BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

2.1.1 Segmentasi Mendalam dan Klasifikasi Tanaman yang Kompleks dengan Citra Satelit Multi-fitur

Penelitian yang dilakukan oleh Lijun Wang, et al. ini membandingkan performa antara UNet++, UNet, Deeplab V3+, PSPNet, dan RF dalam melakukan segmentasi daerah perkebunan menggunakan dataset Sentinel-2A (S-2A). Kelima model tersebut dibandingkan dengan metrik OA (Overall Accuracy), Kappa, dan Macro F1. Hasilnya, UNet++ berhasil memberikan performa yang jauh lebih baik dibandingkan keempat model lainnya pada semua percobaan [8].

Dari penelitian ini, hal yang dijadikan acuan adalah:

- Penggunaan arsitektur UNet++ terbukti memiliki performa lebih baik dibandingkan UNet, Deeplab V3+, PSPNet, dan RF. Oleh karena itu, UNet++ dipilih menjadi arsitektur yang digunakan pada penelitian ini.

2.1.2 Segmentasi dengan Presisi pada Lahan Lepas Pantai dalam Citra SAR Beresolusi Tinggi Berbasis UNet++ yang ditingkatkan

Penelitian yang dilakukan oleh Chuang Yu, et al. ini menggunakan arsitektur UNet++ sebagai basis untuk melakukan segmentasi pada gambar ladang yang didapatkan melalui dataset *high-resolution SAR offshore farm dataset*. Pada penelitian ini, ditambahkan fitur *simulated annealing* yang bertujuan untuk menyesuaikan *learning rate* selama proses training. Proses training juga dilakukan menggunakan 2 resolusi gambar berbeda, yaitu 256×256 piksel, dan 512×512 piksel. Hasilnya, didapatkan bahwa penggunaan resolusi 256×256 piksel memiliki akurasi yang lebih baik, serta kecepatan inferensi yang meningkat hingga 33%. Kemudian, SE-Net dipilih sebagai *feature extraction network* untuk menggantikan bagian *encoding* UNet++ [9].

Dari penelitian ini, hal yang dijadikan acuan adalah:

- Penggunaan gambar berukuran 256×256 piksel terbukti dapat menghasilkan model yang memiliki tingkat akurasi lebih tinggi dan waktu inferensi yang lebih singkat. Oleh karena itu, seluruh gambar pada dataset akan di-*preprocess* dengan metode *patch crop* dengan ukuran 256×256 piksel.

2.1.3 Kerangka UNet dengan *Backbone* Khusus dan Augmentasi DCGAN untuk Segmentasi Penyakit Bercak Daun pada Tanaman Melati dengan Efisien

Penelitian yang dilakukan oleh Shwetha V., et al. ini menggunakan berbagai model dari keluarga U-Net untuk melakukan segmentasi pada daun yang terinfeksi penyakit bercak daun, beserta pembuatan model DCGAN untuk membuat dataset buatan. Ada 5 *backbone* yang digunakan pada penelitian ini, meliputi ResNet, EfficientNet, VGG16, VGG19, dan *custom backbone* berbasis MobileNetV4, sedangkan UNet, WUNet, U2Net, dan ResUNet dipilih sebagai arsitektur model yang dibandingkan satu sama lainnya [10].

Dari penelitian ini, hal yang dijadikan acuan adalah:

- Penggantian *backbone* dapat menjadi opsi yang cukup mudah diimplementasikan dan menghasilkan perbedaan dampak yang cukup signifikan, bahkan dapat mencapai 0.25 untuk metric IoU dan 0.26 untuk *dice coefficient*.

2.1.4 Keuntungan *Transfer Learning* dengan *Pre-trained* Model pada CNN untuk Klasifikasi CT-Scan

Penelitian yang dilakukan oleh Jasman Pardede, Adwityo S. Purohita membandingkan performa model dengan beberapa arsitektur berbeda saat dilatih menggunakan *pretrained weights* ImageNet dan None. Ada 6 arsitektur model yang digunakan pada penelitian ini, yaitu EfficientNetV1, EfficientNetV2, MobileNetV3, Xception, DenseNet, dan InceptionResNetV2. Dari hasil yang didapatkan, nilai F1 Score pada

MobileNetV3L mampu meningkat hingga 0.2, sedangkan Xception meningkat hingga 0.06 [11].

Dari penelitian ini, hal yang dijadikan acuan adalah:

- Penggunaan *pretrained weights* ImageNet terbukti dapat meningkatkan nilai metric F1 Score pada tiap model yang diuji, walaupun peningkatannya berbeda untuk masing-masing model.

2.2 Tinjauan Teori

2.2.1 Segmentasi

Segmentasi merupakan bagian dari *computer vision* yang mempartisi sebuah gambar digital menjadi beberapa segmen, yang kemudian dapat digunakan untuk melakukan object detection dan berbagai operasi lainnya. Dalam operasinya, digunakan istilah *things* dan *stuff* untuk mengkategorikan objek pada gambar. *Things* menggambarkan objek-objek yang dapat dihitung, sedangkan *stuff* merupakan objek-objek yang memiliki jumlah piksel besar dan tidak bisa dihitung, seperti langit, air, dan rumput. *Image segmentation* kemudian terbagi lagi menjadi:

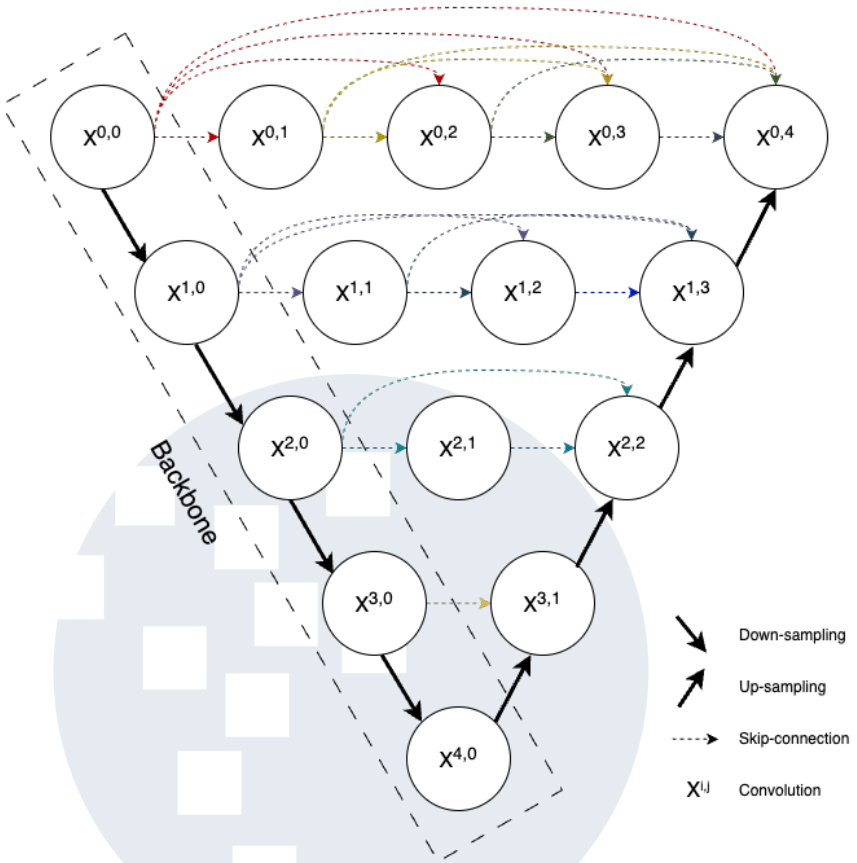
- *Semantic segmentation*, yang menganggap semua piksel sebagai *stuff*, sehingga semua fitur yang memiliki karakteristik piksel serupa dianggap sama tanpa adanya pemisahan antar objek.
- *Instance segmentation*, yang hanya mempartisi *things*. Pada instance segmentation, semua objek yang merupakan *things* akan dipartisi sedetil mungkin sehingga jumlah objek pada setiap kategori dapat dihitung, tetapi semua piksel yang termasuk ke *stuff* tidak akan dipartisi.
- *Panoptic segmentation*, yang merupakan penggabungan *semantic* dan *instance segmentation*. Pada jenis segmentasi ini, semua piksel dianotasi dengan *semantic label* dan juga *instance ID*. Sekelompok piksel yang memiliki label dan ID sama dianggap berada pada 1 kategori yang sama, lalu jika piksel tersebut merupakan bagian dari

stuff, maka ID tidak akan digunakan. Hasil dari segmentasi jenis ini berupa *stuff* yang dipartisi sebagai 1 objek, dan *things* yang dapat dihitung per kategorinya.

Dalam melakukan segmentasi, ada 2 teknik yang dapat digunakan. Yang pertama adalah dengan cara tradisional, yang menggunakan informasi warna dan intensitas pada setiap pikselnya, lalu dilanjutkan dengan *thresholding*, histogram, *edge detection*, *watershed*, *region-based segmentation*, ataupun *clustering-based segmentation*. Yang kedua adalah penggunaan berbagai arsitektur model deep learning seperti FCN, U-Net, Deeplab, Mask R-CNN, dan Transformer [7].

2.2.2 Arsitektur UNet++

UNet++ adalah arsitektur model machine learning untuk segmentasi gambar, yang merupakan pengembangan dari arsitektur UNet. Arsitektur ini menggunakan konsep encoder-decoder. Encoder pada UNet++ menggunakan *backbone* model untuk mengekstrak fitur dari gambar. Pembaharuan yang dibawa oleh arsitektur ini terletak pada koneksi antara *encoder* dan *decoder*, yang menggunakan *nested and dense skip pathways* untuk memproses hasil output dari *encoder* sebelum akhirnya diteruskan ke *decoder* untuk menghasilkan output gambar.



Gambar 2.1 Arsitektur UNet++

Pada masing-masing *skip pathway* yang ada, ditambahkan sebuah *dense convolution block*, yang jumlah *convolution layer*-nya bergantung pada nilai *index decoder stage* pada node penerima. Gambar 2.1 menunjukkan arsitektur UNet++ dengan piramida yang memiliki *index encoder stage* (*i*) dan *index decoder stage* (*j*) masing-masing 5. Dengan mengacu pada gambar yang sama, jumlah *convolution layer* yang ada pada *dense block* antara $X^{0,0}$ hingga $X^{1,3}$ adalah 3, sama halnya dengan informasi dari $X^{0,1}$ ke $X^{0,3}$ juga diproses dengan sebuah *dense block* dengan 3 *convolutional layer*, tetapi informasi dari $X^{0,2}$ ke $X^{0,4}$ diproses dengan *dense block* yang memiliki 4 *convolutional layers*.

Di sisi lain, *j* menggambarkan jumlah input yang diterima dari *layer encoder* sebelumnya dengan perhitungan berupa *j*+1. Misalnya, pada j=0 maka *node* akan menerima 1 input yang berasal dari *encoder* dari *layer* sebelumnya, sedangkan pada j=4, maka akan ada 5 informasi dari

encoder yang diterima, yang dimulai dari $X^{i,0}$ hingga $X^{i,4}$. Hasil yang didapatkan pada layer terakhir, dalam kasus ini pada $X^{4,0}$, akan dikembalikan ke *layer-layer* di atasnya dengan proses cara up-sampling untuk mengembalikan dimensionalitas fitur yang diproses.

Selain itu, pada UNet++ dikenalkan juga *deep supervision* yang memungkinkan model untuk beroperasi pada mode *accuracy*, yang output dari semua *segmentation branch* akan dirata-rata, dan mode *fast*, yang hasil segmentasi akhirnya hanya mengambil dari salah satu *segmentation branch* [12].

2.2.3 EfficientNetB0

Keluarga model EfficientNet merupakan model yang berbasis dari ConvNets, seperti ResNet dan MobileNet, yang dimodifikasi menggunakan skala tertentu pada ketiga dimensinya, yaitu *depth*, *width*, dan *resolution*. Metode *scaling* yang digunakan dinamakan *compound model scaling*, dengan mengikuti *baseline model* EfficientNetB0 yang memiliki arsitektur seperti pada tabel 2.1

Tabel 2.1 Arsitektur *baseline model* EfficientNet

Stage	Operator	Resolution	#Channels	#Layers
1	Conv3×3	224 × 224	32	1
2	MBConv1, k3×3	112×112	16	1
3	MBConv6, k3×3	112×112	24	2
4	MBConv6, k3×3	56×56	40	2
5	MBConv6, k3×3	28×28	80	3
6	MBConv6, k3×3	14×14	112	3
7	MBConv6, k3×3	14×14	192	4
8	MBConv6, k3×3	7×7	320	1
9	Conv1×1 & Pooling & FC	7×7	1280	1

Sedangkan, rumus yang digunakan pada *compound model scaling* adalah sebagai berikut:

depth : $d = a^{\varphi}$

width : $w = \beta^{\varphi}$

resolution: $r = \gamma^{\varphi}$

Di mana φ adalah koefisien yang ditentukan oleh pengguna. Dengan mengikuti rumus tersebut, perubahan pada nilai φ akan merubah total FLOPS sebesar 2^{φ}.

Dalam melakukan *scaling* model keluarga EfficientNet, ada 2 langkah utama yang dilakukan. Pertama, nilai φ diasumsikan sebagai 1 untuk mencari nilai α, β, γ pada EfficientNetB0 mengikuti rumus $\alpha.\beta^2.\gamma^2 \approx 2$. Hasilnya, didapatkan nilai $\alpha = 1.2$, $\beta = 1.1$, dan $\gamma = 1.15$, yang kemudian digunakan sebagai nilai konstan untuk model-model berikutnya. Kemudian, ketiga nilai konstan tersebut digunakan sebagai nilai variabel rumus di atas, dengan perubahan nilai φ untuk setiap modelnya, sehingga rumusnya menjadi:

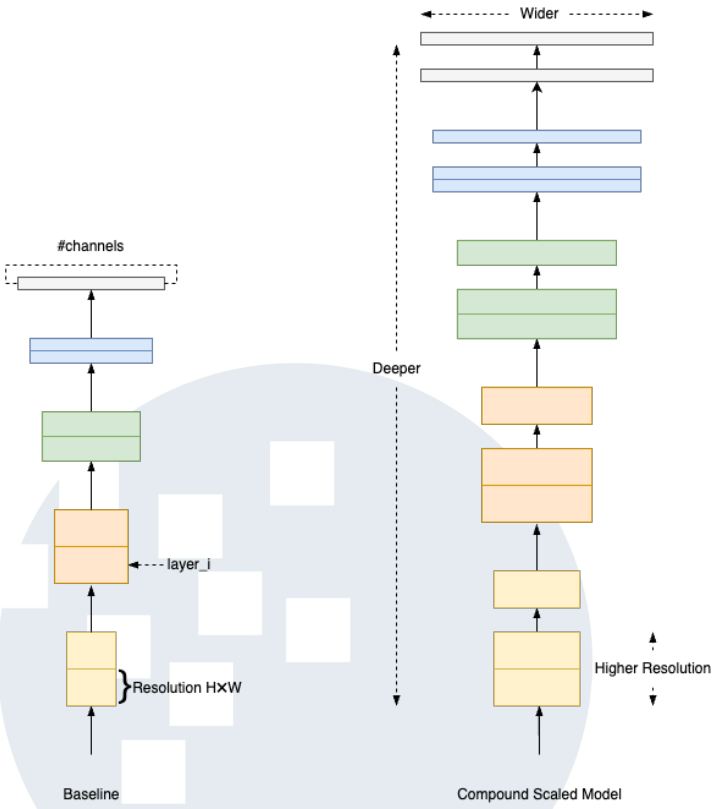
depth : $d = 1.2^{\varphi}$

width : $w = 1.1^{\varphi}$

resolution: $r = 1.15^{\varphi}$

Nilai φ yang digunakan dapat diketahui dari nama model yang digunakan, seperti pada EfficientNetB0, berarti nilai φ-nya adalah 0. Kemudian, hasil perhitungan yang didapatkan akan dijadikan parameter *scaling* terhadap *baseline model* yang ada.

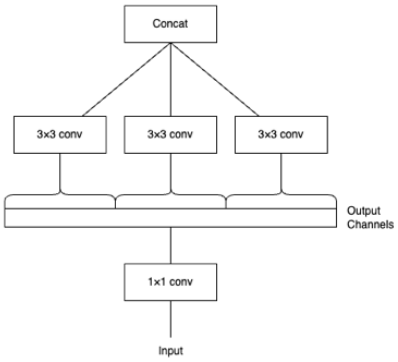
Ilustrasi dari teknik *compound model scaling* dapat dilihat pada gambar 2.2, di mana gambar di kiri merupakan arsitektur model awal atau *baseline*, sedangkan gambar di kanan merupakan arsitektur model melewati proses scaling [14].



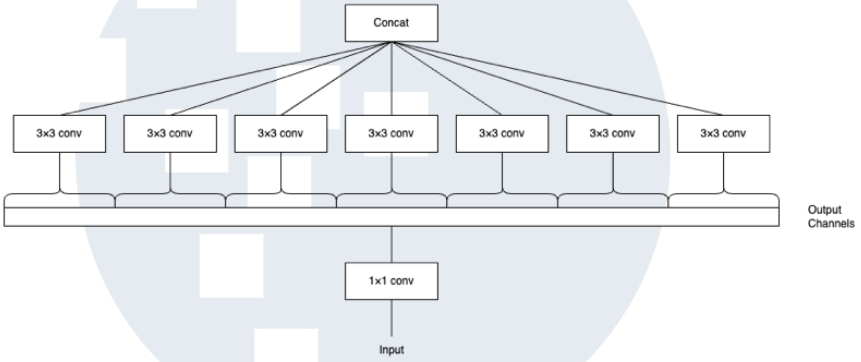
Gambar 2.2 *Compound model scaling* pada keluarga model EfficientNet

2.2.4 Xception

Model Xception merupakan pengembangan dari model keluarga Inception dan merupakan singkatan dari "*Extreme Inception*". Peningkatan dilakukan dengan mengubah modul *inception* menjadi modul *depthwise separable convolution*. Modul baru itu terdiri dari 2 jenis operasi konvolusi, yaitu *depthwise convolution*, yang merupakan *spatial convolution* yang dilakukan secara independent di setiap kanal input, dan diikuti oleh *pointwise convolution*, yang melakukan konvolusi terhadap output dari setiap kanal dengan kernel 1×1. *Layer* ReLU juga ditambahkan untuk menghadirkan *non-linearity*. Ilustrasi dari perubahan ini dapat dilihat pada gambar 2.3 yang menunjukkan arsitektur model keluarga Inception, dan gambar 2.4 yang menunjukkan hasil perubahan yang dilakukan pada arsitektur model Xception.

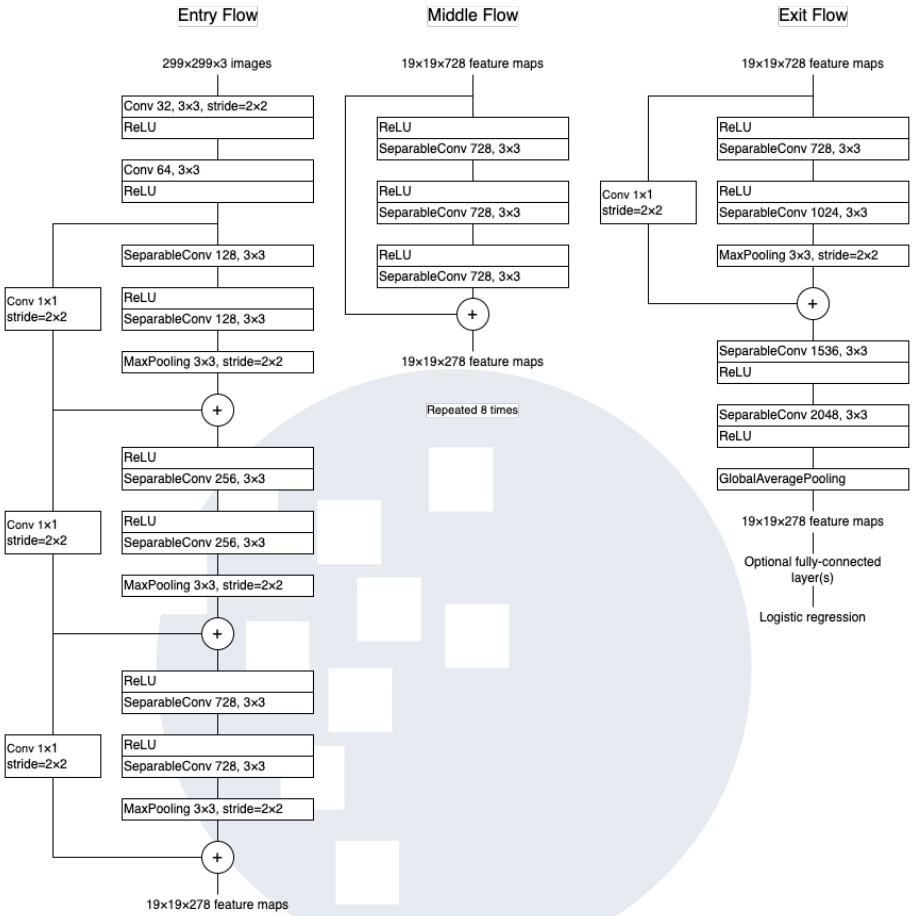


Gambar 2.3 Arsitektur model Inception



Gambar 2.4 Arsitektur model Xception

Model Xception mengandalkan 36 lapisan konvolusi dalam melakukan feature extraction, dengan detail seperti pada gambar 2.5 [15].



Gambar 2.5 Detail arsitektur model Xception

2.2.5 MobileNetV2

MobileNetV2 merupakan model *machine learning* yang didesain untuk perangkat *mobile*, dengan performa yang cukup tinggi dan jumlah operasi yang sedikit. 2 fitur utama pada model ini adalah penggunaan modul *inverted residual* dengan *linear bottleneck*. Selain itu, digunakan juga *depthwise separable convolutions* dan expansion layer ReLU6.

Pada prinsipnya, input yang diterima akan ditambahkan dimensinya melalui *expansion layer*, lalu diteruskan ke blok konvolusi untuk mendapatkan fitur spasial, dan akhirnya hasil yang didapatkan akan diproses pada blok *linear bottleneck* untuk mengembalikan dimensionalitasnya menjadi sama seperti semula. Seluruh proses tersebut disebut sebagai konsep inverted residual, yang membalikkan proses

residual block yang pada umumnya melakukan konvolusi linear pada bottleneck block terlebih dahulu, sebelum akhirnya meneruskannya ke expansion layer.

Detail arsitektur MobileNetV2 dan detail *bottleneck residual block* dapat dilihat pada tabel 2.2 dan tabel 2.3 [16].

Tabel 2.2 Arsitektur MobileNetV2

Input	Operator	*t*	*c*	*n*	*s*
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
1×1×1280	conv2d 1×1	-	k	-	

Tabel 2.3 *Bottleneck residual block* pada MobileNetV2

Input	Operator	Output
$h \times w \times k$	1×1 conv2d, ReLU6	$h \times \mathrm{w} \times (\mathrm{tk})$
$h \times w \times tk$	3×3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{\mathrm{w}}{\mathrm{s}} \times (\mathrm{tk})$
$\frac{h}{s} \times \frac{\mathrm{w}}{\mathrm{s}} \times \mathrm{tk}$	Linear 1×1 conv2d	$\frac{h}{s} \times \frac{\mathrm{w}}{\mathrm{s}} \times \mathrm{tk}'$

dengan penjelasan setiap variabelnya sebagai berikut:

- *t* = faktor perluasan jumlah channel
- *c* = output channels
- *n* = number of repetitions
- *s* = langkah operasi konvolusi pertama (*stride*)

- h = dimensi tinggi input
- w = dimensi lebar input
- k = ukuran kernel konvolusi
- tk = total kernel setelah ekspansi
- tk' = total kernel setelah *bottleneck layer*