BAB II

LANDASAN TEORI

2.1 Software Engineering

Software engineering adalah sebuah disiplin ilmu terkait dengan semua aspek pada produksi piranti lunak. Istilah tersebut pertama kali muncul pada konferensi yang diadakan oleh NATO di tahun 1968. Ilmu software engineering digunakan untuk mengatasi permasalahan yang sering terjadi pada pengembangan piranti lunak, diantaranya seperti tahap pengembangan *software* berlangsung dengan lambat, memakan biaya besar, dan masih belum dapat menghasilkan piranti lunak yang memiliki fungsionalitas sesuai dengan apa yang dibutuhkan *user* (Macnab & Doctolero, 2020). Sepanjang tahun 1970-an dan 1980-an, teknik dan metode *software engineering* baru terus bermunculan, seperti *structured programming, information hiding*, dan *object-oriented development*. *Tools* dan notasi standar juga mulai dikembangkan dan digunakan secara luas (Sommerville, 2011).

Menurut Sami (2012), *software development life cycle* (SDLC) adalah fase yang memberikan gambaran umum mengenai proses pembuatan *software*. Sebagaimana *software* akan direalisasikan dan dikembangkan dari pemahaman bisnis dan fase elisitasi persyaratan untuk mengkonversi ide-ide bisnis dan persyaratan ke dalam fungsi dan fitur sampai mencapai kebutuhan bisnis. *Software engineer* yang baik harus memiliki pengetahuan yang cukup tentang cara memilih model SDLC berdasarkan konteks proyek dan persyaratan bisnis (Sami, 2012).

Requirements engineering merupakan cabang dari *software engineering* yang terkait dengan tujuan dunia nyata untuk fungsi dan batasan pada sistem *software* yang berhubungan dengan faktor dan spesifikasi perilaku *software* yang tepat dan evolusi mereka dari waktu ke waktu (Zave, 1997). Tahap ini digunakan untuk menerjemahkan kebutuhan dan keinginan *user* sehingga *software* dalam spesifikasi yang lengkap dapat memudahkan pencapaian tujuan dari dikembangkannya *software* tersebut.

2.2 Requirements Engineering

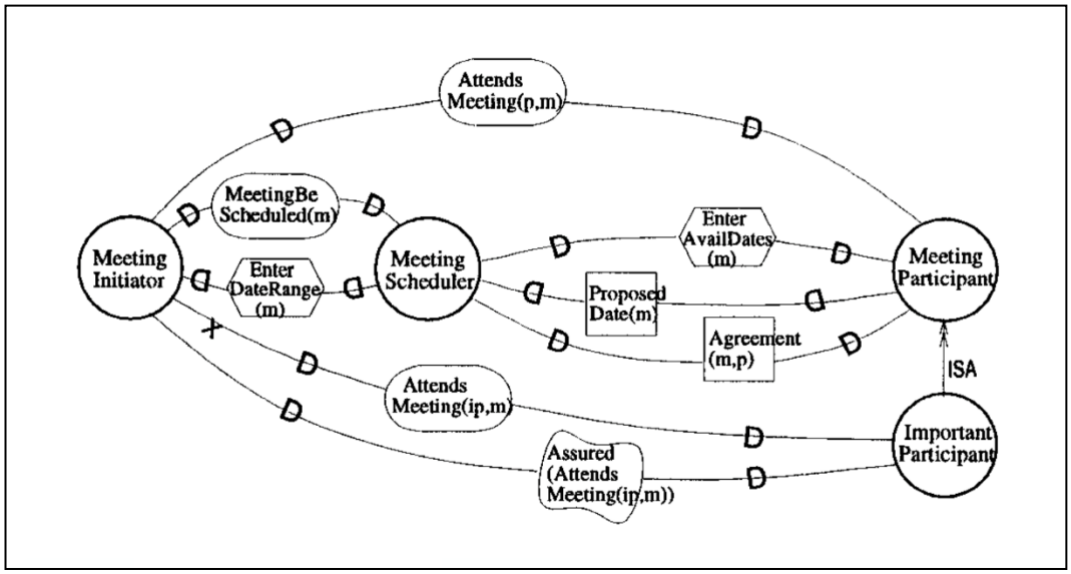
Requirements engineering adalah proses mendefinisikan, mendokumentasi dan memelihara kebutuhan untuk pengembangan *software* (Chemumuturi, 2013). Tahap ini melibatkan sejumlah proses untuk mengumpulkan *requirements* sesuai dengan kebutuhan dan permintaan pengguna dan *stakeholder* dari produk software (Rehman, dkk, 2013). Dalam merancang sebuah *software*, akan terdapat banyak sekali faktor yang dipertimbangkan untuk menjadi pilihan dalam mengembangkan software. Faktor-faktor tersebut biasanya berkaitan dengan kebutuhan software itu sendiri, kebutuhan pengguna, dan batasan-batasan sistem ketika diimplementasikan. Oleh karena itu *requirements engineering* memiliki peran penting dalam *software engineering*. Kesalahan dan ketidaklengkapan pengumpulan *requirements* dalam tahap ini dapat mempengaruhi keberhasilan proyek pengembangan piranti lunak secara keseluruhan (Rehman, dkk, 2013).

Salah satu framework yang bisa digunakan untuk requirements engineering adalah *framework* iStar 2.0. Framework iStar 2.0 adalah *framework* dalam *software engineering* yang mendukung pemodelan dan penalaran berdasarkan *socio-technical* sistem dan organisasi (Franch, dkk, 2011). iStar berfokus pada yang

dimaksud (*why?*), *social* (*who?*), dan strategis (*how? how else?*). Framework iStar telah digunakan dalam banyak bidang seperti, *healthcare, security analysis,* dan *e-commerce* (Dalpiaz, et al., 2016). Framework iStar 2.0 merupakan bentuk penyempurnaan dari yang sebelumnya, yaitu framework iStar sebagai upaya kolektif dari komunitas pengguna iStar untuk mengatasi kesulitan dengan mendefinisikan konsep-konsep dasar (López, dkk, 2016).

2.3 Strategic Dependency Model

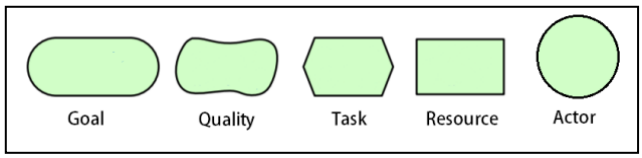
Strategic Dependency (SD) model terdiri dari sekumpulan node dan relasi. Tiap node mewakili *actor* dan tiap relasi antar dua *actor* menunjukkan bahwa satu actor bergantung dengan aktor lainnya agar yang pertama dapat mencapai beberapa tujuan (Yu & Liu, 2001). SD model digunakan untuk mendeskripsikan hubungan ketergantungan antara berbagai aktor dalam konteks organisasi. (Yu, 2001).



Gambar 2.1 Strategic Dependency Model dari *meeting scheduler* (Yu, 2001)

Gambar 2.1 menunjukkan contoh dari skenario penjadwalan rapat menggunakan Strategic Dependency *model* pada diagram iStar 2.0. Terdapat lima

macam notasi pada diagram iStar 2.0 yang menggunakan SD model yang dapat dilihat pada gambar 2.2.



Gambar 2.2 Notasi pada Strategic Dependency Model (Dalpiaz, dkk, 2016)

Pada gambar 2.2, terdapat notasi *Goal, Quality, Task, Resource* dan *Actor*. *Goal* merupakan kondisi yang ingin dicapai oleh *actor* dan memiliki kriteria pencapaian yang jelas. *Quality* merupakan atribut yang diinginkan aktor pada tingkat pencapaian tertentu, dimana *quality* dapat memandu pencarian cara mencapai tujuan, dan juga berfungsi sebagai kriteria untuk mengevaluasi cara-cara alternatif untuk mencapai tujuan. *Task* merepresentasikan tindakan yang ingin dieksekusi oleh aktor, biasanya dengan tujuan mencapai beberapa tujuan. *Resource* merupakan entitas fisik atau informasi yang dibutuhkan oleh aktor untuk melakukan *task*. *Actor* adalah entitas aktif dan otonom yang bertujuan mencapai *goals* mereka dengan menggunakan pengetahuan mereka, bekerja sama dengan aktor lain (Dalpiaz, dkk, 2016).

2.4 Instance Segmentation

Instance segmentation merupakan teknik segmentasi citra hingga ke level piksel. *Instance segmentation* digunakan agar dapat mendeteksi beberapa objek yang berada pada kelas sama maupun beda sebagai sebuah individual objek (Silberman, et al., 2014). Teknik ini menggabungkan elemen-elemen dari teknik klasik *computer vision* yaitu *object detection* dengan cara mengklasifikasikan tiap objek pada citra, dan *localize* tiap objek menggunakan *bounding box* dan *semantic*

segmentation (He, dkk, 2018). Setiap hasil deteksi memberikan beberapa informasi seperti kelas dari tiap objek yang ditemukan, dan lokasi ditemukannya objek tersebut.

2.5 Deep Learning

Deep learning merupakan salah satu bagian dari Machine Learning yang fokus pada pembelajaran yang mengikuti cara kerja jaringan saraf manusia (Petterson & Gibson, 2017). Deep learning melakukan pembelajaran secara bertahap menjadi beberapa level representasi dan abstraksi untuk membangun sebuah informasi yang lebih kompleks dari kumpulan informasi sederhana (Bengio, 2009). Pembelajaran menggunakan rangkaian *layer* unit pemrosesan nonlinear untuk melakukan ekstraksi fitur dan transformasi. Hasil pemrosesan dari tiap *layer* digunakan sebagai input pada *layer* selanjutnya (Deng & Yu, 2014).

2.6 Convolutional Neural Network

Salah satu algoritma yang mengimplementasikan konsep deep learning adalah *Convolutional Neural Network*. *Convolutional Neural Network* (CNN) merupakan algoritma pembelajaran menggunakan representasi jaringan saraf untuk memproses dan mengenali objek visual (Valueva, dkk, 2020). Algoritma CNN terdiri atas *layer* input dan output, serta beberapa *hidden layer*. *Hidden layer* pada algoritma CNN terdiri atas konvolusi *layer*, *pooling layer*, dan *fully connected layer*.

CNN akan menerima input sebuah citra dan menghasilkan output klasifikasi berdasarkan kategori tertentu. Algoritma akan mentransformasi input citra sebagai array tiga dimensi yang mewakili tinggi *h*, lebar *w*, dan *channel c* yang terdiri dari nilai *red*, *green* dan *blue* dari masing-masing piksel (Valueva, dkk, 2020).

Kemudian, citra akan dimasukkan ke dalam *layer* konvolusi untuk ekstraksi fitur. Proses dilakukan dengan mengkalikan konvolusi matriks dengan tinggi *m*, lebar *n* dengan matriks filter berukuran $j \times k$ atau yang biasa disebut sebagai *feature map*. Output dari proses konvolusi selanjutnya akan dijadikan input pada *layer* konvolusi berikutnya. Rumus 2.1 menunjukkan rumus proses konvolusi.

$$G[m,n] = (f * h)[m,n] = \sum_{j}\sum_{k} h[j,k]f[m-j,n-k]$$

Rumus 2.1 Konvolusi (Goodfellow, dkk, 2016)

Citra yang telah melalui proses konvolusi akan dimasukkan ke dalam *pooling layer*. *Pooling layer* berfungsi untuk menyederhanakan ukuran spasial representasi untuk mengurangi jumlah parameter dan komputasi pada network, serta menghindari terjadinya *overfitting* (Mishkin, dkk, 2016). Cara kerja dari *pooling layer* adalah dengan menerapkan operasi pada tiap kedalaman *slice* input dan mengurangi ukuran spasial dengan *max-pooling* secara terpisah. *Max-pooling* adalah cara *pooling* yang paling umum, yaitu dengan mempertahankan nilai piksel yang paling besar dari tiap *slice* input dengan ukuran matriks *m x n* (Mishkin, dkk, 2016). Rumus 2.2 menunjukkan rumus *max-pooling*.

$$y = max_{i,j=1}^{h,w} x_{i,j}$$

Rumus 2.2 *max pooling* (Mishkin, dkk, 2016)

Selain itu, metode *pooling* yang bisa digunakan adalah *average-pooling*. *Average-pooling* dilakukan dengan cara menghitung nilai rata-rata tiap *slice* input dengan ukuran matriks m x n. Rumus 2.3 menunjukkan rumus *average-pooling*.

$$y = \frac{1}{hw} \sum_{i,j=1}^{h,w} x_{ij}$$

Rumus 2.3 *average pooling* (Mishkin, dkk, 2016)

Citra yang telah melalui *layer* konvolusi dan *pooling layer* akan diproses pada *fully connected layer*. *Fully connected layer* merupakan *layer* yang berisi *artificial neural network*. *Layer* tersebut terdiri dari transformasi linear dan transformasi non-linear. Namun, output dari dua *layer* sebelumnya merupakan matriks dua dimensi (Goodfellow, dkk, 2016). Agar dapat diproses ke dalam *fully connected layer*, output dari *layer* sebelumnya harus dikonversi menjadi format satu dimensi terlebih dahulu.

$$Z = b + \sum_{i=1}^{n} x_i w_i$$

Rumus 2.4 Transformasi Linear (Reese, dkk, 2017)

Rumus 2.4 menunjukkan rumus dari transformasi linear. w_i merupakan weight yang terdapat pada tiap iterasi, x_i merupakan input, *n* merupakan jumlah input yang terdapat pada layer, dan *b* merupakan nilai bias.

Transformasi non-linear sering disebut juga sebagai *Rectified Linear Unit* (ReLU). ReLU digunakan untuk menghilangkan nilai negatif dengan mengubah menjadi bernilai nol (Romanuke, 2017). Rumus 2.5 menunjukkan rumus dari ReLU.

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

Rumus 2.5 ReLU (Krizhevsky, dkk, 2017)

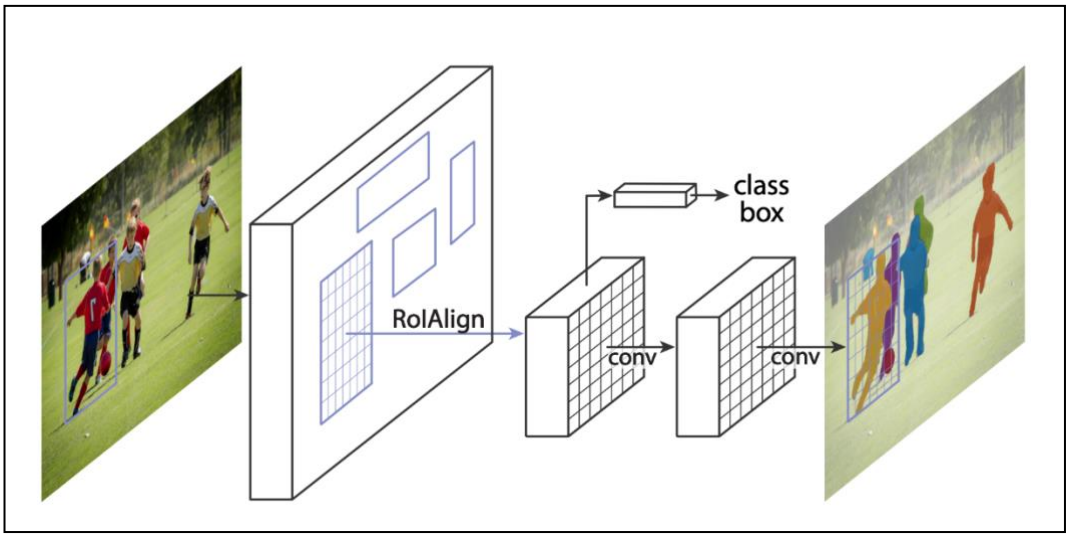
Pembelajaran algoritma CNN dilakukan memerlukan *loss function* untuk memperhitungkan tingkat *error* dari model yang dibangun. Menurunkan nilai *loss function* dapat memperkecil kemungkinan *error* yang terjadi sehingga output dapat diberi peringkat dan dikomparisasi. Rumus 2.6 menunjukkan rumus dari *loss function*.

$$loss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})$$

Rumus 2.6 *loss function* (Goodfellow, dkk, 2016)

2.6 Mask R-CNN

Mask R-CNN merupakan salah satu metode yang dapat digunakan untuk menyelesaikan permasalahan *instance segmentation*. Metode ini mulai diperkenalkan oleh He dkk pada tahun 2018. Mask R-CNN adalah pengembangan dari algoritma sebelumnya, yaitu Faster R-CNN yang merupakan turunan dari algoritma CNN. Pada algoritma Mask R-CNN, He dkk (2018) berhasil menciptakan sebuah *layer* baru untuk memprediksi *segmentation mask* dari tiap *Region of Interest* (ROI) yang berjalan secara parallel dengan *layer* untuk klasifikasi dan *bounding box regression* yang sudah ada sebelumnya (He, dkk, 2018). Mask R-CNN juga dikembangkan untuk memperbaiki ketidakselarasan segmentasi objek pada algoritma Faster R-CNN. Hal ini dilakukan dengan mengganti metode RoIPooling yang telah digunakan pada algoritma Fast R-CNN dan Faster R-CNN, dengan sebuah metode yang dinamakan metode RoIAlign.



Gambar 2.3 Arsitektur dari Mask R-CNN (He, dkk, 2018)

Gambar 2.3 menunjukkan arsitektur dari Mask R-CNN. Mask R-CNN memiliki langkah-langkah sebagai berikut :

a) *Region Proposal Network* (RPN), yaitu pencarian secara cepat kemungkinan lokasi objek dari suatu input gambar. Posisi kemungkinan objek dalam gambar kemudian diberi *boundary* dan ditandai sebagai *region of interest* (ROI). RPN kemudian mengambil sub-citra dari kemungkinan objek dengan berbagai ukuran sebagai input ke dalam jaringan *Convolutional Neural Network* dengan output kumpulan persegi panjang dengan skor kemiripan objek.

b) Prediksi kelas (*classification*) dan *box offset* (*regression*) secara parallel dari kemungkinan objek. Dari langkah ini, Mask R-CNN juga mengeluarkan output yaitu sebuah binary mask untuk setiap RoI.

2.7 Color-to-grayscale

Color-to-grayscale merupakan proses mengubah nilai RGB dari masing-masing pixel di suatu gambar menjadi nilai tunggal (8-bit) dengan *range* 0-255. Hasil citra yang dihasilkan oleh proses *color-to-grayscale* terdiri atas warna abu-abu, yang memiliki variasi pada warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat (Fatta, 2007). Salah satu metode yang dapat digunakan untuk melakukan *color-to-grayscale* adalah metode Gleam.

$$Gleam = \frac{1}{3}(R' + G' + B') \quad \ldots (2.1)$$

Rumus 2.7 *color-to-grayscale* Gleam (Kanan & Cottrell, 2012)

Pada gambar 2.7 menunjukkan rumus dari algoritma *color-to-greyscale* Gleam. Berdasarkan penelitian Kanan dan Cottrell (2012), disimpulkan bahwa untuk mendeteksi objek dan *face recognition*, metode Gleam hampir selalu memiliki performa terbaik.

2.8 Gaussian Noise

Gaussian *noise* sering disebut sebagai gangguan elektronik karena muncul dalam *amplifier* maupun detektor. Gaussian *noise* yang muncul secara alami biasanya disebabkan oleh getaran termal atom dan sifat radiasi diskrit benda hangat. Gaussian noise juga dikenal sebagai stastistik *noise*. Hal ini berkaitan dengan *probability density function* (PDF) yang dimiliki oleh Gaussian *noise* dapat direpresentasikan dalam distribusi normal yang biasa dikenal juga sebagai distribusi Gaussian (Jain, 1989). Menurut Boyat dan Joshi (2015), Gaussian *noise* umumnya mengganggu nilai *gray* dalam gambar digital.

$$P_{(g)}(z) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(z-\mu)^2}{2\sigma^2}} \qquad \ldots(2.2)$$

Rumus 2.8 Gaussian *noise*

Rumus 2.8 menunjukkan rumus Gaussian *noise*. z merepresentasikan tingkat intensitas nilai *gray*, μ merepresentasikan nilai *mean*, dan σ merepresentasikan standar deviasi. Nilai *mean* μ merupakan nilai tengah dari persebaran *noise*. Kemunculan piksel *noise* akan memiliki frekuensi paling tinggi pada nilai *mean* μ. Standar deviasi σ menentukan jarak persebaran *noise*. Nilai *noise* akan selalu berada pada *range* antara *mean* $\mu -$ standar deviasi σ dan *mean* $\mu +$ standar deviasi σ.

2.9 Simple Random Sampling

Simple random sampling merupakan metode pengambilan sampel sebuah populasi besar. Pemilihan sampel yang berbeda dilakukan sebanyak *k* kali dari item sebanyak *n* dalam populasi melalui suatu cara sehingga setiap kombinasi dari item *k* memiliki probabilitas terpilih yang sama (Thompson, 2012). Metode tersebut digunakan untuk membagi dataset yang telah dibuat menjadi dataset *training* dan pengujian. Menurut penelitian Krizhevsky dkk (2017), dataset dapat dibagi dengan perbandingan 90% untuk *training* dan 10% untuk pengujian.

2.10 Intersection over Union

Instance segmentation model sedikit lebih rumit untuk dievaluasi. Sebagaimana *semantic segmentation* model hanya menghasilkan satu output, yaitu *segmentation mask*, sedangkan pada *instance segmentation* menghasilkan

kumpulan *segmentation mask* lokal yang menggambarkan tiap objek yang terdeteksi dalam suatu citra. Untuk mengevaluasi model perlu dilakukan perhitungan terhadap *Intersection over Union* (IoU) dari tiap *segmentation mask*.

Intersection over Union (IoU) atau yang biasa juga dikenal sebagai *Jaccard index* merupakan metrik yang biasa digunakan untuk membandingkan kemiripan antar dua objek *arbitrary,* yaitu *bounding-box* prediksi dan *bounding-box ground-truth* (Rezatofighi, dkk, n.d.). Pengukuran dilakukan berdasarkan luas area *bounding-box* prediksi dan *bounding-box ground-truth* yang saling tumpang tindih. Rumus 2.3 menunjukkan dari rumus IoU.

$$IoU = \frac{ground-truth \cap prediction}{ground-truth \cup prediction} \quad \ldots(2.3)$$

Rumus 2.9 *Intersection over Union (IoU)* (Rezatofighi, et al., n.d.)

Nilai IoU yang telah diperoleh selanjutnya akan digunakan untuk menentukan apakah bounding-box prediksi termasuk dalam kategori True Positive (TP), False Positive (FP), dan False Negative (FN). Menurut Hosang dkk (2015), pada *instance segmentation*, *True Positive* mengindikasikan bahwa pasangan *bounding-box* prediksi dan *bounding-box ground-truth* yang sesuai memiliki nilai IoU melebihi nilai *threshold* yang telah ditetapkan. *False Positive* mengindikasikan ketika *bounding-box* prediksi dan *bounding-box ground-truth* memiliki nilai IoU kurang dari nilai *threshold* yang telah ditetapkan atau memiliki bounding-box prediksi yang saling tumpang tindih. *False Negative* mengindikasikan bahwa *confidence score* dari hasil deteksi seharusnya berada dibawah nilai *threshold*. Menurut Feichtenhofer dkk (2018), nilai IoU *threshold* yang digunakan adalah 0.5. *Confidence score* merupakan nilai dihasilkan oleh *classifier* mengenai probabilitas

sebuah *bounding box* berisi suatu objek yang telah diberi anotasi. Berdasarkan nilai *True Positive*, *False Positive*, *False Negative* dapat diperoleh nilai presisi dan *recall*.

Nilai presisi menggambarkan nilai akurasi sistem dalam memprediksi objek. Hal ini dilakukan dengan membandingkan banyak objek yang memiliki hasil prediksi benar terhadap keseluruhan objek yang berhasil diprediksi oleh sistem. Sedangkan *recall* menggambarkan nilai keberhasilan dalam memprediksi semua objek yang memiliki nilai positif. Rumus dari presisi dan *recall* secara berturut-turut ditunjukkan pada rumus 2.4 dan rumus 2.5.

$$Presisi = \frac{TP}{TP+FP} \quad \ldots(2.4)$$

Rumus 2.10 Presisi (Fawcett, 2005)

$$Recall = \frac{TP}{TP+FN} \quad \ldots(2.5)$$

Rumus 2.11 Recall (Fawcett, 2005)

2.11 Mean Average Precision (mAP)

Average Precision (AP) merupakan salah satu metrik yang paling sering digunakan dalam *object detection*. *Average Precision* muncul akibat selalu adanya *trade-off* antara nilai presisi dan *recall*, dimana umumnya *recall* selalu naik ketika nilai presisi menurun (Zhang & Su, 2012). Untuk mengatasi permasalahan tersebut, maka digunakan *Average Precision* menghitung nilai presisi rata-rata terhadap nilai

recall didalam *range* 0 hingga 1. Hal ini dapat dilakukan juga dengan menghitung luas daerah dibawah kurva presisi-*recall* (Everingham, dkk, 2015). Rumus dari *Average Precision* ditunjukkan pada rumus 2.6.

$$AP = \frac{1}{11}\sum_{r\epsilon(0,0.1,\ldots,1)} p_{interp}(r_{n+1}) \quad \ldots(2.6)$$

$$p_{interp}(r_{n+1}) = \max\ p(\tilde{r})\ , \tilde{r} \geq r_{n+1}$$

Rumus 2. 12 *Average Precision* (Everingham, dkk, 2015)

Mean Average Precision (mAP) merupakan perhitungan nilai rata-rata AP apabila terdapat lebih dari satu kelas pada dataset. Selanjutnya perhitungan mAP disempurnakan pada kompetisi deteksi objek yang diselenggarakan oleh Microsoft Common Object in Context (COCO), yaitu dengan menambahkan perhitungan rata-rata nilai AP untuk masing-masing kelas dengan variasi nilai IoU *threshold* (Microsoft COCO, n.d.). Nilai *threshold* yang umumnya dipakai berada dalam range 0.5 hingga 0.95, dengan nilai step 0.05. Nilai mAP kemudian dinotasikan dengan mAP@[.5, .95].