**BAB IV**

**HASIL DAN PEMBAHASAN**

# BAB IV: HASIL DAN PEMBAHASAN

## 4.1. Tahap *Pre-processing*

Dasar dari menggabungkan suati sistem keerdasan buatan CNN dengan pengolahan citra adalah *database* dari gambar untuk digunakan pada proses *training*. Namun sebelum itu perlu dilakukannya proses *pre-processing*, agar gambar siap diolah menggunakan algoritma CNN. Pada tahap ini dilakukan beberapa subproses, diantaranya adalah pengambilan gambar, pengubahan tipe warna, thresholding, noise removal, bounding object, pemotongan objek, dan resizing.

Blok diagram pada fase *pre-processing* ini adalah.

Video dari Kamera

Konversi warna GBR to HSV

Thresholding

Operasi Morfologi

Operasi Morfologi

Deteksi Kontur

Bounding box

Deteksi Bounding box terluas

Image cropping

Image resizing  
20x20 pixel

Gambar 4.1. Blok Diagram fase *Pre-Processing\*

Pengambilan gambar pada Open CV dilakukan dengan membuka kamera webcam pada program.

import cv2

cam = cv2.VideoCapture(0)

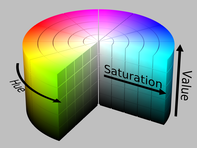
Proses ini akan membuka webcam dengan variabel **cam**.

Dalam proses pendeteksian objek menggunakan warna, maka perlu untuk menyesuaikan tipe warnanya terlebih dahulu. Pada Open CV tipe warna *default* yang digunakan adalah BGR, namun karena perlunya deteksi warna dengan lebih mudah, maka digunakan tipe warna HSV. Untuk mendapatkan sebuah gambar dengan tipe warna HSV, perlu dilakukannya proses konversi warna. Berikut adalah program yang digunakan pada proses konversi warna tersebut.

ret, frame = cam.read()

HSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

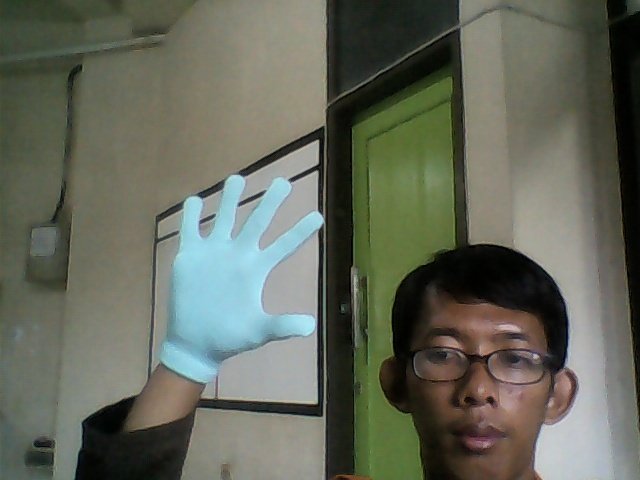
Variabel frame memuat gambar dengan tipe BGR, namun setelah dikonversi dengan cv2.cvtcolor dan metode cv2.COLOR\_BGR2HSV gambar yang ada pada variabel HSV sudah bertipe warna HSV. Tipe warna HSV memiliki tiga komponen yaitu hue, saturation, dan value. Yang mana jika ditampilkan dalam spektrum warna maka akan berbentuk seperti gambar 4.1. berikut.



Gambar 4.1. Spektrum HSV

*(sumber :* [*https://en.wikipedia.org/wiki/HSL\_and\_HSV*](https://en.wikipedia.org/wiki/HSL_and_HSV)*)*

Konversi warna dari BGR ke HSV akan membuat perubahan, perubahan tersebut akibat dari perbedaan spektrum warna pada BGR dan HSV. Berikut adalah contoh konversi pada penelitian ini.



Gambar 4.2 . Gambar asli yang bertipe BGR



Gambar 4.3. Gambar hasil konversi dari BGR ke HSV

Pendeteksian warna dengan metode HSV adalah dengan cara memotong sebagian atau *thresholding* warna dalam rentang tertentu. Penentuan warna yang akan dijadikan rentang tersebut harus sesuai dengan kondisi pencahayaan atau lux pada saat pengambilan gambar. Maka perlu untuk melakukan pengujian terlebih dahulu, dalam hal ini adalah sarung tangan berwarna hijau yang akan dideteksi. Warna sarung tangan tersebut dilakukan beberapa kali pengujian sebagai berikut.

|  |  |  |
| --- | --- | --- |
| **No.** | **Lux** | **Bagian paling gelap** |
| 1. | 60 | (97, 89, 226) |
| 2. | 242 | (92, 53, 252) |
| 3. | 527 | (91, 33, 255) |
| 4. | 182 | (93, 99, 252) |
| 5. | 266 | (89, 79, 255) |
| 6. | 85 | (90, 63, 181) |
| 7. | 162 | (91, 76, 226) |
| 8. | 148 | (86, 28, 196) |
| 9. | 216 | (90, 53, 229) |
| 10. | 378 | (92, 66, 247) |
| **HSV Maksimum** | | (97, 99, 255) |
| **HSV Minimum** | | (86, 28, 181) |

Dari tabel hasil pengujian untuk mencari rentang warna yang akan digunakan pada program, didapat rentang warna terendah hingga tertinggi atau minimum dan maximum yaitu (86, 28, 181) hingga (97, 99, 255) dan dengan memberikan tambahan selisih 10 untuk masing-masing nilai H,S,dan V minimum dan maksimum, maka nilainya jadi (76, 18, 171) hingga (107, 109, 255).

HSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

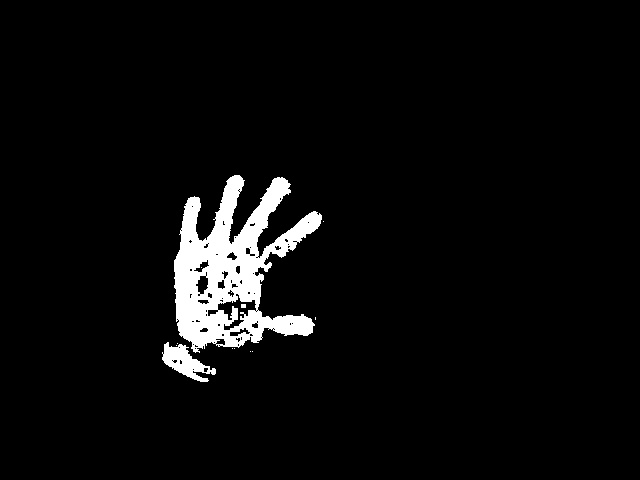
min\_color = np.array([86, 28, 181], dtype = "uint8")

max\_color = np.array([97, 99, 255], dtype = "uint8")

mask = cv2.inRange(HSV, min\_color, max\_color)

berdasarkan program diatas, gambar hasil konversi warna pada variabel HSV dilakukan proses thresholding menggunakan metode cv2.inRange dengan warna minimum pada variabel min\_color, dan warna maksimum pada variabel max\_color.

Proses ini akan mengubah Gambar 4.3. yang merupakan gambar hasil konversi HSV. Menjadi Gambar 4.4. berikut yang merupakan hasil proses thresholding warna pada HSV.



Gambar 4.4. Hasil *thresholding* warna menggunakan metode HSV

Hasil dari proses thresholding adalah gambar biner dengan warna putih merupakan warna yang terdeteksi masuk dalam rentang yang diinginkan. Sedangkan warna hitam adalah gambar yang berwarna lain, atau di luar dari rentang warna tersebut. Terlihat bahwa masih sangat banyak bitnik warna putih kecil yang tersebar, hal ini dikarenakan warna tersebut masuk kedalam rentang warna meskipun bukan pada objek sarung tangan hijau.

Titik putih yang kecil tersebar tersebut biasa disebut dengan *noise*, karena itu adalah hasil *threshold* yang tidak diinginkan. Karenanya perlu dilakukan proses penghapusan *noise* tersebut. Penghapusan *noise* dapat dilakukan menggunakan teknik oprasi morfologi erosi dan dilasi *(closing)*. Syntax yang digunakan untuk proses ini adalah sebagai berikut.

ImgErode = cv2.erode(mask, cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10)))

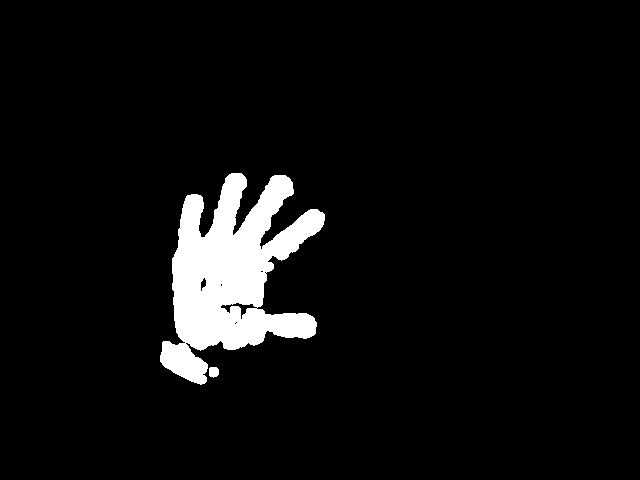
ImgDilate = cv2.dilate(ImgErode, cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10)))

Proses morfologi erosi dilakukan pada variabel ImgErode yang mana menggunakan metode cv2.erode. Proses ini menggambil sumber gambar dari variabel mask sebagai gambar yang perlu dilakukan proses morfologi erosi. Morfologi erosi ini dilakukan dengan elemen ellipse dengan ukuran 10x10 pixel. Hasil dari proses erosi ini ditampilkan seperti pada Gambar 4.4. berikut.



Gambar 4.4. Hasil dari proses morfologi erosi

Lalu setelah dilakukan proses erosi selanjutnya dilakukan proses dilasi menggunakan metode cv2.dilate dengan mengambil gambar sumber dari hasil erosi pada variabel ImgErode. Proses morfologi dilasi ini menggunakan elemen ellipse dan dengan ukuran 10x10 pixel Ukuran pixel yang digunakan sama seperti proses erosi, hal ini dikarenakan perlunya menghilangkan noise tanpa merusak citra pada objek yang sesungguhnya. Hasil dari proses dilasi ini ada pada Gambar 4.5. berikut.



Gambar 4.5. Hasil dari proses morfologi dilasi

Proses ini akan menghasilkan sebuah gambar tangkapan kamera secara penuh. Dikarenakan pada penelitian perlu untuk melakukan proses pendeteksian gestur tangan yang lebih teliti, maka perlu untuk dilakukan pemotongan gambar sehingga hanya menyisakan gambar gestur tangan. Proses ini dapat dilakukan dengan menggunakan beberapa tahap proses. Pertama adalah memberikan *bounding box* pada setiap daerah putih yang merupakan hasil *threshold* dan denoise. Proses ini dilakukan dengan menggunakan syntax berikut.

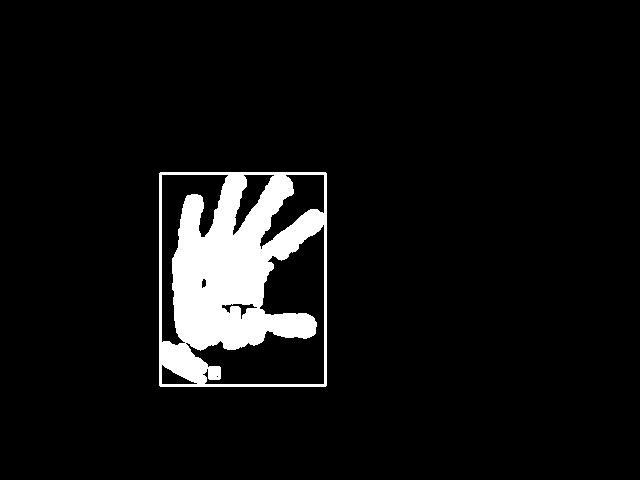
contours, hierarchy = cv2.findContours(ImgDilate,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)[-2:]

for cnt in contours:

x,y,w,h = cv2.boundingRect(cnt)

cv2.rectangle(frame,(x,y),(x+w,y+h),(25,100,255),2)

Pada syntax ini pertama dilakukan deteksi contour terlebih dahulu dengan memasukkan citra hasil proses morfologi closing yaitu ImgDilate. Lalu pada setiap contour yang terdeteksi, dilakukan proses mengkotak-kotakkan tiap bagian yang terdeteksi sebagai objek pada citra tersebut. Pada variabel **contours** akan didapatkan contour dari semua objek yang terdeteksi pada citra, oleh karena itu perlu dilakukan looping for sehingga setiap contour pada objek dapat dilakukan proses terpisah satu persatu untuk mendapatkan bounding box. Hasil dari proses bounding box ini ditunjukkan pada Gambar 4.5. berikut.



a) b)

Gambar 4.5. Hasil proses bounding box a) pada citra hasil dilasi b) jika ditampilkan pada citra asli

Proses ini akan menghasilkan deteksi citra yang menjadi objek hasil secara keseluruhan, sehingga noise yang kecil seharusnya tidak masuk kedalam proses pendeteksian. Oleh karena itu ditambahkan kembali sebuah algoritma untuk menentukan yang mana bounding box yang merupakan tangan pemain. Pada hasil tersebut pada Gambar 4.5. dapat dilihat bahwa sarung tangan mendapat bounding box dengan ukuran luas yang paling besar. Oleh karena itu algoritma yang digunakan berdasar pada luas dari bounding box tersebut.

Untuk menentukan luas dari bounding box digunakan persamaan luas persegi panjang yaitu p x l, yang dalam syntax tersebut adalah w x h. Tambahkan persamaan luas tersebut kedalam syntax.

wh = w\*h

Proses pencarian luas ini berada pada proses looping setiap contour pda objek yang terdeteksi. Untuk mencari luas maksimal dari objek yang terdeteksi pada citra, maka perlu untuk mengumpulkan data dari semua luas objek tersebut terlebih dahulu kedalam sebuah list.

contours, hierarchy = cv2.findContours(ImgDilate,cv2.RETR\_LIST,cv2.CHAIN\_APPROX\_SIMPLE)[-2:]

wh\_box = [0]

coordinateFull = [(0,0,0,0)]

for cnt in contours:

x,y,w,h = cv2.boundingRect(cnt)

cv2.rectangle(frame,(x,y),(x+w,y+h),(25,100,255),2)

cv2.rectangle(ImgDilate,(x,y),(x+w,y+h),(255,255,255),2)

wh = w\*h

wh\_box.append(wh)

coordinateFull.append((x,y,w,h))

indexMaxBox = wh\_box.index(max(wh\_box))

x,y,w,h = coordinateFull[indexMaxBox]

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)

wh\_box.clear()

wh\_box.append(0)

coordinateFull.clear()

coordinateFull.append((0,0,0,0))

Sebelum looping diberikan sebuah list kosong yang akan menampung beberapa hal yaitu luas tiap bounding box yang terdeteksi, yaitu list **wh\_box = [0]** dan sebuah list yang menampung semua koordinat lengkap dari masing-masing bounding box yaitu list **coordinateFull = [(0,0,0,0)]**.

Permulaan dari list ini diberikan angka nol dan bukan sebuah list kosong, hal ini karena penentuan bounding box terluas yang dilakukan akan terus terjadi meski tidak ada objek yang terdeteksi atau bounding box yang dibentuk. Sehingga akan terjadi error ketika list luas bounding box dan koordinatnya kosong.

List wh\_box akan diberikan hasil dari perhitungan luas setiap bounding box, yang mana luast tersebut ditambahkan kedalam list dengan fungsi **wh\_box.append(wh).** Lalu untuk koordinat dari masing-masing objek, juga disimpan pada list dengan fungsi **coordinateFull.append((x,y,w,h))**.

Akan terbentuk sebuah list yang berisikan semua luas bounding box, dan sebuah list yang berisikan semua koordinat bounding box tersebut secara berurutan. Maka untuk mengetahui mana bounding box dengan luas maksimal, digunakan fungsi **indexMaxBox = wh\_box.index(max(wh\_box))**. Fungsi tersebut akan menghasilkan index yang merupakan urutan keberapa luas maksimal yang ada pada list wh\_box. Setelah didapatkan index atau urutan berapa nilai luas maksimum tersebut, selanjutnya adalah mencocokkan index itu untuk memperoleh koordinatnya secara lengkap. Proses pengambilan data pada list padat dilakukan dengan fungsi **x,y,w,h = coordinateFull[indexMaxBox]**.

Untuk menggambarkan bounding box terbesar, persis dibawah variabel yang memberikan koordinat secara lengkap. Dilakukan penggambaran persegi panjang sebagai bounding box dengan metode cv2.rectangle. Dengan variabel koordinat x sebagai letak ujung kiri atas persegi panjang berdasar sumbu x, dan variabel koordinat y sebagai letak berdasarkan sumbu y. Lalu variabel w sebagai panjang dan h sebagai lebar dari persegi panjang tersebut. Maka penggambaran bounding box terluas dapat dilakukan dengan fungsi **cv2.rectangle(frame,(x,y),(x+w,y+h),(****0,0,255),2)**. Hasil dari bounding box maksimal ini ditunjukkan pada gambar 4.5. berikut sebagai sebuah persegi panjang berwarna merah.



Gambar 4.5. Deteksi bounding box terluas

Hal selanjutnya yang perlu dilakukan adalah mengatur ulang atau reset list yang telah terbentuk, agar ketika datang frame baru untuk diolah, tidak ada sisa data dari proses sebelumnya. Terdapat beberapa fungsi yang dilakukan yaitu wh\_box.clear() sebagai penghapus list wh\_box, lalu wh\_box.append(0) untuk mengisi list agar tidak kosong. Kemudian juga untuk koordinat coordinateFull.clear() untuk menghapus data koordinat yang lalu dan coordinateFull.append((0,0,0,0)) untuk menambahkan data nol kedalam list agar tidak terjadi *error*.

Proses selanjutnya adalah pemotongan gambar atau proses *cropping*, proses ini akan menghasilkan sebuah gambar terfokus pada objek yang akan diobservasi lebih lanjut yaitu sarung tangan. Sarung tangan akan dipotong sesuai dengan ukuran *bounding box*-nya. Proses ini menggunakan fungsi berikut.

cropImage = ImgDilate[y:y+h, x:x+w]

Pada fungsi diatas terdapat beberapa hal yang harus diperhatikan. Variabel cropImage adalah variabel yang akan menyimpan gambar hasil pemotongan. Variabel ImgDilate merupakan citra yang akan dipotong, dalam hal ini adalah gambar hasil proses filter morfologi *closing.* Lalu terdapat [y:y+h, x:x+w], ini berarti bagian yang diambil pada citra tersebut merupakan citra yang berada pada pixel sumbu Y dari titik awal y, merupakan variabel penggambaran *bounding box* hingga y+h yang berarti sesuai dengan tinggi bounding box. Begitu pula dengan x:x+w yang berarti pada bagian sumbu X, citra terpotong mulai dari tiitk x permulaan *bounding box* hingga titik x+w yang merupakan panjang dari *bounding box*. Hasil dari proses pemotongan ini adalah pada Gambar 4.5. sebagai berikut.



Gambar 4,5. Hasil pemotongan citra menggunakan ukuran bounding box terbesar.

Proses selanjutnya adalah mengubah ukuran citra atau *resizing*. Proses ini adalah proses akhir dari pre-processing yang mana tujuan dari proses ini adalah mengubah gambar menjadi memeiliki dimensi yang sama, memiliki ukuran yang lebih kecil, dan masih bisa untuk dikenali. Proses ini dilakukan dengan mengubah secara strecth citra hasil *cropping* menjadi ukuran 20x20 pixel. Fungsi python yang digunakan pada proses ini adalah sebagai berikut.

if (cropImage.shape[0] != 0) and (cropImage.shape[1] != 0):

resized = cv2.resize(cropImage, (20,20), interpolation = cv2.INTER\_AREA)

cv2.imshow('CROP RESIZED Camera Vision', resized)

Kode program di atas merupakan algoritma pemotongan gambar pada program ini. Dengan tambahan fungsi **if (cropImage.shape[0] != 0) and (cropImage.shape[1] != 0):** hal ini dimaksudkan agar tidak terjadi *error* apabila tidak ada bounding box yang terdeteksi pada kamera. Fungsi dari **cropImage.shape[0] != 0** dan **cropImage.shape[1] != 0** adalah untuk melihat terlebih dahulu, jika **cropImage.shape[0]** yaitu panjang dari citra hasil cropping adalah nol atau tidak ada citra hasil cropping dari *bounding box*, maka program pengubahan ukuran gambar tidak dilakukan karena jika dilakukan akan terjadi error. Begitu juga dengan pengecekkan tinggi bounding box dengan **cropImage.shape[0],** yang berarti bila tinggi citra hasil cropping adalah nol, atau tidak ada citra yang terdeteksi maka progam pengubah ukuran gambar *(resizing)* dalam fingsi if tidak dijalankan.

Ketika gambar hasil cropping terdeteksi ada, maka selanjutnya adalah melakukan pengubahan ukuran dengan fungsi **resized = cv2.resize(cropImage, (20,20), interpolation = cv2.INTER\_AREA).** Yang mana resized berarti variabel untuk menyimpan gambar hasil resizing. Dengan menggunakan metode cv2.resize gambar hasil cropping yaitu cropImage diubah ukurannya menjadi 20x20 pixel, dengan metode interpolasi cv2.INTER\_AREA.

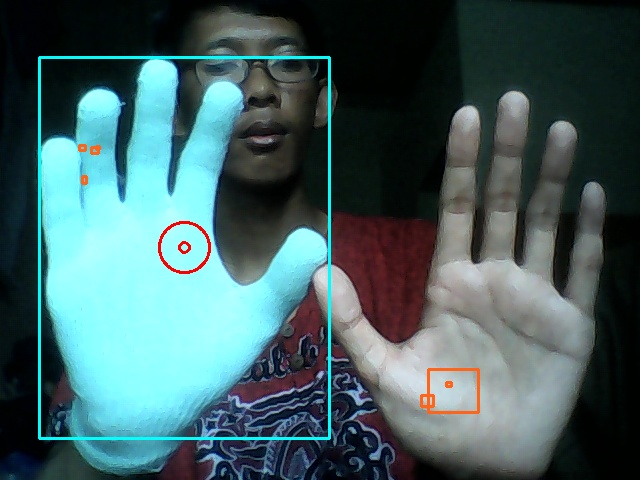
Gambar hasil proses resizing ini ditampilkan dengan fungsi **cv2.imshow('CROP RESIZED Camera Vision', resized),** sehingga hasilnya ditunjukkan seperti pada Gambar 4.5. berikut.



Gambar 4.5. Hasil proses resizing dengan ukuran 20x20 pixel

Pada Gambar 4.5. ukuran citra tersebut menjadi lebih kecil, namun tidak menghilangkan informasi baha citra tersebut masih terlihat sebagai tangan terbuka seperti sebelum hasil resizing.

Dalam penelitian ini digunakan sarung tangan sebagai peralatan tambahan, hal ini bertujuan untuk bisa membedakan antara tangan biasa dengan tangan pemain. Karena pada prakteknya nanti akan ada tangan-tangan yang menjadi noise, dan untuk mengatasinya yaitu salah satunya menggunakan sarung tangan. Berikut adalah Gambar 4.5. yang menunjukkan efek dari sarung tangan dan tangan biasa untuk program pengenalan ini.



Gambar 4.5. Hanya tangan yang menggunakan sarung tangan saja yang bisa terdeteksi oleh program

Penggunaan sarung tangan membuat sistem hanya mendeteksi bagian tersebut saja, sehingga mengurangi noise akibat banyaknya tangan ataupun kulit yang masuk ke kamera webcam.

## 4.2. Pembuatan *Database* Citra

*Convolutional Neural Network* (CNN) adalah salah satu kecerdasan buatan yang menggunakan database sebagai sumber informasinya. Dalam penelitian ini database berisikan citra hasil *pre-processing* yang berukuran 20x20 pixel. Database ini dihasilkan oleh program yang dirancang dengan menggabungkan program pada proses *pre-processing* dengan program tambahan untuk menyimpan gambar.

Proses ini menggunakan program dari *pre-processing* secara *realtime* pada *webcam* dan ditambahkan beberapa baris kode untuk menyimpan gambar. Blok diagram pada proses pembuatan *database* citra ini adalah sebagai berikut.

Citra ukuran 20x20 dari tahap *pre-processing*

Inisialisasi nama file untuk kategori citra

Penyimpanan citra dengan *cv2.imwrite*

Pemisahan citra pada folder sesuai kategori

Gambar 4.4. Blok diagram tahap pembuatan database citra

Tahap pertama adalah melakukan *pre-processing*, sehingga program ini bisa berjalan dan menghasilkan citra yang siap pakai sebagai *database*, bukan citra asli. Citra yang sudah diolah pada tahap *pre-processing* lalu kemudian dijadikan *database* akan lebih baik dan meringankan beban proses *training* nantinya. Hasil dari proses ini adalah citra biner dengan dimensi 20x20 pixel. Berikut adalah program yang digunakan pada proses pembuatan database citra pada penelitian ini.



Gambar 4.4. Kode program untuk proses penyimpanan gambar

Proses inisialisasi nama file akan memudahkan untuk mengidentifikasi citra tersebut oleh manusia. Inisialisasi ini tidak akan berpengaruh untuk proses training, karena kategori pada proses training ditentukan oleh kategori berdasarkan folder. Selain citra hasil crop, pada proses ini juga disimpan beberapa file hasil proses pengolahan citra yaitu pada variabel **bgrFile, hsvFile, maskFile, erodeFile, dilateFile, cropFile, resizedFile**. Lalu semua citra pada variabel tersebut disimpan dengan fungsi cv2.imwrite dengan nama yang berbeda.

Hasil dari proses pembuatan database ini nantinya akan dibagi untuk beberapa kategori dan menyesuaikan tahapnya. Citra tersebut dipisah kategorinya dengan membuat folder yang berbeda. Jumlah citra tiap kategorinya ditunjukkan pada Tabel 4.1. berikut.

Tabel 4.1. Daftar jumlah citra pada *database* dan penggunaannya

|  |  |  |
| --- | --- | --- |
| **Tahap Proses** | **Kategori** | |
| **Tangan Melepas** | **Tangan Menggenggam** |
| *Learning* | 400 | 400 |
| *Validating* | 100 | 100 |
| *Testing* | 100 | 100 |
| **Total Gambar** | **600** | **600** |

## 4.3. Tahap *training* CNN

Setelah terbentuk database citra yang dipisahkan kategorinya menggunakan folder, tahap selanjutnya yang dilakukan adalah training. Tahap ini menentukan proses pengenalan dari sistem kecerdasan buatan dengan metode CNN. Pada tahap ini input berupa citra masuk ke dalam model *layer* CNN untuk proses *update* *weight* dan *loss*. Arsitektur pada CNN ini ditunjukkan pada Bab 3, yaitu Gambar 3.7. yang pada penerapannya akan diubah kedalam kode program python.

Pada tahap *training* ini dilakukan beberapa proses seperti pada blok diagram Gambar 4.4. berikut.

*Training Database*

*Validation Database*

Arsitektur CNN

*Training Generator*

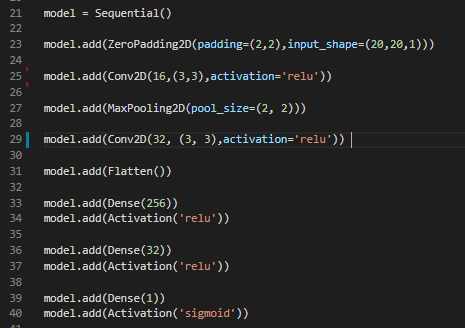
Model CNN

(file .h5)

Gambar 4.4. Blok diagram proses training CNN

Proses ini melibatkan dua buah database yaitu *training database* dan *validation database.* Pada *database training*, terdapat dua buah kategori yang dipisahkan dengan folder, yaitu folder **0-tertutup** dan **1-terbuka.** Folder 0-tertutup berisi 400 database citra tangan tertutup dan folder 1-terbuka berisi 400 citra tangan terbuka.*Output* darimodel CNN adalah sebuah angka 0 atau 1**.** Yang mana *output* dari proses ini meninjau dari urutan *folder* yang dimasukkan kedalam *database* berdasarkan urutan alfabet, sehingga *folder* teratas atau 0-tertutup akan merepresentasikan hasil 0 dan folder 1-terbuka akan merepresentasikan hasil 1. Hal yang sama juga dibentuk pada database validating, yaitu terdapat folder 0-tertutup dan 1-terbuka.

Arsitektur CNN adalah susunan dari *layer* yang akan digunakan pada proses *training*. *Layer* ini tersusun dari beberapa jenis *hidden layer* diantaranya yaitu *zero padding layer, convolution layer, max pooling layer,* dan *flatten layer*. *Layer* ini disusun dengan menggunakan *library* Keras yang merupakan *interface* dari Tensorflow. Berdasarkan rancangan yang te;ah ada sebelumnya mengenai arsitektur ini, kemudian diterapkan pada kode program yang ditunjukkan pada Gambar 4.4. berikut.



Gambar 4.4. Arsitektur CNN yang digunakan pada penelitian

Model ini adalah representasi dari rancangan pada BAB III mengenai arsitektur CNN yang digunakan pada penelitian. Yaitu dengan input citra berukuran 20x20x1 pixel yang berasal dari folder database citra. Dan output dari kode ini adalah sebuah hidden layer yang akan bernilai nol dan satu. Pada program ini nilai dari nol dan satu itu disesuaikan dengan urutan folder yang menjadi input database, nol merupakan kondisi tangan tertutup dan satu adalah kondisi tangan terbuka.

Layer CNN ini selanjutnya akan masuk kedalam proses compile menggunakan perintah model.compile() untuk menghasilkan sebuah struktur CNN utuh. Lalu selanjutnya masuk kedalam proses feedforward dan backpropagation, proses ini bertujuan untuk menyesuaikan weight tiap neuron pada CNN. Pada proses feedforward, data citra dari input akan masuk kedalam setiap neuron yang sudah diatur pada arsitektur CNN, lalu melakukan proses activation pada setiap neuron sebelum selanjutnya diteruskan ke neuron berikutnya. Kemudian terdapat tahap validation, yaitu mencocokkan hasil dari proses training pada neuron output dengan citra yang terdapat pada folder validasi. Hasilnya adalah mengembalikan error pada output kedalam arsitektur CNN, proses pengembalian error dan evaluasi weight ini disebut sebagai proses backpropagation. Hal ini berlangsung terus menerus hingga total epoch yang diinginkan selesai. Proses ini dilakukan dengan menggunakan fungsi model.fit\_generator().

Setelah semua proses training selesai, dan weight pada setiap neuron sudah selesai update, maka proses selanjutnya adalah menyimpan data arsitektur CNN tersebut dan juga nilai weight terbarunya kedalam sebuah file berformat .h5. Proses penyimpanan ini menggunakan perintah model.save().

## 4.4. Tahap testing CNN

Setelah dilakukan proses update weight pada proses training, hal selanjutnya yang perlu untuk dilakukan adalah melakukan tes. File yang digunakan sebagai model arsitektur CNN tersebut adalah file .h5 yang sudah disimpan pada proses training. Model ini terlebih dahulu di load dengan menggunakan fungsi load\_model() pada Keras. Dan selanjutnya dilakukan proses compile terlebih dahulu dengan metode model.compile(). Proses testing ini melibatkan 100 citra asli yang berisikan 50 gambar kondisi tangan tertutup dan 50 gambar kondisi tangan terbuka. 100 citra ini kemudian menjadi input lalu dievaluasi untuk tiap perbedaan epoch pada arsitektur CNN yang digunakan. Berikut Tabel 4.4. yang menunjukkan pengaruh dari jumlah epoch yang digunakan terhadap ketepatan CNN dalam mengidentifikasi kondisi tangan.

Tabel 4.4. Pengaruh epoch terhadap error untuk 100 gambar uji

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Epochs** | **Jumlah Gambar** | **Jumlah Benar** | **Persentase Benar** |
| 1 | 1 | 100 |  |  |
| 2 | 5 | 100 |  |  |
| 3 | 10 | 100 |  |  |
| 4 | 20 | 100 |  |  |
| 5 | 50 | 100 |  |  |
| 6 | 100 | 100 |  |  |

## 4.5. Penerapan CNN Pada Aplikasi Permainan Catur

Cara kerja pyautogui

Koordinasi mouse, mekanisme game

Penjelasan secara umum

**BAB IV**

**HASIL DAN PEMBAHASAN**

# BAB V: PENUTUP

## 5.1. Kesimpulan

## 5.2. Saran