

VERIFIKASI KEMIRIPAN WAJAH MENGGUNAKAN *DEEP LEARNING* DENGAN ARSITEKTUR JARINGAN SIAMESE

SKRIPSI



Oleh:

**HAIRUL IMAM
1310520075**

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS TEKNIK DAN KESEHATAN
UNIVERSITAS BUMIGORA
MATARAM
2019**

VERIFIKASI KEMIRIPAN WAJAH MENGGUNAKAN *DEEP LEARNING* DENGAN ARSITEKTUR JARINGAN *SIAMESE*

SKRIPSI



Diajukan Sebagai Salah Satu Syarat untuk Memenuhi Kebulatan Studi
Jenjang Strata Satu (S1) Program Studi Ilmu Komputer
Pada Universitas Bumigora

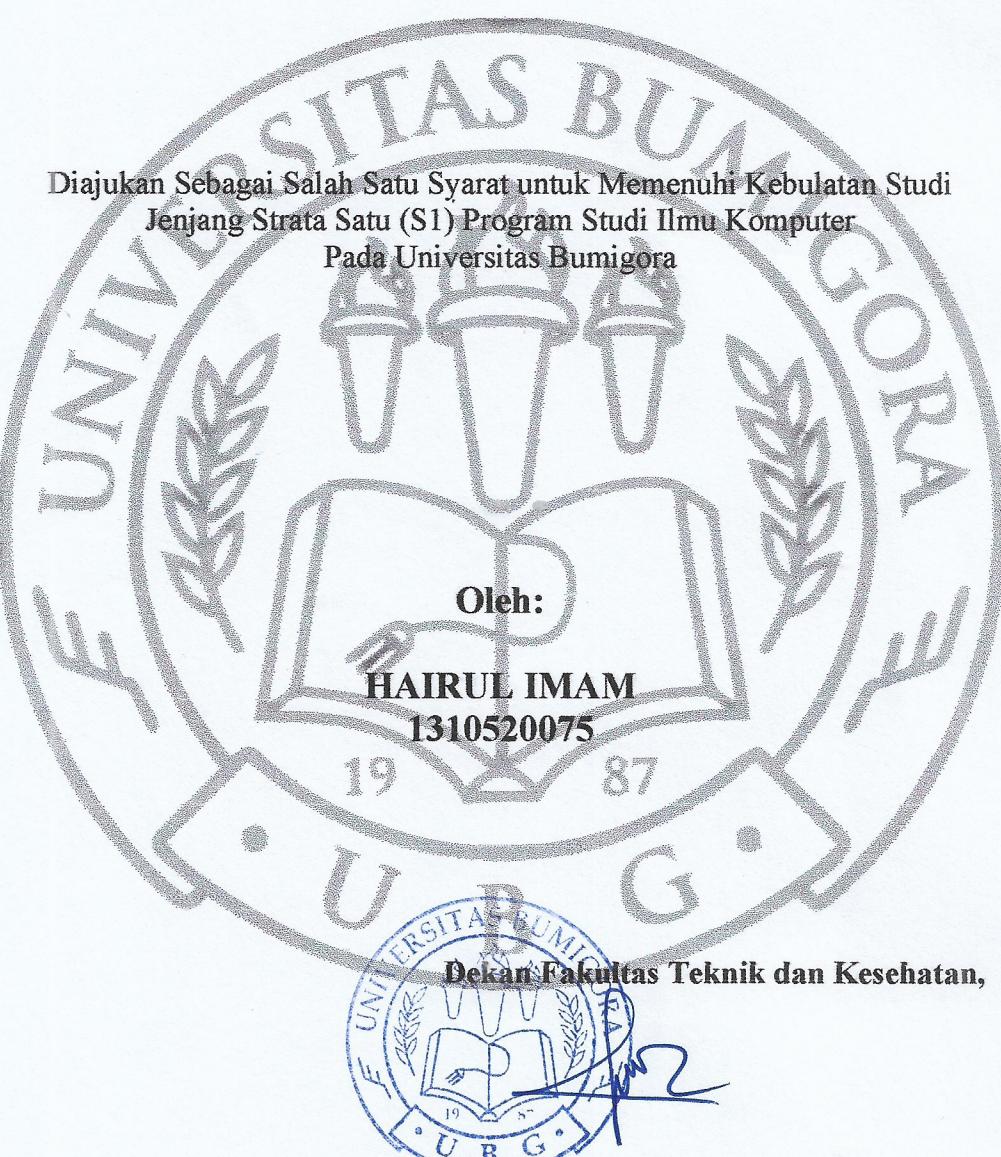
Oleh:

**HAIRUL IMAM
1310520075**

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS TEKNIK DAN KESEHATAN
UNIVERSITAS BUMIGORA
MATARAM
2019**

**VERIFIKASI KEMIRIPAN WAJAH MENGGUNAKAN DEEP
LEARNING DENGAN ARSITEKTUR JARINGAN SIAMESE**

SKRIPSI





UNIVERSITAS BUMIGORA

FAKULTAS TEKNIK DAN KESEHATAN
PROGRAM STUDI ILMU KOMPUTER

Jln. Ismail Marzuki, Cakranegara, Mataram

Telp/fax (0370)-638369 | Whatsapp 0859-3615-9726 | Email: univ.bumigora@gmail.com
www.universitasbumigora.ac.id

SKRIPSI

JUDUL : Verifikasi Kemiripan Wajah Menggunakan *Deep Learning* Dengan Arsitektur Jaringan *Siamese*

NAMA : HAIRUL IMAM

NIM : 1310520075

NPM : 13.8.349.74.75.0.5.0075

PROGRAM STUDI : Ilmu Komputer

JENJANG : Strata Satu (S1)

DIUJIKAN : 07 Agustus 2019

Menyetujui,

Jian Budiarto, S.T., M.Eng
Pembimbing I

Kartarina, S.Kom., M.Kom
Pembimbing II

Tanggal Menyetujui : 12/7/19 Tanggal Menyetujui : 12/21/2019

Telah diterima dan disetujui sebagai salah satu syarat untuk memperoleh
Gelar Akademik Sarjana Komputer (S.Kom)

Mengetahui :

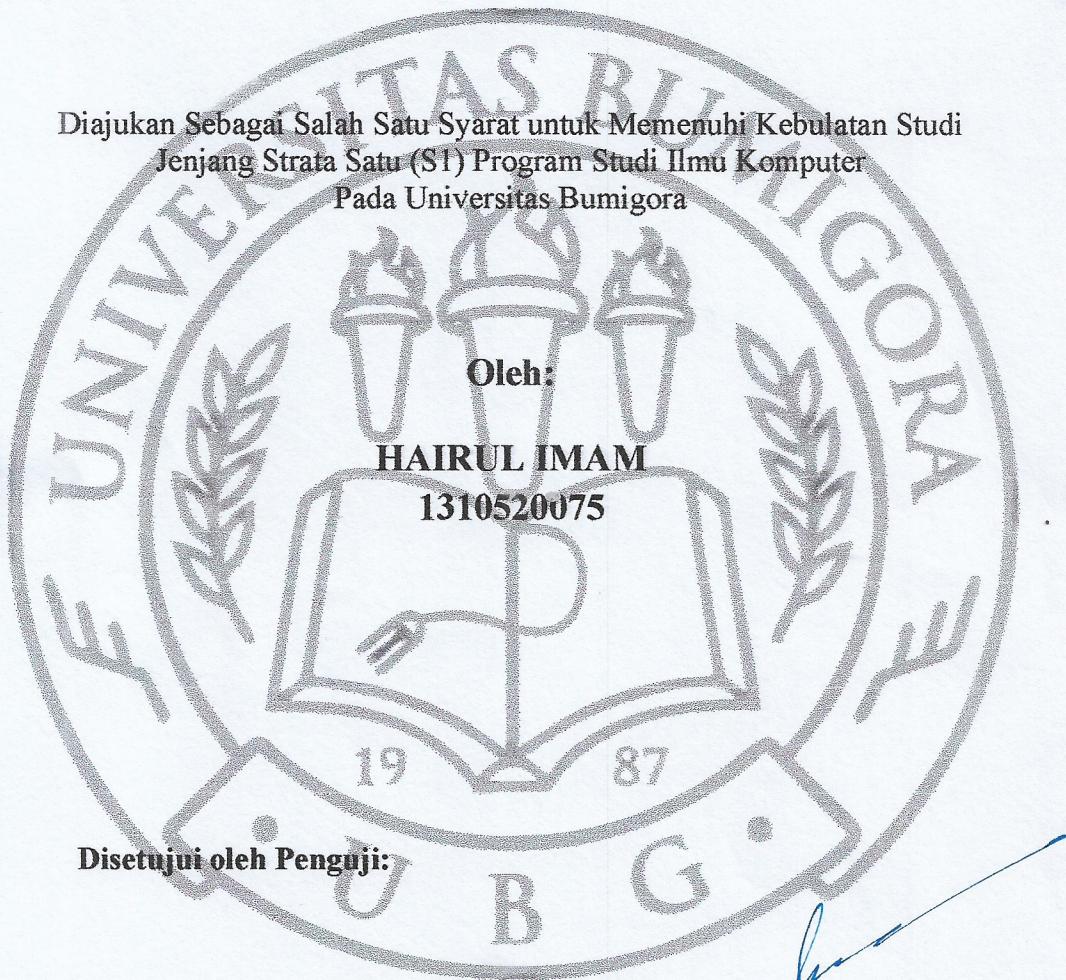
Heroe Santoso, M.Kom.
Ketua Program Studi S1 Ilmu Komputer

Tanggal Mengetahui :

VERIFIKASI KEMIRIPAN WAJAH MENGGUNAKAN *DEEP LEARNING* DENGAN ARSITEKTUR JARINGAN SIAMESE

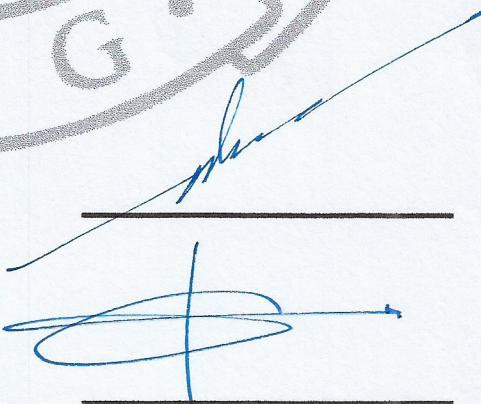
LEMBAR PENGESAHAN PENGUJI

Diajukan Sebagai Salah Satu Syarat untuk Memenuhi Kebutuhan Studi
Jenjang Strata Satu (S1) Program Studi Ilmu Komputer
Pada Universitas Bumigora

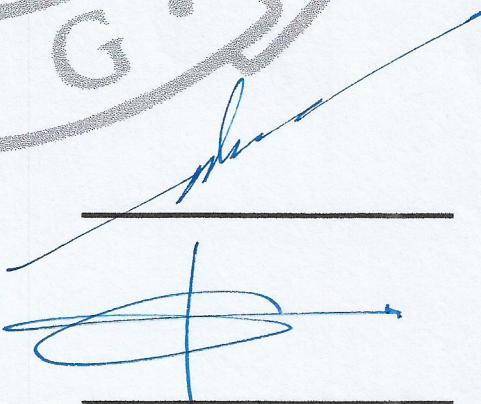


Disetujui oleh Penguji:

1. Adam Bachtiar, S.Kom., M.MT
15.6.231



2. Khurniawan Eko S. M.Eng
18.6.339



KATA PENGANTAR

Dengan nama Allah Yang Maha Pemurah lagi Maha Penyayang. Penulis panjatkan puji syukur dan terimakasih yang sebesar-besarnya atas rahmat, nikmat, kebahagiaan serta seluruh anugerah dalam bentuk materi ataupun spiritual yang telah dilimpahkan kepada penulis dan kepada seluruh hamba-hamba-Nya, karena hanya dengan rahmat, taufik dan nikmat-Nya lah penulis bisa menyelesaikan dan mengantarkan penelitian yang berjudul “Verifikasi Kemiripan Wajah Menggunakan Deep Learning Dengan Arsitektur Jaringan Siamese” ini sampai pada proses terakhir.

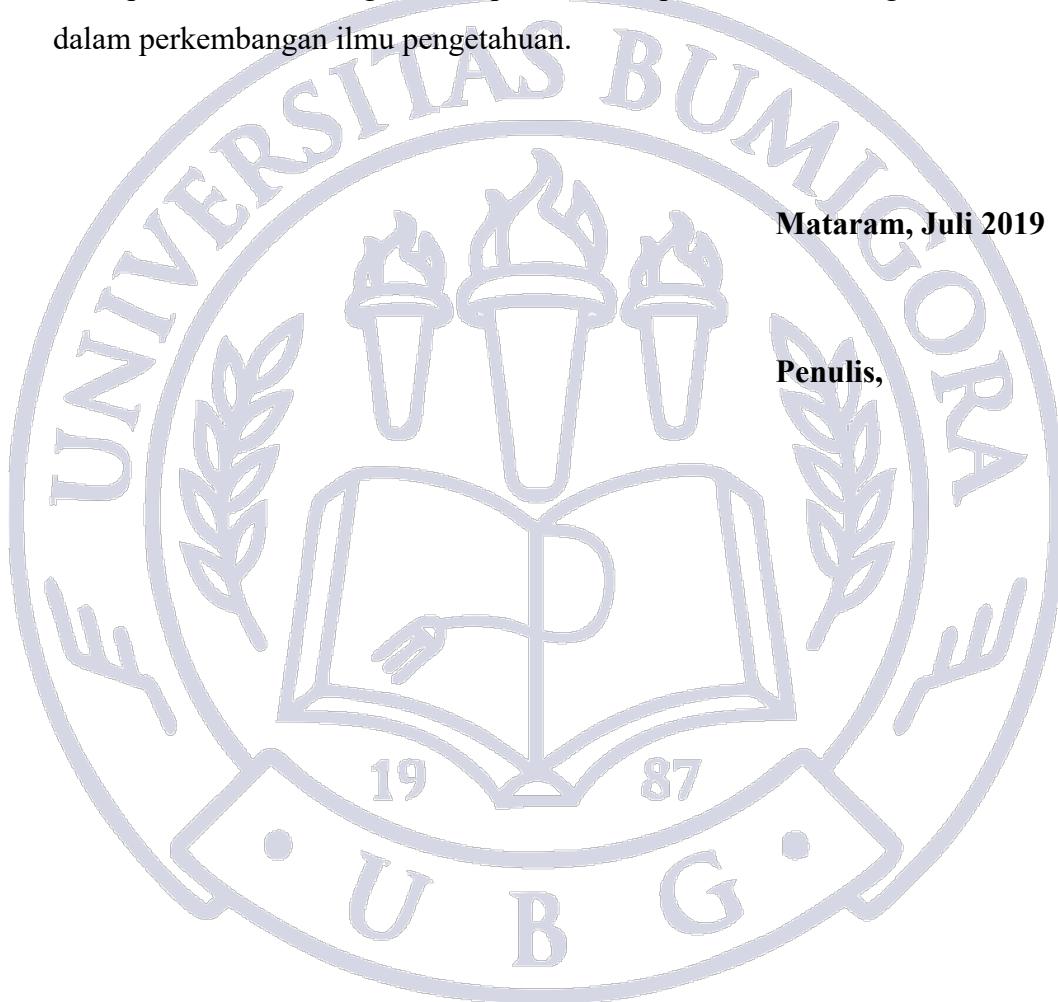
Tak luput pula penulis ingin menyampaikan ucapan terimakasih kepada pihak-pihak yang memiliki kontribusi besar atas kelancaran proses penggerjaan Skripsi ini. Pada kesempatan ini penulis menyampaikan terima kasih sebesar-besarnya kepada:

1. Bapak dan Ibunda tercinta yang tiada hentinya dengan sabar, tulus danikhlas mencerahkan do'a dan kasih sayang serta dukungan baik morilataupun materil kepada penulis. Tidak luput pula ucapan terimakasih kepada keluarga besar yang telah mendukung dalam pelaksanaan pendidikan yang dijalankan penulis.
2. Bapak Jian Budiarto S.T., M.Eng, selaku Dosen Pembimbing Satu dalam penggerjaan Skripsi ini.
3. Ibu Kartarina, S.Kom., M.Kom, selaku Dosen Pembimbing Dua dalam penggerjaan Skripsi ini.
4. Bapak Ir. Anthony Anggrawan, M.T., Ph.D, selaku rektor Universitas Bumigora.
5. Bapak Heroe Santoso, M.Kom, selaku Ketua Program Studi S1 Ilmu Komputer.
6. Bapak Ibu Dosen pengajar Universitas Bumigora yang telah mengajarkan segala pengetahuan yang penulis dapat selama melaksanakan pendidikan di Universitas Bumigora.

Penulis sebagai manusia biasa, menyadari dengan sepenuhnya bahwa dalam penulisan Skripsi ini masih jauh dari kesempurnaan, maka penulis senantiasa mengharapkan teguran, kritik serta saran yang sifatnya membangun untuk bersama-sama memajukan dan meningkatkan segala aspek pengetahuan pada penelitian ini agar senantiasa bersama-sama memberikan kontribusi pada perkembangan ilmu pengetahuan. Akhirnya penulis berharap semoga Skripsi ini dapat bermanfaat bagi semua pihak dan dapat diterima sebagai kontribusi dalam perkembangan ilmu pengetahuan.

Mataram, Juli 2019

Penulis,





LEMBAR PERNYATAAN KEASLIAN

Saya yang bertanda tangan di bawah ini:

Nama : Hairul Imam
Nim : 1310520075
Program Studi : S1 Ilmu Komputer
Kompetensi : Rekayasa Perangkat Lunak

Menyatakan bahwa skripsi yang berjudul

VERIFIKASI KEMIRIPAN WAJAH MENGGUNAKAN DEEP LEARNING DENGAN ARSITEKTUR JARINGAN SIAMESE

Dengan ini saya menyatakan bahwa dalam skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesejarnaan di suatu perguruan tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Jika dikemudian hari ditemukan plagiat (penjiplakan) suatu karya, maka saya bersedia untuk menerima sanksinya, yaitu berupa pembatalan / pencabutan gelar akademik yang telah saya terima.

Mataram, Juli 2019


Hairul Imam
1310520075

IZIN PENGGUNAAN

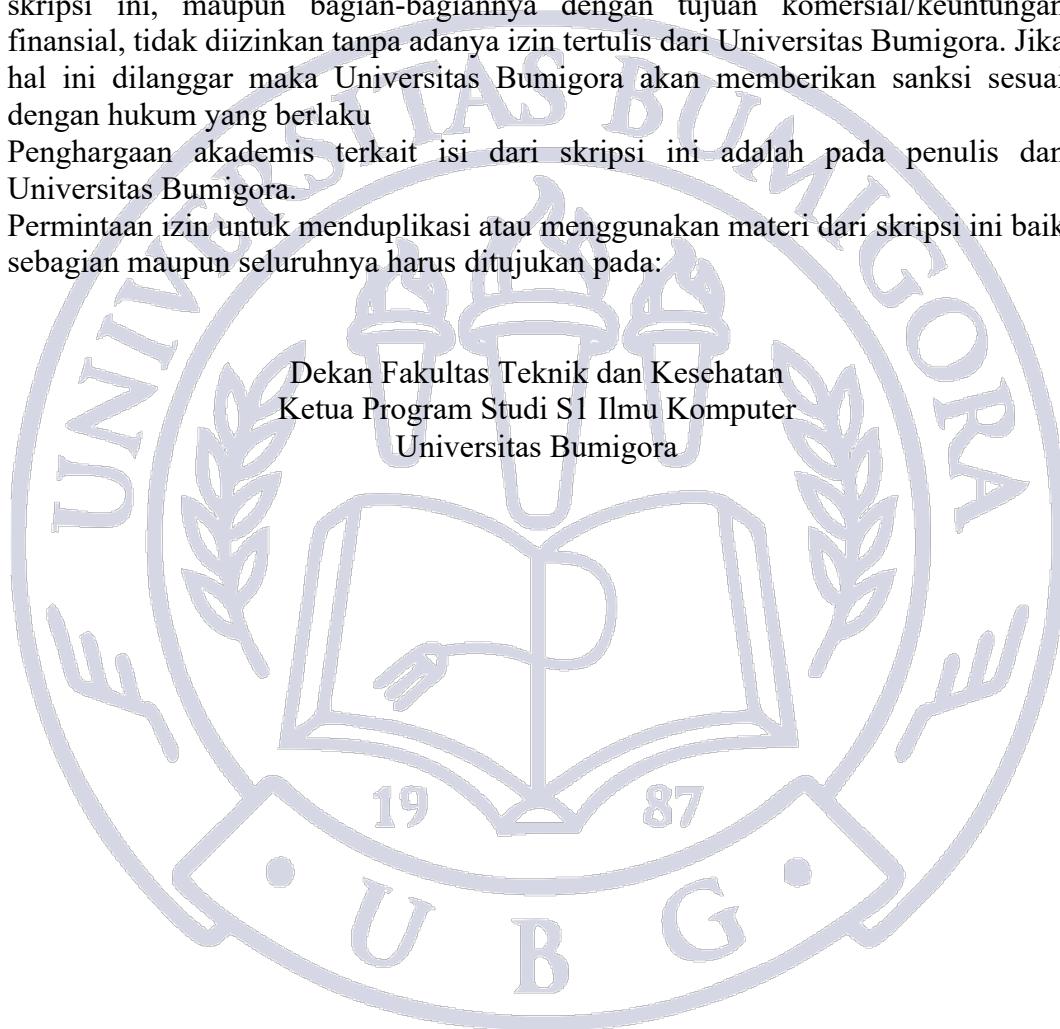
Skripsi ini merupakan syarat kelulusan pada Program Studi S1 Ilmu Komputer Universitas Bumigora, dengan ini penulis setuju jika skripsi ini digandakan (diduplikasi) baik sebagian maupun seluruhnya, ataupun dikembangkan untuk kepentingan akademis yang disetujui oleh pembimbing penulis, Ketua Program Studi, Dekan Fakultas Teknik dan Kesehatan.

Untuk dimaklumi, bahwa menduplikasi, mempublikasikan atau menggunakan skripsi ini, maupun bagian-bagiannya dengan tujuan komersial/keuntungan finansial, tidak diizinkan tanpa adanya izin tertulis dari Universitas Bumigora. Jika hal ini dilanggar maka Universitas Bumigora akan memberikan sanksi sesuai dengan hukum yang berlaku

Penghargaan akademis terkait isi dari skripsi ini adalah pada penulis dan Universitas Bumigora.

Permintaan izin untuk menduplikasi atau menggunakan materi dari skripsi ini baik sebagian maupun seluruhnya harus ditujukan pada:

Dekan Fakultas Teknik dan Kesehatan
Ketua Program Studi S1 Ilmu Komputer
Universitas Bumigora



ABSTRAK

Verifikasi wajah adalah masalah yang cukup populer dalam bidang *computer vision*. Banyak pendekatan yang telah dilakukan untuk menyelesaikan masalah tersebut baik menggunakan model matematika murni dengan mempelajari pola geometri pada wajah secara manual maupun cara otomatis menggunakan pendekatan pembelajaran mesin.

Penelitian ini mencoba memecahkan masalah tersebut dengan pendekatan *deep learning*, dimana model dilatih menggunakan *triplet loss* yang didefinisikan pada paper *FaceNet*. Rancangan model yang digunakan adalah *Siamese* dengan menerapkan ResNet-50 yang telah dimodifikasi untuk mempelajari fitur yang ada pada gambar sehingga mampu mereduksi dimensi gambar yang tinggi menjadi vektor baris yang rendah berdimensi 1x128 yang disebut sebagai *embedding*.

Setelah model berhasil mempelajari *embedding* yang baik pada gambar maka masalah verifikasi wajah bisa diselesaikan dengan membandingkan jarak *embedding* antar gambar dimana jarak yang dekat dapat diartikan sebagai wajah yang mirip (*genuine*) dan jarak yang jauh dapat diartikan sebagai wajah yang berbeda (*impostor*).

Pada penelitian ini, model berhasil dilatih pada dataset VGG Face v2 (*Visual Geometry Group*) dengan nilai akurasi 92% pada dataset LFW (*Labeled Faces in the Wild*) sebagai data *testing* dan mendapatkan nilai AUC (*Area Under the Curve*) 97%. Nilai AUC yang tinggi dapat diartikan bahwa model dapat memverifikasi dengan baik gambar wajah orang yang sama sebagai *genuine* dan gambar wajah orang yang berbeda sebagai *impostor*.

Kata Kunci: *Siamese*, *Triplet Loss*, Verifikasi Wajah, *Face Embedding*, *Dimensionality Reduction*.

DAFTAR ISI

LEMBAR PENGESAHAN

KATA PENGANTAR	i
----------------------	---

LEMBAR PERNYATAAN KEASLIAN	iii
----------------------------------	-----

IZIN PENGGUNAAN	iv
-----------------------	----

ABSTRAK	v
---------------	---

DAFTAR ISI	vi
------------------	----

DAFTAR GAMBAR	ix
---------------------	----

DAFTAR TABEL	xi
--------------------	----

BAB I PENDAHULUAN	1
-------------------------	---

1.1. Latar Belakang	1
1.2. Rumusan Masalah	3
1.3. Batasan Masalah	3
1.4. Tujuan Penulisan	3
1.5. Manfaat Penulisan	3
1.6. Sistematika Penulisan	4

BAB II LANDASAN TEORI	5
-----------------------------	---

2.1. Citra Digital	5
2.2. Pengolahan Citra Digital	5
2.3. Machine Learning	5
2.3.1. Supervised Learning	6
2.3.2. Unsupervised Learning	6
2.4. Artificial Neural Network (ANN)	6
2.4.1. Input	8
2.4.2. Weight	8
2.4.3. Bias	9
2.4.4. Aktivasi	9
2.5. Deep Learning	11
2.5.1. Layer Konvolusi	13
2.5.2. Max Pool	14
2.5.3. Batch Normalization	14
2.5.4. Dropout	15
2.6. Training	16
2.6.1. Epoch	16
2.6.2. Batch Size	16
2.6.3. Backpropagation	16
2.7. Evaluasi dan Pengujian	19

2.7.1. Confusion Metric	19
i. Akurasi	20
ii. Recall	20
iii. Precision	20
iv. Pengukuran F1	21
v. Kurva ROC dan AUC	21
BAB III METODOLOGI PENELITIAN	22
3.1. Dataset	22
3.1.1. Data Latih (<i>Training</i>)	22
3.1.2. Data Uji (<i>Testing</i>)	24
3.1.3. MTCNN	26
3.1.4. Preprocessing	27
3.2. Siamese	32
3.2.1. Residual Network (ResNet)	34
i. ResNet-50	37
ii. Konfigurasi ResNet	38
iii. Modifikasi Layer Resnet-50	38
3.3. Training	40
3.3.1. Triplet Loss	40
3.3.2. Pemilihan Triplet	42
i. Easy Triplet	43
ii. Hard Triplet	43
iii. Semi Hard Triplet	43
3.3.3. Proses Training dan Testing	44
i. Gambaran Umum Training dan Testing	43
ii. Generate Random Triplet	48
iii. Proses Training	50
iv. Proses Testing	51
BAB IV HASIL DAN PEMBAHASAN	54
4.1. Konfigurasi Hardware dan Software	54
4.2. Proses Training dan Testing	54
4.2.1. Layer FC	54
4.2.2. Layer Ekstraksi Fitur	58
4.3. Hasil Belajar	61
4.3.1. Layer Conv1	61
4.3.2. Layer BN1	62
4.3.3. Layer ReLU	63
4.3.4. Layer Max Pool	64
4.3.5. Layer1	65
4.3.6. Layer2	65
4.3.7. Layer3	66
4.3.8. Layer4	67
4.3.9. Layer FC	68

4.4. Percobaan Verifikasi Wajah	69
BAB V PENUTUP	71
5.1. Kesimpulan	71
5.2. Saran	71
DAFTAR REFERENSI	73

LAMPIRAN



DAFTAR GAMBAR

Gambar 2.1 Struktur Neuron Pada Otak Manusia	7
Gambar 2.2 Struktur Perceptron Pada ANN	7
Gambar 2.3 Grafik Aktivasi ReLU	10
Gambar 2.4 Grafik Aktivasi Sigmoid	10
Gambar 2.5 Grafik Fungsi Tanh	11
Gambar 2.6 Perbedaan Machine Learning dan Deep Learning	13
Gambar 2.7 Ilustrasi Proses Konvolusi	14
Gambar 2.8 Ilustrasi Operasi Max Pool	14
Gambar 2.9 Ilustrasi Proses Dropout	15
Gambar 2.10 Ilustrasi Forward dan Backward Pass	18
Gambar 3.1 Grafik Distribusi Dataset VGGv2	23
Gambar 3.2 Contoh Gambar Dataset VGG Face v2	23
Gambar 3.3 Struktur Folder Dataset VGG Face v2	24
Gambar 3.4 Struktur Folder Dataset LFW	25
Gambar 3.5 Contoh Gambar Dataset LFW	25
Gambar 3.6 Pipeline MTCNN Dalam Menghasilkan Face Landmark	26
Gambar 3.7 Arsitektur MTCNN	27
Gambar 3.8 Visualisasi Normalisasi Data	30
Gambar 3.9 Transformasi Gambar Ke Tensor	32
Gambar 3.10 Contoh Jaringan Siamese	33
Gambar 3.11 Error Lebih Tinggi Pada Arsitektur Yang Lebih Dalam	34
Gambar 3.12 Blok Residual Dengan Sambungan Shortcut	35
Gambar 3.13 Arsitektur Jaringan Plain dan Residual Block	36
Gambar 3.14 Perbandingan Kesalahan ResNet dan Plain	37
Gambar 3.15 Beberapa Jenis Arsitektur ResNet	37
Gambar 3.16 Blok Jaringan ResNet Yang Utuh	39

Gambar 3.17 Blok Jaringan ResNet Tanpa Layer Terakhir	39
Gambar 3.18 Ilustrasi Operasi L2 Norm dan Skala α	39
Gambar 3.19 Layer FC Setelah Penambahan L2 Norm & Skala α	40
Gambar 3.20 Ilustrasi Belajar Triplet Loss	41
Gambar 3.21 Contoh Gambar Anchor, Positive dan Negative	41
Gambar 3.22 Ilustrasi Pembagian Jenis Triplet	43
Gambar 3.23 Gambaran Besar Proses Training dan Testing	45
Gambar 3.24 Flowchart Pembuatan Triplet Acak	49
Gambar 3.25 Flowchart Training	50
Gambar 3.26 Flowchart Proses Testing	52
Gambar 4.1 Grafik Triplet Loss Hasil Training & Testing Layer FC	55
Gambar 4.2 Grafik Akurasi Hasil Training & Testing Layer FC	55
Gambar 4.3 Grafik Triplet Loss Hasil Training & Testing Layer Eks. Fitur ..	58
Gambar 4.4 Grafik Akurasi Hasil Training & Testing Layer Ekstraksi Fitur ..	58
Gambar 4.5 Grafik ROC Proses Training Terakhir	59
Gambar 4.6 Gambar Input Setelah Tahap Preprocessing	61
Gambar 4.7 Output Layer Conv1	62
Gambar 4.8 Output Layer BN1	63
Gambar 4.9 Output Layer ReLU	64
Gambar 4.10 Output Layer Max Pool	64
Gambar 4.11 Output Layer1	65
Gambar 4.12 Output Layer2	66
Gambar 4.13 Output Layer3	67
Gambar 4.14 Output Layer4	68
Gambar 4.15 Hasil Embedding Layer FC	68
Gambar 4.16 Jumlah Verifikasi Benar Setiap Nilai Threshold	69

DAFTAR TABEL

Tabel 3.1 Ketentuan Nilai Mean Yang Digunakan	30
Tabel 3.2 Ketentuan Nilai Std Yang Digunakan	30
Tabel 3.3 Ilustrasi Preprocessing Gambar Untuk Training	31
Tabel 3.4 Ilustrasi Preprocessing Gambar Untuk Testing	31
Tabel 3.5 Perbandingan Jumlah Parameter Model ResNet	38
Tabel 3.6 Contoh Jenis Triplet	44
Tabel 3.7 Rincian Input dan Output Setiap Layer	46
Tabel 3.8 Rincian Arsitektur Setiap Operasi Konvolusi	47
Tabel 4.1 Training Layer FC Learning Rate 0.001	56
Tabel 4.2 Training Layer FC Learning Rate 0.0001	57
Tabel 4.3 Training Layer Ekstraksi Fitur	60
Tabel 4.4 Hasil Jumlah Benar Setiap Nilai Threshold	70
Tabel 4.5 Jarak Gambar Hard Negative Setelah Training	70

BAB I

PENDAHULUAN

1.1. Latar Belakang

Deep Learning adalah suatu teknik pembelajaran mesin (*Machine Learning*) yang memanfaatkan arsitektur jaringan syaraf tiruan (JST). Jaringan syaraf tiruan/*Artificial Neural Network (ANN)* merupakan sebuah struktur yang memanfaatkan model dasar pada jaringan syaraf otak manusia untuk mengolah data.

Deep Learning telah digunakan secara luas oleh banyak kalangan baik industri maupun akademik pada objek digital untuk menganalisa informasi-informasi penting dan menyelesaikan berbagai jenis permasalahan pada setiap bit *frame*-nya. Tidak terkecuali pengenalan wajah pada gambar.

Pengenalan wajah pada gambar merupakan objek penelitian yang sangat banyak diminati dengan potensi penerapan pada berbagai industri dan bidang. Pendekatan yang dilakukan pun berbagai macam seperti menggunakan teknik *computer vision*, *machine learning/deep learning*. Setiap teknik memiliki kehandalan yang beragam, namun akhir-akhir ini teknik *deep learning* memiliki kemajuan yang pesat pada penyelesaian masalah tersebut dengan nilai akurasi tinggi karena mampu belajar dari data dalam jumlah besar tanpa ketergantungan pada perekayasaan fitur secara manual untuk dapat melatih model. Proses pembelajaran fitur pada *deep learning* dapat dilakukan dengan otomatis sehingga jumlah data menentukan tingkat pemahaman mesin mengenai sebuah topik.

Kemampuan *Deep Learning* belajar berdasarkan data dalam jumlah besar membuat teknik ini banyak digunakan oleh banyak pihak dalam menyelesaikan masalah serupa karena didukung oleh kemajuan kekuatan komputasi modern seperti GPU (*Graphics Processing Unit*) yang sangat cepat dalam hal melakukan operasi matrik. Berdasarkan kelebihan ini, penulis memilih menggunakan pendekatan *deep learning* dengan arsitektur jaringan *siamese* dalam melakukan penelitian verifikasi kemiripan wajah.

Penelitian verifikasi wajah dengan pendekatan ANN juga telah dilakukan oleh beberapa orang sebelumnya seperti Dimas Achmad Akbar Kusuma berjudul Verifikasi Citra Wajah Menggunakan Metode *Discrete Cosine Transform* Untuk Aplikasi Login (Kusuma, Ardilla, & Dewantara, 2011). Dalam penelitian tersebut ekstraksi fitur yang dilakukan menggunakan metode *Discrete Cosine Transform*, lalu hasil fitur yang telah diekstrak dari beberapa sampel gambar wajah akan disimpan dalam file kemudian dilatih menggunakan fungsi MSE (*Mean Squared Error*) untuk mempelajari nilai parameter yang optimal pada arsitektur ANN yang digunakan, namun tidak dijelaskan apakah penelitian tersebut menggunakan arsitektur *deep learning* dalam arsitektur jaringannya. Data latih penelitian tersebut menggunakan data yang dikumpulkan secara mandiri yang terdiri dari 10 individu dengan masing-masing individu memiliki 10 gambar dengan pose dan kemiringan serta gaya yang berbeda-beda sehingga total dataset sebanyak 100 gambar. Melatih jaringan dengan *dataset* yang terbatas dapat menyebabkan generalisasi yang tidak baik ketika diuji pada data baru karena sedikitnya fitur wajah yang dipelajari. Selain itu fungsi MSE bukanlah fungsi *dimensionality reduction* karena *output* dari fungsi tersebut adalah nilai skalar yang akan digunakan untuk membandingkan antara nilai *output* dengan nilai sebenarnya (label) sehingga fungsi tersebut tidak mempelajari sebuah *embedding*.

Fungsi yang dapat melatih jaringan dengan objektif mempelajari dimensi gambar yang tinggi menjadi dimensi yang rendah (*dimensionality reduction*) sangat diperlukan pada kasus verifikasi wajah. Karena kasus verifikasi wajah adalah proses pencocokan antar dua gambar sehingga representasi fitur (*embedding*) dari kedua gambar tersebut harus benar-benar optimal. Karena keadaan inilah fungsi yang dapat mempelajari *embedding* sangat dibutuhkan. Dengan demikian, dilakukan penelitian dengan metode *deep learning* menggunakan arsitektur jaringan *Siamese* yang dilatih dengan fungsi *triplet loss* untuk mempelajari *embedding* pada gambar.

1.2. Rumusan Masalah

Dari pemaparan latar belakang diatas, maka permasalahan yang ingin penulis teliti adalah Verifikasi Kemiripan Wajah Menggunakan *Deep Learning* Dengan Arsitektur Jaringan *Siamese*.

1.3. Batasan Masalah

Dari rumusan masalah diatas, penulis membatasi ruang lingkup masalah penelitian seperti berikut:

1. Arsitektur yang akan dilatih untuk melakukan verifikasi kemiripan wajah adalah arsitektur jaringan *Siamese* dengan blok jaringan ResNet-50 yang telah dimodifikasi.
2. Dataset yang digunakan dalam penelitian adalah file gambar dengan rincian sesuai yang dibutuhkan penulis.
3. Data latih pada penelitian ini menggunakan dataset VGG *Face* v2.
4. Data uji pada penelitian ini menggunakan dataset LFW.
5. Keberhasilan dari model (hasil latihan pembelajaran dataset) ini akan diukur melalui nilai akurasi pada *confusion metric* dan nilai AUC (*Area Under the Curve*).

1.4. Tujuan

Tujuan dari penelitian ini adalah melakukan verifikasi kemiripan wajah dengan model yang penulis latih menggunakan *triplet loss* dengan arsitektur jaringan *Siamese* dan menggunakan ResNet-50 yang telah dimodifikasi sebagai badan dari jaringan *Siamese*.

1.5. Manfaat

Melalui penelitian ini beberapa manfaat yang bisa didapatkan sebagai berikut:

- a. Menyediakan pengetahuan kepada pembaca mengenai tingkat akurasi verifikasi kemiripan wajah menggunakan arsitektur jaringan *Siamese* melalui pembelajaran pada dataset VGG *Face* v2.

- b. Menyediakan pengetahuan kepada pembaca dalam hal penerapan arsitektur jaringan *Siamese* dengan fungsi *triplet loss* pada kasus verifikasi kemiripan wajah.

1.6. Sistematika Penulisan

Sistematika penulisan dalam penelitian ini disusun sebagai berikut:

BAB I: PENDAHULUAN

Berisi uraian tentang kondisi dan permasalahan yang mendasari dilakukannya penelitian, hal-hal yang akan dilakukan dalam penelitian beserta batasan-batasannya serta manfaat yang bisa didapatkan melalui penelitian yang dilakukan.

BAB II: LANDASAN TEORI

Berisi teori-teori dan definisi-definisi yang digunakan dalam penelitian yang didasarkan dari sumber-sumber penelitian yang berkaitan.

BAB III: METODOLOGI PENELITIAN

Pada bab ini membahas tentang metodologi yang digunakan dan cara kerja setiap metode yang digunakan sehingga mampu digunakan untuk menyelesaikan masalah yang dihadapi.

BAB IV: HASIL DAN PEMBAHASAN

Bab ini akan membahas tentang hasil penerapan metode yang digunakan dan penjelasan setiap hasil pembelajaran dari model yang digunakan.

BAB V: PENUTUP

Bab ini berisi tentang rangkuman penelitian dan memberikan masukan dan saran yang dapat dilakukan untuk meningkatkan hasil dari penelitian.

BAB II

LANDASAN TEORI

2.1. Citra Digital

Citra atau gambar dapat didefinisikan sebagai sebuah fungsi dua dimensi, $f(x, y)$, dimana x dan y adalah koordinat bidang datar, dan harga fungsi f di setiap pasangan koordinat (x, y) disebut intensitas atau level keabuan (*grey level*) dari gambar di titik itu (Hermawati, 2013).

Jika x , y dan f semuanya berhingga (*finite*), dan nilainya diskrit, maka gambarnya disebut citra digital (gambar digital). Sebuah citra digital terdiri dari sejumlah elemen yang berhingga, di mana masing-masing mempunyai lokasi dan nilai tertentu. Elemen-elemen ini disebut sebagai *picture element*, *image element*, pels atau pixels.

2.2. Pengolahan Citra Digital

Pengolahan citra digital merupakan suatu disiplin ilmu yang mempelajari hal-hal yang berkaitan dengan perbaikan kualitas gambar (peningkatan kontras, transformasi warna, restorasi citra), transformasi gambar (rotasi, translasi, transformasi geometrik, skala), agar mudah diinterpretasi oleh manusia/mesin (komputer). Masukannya adalah citra dan keluarannya juga citra tapi dengan kualitas lebih baik daripada citra masukan misal citra warna kurang tajam, kabur (*blur*), dan mengandung *noise* (misal bintik-bintik putih) sehingga perlu ada pemrosesan untuk memperbaiki citra karena citra tersebut menjadi sulit diinterpretasikan karena informasi yang disampaikan menjadi berkurang (Universitas Sumatra, 2014, p. 16).

2.3. Machine Learning

Machine Learning adalah salah satu terapan bidang Kecerdasan Buatan/*Artificial Intelligence* (AI) yang mampu secara otomatis mempelajari dan berkembang dari pengalaman tanpa harus diprogram secara khusus. *Machine Learning* berfokus pada pengembangan program komputer melalui akses terhadap data lalu menggunakan data tersebut untuk mengajari dirinya

sendiri (Expert System, 2017). Beberapa macam pembelajaran pada *machine learning* adalah sebagai berikut:

2.3.1. Supervised Learning

Supervised Learning adalah metode pembelajaran *machine learning* yang dapat menerapkan hasil pembelajaran pada data baru. Hasil pembelajaran pada *supervised learning* diperoleh dari hasil latih pada data yang sudah diberi label sebelumnya, sehingga bisa memprediksi kesimpulan dari data baru yang ada pada masa mendatang.

Metode pembelajaran *supervised learning* dimulai dari menganalisa data yang telah diberi label (dataset), lalu algoritma pembelajaran yang digunakan akan menghasilkan pengetahuan yang bisa digunakan untuk membuat prediksi mengenai label data pada dataset yang baru setelah algoritma dilatih dengan cukup. Hasil dari algoritma pembelajaran ini juga dapat dibandingkan dengan label sebenarnya lalu kesalahan prediksi dari label sebenarnya bisa digunakan untuk mengubah model sesuai kesalahan yang dipelajari.

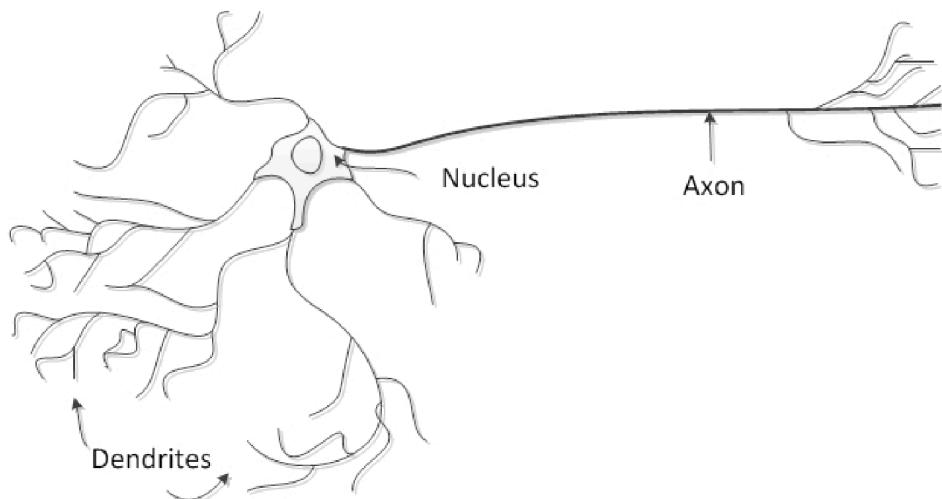
2.3.2. Unsupervised Learning

Algoritma *unsupervised learning* digunakan ketika data yang digunakan untuk mengajari algoritma pembelajaran tidak diberi label dan tidak juga telah diklasifikasi. Algoritma *unsupervised learning* mempelajari bagaimana mendapatkan pengetahuan dari data yang tidak memiliki label dan tidak terstruktur. Algoritma *unsupervised learning* tidak memberi *output* secara langsung melainkan memberikan gambaran dan struktur dari sebuah data yang tidak terstruktur.

2.4. Artificial Neural Network (ANN)

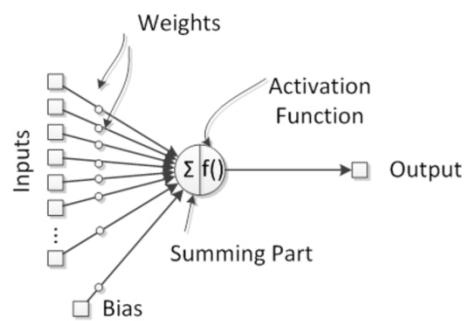
Artificial Neural Network atau Jaringan Syaraf Tiruan adalah sebuah sistem komputasi yang dirancang dengan menirukan cara otak manusia menganalisa dan memproses informasi. Sistem ANN memiliki kemampuan untuk belajar secara mandiri dimana hasil pembelajaran dipengaruhi oleh jumlah data yang digunakan dalam belajar (Jake Frankenfield, 2018).

Adapun cara ANN bekerja adalah dengan menirukan cara otak manusia bekerja, dimana ANN terbentuk dari satuan unit komputasi yang kecil-kecil yang disebut *node* yang saling terhubung membentuk sebuah jaringan. Otak manusia memiliki ratusan miliar *node* yang disebut dengan *neuron* dimana setiap *neuron* terdiri dari *dendrites* sebagai *input*, *nucleus* sebagai pemroses dan *axon* sebagai penyalur informasi yang telah diproses, dimana *axon* akan tersambung lagi ke *dendrites* dari *neuron* yang lain (Soares & Souza, 2016).



Gambar 2.1 Struktur Neuron Pada Otak Manusia

Menirukan sistem *neuron* maka pada ANN terdapat unit komputasi *neuron* terkecil yang disebut dengan *perceptron* yang memiliki cara kerja seperti *neuron* pada otak manusia.



Gambar 2.2 Struktur Perceptron Pada ANN

Gambar 2.2 diatas dapat diekspresikan dengan persamaan matematis seperti dibawah:

$$y_j = f \left(\sum_{i=1}^n w_{ij} x_i + b_i \right)$$

Dimana y adalah *output*, f adalah fungsi aktivasi, w adalah *weight*, x adalah *input* dan b adalah bias, setiap komponen diatas memiliki penjelasan sebagai berikut:

2.4.1. Input

Input adalah data yang masuk yang akan diproses, dimana pada proses pembelajaran (*training*) *perceptron* akan mempelajari data *input* dengan melatih *learnable parameters* (*weight* dan *bias*) agar mampu mencapai tujuan pembelajaran yang diinginkan. Pada saat *testing* *perceptron* akan membuat prediksi dari *input* dengan *weight* dan *bias* yang sudah dilatih pada proses *training*.

2.4.2. Weight

Weight adalah salah satu komponen *learnable parameter* pada *perceptron*, dalam *perceptron* *weight* memiliki peran sebagai penyambung antar *input* dengan *neuron* dan memiliki kemampuan untuk menguatkan atau melemahkan signal yang diterima (*input*) dengan cara melakukan operasi perkalian dengan nilai *weight* (bobot). Dimana semakin besar nilai bobot yang digunakan maka signal input akan semakin menguat begitu juga sebaliknya (Soares & Souza, 2016). Pada proses *training* kekuatan *weight* (bobot) akan disesuaikan (dilatih) agar nilai *weight* dapat sesuai dengan tujuan yang ingin dicapai. Sebagai contoh persamaan linear sederhana dibawah.

$$2w = 8$$

Dalam kasus persamaan linear sederhana diatas tujuan yang ingin dicapai adalah 8 dan *input* yang akan diproses adalah 2 dan *weight* yang akan dipelajari agar mampu memenuhi tujuan adalah variabel w . Dimana *perceptron* akan melatih variabel w agar mampu mendekati angka 4 sehingga tujuan pembelajaran dapat dimaksimalkan.

2.4.3. Bias

Bias juga adalah komponen *learnable parameter* pada *perceptron*. Dalam struktur *perceptron* bias memiliki kemampuan untuk mengubah hasil proses variabel *weight*. Kemampuan bias dalam mengubah hasil proses *weight* sangat diperlukan untuk menggenapkan kekurangan hasil proses variabel *weight*. Sebagai contoh persamaan linear yang sama:

$$2w + b = 8$$

Dalam kasus linear diatas sebagai contoh nilai *w* yang berhasil dipelajari adalah 3.5 tanpa variabel *b* hasil dari $2 \times 3.5 = 7$, dalam kasus ini *perceptron* akan melatih variabel *b* agar bisa menggenapkan hasil proses $2w$ agar dapat memenuhi nilai 8, dengan memaksimalkan variabel *b* mendekati nilai 1 maka tujuan pembelajaran dapat dimaksimalkan.

2.4.4. Aktivasi

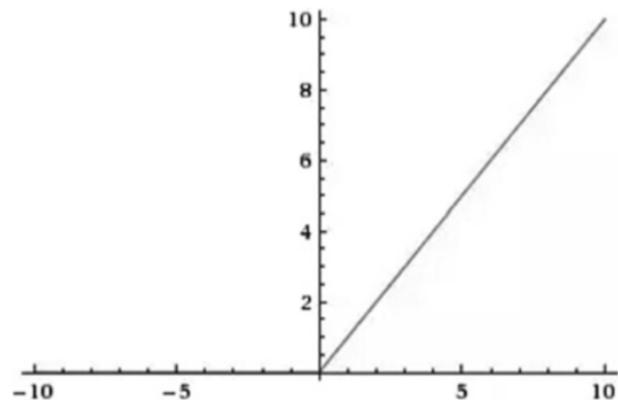
Output dari unit *neuron* dihasilkan dari sebuah fungsi aktivasi. Fungsi aktivasi dalam *perceptron* berfungsi untuk menghasilkan nilai *nonlinear* dimana ini dibutuhkan karena secara alami *neuron* pada otak manusia juga berperilaku secara *nonlinear*. Fungsi aktivasi biasanya dibatasi antar dua rentang nilai yang menyebabkan hasilnya *nonlinear*, namun ada beberapa fungsi aktivasi yang menghasilkan fungsi linear (Soares & Souza, 2016). Ada banyak fungsi aktivasi yang bisa digunakan dalam ANN, beberapa contoh fungsi aktivasi yang bisa digunakan adalah sebagai berikut:

i. ReLU

Rectified Linear Unit (ReLU) adalah fungsi aktivasi linear yang membatasi nilai output minimal pada nilai 0 (Wani, Bhat, Afzal, & Khan, 2018). Bentuk fungsi aktivasi ini bisa dilihat seperti dibawah.

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

Jika dibentuk dalam grafik output dari ReLU terlihat seperti gambar 2.3 dibawah.



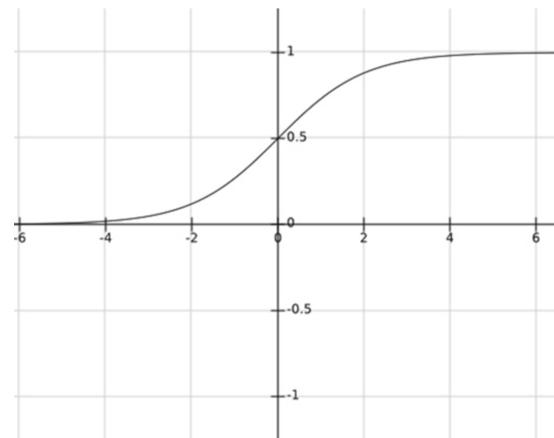
Gambar 2.3 Grafik Aktivasi ReLU

ii. Sigmoid

Sigmoid adalah fungsi nonlinear yang juga dikenal dengan istilah *logistic function*. Fungsi *Sigmoid* memiliki *output* dalam rentang [0,1] (Wani et al., 2018). Bentuk fungsi *Sigmoid* terlihat dalam persamaan dibawah.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Persamaan diatas jika dibentuk dalam grafik akan terlihat seperti gambar 2.4 dibawah.



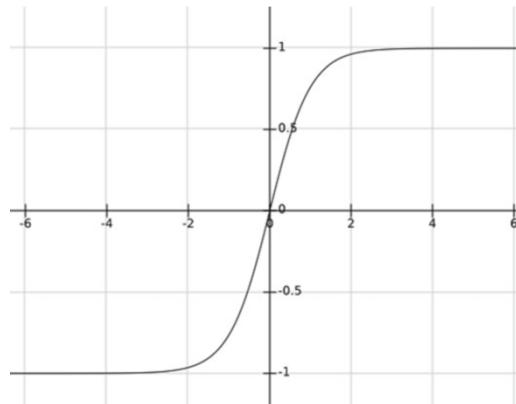
Gambar 2.4 Grafik Aktivasi *Sigmoid*

iii. Hyperbolic Tangent (Tanh)

Fungsi aktivasi Hyperbolic Tangent atau dikenal juga dengan nama fungsi Tanh adalah fungsi nonlinear seperti *Sigmoid* namun memetakan output dalam rentang [-1,1]. Kelebihan fungsi Tanh dari Sigmoid adalah nilai input negatif akan ditekan dengan kuat menjadi nilai negatif dan nilai input nol akan dipetakan menjadi nilai mendekati 0 (Wani et al., 2018). Bentuk matematis dari fungsi Tanh terlihat seperti dibawah.

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

Persamaan diatas jika dipetakan dalam grafik akan terlihat seperti gambar 2.5 dibawah.



Gambar 2.5 Grafik Fungsi Tanh

iv. Softmax

Fungsi aktivasi Softmax adalah fungsi yang banyak digunakan dalam kasus klasifikasi. Bentuk aktivasi ini terlihat seperti dibawah.

$$f(x_j) = \frac{e^{x_j}}{\sum_{k=1}^n e^{x_k}}$$

2.5. Deep Learning

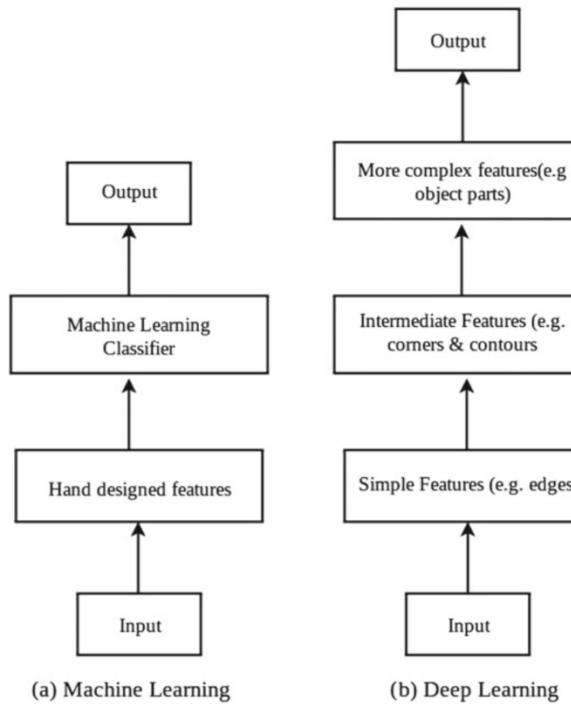
Deep Learning adalah area baru dari *machine learning* yang semakin populer pada dekade ini. Istilah *Deep Learning* merujuk pada arsitektur ANN yang terdiri dari beberapa *hidden layers* (*Deep Networks*) yang berfungsi untuk mengenali fitur-fitur yang berbeda pada *input*. Algoritma *Deep Learning*

bertujuan untuk mencari struktur yang tidak diketahui untuk mendapatkan representasi yang baik dari *input* (Wani et al., 2018).

Metode *machine learning* yang konvensional terbatas dalam cara memproses data mentah. Dalam waktu yang lama untuk dapat mengenal pola menggunakan metode *machine learning* membutuhkan keahlian khusus dalam bidang-bidang tertentu dan membutuhkan keahlian rekayasa yang sangat hati-hati sesuai kasus yang dihadapi untuk dapat mengekstraksi fitur dari sebuah *input*.

Namun dengan *Deep Learning* proses pengenalan pola dan rekayasa secara manual yang membutuhkan keahlian dalam domain tertentu tersebut dapat dilakukan secara otomatis selama proses pembelajaran. Dalam proses pembelajaran (*training*) metode *Deep Learning* dapat mengenali fitur-fitur yang ada pada data mentah secara otomatis dengan hasil yang lebih baik dari cara manual (Wani et al., 2018).

Metode pembelajaran fitur pada *Deep Learning* dilakukan dengan membentuk layer hirarki dimana setiap layer dibangun diatas layer yang lain. Layer yang paling bawah dari model ini bertanggung jawab untuk mempelajari representasi dasar dari masalah, dan layer-layer diatasnya bertanggung jawab untuk membentuk konsep yang lebih kompleks dari data. Sebagai contoh dalam kasus pengenalan wajah pada gambar, setiap pixel yang ada pada gambar akan dimasukkan ke hirarki layer. Setiap hidden layer pada struktur hirarki ini kemudian akan mengekstrak fitur-fitur dari gambar input. Pada layer pertama hirarki ini akan mendeteksi tepi-tepi dari wajah, kemudian pada layer kedua akan dipelajari garis-garis pipi, alis, cekungan pada dagu, pada layer berikutnya garis-garis yang telah dipelajari akan membentuk gambaran yang lebih abstrak dari gambar. Semua pembelajaran yang dilakukan pada setiap layer ini dilakukan secara otomatis tanpa campur tangan manusia. Secara visual perbedaan teknik *machine learning* konvesional dan *deep learning* bisa dilihat pada gambar 2.6 dibawah.



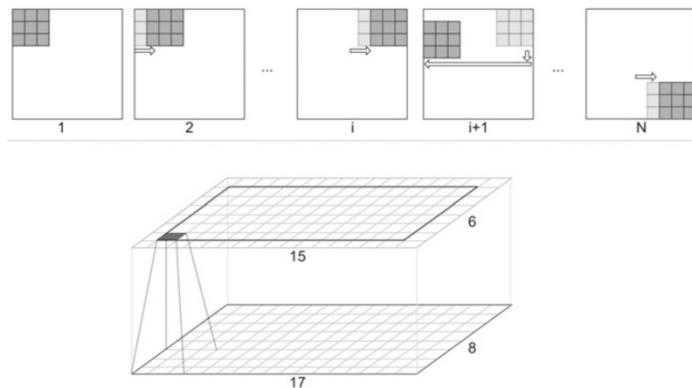
Gambar 2.6 Perbedaan *Machine Learning* dan *Deep Learning*

Adapun layer ekstraksi fitur yang bisa digunakan pada arsitektur *Deep Learning* adalah sebagai berikut.

2.5.1. Layer Konvolusi

Layer Konvolusi (*Convolutional Layer*) adalah layer operasi konvolusi 2D, dimana dengan operasi konvolusi setiap blok wilayah pada gambar dapat dipelajari dan menjadi fitur dari gambar tersebut. Komponen dari operasi konvolusi adalah matriks persegi yang disebut filter/kernel umumnya berdimensi 3x3, filter/kernel pada operasi konvolusi berfungsi untuk mengekstrak fitur setiap wilayah yang ada pada gambar. Dalam proses *training* filter/kernel akan menjadi *learnable parameter* yaitu setiap nilai matriks kernel akan dipelajari pada saat *training*. Pada setiap layer konvolusi jumlah kernel/filter bisa lebih dari satu sesuai kompleksitas dan jumlah fitur yang ingin dipelajari dari input gambar. Sebagai contoh jika pada sebuah gambar dengan satu kanal warna ingin dipelajari 64 jenis fitur maka pada layer *convolutional* bisa memiliki 64 jenis kernel/filter dengan dimensi

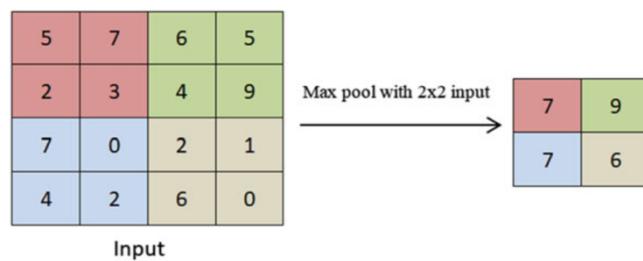
$R \times C$ sehingga hasil dari operasi layer *convolutional* adalah 64 gambar atau juga disebut 64 kanal (Skansi, 2018). Ilustrasi proses konvolusi bisa dilihat pada gambar 2.7 dibawah.



Gambar 2.7 Ilustrasi Proses Konvolusi

2.5.2. Max Pool

Layer *Max Pool* pada lapisan jaringan ANN berfungsi untuk mereduksi ukuran gambar dengan metode *down sampling*. Layer *Max Pool* akan menyimpulkan nilai tertinggi pada gambar dalam ukuran region tertentu (Wani et al., 2018). Sebagai contoh gambar 2.8 dibawah operasi *Max Pool* akan mengambil nilai tertinggi pada gambar dalam setiap 2x2 wilayah pada gambar.



Gambar 2.8 Ilustrasi Operasi *Max Pool*

2.5.3. Batch Normalization

Proses *Batch Normalization* (BN) pada lapisan ANN berfungsi untuk menormalisasi skala nilai *input*. Menurut Sergey Ioffe (2015) dalam proses *training* menambahkan lapisan BN dapat meningkatkan

generalisasi pada model sehingga model dapat belajar dengan baik tanpa *overfit* (Ioffe & Szegedy, 2015). Proses BN bisa dilihat dalam persamaan dibawah (Wani et al., 2018).

$$x_i^* = x_i \frac{M(x)}{\sqrt{V(x)}}$$

Dimana:

x_i^* adalah hasil proses BN

x_i adalah data yang akan dinormalisasi

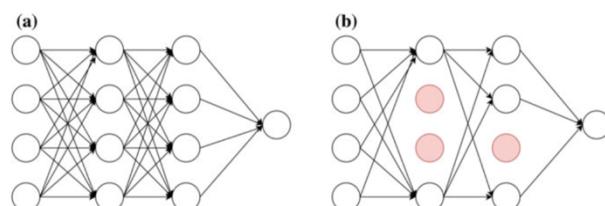
$M(x)$ adalah nilai rata-rata pada *batch*

$V(x)$ adalah nilai variansi dalam batch

Menumpuk layer Konvolusi dengan *Max Pool* menjadi beberapa layer disebut dengan CNN (*Convolutional Neural Network*).

2.5.4. Dropout

Lapisan jaringan ANN yang terdiri dari beberapa lapis yang kompleks dapat membuat model mempelajari fitur yang rumit pada data, setiap fitur yang dipelajari akan tersambung ke sejumlah *neuron* atau disebut *Fully Connected Layer* (FC) untuk membuat kesimpulan dari fitur yang didapat. Layer FC akan saling terhubung dengan layer FC yang lain dan sangat mudah mengalami *overfit*. *Overfit* adalah keadaan dimana model mempelajari data *training* dengan sangat baik namun gagal dalam memprediksi kesimpulan dari data baru, untuk mengatasi hal ini layer *dropout* dapat ditambahkan pada lapisan ANN. Layer *dropout* akan memutus secara acak sambungan FC pada saat *training* sehingga hanya sambungan yang masih terhubung yang akan dilatih pada saat *training* (Wani et al., 2018). Ilustrasi proses *dropout* bisa dilihat pada gambar 2.9 dibawah.



Gambar 2.9 Ilustrasi Proses *Dropout*

2.6. Training

Training dalam *Deep Learning* adalah proses melatih model untuk mempelajari data untuk mendapatkan hasil yang diinginkan. Dalam proses *training* ada beberapa komponen yaitu:

2.6.1. Epoch

Epoch dalam *training* merujuk pada berapa kali semua data *training* akan digunakan untuk belajar. Menggunakan data *training* dalam melatih model tidak cukup hanya sekali, dibutuhkan beberapa kali agar model dapat belajar dengan baik.

2.6.2. Batch Size

Batch Size dalam *training* merujuk pada jumlah data yang akan diproses untuk latihan dalam satu waktu. Metode *deep learning* pada umumnya membutuhkan data dalam jumlah besar agar dapat belajar dengan baik. Dalam proses belajar, memasukkan keseluruhan data ke dalam jaringan dalam waktu bersamaan akan memakan banyak sumber daya memory dan operasi komputasi. Agar dapat diproses dengan baik data yang dalam jumlah banyak akan dipecah menjadi beberapa bagian yang disebut dengan *mini batch* dimana jumlah data setiap *mini batch* disebut dengan *batch size* dan pemrosesan setiap *batch* disebut dengan iterasi. Umumnya ukuran *batch size* yang digunakan adalah kelipatan 2 misal 32, 64, 128 jumlah data.

2.6.3. Backpropagation

Backpropagation adalah algoritma yang sangat banyak digunakan untuk melatih model *deep learning*. Algoritma *backpropagation* akan mengubah nilai *learnable parameters* (*weight* dan *bias*) dimulai dari layer terakhir hingga layer pertama berdasarkan selisih (*error/cost/loss*) dari nilai prediksi dengan nilai yang diinginkan hingga nilai selisih (*error*) dapat diminimalisir.

Dalam ANN yang terdiri dari satu layer (*perceptron*) mengubah nilai *weight* sangat mudah dilakukan dengan algoritma *Gradient*

Descent dimana perubahan nilai weight baru bisa diberikan dengan persamaan dibawah (Sebastian Raschka, 2015).

$$\begin{aligned}
 w_j &= w_j + \Delta w_j \\
 \Delta w_j &= -\eta \frac{\partial J}{\partial w_j} \\
 \frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (t^i - o^i)^2 \\
 &= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (t^i - o^i)^2 \\
 &= \frac{1}{2} \sum_i 2(t^i - o^i) \frac{\partial}{\partial w_j} (t^i - o^i) \\
 &= \sum_i (t^i - o^i) \frac{\partial}{\partial w_j} \left(t^i - \sum_j w_j x_j^i \right) \\
 &= \sum_i (t^i - o^i) (-x_j^i)
 \end{aligned}$$

Sehingga:

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta \sum_i (t^i - o^i) (-x_j^i) = \eta \sum_i (t^i - o^i) x_j^i$$

Δw_j adalah nilai perubahan *weight* yang baru yang didefinisikan melalui turunan parsial hingga mendapatkan persamaan akhir yaitu $\eta \sum_i (t^i - o^i) x_j^i$

Dimana:

η adalah konstanta *learning rate*

t^i adalah nilai target yang diinginkan

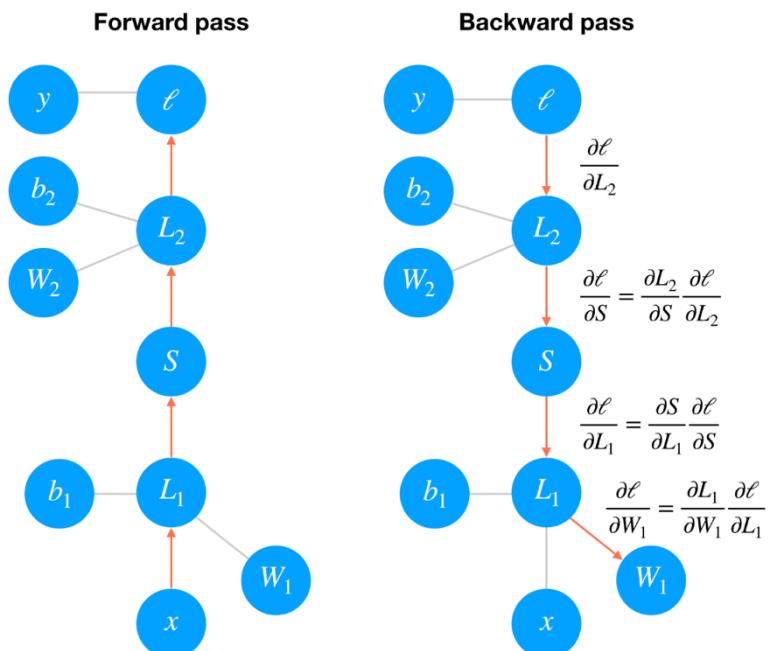
o^i adalah nilai prediksi

x_j^i adalah data *training*

Dengan persamaan diatas perubahan nilai *weight* akan ditekan sesuai selisih dari target dan prediksi, jika nilai target dan prediksi sama maka nilai $\Delta w_j = 0$ artinya tidak akan terjadi perubahan nilai *weight*,

perubahan yang dilakukan dalam algoritma *gradient descent* adalah untuk mencari *local/global minima*.

Berbeda dengan kasus *single layer* diatas, jaringan yang memiliki banyak layer cukup sulit untuk dilatih dengan metode *Gradient Descent* biasa, disinilah peran *backpropagation* (Udacity Course, 2018). *Backpropagation* digunakan untuk melatih jaringan dengan banyak layer, menurut Jeremy Howard dalam penerapannya *backpropagation* adalah terapan dari *chain rule* dalam aturan turunan fungsi (*derivative*). Sebagai contoh gambar 2.10 dibawah.



Gambar 2.10 Ilustrasi *Forward* dan *Backward Pass*

Dalam gambar *Forward pass* diatas data akan dimasukkan dari bawah hingga atas. Data akan diproses pada layer \$L_1\$ dengan *weight* \$W_1\$ dan *bias* \$b_1\$ lalu hasil proses akan ditransformasi pada fungsi aktivasi Sigmoid \$S\$ lalu terdapat operasi pada layer kedua \$L_2\$ dengan *weight* \$W_2\$ dan *bias* \$b_2\$ kemudian akan dihitung kesalahan prediksi dari nilai target yang diinginkan menggunakan fungsi *loss* \$\ell\$, setelah mendapatkan nilai *loss* (*error*) akan dilakukan proses pengubahan nilai \$W_1, b_1, W_2, b_2\$

hingga nilai *loss* (*error*) dapat diminimalisir. Agar dapat melatih jaringan diatas menggunakan *gradient descent*, akan dihitung nilai *gradient* dari *loss* lalu disebarluaskan (*propagate*) hingga layer pertama pada jaringan. Setiap operasi memiliki *gradient* antara *input* dan *output*, pada saat proses penyebaran (*propagate*) *gradient* ke layer pertama (*backward*) akan dilakukan proses perkalian antara nilai *gradient* layer sebelumnya dengan nilai *gradient* pada operasi layer saat ini, seperti terlihat pada gambar *backward pass* gambar 2.10 diatas. Secara matematis, proses ini akan menghitung nilai *gradient* dari *loss* sesuai nilai *weight* dengan aturan berantai (*chain rule*).

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial L_1}{\partial W_1} \frac{\partial S}{\partial L_1} \frac{\partial L_2}{\partial S} \frac{\partial \ell}{\partial L_2}$$

Nilai *weight* baru adalah hasil dari perhitungan *gradient* diatas dengan perkalian *learning rate* sehingga nilai *weight* baru bisa dihitung dengan persamaan dibawah.

$$W'_1 = W_1 - \eta \frac{\partial \ell}{\partial W_1}$$

Nilai konstanta *learning rate* η diberikan sehingga perubahan nilai *weight* dapat bergerak dengan cukup pada proses iterasi agar nilai *loss* tetap berada pada area minimum.

2.7. Evaluasi dan Pengujian

Untuk menguji tingkat pencapaian pada penelitian ini ada beberapa cara yang digunakan seperti yang dijelaskan pada poin-poin dibawah.

2.7.1. Confusion Metric

Confusion metric adalah metode untuk mengukur performa sebuah model klasifikasi (Narkhede, 2018b). Ada beberapa variabel penentu yang digunakan untuk mengukur performa yaitu:

- i. True Positive (TP)

True positive adalah variabel jumlah prediksi yang dianggap benar dan memang benar.

- ii. True Negative (TN)

True negative adalah variabel jumlah prediksi yang dianggap salah dan memang salah.

iii. False Positive (FP)

False positive adalah variabel jumlah prediksi yang dianggap benar namun sebenarnya salah.

iv. False Negative (FN)

False negative adalah variabel jumlah prediksi yang dianggap salah namun sebenarnya benar.

Dari keempat variabel diatas ada beberapa informasi yang dapat dihasilkan berkaitan dengan performa sebuah model seperti dijelaskan dibawah.

i. Akurasi

Keakuratan sebuah model dalam memprediksi bisa didapatkan dari jumlah prediksi yang benar sehingga bisa dihitung dengan rumus dibawah.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

ii. Recall

Untuk mengetahui seberapa sensitif sebuah model dalam memprediksi bisa dilakukan dengan menghitung nilai *recall*. *Recall* akan sangat berguna untuk mengetahui performa model dalam mendapatkan data yang benar oleh karena itu *recall* juga dikenal dengan sebutan *true positive rate*, rumusnya seperti dibawah.

$$\frac{TP}{TP + FN}$$

iii. Precision

Menghitung nilai *precision* sebuah model sangat berguna untuk mengetahui seberapa relevan prediksi yang dihasilkan. Dimana untuk menghitung nilai presisi bisa dilakukan dengan cara dibawah.

$$\frac{TP}{TP + FP}$$

iv. Pengukuran F1

Nilai *recall* dan *precision* pada sebuah model sulit dibandingkan, jika nilai *recall* tinggi maka nilai *precision* akan rendah begitu juga sebaliknya. Untuk mengetahui persentase ketepatan dan relevansi sebuah model dalam memprediksi bisa menggunakan nilai F1, F1 adalah penggabungan antara *precision* dan *recall* dimana rumusnya sebagai berikut:

$$2 \times \frac{recall \times precision}{recall + precision}$$

v. Kurva ROC dan AUC

Kurva ROC (*Receiver Operating Characteristic*) adalah yang paling umum digunakan untuk mengukur performa sebuah model klasifikasi dengan beberapa parameter *threshold*. Kurva ini didapatkan dari hasil memetakan nilai *true positive rate* (TPR) pada axis-y dan memetakan *false positive rate* (FPR) pada aksis-x, kurva ROC yang baik adalah yang memiliki nilai AUC (*Area Under the Curve*) mendekati nilai 1 karena semakin tinggi nilai AUC maka semakin baik sebuah model dalam memprediksi (Narkhede, 2018)

BAB III

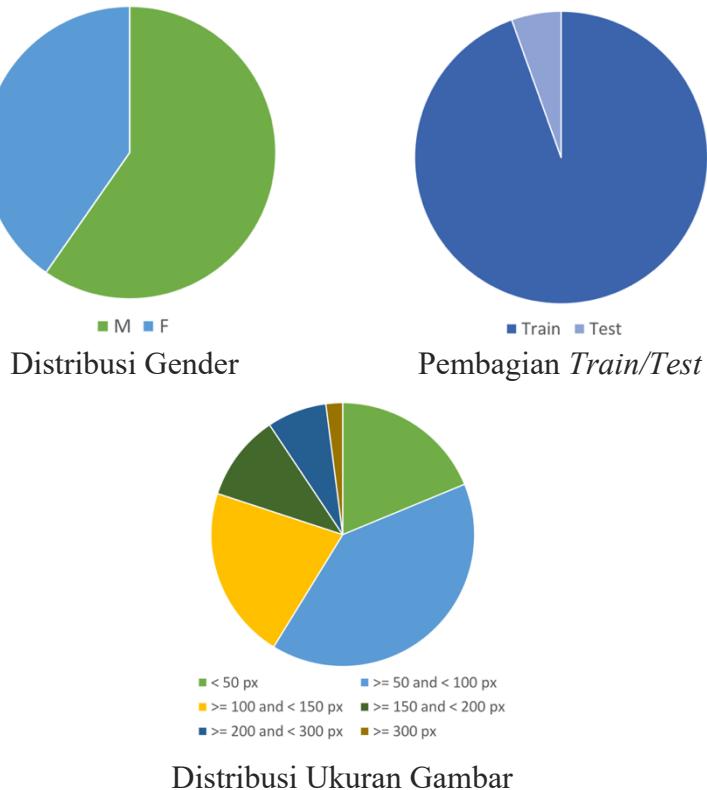
METODOLOGI PENELITIAN

3.1. Dataset

Pada penelitian ini, dataset yang akan digunakan sebagai masukan (*input*) kedalam arsitektur jaringan adalah gambar wajah yang valid, gambar wajah yang valid yang dibutuhkan peneliti adalah gambar wajah satu orang pada setiap gambar yang mewakili gambar wajah individu tersebut, jenis/tipe gambar yang digunakan adalah jpg 3 kanal (RGB). Dataset yang digunakan peneliti diperoleh dari beberapa sumber terbuka seperti dijelaskan sebagai berikut:

3.1.1. Data Latih (*Training*)

Data *training* yang digunakan dalam penelitian ini adalah VGG *Face* 2. VGG *Face* 2 adalah dataset *face recognition* dalam jumlah sangat besar, gambar-gambar pada dataset ini didownload dari pencarian gambar Google dan memiliki banyak variasi dalam hal pose, iluminasi, etnik dan profesi (Qiong Cao, Shen, Weidi Xie, Parkhi, & Zisserman, 2018). Dataset ini disebarluaskan secara bebas menggunakan lisensi *Creative Common Attribution-ShareAlike 4.0* namun hak cipta setiap gambar yang ada pada dataset ini tetap menjadi milik pemilik asal. VGG *Face* 2 memiliki lebih dari 9000 jenis identitas dengan variasi etnis, aksen, profesi dan umur yang berbeda dan memiliki 3,3 juta gambar wajah dengan berbagai keadaan latar belakang, pose, ekspresi wajah dan pencahayaan yang berbeda-beda dengan rata-rata 362 gambar wajah pada setiap subjek (Qiong Cao et al, 2018). Grafik distribusi gambar dan contoh gambar pada dataset ini bisa dilihat pada gambar 3.1 dan gambar 3.2 dibawah.



Gambar 3.1 Grafik Distribusi Dataset VGG Face v2¹



Gambar 3.2 Contoh Gambar Dataset VGG Face v2²

¹ www.robots.ox.ac.uk/~vgg/data/vgg_face2.

² Loc.cit.

Dataset VGG *Face* v2 disusun menjadi beberapa folder sesuai jumlah individu yang dijelaskan diatas, dimana setiap folder adalah label/*class*. Struktur folder pada dataset VGG *Face* v2 disusun seperti gambar 3.3 dibawah.

```

VGGV2
├── n000038
├── n000052
├── n000071
├── n000124
├── n000154
├── n000201
├── n000234
├── n000288
├── n000296
├── n000299
├── n000334
├── n000345
├── n000403
├── n000411
├── n000566
├── n000588
├── n000645
├── n000711
├── n000721
├── n000724
├── n000727
└── n000742

```

Gambar 3.3 Struktur Folder Dataset VGG *Face* v2

3.1.2. Data Uji (*Testing*)

Data *testing* yang akan digunakan pada penelitian ini adalah dataset LFW (*Labeled Face in the Wild*). Dataset ini disediakan oleh University of Massachusetts, motivasi utama dari dataset ini adalah untuk menyediakan gambar wajah yang bervariasi sesuai dengan yang ditemukan pada kehidupan sehari-hari dengan beragam variasi seperti pose, pencahayaan, ekspresi, latar belakang, ras, etnis, umur, jenis kelamin, pakaian, gaya rambut, kualitas kamera, saturasi warna, fokus gambar dan parameter lainnya (Huang, Ramesh, Berg, & Learned-Miller, 2008).

Dataset ini berisi 13,233 gambar, beberapa gambar ada yang memiliki lebih dari 1 wajah namun wajah yang akan diambil adalah wajah yang mendominasi bagian tengah gambar, yang lainnya akan dianggap sebagai latar belakang.

Dataset LFW memiliki 5749 jenis individu, 1680 diantaranya memiliki lebih dari 1 gambar dan sisanya (4069) hanya memiliki satu gambar, gambar tersedia dalam format JPEG 3 kanal warna dan ada beberapa gambar berwarna abu (*grayscale*). Gambar struktur folder dan contoh dataset LFW bisa dilihat pada gambar 3.4 dan 3.5 dibawah.

```

LFW
├── AJ_Cook
├── AJ_Lamas
├── Aaron_Eckhart
├── Aaron_Guiel
├── Aaron_Patterson
├── Aaron_Peirsol
├── Aaron_Pena
├── Aaron_Sorkin
├── Aaron_Tippin
├── Abba_Eban
├── Abbas_Kiarostami
├── Abdel_Aziz_Al-Hakim
├── Abdel_Madi_Shabneh
├── Abdel_Nasser_Assidi
├── Abdoulaye_Wade
├── Abdul_Majeed_Shobokshi
└── Abdul_Rahman

```

Gambar 3.4 Struktur Folder Dataset LFW



Gambar 3.5 Contoh Gambar Dataset LFW³

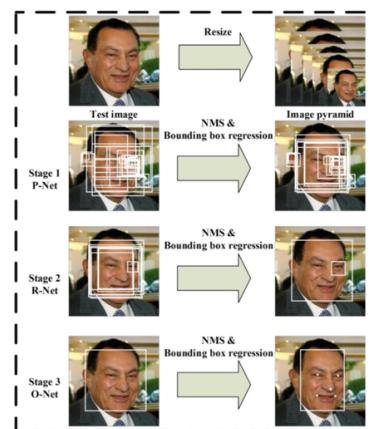
³ www.cs.umass.edu/lfw/sets1.html.

3.1.3. MTCNN

Sebelum dataset digunakan untuk keperluan *training* maupun *testing* akan dilakukan persiapan terlebih dahulu. Persiapan ini sangat penting untuk memastikan dataset yang digunakan adalah dataset yang benar-benar valid sesuai kebutuhan training (poin 1 dataset). Adapun persiapan dataset yang digunakan pada penelitian ini adalah *Multi-task Cascaded Convolutional Networks* (MTCNN).

MTCNN adalah *state of the art (sota)* saat ini dalam mencari 5 titik wajah (*face landmark*) yaitu kedua mata (kiri dan kanan), hidung dan kedua ujung bibir samping kiri dan samping kanan (Zhang et al., 2016). MTCNN pada penelitian ini digunakan untuk memotong (krop) secara otomatis bagian wajah pada setiap gambar dataset dengan ukuran tertentu (penelitian ini menggunakan ukuran 182x182).

Adapun *framework* ini melalui 3 tahap CNN untuk melakukan proses pendekripsi *face landmark* seperti ditunjukkan pada gambar 3.6 dibawah.



Gambar 3.6 *Pipeline* MTCNN Dalam Menghasilkan *Face Landmark*⁴

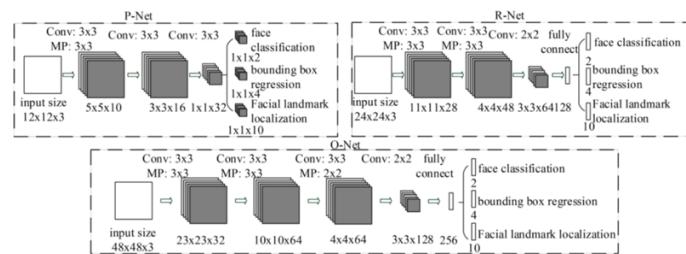
Pada gambar 3.6 diatas gambar terlebih dahulu di-*resize* menjadi beberapa ukuran (*image pyramid*) sebelum diteruskan ke 3 buah tahap CNN yaitu:

⁴ Zhang, K., Zhang et al. (2016). Joint Face Detection and Alignment using Multi-ta, hal 2.

Tahap 1: gambar yang telah di resize menjadi beberapa ukuran tertentu sebelumnya diteruskan ke arsitektur CNN yang disebut P-Net (*Proposal Network*). Fungsi dari jaringan ini adalah untuk mendapatkan beberapa *bounding box* (kotak batas sebagai acuan krop) lalu mengkalibrasinya agar mendapatkan hasil yang lebih cocok menggunakan estimated *bounding box regression vectors*, kemudian dari beberapa *bounding box* yang telah didapatkan dilakukan penggabungan *bounding box* menggunakan *Non-Maxima Suppression* (NMS) pada *bounding box* yang bertumpukan dengan sangat rapat.

Tahap 2: setiap calon yang didapatkan pada tahap 1 kemudian diteruskan ke arsitektur CNN yang lain diberi nama *Refine Network* (F-Net). Fungsi dari jaringan ini adalah mengeliminasi calon yang salah (prediksi wajah yang tidak benar) lalu melakukan kalibrasi lagi dan melakukan proses NMS lagi.

Tahap 3: proses ini sama dengan tahap 2, namun pada tahap ini arsitektur jaringan ini akan mendeskripsikan lebih detail lagi *bounding box* yang didapatkan pada tahap 2 dan mengeluarkan *face landmark*. Arsitektur jaringan dari 3 tahap ini bisa dilihat pada gambar 3.7 dibawah.



Gambar 3.7 Arsitektur MTCNN⁵

3.1.4. Preprocessing

Pada tahap *preprocessing*, data *training* maupun *testing* akan melalui beberapa perubahan (*transformasi*) agar dapat diterima dan bisa diproses sesuai desain arsitektur jaringan yang digunakan. Data

⁵ Ibid, hal 3.

training dan data *testing* akan melalui tahap proses transformasi yang berbeda sesuai yang dijelaskan pada poin-poin dibawah.

i. Transformasi data *training*

Tahap transformasi pada *batch* data *training* akan melalui beberapa proses *transformasi* acak ini bertujuan untuk menambah variasi gambar (data augmentasi), beberapa *transformasi* acak yang dilakukan adalah sebagai berikut:

a. *Random Rotation*

Transformasi ini akan merotasi gambar secara acak sesuai sudut derajat yang ditentukan. Pada penelitian ini sudut derajat yang digunakan adalah 15° .

b. *Random Resized Crop*

Transformasi ini akan melakukan operasi krop pada gambar dengan ukuran yang bervariasi dari 8% sampai 100% persen dengan aspek rasio $\frac{3}{4}$ atau $4/3$ dari ukuran gambar aslinya secara acak. Kemudian gambar yang telah dikrop akan diresize ke ukuran yang sudah ditentukan. Ukuran *resize* yang digunakan pada penelitian ini adalah 224 karena arsitektur jaringan yang digunakan membutuhkan gambar dengan ukuran 224x224 sebagai masukan.

c. *Random Horizontal Flip*

Transformasi ini berfungsi untuk membalik gambar pada arah horizontal secara acak. Proses pembalikan akan dilakukan dengan menentukan batas probabilitas gambar akan dibalik, lalu memilih angka secara acak diantara 0 sampai 1, jika angka acak yang didapatkan lebih kecil dari batas probabilitas yang telah ditentukan maka proses pembalikan horizontal akan dilakukan. Pada penelitian ini batas probabilitas yang digunakan adalah nilai *default* yaitu 0.5.

ii. Transformasi data *testing*

Transformasi pada data *testing* sama namun sedikit berbeda dengan transformasi yang dilakukan pada data *training*, jika pada

data *training* menggunakan beberapa transformasi acak untuk keperluan augmentasi data, pada data *testing* transformasi acak tidak dilakukan yang akan dilakukan hanyalah merubah (transformasi) data *testing* agar dapat diterima sebagai masukan pada arsitektur jaringan. Dibawah adalah transformasi yang dilakukan pada data *testing*:

a. *Resize*

Transformasi *resize* adalah proses merubah ukuran gambar dengan ukuran tertentu menggunakan algoritma interpolasi bilinear. Pada penelitian ini ukuran *resize* gambar yang digunakan adalah 224x224 sesuai kebutuhan arsitektur jaringan yang digunakan.

b. *Center Crop*

Sesuai namanya, proses transformasi ini akan memotong gambar pada posisi tengah dengan ukuran tertentu. Ukuran yang digunakan pada penelitian ini adalah 224x224 sesuai kebutuhan arsitektur jaringan yang digunakan.

iii. Normalisasi Gambar

Normalisasi data pada umumnya dilakukan agar setiap data berada pada skala distribusi tertentu. Namun pada kasus gambar digital sebenarnya normalisasi tidak dibutuhkan lagi (Andrej Karpathy, 2017) karena distribusi setiap pixel sudah berada pada rentang tertentu yaitu 0-255 atau biasa disebut skala keabuan, namun pada penelitian ini normalisasi data tetap dilakukan karena alasan spesifikasi arsitektur jaringan yang digunakan, dijelaskan lebih rinci pada sub bab 3.2.1 poin ii dibawah. Proses normalisasi data setiap pixel diperoleh melalui perhitungan *z-score* setiap pixel:

$$f(x) = \frac{x - \text{mean}}{\text{std}}$$

Dimana:

$f(x)$ = hasil normalisasi.

x = nilai pixel yang akan dinormalkan.

$mean$ = nilai rata-rata.

std = nilai simpangan baku (*standard deviation*).

Pada penelitian ini, nilai $mean$ dan std telah ditentukan sesuai ketentuan arsitektur jaringan yang digunakan sesuai tabel 3.1 dan tabel 3.2 dibawah.

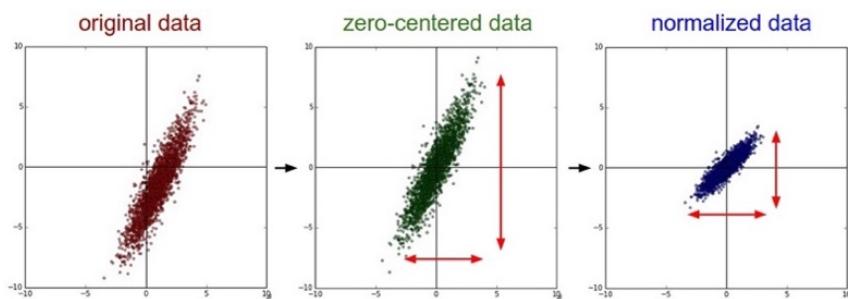
Tabel 3.1 Ketentuan Nilai *Mean* Yang Digunakan

Nilai <i>mean</i> masing-masing kanal warna	
R	0.485
G	0.456
B	0.406

Tabel 3.2 Ketentuan Nilai *Std* Yang Digunakan

Nilai <i>std</i> masing-masing kanal warna	
R	0.229
G	0.224
B	0.225

Persamaan diatas secara intuisi bisa dijelaskan seperti gambar 3.8 dibawah.



Gambar 3.8 Visualisasi Normalisasi Data⁶

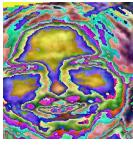
⁶ A. Karpathy, “Cs231n convolutional neural networks for visual recognition.” (Stanford University).

Komponen pertama pada persamaan diatas yaitu $x - mean$ bertujuan untuk membuat setiap data *point* (nilai setiap pixel) terdistribusi secara relatif ke tengah. Lalu komponen kedua yaitu pembagian dengan nilai *std* untuk mengurangi *skewness* (kemerongan) pada data sehingga panjang dan tinggi data terdistribusi secara merata seperti divisualisasikan pada gambar 3.8 paling kanan diatas.

iv. Transformasi Gambar Ke Tensor

Tahap ini akan mengubah gambar menjadi *tensor* dengan nilai setiap elemen adalah nilai skala keabuan pada setiap pixel gambar. Ilustrasi hasil pemrosesan data pertahap bisa dilihat pada tabel 3.3 dan 3.4 dibawah dimulai dari gambar *raw* (gambar mentah) sebelum diproses kemudian dilanjutkan ke kanan sampai pada proses *data normalization*, hasil transformasi gambar ke tensor bisa dilihat pada gambar 3.9 dibawah.

Tabel 3.3 Ilustrasi *Preprocessing* Gambar Untuk *Training*

Raw input	MTCNN 182x182	<i>Random Rotation</i> 15°	<i>Random Resized Crop</i> 224x224	<i>Random Horizontal Flip</i>	Data normalization
					

Tabel 3.4 Ilustrasi *Preprocessing* Gambar Untuk *Testing*

Raw input	MTCNN 182x182	<i>Resize</i> 224x224	<i>Center Crop</i> 224x224	Data normalization
				

```

tensor([[[0.2078, 0.2157, 0.2275, ..., 0.4118, 0.4314, 0.4471],
        [0.2039, 0.2118, 0.2235, ..., 0.4157, 0.4392, 0.4549],
        [0.2000, 0.2039, 0.2157, ..., 0.4235, 0.4510, 0.4667],
        ...,
        [0.0706, 0.0667, 0.0510, ..., 0.0353, 0.0471, 0.0510],
        [0.0941, 0.0824, 0.0627, ..., 0.0431, 0.0588, 0.0667],
        [0.1098, 0.0941, 0.0706, ..., 0.0471, 0.0627, 0.0745]],

       [[0.2157, 0.2235, 0.2353, ..., 0.3137, 0.3333, 0.3490],
        [0.2118, 0.2196, 0.2314, ..., 0.3137, 0.3373, 0.3529],
        [0.2078, 0.2118, 0.2235, ..., 0.3137, 0.3412, 0.3569],
        ...,
        [0.0745, 0.0745, 0.0667, ..., 0.0000, 0.0000, 0.0000],
        [0.0980, 0.0902, 0.0784, ..., 0.0000, 0.0078, 0.0118],
        [0.1137, 0.1020, 0.0863, ..., 0.0000, 0.0118, 0.0157]],

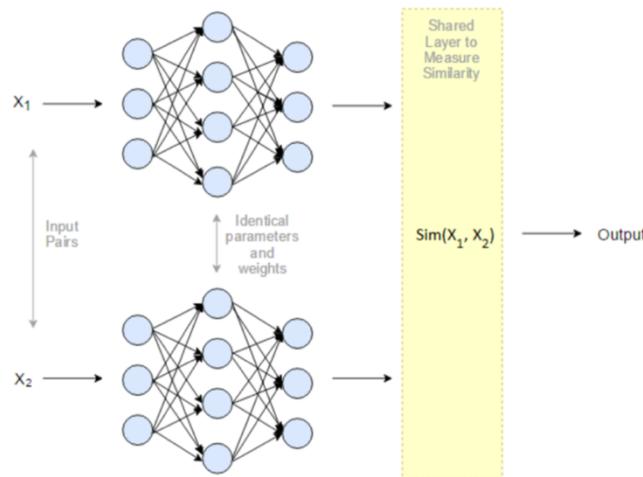
       [[0.1961, 0.2039, 0.2157, ..., 0.1569, 0.1765, 0.1922],
        [0.1922, 0.2000, 0.2118, ..., 0.1569, 0.1804, 0.1961],
        [0.1882, 0.1922, 0.2039, ..., 0.1569, 0.1843, 0.2000],
        ...,
        [0.0824, 0.0784, 0.0706, ..., 0.0431, 0.0588, 0.0667],
        [0.1059, 0.0980, 0.0824, ..., 0.0549, 0.0706, 0.0824],
        [0.1216, 0.1098, 0.0902, ..., 0.0627, 0.0784, 0.0902]]])

```

Gambar 3.9 Transformasi Gambar Ke Tensor

3.2. Siamese

Arsitektur *siamese* pertama kali diperkenalkan oleh Jane Bromley, Yan LeCun et al pada tahun 1994. Arsitektur jaringan ini pada saat itu digunakan untuk menyelesaikan masalah verifikasi tanda tangan dengan capaian akurasi sampai 95%. Konsep utama dari jaringan ini adalah agar melewatkkan pasangan *input* melalui jaringan yang identik (memiliki nilai parameter yang sama). Contoh arsitektur jaringan *Siamese* bisa dilihat pada gambar 3.10 dibawah.



Gambar 3.10 Contoh Jaringan *Siamese*⁷

Pada gambar 3.10 diatas *input* x_1 dan x_2 melewati dua jaringan yang sama (identik) yang memiliki nilai parameter yang sama, hasil dari jaringan ini adalah *feature vectors/embedding* dari input x_1 dan x_2 , *embedding* dari kedua *input* ini kemudian bisa dibandingkan kedekatannya dengan fungsi $\text{sim}(f(x_1), f(x_2))$ dimana fungsi **sim** bisa berbentuk *Euclidean distance* atau *cosine similarity* sehingga dengan *threshold* tertentu kedua *input* bisa ditentukan apakah sama (*genuine*) atau berbeda (*impostor*). Proses pemetaan dimensi gambar yang tinggi menjadi representasi berdimensi lebih rendah/kecil (*embedding/feature vectors*) juga disebut dengan *dimensionality reduction*. Ada beberapa algoritma yang bisa digunakan dalam proses *dimensionality reduction*, dalam penelitian ini penulis menggunakan algoritma *triplet loss* yang akan dijelaskan lebih rinci pada poin 3.3.1.

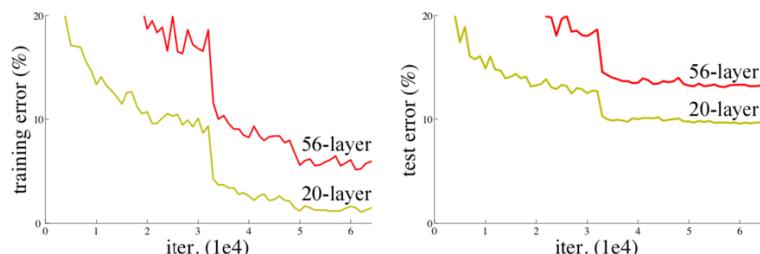
Arsitektur jaringan identik yang dapat digunakan *Siamese* bisa bermacam-macam. Merancang arsitektur jaringan sendiri adalah proses yang rumit dan membutuhkan banyak percobaan dan memakan banyak waktu. Penulis dalam melakukan penelitian ini memanfaatkan arsitektur jaringan Resnet karena kehandalannya dalam melakukan klasifikasi dan ekstraksi fitur

⁷ Martin et al. (2017). A convolutional siamese network for developing similarity knowledge in the SelfBACK dataset. *CEUR Workshop Proceedings*, 2028, 85–94, hal 3.

sebuah gambar, arsitektur ini mendapatkan posisi pertama pada kompetisi *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) 2015 dengan kesalahan (*error*) hanya 3.57% pada dataset *ImageNet*.

3.2.1. Residual Network (ResNet)

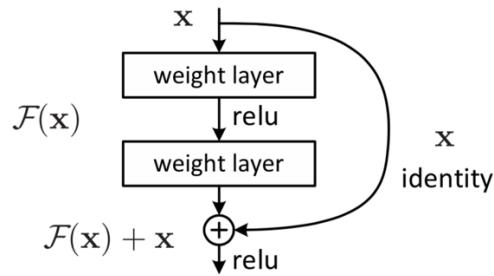
Arsitektur CNN yang cukup dalam telah berhasil membuat terobosan-terobosan dalam masalah klasifikasi gambar, telah banyak masalah-masalah klasifikasi gambar yang rumit memanfaatkan arsitektur model yang sangat dalam. Namun semakin dalam arsitektur sebuah jaringan semakin susah untuk dilatih dan ketika arsitektur yang sangat dalam berhasil *convergent* masalah baru seperti *degradation* muncul, dimana akurasi jaringan tersebut menurun secara cepat dan kesalahannya pun meningkat (He, Zhang, Ren, & Sun, 2015), seperti terlihat pada gambar 3.11 dibawah.



Gambar 3.11⁸ Error Lebih Tinggi Pada Arsitektur Yang Lebih Dalam.

Mengatasi masalah ini kerangka arsitektur *deep residual learning* pun dibuat yang dikenal dengan ResNet (*Residual Network*). Kerangka *residual* ini memperkenalkan sambungan *shortcut* seperti gambar 3.12 dibawah.

⁸ He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition, hal 1.

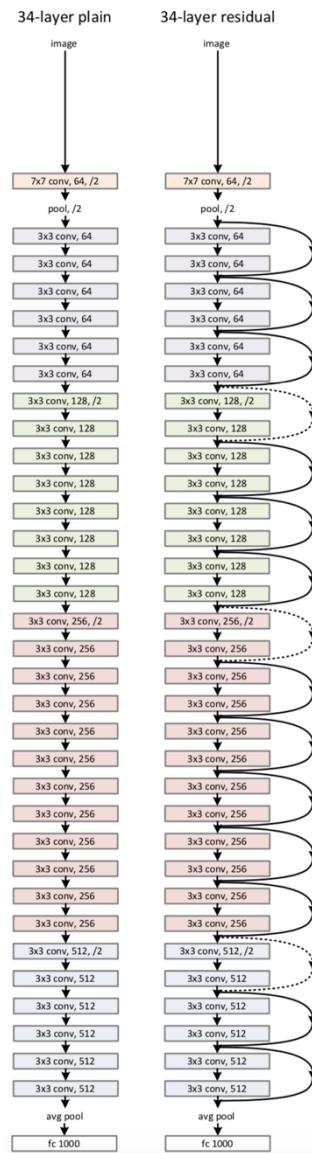


Gambar 3.12⁹ Blok *Residual* Dengan Sambungan *Shortcut*.

Masalah degradasi (akurasi menurun dan *error* meningkat) pada arsitektur jaringan yang sangat dalam ditemukan ketika pemetaan fungsi dari blok CNN_i ($H_i = f_i(x_i)$) bernilai 0 sehingga sangat sulit untuk menentukan parameter yang akan dipelajari pada blok CNN_{i+1} ($f_{i+1}(H_i)$) karena tidak ada informasi yang dilanjutkan ke f_{i+1} yang disebabkan oleh nilai $H_i = 0$. Disinilah peran *residual learning*, dengan melanjutkan inputan x_i ke ujung dari blok CNN_i sehingga fungsi CNN_i menjadi $H_i = f_i(x_i) + x_i$, dengan demikian walaupun pemetaan fungsi $f_i(x_i) = 0$ namun informasi dari x_i masih utuh sehingga nilai $H_i = x_i$ (*identity mapping*), dengan demikian fungsi f_{i+1} tidak lagi memproses nilai 0 melainkan memproses nilai inputan yang identik dengan blok f_i .

Proses *identity mapping* dalam blok ResNet diwujudkan dengan menambah garis *shortcut* (*skip connection*) langsung menuju ujung dari blok CNN_i seperti terlihat pada gambar 3.12 diatas. Ini menjelaskan arsitektur ResNet mampu mengatasi masalah *degradation* pada arsitektur jaringan CNN yang sangat dalam. Contoh perbandingan arsitektur jaringan tanpa *residual block* (*plain*) dengan *residual block* dapat dilihat pada gambar 3.13 dibawah.

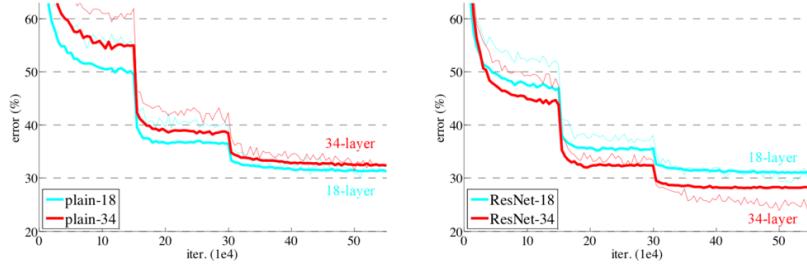
⁹ Ibid, hal 2.



Gambar 3.13¹⁰ Arsitektur Jaringan *Plain* dan *Residual Block*

Perbandingan pembelajaran yang dihasilkan oleh jaringan *plain* dan jaringan dengan tambahan *residual block* dapat dilihat pada gambar 3.14 dibawah.

¹⁰ Ibid, hal 4.



Gambar 3.14¹¹ Perbandingan Kesalahan ResNet dan *Plain*

Bisa dilihat pada gambar grafik 3.14 diatas jaringan yang telah ditambahkan *residual block* dapat meraih *error* yang lebih rendah dengan arsitektur yang lebih dalam.

i. ResNet-50

Adapun arsitektur jaringan ResNet ada beberapa macam seperti terlihat pada gambar 3.15 dibawah.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer	
conv1	112×112			7×7, 64, stride 2			
conv2.x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	
conv3.x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$	
conv4.x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$	
conv5.x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	
	1×1			average pool, 1000-d fc, softmax			
		FLOPs	1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Gambar 3.15¹² Beberapa Jenis Arsitektur ResNet

Seperti terlihat pada gambar 3.15 diatas ResNet memiliki 5 jenis arsitektur masing-masing mewakili jumlah layer yang dimiliki. Tentunya semakin banyak layer yang dimiliki maka bertambah juga jumlah parameter yang dimiliki. Perbandingan jumlah parameter setiap model ResNet bisa dilihat pada tabel 3.5 dibawah.

¹¹ Ibid, hal 5.

¹² Loc.cit.

Tabel 3.5 Perbandingan Jumlah Parameter Model ResNet

Model	Jumlah Parameter
ResNet-18	11.689.512
ResNet-34	21.797.672
ResNet-50	25.557.032
ResNet-101	44.549.160
ResNet-152	60.192.808

Dalam penelitian ini penulis menggunakan model ResNet-50 dengan alasan jumlah parameter yang cukup banyak namun tidak sampai memakan sumber daya *hardware* yang besar seperti model diatasnya (ResNet-101 dan ResNet-152), adapun model ResNet-50 yang digunakan akan dilakukan beberapa modifikasi layer sehingga jumlah parameter ResNet-50 menjadi 36.353.216, adapun modifikasi layer yang dilakukan akan dijelaskan pada poin selanjutnya.

ii. Konfigurasi ResNet

Model ResNet yang digunakan adalah *pretrained model* yaitu model yang sebelumnya telah dilatih dalam klasifikasi objek pada dataset ImageNet. Hasil parameter yang telah dilatih pada dataset ini kemudian dimuat lalu digunakan untuk melatih dataset baru yang digunakan pada penelitian ini.

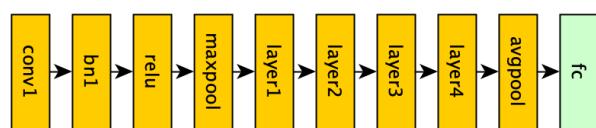
Adapun model ResNet yang telah dilatih (*pretrained*) memiliki beberapa ketentuan yang harus diikuti sebelum digunakan yaitu agar masukan gambar (*input*) harus memiliki 3 kanal warna dengan ukuran 224x224 dan setiap gambar masukan harus mengikuti data normalisasi yang sama dengan yang digunakan pada saat melatih dataset ImageNet. Tabel data normalisasi bisa dilihat pada tabel 3.1 dan 3.2 (Pytorch, 2018).

iii. Modifikasi Layer ResNet-50

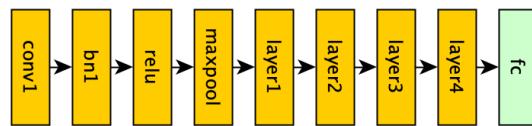
Beberapa modifikasi arsitektur ResNet-50 yang dilakukan dijelaskan sebagai berikut:

a. Pembuangan Layer Terakhir

Arsitektur jaringan ResNet-50 tanpa layer terakhir yang dilatih menggunakan *Triplet Loss* berhasil mendapatkan akurasi yang baik (Hermans, Beyer, & Leibe, 2017). Berdasarkan hal tersebut, penelitian ini mencoba menggunakan arsitektur yang sama. Layer terakhir ResNet adalah *Average Pool*. Adapun perbandingan arsitektur sebelum dan sesudah pembuangan layer terakhir bisa dilihat pada gambar 3.16 dan gambar 3.17 dibawah.



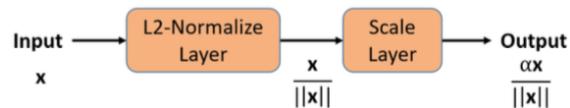
Gambar 3.16 Blok Jaringan ResNet Yang Utuh



Gambar 3.17 Blok Jaringan ResNet Tanpa Layer Terakhir

b. L2 Norm dan Skala α (*alpha*)

Penambahan L2 Norm dan skala α pada penelitian ini berdasarkan pada percobaan yang dilakukan oleh Rajeev Ranjan dkk. Rajeev menyatakan bahwa mengintegrasikan L2 Norm dan skala *alpha* pada arsitektur jaringan yang sudah ada dapat meningkatkan kinerja jaringan (Ranjan, Castillo, & Chellappa, 2017). Adapun proses yang dilakukan pada layer ini bisa dilihat pada gambar 3.18 dibawah.



Gambar 3.18 Ilustrasi Operasi L2 Norm dan Skala α

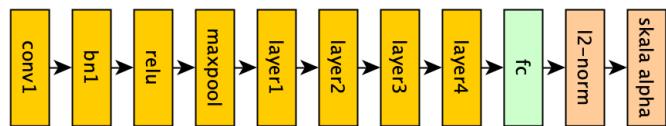
Pada gambar 3.18 input yang diterima oleh layer L2 Norm akan diproses dengan persamaan:

$$y = \frac{x}{\|x\|_2}$$

Lalu hasil dari y akan dikalikan dengan nilai α , seperti dibawah:

$$z = y \cdot \alpha$$

Sesuai arsitektur yang dirancang pada paper tersebut L2 Norm dan skala α digunakan sebagai *penultimate layer* (layer sebelum output) sehingga arsitektur ResNet yang digunakan bisa divisualisasikan seperti gambar 3.19 dibawah.



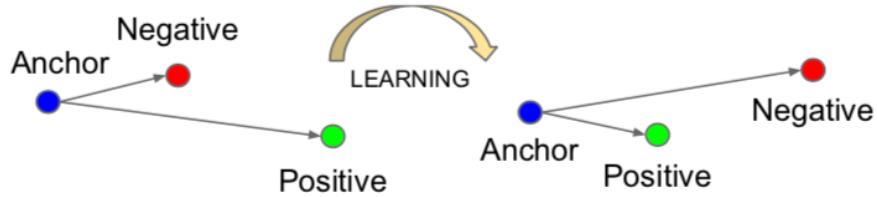
Gambar 3.19 Layer FC Setelah Penambahan L2 Norm & Skala

Adapun skala α memiliki peran penting dalam meningkatkan kinerja L2 Norm. Ada dua acara dalam menentukan nilai α , yaitu bisa dipelajari dengan melatih jaringan dengan membuat α sebagai parameter, atau dengan memberikan nilai tetep (konstanta). Dalam eksperimen Rajeev Ranjan, dengan menetapkan nilai α memberikan performa yang lebih baik. Nilai α yang digunakan pada penelitian ini adalah 10.

3.3. Training

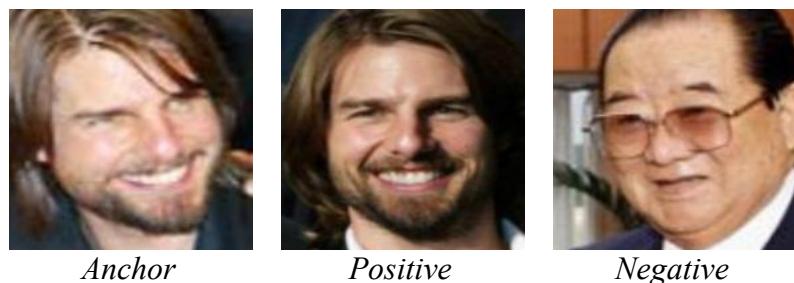
3.3.1. Triplet Loss

Triplet Loss adalah fungsi yang digunakan untuk mempelajari *embedding euclidean* dengan dua tujuan yaitu gambar wajah orang yang sama menghasilkan *embedding* yang berdekatan pada ruang *embedding* sedangkan gambar wajah orang yang berbeda menghasilkan *embedding* yang berjauhan pada ruang *embedding* (Schroff & Philbin, 2015).



Gambar 3.20 Ilustrasi Belajar *Triplet Loss*

Adapun pada saat latihan, setiap *batch* data terdapat 3 jenis gambar yaitu *anchor*, *positive* dan *negative*. Gambar *anchor* adalah gambar wajah acuan, gambar *positive* adalah gambar wajah yang sama dengan *anchor* dan gambar *negative* adalah gambar wajah yang berbeda dengan *anchor*.



Gambar 3.21 Contoh Gambar *Anchor*, *Positive* dan *Negative*

Pada saat *training*, *triplet loss* akan belajar bagaimana menghasilkan *embedding* sehingga jarak antar gambar *anchor* dan *positive* menjadi lebih dekat dan jarak antar gambar *anchor* dan gambar *negative* menjadi lebih jauh seperti terlihat pada gambar 3.20 diatas. Secara matematis tujuan pembelajaran *triplet loss* bisa diekspresikan seperti persamaan dibawah:

$$\|x_i^a - x_i^p\|_2^2 + \alpha < \|x_i^a - x_i^n\|_2^2, \forall (x_i^a, x_i^p, x_i^n) \in T$$

Dimana:

$\|x_i^a - x_i^p\|_2^2$ adalah jarak antara gambar *anchor* dan gambar *positive*.

α adalah nilai konstanta sebagai margin antara *positive* dan *negative*.

$\|x_i^a - x_i^n\|_2^2$ adalah jarak antara gambar *anchor* dan gambar *negative*.

Sehingga nilai *loss* pada setiap *minibatch* bisa dihitung dengan fungsi *triplet loss* dibawah:

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Dimana:

N adalah banyak data pada setiap *mini batch*.

f adalah hasil *embedding*.

x_i^a adalah input gambar *anchor*.

x_i^p adalah input gambar *positive*.

x_i^n adalah input gambar *negative*.

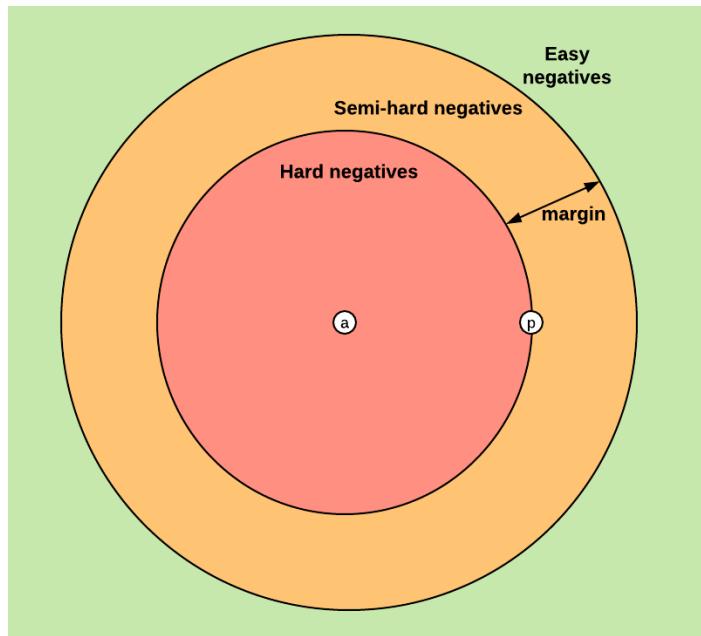
α adalah *margin*.

Menggunakan setiap pasangan *triplet* pada *minibatch* sebagai bahan latih dapat menghasilkan pembelajaran yang tidak optimal disebabkan karena ada kemungkinan pasangan *triplet* yang tidak perlu dilatih/optimasi. Sebagai contoh gambar *anchor* dan *positive* yang memang sudah dekat atau gambar *anchor* dan *negative* yang memang sudah jauh tidak perlu dimasukkan sebagai bahan latihan *triplet loss* karena tidak memiliki kontribusi yang banyak pada hasil latihan jaringan dan dapat menyebabkan *convergent* yang lama (Schroff, Kalenichenko, & Philbin, 2015). Sehingga dibutuhkan strategi dalam pemilihan pasangan *triplet* yang baik, seperti dijelaskan pada poin selanjutnya.

3.3.2. Pemilihan Triplet¹³

Untuk mendapatkan hasil *embedding* yang optimal pasangan *triplet* yang digunakan untuk latihan harus baik, adapun beberapa jenis *triplet* adalah sebagai berikut:

¹³ Moindrot, O. (2018). Triplet Loss and Online Triplet Mining in TensorFlow | Olivier Moindrot blog. Retrieved June 28, 2019, from <https://omoindrot.github.io/triplet-loss#triplet-loss-and-triplet-mining>



Gambar 3.22 Ilustrasi Pembagian Jenis *Triplet*

i. Easy Triplet

Easy triplet adalah ketika jarak antar gambar *anchor* lebih dekat dengan gambar *positive* dibanding jarak ke gambar *negative*, *triplet* dengan jenis ini memiliki nilai *loss* 0 dan tidak perlu dioptimasi lagi.

$$d(f(x^a), f(x^p)) + \alpha < d(f(x^a), f(x^n))$$

Dimana:

d adalah jarak antar dua gambar.

f adalah *embedding* dan α adalah nilai *margin*.

ii. Hard Triplet

Hard triplet adalah ketika jarak antar gambar *anchor* lebih dekat ke gambar *negative* daripada ke gambar *positive*.

$$d(f(x^a), f(x^n)) < d(f(x^a), f(x^p))$$

iii. Semi Hard Triplet

Semi hard triplet adalah ketika jarak antar gambar *anchor* dan *negative* masih didalam radius jarak *anchor positive* + *margin*.

$$d(f(x^a), f(x^p)) < d(f(x^a), f(x^n)) < d(f(x^a), f(x^p)) + \alpha$$

Adapun ketiga jenis *triplet* diatas bergantung pada posisi gambar *negative*, sehingga ketiga jenis triplet diatas bisa diberi istilah *easy negative*, *hard negative* dan *semi hard negative* dan bisa diilustrasikan dengan bidang lingkaran dimana jarak antar *embedding* gambar menjadi radius seperti terlihat pada gambar 3.22 diatas.

Pada penelitian ini jenis *triplet* yang digunakan untuk *training* adalah *semi-hard negative triplet* dan menggunakan optimasi Adam. Contoh gambar yang termasuk *easy* dan *hard negative* bisa dilihat pada table 3.6 dibawah.

Tabel 3.6 Contoh Jenis Triplet

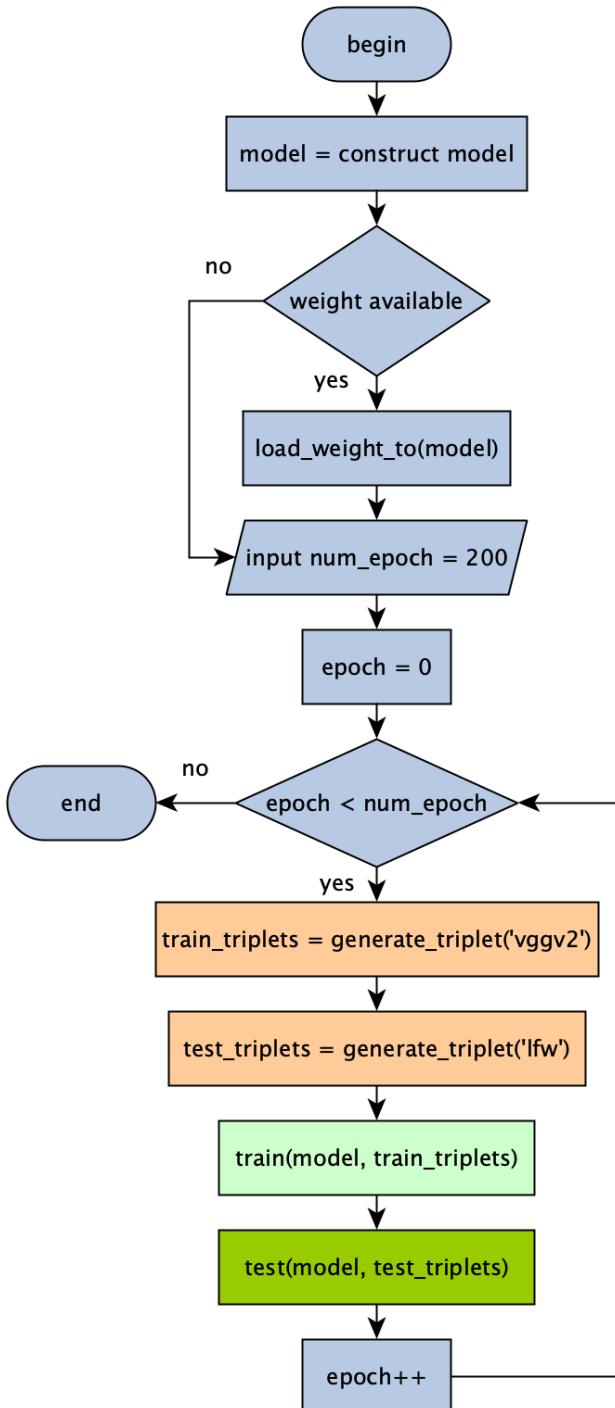
x^a	x^p	x^n	$d(a, p)$	$d(a, n)$	Kategori
			10,5	10,9	Easy negative
			11,26	9,01	Hard negative

3.3.3. Proses Training dan Testing

Setelah menjelaskan tahap-tahap pengumpulan dataset, persiapan dataset dan pembagian jenis-jenis triplet, pada sub bab ini akan dijelaskan tahap-tahap proses training dan testing sebagai berikut.

i. Gambaran Umum Training dan Testing

Secara umum proses training dan testing setiap *epoch* terlihat seperti gambar 3.23 dibawah.



Gambar 3.23 Gambaran Besar Proses *Training* dan *Testing*

Pada tahap konstruksi model akan dilakukan pembuatan objek model yang akan dilatih yaitu model ResNet-50 yang telah dimodifikasi seperti digambarkan pada gambar 3.17, lalu jika

sebelumnya sudah dilakukan proses training maka parameter (*weight* dan *bias*) yang telah latih sebelumnya akan dimuat ke model yang digunakan. Adapun rincian masukan dan keluaran setiap layer pada model tersebut akan dijelaskan melalui tabel 3.7 dibawah.

Tabel 3.7 Rincian Input dan Output Setiap Layer

Layer	Input	Proses	Output
Conv1	1x3x224x224	Conv2D	1x64x112x112
BN1	1x64x112x112	BatchNorm	1x64x112x112
ReLU	1x64x112x112	ReLU	1x64x112x112
MaxPool	1x64x112x112	MaxPool	1x64x56x56
Layer1	1x64x56x56	Bottleneck1	1x256x56x56
	1x256x56x56	Bottleneck2	1x256x56x56
	1x256x56x56	Bottleneck2	1x256x56x56
Layer2	1x256x56x56	Bottleneck3	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
Layer3	1x512x28x28	Bottleneck5	1x1024x14x14
	1x1024x14x14	Bottleneck6	1x1024x14x14
Layer4	1x1024x14x14	Bottleneck7	1x2048x7x7
	1x2048x7x7	Bottleneck8	1x2048x7x7
	1x2048x7x7	Bottleneck8	1x2048x7x7
FC	1x100,352	Linear	1x128

Pada tabel 3.7 diatas dirincikan masukan, proses dan keluaran setiap layer pada arsitektur ResNet-50 yang telah dimodifikasi. Sebagai contoh layer *conv1* menerima input *tensor* dengan dimensi $1 \times 3 \times 224 \times 224$ lalu dilakukan operasi *conv2d* dan menghasilkan *tensor* dengan dimensi $1 \times 64 \times 112 \times 112$. Perlu diketahui angka 1 pada dimensi tensor tersebut adalah jumlah data. Sebagai contoh, jika saat training *batch size* yang digunakan adalah 64 maka jumlah data yang akan diproses adalah 64 sehingga dimensi tensor akan menjadi $64 \times 3 \times 224 \times 224$ dan keluarannya

akan menjadi $64 \times 64 \times 112 \times 112$. Adapun arsitektur setiap proses pada tabel 3.7 di atas dirincikan pada tabel 3.8 dibawah.

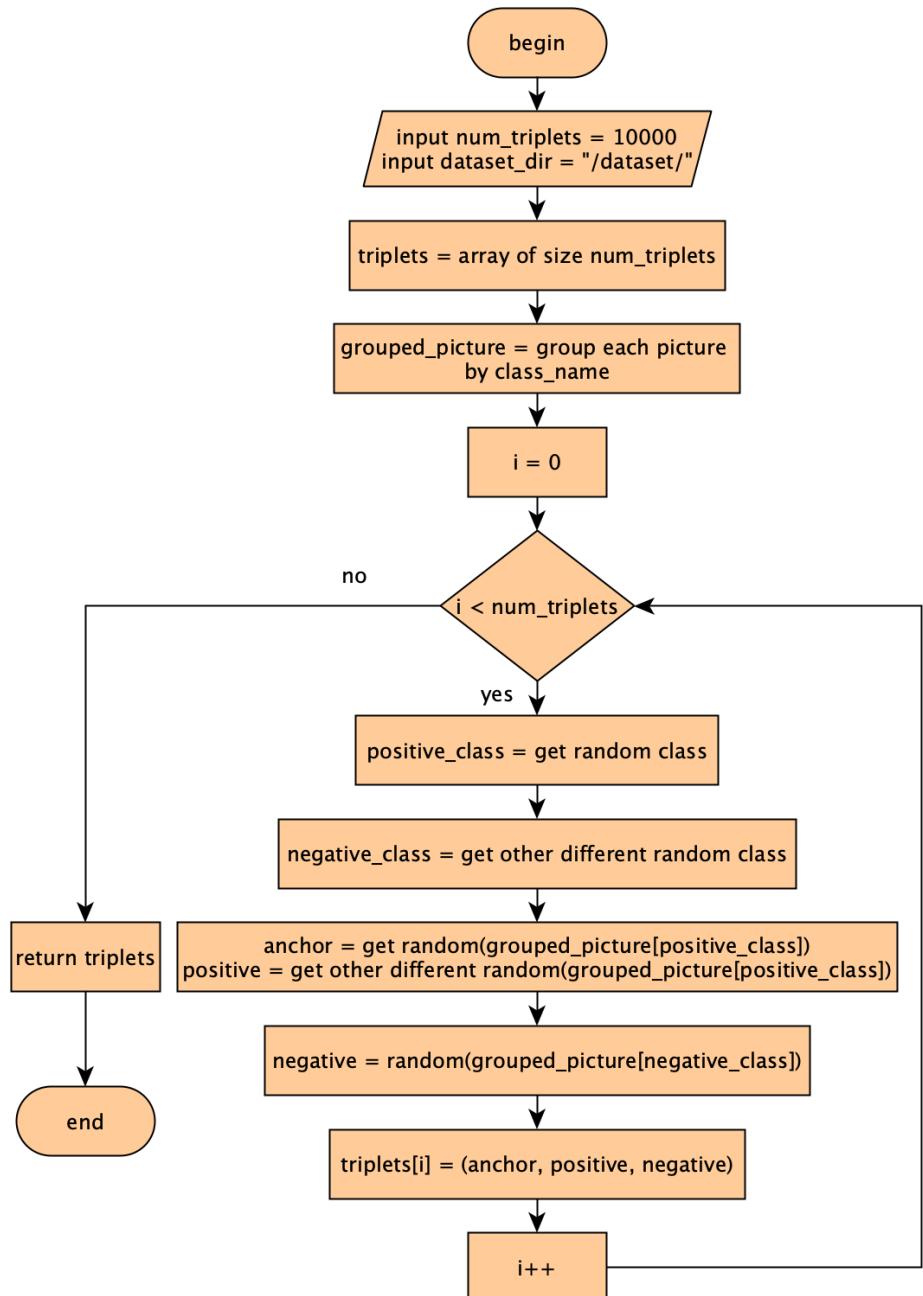
Tabel 3.8 Rincian Arsitektur Setiap Operasi Konvolusi

Nama	Operasi	Kernel	Stride	Padding
Conv2D	Conv2D	64x7x7	2x2	3x3
MaxPool	MaxPool	3x3	2	1
Bottleneck1	Conv2D	64x1x1	1x1	-
	BN	-	-	-
	Conv2D	64x3x3	1x2	1x1
	BN	-	-	-
	Conv2D	256x1x1	1x1	-
	BN	-	-	-
	ReLU	-	-	-
	Conv2D	256x1x1	1x1	1x1
	BN	-	-	-
	Conv2D	64x1x1	1x1	1x1
Bottleneck2	BN	-	-	-
	Conv2D	64x3x3	1x1	1x1
	BN	-	-	-
	Conv2D	256x1x1	1x1	1x1
	BN	-	-	-
	ReLU	-	-	-
	Conv2D	128x1x1	1x1	-
Bottleneck3	BN	-	-	-
	Conv2D	128x3x3	2x2	1x1
	BN	-	-	-
	Conv2D	512x1x1	1x1	-
	BN	-	-	-
	ReLU	-	-	-
	Conv2D	512x1x1	2x2	-
	BN	-	-	-
Bottleneck4	Conv2D	128x1x1	1x1	-
	BN	-	-	-
	Conv2D	128x3x3	1x1	1x1
	BN	-	-	-
	Conv2D	512x1x1	1x1	-
	BN	-	-	-
	ReLU	-	-	-
Bottleneck5	Conv2D	256x1x1	1x1	-
	BN	-	-	-
	Conv2D	256x3x3	2x2	1x1
	BN	-	-	-

	Conv2D	1024x1x1	1x1	-
	BN	-	-	-
	ReLU	-	-	-
	Conv2D	1024x1x1	2x2	-
	BN	-	-	-
Bottleneck6	Conv2D	256x1x1	1x1	-
	BN	-	-	-
	Conv2D	256x3x3	1x1	1x1
	BN	-	-	-
	Conv2D	1024x1x1	1x1	-
	ReLU	-	-	-
Bottleneck7	Conv2D	512x1x1	1x1	-
	BN	-	-	-
	Conv2D	512x3x3	2x2	1x1
	BN	-	-	-
	Conv2D	2048x1x1	1x1	-
	BN	-	-	-
	ReLU	-	-	-
	Conv2D	2048x1x1	2x2	-
Bottleneck8	BN	-	-	-
	Conv2D	512x1x1	1x1	-
	BN	-	-	-
	Conv2D	512x3x3	1x1	1x1
	BN	-	-	-
	Conv2D	2048x1x1	1x1	-
BN	-	-	-	-
	ReLU	-	-	-

ii. *Generate Random Triplet*

Pada tabel 3.7 dan 3.8 telah dijelaskan arsitektur CNN pada ResNet-50 yang telah dimodifikasi, setelah model dimuat lalu proses selanjutnya adalah melakukan perulangan sejumlah epoch yang diinginkan sebagai contoh 200, lalu setelah itu buat data triplet secara acak seperti dijelaskan pada gambar 3.24 dibawah.

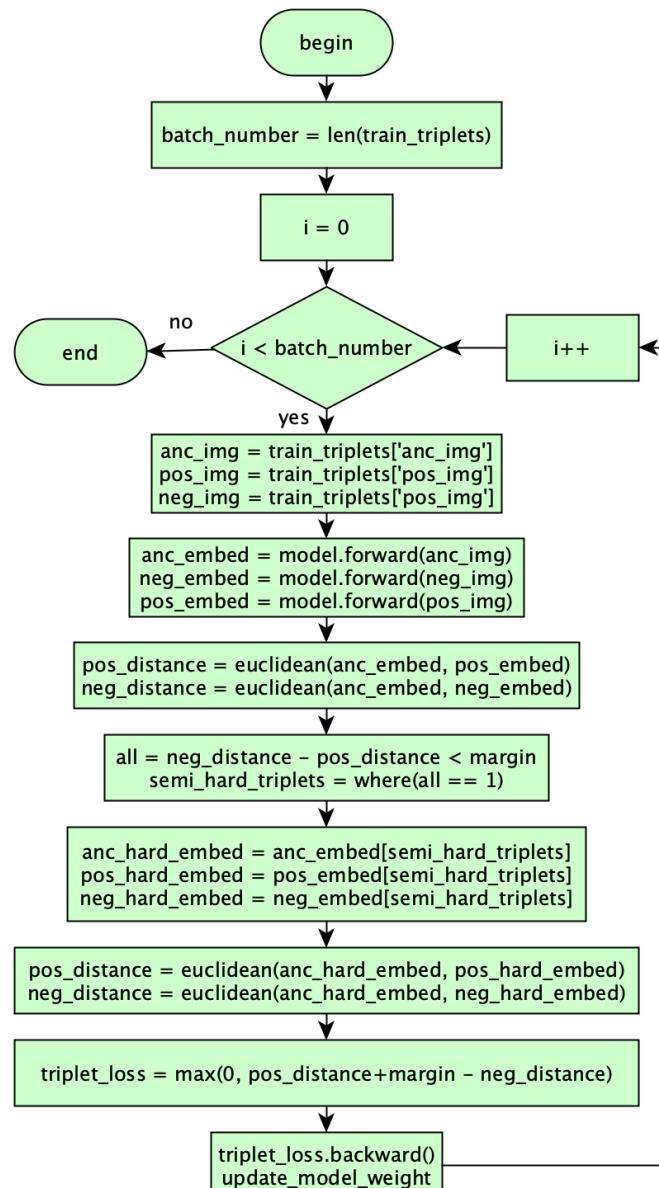


Gambar 3.24 Flowchart Pembuatan Triplet Acak

Pada tahap awal sesuaikan jumlah data *triplet* yang diinginkan dan sesuaikan lokasi folder dataset *training/testing*. Setelah itu buat sebuah *variabel array* dengan ukuran sejumlah dataset yang diinginkan. Setelah itu kelompokkan setiap gambar sesuai nama folder lalu lakukan perulangan sejumlah data *triplet*

yang diinginkan. Ambil gambar *anchor*, *positive* dan *negative* secara acak dan tampung pada variabel *triplets* begitu terus sejumlah triplet yang diinginkan. Setelah data triplet dibuat, data akan melalui tahap *preprocessing* seperti dijelaskan pada sub bab 3.1.4 lalu dibagi menjadi beberapa bagian (*batch size*). Pembuatan data triplet akan dilakukan pada setiap *epoch* untuk memastikan model belajar pada data yang berbeda.

iii. Proses Training

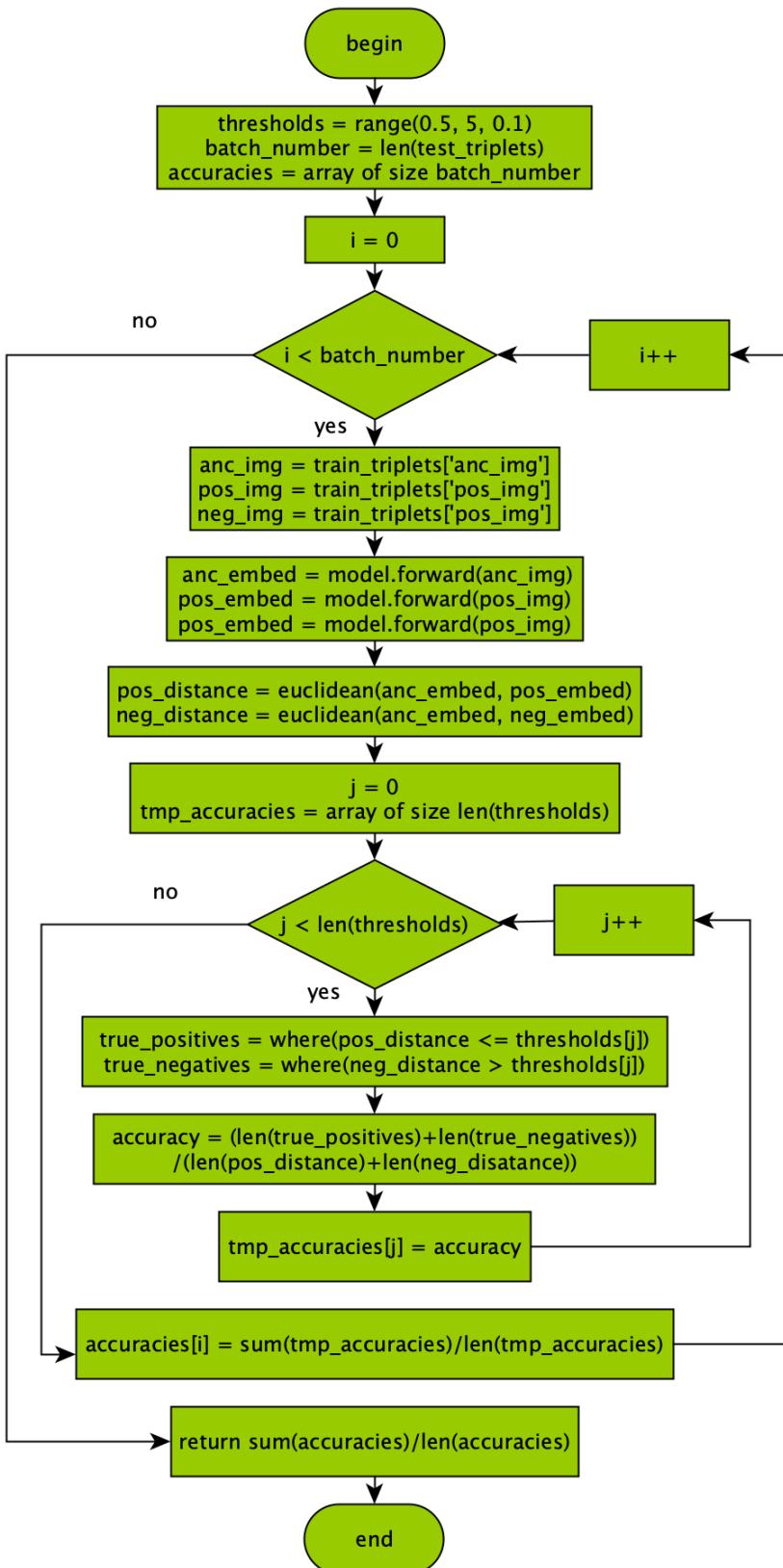


Gambar 3.25 Flowchart *Training*

Setelah proses pembuatan data *triplet* selesai proses selanjutnya adalah melakukan *training*. Alur proses training bisa dilihat pada flowchart gambar 3.25 diatas. Jumlah iterasi pada saat training adalah sesuai nilai *batch_number* yaitu banyaknya bagian data setelah dibagi sesuai *batch_size*. Proses pertama pada iterasi adalah pengambilan data *anchor*, *positive* dan *negative* dimana setiap data adalah tensor dengan dimensi *batch_size* x 3 x 224 x 224. Setelah itu data akan diteruskan ke model dengan operasi *forward* yaitu data akan melalui layer-layer yang telah dijelaskan pada tabel 3.7. Hasil dari operasi *forward* ini adalah *vector embedding* dari gambar *anchor*, *positive* dan *negative* lalu dilakukan penghitungan jarak antar gambar *anchor* dan *positive* dan jarak antara gambar *anchor* dan *negative*. Setelah itu akan dilakukan operasi logika untuk mencari jarak gambar *negative* yang lebih dekat ke *anchor* (*hard negative*), *index* dari gambar-gambar *hard negatives* akan disimpan di variabel *hard_triplets* yang akan digunakan untuk mengambil vektor embedding dari gambar *anchor*, *positive* dan *negative*. Setelah mendapatkan gambar *anchor*, *positive* dan *negative* sesuai index *hard_triplets* lalu dilakukan kembali penghitungan jarak dari gambar *hard_triplets*. Jarak yang didapat pada proses ini yang akan digunakan sebagai acuan untuk menghitung *triplet loss* sehingga jarak antar gambar *anchor* ke *positive* bisa diminimalkan. Setelah penghitungan loss dilakukan lalu model melakukan operasi *backward* (*backpropagation*) kemudian melakukan pembaruan parameter sesuai dengan nilai *gradient* yang didapat pada proses *backpropagation*.

iv. Proses Testing

Setelah proses *training* selesai proses berikutnya adalah *testing*. Adapun alur proses *testing* bisa dilihat pada gambar 3.26 *flowchart* dibawah.



Gambar 3.26 Flowchart Proses Testing

Proses *testing* dimulai dengan mendefinisikan daftar *threshold* dan variabel *accuracies* yang akan menampung nilai akurasi pada setiap iterasi, daftar nilai *threshold* yang digunakan dimulai dari 0,5 sampai 5,0 dengan perbedaan 0,1 setiap nilainya. Pada setiap iterasi data *triplets*, proses yang akan dilakukan adalah melakukan *forward* data gambar *anchor*, *positive* dan *negative* melalui layer-layer pada model untuk mendapatkan *vector embedding*. Adapun model yang digunakan pada proses forward ini adalah model dengan parameter yang telah diperbarui pada proses *training* sehingga bisa diketahui seberapa baik parameter yang telah dipelajari pada saat *training*. Setelah mendapatkan *vector embedding* dari gambar *anchor*, *positive* dan *negative* lalu proses berikutnya adalah menghitung jarak antar gambar *anchor* ke *positive* dan *negative*. Setelah mendapatkan jarak tersebut lalu dihitung berapa banyak embedding *true positive* dan *true negative* pada setiap nilai *threshold* yang telah didefinisikan. Nilai *true positive* bisa didapat dari jumlah jarak *anchor* dan *positive* yang lebih kecil atau sama dengan nilai *threshold*, dan nilai *true negative* bisa didapat dari jumlah jarak *anchor* dan *negative* yang lebih besar dari nilai *threshold*. Jumlah nilai *true positive* dan *true negative* jika dibagi dengan jumlah data pada *mini batch* akan menjadi nilai akurasi pada batch tersebut, lalu akurasi keseluruhan diambil dari rata-rata akurasi setiap *mini batch* yaitu jumlah semua nilai rata-rata akurasi pada setiap *mini batch* dibagi dengan jumlah batch (*batch number*). Proses *training* dan *testing* akan terus dilakukan sejumlah *epoch* yang diinginkan atau sampai model belajar dengan baik dengan diukur melalui nilai akurasi yang didapat pada proses *testing*.

BAB IV

HASIL DAN PEMBAHASAN

4.1. Konfigurasi Hardware dan Software

Proses *training* dan *testing* pada penelitian ini menggunakan layanan dari *Google Cloud Platform* (GCP) dengan konfigurasi *hardware* dan *software* sebagai berikut:

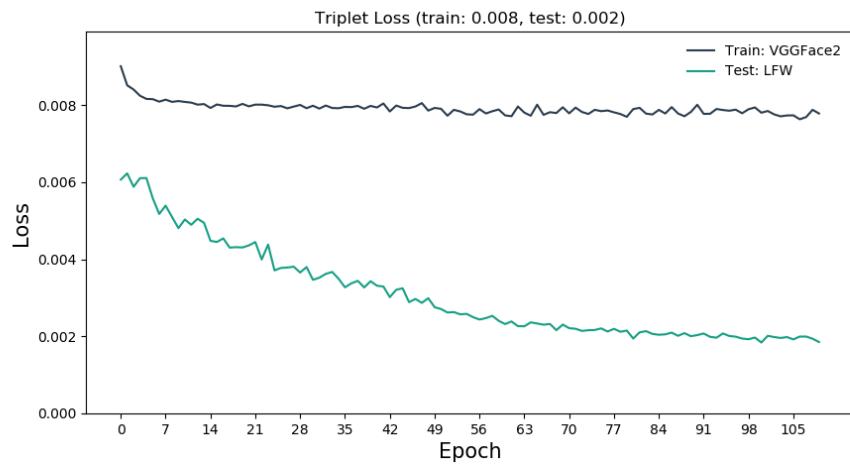
HARDWARE	4 x 12GB NVIDIA Tesla K80 GPU
	8 Core vCPU Intel Broadwell
	52GB RAM
	256GB Boot Disk
	200GB Dataset Disk
SOFTWARE	Debian 4.9.130-2 (2018-10-27) x86_64 GNU/Linux
	Python v3.7.1
	Pytorch v1.0
	CUDA v10.0

4.2. Proses Training dan Testing

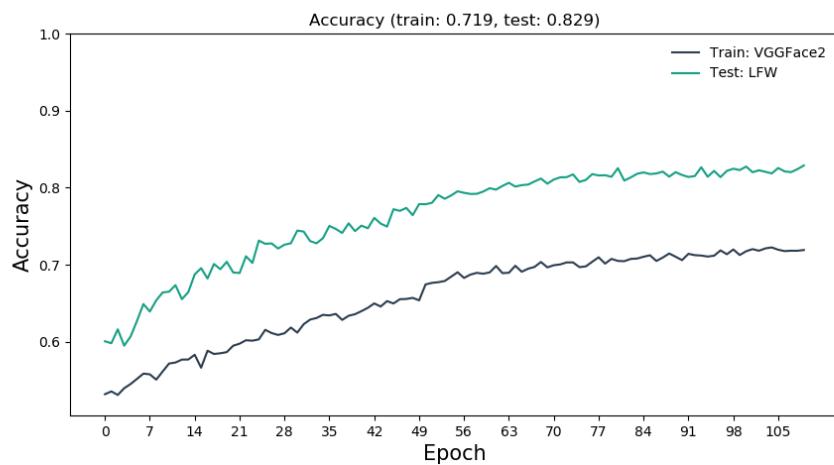
Proses *training* dan *testing* dalam penelitian ini dilakukan dalam beberapa tahap pada setiap layer dimulai dari layer terakhir dilanjutkan sampai layer pertama dengan *learning rate* yang berbeda seperti dijelaskan pada pembahasan berikut ini.

4.2.1. Layer FC

Layer FC (*Fully Connected Layer*) adalah layer yang akan membuat kesimpulan dari fitur yang dipelajari pada sebuah gambar. Dalam hal ini layer FC yang akan mempelajari *embedding* sebuah gambar. *Embedding* yang dipelajari pada tahap ini menggunakan fitur yang dipelajari dari *pretrained* ImageNet, model akan dilatih ulang untuk mempelajari fitur wajah pada dataset *training* melalui layer CNN yang akan dijelaskan pada sub bab 4.2.2. *Embedding* yang dihasilkan adalah vektor berdimensi 1x128. Grafik proses *training* dan *testing* layer ini bisa dilihat pada gambar 4.1 dan 4.2 dibawah.



Gambar 4.1 Grafik *Triplet Loss* Hasil *Training & Testing* Layer FC



Gambar 4.2 Grafik Akurasi Hasil *Training & Testing* Layer FC

Pada proses training Layer FC, model berhasil belajar hingga mendapatkan akurasi 82% pada data *testing* dan mendapatkan akurasi 71% pada data *training*, ini bisa diartikan bahwa model dapat belajar dengan baik hingga bisa mendapatkan akurasi yang lebih tinggi pada data *testing* yaitu data yang tidak pernah dilihat sebelumnya. Adapun sekilas proses *training* dan *testing* per *epoch* bisa dilihat pada tabel 4.1 dan tabel 4.2 dibawah.

Tabel 4.1 *Training Layer FC Learning Rate 0.001*

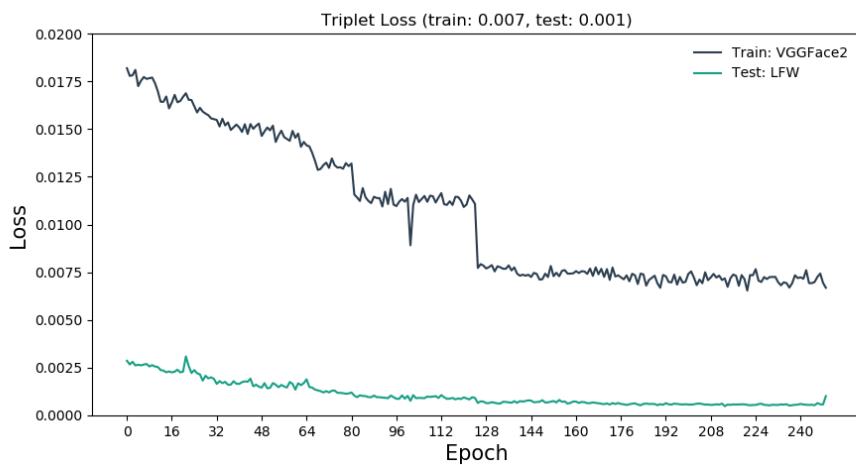
Epoch	Train Acc	Train Loss	Test Acc	Test Loss
0	0.5323	0.009	0.60095	0.00606
1	0.53595	0.00851	0.5984	0.00622
2	0.53135	0.0084	0.61645	0.00588
3	0.54005	0.00824	0.59525	0.0061
4	0.5455	0.00816	0.60715	0.00611
5	0.5522	0.00815	0.62735	0.00558
6	0.559	0.00809	0.64925	0.00517
7	0.5581	0.00814	0.6395	0.00539
8	0.5512	0.00808	0.654	0.0051
9	0.56175	0.0081	0.66425	0.00481
10	0.5721	0.00808	0.66525	0.00503
11	0.5734	0.00806	0.6737	0.00489
12	0.5772	0.00801	0.65555	0.00505
13	0.5773	0.00802	0.66465	0.00494
14	0.58345	0.00792	0.68775	0.00448
15	0.56665	0.00801	0.69575	0.00445
16	0.58875	0.00798	0.68235	0.00454
17	0.5845	0.00798	0.7012	0.0043
18	0.5854	0.00796	0.6944	0.00431
19	0.5869	0.00803	0.70405	0.0043
20	0.5952	0.00796	0.6902	0.00436
21	0.59785	0.00801	0.68945	0.00445
22	0.6023	0.00801	0.7112	0.00399
23	0.6017	0.00799	0.70255	0.00438
24	0.6034	0.00795	0.7315	0.00371
25	0.6158	0.00797	0.72735	0.00377
26	0.6116	0.00791	0.72785	0.00378
27	0.60915	0.00796	0.7212	0.00381
28	0.61135	0.008	0.72615	0.00365
29	0.61875	0.00792	0.72805	0.0038
30	0.6122	0.00798	0.7445	0.00347
31	0.6231	0.00791	0.7432	0.00352
32	0.62905	0.00799	0.73085	0.00362
33	0.63115	0.00792	0.728	0.00367
34	0.6352	0.00791	0.7346	0.00349
35	0.63445	0.00795	0.75055	0.00327
...
...
47	0.6558	0.00805	0.7739	0.00287
48	0.65725	0.00785	0.7645	0.00299
49	0.654	0.00792	0.77885	0.00275

Tabel 4.2 *Training Layer FC Learning Rate 0.0001*

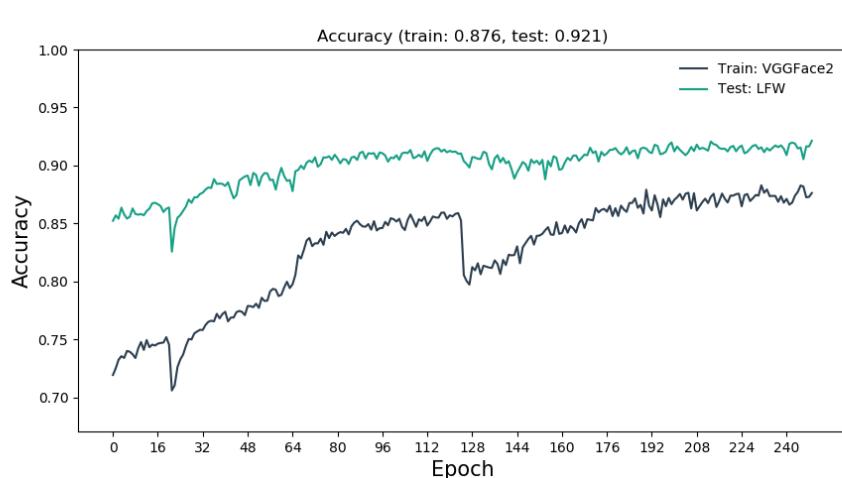
Epoch	Train Acc	Train Loss	Test Acc	Test Loss
50	0.67465	0.0079	0.7786	0.00271
51	0.6767	0.00772	0.7805	0.00262
52	0.67755	0.00787	0.79055	0.00263
53	0.67915	0.00783	0.78575	0.00257
54	0.68475	0.00776	0.79005	0.00258
55	0.69045	0.00775	0.79545	0.0025
56	0.6831	0.00789	0.79345	0.00244
57	0.6874	0.00778	0.792	0.00247
58	0.6897	0.00784	0.79215	0.00253
59	0.68855	0.00788	0.7951	0.0024
60	0.69025	0.00773	0.7994	0.00232
61	0.69855	0.00771	0.7976	0.00238
62	0.6893	0.00796	0.8024	0.00226
63	0.6898	0.0078	0.80645	0.00226
64	0.69885	0.00772	0.8015	0.00236
65	0.6911	0.00801	0.8034	0.00233
66	0.69505	0.00774	0.80415	0.0023
67	0.69735	0.00781	0.8082	0.00232
68	0.70375	0.00779	0.81195	0.00216
69	0.6968	0.00794	0.8053	0.0023
70	0.69955	0.00778	0.8107	0.00221
71	0.7006	0.00793	0.8135	0.0022
72	0.7032	0.00782	0.81375	0.00214
73	0.7032	0.00777	0.8173	0.00216
74	0.6971	0.00787	0.8076	0.00216
75	0.69805	0.00784	0.8102	0.00221
76	0.7041	0.00786	0.8177	0.00212
77	0.70985	0.00781	0.81595	0.00219
78	0.70165	0.00777	0.81625	0.00212
79	0.7078	0.00769	0.81425	0.00215
80	0.70515	0.00789	0.8254	0.00194
81	0.7048	0.00792	0.8093	0.0021
...
...
93	0.7121	0.0079	0.8266	0.00196
94	0.7108	0.00787	0.8143	0.00207
95	0.7119	0.00785	0.82185	0.00201
96	0.71875	0.00788	0.8138	0.00199
97	0.7138	0.00778	0.82195	0.00194
98	0.72	0.00789	0.8247	0.00192
99	0.7128	0.00794	0.82295	0.00197

4.2.2. Layer Ekstraksi Fitur

Layer ekstraksi fitur yang dimaksud adalah layer sebelum FC seperti yang terlampir pada gambar 3.13. Pada tahap ini model akan dilatih untuk mempelajari dataset baru yaitu dataset VGGv2 untuk mempelajari fitur wajah. Fitur *pretrained* yang dipelajari pada dataset ImageNet pada tahap ini akan disesuaikan ulang agar dapat mempelajari fitur wajah yang ada pada dataset VGGv2. Hasil belajar pada tahap ini bisa dilihat pada gambar 4.3 dan 4.4 dibawah.



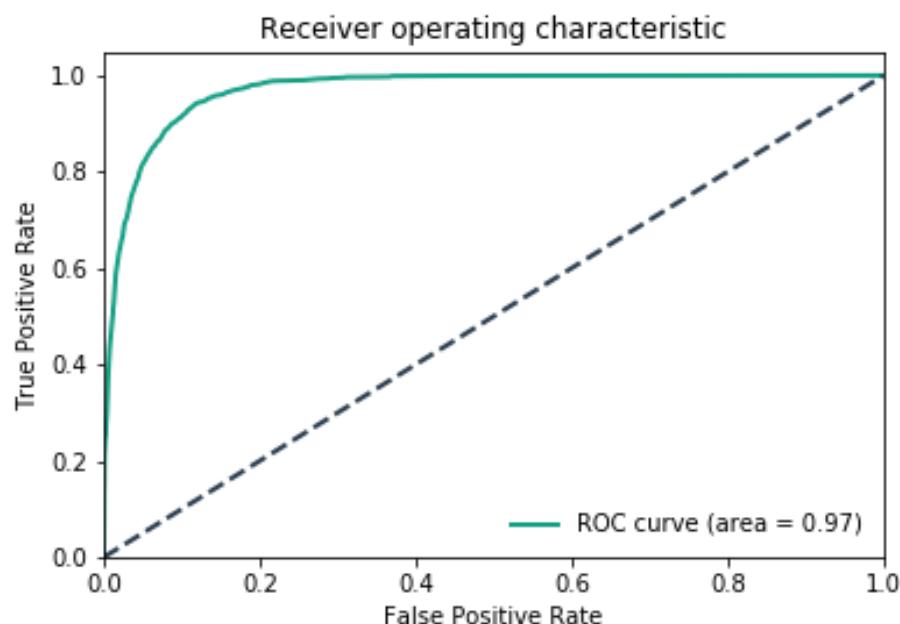
Gambar 4.3 Grafik *Triplet Loss* Hasil *Training & Testing* Layer
Ekstraksi Fitur



Gambar 4.4 Grafik Akurasi Hasil *Training & Testing* Layer Ekstraksi
Fitur

Terlihat pada gambar grafik 4.4 diatas, model dapat mencapai akurasi *testing* 90% dan akurasi *training* 83% pada *epoch* ke 70. Model kemudian dilatih sampai 100 epoch berikutnya model terlihat masih bertahan pada akurasi 90%. Pada *epoch* ke 250 model dapat mencapai akurasi 92% pada data *testing* dan mendapatkan akurasi 87% pada data *training*. Sampai pada *epoch* ke 250 proses *training* dihentikan dengan akurasi maksimal yang dicapai adalah 92%. Data sekilas *training* dan *testing* proses ini bisa dilihat pada tabel 4.3 dibawah.

Nilai akurasi *testing* yang lebih tinggi pada grafik diatas menjelaskan bahwa model mampu melakukan *generalisasi* data sehingga bisa memverifikasi data yang tidak pernah dilihat sebelumnya dengan akurasi yang tinggi. Model juga berhasil mendapat nilai AUC 97% seperti terlihat pada gambar 4.5 dibawah.



Gambar 4.5 Grafik ROC Proses Training Terakhir

Nilai AUC mencapai 97% bisa diinterpretasikan bahwa model dapat memverifikasi dengan benar gambar wajah orang yang sama dan orang yang berbeda dengan persentase 97% selama proses pengujian.

Tabel 4.3 *Training Layer Ekstraksi Fitur*

Epoch	Train Acc	Train Loss	Test Acc	Test Loss
100	0.7196	0.01819	0.85245	0.00286
101	0.7254	0.01778	0.8571	0.00267
102	0.7327	0.01782	0.85425	0.00281
103	0.7358	0.0181	0.86375	0.00262
104	0.73425	0.01725	0.8579	0.00266
105	0.7403	0.01753	0.85435	0.00262
106	0.7396	0.01773	0.85585	0.00267
107	0.73745	0.01763	0.86305	0.0027
108	0.7342	0.01766	0.85825	0.00257
109	0.74245	0.0177	0.8577	0.00263
110	0.748	0.0174	0.8582	0.00256
111	0.74115	0.01698	0.8572	0.00254
112	0.7497	0.01643	0.86075	0.00238
113	0.7435	0.01643	0.86295	0.00235
114	0.74575	0.01671	0.8675	0.00226
115	0.74495	0.01608	0.86795	0.0023
116	0.7466	0.01639	0.8667	0.00225
117	0.74715	0.01679	0.86505	0.00229
118	0.7475	0.0164	0.86	0.00239
119	0.75215	0.01647	0.86285	0.00226
120	0.74565	0.01669	0.86385	0.0023
121	0.70625	0.01688	0.8258	0.00309
122	0.7107	0.01653	0.84655	0.00259
123	0.72645	0.01652	0.8551	0.00222
124	0.73295	0.01619	0.8573	0.00237
125	0.7372	0.01587	0.861	0.00221
126	0.7446	0.01611	0.8647	0.00215
127	0.7506	0.01592	0.87075	0.00182
128	0.7502	0.01581	0.86795	0.00208
129	0.75545	0.01572	0.8727	0.00194
130	0.75705	0.01556	0.87265	0.002
...
340	0.8713	0.00724	0.909	0.0006
341	0.86631	0.00716	0.91875	0.00058
342	0.86812	0.00761	0.9198	0.00056
343	0.87413	0.00693	0.91895	0.00055
344	0.87732	0.00693	0.91475	0.00056
345	0.88285	0.00699	0.9156	0.00053
346	0.88187	0.00726	0.9055	0.00064
347	0.87266	0.00743	0.91655	0.00058
348	0.87302	0.00694	0.9165	0.00057
349	0.87636	0.00669	0.92135	0.00101

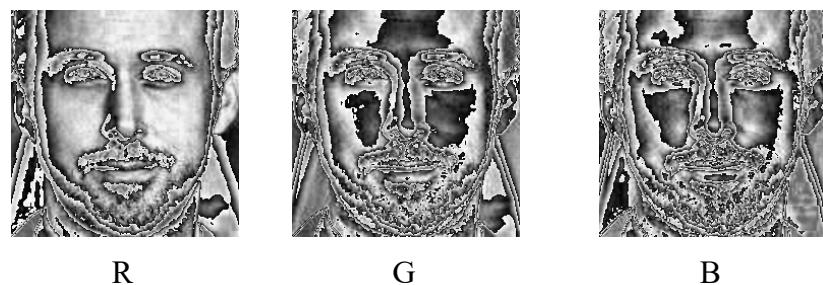
4.3. Hasil Belajar

Pada proses *training*, setiap layer ResNet (gambar 3.13) akan dilatih agar bisa mendapatkan representasi fitur sebuah gambar (dalam hal ini wajah). Setiap layer ResNet memiliki parameter (*weight* dan *bias*) yang akan dilatih agar memenuhi tujuan yang dijelaskan pada sub bab 3.3.1. Setiap masukan (*input*) gambar akan melalui setiap layer ResNet (gambar 3.13) sebagai bahan latih (*training*) layer ResNet agar berhasil mempelajari fitur yang ada pada wajah dengan akurasi maksimal.

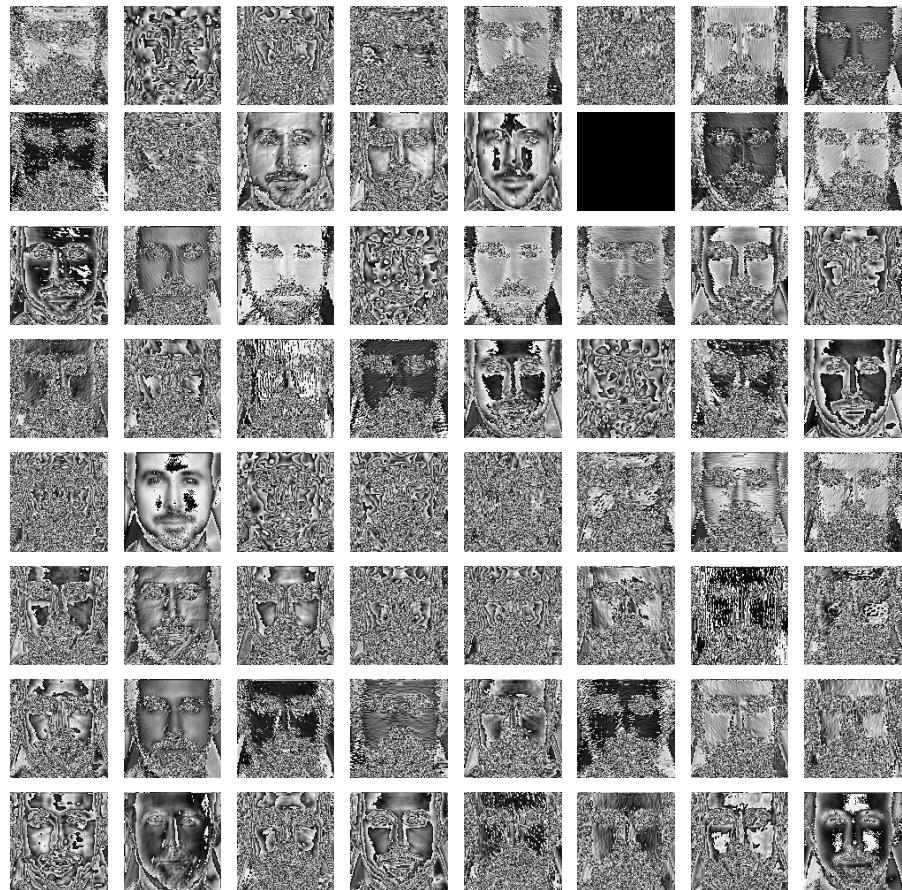
Pada sub bab 4.2 telah dijelaskan proses *training* yang telah dilalui layer ResNet dan mendapatkan akurasi 92% pada data *testing*. Pada sub bab ini akan ditampilkan fitur yang telah berhasil dipelajari layer ResNet sehingga mampu menghasilkan *embedding* yang baik dengan akurasi 92%. Hasil belajar setiap layer akan dijelaskan pada sub bab berikut.

4.3.1. Layer Conv1

Layer ini adalah layer konvolusi yang pertama kali dilalui oleh *input*. Layer ini membutuhkan gambar *input* dengan dimensi 224x224 dan memiliki 3 kanal warna, contoh gambar input yang digunakan bisa dilihat pada gambar 4.6. Layer ini memiliki 64 filter/kernel dengan dimensi 7x7 sehingga output dari layer ini adalah 64 gambar dengan dimensi 112x112. Hasil fitur yang dipelajari pada layer ini bisa dilihat pada gambar 4.7 dibawah.



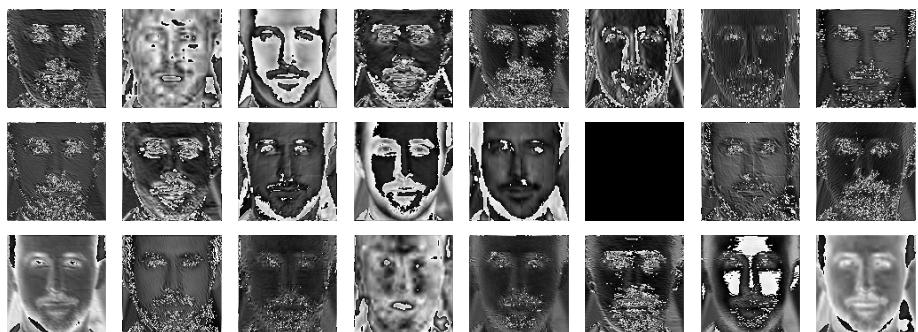
Gambar 4.6 Gambar *Input* Setelah Tahap *Preprocessing*

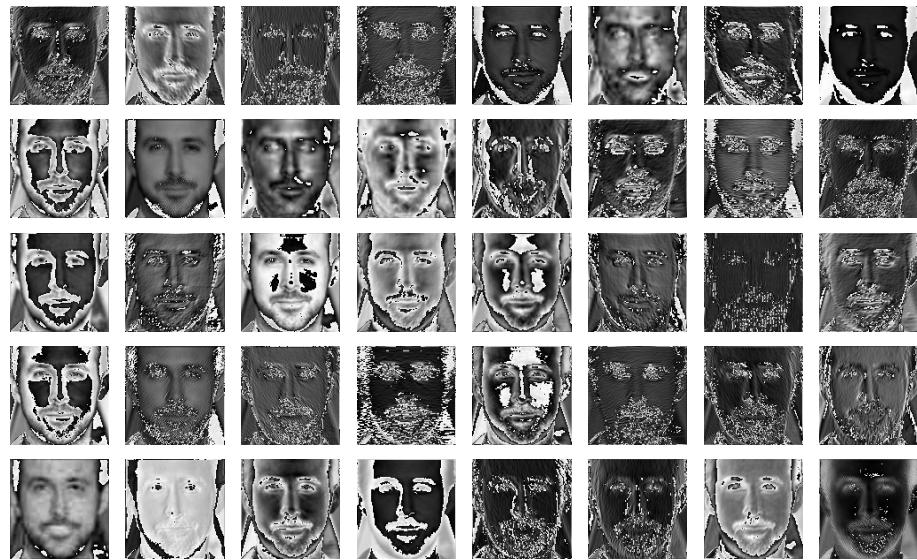


Gambar 4.7 *Output Layer Conv1*

4.3.2. Layer BN1

Layer BN1 adalah layer *Batch Normalization*, pada layer ini hasil conv1 pada layer sebelumnya akan dinormalisasi sebelum dilanjutkan ke layer aktivasi (layer ReLU). *Batch Normalization* berfungsi untuk mencegah *overfit* pada arsitektur jaringan. Hasil proses dari layer ini bisa dilihat pada gambar 4.8 dibawah.

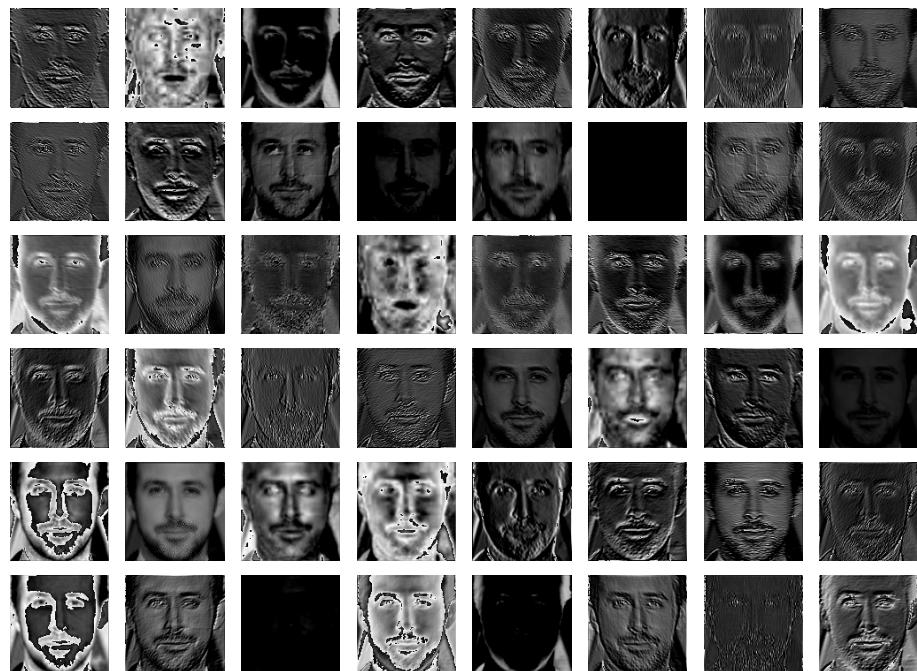


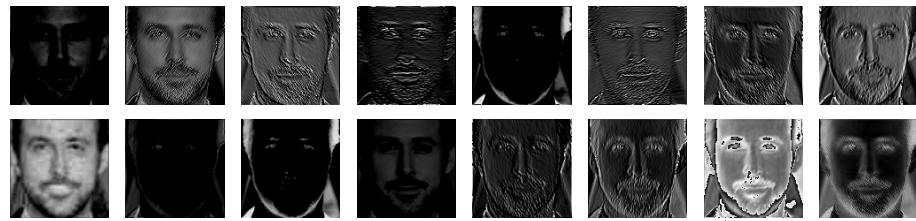


Gambar 4.8 *Output Layer BN1*

4.3.3. Layer ReLU

Layer ReLU adalah layer aktivasi yang akan dilalui oleh *input* setelah melalui layer BN1. Pada layer ini hasil dari BN1 akan diaktifasi dengan fungsi *Rectified Linear Unit* (ReLU). *Output* dari layer ini bisa dilihat pada gambar 4.9 dibawah.





Gambar 4.9 *Output Layer ReLU*

4.3.4. Layer Max Pool

Pada layer ini gambar yang telah diaktivasi pada layer sebelumnya akan melalui proses *Max Pool*. Setiap gambar yang melalui layer ini akan memiliki dimensi 56x56 karena proses *Max Pool*. Hasil dari proses *Max Pool* bisa dilihat pada gambar 4.10 dibawah.



Gambar 4.10 *Output Layer Max Pool*

4.3.5. Layer1

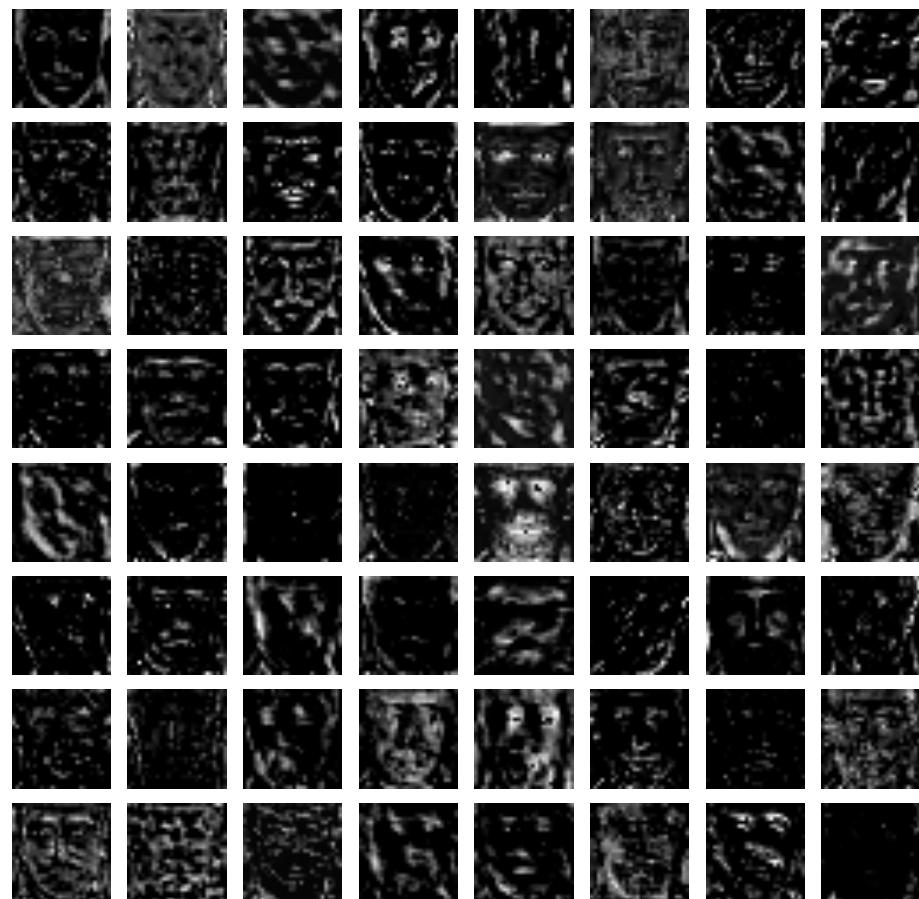
Layer1 adalah layer CNN pertama yang ada pada arsitektur ResNet. Pada layer ini hasil dari MaxPool akan diproses, *output* dari layer ini adalah fitur wajah yang berhasil dipelajari berjumlah 256 gambar dengan dimensi 56x56. Pada gambar 4.11 ditampilkan 64 gambar pertama hasil dari proses layer1.



Gambar 4.11 *Output Layer1*

4.3.6. Layer2

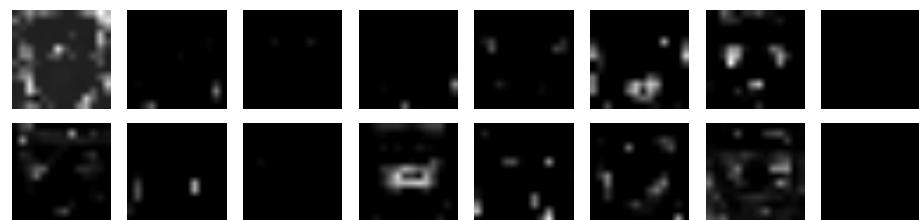
Layer2 adalah layer CNN kedua pada arsitektur ResNet. Hasil dari layer1 sebelumnya akan diproses pada layer ini. Layer2 akan menghasilkan fitur wajah berjumlah 512 gambar dengan dimensi 28x28. Pada gambar 4.12 ditampilkan 64 gambar pertama dari hasil belajar pada layer ini.

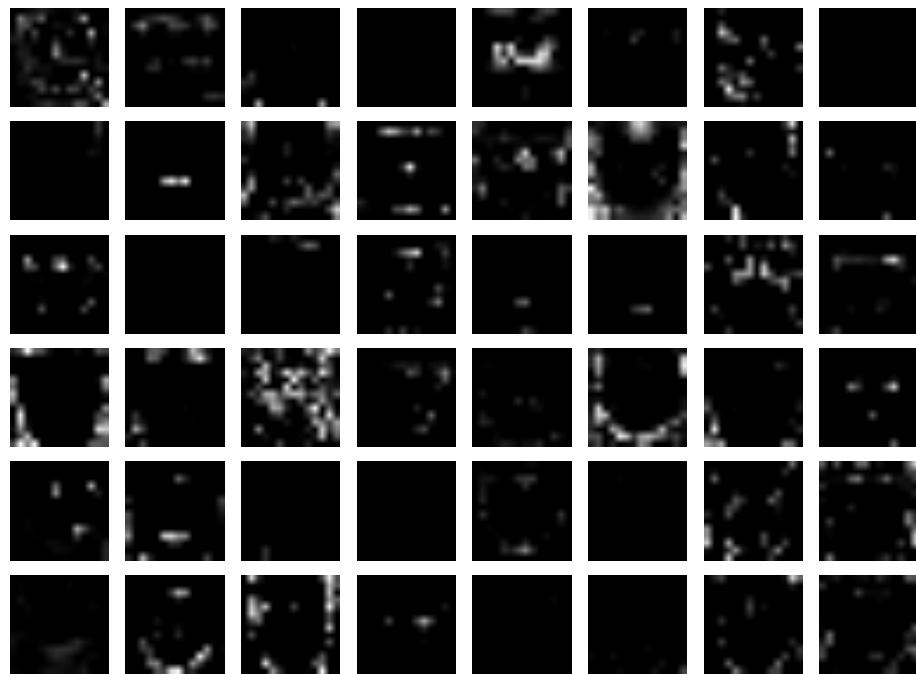


Gambar 4.12 *Output Layer2*

4.3.7. Layer3

Layer3 adalah layer CNN ketiga pada arsitektur ResNet. Layer ini akan mempelajari 512 gambar fitur wajah pada layer2 kemudian menghasilkan 1024 fitur wajah yang berhasil dipelajari dengan dimensi 14x14. Pada gambar 4.13 dibawah ditampilkan 64 dari 1024 gambar fitur wajah yang telah dipelajari.

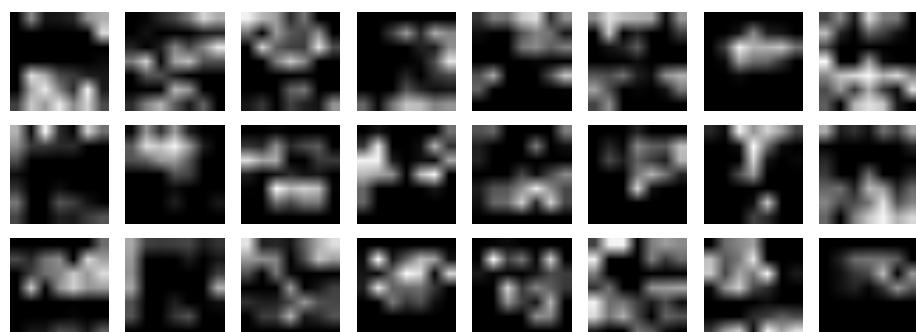


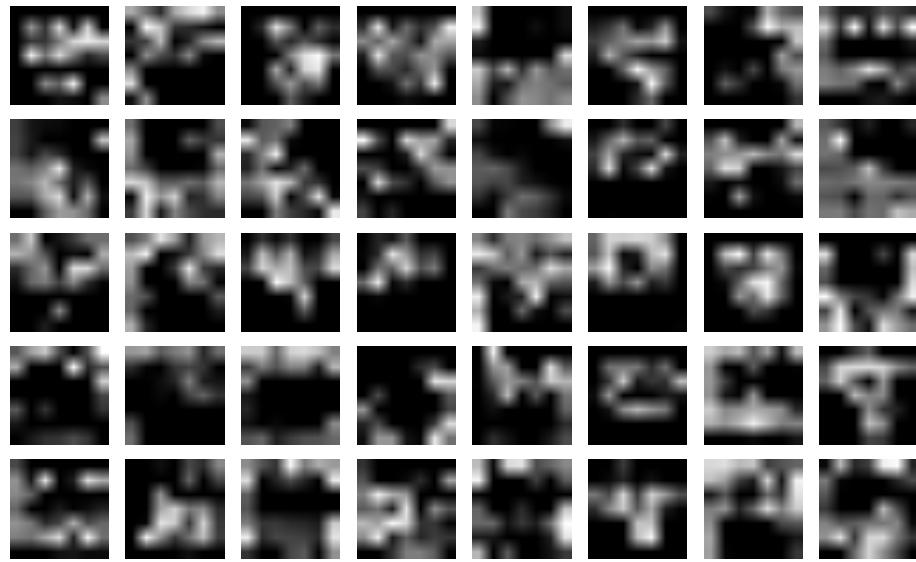


Gambar 4.13 *Output Layer3*

4.3.8. Layer4

Layer4 pada arsitektur ResNet adalah layer CNN terakhir, layer ini yang mempelajari paling banyak pola yang ada pada gambar wajah. Layer ini menghasilkan 2048 gambar fitur wajah yang telah berhasil dipelajari pada layer CNN sebelumnya (layer3). Gambar fitur wajah pada layer ini berdimensi 7x7. Fitur gambar wajah dengan dimensi 7x7 bisa mempelajari setiap detail yang ada pada wajah dengan sangat rinci. Setiap lekukan dan garis yang ada pada wajah akan dipelajari pada layer ini. Pada gambar 4.14 dibawah ditampilkan 64 gambar pertama hasil proses dari layer4.





Gambar 4.14 Output Layer4

4.3.9. Layer FC

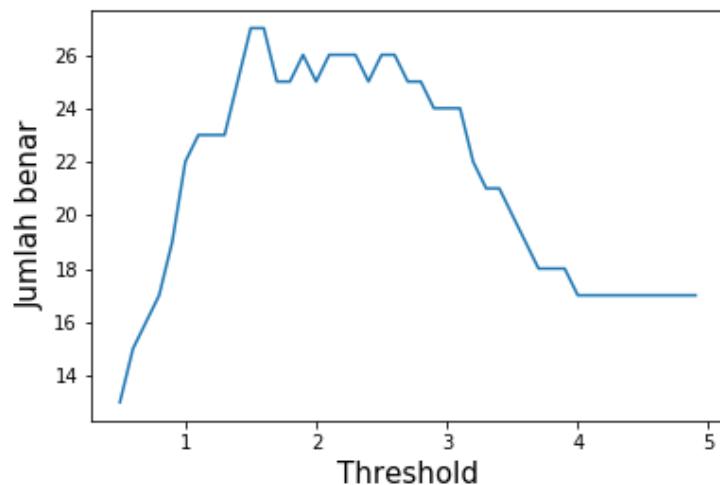
Fully Connected Layer (FC) adalah layer yang akan mempelajari dan mereduksi fitur yang dihasilkan pada layer3 hingga menghasilkan *embedding* dengan dimensi 1x128 yang memenuhi objektif yang dijelaskan pada sub bab 3.3.1. Fitur yang telah direduksi ini yang disebut dengan *feature vector* yang bisa dibandingkan dengan *feature vector* dari input gambar yang lain. Dengan fungsi jarak seperti *Euclidean distance* jarak antar dua *feature vector* bisa dibandingkan kedekatannya. Hasil *embedding* yang berhasil dipelajari dari proses layer3 bisa dilihat pada gambar 4.15 dibawah.

```
tensor([ 1.1557,  0.2307,  1.2943,  0.7842, -0.2532,  0.1531, -1.7464, -1.3914,
        -1.3545, -0.0110, -0.1946, -0.6645,  0.1273,  0.7399, -0.1545, -0.2009,
       -0.6261, -0.8211, -0.9204,  0.0701, -0.9609,  0.2529,  0.3518,  0.0715,
       -1.1649, -0.3410,  0.3977,  0.4475, -0.3396,  1.5479, -0.3602,  0.2861,
       -0.2775, -0.3556,  1.2695, -1.0079,  0.5255, -1.9043, -0.5698,  1.2976,
        0.8079,  1.2215,  0.9384,  1.3212,  0.2731, -1.1993, -1.8645, -1.6526,
       -0.4973,  0.1680,  0.7463, -0.8204,  0.7957, -0.1645, -0.5809,  0.8079,
        1.4552, -0.2576, -0.1386, -0.7120,  0.2104, -0.9524,  0.3615,  0.5889,
        0.5114,  0.3838, -0.7619,  0.1687, -0.0711, -0.0329, -0.6842, -1.0159,
        1.2719, -0.7590,  0.4144, -1.3286,  1.0694,  0.6197,  0.2119, -0.9670,
        1.0672, -0.0510,  0.0510,  1.1356,  0.1705, -0.5784, -0.2030, -0.7471,
        0.2597,  1.7872, -0.3302,  0.7973, -0.1497, -0.6227, -1.6351,  0.7571,
       -1.0362, -0.3064,  2.2854,  0.6904,  0.4053, -0.4887,  0.1502,  0.3052,
        0.6054, -0.3798,  0.0405, -2.0987, -2.0201, -1.0604,  1.1221, -0.0106,
       -0.1288,  0.2399, -0.1406,  0.2472,  0.3163,  1.0047, -1.2981, -1.5490,
       -0.7356, -0.6305,  0.4580,  1.6313,  1.8513, -0.6326, -0.4545,  0.0751])
```

Gambar 4.15 Hasil Embedding Layer FC

4.4. Percobaan Verifikasi Wajah

Pada sub bab ini akan ditampilkan hasil percobaan pembandingan 30 pasangan gambar. Untuk mendapatkan *embedding* dari setiap pasang gambar, akan dilakukan proses yang sama seperti yang dijelaskan pada poin-poin pada sub bab 4.3. Jarak antar pasangan *embedding* akan dihitung menggunakan *Euclidean distance*. Jarak yang didapatkan kemudian akan dibandingkan dengan ambang toleransi kedekatan tertentu (*threshold*), jika jarak *embedding* pasangan gambar lebih besar dari nilai *threshold* maka gambar akan dianggap orang yang berbeda dan jika dibawah nilai *threshold* akan dianggap orang yang sama. Pada percobaan ini akan digunakan beberapa nilai *threshold* yaitu 0.5, 0.6, 0.7, 0.8, 0.9 dan seterusnya sampai 5.0. Hasil dari percobaan ini bisa dilihat pada gambar grafik 4.16 dibawah.



Gambar 4.16 Jumlah Verifikasi Benar Setiap Nilai *Threshold*

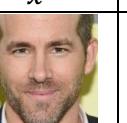
Pada gambar grafik diatas jumlah verifikasi benar setiap nilai *threshold* berbeda-beda. Terlihat nilai *threshold* 1,5 dan 1,6 mendapat jumlah verifikasi benar terbanyak yaitu 27 dari 30 percobaan. Terlihat nilai *threshold* pada rentang 1,5 sampai 2,8 mendapat nilai jumlah benar lebih banyak dari nilai *threshold* yang lain, dan nilai *threshold* 4 sampai 5 mendapat jumlah benar yang sama yaitu 17. Tabel lengkap jumlah benar setiap nilai *threshold* dapat dilihat pada tabel 4.4 dibawah.

Tabel 4.4 Hasil Jumlah Benar Setiap Nilai *Threshold*

Nilai Threshold	Jumlah Benar
0,5	13
0,6	15
0,7	16
0,8	17
0,9	19
1,0	22
1,1 – 1,3	23
1,4	25
1,5 – 1,6	27
1,7 – 1,8	25
1,9	26
2,0	25
2,1 – 2,3	26
2,4	25
2,5 – 2,6	26
2,7 – 2,8	25
2,9 – 3,1	24
3,2	22
3,3 – 3,4	21
3,5	20
3,6	19
3,7 – 3,9	18
4,0 – 4,9	17

Gambar hard negative pada tabel 3.6 pada sub bab 3.3.2 jika dihitung kembali jarak antar gambar *anchor* ke *positive* dan *negative* dengan model yang sudah dilatih menghasilkan jarak seperti tabel 4.5 dibawah.

Tabel 4.5 Jarak Gambar Hard Negative Setelah *Training*

x^a	x^p	x^n	$d(a, p)$	$d(a, n)$
			0,6	0,7

Pada gambar yang sama jarak *anchor* lebih dekat ke gambar *negative* (lihat tabel 3.6), lalu setelah model dilatih jarak gambar *positive* berhasil diminimalkan sehingga jarak gambar *anchor* lebih dekat ke gambar *positive*.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan pemaparan hasil penelitian pada BAB IV bisa disimpulkan beberapa hal yaitu:

1. Arsitektur jaringan Siamese berhasil mempelajari fitur wajah pada dataset VGGv2 menggunakan ResNet-50 sehingga mampu menghasilkan *embedding* yang baik.
2. Arsitektur jaringan Siamese berhasil dalam mempelajari *embedding* gambar menggunakan ResNet-50 yang telah dimodifikasi dengan akurasi 92% pada data *testing*.
3. Hasil pembelajaran pada data *training* cukup baik hingga mendapatkan nilai AUC 97% pada data *testing*.

5.2. Saran

Dalam penelitian ini ada beberapa eksperimen yang masih mungkin dilakukan untuk meningkatkan akurasi pembelajaran atau penerapan selain verifikasi wajah:

1. Melatih jaringan dengan menggunakan *softmax* sebelum dilatih dengan *triplet loss*, melatih jaringan dengan *softmax* bertujuan untuk meningkatkan kemampuan jaringan dalam mengenali fitur-fitur wajah pada dataset *training*.
2. Menggunakan jenis *triplet* semi-hard untuk melatih fungsi *triplet loss* sesuai dengan yang digunakan pada paper *FaceNet*.
3. Membandingkan hasil pembelajaran *embedding* dari *triplet loss* dan *constrastive loss*.
4. Mengganti arsitektur jaringan ResNet-50 dengan Inception Resnet v2 untuk mendapatkan fitur yang lebih kompleks pada gambar.

5. Menerapkan hasil *embedding* yang telah dipelajari pada kasus seperti pencarian gambar (*image query*), klasifikasi wajah dan pengenalan wajah.
6. Menerapkan model yang telah dilatih menjadi aplikasi nyata seperti keamanan gedung, presensi kehadiran dan lain-lain.

DAFTAR REFERENSI

- Andrej Karpathy. (2017). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved July 12, 2019, from <https://cs231n.github.io/neural-networks-2/#datapre>
- Expert System. (2017). What is Machine Learning? A definition - Expert System. Retrieved July 25, 2019, from <https://www.expertsystem.com/machine-learning-definition/>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition
- Hermans, A., Beyer, L., & Leibe, B. (2017). In Defense of the Triplet Loss for Person Re-Identification. Retrieved from <http://arxiv.org/abs/1703.07737>
- Huang, G. B., Ramesh, M., Berg, T., & Learned-Miller, E. (2008). Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.pdf. *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 1–11.
- Hermawati, F. A. (2013). Pengolahan Citra Digital, (January 2013), 198. Retrieved from <http://andipublisher.com/produk-0618006697-pengolahan-citra-digital.html>
- Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Retrieved from <http://arxiv.org/abs/1502.03167>
- Jake Frankenfield. (2018). Artificial Neural Networks (ANN) Defined. Retrieved July 26, 2019, from <https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp>
- Kusuma, D. A. A., Ardilla, F., & Dewantara, B. S. B. (2011). Verifikasi Citra Wajah Menggunakan Metode Discrete Cosine Transform Untuk Aplikasi Login. *Industrial Electronics Seminar*, 8(5), 55.
- Narkhede, S. (2018a). Understanding AUC - ROC Curve – Towards Data Science. Retrieved June 28, 2019, from <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>
- Narkhede, S. (2018b). Understanding Confusion Matrix – Towards Data Science. Retrieved June 28, 2019, from <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>

- Pytorch. (n.d.). *torchvision.models — PyTorch master documentation*. Retrieved July 19, 2019, from <https://pytorch.org/docs/stable/torchvision/models.html?highlight=resnet>
- Qiong Cao, Shen, L., Weidi Xie, Parkhi, O. M., & Zisserman, A. (2018). VGGFace2: A dataset for recognising faces across pose and age, 826.
- Ranjan, R., Castillo, C. D., & Chellappa, R. (2017). L 2 -constrained Softmax Loss for Discriminative Face Verification.
- Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 07-12-June*, 815–823. <https://doi.org/10.1109/CVPR.2015.7298682>
- Schroff, F., & Philbin, J. (n.d.). FaceNet : A Unified Embedding for Face Recognition and Clustering.
- Sebastian Raschka. (2015). Single-Layer Neural Networks and Gradient Descent. Retrieved July 27, 2019, from https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
- Skansi, S. (2018). *Introduction to deep learning: Part 1. Chemical Engineering Progress* (Vol. 114). <https://doi.org/10.1007/978-3-319-73004-2>
- Soares, F., & Souza, A. M. F. (2016). *Neural network programming with Java : unleash the power of neural networks by implementing professional Java code.*
- Udacity Course. (2018). Secure and Private AI Scholarship Challenge - Udacity. Retrieved July 27, 2019, from <https://classroom.udacity.com/nanodegrees/nd185/part/3fe1bb10-68d7-4d84-9c99-9539dedffad5/module/28d685f0-0cb1-4f94-a8ea-2e16614ab421/lesson/d9869c40-de54-4395-9d5f-fa13c8254277/concept/70526adf-40d3-4446-ac32-d3f798739745>
- Wani, M. A., Bhat, F. A., Afzal, S., & Khan, A. I. (2018). *Advances in Deep Learning on Graphs*.
- Zhang, K., Zhang, Z., Li, Z., Member, S., Qiao, Y., & Member, S. (2016). Joint Face Detection and Alignment using Multi-ta, (1), 1–5. <https://doi.org/10.1109/LSP.2016.2603342>

```

class Flatten(nn.Module):
    def forward(self, x):
        return x.view(x.size(0), -1)

class FaceNetModel(nn.Module):
    def __init__(self, embedding_size, num_classes, pretrained=False):
        super(FaceNetModel, self).__init__()

        self.model = resnet50(pretrained)
        self.embedding_size = embedding_size
        self.cnn = nn.Sequential(
            self.model.conv1,
            self.model.bn1,
            self.model.relu,
            self.model.maxpool,
            self.model.layer1,
            self.model.layer2,
            self.model.layer3,
            self.model.layer4)

        self.model.fc = nn.Sequential(
            Flatten(),
            nn.Linear(100352, self.embedding_size))

        self.model.classifier = nn.Linear(self.embedding_size, num_classes)

    def l2_norm(self, input):
        input_size = input.size()
        buffer = torch.pow(input, 2)
        normp = torch.sum(buffer, 1).add_(1e-10)
        norm = torch.sqrt(normp)
        _output = torch.div(input, norm.view(-1, 1).expand_as(input))
        output = _output.view(input_size)

        return output

    def freeze_all(self):
        for param in self.model.parameters():
            param.requires_grad = False

    def unfreeze_all(self):
        for param in self.model.parameters():

```

param.requires_grad = True

```

            def freeze_fc(self):
                for param in self.model.fc.parameters():
                    param.requires_grad = False

            def unfreeze_fc(self):
                for param in self.model.fc.parameters():
                    param.requires_grad = True

            def freeze_classifier(self):
                for param in self.model.classifier.parameters():
                    param.requires_grad = False

            def unfreeze_classifier(self):
                for param in self.model.classifier.parameters():
                    param.requires_grad = True

            def freeze_only(self, freeze):
                for name, child in self.model.named_children():
                    if name in freeze:
                        for param in child.parameters():
                            param.requires_grad = False
                    else:
                        for param in child.parameters():
                            param.requires_grad = True

            def unfreeze_only(self, unfreeze):
                for name, child in self.model.named_children():
                    if name in unfreeze:
                        for param in child.parameters():
                            param.requires_grad = True
                    else:
                        for param in child.parameters():
                            param.requires_grad = False

            # returns face
            embedding(embedding_size)
            def forward(self, x):

```

```

x = self.cnn(x)
x = self.model.fc(x)

features = self.l2_norm(x)
# Multiply by alpha = 10
as suggested in
https://arxiv.org/pdf/1703.09507.pdf
alpha = 10
features = features *
alpha
return features

def forward_classifier(self,
x):
    features = self.forward(x)
    res =
self.model.classifier(features)
    return res

class
TripletLoss(torch.nn.Module):

    def __init__(self, margin):
        super(TripletLoss,
self).__init__()
        self.margin = margin
        self.pdist =
PairwiseDistance(2)

        def forward(self, anchor,
positive, negative):
            pos_dist =
self.pdist.forward(anchor,
positive)
            neg_dist =
self.pdist.forward(anchor,
negative)

            hinge_dist =
torch.clamp(self.margin + pos_dist
- neg_dist, min=0.0)
            loss =
torch.mean(hinge_dist)
            return loss

import os

import numpy as np
import pandas as pd
import torch
from skimage import io
from torch.utils.data import
Dataset
from torchvision import transforms

class TripletFaceDataset(Dataset):

```

```

    def __init__(self, root_dir,
csv_name, num_triplets,
transform=None):

        self.root_dir = root_dir
        self.df =
pd.read_csv(csv_name)
        self.num_triplets =
num_triplets
        self.transform = transform
        self.training_triplets =
self.generate_triplets(self.df,
self.num_triplets)

    @staticmethod
    def generate_triplets(df,
num_triplets):

        def
make_dictionary_for_face_class(df):
            :

            """
            - face_classes =
{'class0': [class0_id0, ...],
'class1': [class1_id0, ...], ...}
            """
            face_classes = dict()
            for idx, label in
enumerate(df['class']):
                if label not in
face_classes:
                    face_classes[label] = []
            face_classes[label].append((df.iloc
[idx]['id'],
df.iloc[idx]['ext']))
            return face_classes

            triplets = []
            classes =
df['class'].unique()
            face_classes =
make_dictionary_for_face_class(df)

            for _ in
range(num_triplets):
                """
                - randomly choose
anchor, positive and negative
images for triplet loss
                - anchor and
positive images in pos_class
                - negative image in
neg_class
                - at least, two
images needed for anchor and
positive images in pos_class

```

```

        - negative image
should have different class as
anchor and positive images by
definition
    '''

        pos_class =
np.random.choice(classes)
        neg_class =
np.random.choice(classes)
            while
len(face_classes[pos_class]) < 2:
                pos_class =
np.random.choice(classes)
                    while pos_class ==
neg_class:
                        neg_class =
np.random.choice(classes)

                pos_name =
df.loc[df['class'] == pos_class,
'name'].values[0]
                neg_name =
df.loc[df['class'] == neg_class,
'name'].values[0]

            if
len(face_classes[pos_class]) == 2:
                ianc, ipos =
np.random.choice(2, size=2,
replace=False)
            else:
                ianc =
np.random.randint(0,
len(face_classes[pos_class]))
                ipos =
np.random.randint(0,
len(face_classes[pos_class]))
                    while ianc ==
ipos:
                        ipos =
np.random.randint(0,
len(face_classes[pos_class]))
                            ineg =
np.random.randint(0,
len(face_classes[neg_class]))

                anc_id =
face_classes[pos_class][ianc][0]
                anc_ext =
face_classes[pos_class][ianc][1]
                pos_id =
face_classes[pos_class][ipos][0]
                pos_ext =
face_classes[pos_class][ipos][1]
                neg_id =
face_classes[neg_class][ineg][0]
                neg_ext =
face_classes[neg_class][ineg][1]

            triplets.append(
[anc_id, pos_id,
neg_id, pos_class, neg_class,
pos_name, neg_name, anc_ext,
pos_ext, neg_ext])

        return triplets

    def __getitem__(self, idx):

        anc_id, pos_id, neg_id,
pos_class, neg_class, pos_name,
neg_name, anc_ext, pos_ext,
neg_ext = \
self.training_triplets[idx]

        anc_img =
os.path.join(self.root_dir,
str(pos_name), str(anc_id) +
f'.{anc_ext}')
        pos_img =
os.path.join(self.root_dir,
str(pos_name), str(pos_id) +
f'.{pos_ext}')
        neg_img =
os.path.join(self.root_dir,
str(neg_name), str(neg_id) +
f'.{neg_ext}')

        anc_img =
io.imread(anc_img)
        pos_img =
io.imread(pos_img)
        neg_img =
io.imread(neg_img)

        pos_class =
torch.from_numpy(np.array([pos_class]).astype('long'))
        neg_class =
torch.from_numpy(np.array([neg_class]).astype('long'))

        sample = {'anc_img':
anc_img, 'pos_img': pos_img,
'neg_img': neg_img, 'pos_class':
pos_class,
'neg_class': neg_class}

        if self.transform:
            sample['anc_img'] =
self.transform(sample['anc_img'])
            sample['pos_img'] =
self.transform(sample['pos_img'])
            sample['neg_img'] =
self.transform(sample['neg_img'])

        return sample

    def __len__(self):

```

```

        return
len(self.training_triplets)

def get_dataloader(train_root_dir,
valid_root_dir,
train_csv_name,
valid_csv_name,
num_train_triplets,
num_valid_triplets,
batch_size,
num_workers):
    data_transforms = {
        'train':
transforms.Compose([
transforms.ToPILImage(),
transforms.RandomRotation(15),
transforms.RandomResizedCrop(224),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])),
        'valid':
transforms.Compose([
transforms.ToPILImage(),
transforms.Resize(224),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])])
    }

    face_dataset = {
        'train':
TripletFaceDataset(root_dir=train_root_dir,
csv_name=train_csv_name,
num_triplets=num_train_triplets,
transform=data_transforms['train']),
        'valid':
TripletFaceDataset(root_dir=valid_root_dir,
csv_name=valid_csv_name,
num_triplets=num_valid_triplets,
transform=data_transforms['valid'])}

    dataloaders = {
        x:
torch.utils.data.DataLoader(face_dataset[x], batch_size=batch_size,
shuffle=False,
num_workers=num_workers)
            for x in ['train', 'valid']}
    data_size = {x:
len(face_dataset[x]) for x in ['train', 'valid']}

    return dataloaders, data_size

import argparse
import time

import numpy as np
import torch
import torch.optim as optim
from torch.nn.modules.distance import PairwiseDistance
from torch.optim import lr_scheduler

from data_loader import get_dataloader
from datasets.write_csv_for_making_data_set import write_csv
from eval_metrics import evaluate, plot_roc
from loss import TripletLoss
from models import FaceNetModel
from utils import ModelSaver, init_log JustCreated

# from utils import VisdomLinePlotter

parser = argparse.ArgumentParser(description='Face Recognition using Triplet Loss')

parser.add_argument('--num-epochs', default=200, type=int,
metavar='NE',
help='number of epochs to train (default: 200)')
parser.add_argument('--num-classes', default=10000, type=int,
metavar='NC',

```

```

                help='number
of classes (default: 10000)')
parser.add_argument('--num-train-
triplets', default=10000,
type=int, metavar='NTT',
                     help='number
of triplets for training (default:
10000)')
parser.add_argument('--num-valid-
triplets', default=10000,
type=int, metavar='NVT',
                     help='number
of triplets for validation
(default: 10000)')
parser.add_argument('--embedding-
size', default=128, type=int,
metavar='ES',
                     help='embedding size (default:
128)')
parser.add_argument('--batch-
size', default=64, type=int,
metavar='BS',
                     help='batch
size (default: 128)')
parser.add_argument('--num-
workers', default=8, type=int,
metavar='NW',
                     help='number
of workers (default: 8)')
parser.add_argument('--learning-
rate', default=0.001, type=float,
metavar='LR',
                     help='learning
rate (default: 0.001)')
parser.add_argument('--margin',
default=0.5, type=float,
metavar='MG',
                     help='margin
(default: 0.5)')
parser.add_argument('--train-root-
dir',
default='/run/media/hoosiki/WareHo
use2/home/mtb/datasets/vggface2/te
st_mtcnnpy_182', type=str,
                     help='path to
train root dir')
parser.add_argument('--valid-root-
dir',
default='/run/media/hoosiki/WareHo
use2/home/mtb/datasets/lfw/lfw_mtc
nnpy_182',
                     type=str,
                     help='path to
valid root dir')
parser.add_argument('--train-csv-
name',
default='./datasets/test_vggface2.
csv', type=str,
                     help='list of
training images')
parser.add_argument('--valid-csv-
name',
default='./datasets/lfw.csv',
type=str,
                     help='list of
validation images')
parser.add_argument('--step-size',
default=50, type=int,
metavar='SZ',
                     help='Decay
learning rate schedules every --
step-size (default: 50)')
parser.add_argument('--unfreeze',
type=str, metavar='UF',
default='',
                     help='Provide
an option for unfreezing given
layers')
parser.add_argument('--freeze',
type=str, metavar='F', default='',
                     help='Provide
an option for freezing given
layers')
parser.add_argument('--pretrain',
action='store_true')
parser.add_argument('--fc-only',
action='store_true')
parser.add_argument('--except-fc',
action='store_true')
parser.add_argument('--load-best',
action='store_true')
parser.add_argument('--load-last',
action='store_true')
parser.add_argument('--continue-
step', action='store_true')
parser.add_argument('--train-all',
action='store_true', help='Train
all layers')

args = parser.parse_args()
device = torch.device('cuda:0' if
torch.cuda.is_available() else
'cpu')
l2_dist = PairwiseDistance(2)
modelsaver = ModelSaver()

# plotter =
VisdomLinePlotter('Siamese
Triplet')

def save_if_best(state, acc):
    modelsaver.save_if_best(acc,
state)

def main():

```

```

init_log_just_created("log/valid.csv")
init_log_just_created("log/train.csv")

pretrain = args.pretrain
fc_only = args.fc_only
except_fc = args.except_fc
train_all = args.train_all
unfreeze =
args.unfreeze.split(',')
freeze =
args.freeze.split(',')
start_epoch = 0
print(f"Transfer learning:
{pretrain}"))
print("Train fc only:", fc_only)
print("Train except fc:", except_fc)
print("Train all layers:", train_all)
print("Unfreeze only:", ', '.join(unfreeze))
print("Freeze only:", ', '.join(freeze))
print(f"Learning rate will decayed every {args.step_size}th epoch")
model =
FaceNetModel(embedding_size=args.embedding_size,
num_classes=args.num_classes,
pretrained=pretrain).to(
    device)
triplet_loss =
TripletLoss(args.margin).to(device)

if fc_only:
    model.freeze_all()
    model.unfreeze_fc()

model.unfreeze_classifier()
if except_fc:
    model.unfreeze_all()
    model.freeze_fc()
    model.freeze_classifier()
if train_all:
    model.unfreeze_all()
if len(unfreeze) > 0:
    model.unfreeze_only(unfreeze)
    if len(freeze) > 0:
        model.freeze_only(freeze)

optimizer =
optim.Adam(filter(lambda p:
p.requires_grad,
model.parameters()),
lr=args.learning_rate)
scheduler =
lr_scheduler.StepLR(optimizer,
step_size=args.step_size,
gamma=0.1)

if args.load_best or
args.load_last:
    checkpoint =
'./log/best_state.pth' if
args.load_best else
'./log/last_checkpoint.pth'
    print('loading',
checkpoint)
    checkpoint =
torch.load(checkpoint)
modelsaver.current_acc =
checkpoint['accuracy']
start_epoch =
checkpoint['epoch'] + 1

model.load_state_dict(checkpoint['state_dict'])
print("Stepping
scheduler")
try:

optimizer.load_state_dict(checkpoint['optimizer_state'])
except ValueError as e:
    print("Can't load last
optimizer")
    print(e)
if args.continue_step:
    scheduler.step(checkpoint['epoch'])
    print(f"Loaded checkpoint
epoch: {checkpoint['epoch']} \n"
          f"Loaded checkpoint
accuracy:
{checkpoint['accuracy']} \n"
          f"Loaded checkpoint
loss: {checkpoint['loss']}")

    model =
torch.nn.DataParallel(model)

    for epoch in
range(start_epoch, args.num_epochs +
start_epoch):
        print(80 * '=')
        print('Epoch
[{} / {}]'.format(epoch,
args.num_epochs + start_epoch -
1))

time0 = time.time()

```

```

        data_loaders, data_size =
get_dataloader(args.train_root_dir,
, args.valid_root_dir,
, args.train_csv_name,
args.valid_csv_name,
args.num_train_triplets,
args.num_valid_triplets,
args.batch_size, args.num_workers)

        train_valid(model,
optimizer, triplet_loss,
scheduler, epoch, data_loaders,
data_size)
        print(f' Execution time
= {time.time() - time0}')
        print(80 * '=')

def save_last_checkpoint(state):
    torch.save(state,
'log/last_checkpoint.pth')

def train_valid(model, optimizer,
triploss, scheduler, epoch,
dataloaders, data_size):
    for phase in ['train',
'valid']:
        labels, distances = [], []
        triplet_loss_sum = 0.0

        if phase == 'train':
            scheduler.step()
            if
scheduler.last_epoch %
scheduler.step_size == 0:
                print("LR decayed
to:", ', '.join(map(str,
scheduler.get_lr())))
                model.train()
            else:
                model.eval()

            for batch_idx,
batch_sample in
enumerate(dataloaders[phase]):
                anc_img =
batch_sample['anc_img'].to(device)
                pos_img =
batch_sample['pos_img'].to(device)
                neg_img =
batch_sample['neg_img'].to(device)

                pos_cls =
batch_sample['pos_class'].to(device)
                neg_cls =
batch_sample['neg_class'].to(device)

                anc_embed =
model(anc_img).cpu().numpy()
                pos_embed =
model(pos_img).cpu().numpy()
                neg_embed =
model(neg_img).cpu().numpy()

                pos_dist =
l2_dist.forward(anc_embed,
pos_embed)
                neg_dist =
l2_dist.forward(anc_embed,
neg_embed)

                all = (neg_dist -
pos_dist <
args.margin).cpu().numpy().flatten
()
                if phase ==
'train':
                    hard_triplets
= np.where(all == 1)
                    if
len(hard_triplets[0]) == 0:
                        continue
                    else:
                        hard_triplets
= np.where(all >= 0)

                    anc_hard_embed =
anc_embed[hard_triplets]
                    pos_hard_embed =
pos_embed[hard_triplets]
                    neg_hard_embed =
neg_embed[hard_triplets]

                    anc_hard_img =
anc_img[hard_triplets]
                    pos_hard_img =
pos_img[hard_triplets]
                    neg_hard_img =
neg_img[hard_triplets]

                    pos_hard_cls =
pos_cls[hard_triplets]
                    neg_hard_cls =
neg_cls[hard_triplets]

```

```

        anc_img_pred =
model.module.forward_classifier(an
c_hard_img)
        pos_img_pred =
model.module.forward_classifier(po
s_hard_img)
        neg_img_pred =
model.module.forward_classifier(ne
g_hard_img)

        triplet_loss =
triploss.forward(anc_hard_embed,
pos_hard_embed, neg_hard_embed)

        if phase ==
'train':

optimizer.zero_grad()

triplet_loss.backward()

optimizer.step()

        dists =
l2_dist.forward(anc_embed,
pos_embed)

distances.append(dists.data.cpu().
numpy())

labels.append(np.ones(dists.size(0
)))

        dists =
l2_dist.forward(anc_embed,
neg_embed)

distances.append(dists.data.cpu().
numpy())

labels.append(np.zeros(dists.size(
0)))

        triplet_loss_sum
+= triplet_loss.item()

        avg_triplet_loss =
triplet_loss_sum /
data_size[phase]
        labels =
np.array([sublabel for label in
labels for sublabel in label])
        distances =
np.array([subdist for dist in
distances for subdist in dist])

        tpr, fpr, accuracy, val,
val_std, far = evaluate(distances,
labels)

        print(' {} set - Triplet
Loss      = {:.8f}'.format(phase,
avg_triplet_loss))
        print(' {} set - Accuracy
= {:.8f}'.format(phase,
np.mean(accuracy)))
}

        write_csv(f'log/{phase}.csv',
[epoch, np.mean(accuracy),
avg_triplet_loss])

        if phase == 'valid':

save_last_checkpoint({'epoch':
epoch,

'state_dict':
model.module.state_dict(),

'optimizer_state':
optimizer.state_dict(),

'accuracy': np.mean(accuracy),

'loss': avg_triplet_loss

})

        save_if_best({'epoch':
epoch,

'state_dict':
model.module.state_dict(),

'optimizer_state':
optimizer.state_dict(),

'accuracy': np.mean(accuracy),

'loss': avg_triplet_loss

},

np.mean(accuracy))

        else:
            plot_roc(fpr, tpr,
figure_name='./log/roc_valid_epoch
_{}.png'.format(epoch))

if __name__ == '__main__':
    main()

import numpy as np
from scipy import interpolate
from sklearn.model_selection
import KFold

def evaluate(distances, labels,
nrof_folds=10):
    # Calculate evaluation metrics

```

```

        thresholds = np.arange(0, 30,
0.01)
        tpr, fpr, accuracy =
calculate_roc(thresholds,
distances,
labels, nrof_folds=nrof_folds)
        thresholds = np.arange(0, 30,
0.001)
        val, val_std, far =
calculate_val(thresholds,
distances,
labels, 1e-3,
nrof_folds=nrof_folds)
        return tpr, fpr, accuracy,
val, val_std, far

def calculate_roc(thresholds,
distances, labels, nrof_folds=10):
    nrof_pairs = min(len(labels),
len(distances))
    nrof_thresholds =
len(thresholds)
    k_fold =
KFold(n_splits=nrof_folds,
shuffle=False)

    tprs = np.zeros((nrof_folds,
nrof_thresholds))
    fprs = np.zeros((nrof_folds,
nrof_thresholds))
    accuracy =
np.zeros((nrof_folds))

    indices =
np.arange(nrof_pairs)

    for fold_idx, (train_set,
test_set) in
enumerate(k_fold.split(indices)):

        # Find the best threshold
for the fold
        acc_train =
np.zeros((nrof_thresholds))
        for threshold_idx,
threshold in
enumerate(thresholds):
            _, _
            acc_train[threshold_idx] =
calculate_accuracy(threshold,
distances[train_set],
labels[train_set])
            best_threshold_index =
np.argmax(acc_train)
            for threshold_idx,
threshold in
enumerate(thresholds):
                tprs[fold_idx,
threshold_idx], fprs[fold_idx,
threshold_idx], _ =
calculate_accuracy(threshold,
distances[test_set],
labels[test_set])
                _, _, accuracy[fold_idx] =
calculate_accuracy(thresholds[best
_threshold_index],
distances[test_set],
labels[test_set])

        tpr = np.mean(tprs, 0)
        fpr = np.mean(fprs, 0)
        return tpr, fpr, accuracy

def calculate_accuracy(threshold,
dist, actual_issame):
    predict_issame = np.less(dist,
threshold)
    tp =
np.sum(np.logical_and(predict_issa
me, actual_issame))
    fp =
np.sum(np.logical_and(predict_issa
me,
np.logical_not(actual_issame)))
    tn =
np.sum(np.logical_and(np.logical_n
ot(predict_issame),
np.logical_not(actual_issame)))
    fn =
np.sum(np.logical_and(np.logical_n
ot(predict_issame),
actual_issame))

    tpr = 0 if (tp + fn == 0) else
float(tp) / float(tp + fn)
    fpr = 0 if (fp + tn == 0) else
float(fp) / float(fp + tn)
    acc = float(tp + tn) /
dist.size
    return tpr, fpr, acc

def calculate_val(thresholds,
distances, labels, far_target=1e-
3, nrof_folds=10):
    nrof_pairs = min(len(labels),
len(distances))
    nrof_thresholds =
len(thresholds)
    k_fold =
KFold(n_splits=nrof_folds,
shuffle=False)

    val = np.zeros(nrof_folds)

```

```

far = np.zeros(nrof_folds)

indices =
np.arange(nrof_pairs)

for fold_idx, (train_set,
test_set) in
enumerate(k_fold.split(indices)):

    # Find the threshold that
gives FAR = far_target
    far_train =
np.zeros(nrof_thresholds)
        for threshold_idx,
threshold in
enumerate(thresholds):
            ,
far_train[threshold_idx] =
calculate_val_far(threshold,
distances[train_set],
labels[train_set])
            if np.max(far_train) >=
far_target:
                f =
interpolate.interp1d(far_train,
thresholds, kind='slinear')
                threshold =
f(far_target)
            else:
                threshold = 0.0

            val[fold_idx],
far[fold_idx] =
calculate_val_far(threshold,
distances[test_set],
labels[test_set])

            val_mean = np.mean(val)
            far_mean = np.mean(far)
            val_std = np.std(val)
            return val_mean, val_std,
far_mean

def calculate_val_far(threshold,
dist, actual_issame):
    predict_issame = np.less(dist,
threshold)
    true_accept =
np.sum(np.logical_and(predict_issame,
actual_issame))
    false_accept =
np.sum(np.logical_and(predict_issame,
np.logical_not(actual_issame)))
    n_same = np.sum(actual_issame)
    n_diff =
np.sum(np.logical_not(actual_issame))
    if n_diff == 0:
        n_diff = 1

    if n_same == 0:
        return 0, 0
    val = float(true_accept) /
float(n_same)
    far = float(false_accept) /
float(n_diff)
    return val, far

def plot_roc(fpr, tpr,
figure_name="roc.png"):
    import matplotlib.pyplot as
plt
    plt.switch_backend('Agg')

    from sklearn.metrics import
auc
    roc_auc = auc(fpr, tpr)
    fig = plt.figure()
    lw = 2
    plt.plot(fpr, tpr,
color='#16a085',
lw=lw, label='ROC
curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1],
color='2c3e50', lw=lw,
linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive
Rate')
    plt.ylabel('True Positive
Rate')
    plt.title('Receiver operating
characteristic')
    plt.legend(loc="lower right",
frameon=False)
    fig.savefig(figure_name,
dpi=fig.dpi)

```