BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Bab ini memaparkan analisis masalah yang diatasi. Analisis masalah dilakukan dengan menjelaskan pendekatan dan alur kerja dari perangkat lunak yang dikembangkan, implementasi dari metode yang digunakan, dan hasil yang ditampilkan sistem.

3.1 Analisis Masalah

Pada jurnal [1] dijelaskan bahwa pemrosesan dengan metode *modified contrast limited adaptive histogram equalization* pada gambar wajah dengan penggunaan arsitektur *convolutional neural network* berupa *GoogleNet (Inception)*, mencapai tingkat akurasi tertinggi dengan nilai 99.89%. Pada penelitian ini, digunakan dua *benchmark dataset* yaitu, *cropped extended yale face database* dan *komnet dataset*.

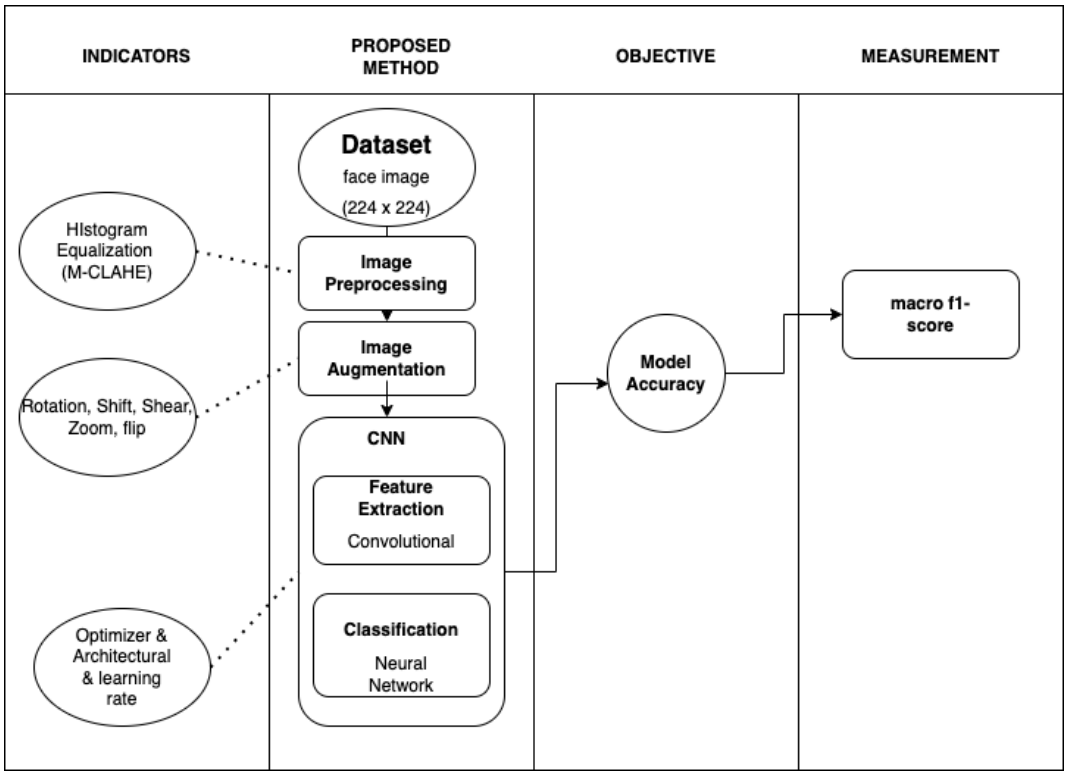
Cropped extended yale face database adalah *yale face database* yang diproses menggunakan algoritma *Viola-Jones* sehingga diperoleh gambar wajah yang menghadap ke depan *(frontal face)*. Sementara *komnet dataset* adalah *dataset* yang dibuat oleh laboraturium politeknik Bali, berisi data gambar *frontal face*.

Selain itu penulis juga menguji dua jenis arsitektur yaitu *Inception* dan *VGG16*, juga menguji lima jenis *optimizer* yaitu *Stochastic Gradient Descent, Nesterov Accelerated Gradient, Adagrad, Adadelta* dan *Adam*. Sebagai pembanding, disiapkan dua jenis *dataset*, original dan yang diproses oleh metode *MCLAHE*. Data gambar selanjutnya diteruskan ke dalam arsitektur *convolutional neural network*, lalu dihitung tingkat akurasinya.

Tinkat akurasi model juga dipengaruhi oleh perubahan *learning rate* saat tahap pelatihan. Meskipun *optimizer* seperti *adagrad*, *adadelta* dan *adam* merupakan *adaptive learning rate* (menggunakan *learning rate* yang dinamis saat pelatihan), menarik untuk diuji apakah perubahan *learning rate* dalam pelatihan model berpengaruh terhadap nilai akurasi. Oleh karena itu, penelitian ini menguji pengaruh perubahan *learning rate* terhadap nilai akurasi pada model yang dilatih dengan arsitektur dan *optimizer* tertentu.

3.2 Kerangka Pemikiran

Gambar 3.1 adalah kerangka pemikiran dari metode yang diusulkan untuk melakukan sistem pengenalan wajah *(face recognition)*.



Gambar 3.1 Kerangka Pemikiran

Berdasarkan Gambar 3.1, penelitian ini dimulai dengan mempersiapkan *dataset* berupa gambar wajah. Gambar wajah nantinya diproses menggunakan algoritma *MCLAHE*. Sebagai pembanding, dipersiapkan juga *dataset* originalnya, nantinya dataset-dataset ini menjadi data latih untuk arsitektur *CNN* yang dibangun. Selain membandingkan jenis *dataset*, jenis *optimizer*, jenis arsitektur *CNN* dan juga perubahan nilai *learning rate* turut diuji pada penelitian kali ini. Penelitian ini bertujuan untuk melihat hasil akurasi *CNN* dalam melakukan pengenalan wajah. Berikut ini dijelaskan setiap bagian pada gambar 3.1.

1. *Indicators* adalah variabel yang mempengaruhi hasil dari metode utama. Indikator dinilai pada saat pengujian *CNN* dimulai. Indikator yang diobservasi selama penelitian ini antara lain sebagai berikut.
   (a) Pemrosesan *MCLAHE* digunakan pada jurnal [1] sebagai pemrosesan awal gambar. Penelitian yang dilakukan pada jurnal [1] menggunakan dataset *Yale* yang merupakan gambar wajah dengan beragam kondisi pencahayaan. *MCLAHE* ditujukan untuk meratakan distribusi intensitas pada gambar. Penelitian ini bertujuan untuk menguji apakah pemrosesan

MCLAHE juga berpengaruh terhadap hasil akurasi model.

MCLAHE adalah singkatan dari *modified contrast limited adaptive histogram equalization*. Merupakan metode pemrosesan gambar melalui proses *CLAHE* yang setelah itu diproses kembali dengan *gaussian blur*. *CLAHE* adalah proses *adaptive histogram equalization* yaitu *histogram equalization* yang berlangsung pada *kernel window* yang digeser ke seluruh *field* gambar. Lalu terdapat atribut *clip limit* yang membatasi frekuensi kemunculan suatu *pixel* pada titik tertentu. Lalu kelebihan dari keseluruhan *pixel* tersebut dibagi merata ke seluruh *pixel* pada *kernel* tersebut, seperti yang dijelaskan pada gambar 2.65. Pada penelitian kali ini dilakukan uji coba mengenai tingkat akurasi model dengan data latih yang sudah diproses dengan *MCLAHE* dengan yang belum. Penggunaan metode ini membagi jumlah frekuensi kemunculan *pixel* pada *channel* L pada gambar. Mempengaruhi tingkat *contrast* gambar.

(b) *Image augmentation* merupakan metode untuk augmentasi gambar. Proses augmentasi yang diaplikasikan pada penelitian ini adalah transformasi rotasi, *shifting, shearing, zoom & flip*. Tujuan dari transformasi ini adalah untuk memperkenalkan *variant* gambar dari satu kelas yang sama pada model *deep learning*. Hal ini mencegah model untuk mempelajari data latih terlalu detail sehingga mencegah terjadinya *overfitting*. Proses augmentasi gambar mengeneralisasi model yang dihasilkan. Jumlah dataset yang digunakan pada penelitian ini, untuk *Yale* adalah 1647 dan untuk Komnet adalah 810. Jumlah batch yang digunakan saat pelatihan adalah 32. Artinya akan ada 51 iterasi untuk *Yale* dan 25 iterasi untuk Komnet pada 1 *epoch*-nya. Data augmentasi diciptakan secara otomatis dengan menggunakan *ImageGenerator* pada pustaka *keras*. *ImageGenerator* menciptakan data augmentasi setiap iterasi *batch* sejumlah nilai *batch*. Ini artinya data sampel untuk Komnet diciptakan sebanyak 25 x 25 x 100 = 62500 data, sementara data sampel untuk *Yale* diciptakan sebanyak 51 X 51 x 100 = 260100 data. Pengalian dengan 100 dikarenakan nilai *epoch* yang dipakai pada penelitian ini adalah 100.

(c) *Architectural* merupakan jenis arsitektur *CNN* yang digunakan saat proses pelatihan model. Jenis arsitektur ini menentukan bagaimana *neural network* mengekstraksi fitur pada gambar yang nantinya dipelajari pada lapisan *artificial neural network* untuk dilakukan proses klasifikasi. Perbedaan pola pada arsitektur *neural network* tentunya hasil fitur yang diperoleh juga berbeda. Penelitian ini bertujuan untuk menganalisa jenis

arsitektur yang menghasilkan akurasi paling tinggi.

(d) *Optimizers* merupakan *hyperparameter* yang bertujuan untuk menurunkan nilai *loss*. Berkaitan erat dengan *learning rate*, terdapat beberapa metode *optimizer* yang menggunakan sistem *decay* atau mengubah nilai *learning rate* setiap *epoch*-nya. Pada penelitian kali ini, jenis *optimizer* yang dipakai adalah seperti yang sudah dijelaskan pada bab 2.1.8 yaitu *Stochastic Gradient Descent, Nesterov Accelarated Gradient, Adagrad, Adadelta* dan *Adam*. Tujuan akhir dari *optimizer* adalah menurunkan nilai *loss*. Tujuan dari penelitian ini adalah menguji pengaruh penggunaan *optimizer* terhadap nilai akurasi model pada arsitektur tertentu.

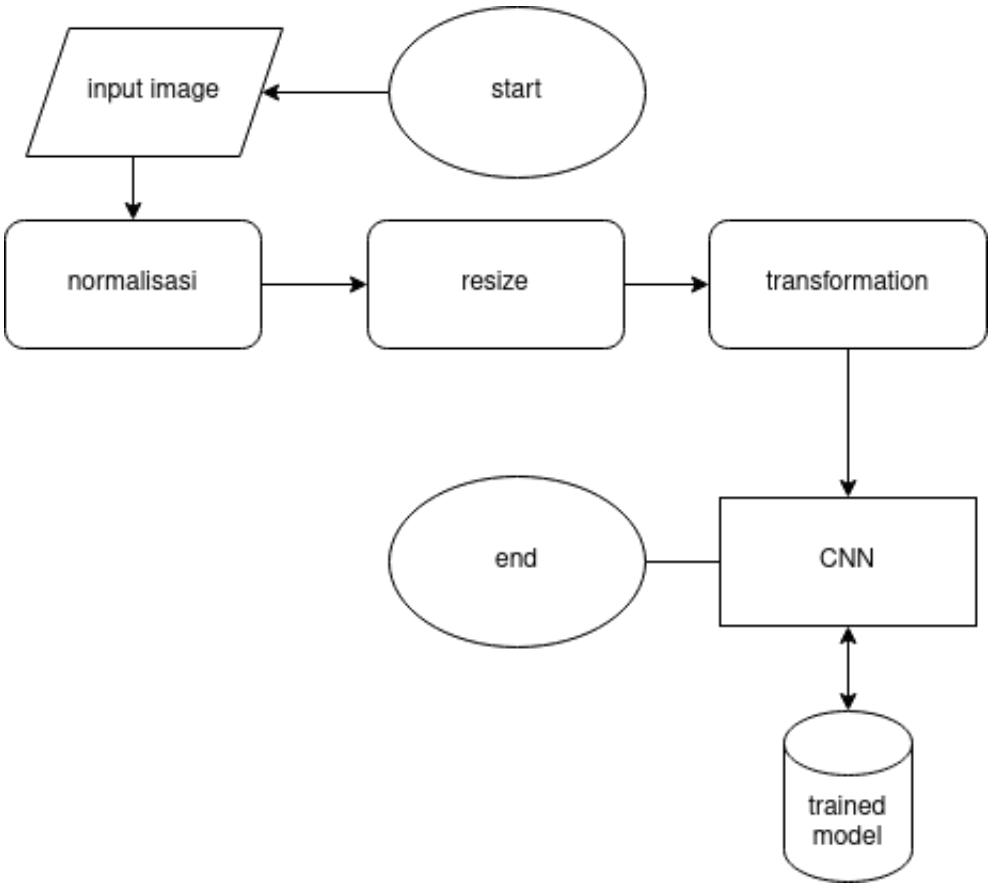
(e) *Learning rate* merupakan *hyperparameter* yang menentukan seberapa besar perubahan pada nilai bobot dan *bias* diaplikasikan pada tahap pelatihan. Gambar 2.19 menjelaskan nilai *learning rate* yang konstan mengakibatkan model membutuhkan waktu lebih lama untuk mencapai konvergen, sementara pada gambar 2.20 menjelaskan bahwa perubahan nilai *learning rate* mempercepat model mencapai konvergen. Pada penelitian kali ini pengujian perubahan nilai *learning rate* menggunakan metode *learning rate scheduler* dan diuji pada arsitektur dan *optimizer* tertentu. Persamaan perubahan *learning rate* yang diuji adalah *step decay*, *time based decay* dan *exponential weight decay*. Penjelasan mengenai persamaan tersebut dapat dibaca pada subbab 2.1.3.2 Penelitian ini bertujuan untuk menguji pengaruh perubahan nilai *learning rate* terhadap nilai akurasi dari model yang dilatih pada arsitektur dan *optimizer* tertentu.

2. *Proposed Method* adalah bagian yang menjelaskan proses penelitian dari awal hingga akhir. Proses ini dimulai dengan mempersiapkan *dataset*. *Dataset* yang ada kemudian diproses dengan menggunakan *MCLAHE*, lalu dilanjutkan dengan meng-*upload dataset* original dengan *dataset* yang sudah diproses ke dalam *google drive*. Lalu *dataset* diteruskan kepada model *CNN*. Sebelum data dilatih oleh model, data terlebih dahulu diaugmentasi. Tujuan dari augmentasi adalah mengaplikasikan regularisasi pada model *CNN*. Tujuan dari regularisasi adalah supaya model yang dihasilkan nanti dapat beradaptasi dengan data lain yang beragam jika dibandingkan dengan data latihnya. Model *CNN* kemudian dilatih dan diuji untuk memperoleh hasil akurasi pengenalan wajah.
3. *Objectives* adalah bagian yang menjelaskan target yang menjadi acuan pengukuran. Dalam penelitian ini, target tersebut adalah akurasi dari pengenalan wajah oleh *CNN*.
4. *Measurement* adalah satuan ukur yang digunakan untuk mengukur hal-hal yang

ada pada bagian *Objectives*. Dalam penelitian ini, karena hendak melakukan *multiclass classification* maka *cost function* yang dipakai adalah *categorical cross entropy*. Pengukuran akurasi menggunakan *macro f1-score*. Pada sistem pengenalan wajah, *false positive* dan *false negative* lebih *crucial* dibandingkan dengan *true positive* dan *true negative*. Karena penting untuk mengetahui mengapa model menebak suatu kelas sebagai kelas yang lain, dibanding hanya mengetahui tebakan benar dan salahnya saja. Oleh karena itu, penelitian ini menggunakan *macro f1-score* sebagai *measurement* [28].

3.3 Analisis Urutan Proses Global

Penelitian ini menggunakan *CNN* untuk melakukan pengenalan wajah. Setelah dilakukan pelatihan, diharapkan arsitektur yang dibangun dapat menghasilkan akurasi yang baik. Gambar 3.2 menunjukkan urutan proses global pada penelitian ini dalam bentuk *flowchart*.



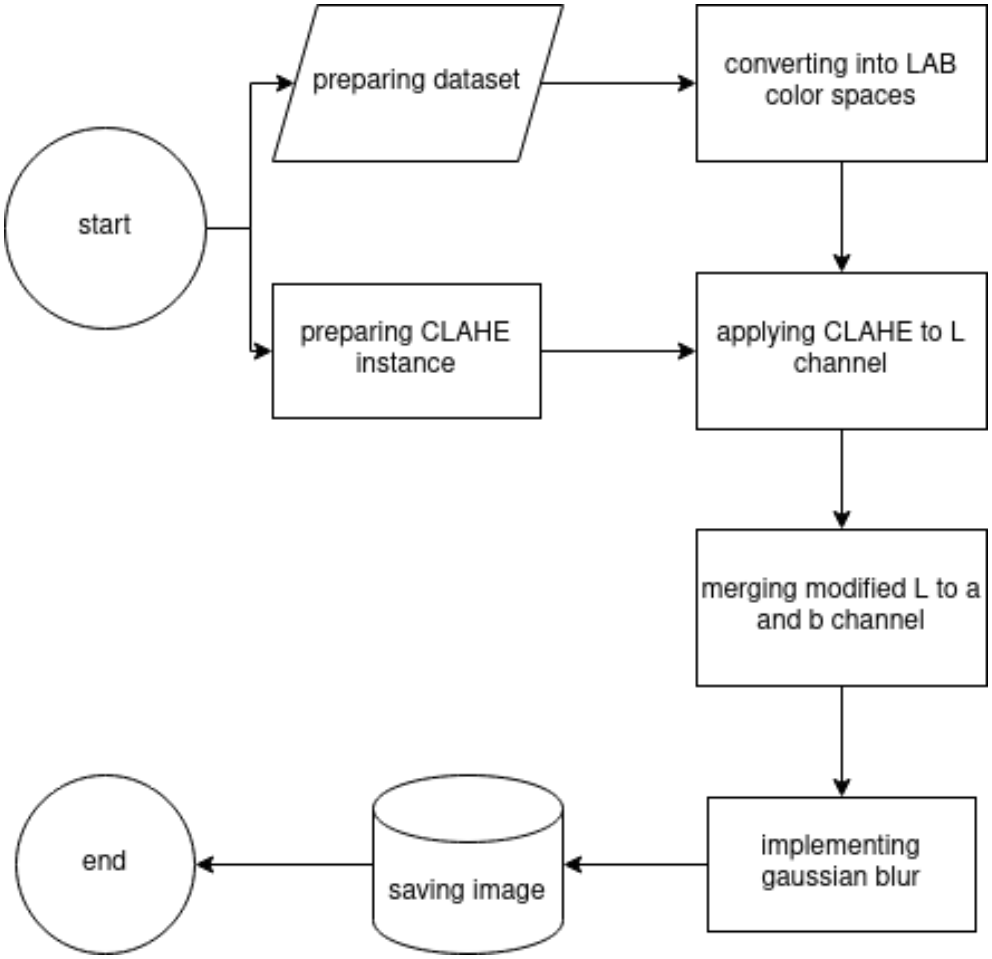
Gambar 3.2 *Flowchart* urutan proses global

Tahapan identifikasi wajah terbagi atas tiga proses, yaitu pemrosesan awal, proses pelatihan dan proses pengujian. Pemrosesan awal dilakukan untuk mengaplikasikan algoritma *Modified Contrast Limited Adaptive Histogram*

Equalization sesuai dengan yang tertulis pada jurnal [1]. Proses pelatihan dilakukan untuk mendapatkan model *CNN* yang optimal untuk menghasilkan akurasi yang tinggi. Proses pengujian dilakukan untuk menguji generalisasi model.

3.3.1 Pemrosesan awal

Pemrosesan awal digambarkan pada 3.3.



Gambar 3.3 *Preprocessing dataset*

Berikut adalah uraian dari proses pada *flowchart* 3.3 yang dilakukan pada penelitian ini.

1. Pada penelitian ini, digunakan dua jenis *dataset* yaitu *Cropped Extended Yale Face Database* dan *Komnet Dataset*. Seperti yang dijelaskan pada subbab 2.3.1, *Cropped Extended Yale Face Database* adalah *Extended Yale Face Database* yang diproses dengan algoritma *Viola-Jones* sehingga hasilnya adalah gambar wajah yang menghadap kedepan *(frontal face)*. *Komnet Dataset*, seperti yang sudah dijelaskan pada subbab 2.3.2, adalah data gambar wajah yang menghadap kedepan *(frontal face)* yang dibuat oleh laboraturium politeknik Bali. *Dataset*

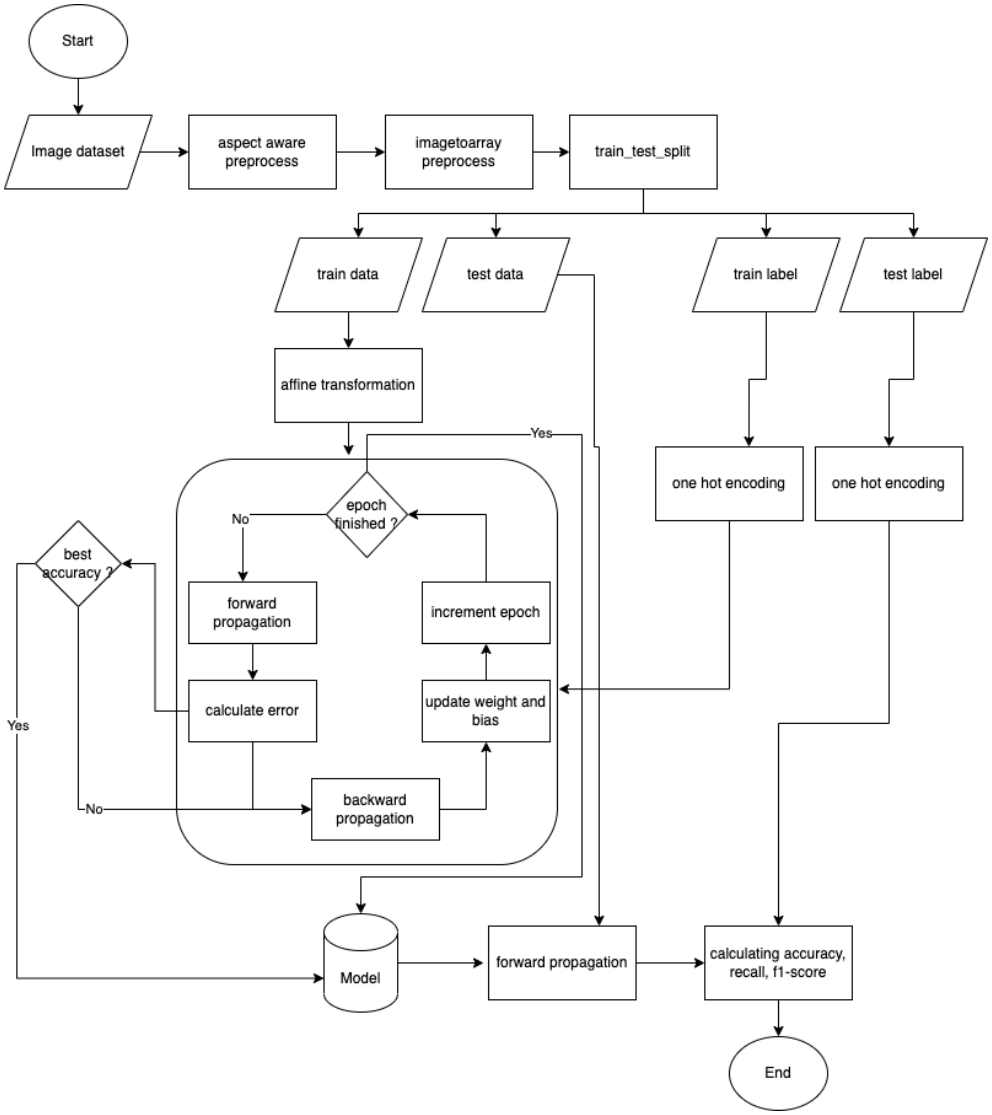
diambil dari banyak sumber, diataranya sosial media, kamera digital dan kamera *smartphone*. Namun, untuk penelitian kali ini hanya menggunakan data gambar yang diambil dari kamera digital.

2. Menyusun *code* untuk mengkonversi gambar dari *BGR channel* ke *LAB channel*. Seperti yang sudah dijelaskan pada subbab 2.1.9.4 *LAB channel* terdiri dari tiga *channel* yaitu, *L* yang merupakan *brightness* dari gambar, *a channel* dan *b channel*. Konversi dari *BGR channel* menuju *Lab channel* dilakukan oleh pustaka *OpenCV* [32]. Algoritma untuk konversi menuju *Lab channel* dapat dilihat pada algoritma 2.3.
3. Mempersiapkan code untuk membangkitkan *CLAHE instance*. Algoritma *CLAHE* diaplikasikan pada gambar dengan menggunakan metode dari *instance* yang dibangun. Gambaran proses dapat dilihat pada 2.1.9.7
4. Selanjutnya mengaplikasikan *MCLAHE* pada gambar sesuai dengan prosedur 2.5

Setelah *CLAHE* diaplikasikan pada gambar, sesuai dengan jurnal [1], gambar selanjutnya diproses dengan menggunakan *gaussian blur*. Selanjutnya hasil gambar yang telah diproses, disimpan pada *disk*. Selanjutnya di-*upload* ke dalam *google drive* untuk dijadikan data latih model *neural network*.

3.3.2 Proses Pelatihan

Pada penelitian ini, proses pelatihan model *CNN* digambarkan pada Gambar 3.4.



Gambar 3.4 *Flowchart* pelatihan sistem pengenalan wajah

Berikut adalah uraian proses pelatihan dari *flowchart* pada Gambar 3.4 yang dilakukan dalam penelitian ini.

1. Sebagai masukan untuk *CNN* yang dibangun, digunakan data gambar wajah *(frontal face)* yang kemudian diubah ukurannya menjadi 224×224 *pixel* lewat proses *AspectAwareRatio* 2.1.9.2
2. Data gambar yang sudah diubah dimensinya kemudian dikonversi ke dalam bentuk larik melalui metode *img_to_array* dari pustaka *keras* 2.3
3. Selanjutnya data dibagi menjadi data uji dan data latih melalui metode pada *scikit-learn* yaitu *train_test_split* 2.9.
4. Label untuk data latih dan data uji masing-masing dikonversi ke dalam bentuk

matrix *(one hot encoding* dengan menggunakan metode *scikit-learn* yaitu *LabelBinarizer* 2.9.

5. Data latih diaugmentasi menggunakan *ImageGenerator* dari *keras*. Proses augmentasi yang dilakukan adalah *random rotation* dengan rentang 0 hingga 30 derajat. Selanjutnya adalah *random shifting* baik pada lebar dan tingginya dengan rentang 0 hingga 10%. Selanjutnya *random shear* dengan rentang 0 hingga 20%. Selanjutnya adalah *random zoom* dengan rentang nilai 0 hingga 20%. Augmentasi *horizontal flip* yaitu mencerminkan gambar berdasarkan sumbu x juga diaplikasikan pada penelitian kali ini. Proses augmentasi ini dijalankan dengan *attribute fill_mode* bernilai *nearest*, ini artinya *pixel* yang kosong karena transformasi gambar diisi dengan nilai *pixel* yang terdekat. Untuk penjelasan mengenai proses augmentasi ini lebih jelasnya dapat dibaca pada subbab 2.1.9.3
6. Selanjutnya adalah tahap pelatihan model. Proses pelatihan model *CNN* terdiri atas dua proses besar, yaitu *forward propagation* dan *backward propagation*. Pada subbab 3.3.2.1 dan 3.3.2.2 dijelaskan tahap pelatihan dengan *CNN* secara *forward propagation* dan *backward propagation*. Untuk pelatihan model dengan menggunakan *generator* digunakan suatu variabel yaitu *steps_per_epoch*. Nilai ini adalah jumlah *generator* menghasilkan gambar *random* satu kali *epoch*-nya.
7. Hasil keluaran dari proses pelatihan model adalah model *deep learning* yang disimpan dalam *disk*.
8. Model yang dihasilkan diuji berdasarkan penilaian *measurement* seperti yang dijelaskan pada subbab 3.2 yaitu *macro f1-score*, maka sebelumnya perlu mendapatkan hasil prediksi model melalui metode *predict*.
9. Setelah hasil dari *model.predict* diperoleh, *macro f1-score* dihitung melalui metode *classification_report* 2.9.

3.3.2.1 *Forward Propagation Convolutional Neural Network*

Berikut ini adalah algoritma dalam proses *forward propagation*. Proses ini dilakukan secara berulang berdasarkan jumlah *epoch*, dan dalam satu *epoch*, ada sejumlah *batch* untuk dilatih. *Batch* merupakan *hyperparameter* yang mengontrol jumlah sampel pelatihan yang harus diproses sistem sebelum nilai bobot dan *bias* diubah. *Epoch* menentukan berapa kali keseluruhan data diproses oleh sistem. Algoritma 3.1 menjelaskan secara umum bagaimana proses *forward propagation* dilakukan.

ALGORITMA 3.1 *Forward Propagation Convolutional Neural Network*

1: Masukan merupakan data wajah yang sudah diproses dengan ukuran 224 × 224 *pixel*.
2: Ubah dimensi gambar ke ukuran 224 × 224 *pixel* menggunakan metode *aspect aware preprocessing* 2.1.9.2
3: Inisialisasi nilai *epoch* dan *learning rate*, *batch_size*, dan *kernel*. Nilai inisiasi mengikuti arsitektur *neural network* yang dipakai yaitu *VGG16* dan *Inception (GoogleNet)*. Inisialisasi bobot menggunakan metode *He Init* berdasarkan persamaan 2.27 dan 2.28.
4: Lakukan *padding* dengan metode *same padding*
5: Lakukan proses konvolusi pada setiap lapisan konvolusi sesuai dengan persamaan 2.2.
6: Operasikan fungsi aktivasi *ReLU* untuk setiap keluaran dari lapisan konvolusi 2.1.5.1
7: Lakukan operasi berikut ini sesuai urutan dari rancangan arsitektur yang dibangun:
   1. Lakukan proses *batch normalization* untuk normalisasi hasil konvolusi 2.1.7.2
   2. Lakukan proses *concatenate* untuk menggabungkan hasil keluaran dari beberapa blok konvolusi.
   3. Lakukan *max pooling* dari lapisan yang telah dikonvolusi untuk mengurangi dimensi keluaran *feature map* 2.15.
   4. Lakukan *average pooling* dari lapisan yang telah dikonvolusi untuk mengurangi dimensi keluaran *feature map*.
8: Lakukan proses *dropout* dan inisialisasi nilai *dropout*. *Dropout* dilakukan agar tidak terjadi *overfitting* 2.1.7.3
9: Lakukan proses *flatten* untuk merubah *tensor* kedalam bentuk larik 1 dimensi agar dapat diproses pada *fully connected layer*.
10: Masukkan seluruh *feature map* setelah proses *flatten* ke dalam *dense layer*. Disini terjadi proses *forward propagation* seperti pada persamaan 2.3
11: Hitung keluaran dari *dense layer* untuk mencari probabilitas dari setiap objek (*face image*) serta menghitung nilai *loss* atau *error* 2.1.3.3
12: Lakukan proses *backward* untuk menghitung nilai gradien lokal dan melakukan perubahan bobot pada setiap lapisan 2.1.3.
13: Simpan model untuk digunakan pada saat pengujian. Karena pada penelitian

ini digunakan *callback* berupa *ModelCheckpoint*, maka hanya model yang terdapat perkembangan dari model sebelumnya yang disimpan. *ModelCheckpoint* merupakan *callback* yang memiliki parameter *file_path* yaitu alamat dimana model disimpan. *monitor* yaitu apa yang hendak dijadikan indikator pengukuran model.*save_best_only* yaitu berupa *boolean* bila bernilai *true* maka hanya menyimpan model yang nilainya paling baik.

14: Untuk pengujian *learning rate* terdapat *callback* lain yang digunakan yaitu *LearningRateScheduler*. User memasukan algoritma perubahan *learning rate* dalam *callback* ini. Pada saat pelatihan, sistem secara otomatis mengubah nilai *learning rate* berdasarkan perhitungan algoritma yang dikonfigurasi pada *learning rate scheduler*. Algoritma konfigurasi yang dipakai pada penelitian ini dapat dilihat pada subbab 2.1.3.2

3.3.2.2 *Backward Propagation Convolutional Neural Network*

backward propagation bertujuan untuk memperbaharui nilai bobot (*weight*) pada *hidden layer neural network* untuk meminimalkan nilai *loss* agar melakukan pengenalan wajah dengan lebih baik. Pembaruan nilai bobot dilakukan dengan menggunakan beragam *optimizer* seperti yang dijelaskan pada bab 2.1.8. Untuk penjelasan matematis bagaimana *backward propagation* berlangsung dapat dilihat pada bab 2.1.3.

ALGORITMA 3.2 *Backward Propagation Convolutional Neural Network without optimizer*

1: Masukan berupa nilai *error* pada *batch* pelatihan sebelumnya.
2: Menghitung nilai error pada masing-masing *hidden layer* dengan cara melakukan operasi *dot product* antara *transpose weight* dengan nilai *cost function*.
3: Mencari turunan pertama dari fungsi aktivasi yang dipakai.
4: Mencari nilai *transpose* dari input masukan
5: mengacu kepada persamaan 2.7, nilai error yang didapat, turunan pertama dari fungsi aktivasi dan *transpose* dari nilai masukan pada persamaan sebelumnya digunakan untuk mengubah nilai bobot.
6: keluaran dari algoritma ini adalah nilai bobot dan bias yang baru.

ALGORITMA 3.3 *Backward Propagation Convolutional Neural Network with optimizer*

1: Masukan berupa nilai *error* pada *batch* pelatihan sebelumnya.
2: Menghitung nilai error pada *hidden layer* dengan cara melakukan operasi *dot product* antara *transpose weight* dengan nilai *cost function*.
3: Menghitung nilai *gradient* pada persamaan 2.20.
4: Menggunakan nilai *gradient* untuk mengkalkulasi bobot dan bias yang baru menggunakan algoritma *optimizer* yang dipakai. Teori mengenai *optimizer* dapat dilihat pada subbab 2.1.8, sedangkan implementasinya dapat dilihat untuk masing-masing *optimizer* yang dipakai pada subbab 3.4.3.
5: keluaran dari algoritma ini adalah nilai bobot dan bias yang baru.

Algoritma *backward propagation* secara garis besar tergambar pada algoritma 3.2. Untuk setiap *optimizer* terdapat rumusan khusus masing masing seperti yang tergambar pada subbab 2.1.8.

3.3.3 Proses Pengujian

Pada penelitian ini, proses pengujian model digambarkan pada Gambar 3.4. Pengujian dilakukan setelah proses pelatihan selesai. Model yang diperoleh dari tahap pelatihan digunakan pada tahap pengujian. Berikut ini adalah uraian proses pengujian dari *flowchart* pada Gambar 3.4 yang dilakukan dalam penelitian ini.

1. Masukan merupakan data gambar wajah hasil dari pemrosesan awal dengan ukuran 224 $\times$ 224 *pixel*. Data yang digunakan adalah data yang sudah dipisahkan dan dikategorikan sebagai data uji.
2. Lakukan proses *one hot encoding* pada label uji dengan menggunakan metode dari *scikit-learn* yaitu *Label Binarizer* 2.9.
3. Ambil model dari *CNN* yang sudah dihasilkan pada proses pelatihan.
4. Gunakan model tersebut untuk melakukan pengenalan wajah terhadap data uji. Model ini digunakan untuk melakukan *forward propagation* pada data pengujian. Proses *forward propagation* pada tahap pengujian sama dengan proses *forward propagation* pada tahap pelatihan pada algoritma 3.1. Hasil dari proses prediksi yang dilakukan model ini adalah larik 1 dimensi *one hot encoding* yang menunjukan klasifikasi dari *datapoint* tersebut.
5. Setelah didapatkan hasil estimasi yang adalah larik 1 dimensi *one hot encoding*, sekuens tersebut dibandingkan dengan label keluaran dari *dataset* untuk

masing-masing *datapoint*. Perbandingan tersebut berlangsung pada sebuah metode yang disebut *classification_report* dari pustaka *scikit-learn* 2.9. Metode ini pada akhirnya mengeluarkan *report* berupa informasi *accuracy, recall* dan *f1-score* dari masing-masing *datapoint*.

3.4 Analisis Manual

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

3.4.1 Analisis *Dataset*

Dataset yang digunakan dalam penelitian ini ada dua jenis, yaitu *cropped extended yale face database* dan *komnet dataset*. Seperti yang sudah dijelaskan pada subbab 2.3.1, *cropped extended yale face database* adalah *extended yale face database* yang diaplikasikan algoritma *Viola-Jones*, sehingga yang dihasilkan adalah gambar wajah yang menghadap ke depan *(frontal face)*. *Cropped extended yale face database* mengandung 38 kelas dengan masing masing kelas terdiri dari 64 gambar dengan beragam jenis iluminasi pencahayaan. Gambar 2.71 adalah contoh dari dataset ini. Dimensi awal dari *cropped Yale* dataset adalah 168×192 pixel. Dengan format data .PNM *(Portable Any Map Image)*, yang menyimpan satu gambar tanpa kompresi.

Dataset berikutnya yang dipakai pada penelitian ini adalah *komnet dataset*. *Dataset* ini dibuat oleh laboraturium politeknik Bali, mengandung 50 kelas dengan masing masing kelas terdiri dari 24 gambar berukuran 224 x 224 *pixel*. Gambar berupa wajah yang menghadap ke depan *(frontal face)*. Gambar diambil dengan menggunakan media kamera digital, sosial media dan juga kamera *smartphone*. Gambar 2.72 adalah contoh dari *dataset* ini. Dimensi awal dari gambar ini adalah 224×224 *pixel* dengan format *jpeg*.

3.4.1.1 *Aspect Aware Preprocessor & Image to Array preprocessor*

Dalam rangka *neural network* dapat bekerja dengan optimal, perlu penyeragaman ukuran dimensi gambar masukan. Teknik manipulasi dimensi gambar yang dipakai pada penelitian kali ini merupakan *aspect aware preprocessor* seperti yang dijelaskan pada algoritma 2.1.

Setelah gambar diubah dimensinya, langkah selanjutnya adalah mengkonversi gambar kedalam bentuk *numpy array*. Metode ini diselesaikan dengan menggunakan pustaka *keras* dari modul *image preprocessing* yaitu *img_to_array* 2.3.

3.4.1.2 *Modified Contrast Limited Adaptive Histogram Equalization*

MCLAHE sejatinya adalah *CLAHE* yang setelahnya diaplikasikan *gaussian blur*. Fungsi dasar dari *MCLAHE* adalah *histogram equalization*, proseduralnya dapat dilihat pada 2.4. Untuk prosedur bagaimana *MCLAHE* diaplikasikan dapat dilihat pada 3.3.1. Pemrosesan *MCLAHE* hanya berlangsung pada *channel L* pada *channel* warna *LAB*. *Channel* warna gambar perlu dikonversi dari *RGB* menjadi *LAB* terlebih dahulu. Pemrosesan dapat dilihat pada subbab 2.1.9.4

3.4.1.3 Augmentasi Dataset

Dataset yang digunakan untuk pelatihan model pada penelitian ini melalui proses augmentasi 2.1.9.3 Pada penelitian ini proses augmentasi menggunakan pustaka *tensorflow*. Augmentasi yang dilakukan pada penelitian kali ini adalah:

1. *Rotation*

   proses rotasi berlangsung dengan menggunakan matriks seperti pada persamaan 2.60. Matriks dikalikan dengan koordinat x dan y pada gambar untuk menghasilkan koordinat baru. θ yaitu rentang varian sudut yang digunakan pada penelitian kali ini bernilai 0 hingga 30.
2. *Shifting*

   proses *shifting* pada gambar berlangsung *horizontal* maupun *vertical*. *Shifting* secara *horizontal* berlangsung sesuai dengan persamaan 2.62, secara *vertical* berlangsung sesuai persamaan 2.63.
3. *Shear*

   proses *shearing* terhadap sumbu x menggunakan matriks seperti pada persamaan 2.66 sementara pada sumbu y menggunakan matriks seperti pada persamaan 2.67. Contoh perhitungan operasi *shear* dapat dilihat pada gambar 2.68.
4. *Zoom*

   proses *zoom* pada penelitian kali ini menggunakan *fill_mode* bernilai *nearest* yang artinya *pixel* terdekat direplikasi sebanyak *n*. Seperti yang tertera pada tabel 2.3, dijelaskan apabila terdapat *pixel* abcd, maka hasil *zoom*-nya adalah aaaaabcddddd. Banyaknya nilai *n* ditentukan oleh rentang nilai pada parameter *width_shift_range* untuk sumbu x maupun *height_shift_range* untuk sumbu y. Nilai tersebut diisi dengan *list* berisi 2 *floating point*, misalkan [0.2,0.3], ini mengartikan bahwa gambar diperbesar pada rentang 20% hingga 30%. Sehingga penambahan jumlah *pixel* disesuaikan.
5. *horizontal flip*

   proses berlangsung dengan menggunakan persamaan 2.64, dapat dilihat

contohnya pada gambar 2.47.

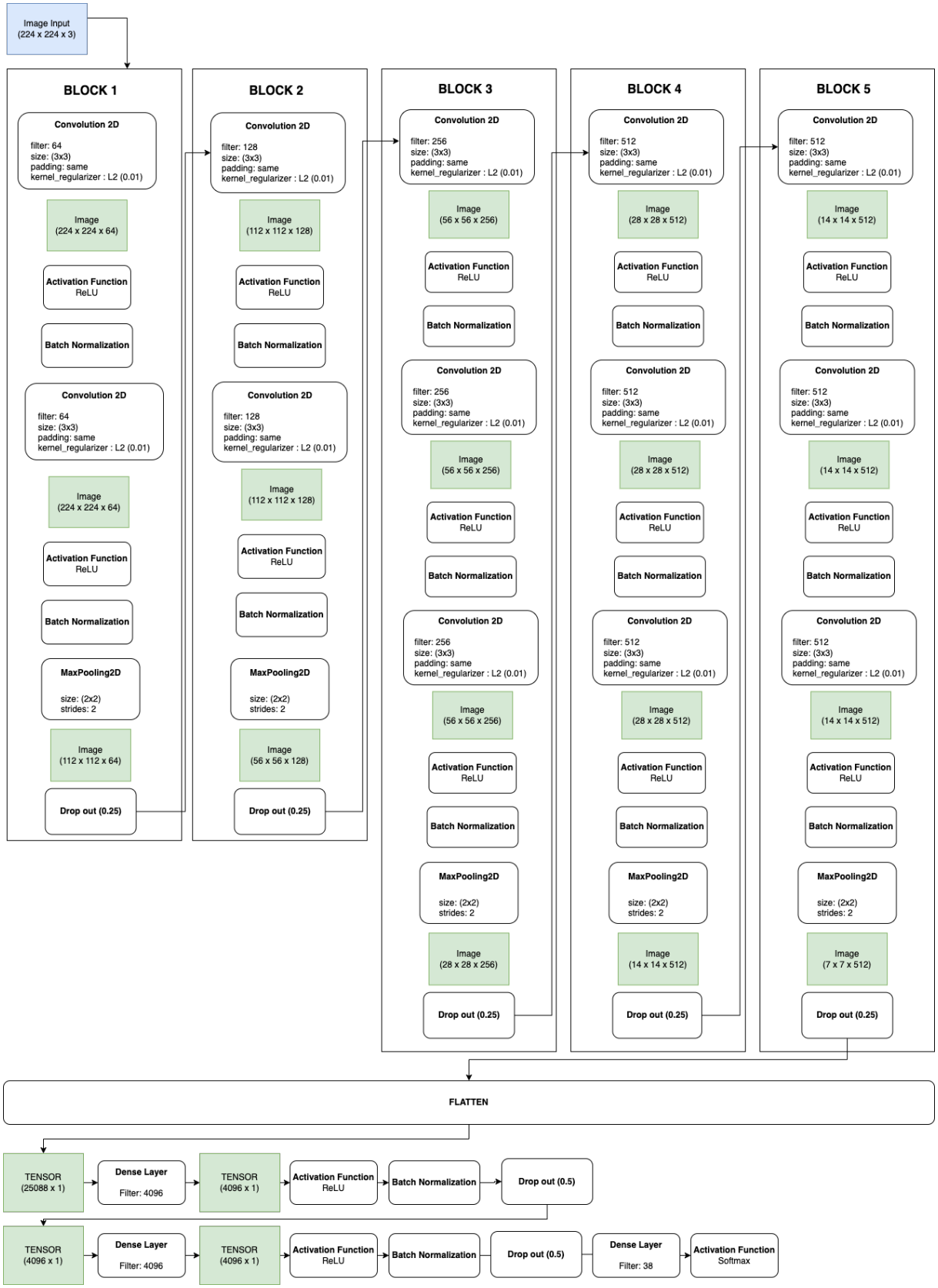
Proses *augmentasi* berlangsung dengan menggunakan *object ImageGenerator* pada pustaka *keras*, data hasil augmentasi tidak disimpan dalam *disk* melainkan diproduksi secara konstan dan langsung digunakan pada pelatihan model.

3.4.2 *Forward Propagation*

Berikut ini dijelaskan perhitungan untuk proses-proses yang dilalui dalam *forward propagation* pada arsitektur *VGG16* dan *Inception*.

3.4.2.1 *Forward Propagation VGG16*

Data gambar pada dasarnya adalah larik dari *pixel*. *Pixel* direpresentasikan sebagai data numerik. Maka representasi gambar oleh komputer adalah berupa data-data numerik yang disimpan dalam bentuk matriks. Contoh nilai matriks citra gambar dapat dilihat pada 3.1. Nilai ini digunakan untuk melakukan operasi konvolusi antara citra dan *kernel*. Pada penelitian ini, operasi konvolusi berjalan secara satu dimensi ke arah samping, hingga akhirnya keseluruhan *pixel* pada gambar berhasil diproses.



Gambar 3.5 arsitektur VGG16

Tabel 3.1 Contoh matriks citra

120	120	100	100	100
120	120	200	100	100
150	150	150	100	100
100	100	100	100	100
50	80	90	100	200

Berdasarkan algoritma 3.1, citra masukan diberikan *padding*. *Padding* digunakan agar citra masukan sepenuhnya tertutup oleh *kernel* sehingga menghasilkan keluaran dengan ukuran yang sama dengan masukan. Tabel 3.2 menunjukkan contoh citra matriks yang sudah diberikan *padding*.

Tabel 3.2 Contoh matriks citra dengan *padding*

0	0	0	0	0	0	0
0	120	120	100	100	100	0
0	120	120	200	100	100	0
0	150	150	150	100	100	0
0	100	100	100	100	100	0
0	50	80	90	100	200	0
0	0	0	0	0	0	0

Kernel yang digunakan dalam penelitian beraneka ragam dimensinya, tergantung pada jenis arsitektur yang dipakai. Pada kasus VGG16 semua *kernel* berukuran 3×3. Pada kasus *Inception*, *kernel* yang dipakai ada yang berukuran 1×1,3×3 maupun 5×5. Dengan mengoperasikan nilai *kernel* kepada citra masukan, didapatkan sebuah *feature map* satu dimensi. Proses konvolusi bergerak secara satu dimensi, di mana pergerakan *kernel* adalah ke arah samping kanan. Nilai *kernel* pada contoh perhitungan ini berukuran 3×3. Nilai awal *kernel* dalam contoh perhitungan manual dan yang digunakan dalam penelitian didapatkan dengan metode *He Initialization* sesuai dengan persamaan 2.27, karena metode ini mendukung penggunaan ReLU.

Tabel 3.3 Contoh nilai *kernel* 3×3

0.14590815	0.02599214	0.04402942
0.00121707	0.15733334	0.05069766
0.13805986	0.00803913	0.11923093

Perhitungan untuk sel [0, 0], sel [0,1], dan sel-sel selanjutnya memiliki pola yang sama yang dapat dilihat pada Persamaan 2.2. Perhitungan dilakukan hingga mencapai sel [x, y] yang berada pada ujung kanan bawah matriks masukan. Hasil perkalian (konvolusi) $w \times x$ disebut sebagai *feature map*. Proses perkalian dilakukan menggunakan *stride* bernilai 1.

Hasil perhitungan operasi konvolusi sesuai dengan Persamaan 2.2. Nilai *bias* yang digunakan adalah 0, di mana digunakan parameter `bias_initializer` = 0. Hasil konvolusi antara citra dan *kernel* pada contoh kasus sel [0, 0] adalah sebagai berikut.

$$
\begin{aligned}
v_k &= (w_{k_{0,0}} x_{0,0} + b_{0,0}) + (w_{k_{0,1}} x_{0,1} + b_{0,1}) + (w_{k_{0,2}} x_{0,2} + b_{0,2}) + (w_{k_{1,0}} x_{1,0} + b_{1,0}) \\
&\quad + (w_{k_{1,1}} x_{1,1} + b_{1,1}) + (w_{k_{1,2}} x_{1,2} + b_{1,2}) + (w_{k_{3,0}} x_{3,0} + b_{3,0}) + (w_{k_{3,1}} x_{3,1} + b_{3,1}) \\
&\quad + (w_{k_{3,2}} x_{3,2} + b_{3,2}) \\
&= (0 * 0.14590815 + 0) + (0 * 0.02599214 + 0) \\
&\quad + (0 * 0.04402942 + 0) + (0 * 0.00121707 + 0) \\
&\quad + (120 * 0.15733334 + 0) + (120 * 0.05069766 + 0) \\
&\quad + (0 * 0.13805986 + 0) + (120 * 0.00803913 + 0) \\
&\quad + (120 * 0.11923093 + 0) \\
&= 39
\end{aligned}
$$

Setelah konvolusi selesai dilakukan, didapatkan *feature map* berukuran 3×3 seperti pada Tabel 3.4. Ukuran *feature map* dapat berubah bergantung pada jenis *padding* yang diaplikasikan, dimensi *kernel* yang digunakan dan juga penggunaan *stride* pada lapisan konvolusi. Jumlah filter berpengaruh kepada jumlah *feature map* yang dihasilkan. Jika filter yang digunakan berjumlah 1048 pada satu lapisan konvolusi, maka dihasilkan 1048 *feature map* pula. Jumlah filter yang digunakan bergantung kepada arsitektur yang dipakai.

Tabel 3.4 Contoh nilai *feature map* hasil konvolusi

39	62	47	46	27
62	86	91	76	40
58	95	82	83	40
50	78	74	89	43
22	41	44	47	49

Berdasarkan jenis arsitektur yang dipakai, setelah proses konvolusi *feature map* kemudian dinormalisasi nilainya menggunakan *batch normalization* pada kasus *Inception*, tetapi pada kasus *VGG16* proses berikutnya adalah ReLU.

Nilai yang ada dalam *feature map* tersebut dimasukkan ke dalam fungsi aktivasi ReLU. Nilai positif tetap dan nilai negatif diubah menjadi nol. Berikut adalah contoh pengubahan nilai pada sel [0,1].

$$\begin{aligned} f(x) &= max(0,x) \\ &= \max(0,39) \\ &= 39 \end{aligned}$$

Dengan menggunakan hasil perhitungan berikut, *feature map* yang dihasilkan menjadi seperti pada Tabel 3.5.

Tabel 3.5 Contoh nilai *feature map* setelah diaktivasi

39	62	47	46	27
62	86	91	76	40
58	95	82	83	40
50	78	74	89	43
22	41	44	47	49

Proses selanjutnya adalah *batch normalization*.Persamaan matematika dari *batch normalization* dapat dilihat pada 2.33. Berikut ini adalah contoh perhitungan untuk sel [0,0]. Adapun nilai gamma γ diinisialisasi 1, nilai beta β diinisialisasi 0, dan nilai epsilon ε diinisialisasi dengan 0.001.

$$
\begin{aligned}
\mu_B &= \frac{39+62+47+...+49}{25} \\
&= 58.84 \\
\sigma_B &= \sqrt{\frac{(39-58.84)^2+...+(58.84)^2}{25}} \\
&= 20.885 \\
\check{x}_i &= \frac{x_i-\mu_B}{\sigma_B+\varepsilon} \\
&= \frac{39-58.84}{20.885+0.001} \\
&= \frac{19.84}{20.886} \\
&= -0.95 \\
y_i &= (\check{x}_i * \gamma)+\beta \\
&= -0.95*1+0 \\
&= -0.95
\end{aligned}
$$

Feature map hasil perhitungan *batch normalization* dapat dilihat pada Tabel 3.6.

Tabel 3.6 Contoh *feature map* hasil perhitungan *batch normalization*

-0.95	0.151	- 0.567	- 0.615	- 1.525
0.151	1.299	1.539	0.8211	- 0.902
-0.041	1.731	1.108	1.156	- 0.902
-0.424	0.917	0.725	1.443	- 0.758
-1.764	- 0.854	- 0.711	- 0.567	- 0.471

Pada arsitektur *VGG16* terdapat 5 blok. Untuk blok yang pertama setelah gambar di konvolusi pada lapisan konvolusi, lalu diaktivasi oleh ReLU dan setelahnya di normalisasi oleh *batch normalization*. Selanjutnya *feature map* diteruskan kembali kepada lapisan konvolusi, ReLU dan *batch normalization*. Selanjutnya *feature map* diteruskan kepada *Max Pooling* dan *dropout layer*. *Max Pooling* yang digunakan pada *VGG16* menggunakan *pool size* = (2,2) dengan *strides* bernilai 2 dan *same padding*.

Berikut adalah contoh perhitungan untuk *max pooling* dimensi 2×2 dengan *same*

padding untuk keluaran pada sel [0, 0]:

$$\begin{aligned} max_{0,0} &= \max\left(x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}\right) \\ &= \max\left(0, 0, -0.95, 0\right) \\ &= 0 \end{aligned}$$

Hasil perhitungan *max pooling* dari *feature map* pada Tabel 3.5 dapat dilihat pada Tabel 3.7.

Tabel 3.7 Contoh hasil perhitungan *max pooling*

0	0	0
0	1.443	0
0	0	0

Dari *feature map* tersebut, selanjutnya dilakukan proses *dropout* dengan menggunakan Persamaan 2.35. Hal ini dilakukan untuk menonaktifkan beberapa nilai fitur agar *neural network* yang dibangun tidak terlalu banyak belajar. Parameter probabilitas yang digunakan adalah 0.25 pada arsitektur *VGG16* dan 0.4 pada arsitektur *Inception*.

Tabel 3.8 Contoh *feature tensor* 1 dimensi

0.125	0.287	0.6281	0.734

Berdasarkan persamaan 2.35, masing masing dari *feature tensor* tersebut mendapat nilai probabilitas secara random dengan metode *uniform distribution* [43]. Misalkan hasil probabilitasnya adalah seperti pada tabel 3.9. Misalkan *dropout rate* yang digunakan adalah 0.25, maka hasil dari proses *dropout* adalah seperti pada tabel 3.10.

Tabel 3.9 Contoh *probability dropout*

0.9	0.1	0.3	0.7

Tabel 3.10 *dropout result*

0.125	0	0.6281	0.734

Sebelum dilalui kepada *fully connected layer*, *feature map* masih harus melalui blok kedua hingga ke lima. Pada blok kedua, prosesnya sama dengan proses pada blok pertama. Jumlah filter pada konvolusi blok pertama adalah 64, sementara pada blok kedua adalah 128. Jika diskemakan adalah sebagai berikut:
$CONV \rightarrow RELU \rightarrow BATCHNORM \rightarrow CONV \rightarrow RELU \rightarrow BATCHNORM \rightarrow MAXPOOLING \rightarrow DROPOUT$.

Proses pada blok ketiga, empat dan lima juga memiliki kemiripan. Perbedaannya hanyalah pada jumlah filter konvolusinya. Pada block ketiga total filternya adalah 256 sementara pada blok ke empat dan ke lima adalah 512. Skemanya adalah sebagai berikut:
$CONV \rightarrow RELU \rightarrow BATCHNORM \rightarrow CONV \rightarrow RELU \rightarrow BATCHNORM \rightarrow CONV \rightarrow RELU \rightarrow BATCHNORM \rightarrow MAXPOOLING \rightarrow DROPOUT$.

Pada *fully connected layer*, pertama tama *feature map* diproses dengan *flatten* sehingga menjadi larik 1 dimensi. Selanjutnya diteruskan kepada *dense layer*.

Tabel 3.11 Contoh *kernel* pada lapisan *dense*

0.0778949	0.0586666	0.05155256	-0.01344849	0.055333	0.0111222	0.009333	0.1232323	0.01222212

Perhitungan pada lapisan ini dapat dilihat sebagai berikut. Pertama-tama, dilakukan perhitungan *dot product* antara masukan *feature map* dengan *kernel*.

$$\begin{aligned} y &= f\left(W^T x + b\right) \\ &= f(0 * 0.0778949 + ... + 0 * 0.0122 + 0) \\ &= f(0.079845519) \end{aligned}$$

Hasil keluaran diteruskan kepada fungsi aktivasi (ReLU) selanjutnya diteruskan kembali kepada *batch normalization* lalu *Dropout* dengan nilai 0.5. Proses ini

berlangsung sebanyak 2 kali. Lalu di akhir lapisan terdapat lapisan keluaran dengan fungsi aktivasi *softmax*.

$$\begin{aligned}\phi_i &= \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ &= \frac{e^{0.0798}}{e^{0.0798} + e^{0} + ... + e^{0}} \\ &= 0.119240563\end{aligned}$$

Hasil akhir perhitungan adalah probabilitas setiap kelas wajah yang terdeteksi pada dataset terkait. Nilai probabilitas yang tertinggi diterima sebagai hasil estimasi

Proses yang sudah dijelaskan di atas belum sempurna sebagai proses pelatihan. Hasil perhitungan masih mengalami *error*. Untuk itu, tahap selanjutnya dari proses pelatihan *CNN* adalah melakukan *backward propagation*. Sebelum masuk ke proses *backward propagation*, dihitung terlebih dahulu nilai *loss* / *error* sesuai dengan persamaan 2.26. Contoh perhitungannya adalah sebagai berikut.

$$\begin{aligned}Loss &= -\log(\phi_i) \\ &= -\log(0.1192405633) \\ &= 0.923575982\end{aligned}$$

Kemudian, hasil regularisasi L2 ditambahkan ke nilai *loss* dengan Persamaan 2.29 untuk bobot dan *bias*. Hasil penjumlahan dengan L2 menjadi nilai *loss* akhir yang dimasukkan ke dalam proses *backward propagation*.

1. Perhitungan L2 untuk *weight kernel* pada *dense layer* adalah sebagai berikut.

$$\begin{aligned} L_{2w} &= \lambda \sum_{m} w_m^2 \\ &= 0.01 * (0.0778949^2 + ... + 0.01222^2) \\ &= 0.01 * 0.38590819 \\ &= 0.003859082 \end{aligned}$$

2. Perhitungan L2 untuk bobot *kernel* pada lapisan konvolusi adalah sebagai berikut.

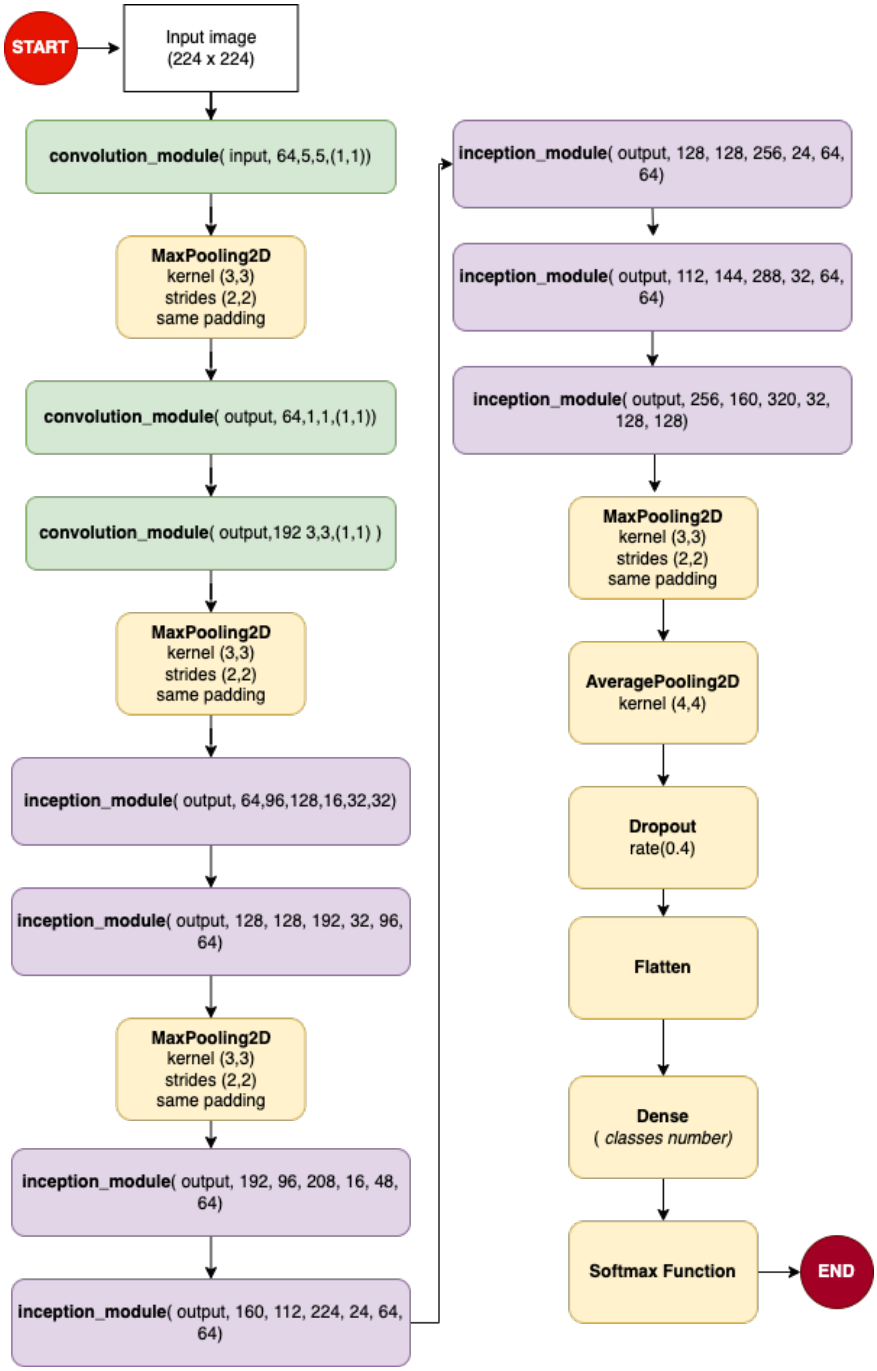
$$\begin{aligned} L_{2w} &= \lambda \sum_{m} w_m^2 \\ &= 0.01 * (0.14590815^2 + ... + 0.11923093^2) \\ &= 0.01 * 0.6905077 \\ &= 0.006905077 \end{aligned}$$

3. Maka, nilai *loss* akhir adalah sebagai berikut.

$$\begin{aligned} Loss &= 0.923575982 + 0.003859082 + 0.006905077 \\ &= 0.934340141 \end{aligned}$$

3.4.2.2 *Forward Propagation Inception*

Perbedaan mendasar antara *VGG16* dengan *Inception* adalah pada jenis modelnya. Pada *VGG16* model berjalan secara *sequential* sementara pada *Inception* berjalan secara paralel. Karenanya penggunaan metode pustakanya berbeda. Pada *Inception* metode yang digunakan adalah *Input*. *Input* memungkinkan *keras* membangun dan menjalankan pelatihan model *neural network* secara paralel.



Gambar 3.6 arsitektur *Inception*

Secara grafis, proses pada arsitektur *Inception* digambarkan pada gambar 3.6. Mengenai penjelasan *inception module* dan *convolution module* dapat dibaca pada subbab 2.1.4.2 Untuk penjelasan lebih mendetail mengenai proses grafik dapat dibaca pada penjelasan berikut.

Pada arsitektur *Inception*, gambar dengan dimensi 224×224 pada mulanya diteruskan kepada lapisan konvolusi dengan filter sebanyak 64, dimensi *kernel* 5×5 dan *strides* bernilai 1. Blok konvolusi ini diberi nama *block1*. Keluaran dari

block1 adalah *feature map* dengan dimensi $224 \times 224 \times 64$. Setelahnya *feature map* diteruskan menuju *Activation Layer*, *Batch Normalization Layer*, lalu dilanjutkan ke *Max pooling layer* dengan dimensi 3×3, nilai *strides* = (2,2) dengan *same padding*. Hasil keluarannya adalah *feature map* dengan dimensi $112 \times 112 \times 64$.

Selanjutnya *feature map* diteruskan kembali ke lapisan konvolusi dengan spesifikasi jumlah filter sebanyak 64, besar dimensi 1×1 untuk mengurangi jumlah parameter seperti yang dibahas pada subbbab 2.1.4.2 dan juga nilai *strides* = (1,1). Blok ini diberi nama *block2*. Keluaran dari *block2* adalah *feature map* dengan dimensi $112 \times 112 \times 64$.Selanjutnya *feature map* diteruskan kembali kepada lapisan konvolusi dengan jumlah filter sebanyak 192, besar dimensi 3×3 dan nilai *strides* = 1. Blok ini diberi nama *block3*. Keluaran dari *block3* adalah *feature map* dengan dimensi $56 \times 56 \times 192$. Jika diskemakan maka menjadi seperti berikut $CONV \rightarrow BATCH \rightarrow RELU \rightarrow MAXPOOLING \rightarrow CONV \rightarrow BATCH \rightarrow RELU \rightarrow CONV \rightarrow BATCH \rightarrow RELU \rightarrow MAXPOOLING$. satu rute dari $CONV \rightarrow BATCH \rightarrow RELU$ adalah 1 modul yang dinamakan *convolution module*. Untuk pembahasan berikutnya setiap kali penyebutan *convolution module* mengacu kepada rute ini.

Selanjutnya *feature map* diproses secara paralel. Diteruskan ke proses yang dinamakan *Inception module*, pada modul ini terdapat 4 pemrosesan, *feature map* diproses secara paralel dan nanti hasilnya digabungkan menjadi satu *feature map*. Modul *inception* yang pertama adalah sebagai berikut :

1. *Feature map* dengan dimensi $56 \times 56 \times 192$ diteruskan kepada *convolution module* dengan nilai filter 64 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3a_first*. Keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 64$.
2. *Feature map* dengan dimensi $56 \times 56 \times 192$ diteruskan kepada *convolution module* dengan nilai filter 96, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3a_second1*. Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 96$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 128, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *3a_second2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 128$.
3. *Feature map* dengan dimensi $56 \times 56 \times 192$ diteruskan kepada *convolution module* dengan nilai filter 16, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3a_third1*. Keluaran dari blok ini adalah *feature map* dengan dimensi

$56 \times 56 \times 16$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 32, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *3a_third2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 32$.

4. *Feature map* dengan dimensi $56 \times 56 \times 192$ diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *3a_pool*. Keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 32$. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 32, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3a_fourth*.Keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 32$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses di atas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran pada *inception module* yang pertama adalah *feature map* dengan dimensi $56 \times 56 \times 256$. Pada proses berikutnya, *feature map* kembali diteruskan kepada *inception module* yang baru, tahapannya adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 128 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3b_first*. Keluaran pada blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 128$.
2. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 128, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3b_second1*. Keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 128$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 192, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *3b_second2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 192$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 32, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3b_third1*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 32$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 96, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *3b_third2*. Hasil keluaran dari modul ini adalah *feature map* dengan dimensi $56 \times 56 \times 96$.
4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *3b_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *3b_fourth*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $56 \times 56 \times 64$.

5. Pada tahap akhir, hasil keluaran dari ke empat proses di atas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran dari *inception module* ini adalah *feature map* dengan dimensi $56 \times 56 \times 480$. Selanjutnya *feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = (2,2), *same padding* dan diberi nama *pool3*. Keluaran dari *pool3* adalah *feature map* dengan dimensi $28 \times 28 \times 480$. Selanjutnya *feature map* diteruskan kembali kepada 5 *inception module*. Modul yang pertama adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 192 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4a_first*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 192$.
2. *Feature map* dilewatkan kepada *convolution module* dengan nilai filter 96, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4a_second1*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 96$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 208, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *4a_second2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 208$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 16, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4a_third1*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 16$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 48, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *4a_third2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 48$.
4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *4a_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4a_fourth*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses diatas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran dari *inception module* ini adalah *feature map* dengan dimensi $28 \times 28 \times 512$. Module *inception* yang kedua adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 160 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4b_first*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 160$.

2. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 112, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4b_second1*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 112$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 224, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *4b_second2*. Hasil keluaran dari blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 224$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 24, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4b_third1*. Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 24$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *4b_third2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *4b_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4b_fourth*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses diatas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran pada *inception module* ini adalah *feature map* dengan dimensi $28 \times 28 \times 512$.Module *inception* yang ketiga adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 128 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4c_first*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 128$.
2. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 128, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4c_second1*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 128$. Nilai keluaran dari fungsi ini diterusan kembali kepada *convolution module* dengan filter 256, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *4c_second2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 256$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 24, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4c_third1*. Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 24$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan

nilai filter 64, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *4c_third2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.

4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *4c_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4c_fourth*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses diatas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran pada *inception module* ini adalah *feature map* dengan dimensi $28 \times 28 \times 512$.Module *inception* yang ke empat adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 112 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4d_first*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 112$.
2. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 144, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4d_second1*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 144$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 288, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *4d_second2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 288$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 32, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4d_third1*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 32$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *4d_third2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *4d_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 64, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4d_fourth*.Hasil keluaran pada block ini adalah *feature map* dengan dimensi $28 \times 28 \times 64$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses diatas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran pada *inception module* ini adalah *feature map* dengan dimensi $28 \times 28 \times 528$.Module *inception* yang ke lima adalah sebagai berikut:

1. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 256 dan dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4e_first*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 256$.
2. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 160, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4e_second1*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 160$. Nilai keluaran dari fungsi ini diteruskan kembali kepada *convolution module* dengan filter 320, dimensi *kernel* 3×3, nilai *strides* = 1 dan diberi nama *4e_second2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 320$.
3. *Feature map* diteruskan kepada *convolution module* dengan nilai filter 32, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4e_third1*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 32$. Keluaran dari fungsi ini kemudian diteruskan kembali kepada *convolution module* dengan nilai filter 128, dimensi *kernel* 5×5, nilai *strides* = 1 dan diberi nama *4e_third2*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 128$.
4. *Feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = 1, *same padding* dan diberi nama *4e_pool*. Lalu keluaran dari fungsi ini diteruskan kepada *convolution module* dengan nilai filter 128, dimensi *kernel* 1×1, nilai *strides* = 1 dan diberi nama *4e_fourth*.Hasil keluaran pada blok ini adalah *feature map* dengan dimensi $28 \times 28 \times 128$.
5. Pada tahap akhir, hasil keluaran dari ke empat proses diatas digabung menjadi 1 *feature map* dan diteruskan ke proses berikutnya.

Hasil keluaran pada *inception module* adalah *feature map* dengan dimensi $28 \times 28 \times 832$.Selanjutnya *feature map* diteruskan kepada *Max Pooling* dengan dimensi *kernel* 3×3, nilai *strides* = (2,2), *same padding* dan diberi nama *pool4*. Hasil keluarannya adalah *feature map* dengan dimensi $14 \times 14 \times 832$.

Selanjutnya *feature map* diteruskan kepada *average pooling* dengan dimensi *kernel* 4×4 dan diberi nama *pool5*. Hasil keluaran dari proses ini adalah *feature map* dengan dimensi $3 \times 3 \times 832$. Lalu *feature map* di *drop out* dengan nilai 0.4. Selanjutnya sebelum memasuki *fuly connected layer*, *fature map* di *flatten* sehingga menjadi larik 1 dimensi dengan jumlah feature 7488. Selanjutnya *feature map* diteruskan ke *dense layer* dengan fungsi aktivasi *softmax*. Hasil keluaran dari

fungsi ini adalah probabilitas terhadap masing-masing kelas klasifikasi.

3.4.3 *Backward Propagation*

Backward propagation pada *CNN* digunakan dalam pelatihan untuk memperbarui bobot dari *kernel*.

3.4.3.1 *Backward propagation* pada *Stochastic Gradient Descent*

Berikut adalah contoh perhitungan menggunakan algoritma *SGD* pada *CNN*:

1. Inisialisasi α = 0.01. Misalkan bobot masukan adalah 0.05981104, dengan asumsi perhitungan dari *loss function* mendapatkan nilai *gradient* g_t 0.934340141 (dapat dilihat pada bagian 3).

2. Perhitungan bobot baru, secara matematika dapat dilihat pada persamaan 2.20. Pada pustaka *keras* perhitungannya mengikuti persamaan 2.23

$$
\begin{aligned}
m_t &= m_0 - \alpha * g \\
m_t &= 0.05981104 - 0.05 * 0.934340141 \\
m_t &= 0.05981104 - 0.046717007 \\
m_t &= 0.01311104
\end{aligned}
$$

3. Nilai bobot sebelumnya adalah 0.05981104 dan diperbarui dengan nilai bobot baru 0.01311104 dalam sekali iterasi pada *neuron* pertama di lapisan keluaran.

4. Ulangi langkah pertama hingga kedua sampai jumlah *epoch* atau iterasi yang ditentukan.

5. Simpan semua bobot, *bias*, lapisan dan *neuron* untuk digunakan pada tahap pengujian.

3.4.3.2 *Backward propagation* pada *Nesterov Accelerated Gradient*

Berikut adalah contoh perhitungan menggunakan algoritma *NAG* pada *CNN*:

1. Inisialisasi α = 0.01, *momentum* = 0.9 dan *Nesterov* = *True*. Misalkan bobot masukan adalah 0.05981104 dan *velocity* awal adalah 0, dengan asumsi perhitungan dari *loss function* mendapatkan nilai *gradient* g_t 0.934340141 (dapat dilihat pada bagian 3).

2. Perhitungan bobot baru, secara matematika dapat dilihat pada persamaan 2.40 dan 2.41.

$$
\begin{aligned}
w_t &= w_0 + m * v_0 - \alpha * g \\
w_t &= 0.05981104 + 0.9 * 0 - 0.01 * 0.934340141 \\
w_t &= 0.05981104 - 0.009343401 \\
w_t &= 0.05051104
\end{aligned}
$$

3. Nilai bobot sebelumnya adalah 0.05981104 dan diperbarui dengan nilai bobot baru 0.01311104 dalam sekali iterasi pada *neuron* pertama di lapisan keluaran.

4. Ulangi langkah kedua hingga ketiga sampai jumlah *epoch* atau iterasi yang ditentukan.

5. Simpan semua bobot, *bias*, lapisan dan *neuron* untuk digunakan pada tahap pengujian.

3.4.3.3 *Backward propagation* pada *Adagrad*

Berikut adalah contoh perhitungan menggunakan algoritma *Adagrad* pada *CNN*:

1. Inisialisasi α = 0.01, *initial accumulator* = 0.1 dan *epsilon* = 0.0000001. Misalkan bobot masukan adalah 0.05981104 dan asumsi perhitungan dari *loss function* mendapatkan nilai *gradient* g_t 0.934340141 (dapat dilihat pada bagian 3).

2. *learning rate* di setiap iterasi mengalami perubahan sesuai dengan persamaan 2.42. Nilai *sigma* adalah nilai *initial accumulator* dari fungsi yaitu 0.1.

$$
\begin{aligned}
\alpha_t &= \frac{\alpha_0}{\sqrt{\sigma_t + \varepsilon}} \\
\alpha_t &= \frac{0.01}{\sqrt{0.1 + 0.0000001}} \\
\alpha_t &= 0.031622761
\end{aligned}
$$

3. Nilai *learning rate* yang dipakai pada iterasi ini adalah 0.031622761. Nilai *learning rate* untuk iterasi berikutnya berubah karena nilai *sigma* dijumlahkan

dengan nilai *gradient* pada iterasi berikutnya.

4. Perhitungan bobot baru, secara matematika dapat dilihat pada persamaan 2.42.

$$
\begin{aligned}
w_t &= w_0 - \alpha * g \\
w_t &= 0.05981104 - 0.031 * 0.934340141 \\
m_t &= 0.05981104 - 0.02896454437 \\
m_t &= 0.03084649563
\end{aligned}
$$

5. Nilai bobot sebelumnya adalah 0.05981104 dan diperbarui dengan nilai bobot baru 0.03084649563 dalam sekali iterasi pada *neuron* pertama di lapisan keluaran.

6. Ulangi langkah kedua hingga kelima sampai jumlah *epoch* atau iterasi yang ditentukan.

7. Simpan semua bobot, *bias*, lapisan dan *neuron* untuk digunakan pada tahap pengujian.

3.4.3.4 *Backward propagation* pada *Adadelta*

Berikut adalah contoh perhitungan menggunakan algoritma *Adadelta* pada *CNN*:

1. Inisialisasi $\alpha = 0.1$, *rho* = 0.9 dan *epsilon* = 0.0000001. Misalkan bobot masukan adalah 0.05981104 dan asumsi perhitungan dari *loss function* mendapatkan nilai *gradient* g_t 0.934340141 (dapat dilihat pada bagian 3).

2. Perhitungan bobot baru, secara matematika dapat dilihat pada persamaan 2.48. Berdasarkan persamaan 2.44, maka nilai dari *exponential weighted average* adalah

$$
\begin{aligned}
E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \\
E[g^2]_t &= 0.9 * 0 + (1 - 0.9) * 0.934340141^2 \\
E[g^2]_t &= 0.1 * 0.872991497 \\
E[g^2]_t &= 0.0872991497
\end{aligned}
$$

Selanjutnya berdasarkan persamaan 2.48 maka perubahan nilai bobotnya adalah sebagai berikut

$$w_t = w_0 - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$w_t = 0.05981104 - \frac{\sqrt{0 + 0.00000001}}{\sqrt{0.0872991497 + 0.0000001}} * 0.93434014$$
$$w_t = 0.05981104 - 0.00031622774$$
$$w_t = 0.05949481226$$

3. Nilai bobot sebelumnya adalah 0.05981104 dan diperbarui dengan nilai bobot baru 0.05949481226 dalam sekali iterasi pada *neuron* pertama di lapisan keluaran. Nilai *weighted average* pada iterasi ini yaitu 0.0872991497 dipakai untuk perhitungan *weighted average* iterasi berikutnya.
4. Ulangi langkah kedua hingga ketiga sampai jumlah *epoch* atau iterasi yang ditentukan.
5. Simpan semua bobot, *bias*, lapisan dan *neuron* untuk digunakan pada tahap pengujian.

3.4.3.5 *Backward propagation* pada *Adam*

Berikut adalah contoh perhitungan menggunakan algoritma *Adam* pada *CNN*:

1. Inisialisasi α = 0.001, β_1 = 0.9, β_2 = 0.999 dan inisialisasi m_0, v_0, dan t dengan nol.
2. Nilai acak bobot pada iterasi *forward propagation* adalah 0.05981104.
3. Asumsi perhitungan dari *loss function* mendapatkan nilai *gradient* g_t 0.934340141 (dapat dilihat pada bagian 3).
4. Perhitungan estimasi momen pertama terhadap bobot yang menerima masukan nilai *gradient* pada lapisan keluaran di *neuron* pertama dengan Persamaan 2.50.

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\ &= 0.9 * 0 + (1-0.9) * 0.934340141 \\ &= 0 + 0.1 * 0.934340141 \\ &= 0.0934340141 \end{aligned}$$

5. Perhitungan Adam pada estimasi momen kedua terhadap bobot yang menerima masukan nilai *gradient* yang sama seperti langkah ketiga yaitu 0.934340141 dengan Persamaan 2.51.

$$\begin{aligned} \upsilon_t &= \beta_2 \cdot \upsilon_{t-1} + (1-\beta_2) \cdot g_t^2 \\ &= 0.999 * 0 + (1-0.999) * (0.934340141)^2 \\ &= 0 + 0.001 * (0.934340141)^2 \\ &= 0.000872991 \end{aligned}$$

6. Menghitung nilai koreksi bobot pada estimasi momen pertama dengan Persamaan 2.53.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ &= \frac{0.0934340141}{1-0.9} \\ &= 0.934340141 \end{aligned}$$

7. Menghitung nilai koreksi bobot pada estimasi momen kedua dengan Persamaan 2.55.

$$\begin{aligned} \hat{\upsilon}_t &= \frac{\upsilon_t}{1-\beta_2^t} \\ &= \frac{0.000872991}{1-0.999} \\ &= 0.872991 \end{aligned}$$

8. Pada tahap ini, nilai bobot yang lama diperbarui dengan nilai bobot yang baru dengan Persamaan 2.57. Nilai w yang merupakan parameter untuk bobot telah diinisialisasi dengan nilai 0.05981104. *Learning rate* (α) yang digunakan pada contoh perhitungan ini adalah 0.001 (10^{-3}), sedangkan nilai *epsilon* yang digunakan pada contoh perhitungan ini (ε) adalah 10^{-7}.

$$
\begin{aligned}
w_t &= w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{\upsilon}_t} + \varepsilon} \\
&= 0.05981104 - 10^{-3} * \frac{0.934340141}{\sqrt{0.872991} + 10^{-7}} \\
&= 0.05981104 - 10^{-3} * \frac{0.934340141}{0,934339874 + 10^{-7}} \\
&= 0.05981104 - 10^{-3} * \frac{0.934340141}{0.934339974} \\
&= 0.05981104 - 10^{-3} * 1.000000179 \\
&= 0.05981104 - 0.001 \\
&= 0.05881104
\end{aligned}
$$

9. Nilai bobot sebelumnya adalah 0.05981104 dan diperbarui dengan nilai bobot baru 0.05881104 dalam sekali iterasi pada *neuron* pertama di lapisan keluaran.

10. Ulangi langkah ketiga hingga kedelapan sampai jumlah *epoch* atau iterasi yang ditentukan.

11. Simpan semua bobot, *bias*, lapisan dan *neuron* untuk digunakan pada tahap pengujian.