitasi 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

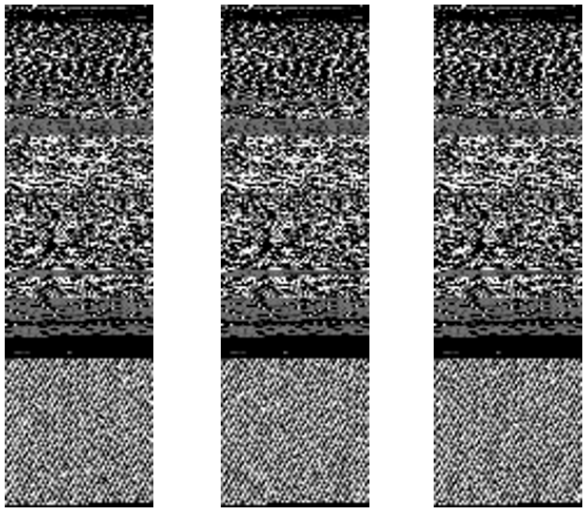
Visual Basic yang menyebar dengan menyisipkan kode pada suatu *file* atau program dalam komputer atau jaringan sehingga program-program yang terinfeksi tidak bisa berjalan dengan benar.

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.

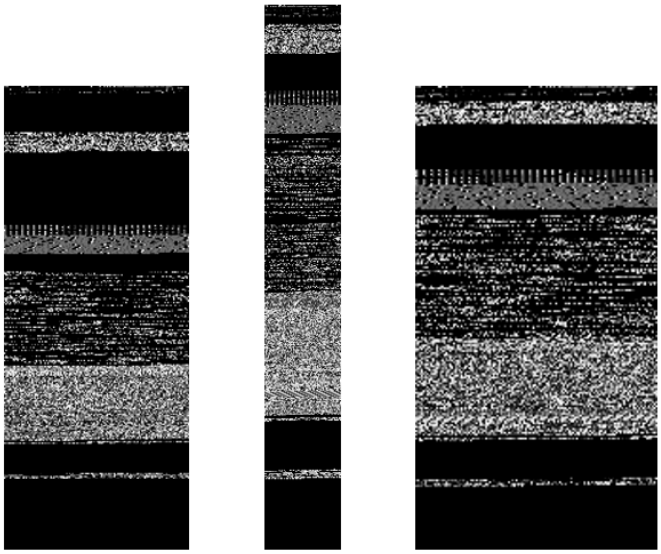
Berikut merupakan contoh *citra grayscale* dari beberapa *malware family* :



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 pelatihan pada model klasifikasi.

3.4.2.1 Principal Component Analysis

Pada tahap ini dilakukan ekstraksi fitur untuk mereduksi dimensi data dan 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 OpenCV dan Numpy.

Sebelum melakukan proses ekstraksi fitur, citra *grayscale malware* dimuat terlebih dahulu dengan menggunakan fungsi imread() dari pustaka OpenCV, parameter yang digunakan adalah *path* atau direktori *file* citra disimpan dan angka integer 0 untuk menentukan citra dimuat dalam mode *grayscale* atau 1 *channel* warna saja. Kode Python untuk memuat citra masukan dapat dilihat pada Gambar 3.8, serta citra masukan yang dimuat berupa matriks atau *array* berukuran 368×256.

```
[ ] image_path = '/dataset/malimg_paper_dataset_imgs/Alueron.gen!J/003b0beb511bc9b940a631c0a902f18d.png'
    image = cv2.imread(image_path, 0)
    image.shape

    (368, 256)
```

Gambar 3.8 Kode Python untuk Memuat Citra Masukan *Grayscale Malware*

Untuk keperluan visualisasi, matriks citra dikonversi menjadi *dataframe* dengan pustaka Pandas agar nilai-nilai *pixel* yang terdapat dalam citra dapat terlihat lebih jelas dan mudah dibaca. Ukuran panjang citra dibaca sebagai baris menjadi 368 baris, sedangkan ukuran lebar citra dibaca sebagai kolom menjadi 256 kolom. Gambar 3.9 merupakan *dataframe* yang berisikan nilai intensitas keabuan yang terdapat dalam setiap *pixel* pada citra *grayscale malware* yang dimuat sebelumnya dengan rentang nilai 0-255.

```
[ ] image_df = pd.DataFrame(image)
    image_df
```

	0	1	2	3	4	5	6	7	8	9	...	246	247	248	249	250	251	252	253	254	255
0	77	90	144	0	3	0	0	0	4	0	...	1	0	0	80	1	0	0	0	0	0
1	237	17	0	0	0	16	0	0	0	128	...	0	96	46	100	97	116	101	0	0	0
2	0	48	1	0	0	128	1	0	0	48	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
363	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
364	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
365	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
366	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
367	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

368 rows × 256 columns

Gambar 3.9 Nilai *Pixel* Awal dalam Citra *Grayscale Malware*

Tahap pertama pada Principal Component Analysis adalah melakukan proses normalisasi data atau nilai *pixel* dengan fungsi *Z-score*. Fungsi *Z-score* dihitung menggunakan Persamaan 2.1 dengan mengurangi nilai data (*pixel*) dengan rata-rata data, kemudian dibagi dengan standar deviasi berdasarkan data yang ada. Perhitungan normalisasi *Z-score* dapat menangani perbedaan yang cukup signifikan antara nilai maksimal dan nilai minimal sehingga memiliki rentang nilai yang sama antar kolom atau variabel. Kode Python dan hasil normalisasi *Z-score* pada citra *grayscale malware* dapat dilihat pada Gambar 3.10 dan Gambar 3.11 sebagai berikut.

```
[ ]  image_mean = image.mean()
     image_std = image.std()

     normalized_image = (image - image_mean) / image_std
     normalized_image_df = pd.DataFrame(normalized_image)
     normalized_image_df
```

Gambar 3.10 Kode Python untuk Normalisasi *Z-score* pada Citra *Grayscale Malware*

	0	1	2	3	4	5	6	7	8	9	...	246	247
0	-0.336383	-0.179260	0.473402	-1.267031	-1.230772	-1.267031	-1.267031	-1.267031	-1.218685	-1.267031	...	-1.254944	-1.267031
1	1.597431	-1.061563	-1.267031	-1.267031	-1.267031	-1.073649	-1.267031	-1.267031	-1.267031	0.280020	...	-1.267031	-0.106742
2	-1.267031	-0.686886	-1.254944	-1.267031	-1.267031	0.280020	-1.254944	-1.267031	-1.267031	-0.686886	...	-1.267031	-1.267031
3	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
4	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
...	...	...	...	...	...	...	...	...	...	...	...	...	...
363	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
364	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
365	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
366	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031
367	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	-1.267031	...	-1.267031	-1.267031

368 rows × 256 columns

Gambar 3.11 Hasil Normalisasi *Z-score* pada Citra *Grayscale Malware*

Selanjutnya, dilakukan perhitungan matriks kovarians untuk melihat besar hubungan korelasi antar variabel data berdasarkan citra yang sudah dinormalisasi. Gambar 3.12 dan Gambar 3.13 merupakan kode Python dan hasil perhitungan matriks kovarians yang memperoleh matriks berukuran 256 × 256.

```
[ ]  cov_matrix = np.cov(normalized_image.T)
     cov_matrix_df = pd.DataFrame(cov_matrix)
     print('Covariance matrix \n')
     cov_matrix_df
```

Gambar 3.12 Kode Python untuk Proses Perhitungan Matriks Kovarians

Covariance matrix

	0	1	2	3	4	5	6	7	8	9	...	246	247	248	249	250
0	1.042467	0.313805	0.358239	0.380714	0.384080	0.334762	0.317642	0.358833	0.335075	0.379482	...	0.327768	0.396335	0.376861	0.370150	0.333739
1	0.313805	0.971323	0.244331	0.303566	0.286510	0.306782	0.366859	0.318651	0.280513	0.313994	...	0.306916	0.299924	0.272563	0.361420	0.309481
2	0.358239	0.244331	0.988154	0.345014	0.386561	0.394129	0.388040	0.343875	0.360592	0.354247	...	0.322685	0.299946	0.340698	0.307084	0.260132
3	0.380714	0.303566	0.345014	1.045528	0.383621	0.393640	0.296556	0.323011	0.361528	0.292295	...	0.318510	0.325804	0.347712	0.392983	0.256366
4	0.384080	0.286510	0.386561	0.383621	1.066068	0.362080	0.338490	0.337213	0.344271	0.314772	...	0.310279	0.289277	0.382873	0.366393	0.356262
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
251	0.348975	0.405845	0.328596	0.363342	0.375360	0.299766	0.347015	0.297313	0.277913	0.405376	...	0.406700	0.404475	0.417326	0.347355	0.372845
252	0.316275	0.310218	0.277729	0.336122	0.274823	0.271309	0.260432	0.335238	0.317969	0.365003	...	0.333621	0.271979	0.306569	0.283112	0.294675
253	0.319857	0.267743	0.306581	0.401163	0.339718	0.307300	0.368505	0.324765	0.337319	0.265528	...	0.382550	0.383993	0.320917	0.390721	0.354177
254	0.299012	0.275638	0.304704	0.333754	0.335595	0.306012	0.377777	0.299617	0.312364	0.309417	...	0.308190	0.358316	0.322188	0.365339	0.320794
255	0.402217	0.384493	0.239120	0.388240	0.335693	0.339185	0.387452	0.337838	0.359007	0.352013	...	0.375958	0.436887	0.320732	0.395701	0.333346

256 rows × 256 columns

Gambar 3.13 Hasil Perhitungan Matriks Kovarians

Matriks kovarians kemudian digunakan untuk menghitung *eigenvector* dengan Persamaan 2.2 dan *eigenvalue* dengan Persamaan 2.3. *Eigenvector*

merupakan vektor yang menunjukkan arah untuk menangkap sebaran informasi atau *variance* terhadap matriks kovarians, sementara *eigenvalue* merupakan koefisien besar informasi dari suatu *eigenvector*. Kode Python untuk mendapatkan *eigenvector* dan *eigenvalue* dari matriks kovarians dapat dilihat pada Gambar 3.14. Gambar 3.15 dan Gambar 3.16 merupakan hasil *eigenvector* dan *eigenvalue* yang diperoleh terhadap matriks kovarians.

```
[ ]  eigenvalue, eigenvector = np.linalg.eig(cov_matrix)
```

Gambar 3.14 Kode Python untuk Menghitung *Eigenvector* dan *Eigenvalue*

```
[ ] Eigenvector
           0         1         2         3         4         5         6    \
0   -0.061636  0.005282  0.109485  0.113827  0.010803  0.038448 -0.040746
1   -0.056405  0.019850  0.031933 -0.140935 -0.042879 -0.078179 -0.102131
2   -0.060520  0.107833  0.001960  0.122902  0.028296 -0.009758 -0.002166
3   -0.062725  0.018104 -0.049958  0.116442  0.113908  0.098628  0.011053
4   -0.063825  0.031078 -0.068213  0.094573 -0.033016 -0.077818 -0.068859
..        ...       ...       ...       ...       ...       ...       ...
251 -0.065588 -0.096599 -0.008111 -0.002082 -0.015551  0.083380 -0.139511
252 -0.058193 -0.000789  0.003314 -0.060831  0.083327 -0.023833 -0.022260
253 -0.063822 -0.084657 -0.004682 -0.022656 -0.004566  0.091897 -0.055793
254 -0.062837 -0.042119 -0.068908 -0.048840  0.118831 -0.025277  0.003437
255 -0.066098 -0.019805  0.045726 -0.081223 -0.027260  0.189001 -0.043047

           7         8         9    ...       246       247       248  \
0    0.146190 -0.129730 -0.019045  ... -0.003261 -0.001519  0.005106
1    0.043380 -0.089790 -0.058641  ...  0.069472 -0.057619  0.054990
2    0.085287  0.025540 -0.176824  ...  0.084137 -0.048227 -0.067283
3    0.019660 -0.021575  0.028126  ... -0.051703  0.006681 -0.116133
4    0.035107 -0.082714 -0.056275  ...  0.075490 -0.089296 -0.021897
..        ...       ...       ...  ...       ...       ...       ...
251  0.004320 -0.024788 -0.086783  ... -0.041180  0.016071  0.058785
252  0.104326  0.033748  0.025548  ...  0.087708 -0.060833  0.026101
253 -0.000326  0.060378 -0.028289  ...  0.042854 -0.002469 -0.069776
254 -0.084789 -0.003751  0.054822  ...  0.053474  0.013555  0.039123
255  0.040301 -0.000810  0.012349  ...  0.090668  0.082901  0.036462

           249       250       251       252       253       254       255
0    0.006169 -0.050349 -0.013863 -0.011310  0.002743 -0.002352 -0.001852
1   -0.027958 -0.046540  0.094877 -0.045983  0.023132 -0.002269 -0.000113
2    0.016812  0.013242  0.039822 -0.062300 -0.111724 -0.047048  0.069537
3    0.151480 -0.019890  0.013831 -0.052358  0.018860 -0.013296  0.008598
4   -0.009002  0.061576 -0.074687  0.030314 -0.042244 -0.062310  0.165296
..        ...       ...       ...       ...       ...       ...       ...
251 -0.059463 -0.008744 -0.042534 -0.053140  0.009221  0.099266  0.047424
252  0.109918  0.008117 -0.036032  0.085739 -0.133585 -0.061621 -0.046411
253 -0.026665  0.069898  0.024859 -0.000749 -0.075946  0.070718 -0.064955
254 -0.001114 -0.082714 -0.009304  0.073392  0.074615 -0.021157  0.099755
255  0.083794 -0.011804  0.060735 -0.053784 -0.036961  0.061385 -0.083112

[256 rows x 256 columns]
```

Gambar 3.15 Nilai *Eigenvector* Terhadap Matriks Kovarians

```
Eigenvalue
              0
0    86.091035
1     2.535300
2     2.383078
3     2.304254
4     2.277364
..         ...
251   0.064251
252   0.075494
253   0.137076
254   0.102036
255   0.105569

[256 rows x 1 columns]
```

Gambar 3.16 Nilai *Eigenvalue* Terhadap Matriks Kovarians

Nilai *threshold* kompresi yang digunakan adalah 0.95 atau 95% untuk tingkat reduksi dimensi data. Berdasarkan nilai *threshold* yang ditentukan, maka 95% dari keseluruhan nilai *eigenvalue* akan dipertahankan sehingga akan mereduksi dimensi data yang awalnya terdiri 256 komponen menjadi 243 komponen. Gambar 3.17 merupakan kode Python yang digunakan untuk proses pemilihan jumlah komponen (*principal component*) yang akan dipertahankan.

```
[ ]  compression = 0.95
     k_component = int(len(eigenvalue)*compression)
     k_component

     243
```

Gambar 3.17 Kode Python untuk Pemilihan Jumlah Komponen yang Dipertahankan

Eigenvector akan dipilih berdasarkan jumlah *principal component* yang diperoleh sebelumnya. *Eigenvector* yang terpilih akan digunakan sebagai *feature vector*. Kode Python untuk memilih *eigenvector* dapat dilihat pada Gambar 3.18. Sedangkan, Gambar 3.19 merupakan hasil matriks *feature vector* berdasarkan *eigenvector* yang terpilih.

```
[ ]  feature_vector = eigenvector[:,:k_component]
```

Gambar 3.18 Kode Python untuk Pemilihan *Eigenvector* sebagai *Feature Vector*

```
[ ] New Feature Vector
           0         1         2         3         4         5         6    \
0   -0.061636  0.005282  0.109485  0.113827  0.010803  0.038448 -0.040746
1   -0.056405  0.019850  0.031933 -0.140935 -0.042879 -0.078179 -0.102131
2   -0.060520  0.107833  0.001960  0.122902  0.028296 -0.009758 -0.002166
3   -0.062725  0.018104 -0.049958  0.116442  0.113908  0.098628  0.011053
4   -0.063825  0.031078 -0.068213  0.094573 -0.033016 -0.077818 -0.068859
..        ...       ...       ...       ...       ...       ...       ...
251 -0.065588 -0.096599 -0.008111 -0.002082 -0.015551  0.083380 -0.139511
252 -0.058193 -0.000789  0.003314 -0.060831  0.083327 -0.023833 -0.022260
253 -0.063822 -0.084657 -0.004682 -0.022656 -0.004566  0.091897 -0.055793
254 -0.062837 -0.042119 -0.068908 -0.048840  0.118831 -0.025277  0.003437
255 -0.066098 -0.019805  0.045726 -0.081223 -0.027260  0.189001 -0.043047

           7         8         9    ...       233       234       235  \
0    0.146190 -0.129730 -0.019045  ...  0.116351 -0.086348 -0.058826
1    0.043380 -0.089790 -0.058641  ... -0.063709  0.029817 -0.004650
2    0.085287  0.025540 -0.176824  ... -0.027485 -0.075549  0.081352
3    0.019660 -0.021575  0.028126  ...  0.056212 -0.077892  0.088145
4    0.035107 -0.082714 -0.056275  ...  0.061921  0.029063 -0.033663
..        ...       ...       ...  ...       ...       ...       ...
251  0.004320 -0.024788 -0.086783  ...  0.045659 -0.051564 -0.061637
252  0.104326  0.033748  0.025548  ... -0.019984  0.022835  0.010635
253 -0.000326  0.060378 -0.028289  ... -0.146195  0.054408 -0.042138
254 -0.084789 -0.003751  0.054822  ... -0.155014  0.083710 -0.091982
255  0.040301 -0.000810  0.012349  ...  0.030981 -0.022170  0.021878

           236       237       238       239       240       241       242
0    0.032941 -0.018443 -0.069326 -0.103152  0.074245 -0.048755 -0.045541
1   -0.042763  0.058179  0.047463  0.039615 -0.055360 -0.055932 -0.002400
2   -0.049309 -0.039675 -0.001310  0.026094 -0.057484 -0.002527  0.026502
3   -0.167103 -0.066632  0.025531  0.049311  0.009483  0.065832  0.020558
4   -0.060686 -0.004247  0.080069 -0.015537 -0.020504 -0.062885 -0.077527
..        ...       ...       ...       ...       ...       ...       ...
251 -0.016986 -0.093916 -0.024851 -0.002289 -0.018851  0.039385 -0.062580
252  0.010799  0.035813  0.021014 -0.076878 -0.018822  0.004306  0.036123
253  0.056825  0.107888 -0.181221 -0.131587 -0.044273 -0.005382 -0.006341
254  0.077179  0.011922 -0.001370 -0.008367  0.095357 -0.065137 -0.019482
255 -0.021041  0.052797  0.010712  0.072029 -0.013554 -0.032285 -0.062065

[256 rows x 243 columns]
```

Gambar 3.19 Matriks *Feature Vector*

Setelah *feature vector* diperoleh, maka dilakukan proses transformasi data citra awal berdasarkan *feature vector*. Proses transformasi data menggunakan Persamaan 2.4 dengan melakukan perkalian *dot* antara matriks citra hasil normalisasi terhadap matriks *feature vector* yang memuat *eigenvector* dengan dimensi yang lebih rendah. Dengan transformasi data terjadi reduksi dimensi, hal ini dilihat dari hasil transformasi data memperoleh matriks berukuran 368×243 yang mempunyai dimensi yang lebih rendah dibandingkan matriks data awal dengan dimensi 368×256. Kode Python terkait perhitungan transformasi data dapat dilihat pada Gambar 3.20, sementara Gambar 3.21 merupakan keluaran untuk hasil transformasi data.

```
[ ]  transformed_image = normalized_image.dot(feature_vector)
```

Gambar 3.20 Kode Python untuk Perhitungan Transformasi Data

```
[ ] Transformed data
           0         1         2         3         4         5         6    \
0    14.831443  2.629629 -0.135134  0.209696  0.659734  1.622487 -0.252988
1    18.026545 -0.633940  1.277016  0.214149 -0.138521  0.137894 -0.477662
2    18.298381  1.740958 -0.032010 -0.562273  0.379915  0.350221 -0.340514
3    20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
4    20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
..         ...       ...       ...       ...       ...       ...       ...
363  20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
364  20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
365  20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
366  20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452
367  20.258747 -0.136094  0.014452 -0.012176 -0.010577 -0.003706  0.106452

          7         8         9    ...       233       234       235  \
0    1.175380  0.112588 -0.714245  ... -0.531302 -0.477269 -0.090234
1    0.247211 -0.097320  0.020497  ...  0.039324 -0.135372  0.138499
2   -0.104068  0.116959 -0.079130  ...  0.157008  0.096772 -0.807600
3   -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
4   -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
..        ...       ...       ...  ...       ...       ...       ...
363 -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
364 -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
365 -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
366 -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229
367 -0.094126 -0.074714  0.012976  ... -0.020286  0.012758 -0.045229

          236       237       238       239       240       241       242
0    0.955763  0.173412  0.228454  0.973296  0.332778  0.333209 -0.280776
1   -0.076311  0.204265  0.555751 -0.277339  0.288228 -0.035800  0.107285
2    0.251265  0.338631  0.045458  0.264628  0.406515  0.358141  0.239642
3   -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
4   -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
..        ...       ...       ...       ...       ...       ...       ...
363 -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
364 -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
365 -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
366 -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006
367 -0.014720  0.066131  0.007025  0.000582 -0.009398 -0.027950  0.003006

[368 rows x 243 columns]
```

Gambar 3.21 Matriks Hasil Transformasi Data

Matriks hasil transformasi data belum bisa diproses sebagai data citra sehingga dimensi ukuran matriks perlu dikembalikan menjadi ukuran citra awal dengan melakukan proses invers data. Untuk mengembalikan ke dimensi citra awal menggunakan Persamaan 2.4 dengan mengalikan matriks hasil transformasi data dengan *transpose* dari *feature vector*. Kode Python untuk melakukan proses invers terhadap data yang sudah ditransformasikan dapat dilihat pada Gambar 3.22. Gambar 3.23 merupakan hasil invers data dan dihasilkan matriks berukuran 368×256, sama seperti ukuran citra awal.

```
[ ]  inversed_image = np.dot(transformed_image, feature_vector.T)
```

Gambar 3.22 Kode Python untuk Proses Invers Data

```
[ ] Inversed data
          0         1         2         3         4         5         6    \
0   -0.346426 -0.209332  0.258699 -1.305511 -1.264310 -1.212577 -1.124198
1    1.572258 -1.150223 -1.289642 -1.113217 -1.268271 -1.041249 -1.215717
2   -1.283446 -0.659224 -1.233494 -1.289353 -1.315916  0.280619 -1.268550
3   -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
4   -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
..        ...       ...       ...       ...       ...       ...       ...
363 -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
364 -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
365 -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
366 -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686
367 -1.265386 -1.263614 -1.261136 -1.260014 -1.270106 -1.280166 -1.272686

          7         8         9    ...       246       247       248  \
0   -1.160936 -1.170328 -1.116914  ... -1.174130 -1.370124 -1.192052
1   -1.118348 -1.165754  0.209446  ... -1.168171 -0.160537 -0.662419
2   -1.225246 -1.141325 -0.748963  ... -1.249982 -1.231583 -1.196089
3   -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
4   -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
..        ...       ...       ...  ...       ...       ...       ...
363 -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
364 -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
365 -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
366 -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423
367 -1.265430 -1.283646 -1.255800  ... -1.270011 -1.250360 -1.263423

          249       250       251       252       253       254       255
0   -0.438478 -1.294143 -1.192952 -1.563298 -1.322713 -1.171623 -1.394608
1   -0.064205 -0.209321  0.137704 -0.122640 -1.378758 -1.291312 -1.329148
2   -1.231810 -1.273590 -1.360274 -1.248793 -1.359841 -1.292757 -1.182922
3   -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
4   -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
..        ...       ...       ...       ...       ...       ...       ...
363 -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
364 -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
365 -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
366 -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793
367 -1.270752 -1.258862 -1.281118 -1.274272 -1.267714 -1.274301 -1.268793

[368 rows x 256 columns]
```

Gambar 3.23 Matriks Hasil Invers Data

Data yang dihasilkan proses invers masih berupa nilai-nilai *Z-score*. Oleh karena itu, proses denormalisasi data perlu dilakukan untuk mengubah nilai *Z-score* menjadi nilai intensitas keabuan pada citra yang memiliki rentang nilai 0-255. Secara matematis, proses denormalisasi menggunakan Persamaan 2.1 untuk menghitung nilai *pixel* awal. Nilai *Z-score* dikalikan dengan standar deviasi, lalu ditambahkan dengan rata-rata atau *mean*. Kode Python untuk melakukan proses denormalisasi data dan hasil akhir keluaran dapat dilihat pada Gambar 3.24 dan 3.25 sebagai berikut.

```
[ ]  compressed_image = (inversed_image * image_std) + image_mean
     compressed_image_df = pd.DataFrame(compressed_image)
     compressed_image_df
```

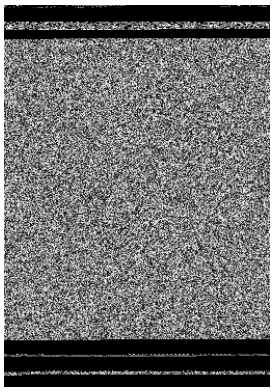
Gambar 3.24 Kode Python untuk Proses Denormalisasi Data

	0	1	2	3	4	5	6	7	8	9	...	246
0	76.169072	87.511921	126.235937	-3.183772	0.225095	4.505402	11.817728	8.778067	8.001008	12.420369	...	7.686377
1	234.917270	9.664428	-1.870792	12.726271	-0.102624	18.680719	4.245577	12.301708	8.379447	122.160854	...	8.179421
2	-1.358198	50.288747	2.774761	-1.846943	-4.044690	128.049539	-0.125758	3.457157	10.400650	42.863922	...	1.410564
3	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
4	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
...	...	...	...	...	...	...	...	...	...	...	...	...
363	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
364	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
365	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
366	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607
367	0.136088	0.282704	0.487731	0.580506	-0.254474	-1.086782	-0.467928	0.132388	-1.374765	0.929230	...	-0.246607

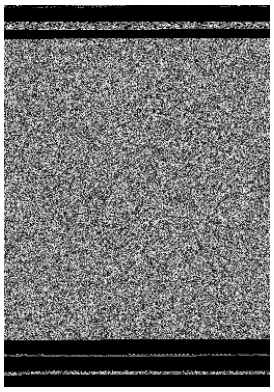
368 rows × 256 columns

Gambar 3.25 Hasil Keluaran setelah Denormalisasi Data

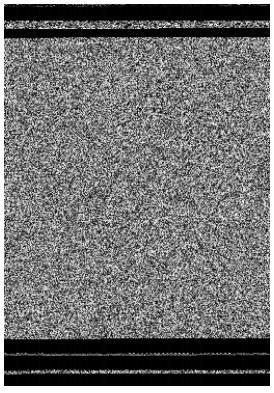
Gambar 3.26 merupakan contoh beberapa citra *grayscale malware family* Alueron.gen!J setelah dilakukan proses ekstraksi fitur dengan PCA.



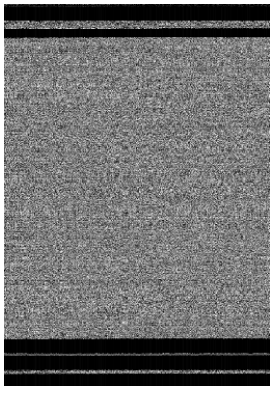
(a) Citra *Grayscale*



(b) Kompresi 95%



(c) Kompresi 50%



(d) Kompresi 10%

Gambar 3.26 Citra *Grayscale Malware* setelah PCA dengan Berbagai Tingkat Kompresi

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)
```

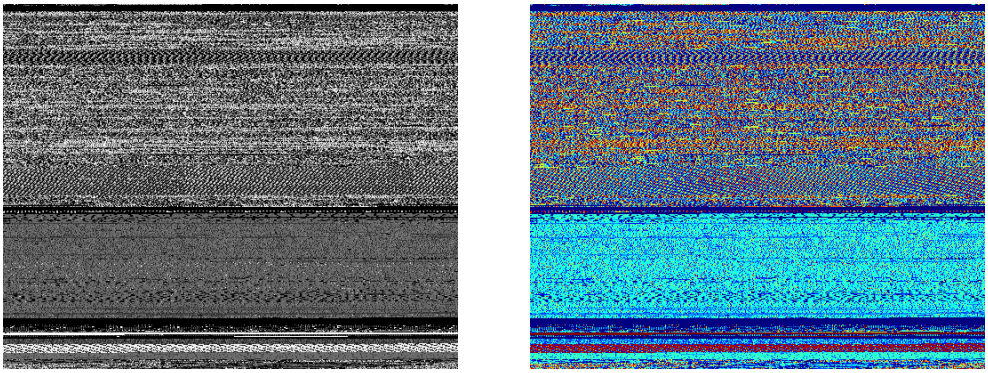
Gambar 3.27 Kode Python untuk *Pseudocolor*

Gambar 3.27 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*. Terdapat 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 3.28. 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 3.28 Skala Warna COLORMAP_JET

Gambar 3.29 merupakan perbandingan antara citra *grayscale* dan citra RGB pada kelas *malware family* Adialer.C.



(a) Citra *Grayscale* (b) Citra RGB

Gambar 3.29 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 berpindah secara satu dimensi, pergerakan *kernel* dilakukan 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×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.

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

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 dikalkulasikan 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.

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

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.

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

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.

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

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

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

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

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