

**SEGMENTASI CITRA MIKROSKOPIS EMBRIO
MENGGUNAKAN *CONVOLUTIONAL NEURAL NETWORK***

LAPORAN TUGAS AKHIR



Disusun oleh:
ADITYA PRATAMA DHARMAWAN
0102516002

**PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS AL AZHAR INDONESIA
2021**

LEMBAR PENYATAAN KEASLIAN

Yang bertanda tangan dibawah ini:

Nama : Aditya Pratama Dharmawan
NIM : 0102516002
Fakultas : Sains dan Teknologi
Program Studi : Informatika
Judul Tugas Akhir : Segmentasi Citra Mikroskopis Embrio Menggunakan
Convolutional Neural Network

Menyatakan bahwa sesungguhnya tugas akhir yang akan dibuat ini belum pernah diajukan pada Perguruan Tinggi atau Instansi manapun untuk tujuan memperoleh gelar akademik tertentu. Saya juga menyatakan bahwa tugas akhir ini benar-benar hasil karya saya sendiri dan tidak mengandung bahan-bahan yang pernah ditulis atau diterbitkan oleh pihak lain kecuali sebagai bahan rujukan yang dinyatakan dalam naskah.

Jakarta, 19 Januari 2021



Aditya Pratama Dharmawan
NIM. 0102516002

LEMBAR PENGESAHAN

Yang bertanda tangan dibawah ini:

Nama : Aditya Pratama Dharmawan
NIM : 0102516002
Fakultas : Sains dan Teknologi
Program Studi : Informatika
Judul Tugas Akhir : Segmentasi Citra Mikroskopis Embrio Menggunakan
Convolutional Neural Network

telah diperiksa dan disetujui oleh :

Jakarta, 19 Januari 2021

Menyetujui,

Pembimbing I



Dr. Ir. Ade Jamal

Pembimbing II



Ali Akbar Septiandri, S.T.,

M.Sc.

Mengetahui,

Ketua Program Studi Informatika
Fakultas Sains dan Teknologi



Riri Safitri, S.Si., M.T.

LEMBAR PERSETUJUAN PENGUJI TUGAS AKHIR

Tanda Tangan

Tanggal

Ketua Sidang :

Dr. Ir. Ade Jamal

18 Februari 2021

.....

Penguji I :

Prof. Dr. Rahmat

Budiarto, M.Sc.

17 Februari 2021

.....

Penguji II :

Ardiansyah Musa

Effendi

18 Februari 2021

.....

Pembimbing I :

Dr. Ir. Ade Jamal

18 Februari 2021

.....

Pembimbing II :

Ali Akbar Septiandri,

S.T., M.Sc.

19 Februari 2021

.....

Dinyatakan Lulus pada Tanggal 4 Februari 2021

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh.

Segala puji dan syukur penulis panjatkan kepada Allah Subhanahu Wa Ta'ala yang telah melimpahkan kasih dan saying-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul “Segmentasi Citra Mikroskopis Embrio Menggunakan *Convolutional Neural Network*” dengan baik dan lancar.

Penyusunan laporan ini dibuat untuk memenuhi salah satu syarat kelulusan dan mendapatkan gelar kesarjanaan Strata Satu (S-1) pada Program Studi Informatika, Fakultas Sains dan Teknologi, Universitas Al Azhar Indonesia. Keberhasilan dalam penyusunan tugas akhir ini tidak akan lepas dari bantuan, dukungan, dan doa dari berbagai pihak, yang telah dengan tulus dan ikhlas menyemangati serta memberi masukan terhadap penulis. Oleh karena itu, dengan segala kerendahan hati penulis ucapan terima kasih kepada:

1. Allah Subhanahu Wa Ta'ala karena selalu melimpahkan banyak kemudahan kepada penulis dalam Menyusun Tugas Akhir ini.
2. Kedua Orang Tua dan dua adik kandung penulis. Terima kasih atas kasih saying, doa, dukungan, dorongan, dan semangat yang selalu diberikan.
3. Bapak Dr. Ir. Ade Jamal dan Bapak Ali Akbar Septiandri,S.T., M.Sc. selaku dosen pembimbing Tugas Akhir yang selalu memberikan ilmu, motivasi, dan masukan-masukan demi terciptanya tugas akhir ini. Mohon maaf apabila penulis masih banyak kekurangannya.
4. Para dosen Informatika UAI yang dengan ikhlas mengerahkan waktu, tenaga, dan pikiran dalam memberikan ilmu selama masa perkuliahan. Semoga dibalas kebaikan dan keberkahan oleh Allah Subhanahu Wa Ta'ala.
5. Teman-teman terdekat penulis, yang tergabung dalam Toxic Team: Galih, Gilang, Jefri, Michael, Andre, Kebon, Aji, dan Bayu yang selalu menghibur ketika penulis sedang merasa jemu.
6. Teman-teman seperjuangan Informatika 2016, yang menemani masa-masa selama kuliah, serta Wita dan Dhean yang selalu menemani dan memberikan masukan kepada penulis saat mengerjakan tugas akhir.

7. Seluruh keluarga besar Informatika UAI yang baik, senior maupun junior. Terima kasih untuk segalanya. Teruntuk Kak Sabil, Kak Sissy, Kak Wilda, Kak Latifah yang menjadikan tugas akhirnya menjadi sumber referensi penulis.
8. Teman-teman penulis (teman yang tegabung dalam grup “Uretra Binams”, teman bermain game, dll) yang tidak dapat disebutkan satu persatu. Terima kasih.

Semoga Allah Subhanahu Wa Ta’ala membalas semua kebaikan, ilmu, serta semangat yang telah diberikan kepada penulis. Penulis menyadari bahwa penulisan laporan ini masih terdapat banyak kesalahan dan kekurangan. Oleh karena itu, besar harapan penulis untuk mendapat masukan, baik saran maupun kritik yang bersifat membangun dari semua pihak yang telah membacanya. Semoga laporan ini dapat memberikan banyak pengetahuan dan manfaat serta inspirasi bagi penulis maupun pembaca.

Wassalamu’alaikum Warahmatullahi Wabarakatuh

Jakarta, 19 Januari 2021



Aditya Pratama Dharmawan
NIM. 0102516002

RIWAYAT HIDUP



Nama : Aditya Pratama Dharmawan
NIM : 0102516002
Agama : Islam
Alamat : Jl. Kalisari, Gg.H.Saiyan No. 29
RT/RW 03/01, Kel. Kalisari, Kec.
Pasar Rebo, Jakarta Timur
Email : aditya93pratama@gmail.com /
adityapratama@if.uai.ac.id

Penulis dilahirkan pada tanggal 25 Oktober 1998 di Jakarta. Penulis merupakan anak pertama dari 3 bersaudara. Penulis melanjutkan Pendidikan di Universitas Al Azhar Indonesia dengan pilihan Program Studi Informatika melalui jalur tes regular.

©Hak cipta milik UAI, tahun 2021

Hak cipta dilindungi

Dilarang mengutip dan memperbanyak tanpa izin tertulis dari

Universitas Al Azhar Indonesia, sebagian atau seluruhnya dalam bentuk apapun,
baik cetak, fotokopi, microfilm, dan sebagainya.

**SEGMENTASI CITRA MIKROSKOPIS EMBRIO
MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK**

ADITYA PRATAMA DHARMAWAN

0102516002

Skripsi

Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer pada
Program Studi Informatika

**PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS AL AZHAR INDONESIA**

2021

ABSTRAK

ADITYA PRATAMA DHARMAWAN. **Segmentasi Citra Mikroskopis Embrio Menggunakan Convolutional Neural Network.**

Bayi tabung (*in vitro fertilization* / IVF) merupakan proses pembuahan di mana sel telur dikombinasikan dengan sperma di luar tubuh. Proses IVF sangat bergantung pada pemilihan embrio. Untuk itu, perlu dilakukan identifikasi kualitas embrio sebelum dipindahkan ke dalam rahim calon ibu. Saat ini, metode pemilihan embrio masih menggunakan penilaian morfologi visual secara rutin untuk mengevaluasi kualitas embrio. Namun, penilaian morfologi masih bergantung dengan kemampuan dan pengalaman seorang dokter atau ahli embrio. Dengan pemanfaatan *Artificial Intelligence* (AI), penilaian embrio otomatis yang tidak memiliki bias merupakan salah satu solusi yang dapat diterapkan. Salah satu metode yang dapat digunakan adalah *Convolutional Neural Network* (CNN), untuk melakukan klasifikasi embrio. Sebelum membangun model klasifikasi, dilakukan *pre-processing* terhadap data yang dimiliki, yaitu salah satunya dengan melakukan segmentasi. Untuk melakukan segmentasi dapat dilakukan dengan dua metode, yaitu *unsupervised segmentation* dan *supervised segmentation*. Pada penelitian ini menggunakan metode *supervised segmentation* yaitu dengan menggunakan salah satu metode dari CNN yaitu *U-Net*. Penelitian ini akan menghasilkan citra embrio yang telah tersegmentasi dan model yang terbaik untuk melakukan segmentasi. Dari hasil evaluasi, model terbaik dari penelitian ini yaitu ketika menggunakan data latih normal dan *batch size* senilai 16, yang memiliki akurasi 99,8% dan jumlah embrio yang berhasil tersegmentasi berjumlah 623 dari 624 embrio.

Kata kunci : Embrio, *in vitro fertilization* (IVF), *Convolutional Neural Network*, *pre-processing*, segmentasi, *U-Net*.

ABSTRACT

ADITYA PRATAMA DHARMAWAN. Embryo Microscopic Image Segmentation Using Convolutional Neural Network

In vitro fertilization (IVF) is a process of fertilization in which an egg cell is combined with sperm outside the body. The IVF process relies heavily on embryo selection. For that, it is necessary to identify the quality of the embryo before it is transferred to the mother's womb. Currently, the embryo selection method still uses visual morphological assessment to evaluate embryo quality. However, the morphological assessment still depends on the ability and experience of a doctor or embryologist. Artificial Intelligence (AI) can be used as a solution for implementing unbiased automatic embryo, to assess the embryo images, the embryo images must go through the preprocessing stages, one of which is segmentation. There are two methods to perform segmentation, using the unsupervised segmentation method and supervised segmentation method. This research focuses supervised segmentation (U-Net) method to segment the embryo image. From the evaluation results, the best model of this research is using normal training data and a batch size of 16, which has an accuracy of 99.8% and the number of successfully segmented embryos is 623 out of 624 embryos.

Keywords : Embryo, *in vitro fertilization* (IVF), *Convolutional Neural Network*, *pre-processing*, segmentation, *U-Net*..

DAFTAR ISI

LEMBAR PENYATAAN KEASLIAN	i
LEMBAR PENGESAHAN	ii
LEMBAR PERSETUJUAN PENGUJI TUGAS AKHIR	iii
KATA PENGANTAR	iv
RIWAYAT HIDUP	vi
ABSTRAK	ix
ABSTRACT	x
DAFTAR ISI	xi
DAFTAR TABEL	xiv
DAFTAR GAMBAR	xv
BAB I PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	2
1.3. Tujuan Penelitian.....	2
1.4. Manfaat Penelitian.....	2
1.5. Batasan Masalah.....	3
1.6. Metodologi Penelitian	3
1.7. Sistematika Penulisan	4
BAB II LANDASAN TEORI	5
2.1. Embrio	5
2.1.1. Tahapan perkembangan embrio	5
2.2. Data <i>Pre-processing</i>	6
2.3. Segmentasi.....	7
2.3.1. Metode Segmentasi	7

2.3.2.	Kategori Segmentasi	8
2.3.3.	Hasil Segmentasi	9
2.4.	Neural Networks.....	10
2.4.1.	Weight dan Bias	11
2.4.2.	Fungsi Aktivasi	11
2.4.3.	Convolutional Neural Network (CNN).....	13
2.4.4.	SegNet.....	17
2.4.5.	U-Net.....	17
	BAB III METODE PENELITIAN.....	19
3.1.	Deskripsi Sumber Data.....	19
3.2.	Library / Modul yang digunakan selama penelitian	20
3.2.1.	Pandas	20
3.2.2.	Matplotlib.....	20
3.2.3.	Open CV2	20
3.2.4.	Numpy.....	21
3.2.6.	Tensorflow Keras	21
3.3.	Data <i>Pre-processing</i>	22
3.2.1.	Image Enhancement	22
3.2.2.	Augmentasi Data	22
3.2.3.	Masking.....	23
3.4.	Segmentasi menggunakan U-Net	23
3.5.	Data <i>Post-processing</i>	23
3.5.1.	Mengubah citra yang tersegmentasi menjadi citra asli	23
3.5.2.	Cropping.....	23
3.5.3.	Padding.....	24
3.6.	Evaluasi	24

BAB IV HASIL DAN PEMBAHASAN	25
4.1. Pre-processing	25
4.1.1. Image Enhancement	25
4.1.2. Augmentasi	26
4.1.3. Masking.....	27
4.2. U-Net	27
4.3. Post-processing.....	30
4.3.1. Mengubah citra yang tersegmentasi menjadi citra asli	30
4.3.2. Cropping.....	31
4.4. Analisis Hasil Uji	32
BAB V KESIMPULAN DAN SARAN.....	37
5.1. Kesimpulan.....	37
5.2. Saran	38
REFERENSI	39
LAMPIRAN	42

DAFTAR TABEL

Tabel 1. Jumlah Embrio yang tersegmentasi berdasarkan Batch Size.....	32
Tabel 2. Sampel Embrio yang telah terpotong berdasarkan Batch Size	33
Tabel 3. Success rate dan Error Rate model berdasarkan Batch Size	33
Tabel 4. Jumlah Embrio yang tersegmentasi berdasarkan Data Latih yang digunakan.....	34
Tabel 5. Sampel Embrio yang telah terpotong berdasarkan Data Latih yang digunakan.....	34
Tabel 6. Success rate dan Error Rate model berdasarkan Data Latih yang digunakan	35
Tabel 7. Grafik Kurva Pelatihan	35

DAFTAR GAMBAR

Gambar 1. Konsep proses segmentasi [6]	7
Gambar 2. Supervised Segmentation [7]	8
Gambar 3. Object Detection [9]	10
Gambar 4. Semantic Segmentation (kiri) dan Instance Segmentation (kanan) [9]	10
Gambar 5. Arsitektur Neural Networks [10].....	11
Gambar 6. Fungsi Sigmoid [12].....	12
Gambar 7. Fungsi Tanh [12]	12
Gambar 8. Fungsi ReLu [12]	13
Gambar 9. Contoh proses no zero padding, unit strides [14].....	15
Gambar 10. Contoh proses zero padding, unit strides [14].....	16
Gambar 11. Contoh proses half (same) padding [14]	16
Gambar 12. Contoh proses full padding [14].....	16
Gambar 13. Arsitektur SegNet [15]	17
Gambar 14. Arsitektur U-Net [17]	18
Gambar 15. Workflow Penelitian	19
Gambar 16. Sampel data	20
<i>Gambar 17. Contoh Citra yang telah dilakukan Image Enhancement</i>	25
Gambar 18. Histogram Citra Normal – Citra Enhancement	26
<i>Gambar 19. Contoh citra yang telah teraugmentasi</i>	26
Gambar 20. Contoh citra yang telah di-masking	27
Gambar 21. Contoh output yang dihasilkan tiap layer U-Net.....	30
Gambar 22. Contoh citra yang telah diubah menjadi citra asli	31
Gambar 23. Contoh citra yang telah terpotong	31
Gambar 24. Contoh citra yang telah ter-padding	32

BAB I

PENDAHULUAN

1.1. Latar Belakang

Bayi tabung atau yang dikenal dengan nama dalam istilah kedokteran, yaitu fertilisasi *in vitro* (Bahasa Inggris : *in vitro fertilization*, IVF) merupakan proses pembuahan di mana sel telur dikombinasikan dengan sperma di luar tubuh, secara *in vitro* (“dalam kaca”). Proses ini melibatkan pemantauan dan merangsang proses ovulasi wanita, mengeluarkan sel telur dari ovarium wanita dan membiarkan sperma membuaohnya pada laboratorium, dan diikuti dengan penanaman embrio langsung ke dalam rahim pengganti.

Sekitar 15% populasi dunia menderita infertilitas, menghalangi atau sepenuhnya mencegah reproduksi alami. [1] IVF dapat digunakan untuk mengatasi kesulitan para pasangan untuk hamil secara permanen. Tingkat keberhasilan IVF bergantung pada faktor-faktor variabel seperti usia ibu, penyebab infertilitas, status embrio, riwayat reproduksi, dan faktor gaya hidup. Usia ibu yang lebih muda akan lebih mungkin untuk hamil (di bawah 41 tahun), dan wanita yang sebelumnya sudah pernah hamil akan lebih berhasil dibandingkan dengan wanita yang tidak pernah hamil.

Salah satu penyebab rendahnya tingkat keberhasilan IVF disebabkan oleh kualitas embrio yang kurang baik. Kualitas embrio ditentukan oleh dua faktor pembentuknya, yaitu sel telur dan sperma. Oleh karena itu, para pasangan harus menjaga gaya hidupnya untuk menghasilkan kualitas embrio yang baik. Ketidakpastian dalam kualitas embrio dapat diatasi dengan menanamkan banyak embrio, sehingga menghasilkan kehamilan dan tidak adanya komplikasi yang tidak diinginkan.

Saat ini, metode pemilihan embrio masih menggunakan penilaian morfologi visual secara rutin untuk mengevaluasi kualitas embrio. Namun, penilaian morfologi masih bergantung dengan kemampuan dan pengalaman seorang dokter atau ahli embrio. [2] Pemanfaatan *Artificial Intelligence* (AI) untuk penilaian embrio otomatis yang tidak memiliki bias masih belum

diterapkan. Dengan menggunakan AI, proses pemilihan embrio terbaik untuk calon bayi akan lebih mudah untuk diproses.

Beberapa penelitian mengenai embrio mulai berkembang. Klasifikasi menggunakan pembelajaran mesin merupakan salah satu metode yang digunakan. Algoritma *Convolutional Neural Network* (CNN) merupakan pilihan yang paling banyak digunakan, karena parameter dan *hyper* parameter CNN lebih banyak dibandingkan dengan algoritma pembelajaran mesin lainnya. *Transfer Learning* juga merupakan pilihan alternatif untuk digunakan, karena model *neural network* tidak perlu dibangun dari awal, seperti penggunaan Xception pada penelitian [1] dan penggunaan Inception untuk membangun model klasifikasi STORK. [3]

Penelitian ini berfokus untuk membangun model dengan arsitektur CNN dengan memodifikasi model yang sudah ada untuk melakukan segmentasi citra sebagai awalan untuk melakukan klasifikasi embrio. Hasil dari model tersebut yaitu embrio dapat tersegmentasi

1.2. Rumusan Masalah

Rumusan masalah berdasarkan uraian latar belakang yang telah diuraikan sebelumnya adalah :

1. Bagaimana melakukan *pre-processing* terhadap citra mikroskopis embrio?
2. Bagaimana membangun model *machine learning* untuk melakukan segmentasi pada citra mikroskopis embrio?
3. Bagaimana hasil evaluasi dari model yang telah dibangun?

1.3. Tujuan Penelitian

Tujuan penelitian ini adalah untuk menemukan model terbaik untuk melakukan segmentasi dari citra mikroskopis embrio.

1.4. Manfaat Penelitian

Manfaat penelitian ini adalah:

1. Membantu dalam melakukan *pre-processing* citra mikroskopis embrio, sehingga dapat mengoptimasi pekerjaan suatu model untuk melakukan klasifikasi.

2. Membantu mengambil informasi penting dari citra mikroskopis embrio.

1.5. Batasan Masalah

Agar penyusunan penelitian ini tidak keluar dari pokok permasalahan yang dirumuskan, maka ruang lingkup pembahasan dibatasi pada :

1. Data yang digunakan merupakan citra mikroskopis embrio yang diperoleh dan mendapatkan izin dari *Indonesian Medical Education and Research Institute* (IMERI, FKUI).
2. Citra embrio yang digunakan memiliki 3 tingkatan kualitas, yaitu *Grade 1*, *Grade 2*, dan *Grade 3*.
3. Hasil dari penelitian merupakan model CNN, nilai untuk evaluasi model (akurasi), dan citra mikroskopis embrio yang telah tersegmentasi.

1.6. Metodologi Penelitian

Berikut ini merupakan penjelasan metodologi penelitian yang akan dilakukan:

1. Data citra mikroskopis embrio

Data yang digunakan pada penelitian ini berupa citra mikroskopis embrio dari hasil laboratorium *Indonesian Medical Education and Research Institute* (IMERI, FKUI).

2. Data *Pre-processing*

Tahap *pre-processing* yaitu tahap melakukan pembuatan *masking* pada citra mikroskopis embrio untuk terlihat lebih jelas.

3. Membangun Model

Membangun model untuk melakukan segmentasi dari data yang telah dilakukan *pre-processing*.

4. Hasil dan Evaluasi

Hasil dari model yang dibangun berupa citra mikroskopis embrio yang telah tersegmentasi dan evaluasi model berupa akurasi sebagai pengukuran performa model.

1.7. Sistematika Penulisan

BAB I : PENDAHULUAN

Pada bab ini, dijelaskan latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, metodologi penelitian, serta sistematika penulisan.

BAB II : LANDASAN TEORI

Pada bab ini, menjelaskan teori-teori yang berkaitan dengan penelitian dan menjadi acuan dalam penelitian yang dilakukan, yaitu melakukan segmentasi embrio.

BAB III : METODOLOGI PENELITIAN

Pada bab ini, menjelaskan metodologi yang digunakan penulis untuk melakukan penelitian.

BAB IV : HASIL DAN PEMBAHASAN

Pada bab ini, menjelaskan hasil dari penelitian, berupa eksplorasi, implementasi, dan evaluasi terhadap model yang telah dibangun.

BAB V : KESIMPULAN DAN SARAN

Pada bab ini, dituliskan kesimpulan akhir dari penelitian yang dilakukan dan saran untuk pengembangan penelitian selanjutnya.

BAB II

LANDASAN TEORI

2.1. Embrio

Embrio merupakan sel atau organisme yang hidup pada masa awal pertumbuhan yang tidak dapat bertahan hidup sendiri. Pertumbuhan embrio terjadi setiap harinya, dengan ditandai berubahnya bentuk dan bertambahnya sel-sel di dalam embrio. Sel-sel itulah yang nantinya akan berkembang menjadi organ tubuh dan membentuk janin manusia seutuhnya.

Secara singkat, prosedur IVF adalah menggabungkan sel telur dan sperma di luar tubuh. Kemudian, sel telur yang sudah dibuahi dan sudah dalam fase siap akan dipindahkan ke dalam rahim wanita. Sebelum dipindahkan, embrio diteliti terlebih dahulu untuk menentukan kualitasnya dan akan dijadikan calon bayi.

2.1.1. Tahapan perkembangan embrio

Ada beberapa tahapan perkembangan embrio yang dilalui pada saat melakukan IVF, yaitu:

- Hari ke-0 (Tahap pengumpulan sel telur dan sperma)**

Tahap ini merupakan tahap awal dilakukannya fertilisasi, dengan mengumpulkan sel telur dan sperma dari calon orang tua.

- Hari ke-1 (Pemeriksaan fertilisasi)**

Proses pembuahan akan terjadi sekitar 18 jam. Pemeriksaan fertilisasi dilakukan dengan meneliti sel telur manakah yang membuahi secara normal dan tidak.

- Hari ke-2 (Perkembangan)**

Sel telur yang telah dibuahi akan membentuk embrio dengan dua atau empat sel embrio. Dalam tahap ini, sel telur hanya dilihat perkembangannya, tanpa mengganggu perkembangannya.

- **Hari ke-3 (Penilaian embrio)**

Sel telur yang telah dibuahi akan memiliki enam atau delapan sel embrio. Lalu, embrio akan dinilai berdasarkan kualitasnya.

- **Hari ke-4 (Istirahat)**

Pada tahap ini, tidak adanya pemeriksaan terhadap embrio, melainkan embrio dibiarkan tetap di dalam inkubator untuk menjalani transisi, yang dapat disebut dengan pemandatan dan blastulasi (pembentukan blastula) dini.

- **Hari ke-5 (Penilaian blastokista)**

Pada tahap ini, embrio seharusnya telah berkembang menjadi blastokista. Blastokista adalah tahap perkembangan selanjutnya dari embrio. Apabila embrio membentuk blastokista berkualitas baik, maka potensi untuk penanaman juga akan baik.

- **Tahap pemindahan embrio**

Setelah melakukan pemilihan embrio yang baik, maka embrio sudah siap untuk ditanamkan pada rahim calon ibu.

- **Tahap vitrifikasi blastokista**

Apabila terdapat blastokista berkualitas baik tersisa selelah melakukan pembindahan embrio, maka dapat dilakukan vitrifikasi. Vitrifikasi merupakan proses untuk menjaga kondisi organ atau sel hidup dengan mendinginkan secara cepat agar struktur sel tetap terjaga.

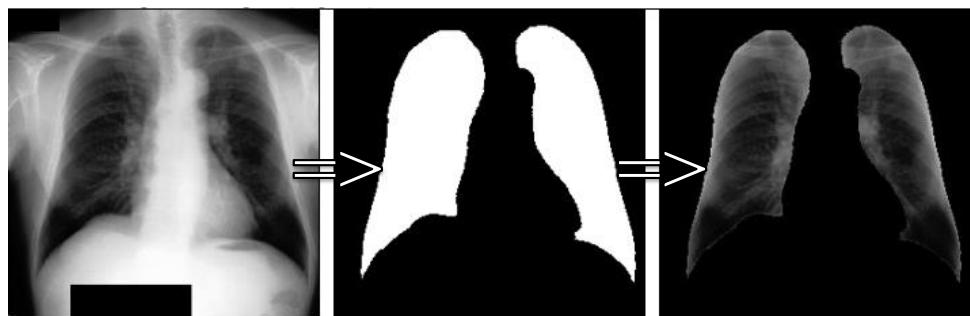
2.2. Data *Pre-processing*

Data *pre-processing* merupakan proses yang perlu dilakukan untuk menghilangkan atau setidaknya meringankan beberapa masalah yang terkait dengan data. Namun, setiap dataset memiliki *pre-processing* yang berbeda, dan tidak ada metode *pre-processing* yang terbaik di semua dataset. [4]

Metode *pre-processing* yang terbaik merupakan proses yang telah dilakukan uji coba dan telah mengalami perbandingan antara *pre-processing* yang satu dengan *pre-processing* yang lainnya.

2.3. Segmentasi

Segmentasi adalah proses untuk mendapatkan objek yang terkandung dalam citra atau membagi citra ke dalam beberapa daerah ke dalam setiap objek berdasarkan beberapa kriteria yang dapat membuat citra menjadi bermakna dan terpisah. [5] Masing-masing piksel pada suatu wilayah (*region*) memiliki kesamaan karakteristik, seperti tingkat keabuan (*grayscale*), tekstur, intensitas, atau warna. Tujuan dari segmentasi adalah untuk menyederhanakan dan/atau mengubah representasi suatu gambar menjadi sesuatu yang lebih bermakna dan lebih mudah untuk dianalisis.

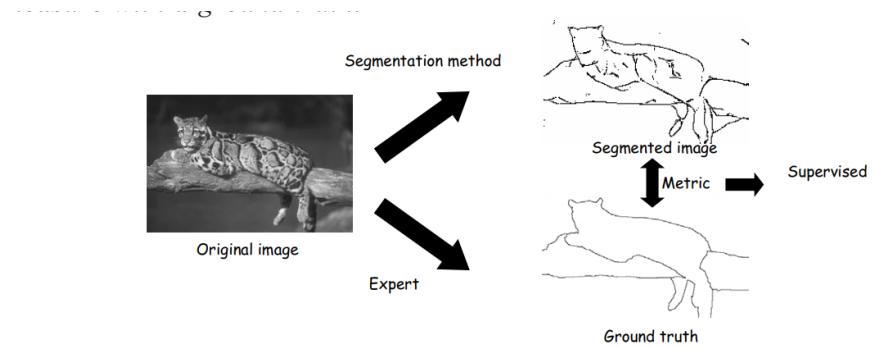


Gambar 1. Konsep proses segmentasi [6]

Dari Gambar 1, konsep proses segmentasi diketahui dengan memasukkan citra asli lalu dilakukan segmentasi menghasilkan citra hitam-putih, lalu citra yang telah tersegmentasi (citra yang berwarna putih) akan dikembalikan ke citra aslinya.

2.3.1. Metode Segmentasi

Segmentasi terbagi menjadi dua metode, yaitu *supervised segmentation* dan *unsupervised segmentation*. *Supervised segmentation* adalah segmentasi yang menggunakan *ground truth* sebagai acuan untuk melakukan segmentasi, sedangkan *unsupervised segmentation* adalah tidak memerlukan *ground truth* sebagai acuan untuk melakukan segmentasi. [7]



Gambar 2. Supervised Segmentation [7]

Dari Gambar 2, citra kucing tersebut diperlukan label atau *ground truth* sebagai acuan model ketika melakukan pelatihan.

2.3.2. Kategori Segmentasi

Terdapat beberapa kategori dalam segmentasi citra, yaitu segmentasi berdasarkan *threshold*, segmentasi berdasarkan garis tepi (*edge*), segmentasi berdasarkan wilayah (*region*), segmentasi berbasis *neural network*, dan sebagainya. [8]

Metode segmentasi berdasarkan *threshold* adalah metode yang paling sederhana untuk segmentasi citra, yaitu dengan membagi piksel citra sesuai dengan tingkat intensitasnya menggunakan histogram dan teknik pengirisan untuk segmentasi gambarnya. Metode ini memiliki tiga jenis, yaitu *global thresholding*, *variable thresholding*, dan *multiple thresholding*.

Metode segmentasi berdasarkan garis tepi (*edge*) adalah teknik pemrosesan citra yang didasarkan pada perubahan nilai intensitas suatu citra. Metodi ini mengasumsikan tepian yang terdeteksi dalam suatu citra sebagai batasan dari objek tersebut dan digunakan untuk mengidentifikasi objek tersebut. Salah satu jenis metode segmentasi berdasarkan garis tepi, yaitu *sobel operator*, *canny operator*, *robert's operator*, dan sebagainya.

Metode segmentasi berdasarkan wilayah (*region*) adalah metode yang menyegmentasikan citra ke dalam berbagai wilayah

yang memiliki karakteristik serupa atau keseragaman antara wilayah-wilayah objek yang satu dengan wilayah-wilayah objek yang lainnya. Metode ini memiliki dua jenis teknik dasar, yaitu *region growing*, dan *region splitting – merging*.

Metode segmentasi berbasis *neural network* adalah metode yang sekarang banyak digunakan untuk segmentasi citra medis. *Neural network* terbuat dari sejumlah *node* yang terhubung dan memiliki *weight* / bobot tertentu. Metode ini memiliki dua langkah dasar, yaitu mengekstraksi fitur dan segmentasi dengan *neural network*.

2.3.3. Hasil Segmentasi

Segmentasi citra memiliki beberapa citra yang dapat dihasilkan, yaitu *semantic segmentation* dan *instance segmentation*. [9]

- **Semantic Segmentation**

Semantic segmentation adalah suatu proses untuk menetapkan setiap piksel ke salah satu dari beberapa label kelas semantik. *Semantic segmentation* termasuk ke dalam *supervised segmentation* karena algoritma ini memerlukan informasi berupa sebuah kumpulan kelas yang sudah diberikan label. *Semantic segmentation* dapat digunakan pada bidang biomedik untuk melakukan segmentasi citra mikroskopis.

- **Instance Segmentation**

Instance segmentation merupakan gabungan antara *semantic segmentation* dan *object detection*. Dimana *semantic segmentation* berguna untuk memberikan informasi setiap piksel ke salah satu dari beberapa label, dan *object detection* berguna untuk memberikan label unik ke masing-masing objek yang telah dilakukan *semantic segmentation*, sehingga *instance segmentation* memperlakukan beberapa objek dari kelas yang sama dengan entitas yang berbeda.



Gambar 3. Object Detection [9]



Gambar 4. Semantic Segmentation (kiri) dan Instance Segmentation (kanan) [9]

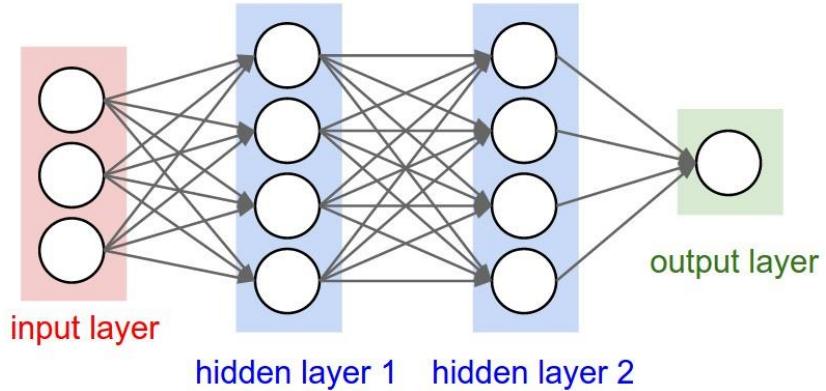
Dari Gambar 3, diketahui hasil dari *object detection*, sedangkan dari Gambar 4, diketahui macam-macam hasil dari segmentasi, dimana yang sebelah kiri merupakan *semantic segmentation*, dimana masing-masing kelas didefinisikan sebagai satu kelas, sedangkan sebelah kanan merupakan *instance segmentation*, dimana masing-masing objek merupakan kelas itu sendiri.

2.4. Neural Networks

Neural Networks (NN) adalah sekumpulan *layer* yang saling berhubungan yang membuat serangkaian transformasi pada data untuk menghasilkan fitur tertentu. NN memiliki mesin yang meniru cara kerja saraf otak. Hasil dari setiap layer tersebut berupa informasi dari data yang dipelajari, yang biasa disebut dengan bobot (*weight*). NN terdiri dari 3 jenis layer, yaitu *Input Layer*, *Hidden Layer*, dan *Output Layer*. [10] [11]

Input Layer merupakan layer untuk mengirimkan input data (gambar, teks, atau lainnya), *Hidden Layer* adalah layer di antara layer input dan output. Layer ini merupakan layer untuk mempelajari pemetaan yang

diberikan, dan *Output Layer* adalah layer untuk memberikan umpan balik dari NN yang telah dibangun.



Gambar 5. Arsitektur Neural Networks [10]

Gambar 5 menunjukkan arsitektur dasar dari *neural network*, yang terdiri dari *input layer*, *hidden layer*, dan *output layer*.

2.4.1. Weight dan Bias

Weight merupakan nilai yang terdapat pada koneksi antar neuron yang merupakan nilai yang harus dipelajari NN untuk digeneralisasikan ke suatu masalah, sedangkan *Bias* adalah nilai yang menunjukkan apa yang harus ditambahkan setelah mengalikan *weight* dengan data. [10]

2.4.2. Fungsi Aktivasi

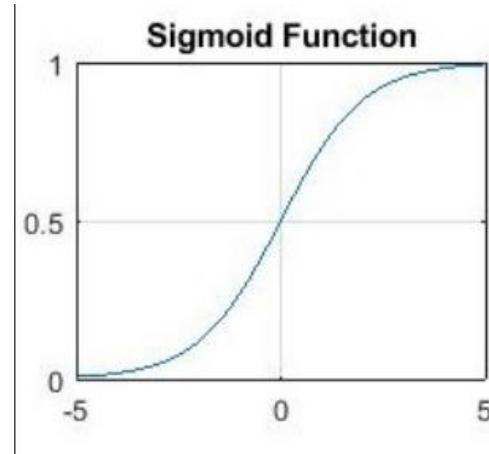
Fungsi aktivasi juga dikenal sebagai fungsi pemetaan. Fungsi ini diterapkan setelah selesai melakukan operasi konvolusi dan sebelum memasuki *pooling layer*. [12] Fungsi ini mengambil beberapa input pada sumbu x dan mengeluarkan nilai dalam rentang terbatas. Ada beberapa fungsi aktivasi, yaitu :

- **Sigmoid**

Fungsi aktivasi sigmoid menghasilkan nilai di antara (0 hingga 1). Oleh karena itu, fungsi ini digunakan apabila mengklasifikasikan *binary*, dan memberikan probabilitas sebagai output.

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

Formula Matematik 1. Fungsi Sigmoid



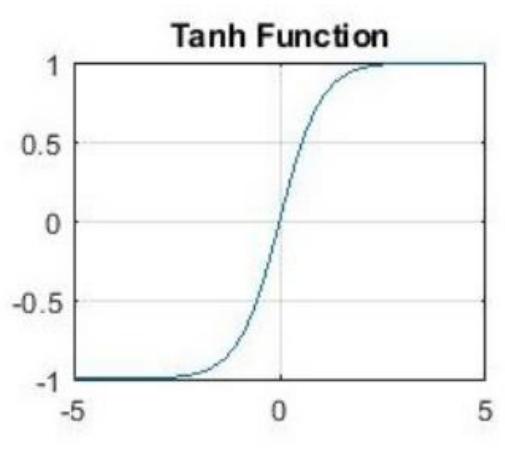
Gambar 6. Fungsi Sigmoid [12]

- **Tanh (Hyperbolic Tangent)**

Fungsi aktivasi tanh memiliki performa yang lebih baik dibandingkan dengan fungsi aktivasi sigmoid, dan menghasilkan nilai di antara (-1 hingga 1) dengan nilai pusat pada 0 (nol).

$$\tanh x = \frac{\sinh x}{\cosh x}$$

Formula Matematik 2. Fungsi TanH



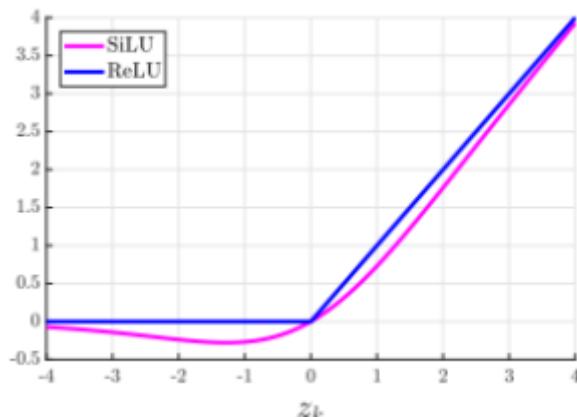
Gambar 7. Fungsi Tanh [12]

- **ReLU**

Fungsi aktivasi ReLU merupakan fungsi aktivasi yang paling banyak digunakan. Fungsi ini menghasilkan nilai di antara (0 hingga tak terbatas)

$$f(x) = \max(0, x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0 \end{cases}$$

Formula Matematik 3. Fungsi ReLU



Gambar 8. Fungsi ReLu [12]

- **SoftMax**

Fungsi aktivasi SoftMax merupakan fungsi yang hampir sama dengan sigmoid, karena memiliki nilai di antara (0 hingga 1). Fungsi aktivasi ini biasanya digunakan untuk klasifikasi multikelas dengan memberikan output berupa probabilitas terbesar.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

for $i = 1, \dots, K$ and $\mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$

Formula Matematik 4. Fungsi SoftMax

2.4.3. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) merupakan salah satu metode NN yang digunakan untuk mengekstraksi gambar tingkat

tinggi ke dalam atribut dimensi (panjang, lebar, dan tinggi). Dalam pekerjaannya, CNN mengambil piksel pada gambar sebagai input kemudian mempelajari cara mengekstrak fitur tersebut, dan hasil akhirnya CNN dapat membuat keputusan dari gambar tersebut.

Terdapat 3 jenis layer utama yang tersusun pada arsitektur CNN, yaitu Convolution Layer, Pooling Layer, dan Fully-connected Layer. [10]

- **Convolution Layer**

Convolutional Layer merupakan layer yang bertujuan mempelajari fitur citra. Layer ini mengoperasikan konvolusi dengan melakukan ekstraksi fitur pada citra input dengan menggunakan filter tertentu. Filter tersebut digeser ke seluruh bagian gambar dengan menerapkan operasi *dot product* pada *input*. Pergeseran filter tersebut ditentukan dari besar *Stride*. Hasil dari proses konvolusi adalah *feature map*.

- **Pooling Layer**

Pooling Layer merupakan proses pengurangan ukuran matriks. Hal ini dilakukan untuk mempercepat proses komputasi. Untuk mendapatkan outputnya, pooling layer memiliki dua metode untuk pengambilan nilai, yaitu:

1. *Max-Pooling* adalah metode pengambilan nilai terbesar pada piksel dari sekelompok piksel.
2. *Average Pooling* adalah metode pengambilan nilai rata-rata pada piksel dari sekelompok piksel

- **Fully-connected Layer**

Fully-Connected Layer (FC) merupakan layer yang terhubung penuh dan memiliki koneksi penuh ke semua aktivasi pada layer sebelumnya. Layer ini digunakan untuk mempelajari kombinasi non-linear dari fitur tingkat tinggi.

Terkadang, *output* dari *convolutional* mengalami pengurangan ukuran. Di lain waktu, *output* dari *convolutional*

tidak ingin mengalami pengurangan ukuran. Oleh karena itu, padding dapat berguna untuk mengatasi masalah tersebut.

- **Padding dan Strides**

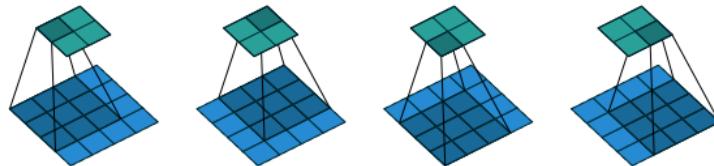
Strides merupakan langkah bergeser yang diambil ketika melakukan *filter* di tiap *convolutional layer*. Strides memiliki kekurangan yaitu dapat menghilangkan informasi yang mungkin ada di bagian batas citra (border). Sedangkan padding merupakan penambahan lapisan pada citra *input* agar meminimalisir hilangnya informasi apabila nilai *strides* lebih dari 1. [13] *Padding* dan *strides* merupakan proses untuk menentukan ukuran *output* dengan menetapkan lapisan dari nilai *input*. Ada 4 jenis *padding* dan *strides*, yaitu : *No zero padding, unit strides; Zero padding, unit strides; Half (same) padding; Full Padding*. [14]

1. *No zero padding, unit strides*

Dengan nilai bebas dari i dan k , serta $s=1$ dan $p=0$

$$o = (i-k) + 1$$

Formula Matematik 5. Fungsi No zero padding, unit strides



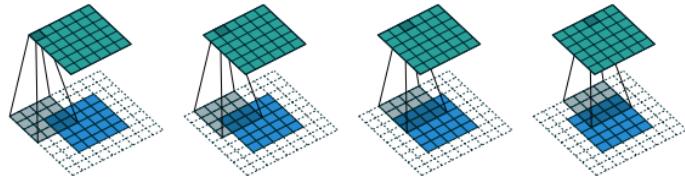
Gambar 9. Contoh proses no zero padding, unit strides [14]

2. *Zero padding, unit strides*

Dengan nilai bebas dari i , k , dan p , serta $s=1$

$$o = (i-k) + 2p + 1$$

Formula Matematik 6. Fungsi Zero padding, unit strides



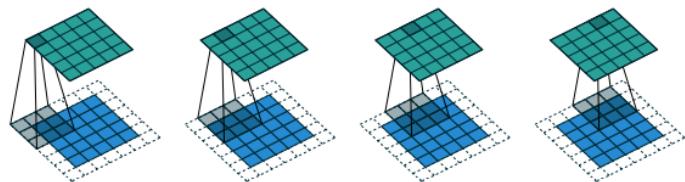
Gambar 10. Contoh proses zero padding, unit strides [14]

3. Half(same) padding

Dengan nilai bebas dari i dan nilai k ganjil ($k = 2n + 1$, $n \in N$), serta $s=1$ dan $p = \lceil k/2 \rceil = n$

$$\begin{aligned} o &= i + 2\lceil k/2 \rceil - (k-1) \\ &= i + 2n - 2n \\ &= i \end{aligned}$$

Formula Matematik 7. Fungsi Half(same) padding



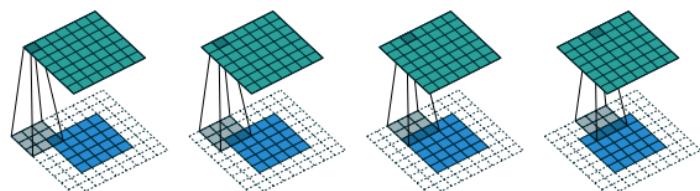
Gambar 11. Contoh proses half(same) padding [14]

4. Full padding

Dengan nilai bebas dari i dan k , serta $s = 1$ dan $p = k-1$

$$\begin{aligned} o &= i + 2(k-1) - (k-1) \\ &= I + (k-1) \end{aligned}$$

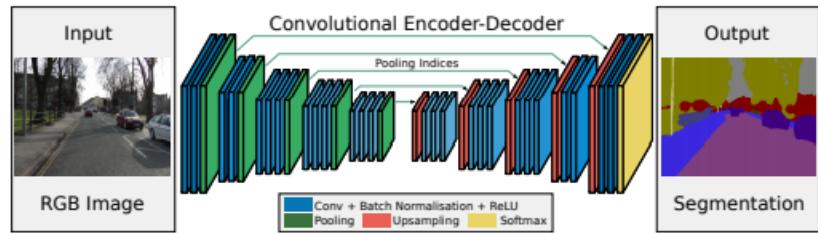
Formula Matematik 8. Fungsi Full Padding



Gambar 12. Contoh proses full padding [14]

2.4.4. SegNet

SegNet merupakan arsitektur untuk melakukan segmentasi semantic berbasis piksel. SegNet merupakan arsitektur yang diperuntukan untuk memodelkan jalan, bangunan, mobil, pejalan kaki, dan memahami hubungan spasial antara kelas yang berbeda seperti jalan utama dan jalan pejalan kaki. Arsitektur SegNet secara topologis identik dengan *convolutional layer* pada VGG16. [15] Berdasarkan [16], nilai performa *F1 Score* SegNet dengan jenis data berupa tangkapan layar jalanan yang terlihat dari sisi atas untuk melakukan segmentasi jalan, gedung, pohon, dan mobil senilai 87.4%.

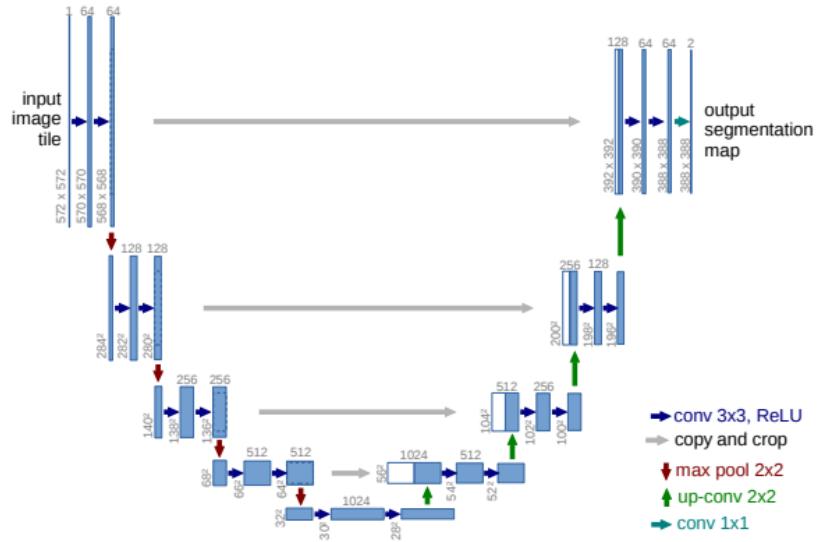


Gambar 13. Arsitektur SegNet [15]

Dari Gambar 13, diketahui bahwa SegNet memiliki jaringan *encoder* dan jaringan *decoder* yang diakhiri dengan *output layer* dengan menggunakan *softmax*.

2.4.5. U-Net

U-Net merupakan bagian dari *Convolutional Neural Network*.



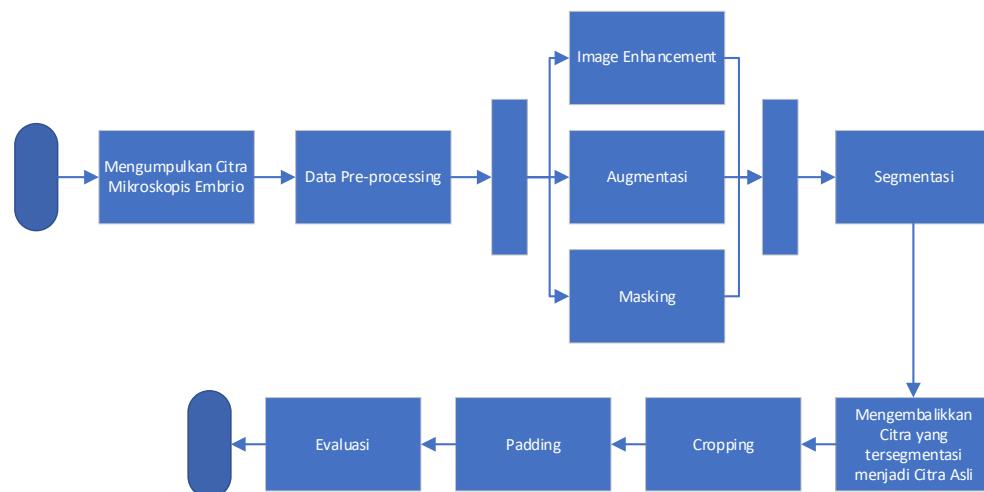
Gambar 14. Arsitektur U-Net [17]

Dari Gambar 14, diketahui U-Net memiliki struktur *encoder-decoder* [17], dengan *skip connection* antara *layer encoding* dan *decoding*, dimana *skip connection* adalah *layer encoding* akan digabungkan (*concatenate*) dengan *layer decoding* yang dapat memungkinkan jaringan untuk mempertahankan fitur tingkat rendah untuk prediksi akhir. Berdasarkan [16], nilai performa *F1 Score* U-Net dengan jenis data berupa tangkapan layar jalanan yang terlihat dari sisi atas untuk melakukan segmentasi jalan, gedung, pohon, dan mobil senilai 83%, serta berdasarkan [17], nilai performa IOU (*intersection over union*) dengan jenis data sel biomedis (ISBI cell challenge 2015) senilai 92%.

BAB III

METODE PENELITIAN

Penelitian ini dibagi ke dalam empat tahapan, yaitu pengumpulan dataset, data *pre-processing*, dan evaluasi. Pengumpulan dataset menjelaskan tentang deskripsi dari data. Data *pre-processing* merupakan tahap perbaikan citra agar lebih mudah dilakukan ekstraksi fitur. Membangun model merupakan tahapan modifikasi dari model yang ada untuk melakukan penelitian. Evaluasi merupakan tahap terakhir pada penelitian ini, yang bertujuan untuk melihat kinerja dari model yang dihasilkan.



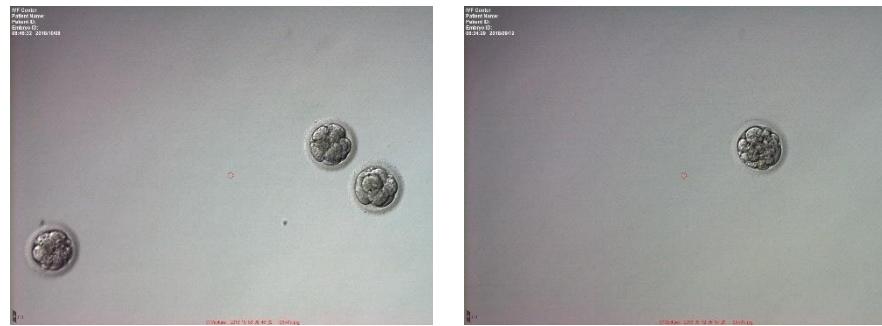
Gambar 15. Workflow Penelitian

Dari Gambar 15, merupakan alur kerja dari penelitian, dimana diawali dengan pengumpulan data, data pre-processing (*Image Enhancement*, Augmentasi, dan Masking), segmentasi, data post-processing (mengembalikkan citra yang tersegmentasi menjadi citra asli, Cropping, Padding), dan diakhiri evaluasi dari model.

3.1. Deskripsi Sumber Data

Data yang digunakan pada penelitian ini berupa citra mikroskopis embrio dari hasil laboratorium *Indonesian Medical Education and Research Institute* (IMERI, FKUI). Dataset yang diberikan terdiri dari 327 Citra mikroskopis yang masing-masing citra berukuran 1360 x 1024 piksel dengan

kanal RGB dan format ekstensi jpg, dan jumlah embrio sebanyak 624 embrio. [2]



Gambar 16. Sampel data

Dalam satu citra terdapat minimal 1 embrio dan maksimal 3 embrio di dalamnya (Gambar 16) dengan memungkinkan perbedaan *grade*. Terdapat 4 jenis *grade* pada data, yaitu *Grade 1*, *Grade 2*, *Grade 3*, dan *Grade X*. Namun, pada penelitian ini, hanya akan menggunakan 3 *grade*, dengan tanpa menggunakan *grade X*. Data citra pada penelitian ini, akan dibagi menjadi data latih dan data uji menjadi 60% data latih, 30% data Validasi, dan 100% data uji. Data uji digunakan 100% karena akan melakukan segmentasi keseluruhan citra yang ada.

3.2. Library / Modul yang digunakan selama penelitian

3.2.1. Pandas

Pada penelitian ini, Pandas digunakan untuk membuat dan memuat *dataframe* guna menyimpan dataset yang digunakan.

3.2.2. Matplotlib

Pada penelitian ini, Matplotlib digunakan untuk menampilkan citra dan grafik dari penelitian.

3.2.3. Open CV2

Pada penelitian ini, Open CV2 digunakan untuk menyelesaikan beberapa masalah, yaitu :

- `cv2.imread` digunakan untuk memuat citra yang digunakan menjadi data selama penelitian.

- `cv2.imwrite` digunakan untuk menyimpan citra yang telah tersegmentasi pada penelitian.
- `cv2.resize` digunakan untuk mengubah skala ukuran dari citra yang ada.
- `cv2.cvtColor` digunakan untuk mengubah kanal pada citra yang digunakan (RGB menjadi *grayscale* / sebaliknya).
- `cv2.findContours` digunakan untuk mencari kontur pada citra yang digunakan.
- `cv2.boundingRect` digunakan untuk membuat sebuah segi empat sebagai batasan untuk melakukan *cropping* citra.

3.2.4. Numpy

Pada penelitian ini, Numpy digunakan untuk melakukan manipulasi citra yang digunakan, seperti mengubah nilai pada data, menambahkan *padding* pada data, menghapus nilai pada data, dan mengubah data menjadi array

3.2.5. Pillow

Pada penelitian ini, Pillow digunakan untuk melakukan *Image Enhancement* dan melakukan rotasi dari citra yang digunakan.

3.2.6. Tensorflow Keras

- `Conv2D` digunakan untuk membuat *layer* konvolusi dari model yang dibangun.
- `MaxPooling2D` digunakan untuk membuat *layer pooling* dari model yang dibangun.
- `Dropout` digunakan untuk menetapkan *input unit* ke 0 dengan frekuensi di setiap *step* selama waktu pelatihan (*train*) yang membantu mencegah overfitting.
- `Concatenate` digunakan untuk menggabungkan array atau nilai dari data.

- `Conv2DTranspose` digunakan untuk membuat *layer* konvolusi ditambah dengan *transpose* dari model yang dibangun.
- `Optimizer` digunakan untuk mengoptimisasi model yang digunakan.
- `EarlyStopping` untuk memberhentikan model ketika sedang melakukan pelatihan (*train*) yang disebabkan tidak adanya perubahan untuk memperbaiki model yang dibangun.

3.3. Data *Pre-processing*

Untuk mendapatkan model segmentasi dengan performa yang baik, maka diperlukannya data *pre-processing*. Dari data yang telah dikumpulkan, dilakukan beberapa tahapan pemrosesan sebelum membangun model, yaitu :

3.2.1. Image Enhancement

Image Enhancement adalah salah satu tahapan *pre-processing* untuk meningkatkan fitur gambar atau menghilangkan fitur yang tidak diinginkan pada gambar, sehingga model dapat mengambil informasi mengenai gambar dengan baik.

Teknik yang dapat digunakan untuk melakukan *Image Enhancement*, yaitu dengan peningkatan kontras dan peningkatan kecerahan (*brightness*) citra.

3.2.2. Augmentasi Data

Augmentasi Data diperlukan untuk memperbesar dataset yang dapat mengurangi *overfitting* pada data, dan meningkatkan kinerja algoritma. Ada beberapa tipe transformasi untuk Augmentasi Data, yaitu Rotasi ($0^\circ - 360^\circ$), *Flipping* (0 = Tidak, 1 = *Flip*), dan *Rescalling* (mengubah skala gambar).

Pada penelitian ini, augmentasi data yang digunakan adalah *Flipping* / membalikkan secara dicerminkan antara kiri dan kanan.

3.2.3. Masking

Masking merupakan proses untuk pengubahan / memanipulasi citra dengan menyembunyikan beberapa bagian gambar dan menimbulkan beberapa bagian gambar.

3.4. Segmentasi menggunakan U-Net

Proses segmentasi dilakukan dengan menggunakan U-Net. U-Net biasa digunakan untuk melakukan segmentasi citra biomedik. U-Net merupakan metode segmentasi yang tergolong *supervised segmentation*, dikarenakan u-net membutuhkan data label sebagai acuan untuk melatih model.

3.5. Data Post-processing

Post-processing merupakan proses mengedit data untuk menyempurnakan citra yang telah dihasilkan dari model yang dibangun. *Post-processing* dapat membantu meningkatkan kualitas citra guna memberikan informasi yang lebih jelas. *Post-processing* dilakukan apabila citra yang telah dihasilkan dari model yang dibangun masih belum memiliki informasi secara jelas.

Pada penelitian ini, *post-processing* yang dilakukan, yaitu :

3.5.1. Mengubah citra yang tersegmentasi menjadi citra asli

Pada tahap ini, citra yang telah tersegmentasi akan diubah menjadi citra asli, dimana citra yang tersegmentasi hanya berwarna putih, dan akan diubah menjadi citra asli yaitu berupa embrio yang sesuai dengan titik koordinatnya.

3.5.2. Cropping

Pada tahap ini, citra yang telah tersegmentasi akan dipotong berdasarkan kontur dari masing-masing embrio yang dapat diidentifikasi menggunakan `cv2.findContours` dengan batasan maksimum dari kontur yang dihasilkan dari `cv2.boundingRect`.

3.5.3. Padding

Pada tahap ini, citra yang telah terpotong akan memiliki ukuran piksel yang tidak seragam antara embrio yang satu dengan embrio yang lainnya. Oleh karena itu, ditambahkan *padding* guna menyamaratakan ukuran piksel.

3.6. Evaluasi

Model yang telah dibangun pada penelitian ini, akan dievaluasi dengan menilai *success rate* dan *error rate* antara total embrio yang telah tersegmentasi oleh model yang dibangun dan total embrio keseluruhan.

Nilai *success rate* diperoleh dengan rumus sebagai berikut :

$$\text{Success rate} = \frac{\text{total embrio tersegmentasi}}{\text{total embrio keseluruhan}} \times 100\%$$

Formula Matematik 9. Success rate

sedangkan, nilai *error rate* diperoleh dengan rumus sebagai berikut :

$$\text{Error rate} = \frac{\text{total embrio keseluruhan} - \text{total embrio tersegmentasi}}{\text{total embrio keseluruhan}} \times 100\%$$

atau

$$\text{Error rate} = 100\% - \text{success rate}$$

Formula Matematik 10. Error rate

BAB IV

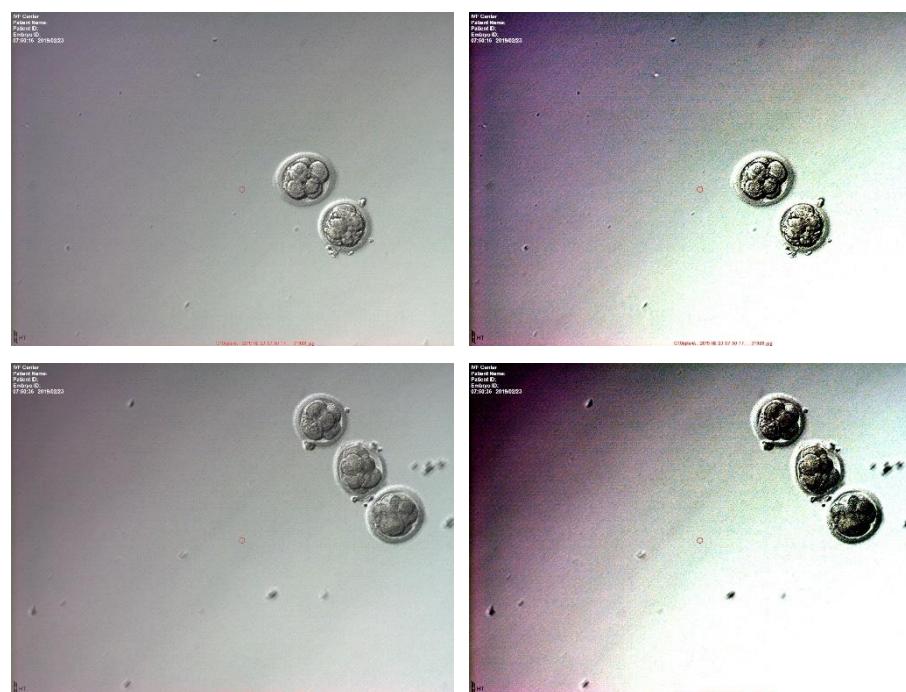
HASIL DAN PEMBAHASAN

4.1. Pre-processing

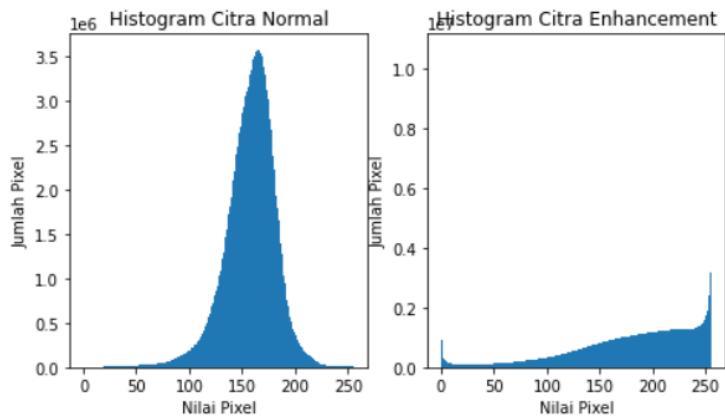
Pada tahap ini, dilakukan untuk memperbaiki citra agar informasi yang akan diolah menjadi mudah untuk diterima oleh model segmentasi yang dibangun. Ada beberapa tahapan *pre-processing* yang dilakukan, yaitu :

4.1.1. Image Enhancement

Pada tahap ini, penulis melakukan beberapa percobaan *image enhancement* dengan mengubah nilai *input* untuk mengubah tingkatan kontras, dan brightness dari citra yang ada.



Gambar 17. Contoh Citra yang telah dilakukan *Image Enhancement*

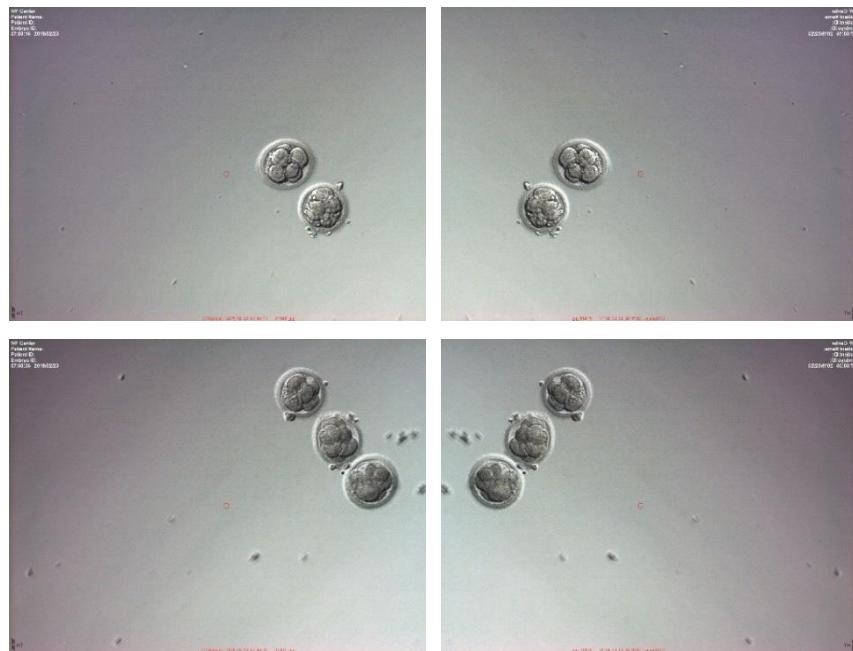


Gambar 18. Histogram Citra Normal – Citra Enhancement

Dari Gambar 17, terlihat perbedaan dari citra asli dan citra yang telah di-*enhance*; dan dari Gambar 18, terlihat dari histogram, citra yang telah di-*enhance* cenderung lebih terang dibandingkan citra normal. Hal tersebut dikarenakan citra telah dilakukan peningkatan nilai *contrast* sejumlah 2.5 kali lipat dan *brightness* sejumlah 1.2 kali lipat dari citra normal.

4.1.2. Augmentasi

Pada tahap ini, penulis melakukan augmentasi dengan mengubah citra asli menjadi terlihat seperti citra yang dicerminkan.

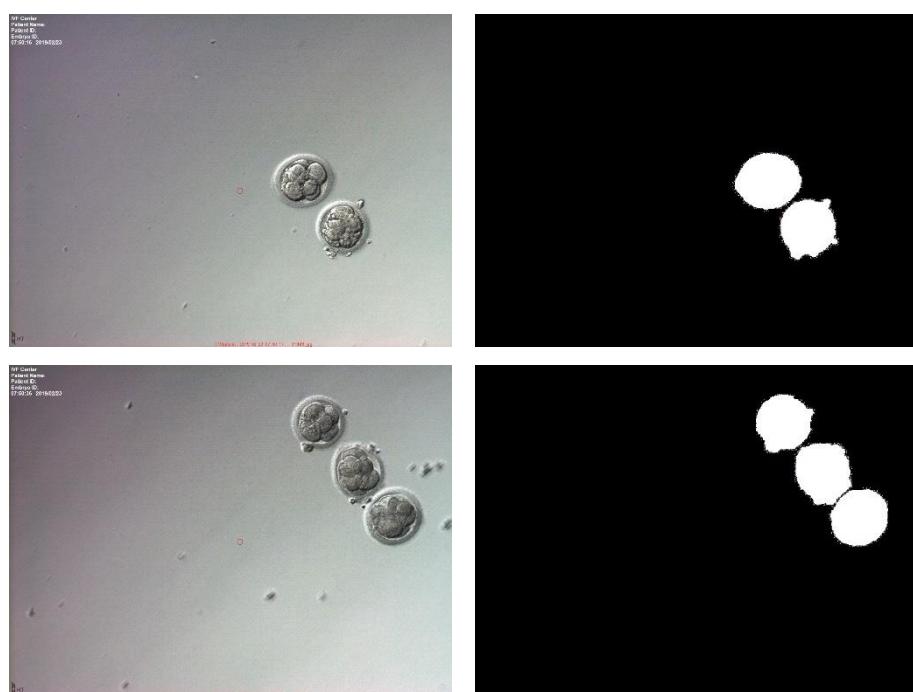


Gambar 19. Contoh citra yang teraugmentasi

Dari Gambar 19, dilakukan rotasi dari citra asli secara dicerminkan menggunakan *library* dari *pillow*, yaitu *ImageEnhance*.

4.1.3. Masking

Pada tahap ini, penulis melakukan masking dari citra yang telah dilakukan *image enhancement*, *augmentasi*, maupun data aslinya. *Masking* ini digunakan untuk label atau acuan citra pada saat melakukan pelatihan model.



Gambar 20. Contoh citra yang telah di-masking

Dari Gambar 20, citra dilakukan *masking* secara manual dengan menggunakan *Adobe Photoshop* dengan menyeleksi embrio dari masing-masing citra, dan diubah menjadi hitam-putih.

4.2. U-Net

Pada penelitian ini, *hyperparameter* dari model yang dibangun meliputi Batch Size (4,8,16), Epoch sejumlah 100, *Loss Function* menggunakan *categorical cross entropy*, *Optimizer* menggunakan *Adam*, dan *learning rate* senilai 0.0001.

Categorical cross entropy digunakan sebagai *loss function*, karena segmentasi semantik merupakan bagian dari *multi-class classification*. Oleh karena itu, *categorical cross entropy* dipilih sebagai *loss function*, sedangkan *binary cross entropy* digunakan untuk *classification two classes*.

Dari model yang dibangun juga menggunakan *Callbacks (Early Stopping)* guna memberhentikan model ketika dilatih apabila tidak adanya peningkatan kualitas dari model yang dibangun.

Pada penelitian ini, model yang dibangun terdiri dari beberapa layer, yaitu :

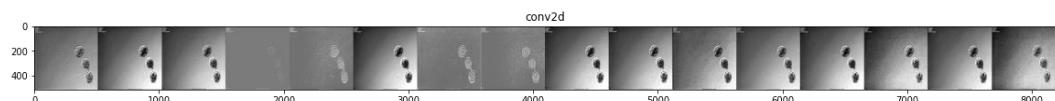
Layer	Input Shape	Total Parameter	Output Shape
Input	512 x 512 x 3	0	512 x 512 x 3
Conv2D	512 x 512 x 3	448	512 x 512 x 16
Conv2D	512 x 512 x 16	2320	512 x 512 x 16
Dropout	512 x 512 x 16	0	512 x 512 x 16
Maxpooling	512 x 512 x 16	0	256 x 256 x 16
Conv2D	256 x 256 x 16	4640	256 x 256 x 32
Conv2D	256 x 256 x 32	9248	256 x 256 x 32
Dropout	256 x 256 x 32	0	256 x 256 x 32
Maxpooling	256 x 256 x 32	0	128 x 128 x 32
Conv2D	128 x 128 x 32	18496	128 x 128 x 64
Conv2D	128 x 128 x 64	36928	128 x 128 x 64
Dropout	128 x 128 x 64	0	128 x 128 x 64
Maxpooling	128 x 128 x 64	0	64 x 64 x 64
Conv2D	64 x 64 x 64	73856	64 x 64 x 128
Conv2D	64 x 64 x 128	147584	64 x 64 x 128
Dropout	64 x 64 x 128	0	64 x 64 x 128
Maxpooling	64 x 64 x 128	0	32 x 32 x 128
Conv2D	32 x 32 x 128	295169	32 x 32 x 256
Conv2D	32 x 32 x 256	590080	32 x 32 x 256
Dropout	32 x 32 x 256	0	32 x 32 x 256

Conv2DTranspose	$32 \times 32 \times 256$	131200	$64 \times 64 \times 128$
Concatenate	$64 \times 64 \times 128$	0	$64 \times 64 \times 256$
Conv2D	$64 \times 64 \times 256$	295040	$64 \times 64 \times 128$
Conv2D	$64 \times 64 \times 128$	147584	$64 \times 64 \times 128$
Conv2DTranspose	$64 \times 64 \times 128$	32832	$128 \times 128 \times 64$
Concatenate	$128 \times 128 \times 64$	0	$128 \times 128 \times 128$
Conv2D	$128 \times 128 \times 128$	73792	$128 \times 128 \times 64$
Conv2D	$128 \times 128 \times 64$	36928	$128 \times 128 \times 64$
Conv2DTranspose	$128 \times 128 \times 64$	8224	$256 \times 256 \times 32$
Concatenate	$256 \times 256 \times 32$	0	$256 \times 256 \times 64$
Conv2D	$256 \times 256 \times 64$	18464	$256 \times 256 \times 32$
Conv2D	$256 \times 256 \times 32$	9248	$256 \times 256 \times 32$
Conv2DTranspose	$256 \times 256 \times 32$	2064	$512 \times 512 \times 16$
Concatenate	$512 \times 512 \times 16$	0	$512 \times 512 \times 32$
Conv2D	$512 \times 512 \times 32$	4624	$512 \times 512 \times 16$
Conv2D	$512 \times 512 \times 16$	2320	$512 \times 512 \times 16$
Output	$512 \times 512 \times 16$	17	$512 \times 512 \times 1$

Total Parameter = 1,941,105

Berikut beberapa sampel *output* dari masing-masing layer

Input Layer

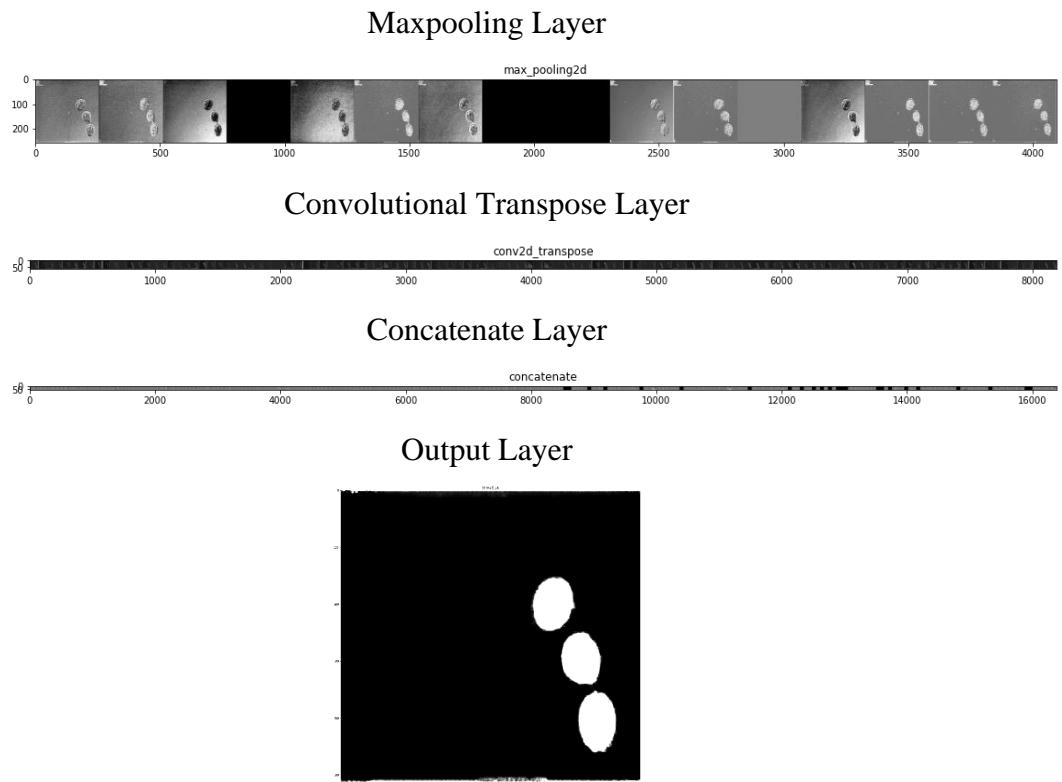


Convolutional Layer



Dropout Layer





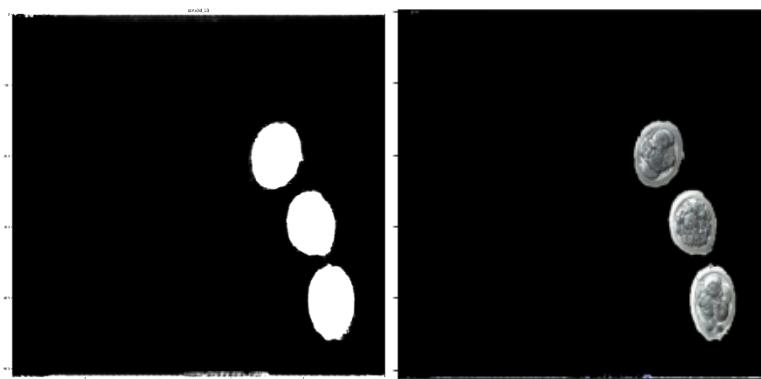
Gambar 21. Contoh output yang dihasilkan tiap layer U-Net

Dari Gambar 21, masing-masing *layer* menghasilkan output yang berbeda-beda, hingga pada *output layer*, citra hanya menjadi hitam-putih, dimana yang berwarna putih tersebut merupakan citra yang tersegmentasi.

4.3. Post-processing

4.3.1. Mengubah citra yang tersegmentasi menjadi citra asli

Setelah pelatihan model dilakukan, maka *output* yang dihasilkan dari model tersebut akan menghasilkan citra yang tersegmentasi. Oleh karena itu, citra yang telah tersegmentasi tersebut diubah menjadi citra asli. Berikut hasil citra yang telah diubah menjadi citra asli :



Gambar 22. Contoh citra yang telah diubah menjadi citra asli

Setelah model menghasilkan citra seperti pada Gambar 21, citra yang telah tersegmentasi akan dikembalikan menjadi citra aslinya, yaitu terlihat pada Gambar 22.

4.3.2. Cropping

Setelah citra diubah menjadi citra asli, masing-masing citra akan dipotong sesuai dengan kontur yang teridentifikasi oleh cv2.findContours, dan kemudian akan dibuat sebuah segi empat yang dihasilkan dari cv2.boundingRect. Berikut hasil citra yang telah dipotong :



Gambar 23. Contoh citra yang telah terpotong

Dari Gambar 23 di sisi kiri, diketahui bahwa citra berjumlah 3 buah, dan dilakukan proses pemotongan (*cropping*) menjadi masing-masing citra berisikan 1 buah embrio.

4.3.3. Padding

Setelah itu, citra yang telah dipotong akan memiliki ukuran piksel yang berbeda-beda di tiap embrionya. Oleh karena itu, agar citra yang diperoleh memiliki keseragaman ukuran, ditambahkan padding di masing-masing embrio yang telah dipotong. Berikut hasil citra embrio yang telah ditambahkan padding :



Gambar 24. Contoh citra yang telah ter-padding

Dari Gambar 24, proses *padding* dilakukan dengan cara menambahkan piksel hitam pada citra yang telah dilakukan *cropping* dan setelah melakukan proses *padding*, ukuran keseluruhan citra menjadi seragam, yaitu 256x256 piksel.

4.4. Analisis Hasil Uji

Setelah semua rangkaian tahap-tahap selesai dilakukan, penulis akan membandingkan model yang dibangun dari beberapa eksplorasi parameter yang dilakukan. Berikut tabel pengujian model yang dilakukan :

Tabel 1. Jumlah Embrio yang tersegmentasi berdasarkan Batch Size

Batch Size	Jumlah Embrio yang tersegmentasi	Jumlah Kontur yang teridentifikasi	Total Jumlah Embrio Asli
4	611	617	624
8	619	621	

16	623	623	
----	-----	-----	--

Tabel 2. Sampel Embrio yang telah terpotong berdasarkan Batch Size

Batch Size	Sampel Embrio yang telah terpotong					
4						
8						
16						

Dari jumlah embrio yang tersegmentasi, dapat diketahui *success rate* dan *error rate* dari model yang dibangun, yaitu :

Tabel 3. Success rate dan Error Rate model berdasarkan Batch Size

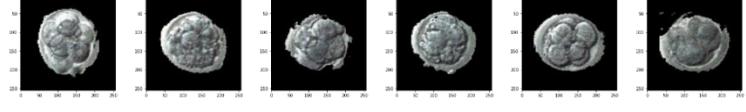
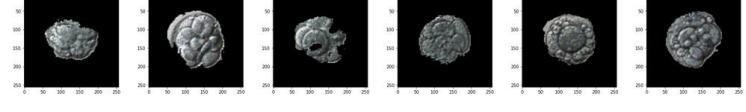
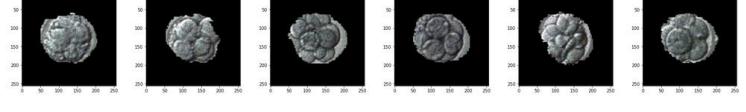
Batch Size	Success Rate
4	$\frac{611}{624} \times 100\% = 97,92\%$
8	$\frac{619}{624} \times 100\% = 99,2\%$
16	$\frac{623}{624} \times 100\% = 99,85\%$

Ada pula beberapa eksplorasi berdasarkan data latih yang digunakan. Berikut tabel pengujian model yang dilakukan :

Tabel 4. Jumlah Embrio yang tersegmentasi berdasarkan Data Latih yang digunakan

Data yang digunakan	Jumlah Embrio yang tersegmentasi	Jumlah Kontur yang teridentifikasi	Total Jumlah Embrio Asli
Normal	623	623	624
Enhance	82	82	
Augmentasi	554	586	
Augmentasi + Enhance	123	123	

Tabel 5. Sampel Embrio yang telah terpotong berdasarkan Data Latih yang digunakan

Data yang digunakan	Sampel Embrio yang telah terpotong					
Normal						
Enhance						
Augmentasi						
Augmentasi + Enhance						

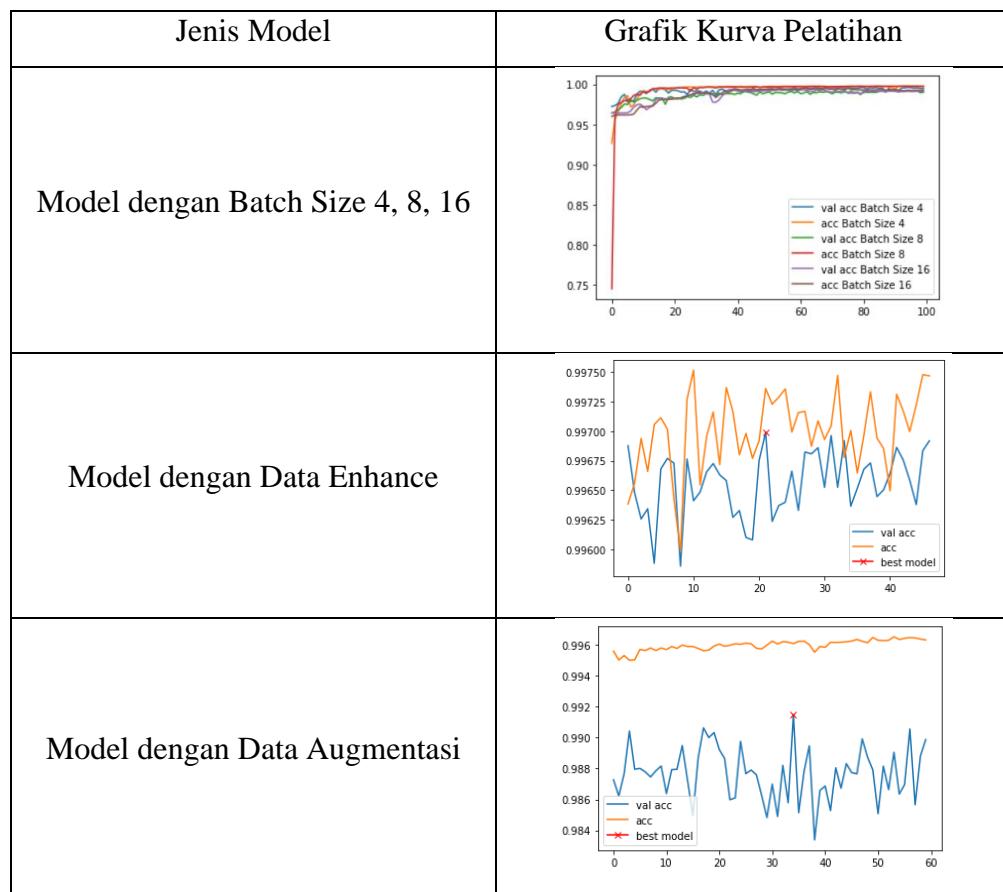
Dari jumlah embrio yang tersegmentasi, dapat diketahui *success rate* dan *error rate* dari model yang dibangun, yaitu :

Tabel 6. Success rate dan Error Rate model berdasarkan Data Latih yang digunakan

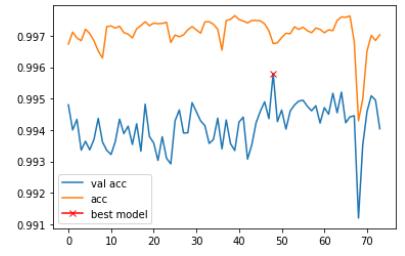
Data yang digunakan	Success Rate
Normal	$\frac{623}{624} \times 100\% = 99,85\%$
Enhance	$\frac{82}{624} \times 100\% = 13,14\%$
Augmentasi	$\frac{554}{624} \times 100\% = 88,78\%$
Augmentasi + Enhance	$\frac{123}{624} \times 100\% = 19,71\%$

Dari beberapa eksplorasi *hyperparameter*, dapat diketahui grafik kurva pelatihan dari masing-masing model, yaitu :

Tabel 7. Grafik Kurva Pelatihan



Model dengan Data Augmentasi + Enhance



Dari tabel 7, diketahui grafik kurva pelatihan dari masing-masing model. Dimana pada model dengan data enhance, data augmentasi + enhance, model mengalami *overfitting* dikarenakan grafik terlihat tidak stabil, dan tidak adanya kenaikan akurasi seiring jumlah *epoch*-nya bertambah.

Berdasarkan hasil evaluasi, masing-masing model diuji untuk melakukan segmentasi dengan menggunakan citra asli (citra normal), dimana model *enhance* memiliki nilai *success rate* 13.14% menandakan grafik pelatihan model pada model *enhance* tidak merepresentasi performa model ketika dilakukan pengujian terhadap data ujinya, karena ketika melakukan pelatihan model, akurasi validasi dari model menunjukkan $\pm 99\%$, serta model tidak dapat melakukan segmentasi citra asli (citra normal) dengan baik, hal tersebut menunjukkan bahwa model tersebut mengalami *overfitting*. Seperti halnya model *enhance*, model augmentasi + *enhance* juga mengalami *overfitting*, ketika melakukan pelatihan menunjukkan akurasi 99%, sedangkan pada saat pengujian 19,71%.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Kesimpulan dari penelitian yang telah dilakukan, yaitu :

1. *Pre-processing (image enhancement, augmentasi)* yang dilakukan pada penelitian ini tidak menghasilkan model yang lebih baik, dikarenakan embrio pada citra yang telah dilakukan *pre-processing* tidak dapat diidentifikasi oleh model.
2. *Pre-processing (Masking)* manual yang dilakukan pada penelitian ini digunakan sebagai label dari data latih yang digunakan.
3. Citra yang digunakan untuk melatih model akan *di-resize* sebagai input model yang dibangun, guna mempercepat komputasi. Pada penelitian ini, dengan RAM 12 GB masih dapat dijalankan, sedangkan citra yang tidak *di-resize* akan membuat *crash* pada program yang digunakan.
4. Semakin tinggi nilai Batch Size yang digunakan, semakin tinggi nilai *success rate* dan semakin kecil nilai *error rate*. Model yang terbaik, terjadi dengan menggunakan Batch Size 16.
5. Dari model yang terbaik, diperoleh citra 623 embrio dari 624 total keseluruhan embrio, dari 327 citra asli yang masing-masing berisikan 1-3 embrio, dan memiliki *success rate* 99.85%.

5.2. Saran

1. Pada saat melakukan *masking*, lebih baik mencoba menggunakan metode-metode *computer vision* yang ada, tidak dengan manual. Metode manual lebih memakan banyak waktu dibandingkan dengan melakukan masking menggunakan metode *computer vision*.
2. Pada saat melakukan *image enhancement*, perlu mengeksplorasi *parameter* yang optimal untuk meningkatkan kualitas dari citra agar tetap terjaga informasinya
3. Model yang dihasilkan dari penelitian ini dapat dilakukan penelitian lebih lanjut ke tahap klasifikasi.

REFERENSI

- [1] M. F. Kragh, J. Rimestad, J. Bemsten and H. Karstoft, "Automatic grading of human blastocysts from time-lapse imaging," *Computers in Biology and Medicine* 115, 2019.
- [2] A. Akbar Septiandri, A. Jamal, P. Ameilia Iffanolida, O. Riayati and B. Wiweko, "Human Blastocyst Classification after In Vitro Fertilization Using Deep Learning," *arXiv:2008.12480v1 [eess.IV]*, Aug 2020.
- [3] P. Khosravi, E. Kazemi, Q. Zhan, J. E. Malmsten, M. Toschi, P. Zisimopoulos, A. Sigaras, S. Lavery, L. A. D. Cooper, C. Hickman, M. Meseguer, Z. Rosenwaks, O. Elemento, N. Zaninovic and I. Hajirasouliha, "Deep learning enables robust assessment and selection of human blastocysts after in vitro fertilization," *npj Digital Medicine*, 2019.
- [4] S. AlMuhaideb and M. El Bachir Menai, "An individualized preprocessing for medical data classification," *Procedia Computer Science* 82, 2016.
- [5] R. Fahmi and S. Sinurat, "PERANCANGAN APLIKASI DETEKSI TEPI PADA CITRA DIGITAL DENGAN METODE OPERATOR SOBEL DAN OPERATOR LAPLACIAN OF GAUSSIAN (LOG)," *Seminar Nasional Inovasi dan Teknologi Informasi (SNITI-3)*, 2016.
- [6] Y. Gordienko, P. Gang, J. Hui, W. Zeng, Y. Kochura, O. O. R. and S. S. , "Deep Learning with Lung Segmentation and Bone Shadow Exclusion Techniques for Chest X-Ray Analysis of Lung Cancer".

- [7] C. Rosenberger, S. Chabrier, H. Laurent and B. Emile, "Unsupervised and Supervised Image Segmentation Evaluation," 2006.
- [8] D. Kaur and Y. Kaur, "Various Image Segmentation Techniques: A Review," *International Journal of Computer Science and Mobile Computing (IJCSMC)*, vol. 3, no. 5, pp. 809-814, 2014.
- [9] A. Arnab, S. Zheng, S. Jayasumana, B. Romera-Paredes, M. Larsson, A. Kirillov, B. Savchynskyy, C. Rother, F. Kahl and P. Torr, "Conditional Random Fields Meet Deep Neural Network for Semantic Segmentation," *IEEE SIGNAL PROCESSING MAGAZINE*, vol. XX, 2018.
- [10] Standford University, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: <https://cs231n.github.io/convolutional-networks/>. [Accessed 2021].
- [11] Y. LeCun, Y. Bengio and G. Hinton, "Deep Learning," *Nature*, vol. 521, 2015.
- [12] C. Enyinna Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," *arXiv:1811.03378v1 [cs.LG]*, 2018.
- [13] S. Albawi and T. Abed Mohammed, "Understanding of a Convolutional Neural Network," *ICET2017*, August 2017.
- [14] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep," *arXiv:1603.07285v2 [stat.ML]*, January 12, 2018.

- [15] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arXiv:1511.00561v3 [cs.CV]*, 2016.
- [16] Q. Liu, A.-B. Salberg and R. Jenssen, "A COMPARISON OF DEEP LEARNING ARCHITECTURES FOR SEMANTIC MAPPING OF VERY HIGH RESOLUTION IMAGES," *IGARSS*, 2018.
- [17] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015.

LAMPIRAN

Lampiran 1. Source Code Pre-processing

Import Library

```
In [ ]:  
import os  
from glob import glob  
import pandas as pd  
from PIL import Image, ImageEnhance  
import matplotlib.pyplot as plt  
import cv2  
import numpy as np  
from numpy import asarray
```

Turn on GPU

```
In [ ]:  
%tensorflow_version 2.x  
import tensorflow as tf  
device_name = tf.test.gpu_device_name()  
if device_name != '/device:GPU:0':  
    raise SystemError('GPU device not found')  
print('Found GPU at: {}'.format(device_name))
```

Found GPU at: /device:GPU:0

Mount to Google Drive

```
In [ ]:  
from google.colab import drive  
drive.mount('/content/drive')
```

Mounted at /content/drive

Load Data

```
In [ ]:  
path = os.path.join("../content/drive/My Drive/Colab Notebooks/Dataset/", "Embrio")  
path_image = {os.path.splitext(os.path.basename(x))[0]: x  
             for x in glob(os.path.join(path, "Embryo", "*"))}  
path_imagemask = {os.path.splitext(os.path.basename(x))[0]: x  
                  for x in glob(os.path.join(path, "EmbryoMask", "*"))}
```

```
In [ ]:  
df = pd.read_csv("../content/drive/My Drive/Colab Notebooks/Dataset/Embrio/Embryo.csv")
```

```
In [ ]:  
df.head()  
Out[ ]:
```

	id	label
0	Amirah_1atas-2-1.jpg	train
1	Amirah_1-2.jpg	train
2	Amirah_1-x-1.jpg	train
3	Anggi_1.jpg	train
4	Anggi_1-2-1.jpg	train

```
In [ ]:  
for i in range(len(df)):  
    df['id'][i] = df['id'][i].replace('.jpg', '')
```

Get Path Image

```
In [ ]:  
df['path_image'] = df['id'].map(path_image.get)  
df['path_imagemask'] = df['id'].map(path_imagemask.get)
```

```
In [ ]:  
raw_id = []  
raw_data = []  
label_data = []
```

```

for i in range(len(df)):
    imgX = Image.open(df['path_image'][i])
    imgY = Image.open(df['path_imagemask'][i])
    raw_id.append(df['id'][i])
    raw_data.append(imgX)
    label_data.append(imgY)

In [ ]:
raw_flip = []
label_flip = []
raw_enhance = []
raw_flip_enhance = []

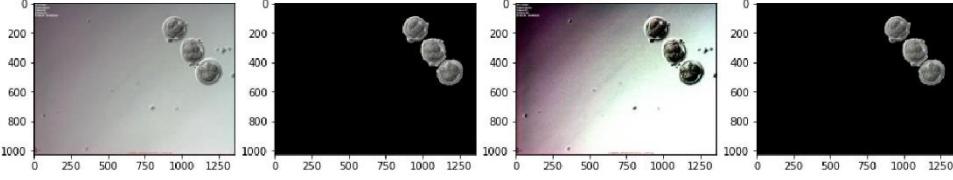
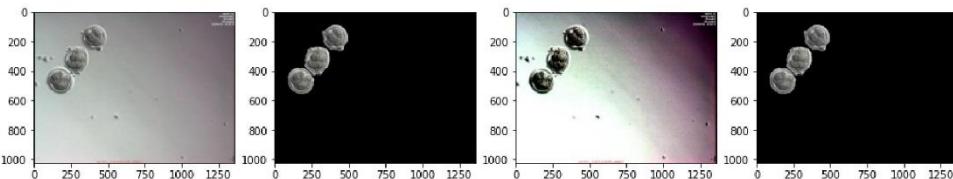
for i in range(len(raw_data)):
    img = raw_data[i].copy()
    imgEnhance = ImageEnhance.Brightness(img)
    brightness = 1.2
    imgEnhance = imgEnhance.enhance(brightness)
    imgEnhance = ImageEnhance.Contrast(imgEnhance)
    contrast = 2.5
    imgEnhance = imgEnhance.enhance(contrast)
    imgFlip = img.transpose(Image.FLIP_LEFT_RIGHT)
    imgFlipEnhance = imgEnhance.transpose(Image.FLIP_LEFT_RIGHT)
    raw_flip.append(imgFlip)
    raw_enhance.append(imgEnhance)
    raw_flip_enhance.append(imgFlipEnhance)

for i in range(len(label_data)):
    img = label_data[i].copy()
    imgFlip = img.transpose(Image.FLIP_LEFT_RIGHT)
    label_flip.append(imgFlip)

In [ ]:
no = 2
n=4
fig_width = n*4
fig_height = n*2
fig, ax = plt.subplots(2, n)
fig.set_figwidth(fig_width)
fig.set_figheight(fig_height)

ax[0,0].imshow(raw_data[no])
ax[0,1].imshow(label_data[no], cmap='gray')
ax[0,2].imshow(raw_enhance[no])
ax[0,3].imshow(label_data[no], cmap='gray')
ax[1,0].imshow(raw_flip[no])
ax[1,1].imshow(label_flip[no], cmap='gray')
ax[1,2].imshow(raw_flip_enhance[no])
ax[1,3].imshow(label_flip[no], cmap='gray')

Out[ ]:
<matplotlib.image.AxesImage at 0x7f17903565f8>

```

```
for i in range(len(raw_flip)):
    raw_flip[i].save(directoryFlip + '/' + raw_id[i] + ".jpg")

for i in range(len(label_flip)):
    label_flip[i].save(directoryMask + '/' + raw_id[i] + ".jpg")

for i in range(len(raw_flip_enhance)):
    raw_flip_enhance[i].save(directoryFlipEnhance + '/' + raw_id[i] + ".jpg")
```

Lampiran 2. Source Code Model

Import Library

```
In [1]:  
import pandas as pd  
import cv2  
import numpy as np  
from numpy import asarray  
import os  
import matplotlib.pyplot as plt  
from glob import glob  
import tensorflow as tf  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Concatenate, Conv2DTranspose  
from tensorflow.keras import Sequential, optimizers  
from keras.callbacks import EarlyStopping
```

Turn on GPU

```
In [2]:  
%tensorflow_version 2.x  
import tensorflow as tf  
device_name = tf.test.gpu_device_name()  
if device_name != '/device:GPU:0':  
    raise SystemError('GPU device not found')  
print('Found GPU at: {}'.format(device_name))
```

Found GPU at: /device:GPU:0

Mount to Google Drive

```
In [3]:  
from google.colab import drive  
drive.mount('/content/drive')  
Mounted at /content/drive
```

Load Data

```
In [4]:  
path = os.path.join("../content/drive/My Drive/Colab Notebooks/Dataset/", "Embrio")  
path_image = {os.path.splitext(os.path.basename(x))[0]: x  
             for x in glob(os.path.join(path, "Embryo", "*"))}  
path_imageenhance = {os.path.splitext(os.path.basename(x))[0]: x  
                     for x in glob(os.path.join(path, "EmbryoEnhance", "*"))}  
path_imageflip = {os.path.splitext(os.path.basename(x))[0]: x  
                  for x in glob(os.path.join(path, "EmbryoFlip", "*"))}  
path_imageflipenhance = {os.path.splitext(os.path.basename(x))[0]: x  
                           for x in glob(os.path.join(path, "EmbryoFlipEnhance", "*"))}  
path_imagemask = {os.path.splitext(os.path.basename(x))[0]: x  
                   for x in glob(os.path.join(path, "EmbryoMask", "*"))}  
path_imagemaskflip = {os.path.splitext(os.path.basename(x))[0]: x  
                      for x in glob(os.path.join(path, "EmbryoMaskFlip", "*"))}
```

```
In [5]:  
df = pd.read_csv("../content/drive/My Drive/Colab Notebooks/Dataset/Embrio/Embrio.csv")
```

```
In [6]:  
df.head()  
Out[6]:
```

	id	label
0	Amirah_1(atas)-2-1.jpg	train
1	Amirah_1-2.jpg	train
2	Amirah_1-x-1.jpg	train
3	Anggi_1.jpg	train
4	Anggi_1-2-1.jpg	train

```
In [7]:  
for i in range(len(df)):  
    df['id'][i] = df['id'][i].replace('.jpg', '')
```

Get Path Image

In [8]:

```
df['path_image'] = df['id'].map(path_image.get)
df['path_imageenhance'] = df['id'].map(path_imageenhance.get)
df['path_imageflip'] = df['id'].map(path_imageflip.get)
df['path_imagedflipenhance'] = df['id'].map(path_imagedflipenhance.get)
df['path_imagemask'] = df['id'].map(path_imagemask.get)
df['path_imagemaskflip'] = df['id'].map(path_imagemaskflip.get)
```

Set Image Size

In [9]:

```
IMG_HEIGHT = 512
IMG_WIDTH = 512
IMG_CHANNELS = 3
```

In [24]:

```
x_train = []
x_train_enhance = []
x_train_flip = []
x_train_flip_enhance = []
y_train = []
y_train_flip = []
x_val = []
x_val_enhance = []
x_val_flip = []
x_val_flip_enhance = []
y_val = []
y_val_flip = []
for i in range(len(df)):
    if df['label'][i]=='train':
        imgX = cv2.imread(df['path_image'][i])
        imgX = cv2.resize(imgX, (IMG_WIDTH,IMG_HEIGHT))
        x_train.append(imgX)
        imgXE = cv2.imread(df['path_imageenhance'][i])
        imgXE = cv2.resize(imgXE, (IMG_WIDTH,IMG_HEIGHT))
        x_train_enhance.append(imgXE)
        imgXF = cv2.imread(df['path_imageflip'][i])
        imgXF = cv2.resize(imgXF, (IMG_WIDTH,IMG_HEIGHT))
        x_train_flip.append(imgXF)
        imgXFE = cv2.imread(df['path_imagedflipenhance'][i])
        imgXFE = cv2.resize(imgXFE, (IMG_WIDTH,IMG_HEIGHT))
        x_train_flip_enhance.append(imgXFE)
        imgY = cv2.imread(df['path_imagemask'][i])
        imgY = cv2.cvtColor(imgY, cv2.COLOR_BGR2GRAY)
        y_train.append(imgY)
        # imgYF = cv2.imread(df['path_imagemaskflip'][i])
        # imgYF = cv2.resize(imgYF, (IMG_WIDTH,IMG_HEIGHT))
        # imgYF = cv2.cvtColor(imgYF, cv2.COLOR_BGR2GRAY)
        # y_train_flip.append(imgYF)
    elif df['label'][i]=='val':
        imgX = cv2.imread(df['path_image'][i])
        imgX = cv2.resize(imgX, (IMG_WIDTH,IMG_HEIGHT))
        x_val.append(imgX)
        # imgXE = cv2.imread(df['path_imageenhance'][i])
        # imgXE = cv2.resize(imgXE, (IMG_WIDTH,IMG_HEIGHT))
        # x_val_enhance.append(imgXE)
        # imgXF = cv2.imread(df['path_imageflip'][i])
        # imgXF = cv2.resize(imgXF, (IMG_WIDTH,IMG_HEIGHT))
        # x_val_flip.append(imgXF)
        # imgXFE = cv2.imread(df['path_imagedflipenhance'][i])
        # imgXFE = cv2.resize(imgXFE, (IMG_WIDTH,IMG_HEIGHT))
        # x_val_flip_enhance.append(imgXFE)
        imgY = cv2.imread(df['path_imagemask'][i])
        imgY = cv2.resize(imgY, (IMG_WIDTH,IMG_HEIGHT))
        imgY = cv2.cvtColor(imgY, cv2.COLOR_BGR2GRAY)
        y_val.append(imgY)
        # imgYF = cv2.imread(df['path_imagemaskflip'][i])
        # imgYF = cv2.resize(imgYF, (IMG_WIDTH,IMG_HEIGHT))
        # imgYF = cv2.cvtColor(imgYF, cv2.COLOR_BGR2GRAY)
        # y_val_flip.append(imgYF)
```

In [25]:

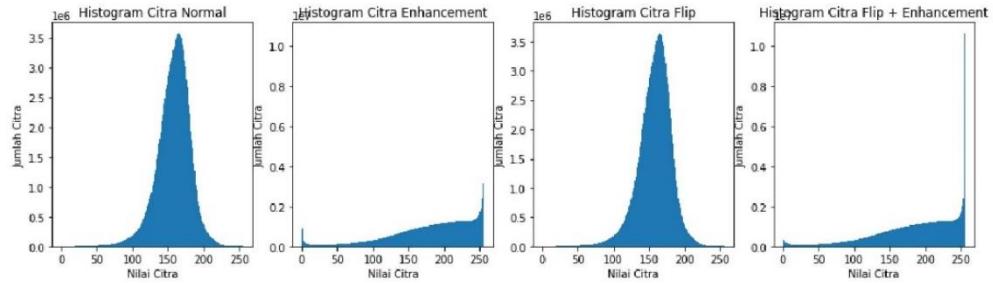
```
x_train = np.asarray(x_train)
x_train_enhance = np.asarray(x_train_enhance)
x_train_flip = np.asarray(x_train_flip)
x_train_flip_enhance = np.asarray(x_train_flip_enhance)
y_train = np.asarray(y_train)
y_train_flip = np.asarray(y_train_flip)
x_val = np.asarray(x_val)
x_val_enhance = np.asarray(x_val_enhance)
x_val_flip = np.asarray(x_val_flip)
x_val_flip_enhance = np.asarray(x_val_flip_enhance)
y_val = np.asarray(y_val)
y_val_flip = np.asarray(y_val_flip)
```

Histogram

```
In [26]:
n=4
fig_width = n*n
fig_height = n
fig, ax = plt.subplots(1, n)
fig.set_figwidth(fig_width)
fig.set_figheight(fig_height)

ax[0].hist(x_train.ravel(),256,[0,256])
ax[0].set_title("Histogram Citra Normal")
ax[0].set_xlabel("Nilai Citra")
ax[0].set_ylabel("Jumlah Citra")
ax[1].hist(x_train_enhance.ravel(),256,[0,256])
ax[1].set_title("Histogram Citra Enhancement")
ax[1].set_xlabel("Nilai Citra")
ax[1].set_ylabel("Jumlah Citra")
ax[2].hist(x_train_flip.ravel(),256,[0,256])
ax[2].set_title("Histogram Citra Flip")
ax[2].set_xlabel("Nilai Citra")
ax[2].set_ylabel("Jumlah Citra")
ax[3].hist(x_train_flip_enhance.ravel(),256,[0,256])
ax[3].set_title("Histogram Citra Flip + Enhancement")
ax[3].set_xlabel("Nilai Citra")
ax[3].set_ylabel("Jumlah Citra")

Out[26]:
Text(0, 0.5, 'Jumlah Citra')
```



```
In [13]:
x_train = x_train/255
x_train_enhance = x_train_enhance/255
x_train_flip = x_train_flip/255
x_train_flip_enhance = x_train_flip_enhance/255
y_train = y_train/255
y_train_flip = y_train_flip/255
x_val = x_val/255
x_val_enhance = x_val_enhance/255
x_val_flip = x_val_flip/255
x_val_flip_enhance = x_val_flip_enhance/255
y_val = y_val/255
y_val_flip = y_val_flip/255
```

```
In [14]:
y_train = np.where(y_train>0,1,0)
# y_train_flip = np.where(y_train_flip>0,1,0)
y_val = np.where(y_val>0,1,0)
# y_val_flip = np.where(y_val_flip>0,1,0)
```

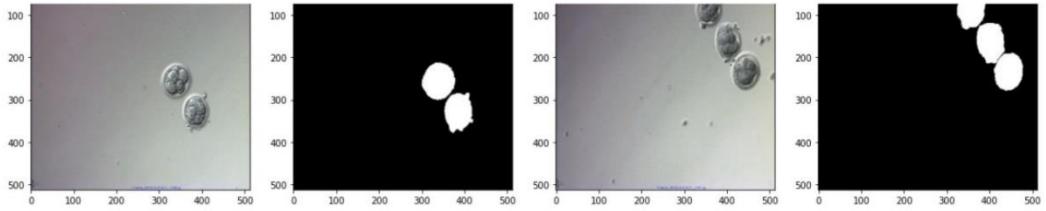
```
In [15]:
n=4
fig_width = 5*n
fig_height = 5*2
fig, ax = plt.subplots(1, n)
fig.set_figwidth(fig_width)
fig.set_figheight(fig_height)

ax[0].imshow(x_train[1])
ax[1].imshow(y_train[1],cmap='gray')
ax[2].imshow(x_train_enhance[1])
ax[3].imshow(y_train_enhance[1])

# ax[0, 0].imshow(x_train[1])
# ax[0, 1].imshow(y_train[1],cmap='gray')
# ax[0, 2].imshow(x_train_enhance[1])
# ax[0, 3].imshow(y_train_enhance[1])
# ax[1, 0].imshow(x_train_flip[1])
# ax[1, 1].imshow(y_train_flip[1],cmap='gray')
# ax[1, 2].imshow(x_train_flip_enhance[1])
# ax[1, 3].imshow(y_train_flip_enhance[1])
```

```
Out[15]:
<matplotlib.image.AxesImage at 0x7fb49d754630>
```





Import Model U-Net

In [16]:

```
# Build model
inputs = tf.keras.layers.Input(shape=(IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS))
print("Input layer = ", inputs.shape,"\\n")

# Encoder
C1 = Conv2D(16, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu")(inputs)
C1 = Conv2D(16, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu")(C1)
C1 = Dropout(0.5)(C1)
print("Convolutional layer 1 = ",C1.shape)
P1 = MaxPooling2D((2,2),strides=2)(C1)
print("MaxPool layer 1 = ",P1.shape,"\\n")

C2 = Conv2D(32, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu")(P1)
C2 = Conv2D(32, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same")(C2)
C2 = Dropout(0.5)(C2)
print("Convolutional layer 2 = ",C2.shape)
P2 = MaxPooling2D((2,2),strides=2)(C2)
print("MaxPool layer 2 = ",P2.shape,"\\n")

C3 = Conv2D(64, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu")(P2)
C3 = Conv2D(64, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same")(C3)
C3 = Dropout(0.5)(C3)
print("Convolutional layer 3 = ",C3.shape)
P3 = MaxPooling2D((2,2),strides=2)(C3)
print("MaxPool layer 3 = ",P3.shape,"\\n")

C4 = Conv2D(128, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu")(P3)
C4 = Conv2D(128, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same")(C4)
C4 = Dropout(0.5)(C4)
print("Convolutional layer 4 = ",C4.shape)
P4 = MaxPooling2D((2,2),strides=2)(C4)
print("MaxPool layer 4 = ",P4.shape,"\\n")

C5=Conv2D(256, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu") (P4)
C5=Conv2D(256, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same") (C5)
C5 = Dropout(0.5)(C5)
print("Convolutional layer 5 = ",C5.shape,"\\n")

# Decoder
U6 = Conv2DTranspose(128, kernel_size=(2,2),activation="relu", kernel_initializer='he_normal',padding="same",strides=(2,2))(C5)
print("UpSampling 1 = ",U6.shape)
U6 = Concatenate()([U6, C4])
print("Concatenate 1 = ",U6.shape)
C6 = Conv2D(128, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu") (U6)
C6 = Conv2D(128, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same") (C6)
print("Convolutional layer 6 = ",C6.shape,"\\n")

U7 = Conv2DTranspose(64, kernel_size=(2,2),activation="relu", kernel_initializer='he_normal',padding="same",strides=(2,2))(C6)
print("UpSampling 2 = ",U7.shape)
U7 = Concatenate()([U7, C3])
print("Concatenate 2 = ",U7.shape)
C7 = Conv2D(64, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu") (U7)
C7 = Conv2D(64, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same") (C7)
print("Convolutional layer 7 = ",C7.shape,"\\n")

U8 = Conv2DTranspose(32, kernel_size=(2,2),activation="relu", kernel_initializer='he_normal',padding="same",strides=(2,2))(C7)
print("UpSampling 3 = ",U8.shape)
U8 = Concatenate()([U8, C2])
print("Concatenate 3 = ",U8.shape)
C8 = Conv2D(32, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu") (U8)
C8 = Conv2D(32, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same") (C8)
print("Convolutional layer 8 = ",C8.shape,"\\n")

U9 = Conv2DTranspose(16, kernel_size=(2,2),activation="relu", kernel_initializer='he_normal',padding="same",strides=(2,2))(C8)
print("UpSampling 4 = ",U9.shape)
U9 = Concatenate()([U9, C1])
print("Concatenate 4 = ",U9.shape)
C9 = Conv2D(16, kernel_size=(3,3),padding="same", kernel_initializer='he_normal',activation="relu") (U9)
C9 = Conv2D(16, kernel_size=(3,3),activation="relu", kernel_initializer='he_normal',padding="same") (C9)
print("Convolutional layer 9 = ",C9.shape,"\\n")

output = Conv2D(1, kernel_size=(1,1),activation="sigmoid") (C9)
print("Output layer = ",output.shape)
```

```

model = tf.keras.Model(inputs=[inputs],outputs=[output])
Input layer = (None, 512, 512, 3)
Convolutional layer 1 = (None, 512, 512, 16)
MaxPool layer 1 = (None, 256, 256, 16)
Convolutional layer 2 = (None, 256, 256, 32)
MaxPool layer 2 = (None, 128, 128, 32)
Convolutional layer 3 = (None, 128, 128, 64)
MaxPool layer 3 = (None, 64, 64, 64)
Convolutional layer 4 = (None, 64, 64, 128)
MaxPool layer 4 = (None, 32, 32, 128)
Convolutional layer 5 = (None, 32, 32, 256)
UpSampling 1 = (None, 64, 64, 128)
Concatenate 1 = (None, 64, 64, 256)
Convolutional layer 6 = (None, 64, 64, 128)
UpSampling 2 = (None, 128, 128, 64)
Concatenate 2 = (None, 64, 64, 256)
Convolutional layer 7 = (None, 128, 128, 64)
UpSampling 3 = (None, 256, 256, 32)
Concatenate 3 = (None, 256, 256, 64)
Convolutional layer 8 = (None, 256, 256, 32)
UpSampling 4 = (None, 512, 512, 16)
Concatenate 4 = (None, 512, 512, 32)
Convolutional layer 9 = (None, 512, 512, 16)
Output layer = (None, 512, 512, 1)

In [17]:
model.compile(loss='categorical_crossentropy',optimizer=optimizers.Adam(),metrics=['accuracy'])
model.summary()

Model: "model"

```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[None, 512, 512, 3]	0	
conv2d (Conv2D)	(None, 512, 512, 16)	448	input_1[0][0]
conv2d_1 (Conv2D)	(None, 512, 512, 16)	2320	conv2d[0][0]
dropout (Dropout)	(None, 512, 512, 16)	0	conv2d_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 256, 16)	0	dropout[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 32)	4640	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 256, 256, 32)	9248	conv2d_2[0][0]
dropout_1 (Dropout)	(None, 256, 256, 32)	0	conv2d_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 32)	0	dropout_1[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 64)	18496	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 64)	36928	conv2d_4[0][0]
dropout_2 (Dropout)	(None, 128, 128, 64)	0	conv2d_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 64)	0	dropout_2[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_6[0][0]
dropout_3 (Dropout)	(None, 64, 64, 128)	0	conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 128)	0	dropout_3[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_8[0][0]
dropout_4 (Dropout)	(None, 32, 32, 256)	0	conv2d_9[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 64, 64, 128)	131200	dropout_4[0][0]
concatenate (Concatenate)	(None, 64, 64, 256)	0	conv2d_transpose[0][0] dropout_3[0][0]
conv2d_10 (Conv2D)	(None, 64, 64, 128)	295040	concatenate[0][0]
conv2d_11 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_10[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 128, 128, 64)	32832	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 128, 128, 128)	0	conv2d_transpose_1[0][0]

		dropout_2[0][0]
conv2d_12 (Conv2D)	(None, 128, 128, 64) 73792	concatenate_1[0][0]
conv2d_13 (Conv2D)	(None, 128, 128, 64) 36928	conv2d_12[0][0]
conv2d_transpose_2 (Conv2DTrans)	(None, 256, 256, 32) 8224	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 256, 256, 64) 0	conv2d transpose_2[0][0] dropout 1[0][0]
conv2d_14 (Conv2D)	(None, 256, 256, 32) 18464	concatenate_2[0][0]
conv2d_15 (Conv2D)	(None, 256, 256, 32) 9248	conv2d_14[0][0]
conv2d_transpose_3 (Conv2DTrans)	(None, 512, 512, 16) 2064	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 512, 512, 32) 0	conv2d transpose_3[0][0] dropout[0][0]
conv2d_16 (Conv2D)	(None, 512, 512, 16) 4624	concatenate_3[0][0]
conv2d_17 (Conv2D)	(None, 512, 512, 16) 2320	conv2d_16[0][0]
conv2d_18 (Conv2D)	(None, 512, 512, 1) 17	conv2d_17[0][0]

Total params: 1,941,105
Trainable params: 1,941,105
Non-trainable params: 0

In [18]:

```
#checkpoint
checkpoint=tf.keras.callbacks.ModelCheckpoint("../content/drive/My Drive/Colab Notebooks/Dataset/Embrio/Skripsi.h5",
verbose=1, save_best_only=True)

#Callbacks
Callbacks = [
    # EarlyStopping(monitor="val_accuracy", min_delta=0,patience=25, verbose=1, mode='max',baseline=None, re
    store_best_weights=True),
    # tf.keras.callbacks.TensorBoard(log_dir='log'),
    checkpoint
]
# Train the model
result4 = model.fit(x_train, y_train,batch_size=4,epochs=100,verbose=1,validation_data=(x_val, y_val),
validation_split=0.5,callbacks=Callbacks)

Epoch 1/100
29/29 [=====] - 17s 248ms/step - loss: 112.7702 - accuracy: 0.5476 - val_loss: 113.7478 - val
_accuracy: 0.9666
Epoch 0001: val_loss improved from inf to 113.74780, saving model to ../content/drive/My Drive/Colab Notebooks/D
ataset/Embrio/Skripsi.h5
Epoch 2/100
29/29 [=====] - 5s 169ms/step - loss: 94.6736 - accuracy: 0.9356 - val_loss: 84.1871 - val_ac
curacy: 0.9798
Epoch 0002: val_loss improved from 113.74780 to 84.18710, saving model to ../content/drive/My Drive/Colab Notebooks/D
ataset/Embrio/Skripsi.h5
Epoch 3/100
29/29 [=====] - 5s 171ms/step - loss: 84.2158 - accuracy: 0.9741 - val_loss: 84.9394 - val_ac
curacy: 0.9792
Epoch 0003: val_loss did not improve from 84.18710
Epoch 4/100
29/29 [=====] - 5s 172ms/step - loss: 82.9740 - accuracy: 0.9779 - val_loss: 80.4492 - val_ac
curacy: 0.9821
Epoch 0004: val_loss improved from 84.18710 to 80.44923, saving model to ../content/drive/My Drive/Colab Notebooks/D
ataset/Embrio/Skripsi.h5
Epoch 5/100
29/29 [=====] - 5s 172ms/step - loss: 84.8051 - accuracy: 0.9766 - val_loss: 82.6659 - val_ac
curacy: 0.9793
Epoch 0005: val_loss did not improve from 80.44923
Epoch 6/100
29/29 [=====] - 5s 173ms/step - loss: 82.2254 - accuracy: 0.9792 - val_loss: 79.1913 - val_ac
curacy: 0.9845
Epoch 0006: val_loss improved from 80.44923 to 79.19128, saving model to ../content/drive/My Drive/Colab Notebooks/D
ataset/Embrio/Skripsi.h5
Epoch 7/100
29/29 [=====] - 5s 174ms/step - loss: 84.3315 - accuracy: 0.9784 - val_loss: 80.1350 - val_ac
curacy: 0.9823
Epoch 0007: val_loss did not improve from 79.19128
Epoch 8/100
29/29 [=====] - 5s 175ms/step - loss: 88.0524 - accuracy: 0.9818 - val_loss: 80.2733 - val_ac
curacy: 0.9823
Epoch 0008: val_loss did not improve from 79.19128
Epoch 9/100
29/29 [=====] - 5s 174ms/step - loss: 81.8541 - accuracy: 0.9837 - val_loss: 82.5510 - val_ac
curacy: 0.9816
Epoch 0009: val_loss did not improve from 79.19128
Epoch 10/100
```

```

29/29 [=====] - 5s 175ms/step - loss: 86.3003 - accuracy: 0.9813 - val_loss: 79.8689 - val_accuracy: 0.9795
Epoch 00010: val_loss did not improve from 79.19128
Epoch 11/100
29/29 [=====] - 5s 180ms/step - loss: 85.1607 - accuracy: 0.9823 - val_loss: 80.0058 - val_accuracy: 0.9826
Epoch 00011: val_loss did not improve from 79.19128
Epoch 12/100
29/29 [=====] - 5s 176ms/step - loss: 85.2552 - accuracy: 0.9784 - val_loss: 79.1765 - val_accuracy: 0.9835
Epoch 00012: val_loss improved from 79.19128 to 79.17654, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 13/100
29/29 [=====] - 5s 179ms/step - loss: 89.4322 - accuracy: 0.9798 - val_loss: 78.8561 - val_accuracy: 0.9853
Epoch 00013: val_loss improved from 79.17654 to 78.85606, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 14/100
29/29 [=====] - 5s 180ms/step - loss: 88.2112 - accuracy: 0.9843 - val_loss: 79.8267 - val_accuracy: 0.9850
Epoch 00014: val_loss did not improve from 78.85606
Epoch 15/100
29/29 [=====] - 5s 181ms/step - loss: 81.5546 - accuracy: 0.9860 - val_loss: 79.6119 - val_accuracy: 0.9845
Epoch 00015: val_loss did not improve from 78.85606
Epoch 16/100
29/29 [=====] - 5s 181ms/step - loss: 86.1421 - accuracy: 0.9876 - val_loss: 79.1115 - val_accuracy: 0.9880
Epoch 00016: val_loss did not improve from 78.85606
Epoch 17/100
29/29 [=====] - 5s 183ms/step - loss: 87.4558 - accuracy: 0.9919 - val_loss: 78.7906 - val_accuracy: 0.9937
Epoch 00017: val_loss improved from 78.85606 to 78.79059, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 18/100
29/29 [=====] - 5s 183ms/step - loss: 87.8235 - accuracy: 0.9916 - val_loss: 79.2801 - val_accuracy: 0.9903
Epoch 00018: val_loss did not improve from 78.79059
Epoch 19/100
29/29 [=====] - 5s 185ms/step - loss: 83.5834 - accuracy: 0.9949 - val_loss: 78.7369 - val_accuracy: 0.9944
Epoch 00019: val_loss improved from 78.79059 to 78.73685, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 20/100
29/29 [=====] - 5s 184ms/step - loss: 90.4955 - accuracy: 0.9953 - val_loss: 79.0009 - val_accuracy: 0.9933
Epoch 00020: val_loss did not improve from 78.73685
Epoch 21/100
29/29 [=====] - 5s 182ms/step - loss: 83.2006 - accuracy: 0.9930 - val_loss: 78.7376 - val_accuracy: 0.9931
Epoch 00021: val_loss did not improve from 78.73685
Epoch 22/100
29/29 [=====] - 5s 182ms/step - loss: 85.5708 - accuracy: 0.9958 - val_loss: 78.6716 - val_accuracy: 0.9931
Epoch 00022: val_loss improved from 78.73685 to 78.67163, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 23/100
29/29 [=====] - 5s 181ms/step - loss: 86.6398 - accuracy: 0.9924 - val_loss: 79.6449 - val_accuracy: 0.9915
Epoch 00023: val_loss did not improve from 78.67163
Epoch 24/100
29/29 [=====] - 5s 181ms/step - loss: 85.3896 - accuracy: 0.9959 - val_loss: 79.2192 - val_accuracy: 0.9914
Epoch 00024: val_loss did not improve from 78.67163
Epoch 25/100
29/29 [=====] - 5s 180ms/step - loss: 78.2899 - accuracy: 0.9967 - val_loss: 78.6959 - val_accuracy: 0.9941
Epoch 00025: val_loss did not improve from 78.67163
Epoch 26/100
29/29 [=====] - 5s 180ms/step - loss: 84.5507 - accuracy: 0.9962 - val_loss: 79.2162 - val_accuracy: 0.9906
Epoch 00026: val_loss did not improve from 78.67163
Epoch 27/100
29/29 [=====] - 5s 179ms/step - loss: 84.6920 - accuracy: 0.9956 - val_loss: 78.4833 - val_accuracy: 0.9960
Epoch 00027: val_loss improved from 78.67163 to 78.48330, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 28/100
29/29 [=====] - 5s 181ms/step - loss: 77.9495 - accuracy: 0.9967 - val_loss: 79.0005 - val_accuracy: 0.9917

```

```

Epoch 00028: val_loss did not improve from 78.48330
Epoch 29/100
29/29 [=====] - 5s 180ms/step - loss: 86.1750 - accuracy: 0.9966 - val_loss: 78.6639 - val_accuracy: 0.9954

Epoch 00029: val_loss did not improve from 78.48330
Epoch 30/100
29/29 [=====] - 5s 181ms/step - loss: 81.9073 - accuracy: 0.9967 - val_loss: 78.4148 - val_accuracy: 0.9963

Epoch 00030: val_loss improved from 78.48330 to 78.41476, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 31/100
29/29 [=====] - 5s 181ms/step - loss: 82.7383 - accuracy: 0.9970 - val_loss: 78.8108 - val_accuracy: 0.9944

Epoch 00031: val_loss did not improve from 78.41476
Epoch 32/100
29/29 [=====] - 5s 182ms/step - loss: 86.5457 - accuracy: 0.9966 - val_loss: 78.5285 - val_accuracy: 0.9944

Epoch 00032: val_loss did not improve from 78.41476
Epoch 33/100
29/29 [=====] - 5s 183ms/step - loss: 83.4108 - accuracy: 0.9963 - val_loss: 78.3965 - val_accuracy: 0.9962

Epoch 00033: val_loss improved from 78.41476 to 78.39653, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 34/100
29/29 [=====] - 5s 185ms/step - loss: 83.5595 - accuracy: 0.9970 - val_loss: 79.2210 - val_accuracy: 0.9937

Epoch 00034: val_loss did not improve from 78.39653
Epoch 35/100
29/29 [=====] - 5s 185ms/step - loss: 81.2220 - accuracy: 0.9970 - val_loss: 78.3493 - val_accuracy: 0.9967

Epoch 00035: val_loss improved from 78.39653 to 78.34927, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 36/100
29/29 [=====] - 5s 186ms/step - loss: 81.0486 - accuracy: 0.9956 - val_loss: 78.6020 - val_accuracy: 0.9953

Epoch 00036: val_loss did not improve from 78.34927
Epoch 37/100
29/29 [=====] - 5s 185ms/step - loss: 87.9317 - accuracy: 0.9971 - val_loss: 78.4457 - val_accuracy: 0.9958

Epoch 00037: val_loss did not improve from 78.34927
Epoch 38/100
29/29 [=====] - 5s 184ms/step - loss: 80.6535 - accuracy: 0.9965 - val_loss: 78.7093 - val_accuracy: 0.9952

Epoch 00038: val_loss did not improve from 78.34927
Epoch 39/100
29/29 [=====] - 5s 184ms/step - loss: 82.8071 - accuracy: 0.9968 - val_loss: 78.4800 - val_accuracy: 0.9960

Epoch 00039: val_loss did not improve from 78.34927
Epoch 40/100
29/29 [=====] - 5s 184ms/step - loss: 85.9484 - accuracy: 0.9971 - val_loss: 79.5425 - val_accuracy: 0.9926

Epoch 00040: val_loss did not improve from 78.34927
Epoch 41/100
29/29 [=====] - 5s 183ms/step - loss: 86.5216 - accuracy: 0.9965 - val_loss: 78.6949 - val_accuracy: 0.9953

Epoch 00041: val_loss did not improve from 78.34927
Epoch 42/100
29/29 [=====] - 5s 181ms/step - loss: 87.3126 - accuracy: 0.9963 - val_loss: 78.8973 - val_accuracy: 0.9941

Epoch 00042: val_loss did not improve from 78.34927
Epoch 43/100
29/29 [=====] - 5s 182ms/step - loss: 82.4582 - accuracy: 0.9948 - val_loss: 79.1941 - val_accuracy: 0.9934

Epoch 00043: val_loss did not improve from 78.34927
Epoch 44/100
29/29 [=====] - 5s 183ms/step - loss: 85.8734 - accuracy: 0.9928 - val_loss: 78.8222 - val_accuracy: 0.9944

Epoch 00044: val_loss did not improve from 78.34927
Epoch 45/100
29/29 [=====] - 5s 183ms/step - loss: 89.2560 - accuracy: 0.9949 - val_loss: 80.4403 - val_accuracy: 0.9898

Epoch 00045: val_loss did not improve from 78.34927
Epoch 46/100
29/29 [=====] - 5s 183ms/step - loss: 81.6635 - accuracy: 0.9903 - val_loss: 78.4865 - val_accuracy: 0.9943

Epoch 00046: val_loss did not improve from 78.34927
Epoch 47/100
29/29 [=====] - 5s 182ms/step - loss: 84.2724 - accuracy: 0.9922 - val_loss: 79.7287 - val_accuracy: 0.9776

```

```

accuracy: 0.9940
Epoch 00047: val_loss did not improve from 78.34927
Epoch 48/100
29/29 [=====] - 5s 182ms/step - loss: 82.7135 - accuracy: 0.9921 - val_loss: 78.7690 - val_accuracy: 0.9938
Epoch 00048: val_loss did not improve from 78.34927
Epoch 49/100
29/29 [=====] - 5s 182ms/step - loss: 85.8394 - accuracy: 0.9955 - val_loss: 78.4279 - val_accuracy: 0.9958
Epoch 00049: val_loss did not improve from 78.34927
Epoch 50/100
29/29 [=====] - 5s 184ms/step - loss: 84.3960 - accuracy: 0.9969 - val_loss: 78.4936 - val_accuracy: 0.9947
Epoch 00050: val_loss did not improve from 78.34927
Epoch 51/100
29/29 [=====] - 5s 183ms/step - loss: 87.7198 - accuracy: 0.9971 - val_loss: 78.5342 - val_accuracy: 0.9956
Epoch 00051: val_loss did not improve from 78.34927
Epoch 52/100
29/29 [=====] - 5s 184ms/step - loss: 89.7753 - accuracy: 0.9965 - val_loss: 78.4991 - val_accuracy: 0.9951
Epoch 00052: val_loss did not improve from 78.34927
Epoch 53/100
29/29 [=====] - 5s 183ms/step - loss: 83.7462 - accuracy: 0.9974 - val_loss: 78.8617 - val_accuracy: 0.9947
Epoch 00053: val_loss did not improve from 78.34927
Epoch 54/100
29/29 [=====] - 5s 184ms/step - loss: 82.1881 - accuracy: 0.9973 - val_loss: 78.9108 - val_accuracy: 0.9946
Epoch 00054: val_loss did not improve from 78.34927
Epoch 55/100
29/29 [=====] - 5s 183ms/step - loss: 83.9898 - accuracy: 0.9973 - val_loss: 78.5099 - val_accuracy: 0.9957
Epoch 00055: val_loss did not improve from 78.34927
Epoch 56/100
29/29 [=====] - 5s 184ms/step - loss: 83.3505 - accuracy: 0.9975 - val_loss: 78.3301 - val_accuracy: 0.9964
Epoch 00056: val_loss improved from 78.34927 to 78.33015, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embrico/Skripsi.h5
Epoch 57/100
29/29 [=====] - 5s 183ms/step - loss: 84.1483 - accuracy: 0.9974 - val_loss: 78.6746 - val_accuracy: 0.9947
Epoch 00057: val_loss did not improve from 78.33015
Epoch 58/100
29/29 [=====] - 5s 184ms/step - loss: 81.8202 - accuracy: 0.9975 - val_loss: 79.2219 - val_accuracy: 0.9944
Epoch 00058: val_loss did not improve from 78.33015
Epoch 59/100
29/29 [=====] - 5s 184ms/step - loss: 81.5145 - accuracy: 0.9972 - val_loss: 78.8046 - val_accuracy: 0.9956
Epoch 00059: val_loss did not improve from 78.33015
Epoch 60/100
29/29 [=====] - 5s 183ms/step - loss: 84.8164 - accuracy: 0.9972 - val_loss: 78.9454 - val_accuracy: 0.9947
Epoch 00060: val_loss did not improve from 78.33015
Epoch 61/100
29/29 [=====] - 5s 184ms/step - loss: 89.2761 - accuracy: 0.9973 - val_loss: 78.3160 - val_accuracy: 0.9966
Epoch 00061: val_loss improved from 78.33015 to 78.31596, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embrico/Skripsi.h5
Epoch 62/100
29/29 [=====] - 5s 182ms/step - loss: 86.2264 - accuracy: 0.9973 - val_loss: 78.2982 - val_accuracy: 0.9968
Epoch 00062: val_loss improved from 78.31596 to 78.29825, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embrico/Skripsi.h5
Epoch 63/100
29/29 [=====] - 5s 183ms/step - loss: 84.8595 - accuracy: 0.9976 - val_loss: 78.8772 - val_accuracy: 0.9955
Epoch 00063: val_loss did not improve from 78.29825
Epoch 64/100
29/29 [=====] - 5s 183ms/step - loss: 84.0604 - accuracy: 0.9976 - val_loss: 79.1444 - val_accuracy: 0.9947
Epoch 00064: val_loss did not improve from 78.29825
Epoch 65/100
29/29 [=====] - 5s 183ms/step - loss: 88.5131 - accuracy: 0.9973 - val_loss: 78.4829 - val_accuracy: 0.9965
Epoch 00065: val_loss did not improve from 78.29825
Epoch 66/100
29/29 [=====] - 5s 183ms/step - loss: 82.7208 - accuracy: 0.9976 - val_loss: 78.6238 - val_accuracy: 0.9955

```

```

curacy: 0.9962
Epoch 00066: val_loss did not improve from 78.29825
Epoch 67/100
29/29 [=====] - 5s 183ms/step - loss: 81.4731 - accuracy: 0.9977 - val_loss: 78.4935 - val_accuracy: 0.9964
Epoch 00067: val_loss did not improve from 78.29825
Epoch 68/100
29/29 [=====] - 5s 181ms/step - loss: 82.6179 - accuracy: 0.9977 - val_loss: 78.6435 - val_accuracy: 0.9963
Epoch 00068: val_loss did not improve from 78.29825
Epoch 69/100
29/29 [=====] - 5s 182ms/step - loss: 79.2470 - accuracy: 0.9977 - val_loss: 78.6432 - val_accuracy: 0.9961
Epoch 00069: val_loss did not improve from 78.29825
Epoch 70/100
29/29 [=====] - 5s 182ms/step - loss: 86.5756 - accuracy: 0.9974 - val_loss: 78.8104 - val_accuracy: 0.9957
Epoch 00070: val_loss did not improve from 78.29825
Epoch 71/100
29/29 [=====] - 5s 185ms/step - loss: 85.9052 - accuracy: 0.9977 - val_loss: 78.8113 - val_accuracy: 0.9958
Epoch 00071: val_loss did not improve from 78.29825
Epoch 72/100
29/29 [=====] - 5s 184ms/step - loss: 83.2967 - accuracy: 0.9978 - val_loss: 79.4422 - val_accuracy: 0.9949
Epoch 00072: val_loss did not improve from 78.29825
Epoch 73/100
29/29 [=====] - 5s 184ms/step - loss: 84.8179 - accuracy: 0.9977 - val_loss: 78.4992 - val_accuracy: 0.9965
Epoch 00073: val_loss did not improve from 78.29825
Epoch 74/100
29/29 [=====] - 5s 184ms/step - loss: 84.1888 - accuracy: 0.9977 - val_loss: 78.5269 - val_accuracy: 0.9965
Epoch 00074: val_loss did not improve from 78.29825
Epoch 75/100
29/29 [=====] - 5s 184ms/step - loss: 88.8977 - accuracy: 0.9977 - val_loss: 79.1429 - val_accuracy: 0.9957
Epoch 00075: val_loss did not improve from 78.29825
Epoch 76/100
29/29 [=====] - 5s 183ms/step - loss: 87.9122 - accuracy: 0.9977 - val_loss: 78.6284 - val_accuracy: 0.9965
Epoch 00076: val_loss did not improve from 78.29825
Epoch 77/100
29/29 [=====] - 5s 183ms/step - loss: 86.4772 - accuracy: 0.9978 - val_loss: 78.9701 - val_accuracy: 0.9955
Epoch 00077: val_loss did not improve from 78.29825
Epoch 78/100
29/29 [=====] - 5s 184ms/step - loss: 92.1129 - accuracy: 0.9976 - val_loss: 79.0219 - val_accuracy: 0.9960
Epoch 00078: val_loss did not improve from 78.29825
Epoch 79/100
29/29 [=====] - 5s 184ms/step - loss: 88.6147 - accuracy: 0.9976 - val_loss: 78.6718 - val_accuracy: 0.9967
Epoch 00079: val_loss did not improve from 78.29825
Epoch 80/100
29/29 [=====] - 5s 183ms/step - loss: 78.5927 - accuracy: 0.9980 - val_loss: 78.7928 - val_accuracy: 0.9964
Epoch 00080: val_loss did not improve from 78.29825
Epoch 81/100
29/29 [=====] - 5s 188ms/step - loss: 86.6038 - accuracy: 0.9978 - val_loss: 78.6740 - val_accuracy: 0.9965
Epoch 00081: val_loss did not improve from 78.29825
Epoch 82/100
29/29 [=====] - 5s 185ms/step - loss: 87.3663 - accuracy: 0.9977 - val_loss: 79.0828 - val_accuracy: 0.9961
Epoch 00082: val_loss did not improve from 78.29825
Epoch 83/100
29/29 [=====] - 5s 183ms/step - loss: 80.6646 - accuracy: 0.9978 - val_loss: 79.2053 - val_accuracy: 0.9958
Epoch 00083: val_loss did not improve from 78.29825
Epoch 84/100
29/29 [=====] - 5s 184ms/step - loss: 84.6275 - accuracy: 0.9977 - val_loss: 78.3568 - val_accuracy: 0.9972
Epoch 00084: val_loss did not improve from 78.29825
Epoch 85/100
29/29 [=====] - 5s 183ms/step - loss: 85.2021 - accuracy: 0.9978 - val_loss: 78.7598 - val_accuracy: 0.9965
Epoch 00085: val_loss did not improve from 78.29825

```

```

Epoch 86/100
29/29 [=====] - 5s 184ms/step - loss: 87.0164 - accuracy: 0.9978 - val_loss: 79.3379 - val_accuracy: 0.9957

Epoch 00086: val_loss did not improve from 78.29825
Epoch 87/100
29/29 [=====] - 5s 184ms/step - loss: 81.2458 - accuracy: 0.9978 - val_loss: 79.2712 - val_accuracy: 0.9953

Epoch 00087: val_loss did not improve from 78.29825
Epoch 88/100
29/29 [=====] - 5s 183ms/step - loss: 80.0443 - accuracy: 0.9975 - val_loss: 80.9801 - val_accuracy: 0.9936

Epoch 00088: val_loss did not improve from 78.29825
Epoch 89/100
29/29 [=====] - 5s 184ms/step - loss: 81.7094 - accuracy: 0.9974 - val_loss: 78.8671 - val_accuracy: 0.9966

Epoch 00089: val_loss did not improve from 78.29825
Epoch 90/100
29/29 [=====] - 5s 183ms/step - loss: 84.4127 - accuracy: 0.9977 - val_loss: 79.1976 - val_accuracy: 0.9959

Epoch 00090: val_loss did not improve from 78.29825
Epoch 91/100
29/29 [=====] - 5s 184ms/step - loss: 77.5640 - accuracy: 0.9980 - val_loss: 78.9671 - val_accuracy: 0.9965

Epoch 00091: val_loss did not improve from 78.29825
Epoch 92/100
29/29 [=====] - 5s 184ms/step - loss: 84.9204 - accuracy: 0.9972 - val_loss: 79.2571 - val_accuracy: 0.9957

Epoch 00092: val_loss did not improve from 78.29825
Epoch 93/100
29/29 [=====] - 5s 183ms/step - loss: 85.6950 - accuracy: 0.9977 - val_loss: 78.5933 - val_accuracy: 0.9967

Epoch 00093: val_loss did not improve from 78.29825
Epoch 94/100
29/29 [=====] - 5s 183ms/step - loss: 87.7581 - accuracy: 0.9975 - val_loss: 78.6781 - val_accuracy: 0.9965

Epoch 00094: val_loss did not improve from 78.29825
Epoch 95/100
29/29 [=====] - 5s 183ms/step - loss: 84.1135 - accuracy: 0.9977 - val_loss: 78.5694 - val_accuracy: 0.9964

Epoch 00095: val_loss did not improve from 78.29825
Epoch 96/100
29/29 [=====] - 5s 183ms/step - loss: 85.4381 - accuracy: 0.9976 - val_loss: 78.8970 - val_accuracy: 0.9963

Epoch 00096: val_loss did not improve from 78.29825
Epoch 97/100
29/29 [=====] - 5s 182ms/step - loss: 83.4283 - accuracy: 0.9979 - val_loss: 78.6821 - val_accuracy: 0.9967

Epoch 00097: val_loss did not improve from 78.29825
Epoch 98/100
29/29 [=====] - 5s 182ms/step - loss: 83.0752 - accuracy: 0.9977 - val_loss: 78.7528 - val_accuracy: 0.9966

Epoch 00098: val_loss did not improve from 78.29825
Epoch 99/100
29/29 [=====] - 5s 183ms/step - loss: 86.1029 - accuracy: 0.9975 - val_loss: 78.8299 - val_accuracy: 0.9960

Epoch 00099: val_loss did not improve from 78.29825
Epoch 100/100
29/29 [=====] - 5s 182ms/step - loss: 90.4661 - accuracy: 0.9978 - val_loss: 79.1073 - val_accuracy: 0.9959

Epoch 00100: val_loss did not improve from 78.29825

In [19]:
#checkpoint
checkpoint=tf.keras.callbacks.ModelCheckpoint("../content/drive/My Drive/Colab Notebooks/Dataset/Embroi/Skripsi.h5",
verbose=1, save_best_only=True)

#Callbacks
Callbacks = [
    # EarlyStopping(monitor="val_accuracy", min_delta=0, patience=25, verbose=1, mode='max', baseline=None, restore_best_weights=True),
    # tf.keras.callbacks.TensorBoard(log_dir='log'),
    checkpoint
]
# Train the model
result8 = model.fit(x_train, y_ttrain,batch_size=8,epochs=100,verbose=1,validation_data=(x_val, y_val),
validation_split=0.5,callbacks=Callbacks)

Epoch 1/100
15/15 [=====] - 8s 399ms/step - loss: 84.0373 - accuracy: 0.9979 - val_loss: 78.9049 - val_accuracy: 0.9963

Epoch 00001: val loss improved from inf to 78.90488, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset
```

```

/Embric/Skrripsi.h5
Epoch 2/100
15/15 [=====] - 5s 353ms/step - loss: 83.9932 - accuracy: 0.9981 - val_loss: 78.9876 - val_accuracy: 0.9965

Epoch 0002: val_loss did not improve from 78.90488
Epoch 3/100
15/15 [=====] - 5s 352ms/step - loss: 83.9944 - accuracy: 0.9981 - val_loss: 79.5010 - val_accuracy: 0.9958

Epoch 0003: val_loss did not improve from 78.90488
Epoch 4/100
15/15 [=====] - 5s 354ms/step - loss: 83.9958 - accuracy: 0.9981 - val_loss: 78.9925 - val_accuracy: 0.9966

Epoch 0004: val_loss did not improve from 78.90488
Epoch 5/100
15/15 [=====] - 5s 353ms/step - loss: 83.9988 - accuracy: 0.9981 - val_loss: 78.8950 - val_accuracy: 0.9967

Epoch 0005: val_loss improved from 78.90488 to 78.89500, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 6/100
15/15 [=====] - 5s 353ms/step - loss: 83.9812 - accuracy: 0.9981 - val_loss: 79.0950 - val_accuracy: 0.9967

Epoch 0006: val_loss did not improve from 78.89500
Epoch 7/100
15/15 [=====] - 5s 354ms/step - loss: 83.9667 - accuracy: 0.9981 - val_loss: 79.3344 - val_accuracy: 0.9961

Epoch 0007: val_loss did not improve from 78.89500
Epoch 8/100
15/15 [=====] - 5s 354ms/step - loss: 83.9768 - accuracy: 0.9981 - val_loss: 79.0939 - val_accuracy: 0.9966

Epoch 0008: val_loss did not improve from 78.89500
Epoch 9/100
15/15 [=====] - 5s 356ms/step - loss: 83.9903 - accuracy: 0.9981 - val_loss: 78.5906 - val_accuracy: 0.9970

Epoch 0009: val_loss improved from 78.89500 to 78.59057, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 10/100
15/15 [=====] - 5s 355ms/step - loss: 83.9576 - accuracy: 0.9981 - val_loss: 79.0017 - val_accuracy: 0.9968

Epoch 0010: val_loss did not improve from 78.59057
Epoch 11/100
15/15 [=====] - 5s 356ms/step - loss: 83.9459 - accuracy: 0.9981 - val_loss: 79.0377 - val_accuracy: 0.9967

Epoch 0011: val_loss did not improve from 78.59057
Epoch 12/100
15/15 [=====] - 5s 355ms/step - loss: 83.9318 - accuracy: 0.9982 - val_loss: 79.2245 - val_accuracy: 0.9966

Epoch 0012: val_loss did not improve from 78.59057
Epoch 13/100
15/15 [=====] - 5s 356ms/step - loss: 83.9472 - accuracy: 0.9981 - val_loss: 79.1418 - val_accuracy: 0.9966

Epoch 0013: val_loss did not improve from 78.59057
Epoch 14/100
15/15 [=====] - 5s 355ms/step - loss: 83.9351 - accuracy: 0.9981 - val_loss: 79.0293 - val_accuracy: 0.9968

Epoch 0014: val_loss did not improve from 78.59057
Epoch 15/100
15/15 [=====] - 5s 354ms/step - loss: 83.9299 - accuracy: 0.9982 - val_loss: 79.6110 - val_accuracy: 0.9962

Epoch 0015: val_loss did not improve from 78.59057
Epoch 16/100
15/15 [=====] - 5s 355ms/step - loss: 83.9446 - accuracy: 0.9982 - val_loss: 79.1645 - val_accuracy: 0.9966

Epoch 0016: val_loss did not improve from 78.59057
Epoch 17/100
15/15 [=====] - 5s 353ms/step - loss: 83.9313 - accuracy: 0.9982 - val_loss: 79.2987 - val_accuracy: 0.9966

Epoch 0017: val_loss did not improve from 78.59057
Epoch 18/100
15/15 [=====] - 5s 354ms/step - loss: 83.9260 - accuracy: 0.9981 - val_loss: 79.4454 - val_accuracy: 0.9964

Epoch 0018: val_loss did not improve from 78.59057
Epoch 19/100
15/15 [=====] - 5s 356ms/step - loss: 83.9322 - accuracy: 0.9981 - val_loss: 79.9409 - val_accuracy: 0.9959

Epoch 0019: val_loss did not improve from 78.59057
Epoch 20/100
15/15 [=====] - 5s 353ms/step - loss: 83.9218 - accuracy: 0.9982 - val_loss: 78.9203 - val_accuracy: 0.9969

Epoch 0020: val_loss did not improve from 78.59057

```

```

Epoch 00021: val_loss did not improve from 78.59057
Epoch 21/100
15/15 [=====] - 5s 352ms/step - loss: 83.9140 - accuracy: 0.9982 - val_loss: 79.7830 - val_accuracy: 0.9962

Epoch 00021: val_loss did not improve from 78.59057
Epoch 22/100
15/15 [=====] - 5s 356ms/step - loss: 83.9175 - accuracy: 0.9982 - val_loss: 79.0319 - val_accuracy: 0.9967

Epoch 00021: val_loss did not improve from 78.59057
Epoch 23/100
15/15 [=====] - 5s 352ms/step - loss: 83.9108 - accuracy: 0.9982 - val_loss: 79.3977 - val_accuracy: 0.9964

Epoch 00023: val_loss did not improve from 78.59057
Epoch 24/100
15/15 [=====] - 5s 354ms/step - loss: 83.9093 - accuracy: 0.9982 - val_loss: 79.5737 - val_accuracy: 0.9963

Epoch 00024: val_loss did not improve from 78.59057
Epoch 25/100
15/15 [=====] - 5s 354ms/step - loss: 83.9128 - accuracy: 0.9982 - val_loss: 79.2219 - val_accuracy: 0.9967

Epoch 00025: val_loss did not improve from 78.59057
Epoch 26/100
15/15 [=====] - 5s 355ms/step - loss: 83.9109 - accuracy: 0.9981 - val_loss: 79.2791 - val_accuracy: 0.9966

Epoch 00026: val_loss did not improve from 78.59057
Epoch 27/100
15/15 [=====] - 5s 354ms/step - loss: 83.8968 - accuracy: 0.9981 - val_loss: 79.9362 - val_accuracy: 0.9961

Epoch 00027: val_loss did not improve from 78.59057
Epoch 28/100
15/15 [=====] - 5s 354ms/step - loss: 83.9022 - accuracy: 0.9982 - val_loss: 79.2926 - val_accuracy: 0.9967

Epoch 00028: val_loss did not improve from 78.59057
Epoch 29/100
15/15 [=====] - 5s 354ms/step - loss: 83.9026 - accuracy: 0.9980 - val_loss: 79.5728 - val_accuracy: 0.9963

Epoch 00029: val_loss did not improve from 78.59057
Epoch 30/100
15/15 [=====] - 5s 352ms/step - loss: 83.9079 - accuracy: 0.9982 - val_loss: 79.2396 - val_accuracy: 0.9968

Epoch 00030: val_loss did not improve from 78.59057
Epoch 31/100
15/15 [=====] - 5s 354ms/step - loss: 83.8837 - accuracy: 0.9982 - val_loss: 79.5220 - val_accuracy: 0.9965

Epoch 00031: val_loss did not improve from 78.59057
Epoch 32/100
15/15 [=====] - 5s 353ms/step - loss: 83.8870 - accuracy: 0.9982 - val_loss: 79.3422 - val_accuracy: 0.9967

Epoch 00032: val_loss did not improve from 78.59057
Epoch 33/100
15/15 [=====] - 5s 354ms/step - loss: 83.8993 - accuracy: 0.9982 - val_loss: 79.8429 - val_accuracy: 0.9962

Epoch 00033: val_loss did not improve from 78.59057
Epoch 34/100
15/15 [=====] - 5s 356ms/step - loss: 83.9170 - accuracy: 0.9982 - val_loss: 79.2274 - val_accuracy: 0.9969

Epoch 00034: val_loss did not improve from 78.59057
Epoch 35/100
15/15 [=====] - 5s 354ms/step - loss: 83.9288 - accuracy: 0.9980 - val_loss: 78.9338 - val_accuracy: 0.9970

Epoch 00035: val_loss did not improve from 78.59057
Epoch 36/100
15/15 [=====] - 5s 353ms/step - loss: 83.8838 - accuracy: 0.9982 - val_loss: 80.1876 - val_accuracy: 0.9961

Epoch 00036: val_loss did not improve from 78.59057
Epoch 37/100
15/15 [=====] - 5s 355ms/step - loss: 83.8906 - accuracy: 0.9982 - val_loss: 79.1694 - val_accuracy: 0.9969

Epoch 00037: val_loss did not improve from 78.59057
Epoch 38/100
15/15 [=====] - 5s 355ms/step - loss: 83.8762 - accuracy: 0.9982 - val_loss: 79.9461 - val_accuracy: 0.9965

Epoch 00038: val_loss did not improve from 78.59057
Epoch 39/100
15/15 [=====] - 5s 354ms/step - loss: 83.8493 - accuracy: 0.9984 - val_loss: 80.0348 - val_accuracy: 0.9963

Epoch 00039: val_loss did not improve from 78.59057
Epoch 40/100
15/15 [=====] - 5s 354ms/step - loss: 83.8795 - accuracy: 0.9983 - val_loss: 79.5666 - val_accuracy: 0.9966

```

```

curacy: 0.9964
Epoch 00040: val_loss did not improve from 78.59057
Epoch 41/100
15/15 [=====] - 5s 355ms/step - loss: 83.8663 - accuracy: 0.9983 - val_loss: 79.4679 - val_accuracy: 0.9966
Epoch 00041: val_loss did not improve from 78.59057
Epoch 42/100
15/15 [=====] - 5s 355ms/step - loss: 83.8743 - accuracy: 0.9982 - val_loss: 79.5686 - val_accuracy: 0.9965
Epoch 00042: val_loss did not improve from 78.59057
Epoch 43/100
15/15 [=====] - 5s 362ms/step - loss: 83.8576 - accuracy: 0.9983 - val_loss: 79.7611 - val_accuracy: 0.9965
Epoch 00043: val_loss did not improve from 78.59057
Epoch 44/100
15/15 [=====] - 5s 355ms/step - loss: 83.8639 - accuracy: 0.9983 - val_loss: 79.4323 - val_accuracy: 0.9967
Epoch 00044: val_loss did not improve from 78.59057
Epoch 45/100
15/15 [=====] - 5s 355ms/step - loss: 83.8580 - accuracy: 0.9983 - val_loss: 80.1363 - val_accuracy: 0.9963
Epoch 00045: val_loss did not improve from 78.59057
Epoch 46/100
15/15 [=====] - 5s 355ms/step - loss: 83.8482 - accuracy: 0.9982 - val_loss: 79.2217 - val_accuracy: 0.9968
Epoch 00046: val_loss did not improve from 78.59057
Epoch 47/100
15/15 [=====] - 5s 355ms/step - loss: 83.8431 - accuracy: 0.9982 - val_loss: 79.5708 - val_accuracy: 0.9967
Epoch 00047: val_loss did not improve from 78.59057
Epoch 48/100
15/15 [=====] - 5s 354ms/step - loss: 83.9326 - accuracy: 0.9974 - val_loss: 79.9918 - val_accuracy: 0.996
Epoch 00048: val_loss did not improve from 78.59057
Epoch 49/100
15/15 [=====] - 5s 354ms/step - loss: 83.8896 - accuracy: 0.9981 - val_loss: 80.2097 - val_accuracy: 0.9960
Epoch 00049: val_loss did not improve from 78.59057
Epoch 50/100
15/15 [=====] - 5s 354ms/step - loss: 83.8687 - accuracy: 0.9982 - val_loss: 79.5361 - val_accuracy: 0.9965
Epoch 00050: val_loss did not improve from 78.59057
Epoch 51/100
15/15 [=====] - 5s 353ms/step - loss: 83.8380 - accuracy: 0.9984 - val_loss: 79.5269 - val_accuracy: 0.9966
Epoch 00051: val_loss did not improve from 78.59057
Epoch 52/100
15/15 [=====] - 5s 356ms/step - loss: 83.8405 - accuracy: 0.9983 - val_loss: 79.3075 - val_accuracy: 0.9968
Epoch 00052: val_loss did not improve from 78.59057
Epoch 53/100
15/15 [=====] - 5s 354ms/step - loss: 83.8485 - accuracy: 0.9983 - val_loss: 79.6345 - val_accuracy: 0.9964
Epoch 00053: val_loss did not improve from 78.59057
Epoch 54/100
15/15 [=====] - 5s 357ms/step - loss: 83.8305 - accuracy: 0.9983 - val_loss: 79.5146 - val_accuracy: 0.9968
Epoch 00054: val_loss did not improve from 78.59057
Epoch 55/100
15/15 [=====] - 5s 356ms/step - loss: 83.8370 - accuracy: 0.9983 - val_loss: 79.5229 - val_accuracy: 0.9967
Epoch 00055: val_loss did not improve from 78.59057
Epoch 56/100
15/15 [=====] - 5s 356ms/step - loss: 83.8274 - accuracy: 0.9983 - val_loss: 79.9604 - val_accuracy: 0.9965
Epoch 00056: val_loss did not improve from 78.59057
Epoch 57/100
15/15 [=====] - 5s 355ms/step - loss: 83.8216 - accuracy: 0.9983 - val_loss: 79.7038 - val_accuracy: 0.9966
Epoch 00057: val_loss did not improve from 78.59057
Epoch 58/100
15/15 [=====] - 5s 356ms/step - loss: 83.8270 - accuracy: 0.9983 - val_loss: 79.4118 - val_accuracy: 0.9968
Epoch 00058: val_loss did not improve from 78.59057
Epoch 59/100
15/15 [=====] - 5s 355ms/step - loss: 83.8281 - accuracy: 0.9982 - val_loss: 79.6292 - val_accuracy: 0.9965
Epoch 00059: val_loss did not improve from 78.59057

```

```

Epoch 60/100
15/15 [=====] - 5s 354ms/step - loss: 83.8302 - accuracy: 0.9984 - val_loss: 79.5238 - val_accuracy: 0.9967

Epoch 00060: val_loss did not improve from 78.59057
Epoch 61/100
15/15 [=====] - 5s 356ms/step - loss: 83.8153 - accuracy: 0.9984 - val_loss: 79.6705 - val_accuracy: 0.9966

Epoch 00061: val_loss did not improve from 78.59057
Epoch 62/100
15/15 [=====] - 5s 358ms/step - loss: 83.8090 - accuracy: 0.9984 - val_loss: 79.4675 - val_accuracy: 0.9968

Epoch 00062: val_loss did not improve from 78.59057
Epoch 63/100
15/15 [=====] - 5s 355ms/step - loss: 83.8188 - accuracy: 0.9983 - val_loss: 79.5662 - val_accuracy: 0.9967

Epoch 00063: val_loss did not improve from 78.59057
Epoch 64/100
15/15 [=====] - 5s 356ms/step - loss: 83.8045 - accuracy: 0.9984 - val_loss: 79.3554 - val_accuracy: 0.9970

Epoch 00064: val_loss did not improve from 78.59057
Epoch 65/100
15/15 [=====] - 5s 354ms/step - loss: 83.8095 - accuracy: 0.9982 - val_loss: 80.1636 - val_accuracy: 0.9963

Epoch 00065: val_loss did not improve from 78.59057
Epoch 66/100
15/15 [=====] - 5s 355ms/step - loss: 83.8215 - accuracy: 0.9983 - val_loss: 80.7996 - val_accuracy: 0.9958

Epoch 00066: val_loss did not improve from 78.59057
Epoch 67/100
15/15 [=====] - 5s 354ms/step - loss: 83.8032 - accuracy: 0.9984 - val_loss: 79.5303 - val_accuracy: 0.9968

Epoch 00067: val_loss did not improve from 78.59057
Epoch 68/100
15/15 [=====] - 5s 355ms/step - loss: 83.7929 - accuracy: 0.9983 - val_loss: 80.0302 - val_accuracy: 0.9964

Epoch 00068: val_loss did not improve from 78.59057
Epoch 69/100
15/15 [=====] - 5s 355ms/step - loss: 83.8140 - accuracy: 0.9984 - val_loss: 79.5726 - val_accuracy: 0.9966

Epoch 00069: val_loss did not improve from 78.59057
Epoch 70/100
15/15 [=====] - 5s 357ms/step - loss: 83.7957 - accuracy: 0.9984 - val_loss: 79.5153 - val_accuracy: 0.9969

Epoch 00070: val_loss did not improve from 78.59057
Epoch 71/100
15/15 [=====] - 5s 355ms/step - loss: 83.8127 - accuracy: 0.9985 - val_loss: 79.3213 - val_accuracy: 0.9968

Epoch 00071: val_loss did not improve from 78.59057
Epoch 72/100
15/15 [=====] - 5s 357ms/step - loss: 83.8029 - accuracy: 0.9984 - val_loss: 79.5981 - val_accuracy: 0.9966

Epoch 00072: val_loss did not improve from 78.59057
Epoch 73/100
15/15 [=====] - 5s 355ms/step - loss: 83.7777 - accuracy: 0.9984 - val_loss: 80.1581 - val_accuracy: 0.9965

Epoch 00073: val_loss did not improve from 78.59057
Epoch 74/100
15/15 [=====] - 5s 355ms/step - loss: 83.7837 - accuracy: 0.9985 - val_loss: 79.5069 - val_accuracy: 0.9967

Epoch 00074: val_loss did not improve from 78.59057
Epoch 75/100
15/15 [=====] - 5s 353ms/step - loss: 83.7709 - accuracy: 0.9985 - val_loss: 79.6151 - val_accuracy: 0.9969

Epoch 00075: val_loss did not improve from 78.59057
Epoch 76/100
15/15 [=====] - 5s 353ms/step - loss: 83.7806 - accuracy: 0.9984 - val_loss: 80.5009 - val_accuracy: 0.9960

Epoch 00076: val_loss did not improve from 78.59057
Epoch 77/100
15/15 [=====] - 5s 354ms/step - loss: 83.7710 - accuracy: 0.9985 - val_loss: 81.2718 - val_accuracy: 0.9959

Epoch 00077: val_loss did not improve from 78.59057
Epoch 78/100
15/15 [=====] - 5s 355ms/step - loss: 83.7763 - accuracy: 0.9985 - val_loss: 79.7215 - val_accuracy: 0.9967

Epoch 00078: val_loss did not improve from 78.59057
Epoch 79/100
15/15 [=====] - 5s 355ms/step - loss: 83.7656 - accuracy: 0.9985 - val_loss: 80.4212 - val_accuracy: 0.9962

```

```

Epoch 00079: val_loss did not improve from 78.59057
Epoch 80/100
15/15 [=====] - 5s 355ms/step - loss: 83.7578 - accuracy: 0.9985 - val_loss: 79.6546 - val_accuracy: 0.9969

Epoch 00080: val_loss did not improve from 78.59057
Epoch 81/100
15/15 [=====] - 5s 357ms/step - loss: 83.7501 - accuracy: 0.9986 - val_loss: 80.6850 - val_accuracy: 0.9962

Epoch 00081: val_loss did not improve from 78.59057
Epoch 82/100
15/15 [=====] - 5s 355ms/step - loss: 83.7528 - accuracy: 0.9985 - val_loss: 79.7918 - val_accuracy: 0.9968

Epoch 00082: val_loss did not improve from 78.59057
Epoch 83/100
15/15 [=====] - 5s 353ms/step - loss: 83.7457 - accuracy: 0.9984 - val_loss: 79.8666 - val_accuracy: 0.9967

Epoch 00083: val_loss did not improve from 78.59057
Epoch 84/100
15/15 [=====] - 5s 353ms/step - loss: 83.7485 - accuracy: 0.9985 - val_loss: 79.7833 - val_accuracy: 0.9968

Epoch 00084: val_loss did not improve from 78.59057
Epoch 85/100
15/15 [=====] - 5s 355ms/step - loss: 83.7547 - accuracy: 0.9985 - val_loss: 79.2514 - val_accuracy: 0.9972

Epoch 00085: val_loss did not improve from 78.59057
Epoch 86/100
15/15 [=====] - 5s 355ms/step - loss: 83.8061 - accuracy: 0.9984 - val_loss: 79.9126 - val_accuracy: 0.9966

Epoch 00086: val_loss did not improve from 78.59057
Epoch 87/100
15/15 [=====] - 5s 356ms/step - loss: 83.7800 - accuracy: 0.9985 - val_loss: 80.3349 - val_accuracy: 0.9962

Epoch 00087: val_loss did not improve from 78.59057
Epoch 88/100
15/15 [=====] - 5s 356ms/step - loss: 83.7711 - accuracy: 0.9984 - val_loss: 80.1269 - val_accuracy: 0.9965

Epoch 00088: val_loss did not improve from 78.59057
Epoch 89/100
15/15 [=====] - 5s 357ms/step - loss: 83.7585 - accuracy: 0.9985 - val_loss: 79.1890 - val_accuracy: 0.9972

Epoch 00089: val_loss did not improve from 78.59057
Epoch 90/100
15/15 [=====] - 5s 356ms/step - loss: 83.7432 - accuracy: 0.9985 - val_loss: 79.5272 - val_accuracy: 0.9971

Epoch 00090: val_loss did not improve from 78.59057
Epoch 91/100
15/15 [=====] - 5s 358ms/step - loss: 83.7378 - accuracy: 0.9986 - val_loss: 79.8400 - val_accuracy: 0.9968

Epoch 00091: val_loss did not improve from 78.59057
Epoch 92/100
15/15 [=====] - 5s 356ms/step - loss: 83.7284 - accuracy: 0.9986 - val_loss: 79.7670 - val_accuracy: 0.9969

Epoch 00092: val_loss did not improve from 78.59057
Epoch 93/100
15/15 [=====] - 5s 355ms/step - loss: 83.7371 - accuracy: 0.9985 - val_loss: 79.8618 - val_accuracy: 0.9968

Epoch 00093: val_loss did not improve from 78.59057
Epoch 94/100
15/15 [=====] - 5s 355ms/step - loss: 83.7315 - accuracy: 0.9986 - val_loss: 79.4534 - val_accuracy: 0.9971

Epoch 00094: val_loss did not improve from 78.59057
Epoch 95/100
15/15 [=====] - 5s 355ms/step - loss: 83.7407 - accuracy: 0.9985 - val_loss: 81.2738 - val_accuracy: 0.9960

Epoch 00095: val_loss did not improve from 78.59057
Epoch 96/100
15/15 [=====] - 5s 356ms/step - loss: 83.7238 - accuracy: 0.9985 - val_loss: 80.1795 - val_accuracy: 0.9967

Epoch 00096: val_loss did not improve from 78.59057
Epoch 97/100
15/15 [=====] - 5s 356ms/step - loss: 83.7176 - accuracy: 0.9986 - val_loss: 79.5438 - val_accuracy: 0.9970

Epoch 00097: val_loss did not improve from 78.59057
Epoch 98/100
15/15 [=====] - 5s 355ms/step - loss: 83.7346 - accuracy: 0.9984 - val_loss: 79.7012 - val_accuracy: 0.9969

Epoch 00098: val_loss did not improve from 78.59057
Epoch 99/100

```

```

Epoch 00000: val_loss did not improve from 78.59057
Epoch 100/100
15/15 [=====] - 5s 356ms/step - loss: 83.7231 - accuracy: 0.9985 - val_loss: 80.1409 - val_accuracy: 0.9966

Epoch 00099: val_loss did not improve from 78.59057
Epoch 100/100
15/15 [=====] - 5s 361ms/step - loss: 83.7151 - accuracy: 0.9986 - val_loss: 80.0497 - val_accuracy: 0.9967

Epoch 00100: val_loss did not improve from 78.59057

In [20]:
#checkpoint
checkpoint=tf.keras.callbacks.ModelCheckpoint("./content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skripsi.h5",
verbose=1, save_best_only=True)

#Callbacks
Callbacks = [
    # EarlyStopping(monitor="val_accuracy", min_delta=0, patience=25, verbose=1, mode='max', baseline=None, restore_best_weights=True),
    # tf.keras.callbacks.TensorBoard(log_dir='log'),
    checkpoint
]
# Train the model
result16 = model.fit(x_train, y_train,batch_size=16,epochs=100,verbose=1,validation_data=(x_val, y_val),
validation_split=0.5,callbacks=Callbacks)

Epoch 1/100
8/8 [=====] - 9s 747ms/step - loss: 83.7048 - accuracy: 0.9986 - val_loss: 80.0497 - val_accuracy: 0.9968

Epoch 00001: val_loss improved from inf to 80.04973, saving model to ./content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skripsi.h5
Epoch 2/100
8/8 [=====] - 5s 659ms/step - loss: 83.7099 - accuracy: 0.9986 - val_loss: 79.7054 - val_accuracy: 0.9970

Epoch 00002: val_loss improved from 80.04973 to 79.70540, saving model to ./content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skripsi.h5
Epoch 3/100
8/8 [=====] - 5s 659ms/step - loss: 83.7079 - accuracy: 0.9986 - val_loss: 79.9670 - val_accuracy: 0.9968

Epoch 00003: val_loss did not improve from 79.70540
Epoch 4/100
8/8 [=====] - 5s 657ms/step - loss: 83.6930 - accuracy: 0.9986 - val_loss: 80.3329 - val_accuracy: 0.9966

Epoch 00004: val_loss did not improve from 79.70540
Epoch 5/100
8/8 [=====] - 5s 660ms/step - loss: 83.6913 - accuracy: 0.9987 - val_loss: 80.5178 - val_accuracy: 0.9966

Epoch 00005: val_loss did not improve from 79.70540
Epoch 6/100
8/8 [=====] - 5s 660ms/step - loss: 83.6917 - accuracy: 0.9986 - val_loss: 79.9813 - val_accuracy: 0.9968

Epoch 00006: val_loss did not improve from 79.70540
Epoch 7/100
8/8 [=====] - 5s 650ms/step - loss: 83.6972 - accuracy: 0.9986 - val_loss: 80.1521 - val_accuracy: 0.9966

Epoch 00007: val_loss did not improve from 79.70540
Epoch 8/100
8/8 [=====] - 5s 658ms/step - loss: 83.6823 - accuracy: 0.9987 - val_loss: 79.8225 - val_accuracy: 0.9969

Epoch 00008: val_loss did not improve from 79.70540
Epoch 9/100
8/8 [=====] - 5s 654ms/step - loss: 83.6810 - accuracy: 0.9987 - val_loss: 79.9814 - val_accuracy: 0.9969

Epoch 00009: val_loss did not improve from 79.70540
Epoch 10/100
8/8 [=====] - 5s 656ms/step - loss: 83.6795 - accuracy: 0.9986 - val_loss: 79.9144 - val_accuracy: 0.9969

Epoch 00010: val_loss did not improve from 79.70540
Epoch 11/100
8/8 [=====] - 5s 659ms/step - loss: 83.6733 - accuracy: 0.9986 - val_loss: 80.4111 - val_accuracy: 0.9966

Epoch 00011: val_loss did not improve from 79.70540
Epoch 12/100
8/8 [=====] - 5s 657ms/step - loss: 83.6679 - accuracy: 0.9987 - val_loss: 80.2219 - val_accuracy: 0.9968

Epoch 00012: val_loss did not improve from 79.70540
Epoch 13/100
8/8 [=====] - 5s 654ms/step - loss: 83.6680 - accuracy: 0.9987 - val_loss: 80.4050 - val_accuracy: 0.9966

Epoch 00013: val_loss did not improve from 79.70540
Epoch 14/100
8/8 [=====] - 5s 658ms/step - loss: 83.6859 - accuracy: 0.9986 - val_loss: 80.5011 - val_accuracy: 0.9965

```

```

Epoch 00014: val_lcss did not improve from 79.70540
Epoch 15/100
8/8 [=====] - 5s 658ms/step - loss: 83.6961 - accuracy: 0.9986 - val_loss: 80.5032 - val_accuracy: 0.9966

Epoch 00015: val_loss did not improve from 79.70540
Epoch 16/100
8/8 [=====] - 5s 660ms/step - loss: 83.6700 - accuracy: 0.9986 - val_loss: 80.6122 - val_accuracy: 0.9965

Epoch 00016: val_lcss did not improve from 79.70540
Epoch 17/100
8/8 [=====] - 5s 659ms/step - loss: 83.6742 - accuracy: 0.9986 - val_loss: 80.8999 - val_accuracy: 0.9965

Epoch 00017: val_lcss did not improve from 79.70540
Epoch 18/100
8/8 [=====] - 5s 658ms/step - loss: 83.6728 - accuracy: 0.9986 - val_loss: 80.3943 - val_accuracy: 0.9967

Epoch 00018: val_loss did not improve from 79.70540
Epoch 19/100
8/8 [=====] - 5s 658ms/step - loss: 83.6611 - accuracy: 0.9986 - val_loss: 80.6245 - val_accuracy: 0.9966

Epoch 00019: val_loss did not improve from 79.70540
Epoch 20/100
8/8 [=====] - 5s 655ms/step - loss: 83.6672 - accuracy: 0.9987 - val_loss: 80.6215 - val_accuracy: 0.9965

Epoch 00020: val_loss did not improve from 79.70540
Epoch 21/100
8/8 [=====] - 5s 661ms/step - loss: 83.6680 - accuracy: 0.9985 - val_loss: 80.0826 - val_accuracy: 0.9968

Epoch 00021: val_lcss did not improve from 79.70540
Epoch 22/100
8/8 [=====] - 5s 660ms/step - loss: 83.6692 - accuracy: 0.9986 - val_loss: 80.1835 - val_accuracy: 0.9968

Epoch 00022: val_lcss did not improve from 79.70540
Epoch 23/100
8/8 [=====] - 5s 654ms/step - loss: 83.6556 - accuracy: 0.9987 - val_loss: 80.1512 - val_accuracy: 0.9968

Epoch 00023: val_loss did not improve from 79.70540
Epoch 24/100
8/8 [=====] - 5s 658ms/step - loss: 83.6575 - accuracy: 0.9986 - val_loss: 80.5467 - val_accuracy: 0.9966

Epoch 00024: val_loss did not improve from 79.70540
Epoch 25/100
8/8 [=====] - 5s 660ms/step - loss: 83.6628 - accuracy: 0.9987 - val_loss: 80.5265 - val_accuracy: 0.9965

Epoch 00025: val_loss did not improve from 79.70540
Epoch 26/100
8/8 [=====] - 5s 659ms/step - loss: 83.6570 - accuracy: 0.9987 - val_loss: 80.2630 - val_accuracy: 0.9968

Epoch 00026: val_loss did not improve from 79.70540
Epoch 27/100
8/8 [=====] - 5s 657ms/step - loss: 83.6550 - accuracy: 0.9987 - val_loss: 79.8626 - val_accuracy: 0.9970

Epoch 00027: val_loss did not improve from 79.70540
Epoch 28/100
8/8 [=====] - 5s 658ms/step - loss: 83.6451 - accuracy: 0.9987 - val_loss: 79.7044 - val_accuracy: 0.9971

Epoch 00028: val_lcss improved from 79.70540 to 79.70444, saving model to ..../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skrripsi.h5
Epoch 29/100
8/8 [=====] - 5s 658ms/step - loss: 83.6374 - accuracy: 0.9987 - val_loss: 80.0996 - val_accuracy: 0.9969

Epoch 00029: val_loss did not improve from 79.70444
Epoch 30/100
8/8 [=====] - 5s 655ms/step - loss: 83.6460 - accuracy: 0.9987 - val_loss: 80.3485 - val_accuracy: 0.9967

Epoch 00030: val_loss did not improve from 79.70444
Epoch 31/100
8/8 [=====] - 5s 653ms/step - loss: 83.6396 - accuracy: 0.9987 - val_loss: 80.8475 - val_accuracy: 0.9964

Epoch 00031: val_lcss did not improve from 79.70444
Epoch 32/100
8/8 [=====] - 5s 657ms/step - loss: 83.6422 - accuracy: 0.9987 - val_loss: 80.5732 - val_accuracy: 0.9966

Epoch 00032: val_loss did not improve from 79.70444
Epoch 33/100
8/8 [=====] - 5s 658ms/step - loss: 83.6584 - accuracy: 0.9986 - val_loss: 80.0499 - val_accuracy: 0.9969

Epoch 00033: val_lcss did not improve from 79.70444

```

```

Epoch 34/100
8/8 [=====] - 5s 665ms/step - loss: 83.6500 - accuracy: 0.9987 - val_loss: 80.5338 - val_accuracy: 0.9966

Epoch 00034: val_loss did not improve from 79.70444
Epoch 35/100
8/8 [=====] - 5s 658ms/step - loss: 83.6403 - accuracy: 0.9988 - val_loss: 80.7682 - val_accuracy: 0.9965

Epoch 00035: val_loss did not improve from 79.70444
Epoch 36/100
8/8 [=====] - 5s 658ms/step - loss: 83.6473 - accuracy: 0.9987 - val_loss: 79.7421 - val_accuracy: 0.9970

Epoch 00036: val_loss did not improve from 79.70444
Epoch 37/100
8/8 [=====] - 5s 655ms/step - loss: 83.6490 - accuracy: 0.9987 - val_loss: 79.8520 - val_accuracy: 0.9970

Epoch 00037: val_loss did not improve from 79.70444
Epoch 38/100
8/8 [=====] - 5s 661ms/step - loss: 83.6424 - accuracy: 0.9987 - val_loss: 80.5343 - val_accuracy: 0.9967

Epoch 00038: val_loss did not improve from 79.70444
Epoch 39/100
8/8 [=====] - 5s 660ms/step - loss: 83.6398 - accuracy: 0.9986 - val_loss: 80.1426 - val_accuracy: 0.9968

Epoch 00039: val_loss did not improve from 79.70444
Epoch 40/100
8/8 [=====] - 5s 654ms/step - loss: 83.6385 - accuracy: 0.9986 - val_loss: 79.6734 - val_accuracy: 0.9971

Epoch 00040: val_loss improved from 79.70444 to 79.67337, saving model to .../content/drive/My Drive/Colab Notebooks/Dataset/Embric/Skripsi.h5
Epoch 41/100
8/8 [=====] - 5s 660ms/step - loss: 83.6421 - accuracy: 0.9988 - val_loss: 79.8540 - val_accuracy: 0.9971

Epoch 00041: val_loss did not improve from 79.67337
Epoch 42/100
8/8 [=====] - 5s 655ms/step - loss: 83.6312 - accuracy: 0.9987 - val_loss: 80.3979 - val_accuracy: 0.9968

Epoch 00042: val_loss did not improve from 79.67337
Epoch 43/100
8/8 [=====] - 5s 659ms/step - loss: 83.6374 - accuracy: 0.9987 - val_loss: 80.3098 - val_accuracy: 0.9968

Epoch 00043: val_loss did not improve from 79.67337
Epoch 44/100
8/8 [=====] - 5s 658ms/step - loss: 83.6265 - accuracy: 0.9988 - val_loss: 79.8579 - val_accuracy: 0.9970

Epoch 00044: val_loss did not improve from 79.67337
Epoch 45/100
8/8 [=====] - 5s 653ms/step - loss: 83.6603 - accuracy: 0.9986 - val_loss: 79.8527 - val_accuracy: 0.9970

Epoch 00045: val_loss did not improve from 79.67337
Epoch 46/100
8/8 [=====] - 5s 655ms/step - loss: 83.6498 - accuracy: 0.9986 - val_loss: 80.5511 - val_accuracy: 0.9967

Epoch 00046: val_loss did not improve from 79.67337
Epoch 47/100
8/8 [=====] - 5s 656ms/step - loss: 83.6396 - accuracy: 0.9987 - val_loss: 80.6359 - val_accuracy: 0.9966

Epoch 00047: val_loss did not improve from 79.67337
Epoch 48/100
8/8 [=====] - 5s 661ms/step - loss: 83.6462 - accuracy: 0.9986 - val_loss: 80.2256 - val_accuracy: 0.9967

Epoch 00048: val_loss did not improve from 79.67337
Epoch 49/100
8/8 [=====] - 5s 659ms/step - loss: 83.6317 - accuracy: 0.9987 - val_loss: 80.4908 - val_accuracy: 0.9967

Epoch 00049: val_loss did not improve from 79.67337
Epoch 50/100
8/8 [=====] - 5s 662ms/step - loss: 83.6261 - accuracy: 0.9986 - val_loss: 80.2281 - val_accuracy: 0.9969

Epoch 00050: val_loss did not improve from 79.67337
Epoch 51/100
8/8 [=====] - 5s 657ms/step - loss: 83.6167 - accuracy: 0.9987 - val_loss: 80.2350 - val_accuracy: 0.9969

Epoch 00051: val_loss did not improve from 79.67337
Epoch 52/100
8/8 [=====] - 5s 658ms/step - loss: 83.6300 - accuracy: 0.9986 - val_loss: 80.3980 - val_accuracy: 0.9968

Epoch 00052: val_loss did not improve from 79.67337
Epoch 53/100
8/8 [=====] - 5s 659ms/step - loss: 83.6262 - accuracy: 0.9986 - val_loss: 80.3354 - val_accuracy: 0.9968

```

```

racy: 0.9968

Epoch 00053: val_loss did not improve from 79.67337
Epoch 54/100
8/8 [=====] - 5s 680ms/step - loss: 83.6260 - accuracy: 0.9986 - val_loss: 80.2767 - val_accuracy: 0.9969

Epoch 00054: val_loss did not improve from 79.67337
Epoch 55/100
8/8 [=====] - 5s 659ms/step - loss: 83.6390 - accuracy: 0.9986 - val_loss: 80.2375 - val_accuracy: 0.9968

Epoch 00055: val_loss did not improve from 79.67337
Epoch 56/100
8/8 [=====] - 5s 662ms/step - loss: 83.6271 - accuracy: 0.9986 - val_loss: 80.2048 - val_accuracy: 0.9968

Epoch 00056: val_loss did not improve from 79.67337
Epoch 57/100
8/8 [=====] - 5s 663ms/step - loss: 83.6342 - accuracy: 0.9986 - val_loss: 80.2986 - val_accuracy: 0.9967

Epoch 00057: val_loss did not improve from 79.67337
Epoch 58/100
8/8 [=====] - 5s 660ms/step - loss: 83.6320 - accuracy: 0.9986 - val_loss: 80.0371 - val_accuracy: 0.9969

Epoch 00058: val_loss did not improve from 79.67337
Epoch 59/100
8/8 [=====] - 5s 658ms/step - loss: 83.6405 - accuracy: 0.9986 - val_loss: 79.7384 - val_accuracy: 0.9969

Epoch 00059: val_loss did not improve from 79.67337
Epoch 60/100
8/8 [=====] - 5s 664ms/step - loss: 83.6467 - accuracy: 0.9987 - val_loss: 80.4374 - val_accuracy: 0.9968

Epoch 00060: val_loss did not improve from 79.67337
Epoch 61/100
8/8 [=====] - 5s 660ms/step - loss: 83.6369 - accuracy: 0.9987 - val_loss: 80.6925 - val_accuracy: 0.9966

Epoch 00061: val_loss did not improve from 79.67337
Epoch 62/100
8/8 [=====] - 5s 657ms/step - loss: 83.6406 - accuracy: 0.9987 - val_loss: 80.7574 - val_accuracy: 0.9966

Epoch 00062: val_loss did not improve from 79.67337
Epoch 63/100
8/8 [=====] - 5s 659ms/step - loss: 83.6344 - accuracy: 0.9987 - val_loss: 80.0190 - val_accuracy: 0.9969

Epoch 00063: val_loss did not improve from 79.67337
Epoch 64/100
8/8 [=====] - 5s 657ms/step - loss: 83.6327 - accuracy: 0.9987 - val_loss: 79.8622 - val_accuracy: 0.9970

Epoch 00064: val_loss did not improve from 79.67337
Epoch 65/100
8/8 [=====] - 5s 660ms/step - loss: 83.6204 - accuracy: 0.9986 - val_loss: 80.1516 - val_accuracy: 0.9969

Epoch 00065: val_loss did not improve from 79.67337
Epoch 66/100
8/8 [=====] - 5s 660ms/step - loss: 83.6175 - accuracy: 0.9987 - val_loss: 80.0623 - val_accuracy: 0.9969

Epoch 00066: val_loss did not improve from 79.67337
Epoch 67/100
8/8 [=====] - 5s 662ms/step - loss: 83.6088 - accuracy: 0.9987 - val_loss: 80.1002 - val_accuracy: 0.9970

Epoch 00067: val_loss did not improve from 79.67337
Epoch 68/100
8/8 [=====] - 5s 657ms/step - loss: 83.6149 - accuracy: 0.9986 - val_loss: 80.2794 - val_accuracy: 0.9968

Epoch 00068: val_loss did not improve from 79.67337
Epoch 69/100
8/8 [=====] - 5s 659ms/step - loss: 83.6016 - accuracy: 0.9987 - val_loss: 80.5203 - val_accuracy: 0.9967

Epoch 00069: val_loss did not improve from 79.67337
Epoch 70/100
8/8 [=====] - 5s 660ms/step - loss: 83.6096 - accuracy: 0.9987 - val_loss: 81.2473 - val_accuracy: 0.9964

Epoch 00070: val_loss did not improve from 79.67337
Epoch 71/100
8/8 [=====] - 5s 657ms/step - loss: 83.6142 - accuracy: 0.9986 - val_loss: 80.2571 - val_accuracy: 0.9968

Epoch 00071: val_loss did not improve from 79.67337
Epoch 72/100
8/8 [=====] - 5s 653ms/step - loss: 83.6121 - accuracy: 0.9988 - val_loss: 79.9527 - val_accuracy: 0.9970

Epoch 00072: val_loss did not improve from 79.67337

```

```

Epoch 00072: val_loss did not improve from 79.25467
Epoch 73/100
8/8 [=====] - 5s 660ms/step - loss: 83.6149 - accuracy: 0.9987 - val_loss: 79.2547 - val_accuracy: 0.9973
Epoch 00073: val_loss improved from 79.67337 to 79.25467, saving model to ../content/drive/My Drive/Colab Notebooks/Dataset/Embrio/Skrapsi.h5
Epoch 74/100
8/8 [=====] - 5s 662ms/step - loss: 83.6439 - accuracy: 0.9986 - val_loss: 79.4196 - val_accuracy: 0.9972
Epoch 00074: val_loss did not improve from 79.25467
Epoch 75/100
8/8 [=====] - 5s 660ms/step - loss: 83.6226 - accuracy: 0.9987 - val_loss: 80.1791 - val_accuracy: 0.9969
Epoch 00075: val_loss did not improve from 79.25467
Epoch 76/100
8/8 [=====] - 5s 661ms/step - loss: 83.6150 - accuracy: 0.9988 - val_loss: 80.1871 - val_accuracy: 0.9969
Epoch 00076: val_loss did not improve from 79.25467
Epoch 77/100
8/8 [=====] - 5s 657ms/step - loss: 83.6054 - accuracy: 0.9988 - val_loss: 80.3673 - val_accuracy: 0.9968
Epoch 00077: val_loss did not improve from 79.25467
Epoch 78/100
8/8 [=====] - 5s 660ms/step - loss: 83.6147 - accuracy: 0.9987 - val_loss: 80.4176 - val_accuracy: 0.9967
Epoch 00078: val_loss did not improve from 79.25467
Epoch 79/100
8/8 [=====] - 5s 661ms/step - loss: 83.6173 - accuracy: 0.9986 - val_loss: 80.7698 - val_accuracy: 0.9965
Epoch 00079: val_loss did not improve from 79.25467
Epoch 80/100
8/8 [=====] - 5s 659ms/step - loss: 83.6046 - accuracy: 0.9987 - val_loss: 80.9373 - val_accuracy: 0.9965
Epoch 00080: val_loss did not improve from 79.25467
Epoch 81/100
8/8 [=====] - 5s 658ms/step - loss: 83.6040 - accuracy: 0.9987 - val_loss: 80.7545 - val_accuracy: 0.9967
Epoch 00081: val_loss did not improve from 79.25467
Epoch 82/100
8/8 [=====] - 5s 658ms/step - loss: 83.6107 - accuracy: 0.9988 - val_loss: 80.7184 - val_accuracy: 0.9967
Epoch 00082: val_loss did not improve from 79.25467
Epoch 83/100
8/8 [=====] - 5s 660ms/step - loss: 83.6032 - accuracy: 0.9987 - val_loss: 80.0518 - val_accuracy: 0.9970
Epoch 00083: val_loss did not improve from 79.25467
Epoch 84/100
8/8 [=====] - 5s 659ms/step - loss: 83.5941 - accuracy: 0.9988 - val_loss: 80.7044 - val_accuracy: 0.9966
Epoch 00084: val_loss did not improve from 79.25467
Epoch 85/100
8/8 [=====] - 5s 661ms/step - loss: 83.5987 - accuracy: 0.9988 - val_loss: 80.4358 - val_accuracy: 0.9968
Epoch 00085: val_loss did not improve from 79.25467
Epoch 86/100
8/8 [=====] - 5s 658ms/step - loss: 83.5946 - accuracy: 0.9987 - val_loss: 81.0536 - val_accuracy: 0.9965
Epoch 00086: val_loss did not improve from 79.25467
Epoch 87/100
8/8 [=====] - 5s 654ms/step - loss: 83.5850 - accuracy: 0.9988 - val_loss: 80.9964 - val_accuracy: 0.9965
Epoch 00087: val_loss did not improve from 79.25467
Epoch 88/100
8/8 [=====] - 5s 659ms/step - loss: 83.5843 - accuracy: 0.9988 - val_loss: 80.4733 - val_accuracy: 0.9968
Epoch 00088: val_loss did not improve from 79.25467
Epoch 89/100
8/8 [=====] - 5s 661ms/step - loss: 83.5820 - accuracy: 0.9987 - val_loss: 80.2215 - val_accuracy: 0.9969
Epoch 00089: val_loss did not improve from 79.25467
Epoch 90/100
8/8 [=====] - 5s 656ms/step - loss: 83.5882 - accuracy: 0.9987 - val_loss: 80.0282 - val_accuracy: 0.9970
Epoch 00090: val_loss did not improve from 79.25467
Epoch 91/100
8/8 [=====] - 5s 661ms/step - loss: 83.5981 - accuracy: 0.9986 - val_loss: 80.5709 - val_accuracy: 0.9967
Epoch 00091: val_loss did not improve from 79.25467
Epoch 92/100

```

```

8/8 [=====] - 5s 662ms/step - loss: 83.5991 - accuracy: 0.9988 - val_loss: 79.9464 - val_accuracy: 0.9971
Epoch 00092: val_loss did not improve from 79.25467
Epoch 93/100
8/8 [=====] - 5s 661ms/step - loss: 83.5884 - accuracy: 0.9987 - val_loss: 79.6531 - val_accuracy: 0.9972
Epoch 00093: val_loss did not improve from 79.25467
Epoch 94/100
8/8 [=====] - 5s 660ms/step - loss: 83.5763 - accuracy: 0.9987 - val_loss: 80.1519 - val_accuracy: 0.9969
Epoch 00094: val_loss did not improve from 79.25467
Epoch 95/100
8/8 [=====] - 5s 660ms/step - loss: 83.5791 - accuracy: 0.9988 - val_loss: 80.6826 - val_accuracy: 0.9967
Epoch 00095: val_loss did not improve from 79.25467
Epoch 96/100
8/8 [=====] - 5s 661ms/step - loss: 83.5900 - accuracy: 0.9987 - val_loss: 80.6221 - val_accuracy: 0.9967
Epoch 00096: val_loss did not improve from 79.25467
Epoch 97/100
8/8 [=====] - 5s 662ms/step - loss: 83.6140 - accuracy: 0.9987 - val_loss: 80.5195 - val_accuracy: 0.9966
Epoch 00097: val_loss did not improve from 79.25467
Epoch 98/100
8/8 [=====] - 5s 659ms/step - loss: 83.6097 - accuracy: 0.9987 - val_loss: 80.7863 - val_accuracy: 0.9966
Epoch 00098: val_loss did not improve from 79.25467
Epoch 99/100
8/8 [=====] - 5s 659ms/step - loss: 83.5910 - accuracy: 0.9987 - val_loss: 80.7952 - val_accuracy: 0.9967
Epoch 00099: val_loss did not improve from 79.25467
Epoch 100/100
8/8 [=====] - 5s 657ms/step - loss: 83.5769 - accuracy: 0.9987 - val_loss: 80.3616 - val_accuracy: 0.9969
Epoch 00100: val_loss did not improve from 79.25467

```

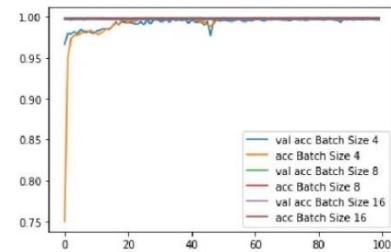
Grafik Akurasi dan Loss pada saat Training Model

In [21]:

```

plt.plot(result4.history['val_accuracy'], label="val acc Batch Size 4")
plt.plot(result4.history['accuracy'], label="acc Batch Size 4")
plt.plot(result8.history['val_accuracy'], label="val acc Batch Size 8")
plt.plot(result8.history['accuracy'], label="acc Batch Size 8")
plt.plot(result16.history['val_accuracy'], label="val acc Batch Size 16")
plt.plot(result16.history['accuracy'], label="acc Batch Size 16")
plt.legend()
plt.show()

```

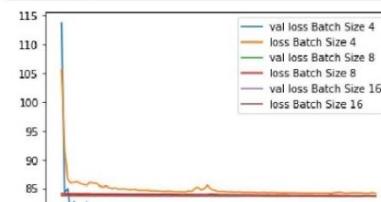


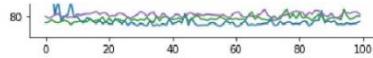
In [22]:

```

plt.plot(result4.history['val_loss'], label="val loss Batch Size 4")
plt.plot(result4.history['loss'], label="loss Batch Size 4")
plt.plot(result8.history['val_loss'], label="val loss Batch Size 8")
plt.plot(result8.history['loss'], label="loss Batch Size 8")
plt.plot(result16.history['val_loss'], label="val loss Batch Size 16")
plt.plot(result16.history['loss'], label="loss Batch Size 16")
plt.legend()
plt.show()

```





```
In [ ]:
path_image_test = [os.path.splitext(os.path.basename(x))[0]: x
                   for x in glob(os.path.join(path, "Embryo", "*"))]
df['path image test'] = df['id'].map(path_image_test.get)
```

```
In [ ]:
x_test = []
segmented = []
for i in range(len(df)):
    imgX = cv2.imread(df['path image test'][i])
    imgX = cv2.resize(imgX, (IMG_WIDTH, IMG_HEIGHT))
    x_test.append(imgX)
    segmented.append(imgX)
```

Menampilkan output dari masing-masing layer

```
In [ ]:
layer_outputs = [layer.output for layer in model.layers[1:]]
visual_model = tf.keras.models.Model(inputs = model.input, outputs = layer_outputs)

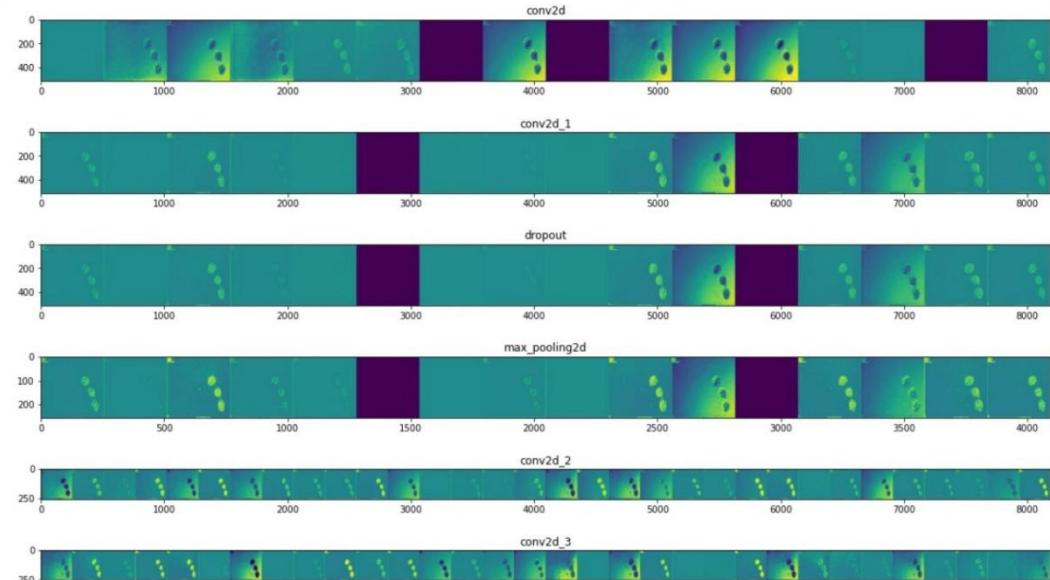
x = x_test[0].copy()
x = x.reshape((1,) + x.shape)

feature_maps = visual_model.predict(x)

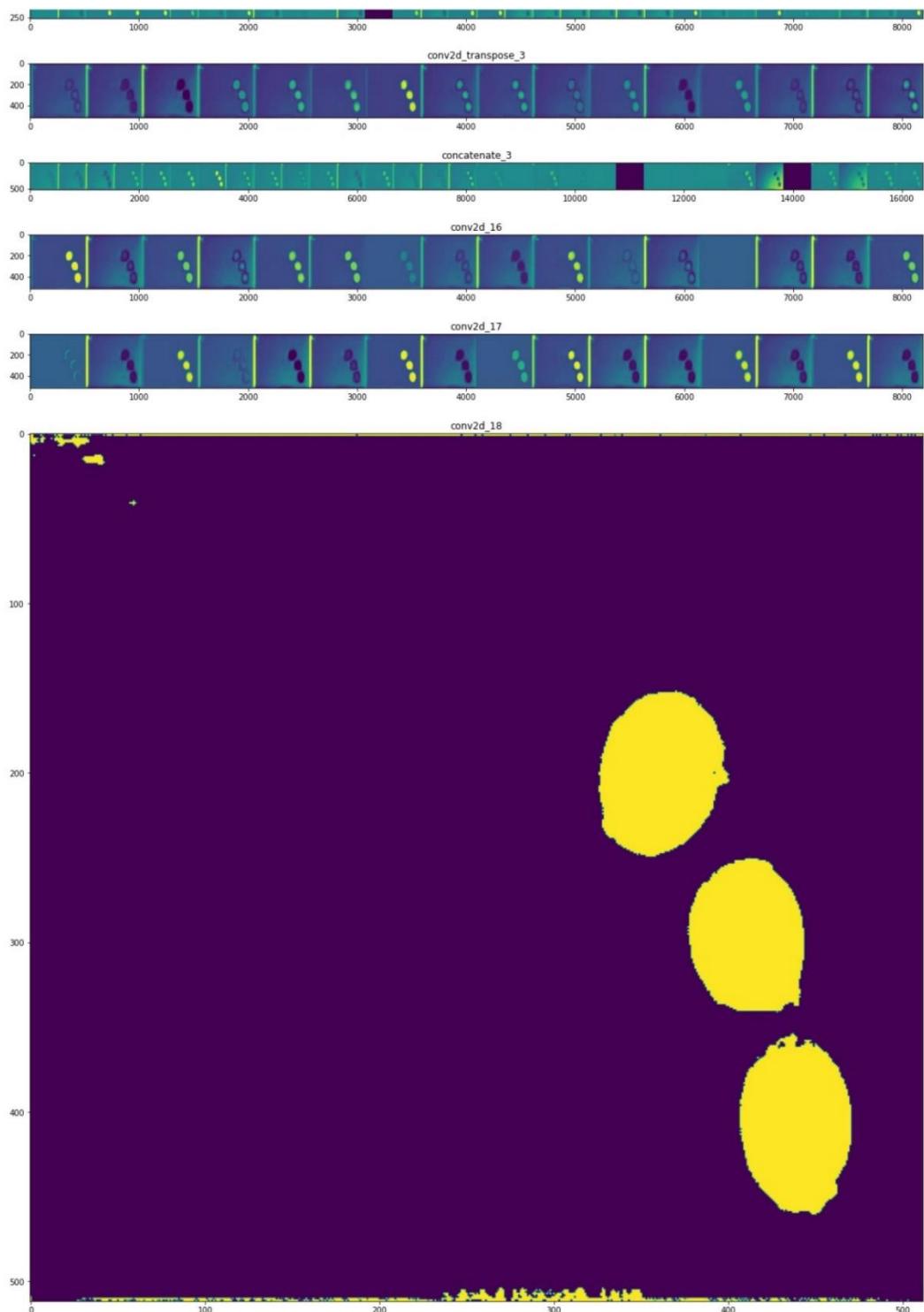
layer_names = [layer.name for layer in model.layers[1:]]

for layer_name, feature_map in zip(layer_names, feature_maps):
    if len(feature_map.shape) == 4:
        n_features = feature_map.shape[-1]
        size = feature_map.shape[1]
        display_grid = np.zeros((size, size * n_features))
        for i in range(n_features):
            x = feature_map[0, :, :, i]
            x -= x.mean()
            x /= x.std()
            x *= 64
            x += 128
            x = np.clip(x, 0, 255).astype('uint8')
            display_grid[:, i * size : (i + 1) * size] = x
        scale = 20. / n_features
        plt.figure(figsize=(scale * n_features, scale))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:20: RuntimeWarning: invalid value encountered in true_div
ide
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:26: RuntimeWarning: More than 20 figures have been opened.
. Figures created through the pyplot interface ('matplotlib.pyplot.figure') are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam 'figure.max_open_warning').

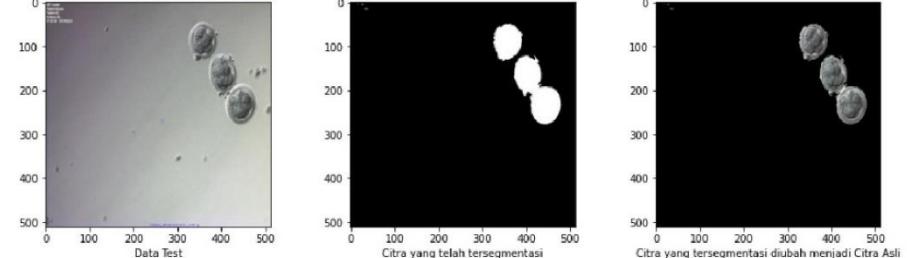






```
In [ ]:  
x_test = np.asarray(x_test)  
x_test = x_test/255
```

Predict Data Test

```
In [ ]:  
predict test = model.predict(x_test,verbose=1)  
11/11 [=====] - 3s 312ms/step  
  
In [ ]:  
predict test t=(predict test>0.5).astype(np.uint8)  
  
In [ ]:  
for i in range(x_test.shape[0]):  
    for j in range(x_test.shape[1]):  
        for k in range(x_test.shape[2]):  
            if predict test_t[i][j,k] < 0.5 :  
                segmented[i][j,k] = predict test t[i][j,k]  
  
In [ ]:  
n= 2  
n=3  
fig_width =5*n  
fig, ax = plt.subplots(1, n)  
fig. set_figwidth(fig_width)  
  
ax[0].imshow(x_test[n])  
ax[0].set_xlabel('Data Test')  
ax[1].imshow(np.squeeze(predict test_t[n]),cmap='gray')  
ax[1].set_xlabel('Citra yang telah tersegmentasi')  
ax[2].imshow(segmented[n],cmap='gray')  
ax[2].set_xlabel('Citra yang tersegmentasi diubah menjadi Citra Asli')  
  
Out[ ]:  
Text(0.5, 0, 'Citra yang tersegmentasi diubah menjadi Citra Asli')  
  

```

Cropping citra yang telah tersegmentasi

```
In [ ]:  
cropped = []  
croppedResize = []  
for n in range(len(x_test)):  
    image = segmented[n].copy()  
    contours, hierarchy = cv2.findContours(predict test_t[n], cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    idx = 0  
    for c in contours:  
        x,y,w,h = cv2.boundingRect(c)  
        if w>50 and h>50:  
            idx+=1  
            new_img=image[y:y+h,x:x+w]  
            new_img = np.asarray(new_img)  
            new_img2 = cv2.resize(new_img,(256,256))  
            new img = cv2.resize(new img,(int(new img.shape[1]*2.75),int(new img.shape[0]*2.0625)))  
            cropped.append(new img)  
            croppedResize.append(new img2)  
  
In [ ]:  
cropped = np.asarray(cropped)  
croppedResize = np.asarray(croppedResize)  
  
/usr/local/lib/python3.6/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray  
    return array(a, dtype, copy=False, order=order)  
  
In [ ]:  
listDelete = []  
for i in range(len(cropped)):  
    if (cropped[i].shape[0] < 256) and (cropped[i].shape[1] < 256):  
        cropped[i] = np.pad(cropped[i], ((int((256-cropped[i].shape[0])/2),int((256-cropped[i].shape[0])/2)),  
                                         (int((256-cropped[i].shape[1])/2),int((256-cropped[i].shape[1])/2)),  
                                         (0, 0)), 'constant')  
    if cropped[i].shape[0]!=256 and cropped[i].shape[1]!=256:
```

```

cropped[i] = np.pad(cropped[i], ((1,0),(1,0),(0,0)), 'constant')
elif cropped[i].shape[0]!=256:
    cropped[i] = np.pad(cropped[i], ((1,0),(0,0),(0,0)), 'constant')
elif cropped[i].shape[1]!=256:
    cropped[i] = np.pad(cropped[i], ((0,0),(1,0),(0,0)), 'constant')
else :
    listDelete.append(i)
    i = i+1

```

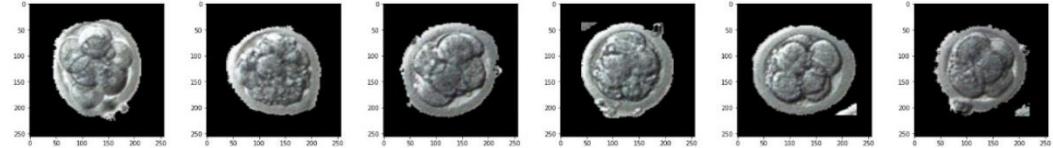
Membuang data yang tidak sesuai

```
In [ ]:
cropped = np.delete(cropped, listDelete)
```

```
In [ ]:
```

```
n=6
fig_width =5*n
fig, ax = plt.subplots(1, n)
fig.set_figwidth(fig_width)
```

```
for i in range(n):
    ax[i].imshow(cropped[i])
```



```
In [ ]:
```

```
print(cropped.shape)
print(croppedResize.shape)

(595,)
(609, 256, 256, 3)
```

Membuat direktori untuk menyimpan citra

```
In [ ]:
```

```
directoryCrop = ".../content/drive/My Drive/Colab Notebooks/Dataset/Embrico/Skripsi"
directoryResizeCrop = ".../content/drive/My Drive/Colab Notebooks/Dataset/Embrico/SkripsiResize"
os.makedirs(directoryCrop, exist_ok=True)
os.makedirs(directoryResizeCrop, exist_ok=True)
```

Menyimpan citra ke dalam direktori

```
In [ ]:
```

```
for i in range(len(cropped)):
    cv2.imwrite(directoryCrop + '/' + str(i) + ".jpg", cropped[i])
```

```
In [ ]:
```

```
for j in range(len(croppedResize)):
    cv2.imwrite(directoryResizeCrop + '/' + str(j) + ".jpg", croppedResize[j])
```

```
In [ ]:
```