BAB II

LANDASAN TEORI

2.1 Deep Learning

Deep learning memungkinkan model komputasi yang terdiri dari beberapa lapisan pengolahan untuk belajar merepresentasikan data dengan berbagai tingkat abstraksi (LeCun, Bengio & Hinton 2015). *Deep learning* merupakan metode yang diadaptasi dari teknik *machine-learning* yang sering digunakan pada umumnya, dimana teknik *machine learning* tersebut tidak dapat mengolah data mentah yang dijadikan input untuk model pada *machine learning*. Keterbatasan dari teknik *machine learning* pada umumnya terletak pada pengolahan data harus diolah melalui sebuah *feature*(ciri) *extractor* yang di desain khusus oleh para *engineer machine learning.*

Penggunaan *deep learning* sebagai teknik pembuatan model komputasi, dapat secara langsung mengolah data mentah tanpa menggunakan *feature extractor* untuk melakukan klasifikasi ataupun pengenalan. Pada umumnya, diantara banyaknya deskripsi pada tingkat tertinggi *deep learning*, ada dua aspek kunci: (1) Model memiliki beberapa *layer* atau tahap pengolah informasi *non-linear;* dan (2) Metode pembelajaran *supervised* atau *unsupervised* untuk merepresentasikan *feature* lebih akurat, dan lebih banyak *layer* abstrak (Deng & Yu, 2013). Setiap layer yang membentuk model *deep learning* mampu menghasilkan aspek penting pada data yang diinput untuk melakukan pengelompokkan atau klasifikasi.

Sebagai contoh, sebuah gambar yang diproses kembali menjadi sebuah larik yang berisikan angka, dimana angka tersebut merupakan hasil dari kumpulan *pixels* dari gambar. Kemudian larik tersebut dijadikan sebagai *input* pada model *deep learning,* pada layer pertama yang memiliki fitur yang dapat mempelajari adanya kehadiran atau ketidakhadiran atas sebuah sudut pada sebuah orientasi dan letaknya pada gambar. Pada layer kedua secara khusus mendeteksi motif dengan melihat susunan partikel sudut, terlepas dari melihat susunan kecil dari setiap posisi sudut. Layer ketiga, mungkin akan menyusun kombinasi yang lebih besar sesuai pada bagian objek yang sudah dikenal oleh model, dan layer yang terakhir akan melakukan klasfikasi untuk mendeteksi kombinasi motif tersebut (Deng & Yu, 2013).

Deep learning membawa perubahan besar dan menjadi metode terbaik untuk pengembangan *artificial intelligence. Deep learning* meningkatkan optimalisasi terhadap teknik machine learning dalam pengenalan gambar, pengenalan suara, dan klasifikasi lainnya yang dapat dilakukan dengan memproses data mentah. Selain itu, *deep learning* dapat diterapkan untuk menyelesaikan beberapa permasalahan mengenai keterbatasan komputer, seperti *face recognition*, *natural language processcing,* dan lain-lain. Salah satu model arsitektur *deep learning* yang telah terbukti efektif yaitu *convolutional neural network* yang biasa digunakan untuk memproses data berupa gambar atau citra, dan *recurrent neural network* yang biasa digunakan untuk memproses data rangkaian seperti suara atau *audio.*

2.2 Supervised Learning

Supervised learning memerlukan pembelajaran sebuah pemetaan antara satu set *variable input x* dan sebuah *output y* dan menerapkan pemetaan ini untuk memprediksi *output* dari data yang belum pernah telihat (Cunningham, Cord & Delany, 2008). *Supervised learning* merupakan salah satu tipe *machine learning* yang sering digunakan dari dua tipe *machine learning* lainnya yaitu, *unsupervised learning* dan *reinforcement learning.* Tipe *machine learning* ini dipanggil sebagai *supervised learning* karena proses algoritma yang belajar dari sebuah *training dataset* dapat dianggap sebagai pengajar yang mengawasi proses belajar. Algoritma tersebut secara iteratif membuat prediksi pada *training dataset* dan dibenarkan oleh pengajar. Proses belajar akan berhenti ketika model mencapai peforma yang baik pada suatu tingkatan.

Sebagai contoh, jika sebuah ingin membangun sebuah model klasifikasi apakah sebuah gambar merupakan sebuah rumah, mobil, atau binatang $\{y'_1, y'_2, \ldots, y'_n\}$. Pertama harus adanya pengumpulan data sebanyak N $\{x_1, x_2, \ldots, x_n\}$ untuk setiap n diberi label (y) sesuai dengan kategori data tersebut. Selama proses pelatihan model akan menghasilkan *output* dalam bentuk vector untuk setiap kategori. Hasil *output* yang diinginkan untuk kategori yang diinginkan adalah angka yang terbesar dari seluruh kategori. Namun, hal tersebut jarang terjadi sebelum adanya proses *training.* Harus adanya komputasi terhadap perhitungan *error* antara output dengan pola angka yang diinginkan. Kemudian, *machine learning* memodifikasi parameter *internal* yang disesuaikan untuk mengurangi *error*. Parameter yang dapat disesuaikan ini sering di denotasikan sebagai *weight,* yang

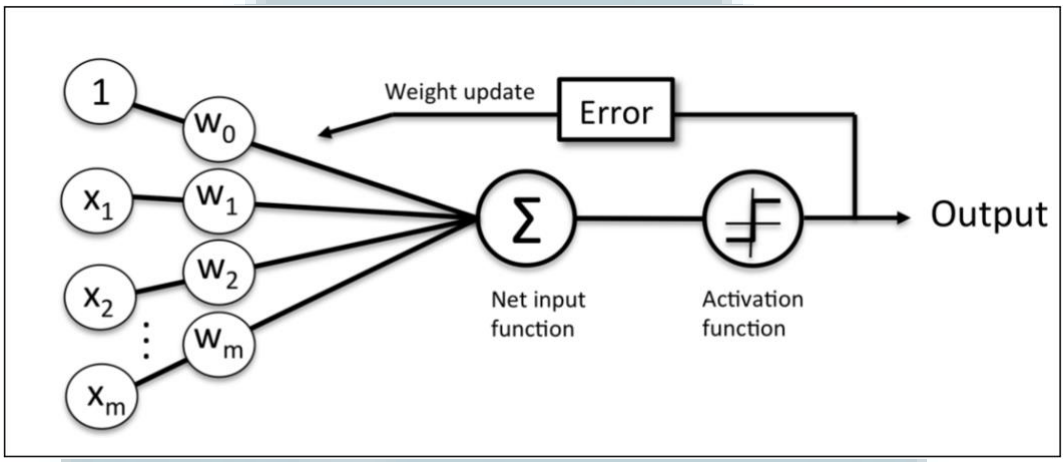
merupakan bilangan ril. Dalam sistem *deep learning*, parameter tersebut dapat berjumlah jutaan dan juga jumlah *label* pada masing-masing data untuk melatih model atau mesin.

2.3 Neural Network

Neural network atau juga dapat dipanggil *artificial neural network* adalah model jaringan syaraf buatan yang di rancang untuk meniru cara kerja otak manusia yang terdiri dari jutaan neuron dan sel di dalamnya. Otak manusia sangatlah kompleks, non-linear, dan secara parallel melakukan komputasi. Kemampuan otak manusia yang dapat mengatur unsut yang terstruktur, yang dinamakan neuron, untuk melakukan komputasi tertentu (contoh: pengenalan pola, persepsi, dan kemampuan kontrol) dan dapat melakukannya dengan sangat cepat. Dalam bentuknya yang paling umum, *neural network* adalah sebuah mesin yang di desain untuk meniru model bagaimana otak melakukan sebuah tugas atau fungsi yang diminati, jaringan pada *neural network* biasanya di implementasikan dengan menggunakan komponen elektronik atau di simulasikan pada sebuah perangkat lunak (Haykin, 1999).

Pada tahun 1957, Frank Rosenblatt mempulikasikan sebuah konsep *perceptron learning rule based* dari MCP *neuron* model pertama kali (Raschka & Olson, 2016). Dengan konsep tersebut, Rosenblatt mengajukan sebuah algoritma yang dapat secara otomatis belajar menentukan koefisien dari sebuah *weight* yang optimal dan kemudian dikalikan dengan sebuah *input features* dengan tujuan

membuat keputusan apakah sebuah neuron secara benar melakukan proses atau tidak. Algoritma tersebut dapat digunakan untuk melakukan sebuah prediksi apakah sebuah *sample* milik dari sebuah *class* atau tidak.



Gambar 2.1 Konsep dari algortima perceptron

Sumber: (Raschka & Olson, 2016)

Pada gambar diatas merupakan model dari sebuah perceptron, bagaimana sebuah perceptron menerima sebuah *input x* dan dikalikan oleh sebuah bobot *w* untuk menghitung jumlah dari X $\{x_1, x_2, \ldots, x_m\}$ dikalikan dengan W $\{w_1, w_2, \ldots, w_m\}$ vektor. Kemudian hasil dari perhitungan tersebut melewati *activation function* (*unit step function*), dimana akan menghasilkan sebuah keluaran berbentuk angka biner 1 atau -1 untuk melakukan prediksi. Dan pada tahap pelatihan model, keluaran dari proses perceptron ini digunakan untuk menghitung *error* pada prediksi dan memperbaharui *weight*. Pada model *perceptron* ini, dapat melakukan klasfikasi jika dua *classes* secara linear terbagi. Rincian pada proses model *perceptron* tersebut adalah sebagai berikut (Raschka, 2016):

1. *Activation function* pada model perceptron diatas adalah *unit step function*, untuk lebih sederhana, angka 1 (kelas positif) dan -1 (kelas negatif) yang merujuk pada dua buah kelas yang berbeda. *Unit step function* dapat di definisikan $\phi(z)$ yang mengambil kombinasi linear pada suatu *x* dan dengan *w* yang disesuaikan dimana *z* yang disebut sebagi *net input.*

$$W = [w_1 \dots w_m], X = [x_1 \dots x_m]$$

$$z = x_1 w_1 + x_2 + \cdots + x_m w_m$$

$$\sum_{j=0}^{m} x_j w_j$$

Rumus 2.3.1 Dot Product of Vector X and Vector W

2. Jika sebuah hasil *activation function* dari sebuah *feature* $x^{(i)}$, keluaran dari $\phi(z)$ lebih besar dari *threshold* θ, kemudian prediks.i kelas adalah 1 (angka positif) dan -1 adalah sebaliknya. $\phi(z) = 1$ if $z \geq \theta$, -1 jika sebaliknya.

3. Konsep di balik *MCP neuron* dan Rosenblatt's *thresholded perceptron model* adalah untuk meniru bagaimana sebuah *neuron* bekerja di dalam otak. *Perceptron model* ini memiliki konsep yang cukup sederhana untuk belajar, setiap *neuron* dalam suatu *layer* hanya perlu memperbaharui *weight* atau sebuah bobotnya jika sebuah *output* (*y'*) yang dihasilkan adalah salah. Untuk memperbaharui *weight* dapat dirumuskan sebagai berikut:

$$\boxed{w_j = w_j + \Delta w_j}$$

$$\boxed{\Delta j = \eta \left(y^i - Y'^i\right) x_j^i}$$

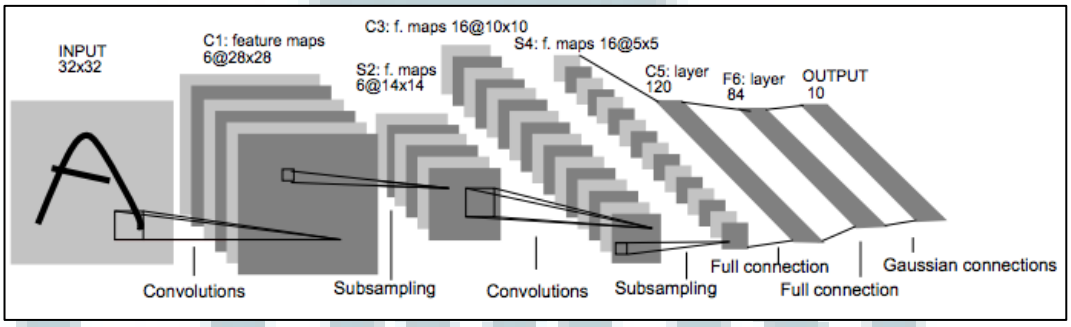
Rumus 2.3.2 Weight Update

4. Jika hasil dari multiplikasi antara sebuah *x* dengan *w* menghasilkan *output* yang salah maka *weight* akan menyesuakan nilai nya sampai *output* menghasilkan prediksi yang tepat. Pada perhitungan untuk menyesuaikan bobot atau *weight* dapat diartikan η merupakan *learning rate* yang bernilai diantara 0.0 hingga 1.0, semakin kecil nilai dari *learning rate* makan perubahan terhadap *weight* akan semakin kecil yang menyebabkan model akan semakin akurat dalam belajar, y'^i adalah kelas *label* yang diprediksi (*output*), y^i adalah kelas sesungguhnya atau *label* dari sebuah sample i^{th}.

Pada umumnya *neural network* memiliki tiga bagian yaitu *input layer*, *output layer* dan *hidden layer*. *Hidden layer* bukan merupakan *input* maupun *output.* Beberapa model *neural network* memiliki lebih dari satu *hidden layer*. *Network* yang memiliki lebih dari satu *hidden layer* biasa disebut sebagai *multi layer perceptron.* Untuk merancang *hidden layer* yang ada pada sebuah jaringan tidak cukup dengan menggunakan beberapa aturan praktis. Penentuan jumlah *hidden layer* dirancang dengan banyak pendekatan untuk mencapai sifat yang dinginkan dari sebuah network.

2.4 Convolutional Neural Network

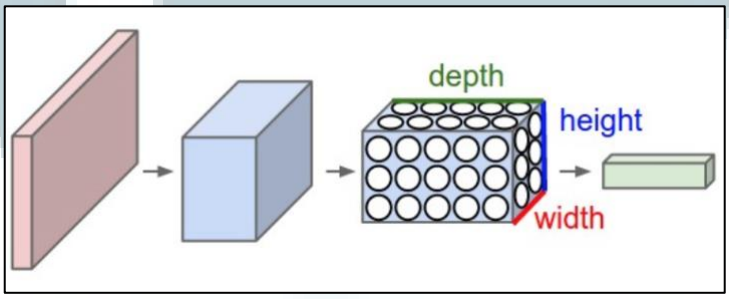
Sejak perkenalan *convolutional neural network* pada (LeCun et al., 1989), *convolutional neural network* telah mendemonstrasikan performa yang sangat baik pada sebuah tugas, salah satunya adalah klasifikasi pengenalan pada tulisan tangan dan deteksi wajah (Zeiler & Fergus, 2013). *Convolutional neural network* sangat mirip dengan *neural network* pada umumnya, model ini di bentuk atas sekumpulan neuron yang dapat belajar untuk menyesuaikan *weight* dan *bias*. *Convolutional neural network* di kombinasi dari tiga ide arstitektural untuk memastikan beberapa tingkat pergeseran dan penyimpangan / distorsi yang tetap: *local receptive field*, *shared weights*, *spatial* atau *temporal subsampling* (LeCun, Bottou, Bengio & Haffier, 1998). Salah satu arsitektur model *convolutional neural network* untuk mengenali sebuah karakter yaitu LeNet-5 terdapat pada gambar 2.2 (LeCun, Bottou, Bengio & Haffier, 1998).



Gambar 2.2 Arsitektur jaringan convolutional neural network Le-Net

Tidak seperti *neural network* pada umumnya, *neural network* akan memiliki banyak neuron jika memproses gambar yang sangat besar, tidak akan jadi masalah jika gambar yang menjadi *input* mempunyai resolusi rendah. *Convolution neural network* keuntungan dalam memproses gambar karena arsitektur yang lebih masuk

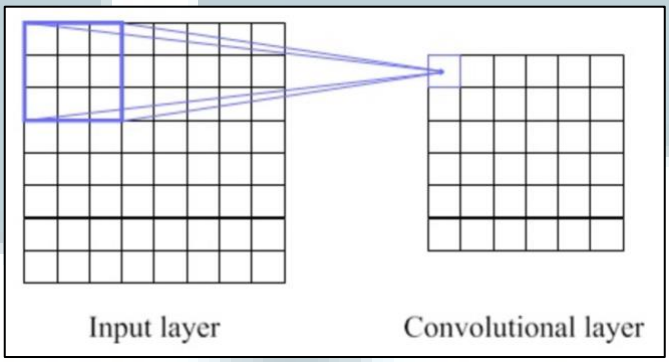
akal. Layer konvolusi tersusun dari 3 dimensi *neurons* yakitu: lebar, panjang, dan kedalaman. Kedalaman yang dimaksud pada *layer convolution* adalah dimensi pada volume *activation map*. *Convolutional neural network* memiliki arstitektur yang spesial, arsitektur CNN pada umumnya tersusun dari layer konvolusi, dan layer *sub-sampling* (*pooling layer*). Dapat dilihat pada gambar 2.3, setiap layer konvolusi mengubah *input* bervolume 3 dimensi menjadi sebuah *output* bervolume 3 dimensi. Pada gambar tersebut lapisan merah merupakan gambar yang di *input* (*feature vector*), lalu lebar dan tinggi merupakan dimensi pada gambar, dan kedalaman adalah 3 yakni: merah, hijau, biru yang merupakan kedalaman dari sebuah gambar RGB.



Gambar 2.3 Convolutional neural network mengatur neuron dalam bentuk 3 dimensi (*width, height, depth*)

Layer konvolusi mengimplementasikan operasi konvolusi, parameter dalam lapisan ini terdiri dari sekumpulan *filter* yang dapat belajar. Setiap filter merupakan spasial kecil, tetapi membentang hingga kedalaman dari volume *input*. Sebagai contoh, tipe filter pada layer konvolusi pertama memiliki 5 x 5 x 3, yaitu 5 pixels lebar dan tinggi, 3 dikarenakan sebuah gambar memiliki 3 kedalaman, saluran warna. Selama proses *forward pass*, *filter* akan melangkah (*convolve*) sepanjang lebar dan tinggi pada volume *input* yang biasa disebut *strides* (langkah) dan

menghasilkan kuantitas dari multiplikasi antara *filter* dengan *input* (*feature vector*) pada setiap posisi. Setiap kali *filter* melangkah, akan menghasilkan 2 dimensi *neuron activation map* yang memberikan respon terhadap *filter* tersebut pada setiap posisi (gambar 2.4). Setiap *neuron* memiliki lokal koneksi terhadap bidang kecil seukuran dengan ukuran *filter* yang disebut sebagai *receptive field* (*hyperparameter*). Sebagai contoh, jika sebuah *input* memiliki volume berukuran [64 x 64 x 3] (3 adalah RGB), dan jika *receptive field* (atau *filter*) berukuran 10 x 10 maka seluruh *neuron* akan memiliki [10 x 10] *weights* dan +1 bias yang sama untuk setiap volume kedalaman parameter ini biasa disebut *shared parameter*.



Gambar 2.4 Konvolusi antara filter dengan gambar yang meghasilkan receptive field

Ada tiga *hyperparameter* yang mengatur ukuran volume dari sebuah *output* dalam *convolutional neural network* yaitu: kedalaman, langkah (*stride*), *zero-padding*. Kedalaman pada volume *output* merupakan *hyperparameter*, Kedalaman pada *output* memiliki jumlah yang sama dengan jumlah *filters* yang digunakan untuk belajar sesuatu yang berbeda dari sebuah *input*. Pada umumnya, *filters* berfungsi untuk mendeteksi kehadiran dari suatu sudut, atau gumpalan warna. *Stride* merupakan jumlah langkah yang digunakan untuk *filter* berpindah sepanjang

gambar. Jika jumlah langkah semakin besar maka *ouput* yang dihasilkan gambar yang semakin kecil. *Zero-padding* merupakan *hyperparameter* untuk memberikan nilai nol di pinggir gambar. *Zero-padding* biasa digunakan untuk mendapatkan hasil *output* yang memiliki ukuran yang sama dengan ukuran gambar pada layer sebelumnya. Voume ukuran dari sebuah *output* dapat dihitung sebagaimana volume ukuran *input* (*W*), besar ukuran dari *receptive field* dalam layer konvolusi (*F*), kemudian langkah (*stride*) yang nilainya ditetapkan (*S*). Jumlah dari *zero-padding* yang digunakan (*P*). Dapat dirumuskan untuk menghitung volume ukuran *output* dengan $(W - F + 2P) / S + 1$. Jika sebuah *input* memiliki ukuran volume 32 x 32 maka $W = 32$, ukuran volume dari *filter* ditentukan 3 x 3 maka $F = 3$, *zero-padding* $P = 1$, langkah (*stride*) $S = 1$ Berdasarkan pada persamaan, dapat disimpulkan $(32 - 3 + 2) / 1 + 1 = 32$. Untuk menentukan jumlah dari *zero-padding* dapat dirumuskan oleh $P = (F - 1) / 2$ dengan syarat jumlah langkah (*stride*) $S = 1$.

Setelah proses konvolusi biasanya disisipkan *pooling layer* atau yang biasa disebut *sub-sampling layer*. *Pooling layer* berfungsi untuk mengurangi ukuran dari konvolusi yang bertujuan untuk mengurangi jumlah parameter dan mengurangi beban komputasi pada jaringan. Namun, pada *pooling layer* ini, jumlah dari kedalaman konvolusi tidak akan berubah karena metode *pooling* hanya merangkum untuk masing-masing konvolusi. Teknik *pooling* yang banyak digunakan biasanya adalah *max pooling*.

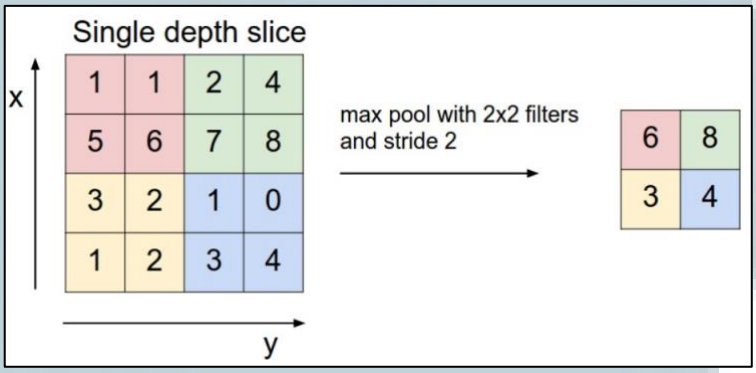
Layer terakhir dari model *convolutional neural network* ini adalah *fully-connected layer*. Seluruh aktivasi pada layer sebelumnya terhubung penuh dengan *fully-connected layer*, jika aktivasi pada konvolusi sebelumnya memiliki dimensi

lebih dari 1 x 1 maka perlunya perhitungan perkalian matriks ditambah dengan bias, untuk menjadikan parameter gambar dari 2 dimensi menjadi 1 dimensi. Namun, jika dimensi dari aktivasi pada konvolusi sebelumnya adalah 1 x 1, maka nilai sudah menjadi 1 dimensi. *Fully-connected layer* merupakan *layer* yang mampu belajar untuk mengolah *low level features* dari *layer* konvolusi dan mampu belajar secara non-linear. Pada layer terakhir ini diperlukan sebuah *activation function* seperti *sigmoid, tanh,* atau *softmax regression* guna untuk melakukan klasifikasi terhadap sebuah kelas berdasarkan probabilitas.

2.5 Max-pooling

Max-pooling adalah teknik *pooling* untuk menghasilkan *acvtivation map* pada layer *sub-sampling*. *Max-pooling* menghasilkan tingkat konvergensi yang lebih cepat dengan memilih invarian *feature* tertinggi atau superior yang meningkatkan kinerja generalisasi (Nagi et al., 2011). Dengan menggunakan *max-pooling* untuk menghasilkan *activation map* pada *sub-sampling* layer dengan cara merangkum suatu ukuran pada gambar. Hal ini dilakukan untuk mengurangi jumlah parameter untuk melakukan konvolusi berikutnya dan meningkatkan peforma serta membantu mengatur terjadinya *overfitting*, parameter model terlalu menjurus ke satu kelas saja. Untuk waktu yang sudah cukup lama, salah satu *max-pooling* yang menjadi implementasi standard dalam membangun *convolutional neural network* adalah 2 x 2 *max-pooling* (Graham, 2015).

2 x 2 *max-pooling* memiliki banyak alasan atar kepopularitasannya, cepat, dengan cepat dan mengurangi ukuran pada *hidden layer*. Ilustrasi proses 2 x 2 *max-pooling* dapat dilihat pada gambar 2.5. Setiap *pooling layer* memungkinkan untuk melihat gambar dalam skala yang berbeda. Melihat gambar pada skala yang tepat akan memudahkan pengenalan *feature* (corak) untuk mengidentifikasi objek pada suatu kelas.



Gambar 2.5 Transformasi hasil konvolusi setelah menggunakan max-pooling 2 x 2

2.6 Rectified Linear Unit (ReLU)

Dalam penelitian *computer vision* fungsi ReLU telah digunakan sebagai fungsi aktivasi dalam *neural network* standard dan sebagai unit dalam *restricted Boltzmann machine* (Dhal, Sainath & Hinton, 2013). Cara yang paling standard untuk keluaran model *f* sebagai fungsi adalah dengan $f(x) = tanh(x)$ atau fungsi *sigmoid* $f(x) = (1 + e^{-x})^{-1}$. Pada waktu pelatihan menggunakan *gradient descent,* fungsi *saturating* non-linear tersebut menunjukkan kecepatan belajar yang lebih lambat dibandingkan fungsi *non-saturating* non-linear $f(x) = \max(0, x)$

(Krizhevsky et al., 2012). Fungsi ReLU terbukti memperlihatkan kemajuan pada kecepatan pelatihan dan mendiskriminasikan peforma fungsi *sigmoid* dalam perbandingan (Tompson et al., 2014).

2.7 Local Response Normalization

Local response normalization merupakan normalisasi yang pertama kali digunakan oleh Krizhevsky, Sutskever & Hinton (2012), pengaplikasian normalisasi ini digunakan pada arstitektur CNN ketika terdapar fungsi aktivasi ReLU (Krizhevsky, Sutskever & Hinton, 2012). LRN terinspirasi dari konsep yang berada di bidang *neurobiology* yaitu *lateral inhibition* dimana sebuah *neuron* yang sedang bersemangat dapat mengurangi aktivitas dari *neuron* tetangganya. Dari konsep tersebut, manusia cenderung dapat menciptakan perbandingan atau kontras yang akan meningkatkan tanggapan pancaindera. LRN digunakan untuk mendukung hal tersebut dalam model CNN. Berikut perhitungan LRN $b^i_{x,y}$ adalah sebagai berikut (Krizhevsky, Sutskever & Hinton, 2012):

$$b^i_{x,y} = \frac{a^i_{x,y}}{(k + \alpha \sum_{j=\max\left(0,i-\frac{n}{2}\right)}^{\min\left(N-1,i+\frac{n}{2}\right)} \left(a^j_x, y\right)^2)\beta}$$

Rumus 2.7.1 Local Response Normalization

Aktivitas dari neuron di denotasikan sebagai $a^i_{x,y}$ dengan menerapkan *kernel* atau *filter* di indeks *i* pada posisi (*x, y*). Jumlah tersebut menjumlahkan dari banyaknya *n adjacent* atau *vertex* atau posisi spasial yang sama pada sebuah *kernel maps, N*

adalah jumlah *kernel* yang ada pada layer. Konstan k, n, α, dan β*hyperparameter* dimana nilai dari konstan tersebut ditentukan dengan validasi model. Berbeda dengan *local contrast normalization*, LRN lebih tepatnya adalah normalisasi kecerahan pada gambar.

2.8 Softmax Regression

Softmax regression adalah salah satu jenis dari *activation function* yang digunakan untuk melakukan klasifikasi pada jumlah kelas lebih dari dua dan digunakan untuk menghasilkan *output* yang tepat berdasarkan probabilitas. *Softmax regression* merupakan generalisasi dari *logistic regression* (*sigmoid*) dimana untuk menangani kelas yang banyak. Pada *logistic regression*, dapat diasumsikan bahwa *labels* adalah sebuah *binary*: $y^{(i)} \in \{0, 1\}$ dimana hanya ada dua buah kombinasi pada *logistic regression* dan hanya dapat digunakan untuk menentukan maksimal dua buah kelas, namun untuk *softmax regression*, *labels* merupakan sebuah sekumpulan nilai: $y^{(i)} \in \{0, 1, \ldots, K\}$ dimana K merupakan jumlah dari banyak kelas, untuk menentukan *label* sebuah kelas pada *softmax regression* diperlukan beberapa kombinasi dari $y^{(i)} \in \{0, 1, \ldots, K\}$ dimana di setiap kombinasi angka satu berada di deret yang berbeda, contoh: {1, 0, 0, 0} merupakan *label* sebuah kelas dari empat kategori kelas. Sasaran yang ingin dicapai dari *logistic regression* adalah menemukan model yang paling sesuai untuk menggambarkan hubungan antara variabel hasil dan sekumpulan prediktor. *Logistic regression* menghasilkan sebuah formula koefisien untuk melakukan

prediksi *logit transformation* pada probabilitas adanya variabel hasil. Perthitungan *sigmoid* dapat dihitung dengan rumus berikut (Bishop, 2006).

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

Rumus 2.8.1 Logistic Regression

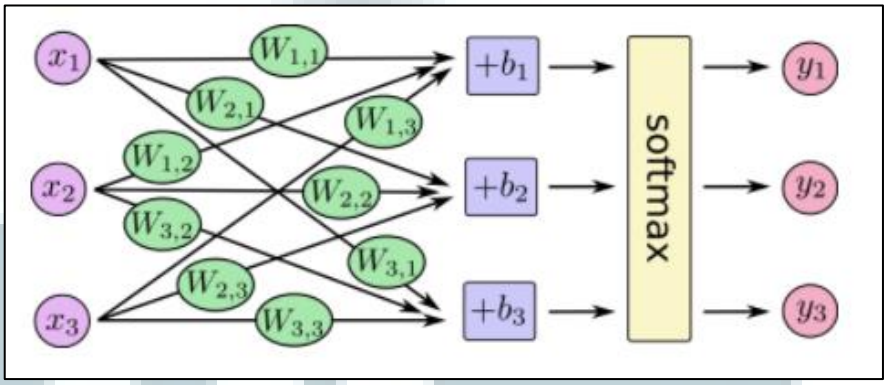
$\sigma(x)$ merupakan *logistic regression* (*sigmoid*), dimana e^{-x} adalah nilai eksponensial dari *x* yaitu logits, Fungsi eksponensial melibatlan sebuah nilai konstan yaitu 2.71828182845904 (dibulatkan 2.72), lalu kita gunakan nilai 2.72 untuk dipangkatkan, jadi untuk e^x sama dengan 2.72^x, sebaliknya untuk *natural logarithm* jika $\ln(2.72)^x$ adalah *x*. Kemudian $\sigma(x)$bisa disebut sebagai *logistic regression* yang juga biasa dikenal dengan sebutan *logit function*. *Logit function* ini mewakili log (kebalikan) dari rasio probabilitas terhadap dua buah kelas (Bishop, 2006). Karena *logistic regression* hanya dapat melakukan klasifikasi terhadap dua jenis kelas saja. Untuk menangani keterbatasan tersebut *softmax regression* digunakan untuk melakukan klasifikasi lebih dari dua kelas atau yang biasa disebut *multilabel classification*. Fungsi *softmax* adalah sebagai berikut (Bishop, 2006).

$$\varphi_{softmax}(z^i) = \frac{e^{z^i}}{\sum_{j=0}^{k} e^{z_j^i}}$$

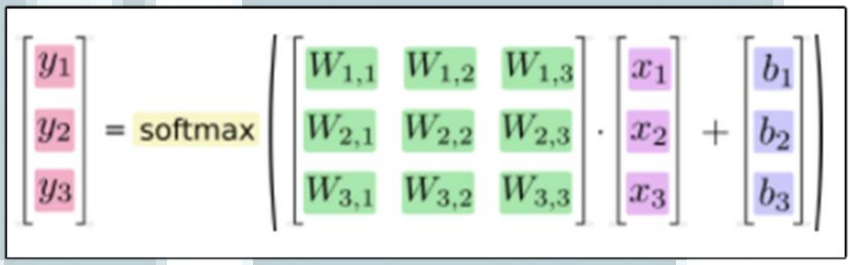
Rumus 2.8.2 Softmax Regression

Persamaan diatas adalah normalisasi eksponensial yang dikenal sebagai *softmax function*, fungsi tersebut menghitung distribusi probabilitas dari seluruh logits z^i. Masing-masing dari z^i akan dicari nilai lognya dari jumlah nilai log pada seluruh z^i. Hasil yang dikeluarkan oleh *softmax function* merupakan sebuah deret

dari probabilitas nilai log pada masing-masing z^i dan deret tersebut akan merepresentasikan distribusi seperti *one-hot encoded label*.



Gambar 2.6 Kalkulasi *softmax* pada *neural network*



Gambar 2.7 Matriks multiplikasi *softmax* pada *neural network*

Pada gambar 2.6 menunjukkan bagaimana *neural network* memperoleh tiga buah output dengan masing-masing nilai probabilitas *output* tersebut. Untuk dapat menghitung distribusi probabilitas untuk tiga buah kelas, dari masing-masing multiplikasi matriks yang ada pada gambar 2.7, akan menjadi *input* untuk persamaan *softmax* yang ada pada rumus 2.8.2. Setiap kelas *y* akan dihitung probabilitas nya dengan membagi nilai eksponensial untuk setiap logits dengan jumlah nilai logits pada hasil multiplikasi matriks. Dengan demikian, hasil yang didapatkan adalah nilai dari 0 sampai 1.

Softmax regression sebagai *activation function* memungkinkan menggunakan *cross-entropy* untuk menghitung *error* pada hasil distribusi probabilitas (Dunne & Campbell, 1997). *Cross-entropy* merupakan metode yang kuat sebagai alat untuk menyelesaikan permasalahan estimasi, kombinatorial dan optimalisasi yang sulit (Kroese, Rubinstein, Cohen, Porotsky & Taimre, 2013). Perhitungan *error* pada *softmax activation* dapat dilakukan dengan rumus *cross-entropy loss* sebagai berikut (Nielsen, 2015).

$$\boxed{L(a, y) = -(ylog(a) + (1 - y)\log(1 - a))}$$

Rumus 2.8.3 Cross Entropy Loss

Persamaan diatas merupakan fungsi untuk menghitung *loss* atau *distance* antara hasil estimasi yang berupa nilai log dengan *true label* atau bisa disebut dengan probabilitas sebenarnya. Pada rumus 2.8.3 y adalah *true label* yang merupakan elemen dari angka 0 atau 1, dan a adalah nilai log probabilitas dimana nilai tersebut dapat bernilai 0 hingga 1. Ketika nilai dari *true label* adalah 1, nilai a yang dinginkan juga harus besar atau mendekati 1 dan sebliknya jika *true label* memiliki nilai 0, maka nilai dari a yang dinginkan harus kecil atau mendekati angka 0. Jika $y = 0$ maka $L(a, y) = -(ylog(a)$, jika $y = 1$, maka $L(a, y) =$ -log$(1 - a)$. Pengunaan *cross entropy loss* juga dapat digunakan untuk menghitung *loss* atau distance untuk *multilabel classification*. Perhitungan *loss* dapat dihitung menggunakan rumus berikut (Nielsen, 2015).

$$L(w) = -\sum_{n=1}^{N} y_n \log(a_n) + (1 - y_n)\log(1 - a_n)$$

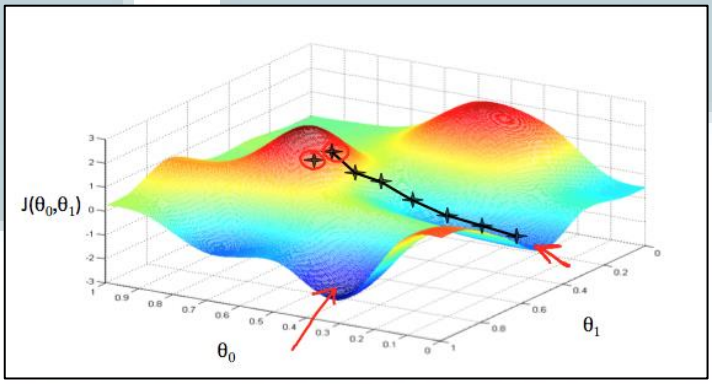
Rumus 2.8.4 Softmax Cross Entropy Loss

Persamaan diatas $L(w)$ merupakan fungsi untuk mencari *loss* atau distance dari *multilabel distribution* atau bisa disebut mencari *loss* dari hasil *softmax regression*. Perhitungan yang dilakukan adalah dengan cara menjumlahkan seluruh loss dari masing-masing estimasi a_n hasil *softmax* terhadap *one-hot encoded* pada masing-masing distribusi *true label* y_n. Jika model menggunakan *stochastic gradient descent*, perhitungan *cost* dapat dilakukan dengan mencari rata-rata dari seluruh *loss* yang ada pada *mini batch*. *Cost* dan *loss* memiliki fungsi yang sama dimana nilai tersebut merupakan *error rate* atau distance dari nilai estimasi dengan nilai sebenarnya. Fungsi *cross entropy* dapat diturunkan dengan *gradient descent* sehingga, *cost* atau distance akan semakin kecil selama pelatihan model. Jika model hanya menggunakan satu *activation fucntion* pada ujung model tanpa menggunakan fungsi untuk meghitung *loss*, model tersebut tidak dapat menghitung turunan atau *slope* yang diperlukan karena model tidak memiliki hubungan dengan nilai sebenarnya atau distribusi probabilitas sebenarnya. Dengan kata lain, model tidak memiliki hubungan dengan variabel nilai kebenaran y_n.

2.9 Gradient Descent Optimization

Gradient descent adalah pendekatan yang digunakan untuk mencari nilai optimal pada *weight* dan bias sehingga dapat meminimalisir nilai *error* pada sebuah

cost function. Gradient descent memperbaharui *weight vector* melalui arah turunan dari *error function* (*slope / rate of change*) (Bishop, 2006). Penggunaan *gradient descent* digunakan untuk memberikan kapabilitas sebuah mesin untuk dapat belajar menyesuaikan parameter agar mencapai *local minimum* yang merupakan titik terendah pada sebuah *cost function* dapat dilihat pada gambar 2.8, *local minimum* dapat dicapai dengan menurunkan hasil dari *cost function*. Untuk dapat mencapai nilai minimum tersebut pada sebuah *cost function* diperlukan prosedur numerik iteratif, dimana untuk setiap iterasi *weight* akan diperbaharui *backward computation*.



Gambar 2.8 Gradient descent yang mencari local minimum dari sebuah fungsi

Turunan pada algoritma *backpropagation* cukup sederhana. Turunan yang dilakukan ini mengikuti penggunaan aturan *chain rule* dan *product rule* dalam kalkulus diferensial. Penerapan aturan ini bergantung pada *activation function* yang digunakan untuk menghitung *error* dalam sebuah *cost function*. Pada *forward propagation* output yang dihasilkan memerlukan nilai $z = w_1 x_1 + w_2 x_2 + b$ sebagai parameter dari sebuah *activation function*, misalnya *activation function* yang digunakan adalah *logistic regression* yakni $a = \alpha(z)$, lalu output terkahir dari

forward propagation dalam sebuah *training computation* adalah nilai dari *loss function* yakni $\theta(a, y)$. Dari hasil perhitungan *forward propagation* fungsi dari *loss function* diturunkan dengan aturan *chain rule* dan *product rule* bisa juga dengan *partial derivatives*. Tahapan dari turunan tersebut adalah sebagai berikut (Sebastian Raschka, 2016).

$$\boxed{\Delta a = \frac{\Delta \theta(a,y)}{\Delta a} = -\frac{y}{a} + \frac{1-y}{1-a}}$$

$$\boxed{\Delta z = \frac{\Delta \theta(a,y)}{\Delta z} = \frac{\Delta \theta}{\Delta a} \times \frac{\Delta a}{\Delta z} = a - y}$$

$$\boxed{\Delta w_i = \frac{\Delta \theta}{\Delta w} = x_i \times \Delta z}$$

$$\boxed{\Delta b = \Delta z}$$

Rumus 2.9.1 Chain Rule and Product Rule for Logistic Regression

Dengan menggunakan aturan *chain rule* dan *product rule* serta *partial derivatives* untuk mendapatkan nilai turunan untuk parameter yang spesifik, Δa di denotasikan sebagai turunan dari fungsi *loss* dengan parameter *a,* Δz merupakan turunan dari fungsi aktivasi dengan aturan *chain rule* untuk mendapatkan nilai turunan dari fungsi *z,* dan Δw adalah nilai turunan dari fungsi *z* dengan menggunakan aturan *product rule,* Δb juga meruapkan turunan dari fungsi *z* dimana nilai Δb sama dengan Δz. Perhitungan ini akan dilakukan berulang kali hingga mendapatkan nilai rata-rata untuk parameter *weight* dan *bias* dengan cara *backpropagation*. Setelah mendapat nilai dari Δw dan Δa. Parameter sebelumnya diperbaharui dengan melakukan substraksi antara nilai parameter sebelumnya

dengan rata-rata dari nilai turunan masing-masing parameter. Dengan *gradient descent* mesin dapat memperbaharui nilai parameter dan mendapatkan parameter yang paling optimal untuk mendapatkan *error rate* yang paling rendah.

2.10 Exponentially Learning Rate Decay

Exponential learning rate decay merupakan teknik untuk mengestimasikan *learning rate* yang baik pada setiap iterasi pada *gradient descent* (Zeiler et al., 2012). Teknik ini digunakan untuk mempercepat *learning rate* saat masih jauh dari *local minima* atau memperlambat pada saat dekat dengan *local minima*. Salah satu teknik ini digunakan adalah ketika *gradient descent* sudah dekat dengan *local minima* tetapi nilai dari parameter akan secara bolak-balik mendapat nilai yang berlebihan sehingga tidak mencapai *local minima*. *Exponentially learnig rate decay* berfungsi untuk mengatasi masalah tersebut dengan mengurangi nilai *learning rate* dimana hal ini dapat dilakukan secara manual ketika akurasi telah mencapai titik yang tertinggi yang dinginkan atau dengan menjadwalkan kapan nilai *learning rate* akan diturunkan setelah jumlah *epochs* dan data yang telah diproses. Untuk menerapkan teknik ini, model akan menambahkan satu *hyperparameter* untuk mengendalikan tingkat penurunan *learning rate*. Perhitungan dapat dilakukan melalui persamaan berikut (Andrew Ng DeepLearning.ai) $\alpha = \lambda^{epochnum} \times \alpha_0$. Pada persamaan tersebut α melambangkan *learning rate* yang diturunkan, λ adalah parameter yang mengontrol tingkat penurunan *learning rate*, *epoch num* adalah konstan yang menetukan kapan *learning rate* akan diturunkan setelah sejumlah

epochs pada nilai konstan, dan α_0 adalah *learning rate* yang digunakan *gradient descent*.

2.11 K-Fold Cross Validation

Cross validation adalah teknik yang digunakan untuk menilai bagaimana performa sebuah *classifier* ketika melakukan klasifikasi terhadap objek baru (Moreno, Sáez & Herrera, 2012). Satu iterasi dalam *cross-validation* melibatkan pembagian data kedalam dua subset, *training set* untuk melatih *classifier* dan *test set* untuk menguji sebuah *classifier*.

Dalam *k-fold cross validation*, dataset dibagi menjadi sebanyak *k* subset dimana *k* adalah jumlah eksperimen. Setiap subset yang terbagi memiliki jumlah data dan ukuran yang sama. Pelatihan dilakukan sebanyak *k*, sejumlah *k - 1 / k* digunakan untuk melakukan *training* dan sisanya digunakan sebagai set untuk melakukan pengujian. Perkiraan tingkat akurasi pada *cross validation* dilakukan degan cara jumlah klasifikasi yang tepat dibagi dengan jumlah subset dalam dataset yang digunakan. Jika $D_{(i)}$ adalah test set yang memiliki objek $x_i = \{v_i , y_i\}$, maka akurasi *cross validation* adalah sebagai berikut (Kohavi, 1995).

$$Acc_{cv} = \frac{1}{k} \sum_{\{vi,xi\} \in D} \delta(\iota\left(\frac{D}{D_i, v_i}\right), y_i)$$

Rumus 2.11.1 Akurasi dari K-Fold Validation

Pada rumus tersebut δ(*delta*) merupakan perubahan atau penjumlahan terhadap ι yang merupakan fungsi identitas dimana seluruh elemen dari $\frac{D}{D_i, v_i}$ dipetakan kedalam dirinya sendiri, kemudian *vi* adalah *output* yang dihasilkan dari masing-masing data dalam subset $D_{(i)}$ dan y_i adalah *label* untuk membandingkan ketepatan v_i.

2.12 Leave-One-Out Cross Validation

Leave-one-out cross validation adalah metode *cross validation* yang paling ekstrem dimana jumlah *k* sama dengan jumlah pola *training* (Cawley & Talbot., 2003). Dalam *leave-one-out*, jika dalam suatu *training set* memiliki *N* buah banyak objek, maka *k = N* dimana satu elemen nantinya akan dipisah dari *training set* tersebut x_i yang akan digunakan untuk melakukan pengujian. Validasi ini diulang hingga banyaknya *k* dengan pola yang sama dimana setiap elemen harus pernah menjadi bahan uji. Untuk mengukur probabilitas *error* pada pengujian *leave-one-out cross validation* pada suatu *training set*:

$$Ep^{l-1}_{err} = \frac{1}{E(x_1 y_1, \dots, x_l y_l)}$$

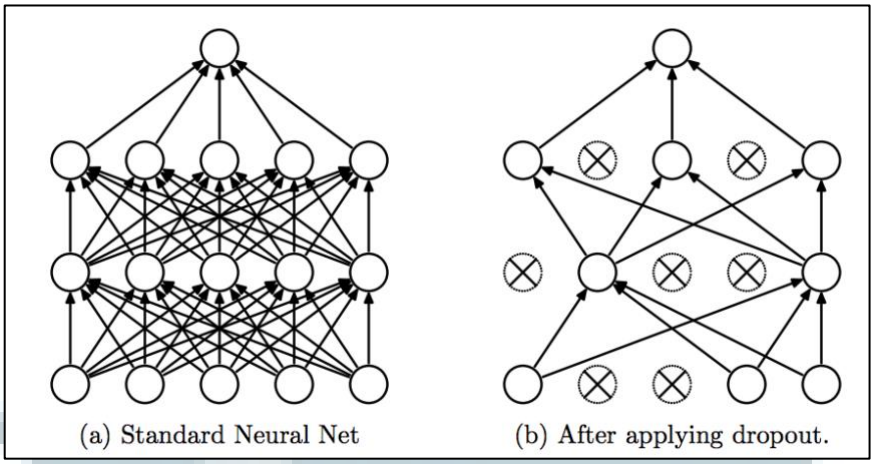
Rumus 2.12.1 Error Average Leave One Out

Pada persamaan tersebut Ep^{l-1}_{err} adalah probabilitas *error* pada pengujian dalam sebuah set yang di uji pada sebuah *trained sample* berukuran *l-1*, kemudian $x_1 y_1, \dots, x_l y_l$ adalah jumlah *errors* dalam *leave-one-out* validation, *l* adalah jumlah seluruh elemen yang ada di dalam set (menggunakan *l* aturan yang berbeda). Meski

dalam pembuktian diatas membuat *leave-one-out* menjadi pilihan yang baik untuk sebuah estimator saat memperkirakan *generalization error*, namun validasi ini sangat berat dalam hal komputasi dikarenakan validasi ini memerlukan untuk menjalankan algoritma pelatihan sebanyak *l* kali.

2.13 Dropout Regularisasi

Dropout regularisasi adalah teknik yang digunakan untuk mencegah terjadinya overfitting pada sebuah model dan memberikan sebuah solusi yang kira-kira menggabungkan secara eksponensial macam-macam *neural network architecture* secara efisien (Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdinov, 2014). Cara kerja *dropout* yaitu dengan memutuskan unit pada *neural network*. Dengan memutuskan unit, tidak selamanya unit tersebut terputus dari jaringan, termasuk jaringan masuk dan keluar yang dapat dilihat pada gambar 2.9. Penentuan unit mana yang akan di putus dilakukan secara acak, secara sederhana unit yang dipertahankan bergantung pada probabilitas tetap *p* yang independen terhadap unit lainnya. Penentuan probabilitas dapat ditentukan berdasarkan validasi atau bisa ditetapkan tidak kurang dari 0.5.



Gambar 2.9 Hasil fully-connected layer menggunakan dropout

Sumber: (Srivastava, 2014)

Pelatihan model menggunakan *dropout* regularisasi, pada umumnya adalah memberi nilai nol pada suatu unit agar unit tersebut tidak dapat belajar. Penggunaan metode ini pada saat pelatihan model bergantung pada satu *hyperparameter* yaitu *keep probability* dimana variabel tersebut menyimpan nilai probabilitas unit yang ingin ditahan pada suatu layer, jumlah unit diputuskan adalah N - (N × *p*) dimana N adalah jumlah unit atau neuron pada layer dan *p* adalah probabilitas unit yang ingin dipertahankan. Setiap iterasi dalam pelatihan model akan melakukan *drop-out* secara acak dimasing-masing layer, hal ini mendukung agar seluruh neuron tetap dapat belajar. Untuk kasus dimana saat validasi dilakukan *dropout* tidak diperlukan, probabilitas untuk menahan unit adalah $p = (N \times 100\%)$.

2.14 L2-Regularization Error

L2-*regularization* atau sering disebut dengan *least squares error* merupakan salah satu teknik standard yang digunakan untuk mencegah *overfitting*,

dimana sebuah syarat ditambahkan pada nilai *weight* yang besar dan kemudian ditambahkan ke rata-rata hasil dari *loss function* (Koh, Kim & Boyd, 2007). Regularisasi ini berfungsi untuk mempertahankan nilai *weight* agar tidak memiliki nilai yang terlalu besar. Penerapan L2-*regularization* yaitu dengan menambahkan *loss function* dengan nilai *weight* kuadrat.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y\right) + \frac{\lambda}{2m}||w||_2^2$$

Rumus 2.14.1 L2-Regularization in Cost Function

$$||w||_2^2 = \sum_{j=1}^{N_x} w_j^2$$

Rumus 2.14.2 Square of all Weight

Persamaan diatas merupakan fungsi untuk mengecilkan nilai *cost* ($J(w,b)$pada sebuah model (Tompson, Stein, Lecun & Perlin, 2014), pada umumnya fungsi tersebut menghitung rata-rata dari *loss* atau fungsi logaritma antara *y* dengan $\hat{y}$ pada kumpulan set tertentu ($\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y\right)$). L2-*regularization* dilakukan dengan cara menambahkan rata-rata dari *loss* dengan nilai *weight* kuadrat dimana pada perhitungan kuadrat $||w||_2^2$ adalah hasil perhitungan kuadrat untuk setiap nilai *weight*. L2-*regularization* memiliki satu buah *hyperparameter* yang digunakan sebagai bobot untuk menghitung hasil perhitungan kudrata pada nilai *weight* yaitu lambda λ.

Objektif dari fungsi L2-*regularization* adalah halus dan cembung (Koh et al., 2007). Artinya, L2-*regularization* perubahan nilai *weight* akan lebih kecil pada

saat mencari titik lokal minimum dan hasil dari perubahan nilai *weight* yang kecil akan mencegah nilai *weight* yang besar sehingga mengurangi *overfitting* pada model.

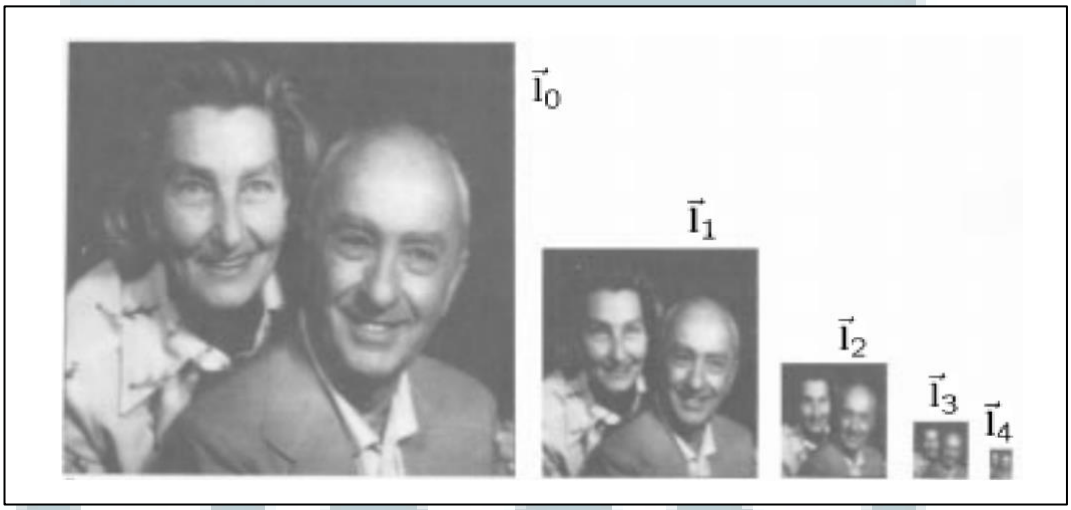
2.15 TensorFlow

TensorFlow adalah sebuah sistem *machine learning* yang beroperasi pada sistem berskala besar dan kondisi yang berbeda-beda (Abadi et al., 2016). Tensorflow menggunakan *graph dataflow*, *nodes* dalam *graph* mewakili operasi matematis, dimana pada ujung *graph* mewakili sebuah *multidimensional data arrays* (tensor). Komputasi tersebut dapat dilakukan lintas perangkat komputasi, *multicore Computer Processing Unit*, *Graphic Processing Unit,* dan *Tensor Processing Unit*. Tensorflow sendiri merupakan library *open source* yang tersedia untuk bahasa pemrograman Python, C++, Java, dan Go.

TensorFlow awalnya dikembangkan oleh para peneliti dan insinyur Google Brain untuk keperluan penelitian *machine learning* dan *deep neural network*, namun TensorFlow sudah mampu di aplikasikan secara umum dibanyak domain. TensorFlow memperbeolehkan para programmer untuk bereksperimen dalam membangun sebuah model *machine learning* dan *deep neural network.*

2.16 Gaussian Pyramid

Gaussian pyramid merupakan salah satu teknik *image processing* yang dikenalkan pada penelitan Adelson et al. 1995. Teknik ini digunakan untuk memudahkan deteksi pola atau objek pada, dimana objek dalam gambar dapat memiliki ukuran yang beragam (Adelson, Anderson, Bergen, Burt & Ogden, 1984). *Gaussian pyramid* juga merupakan alat untuk membuat sebuah gambar menjadi multi resolusi dengan menghilangkan sedikit detail. Teknik ini menggunakan *gaussian distribution* sebagai distribusi nilai *weight* pada sebuah *kernel* atau menggunakan *binomial approximation* dengan segitiga paskal.



Gambar 2.10 Hasil dari image pyramid menghasilkan 5 sampel dari proses subsample

Sumber: (Adelson, 1984)

Terdapat dua langkah untuk melakukan *gaussian pyramid* yaitu (1) Menghaluskan gambar dengan mengkonvolusi gambar dengan sebuah *filter* dimana ukuran *filter* dapat ditentukan secara bebas tetapi bobot dari pada nilai *weight* dalam

filter tersebut bergantung pada sebuah koefisien sigma σuntuk menentukan distribusi normal pada *filter*. Jika nilai sigma semakin besar maka distribusi normal memiliki standard deviasi yang semakin tinggi (2) *Downsample* gambar dengan mengurangi dimensi gambar menjadi setengah kurang (3) Kemudian diulangi sampai dimensi terkecil yang telah ditentukan. Gambar 2.8 menunjukkan hasil dari *gaussian pyramid* dengan 5 buah *tap*. Konvolusi untuk menghaluskan gambar pada *gaussian filter* dapat dilakukan dengan dua yaitu dengan ukuran *kernel* 1D dan 2D. Untuk ukuran 1D dapat menggunakan segitiga paskal sebagai filter tetapi untuk 2D dapat menggunakan distribusi normal dengan standatd deviasi yang telah ditentukan.

2.17 Mean-Shift

Mean-shift adalah adalah sebuah metode yang secara iteratif melakukan perhitungan *mode* terdekat dari sebuah sampel (Ning, Zhang, Zhang & Wu, 2012). Algoritma *mean shift* sering digunakan untuk melakukan pelacakan objek pada video. Dengan menggunakan histogram warna untuk menandai objek yang menjadi target dan perhitungan *mean-shift* akan berjalan secara iteratif sehingga ketika objek yang menjadi target berpindah posisi, *mean shift* yang akan mencari *center of mass* dari objek yang menjadi target tersebut. Berikut rumus *mean shift* (Kheng, 2011).

$$m(x) = \frac{\sum_{i=1}^{n} K(x - xi)xi}{\sum_{i=1}^{n} K\ (x - xi)}$$

Rumus 2.17.1 Mean Shift

Pada persamaan diatas, m(x) adalah mean shift dimana K(x) adalah fungsi kernel yang mengindikasi seberapa besar kontribusi x terhadap estimasi rata-rata. Lalu x_i adalah suatu poin data pada sebuah set yang ada pada ruang euclidean X. Dengan menggunakan persamaan tersebut, pelacakan sangat mungkin dilakukan karena sifat dari mean shift selalu mencari poin data yang paling mirip dengan objek target.

2.18 Penelitian Terdahulu

2.18.1 Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Neural Networks

Penelitian ini bertujuan untuk memberikan solusi terhadap kesulitan dalam menyimpulkan pose tangan secara kontinu dengan cara membangun *database* label kebenaran data dengan proses otomatis dan melatih sebuah sistem yang mampu secara *real-time* menyimpulkan pose. Solusi yang dihasilkan dari penelitian ini dirasa mampu diaplikasi ke objek yang memiliki jangkauan artikulasi yang luas. Metode pada penelitian ini memiliki *latency* yang kecil terhadap kesetaraan dengan satu *frame* pada video, kuat dalam merincikan, tidak memerlukan penanda khusus, dan mampu menangani objek yang serupa seperti jari tangan (Tompson, Stein, Lecun & Perlin, 2014).

Metode pada penelitian ini juga dapat di generalisasi untuk melacak objek yang dapat diartikulasikan dengan tiga kebutuhan: (1) Objek yang dilacak dapat dimodelkan sebagai 3D *bones mesh*, (2) *binary classifier* dapat dibuat untuk memberi label pada piksel gambar pada sebuah objek, dan (3) proyeksi dari ruang pose untuk memproyeksikan gambar 2D secara mendalam kira-kira satu-banding-satu. Model dalam penelitian ini digunakan untuk memberi kedalaman sebuah video sebuah label yang ditangkap secara langsung dari pengguna. Data tersebut digunakan untuk melatih *Randomized Decision Forest* untuk segmentasi gambar dan juga *convolutional neural network* untuk menyimpulkan posisi secara *real-time*. Metodelogi dari penelitian ini terdiri dari 3 tahap: (1) Segmentasi gambar dengan *randomized decision forest*, (2) Ekstraksi *feature* menggunakan *convolutional neural network*, dan (3) *Inverse kinematics pose recovery* menggunakan algortima untuk menyimpulkan derajat kebebasan yang tinggi pada sebuah pose.

Randomized Decision Forest digunakan sebagai binary classifier yang berperan untuk mendeteksi tangan pada sebuah gambar. RDF pada penelitian ini di desain untuk mengklasifikasikan setiap piksel pada kedalaman gambar dimana piksel tersebut termasuk tangan atau latar. Setiap *tree* yang ada dalam RDF mengandung sebuah set keputusan sekuensial deterministik, yang disebut dengan *weak nodes*, *tree* tersebut berfungsi untuk membandingkan kedalaman yang relatif dalam piksel yang terletak pada pengimbang yang telah ditetapkan. Untuk melatih sistem RDF pada penelitian ini, dilakukan *labelling* untuk pelatihan RDF dimana sebuah tangan pengguna di warnai dengan warna merah terang dan menggunakan

HSV-base distance metric sederhana untuk mengestimasi *coarse* (kesat) pada tangan yang di beri label. Pelatihan RDF dilakukan dengan mecoba *weak node* secara acak dari sebuah kumpulan fungsi, sama dengan yang dijelaskan Shotton et al. 2011. Setiap node dalam *decision tree* mengevaluasi,

$$\boxed{I\left(u+\frac{\Delta u}{I(u,v)}, v+\frac{\Delta v}{I(u,v)}-I(u,v)\right)\geq d_i}$$

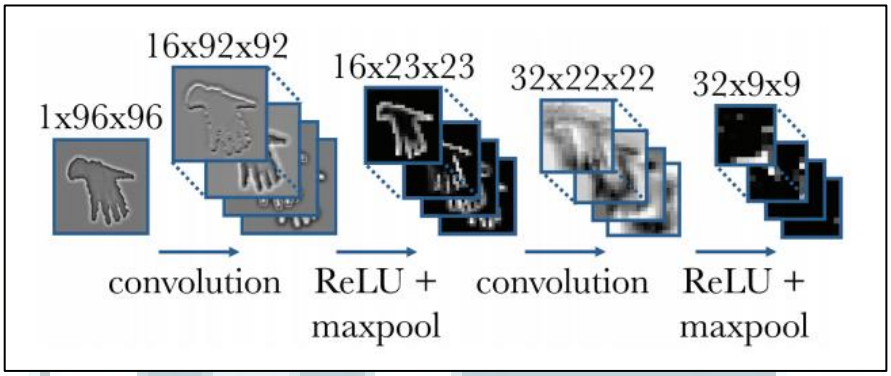
Rumus 2.18.1.1 Evaluasi Node pada Decision Tree

Dimana $I(u, v)$ adalah nilai dari kedalaman piksel dari sebuah gambar, Δu dan Δv adalah *learned pixel offset*, dan d_i adalah *learned depth threshold*. Untuk mencapai performa yang baik dalam klasifikasi dibutuhkan rentang *pixel offset* yang dinamis, hal ini disangka karena jalur keputusan dari sebuah *tree* memerlukan informasi global dan geometri lokal untuk melakukan segmentasi latar tangan secara efisien.

Setelah tangan telah berhasil di segmentasi dengan RDF *binary classification*, langkah berikutnya adalah membuat database sensor gambar RGBD yang merepresentasikan jangkauan luas gestur tangan dengan perkiraan kebenaran label yang akurat terhadap parameter ruas pada setiap gambar yang nantinya akan digunakan untuk melatih *ConvNet*. Perkiraan kebenaran label yang diinginkan mengandung vektor 42 dimensi yang menggambarkan *degree-of-freedom* pada sebuah pose. Penelitian ini menggunakan *direct search method* untuk menyimpulkan parameter dari pose berdasarkan pendekatan dari Oikonomidis, Kyriazis & Argyros. (2011). Algoritma yang digunakan diadopsi dari pendekatan Oikonomidis, Kyriazis & Argyros. (2011) dan di modifikasi, untuk penelitian ini adalah sebagai berikut: dimulai dengan memperkirakan pose tangan, kemudian

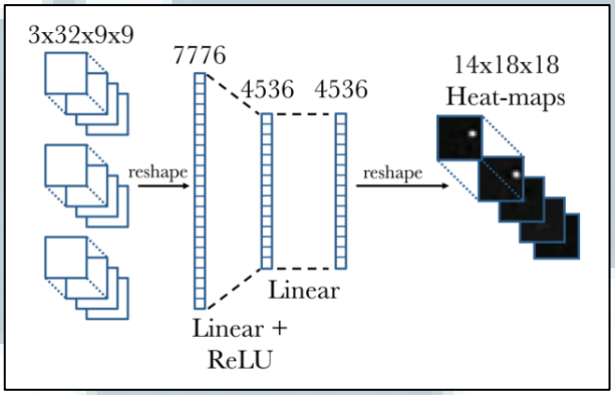
sebuah gambar sintetis di *render* dan di bandingkan dengan kedalaman gambar sebenarnya menggunakan *scalar objective function*. Gambar sintesis di *render* dengan OpenGL-*based framework*. Pada penerapan, estimasi pose tangan menggunakan pose pada *frame* sebelumnya saat sesuai dengan urutan *frame* yang direkam. *Particle swarm optimization* dengan *partial randomization* (*PRPSO*) *direct search method* digunakan untuk menyesuaikan nilai koefisien untuk mencari *best-fit* untuk pose tangan yang akan meminimalisir nilai objektif dari fungsi.

Setelah tahapan sebelumnya sudah selesai, penelitian ini menggunakan *convolutional neural network* untuk mengekstraksi *feature*, dimana output yang dikeluarkan oleh model ini adalah *heatmap.* Model *convlutional neural network* pada penelitian ini memiliki input berupa gambar multi resolusi yaitu: 96x96, 48x48, dan 24x24 piksel gambar. Gambar 2.11 dan gambar 2.12 merupakan model CNN pada penelitian ini, model tersebut memiliki 2 layer konvolusi dan masing-masing layer menggunakan ReLU sebagai fungsi aktiviasi dan maxpool, kemudian diikuti dengan 2 layer *fully-connected* dimana layer pertama menggunakan ReLU untuk fungsi aktivasi. Output yang dihasilkan dari model CNN ini adalah *heatmap* yang merupakan prediksi ruas-ruas tangan pada gambar.



Gambar 2.11 Contoh hasil konvolusi dari model CNN penelitian Tompson, Stein, Lecun & Perlin (2014)

Sumber: (Tompson, 2014)



Gambar 2.12 Arsitektur convolutional neural network dari penelitian Tompson, Stein, Lecun & Perlin (2014)

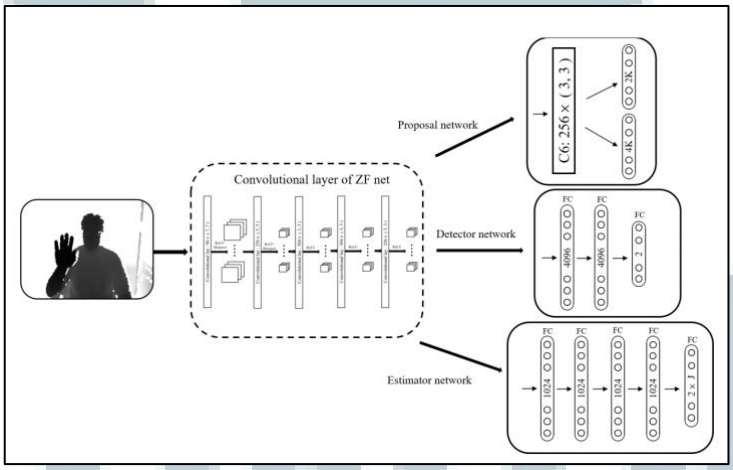
Sumber: (Tompson, 2014)

Setelah *heatmap* berhasil dihasilkan melalui model CNN, pengembalian bentuk pose tangan dalam 3D dilakukan dengan menggunakan algortima *inverse kinematic* yang menggunakan *heatmap* sebagai parameter untuk melakukan algrotima tersebut. Hasil dari penelitian berhasil membuktikan bahwa *heatmap* dapat digunakan untuk membentuk ulang pose, namun akurasi dari sistem ini bergantung pada resolusi dari *heatmap* itu sendiri dan akan tetapi untuk

menghasilkan *heatmap* memerlukan banyak komputasi dan sistem hanya dapat melacak dua tangan jika tangan tidak berinteraksi dengan objek bukan tangan seperti pulpen atau seseorang sedang membuat kerajinan.

2.18.2 Deep Learning for Integrated Hand Detection and Pose Estimation

Penelitian ini membuat suatu hal baru yang mengintegrasikan deteksi tangan manusia dan estimasi pose tangan kedalam satu aliran. Penelitian ini menggunakan NYU *hand datasets* sebagai sampel untuk pelatihan model yang dibangun dalam penelitian ini. Model yang dibuat pada penelitian ini terdiri dari satu *convolution network* dan tiga *neural network* yang terpisah dan memiliki tugas yang berbeda.



Gambar 2.13 Arsitektur convolutional neural network penelitian Chen, Wu, Hsieh & Fu (2016)

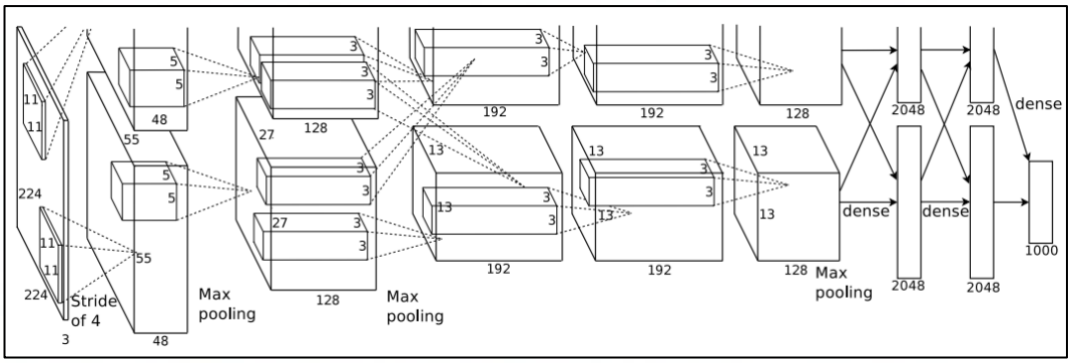
Sumber: (Chen, 2016)

Arstitektur CNN yang digunakan dalam penelitian ini memiliki 5 layer konvolusi dan 3 *neural network* yaitu *proposal network*, *detection network*, dan *estimator network*. *Proposal network* berperan sebagai jaringan yang nantinya akan mencari objek yang memiliki nilai dominan terhadap tangan manusia dimana objek tersebut memiliki probabilitas tertinggi terhadap tangan manusia. Setelah *proposal network* berhasil mencari objek yang dominan, objek tersebut akan diberi sebuah kotak penanda yang disebut *anchor*. *Detector network* bekerja mirip dengan *proposal network*, tugas dari jaringan ini adalah mendeteksi apakah objek yang sudah petakan oleh *proposal network* adalah sebuah tangan atau bukan. Setelah jaringan deteksi berhasil mendeteksi bahwa objek tersebut merupakan tangan, *estimator network* akan bertugas untuk mengestimasi pose tangan berdasarkan ruas tangan pada gambar.

Pelatihan *estimator network* dilakukan dengan menginisialisasi nilai seluruh *kernel* sama dengan nilai *kernel* pada *detector network* setelah pelatihan model. Cara untuk mengestimasi pose tangan pada jaringan ini adalah dengan menggunakan *euclidean distance* sebagai fungsi *cost*. Sehingga nilai keluaran yang akan dihasilkan jaringan ini adalah koordinat ruas tangan pada gambar. Peneltian ini berhasil mencapai akurasi hampir 90% pada jaringan deteksi, tetapi penelitan tersebut masih kurang optimal pada jaringan estimasi, rata-rata dari kesalahan jarak sekitar 8.92 piksel. Sehingga hasil dari estimasi pose menjadi kurang identik seperti bentuk awal sebagai masukkan.

2.18.3 ImageNet Classification with Deep Convolutional Neural Networks

Penelitian ini bertujuan untuk mengklasifikasikan 1000 kelas objek yang berbeda pada ImageNet 2012 menggunakan dataset ILSVRC-2010 dan ILSVRC-2012. Penelitian ini menggunakan arsitektur *convolutional neural network* sebagai model untuk mengklasifikasikan dataset tersebut. Model *convolutional neural network* yang dibangun pada penelitian ini memiliki 5 layer konvolusi dan 2 layer *fully-connected* kemudian keluaran dari model tersebut adalah probabilitas distribusi dari 1 sampai 1000. Penelitian ini juga mengajukan teknik normalisasi baru yaitu *local response normalization* (LRN). Normalisasi tersebut bertujuan untuk meniru *lateral inhibition* yang terdapat pada *neuron* alami manusia. Fungsi dari LRN sendiri adalah sebagai pengkoreksi tingkat kecerahan pada hasil konvolusi di beberapa layer dalam model yang dibangun. Arstitektur CNN yang dibangun dapat dilihat pada gambar 2.14. Penggunaan LRN pada jaringan konvolusi harus diikuti dengan penggunaan fungsi non-linear ReLU sebelum melakukan normalisasi.



Gambar 2.14 Arsitektur CNN Alex-Net

Sumber: (Krizhevsky, 2012)

Model CNN tersebut juga menerapkan *pooling layer* dimana teknik yang digunakan untuk merangkum konvolusi tersebut memiliki besar *kernel* 3 x 3 piksel. Penelitian ini menggunakan regularisasi *dropout* untuk mengurangi *overfitting*. Nilai probabilitas untuk menahan *neuron* pada masing-masing layer *fully-connected* adalah 0.5. Penelitian ini menggunakan validasi dan pengujian error sebagai perbandingan karena dirasa perbedaan antara validasi dan pengujian error tidak terlalu jauh. Penelitian ini berhasil menjadi pemenang dalam kompetisi ImageNet 2012 dan model yang dibangun memiliki *error rate* 16.4% untuk dataset ILSVRC-2012.

2.18.4 Static Hand Gesture Recognition Using Artificial Neural Network

Penelitian ini bertujuan untuk mengenali sikap tangan menggunakan *artificial neural network*. Data pada penelitian ini dikumpulkan melalui dua cara yaitu dengan mengumpulkan dari data set yang dapat digunakan untuk umum dimana dataset tersebut merupakan gambar *american sign language* dan pengumpulan data menggunakan video yang di rekam dengan *webcam* dimana sikap tangan merupakan salah satu dari *american sign language*. Sebelum data digunakan untuk pelatihan *artificial neural network,* data tersebut harus melalui tahap *preprocessing* dimana data akan menggunakan *skin color filter, median filter, select the largest object, fill the holes inside the objet,* dan *remove the arm*. Tahap *preprocessing* secara terurut bertujuan untuk menseleksi warna kulit, mengurangi gangguan pada gambar (*noise*), memilih objek terbesar dalam gambar karena pada

gambar yang dikumpulkan tangan merupakan objek yang paling tersorot, memenuhi corak pada objek yang telah dipilih, dan membuang lengan pada yang tertangkap dalam gambar.

Setelah tahap *preprocessing*, penelitian ini melakukan tahap *selected features* pada data yang telah di proses pada tahap *preprocessing* dengan tujuan menghasilkan deskripsi berupa vektor yang nantinya akan digunakan untuk pelatihan dan identifikasi. Pengumpulan informasi pada masing-masing sikap tangan ini dikumpulkan dengan cara perubahan piksel dimana gambar telah dipecah dengan memiliki 5 axis horizonal dan 5 axis vertical, jarak dari batas sikap tangan ke tiga sudut yaitu kiri, kanan, dan bawah, pengukuran rasio sudut dan rasio luas dari kotak pembatas. Hasil dari pengumpulan informasi dari data yang dikumpulkan akan menjadi *input features* pada pelatihan model dimana bentuk data menjadi 1D.

Hasil dari penelitian ini menunjukkan bahwa penelitian ini berhasil mengenali sepuluh sikap tangan dengan akurasi 98%. Namun, dilihat dari penelitian tersebut pengenalan akan objek yang ingin dikenal masih memiliki informasi yang minim jika objek memiliki variasi dan kompleksitas yang lebih tinggi. Pengujian akurasi juga hanya dilakukan dengan menggunakan data yang memiliki format yang tetap sehingga tingkat generalisasi dari model yang dibuat menggunakan *artificial neural network* terbilang belum cukup baik.

2.18.5 Real-time Vision-based Hand Gesture Recognition Using Haar-like Features

Penelitian ini dilakukan oleh Chen, Georganas & Petriu (2007) yang bertujuan untuk memecahkan permasalahan dalam mengklasifikasi sikap tangan secara *real-time.* Sulitnya pengenalan sikap tangan dikarenakan tinggi nya derajat kebebasan, latar kerja yang sama dan terduplikat, dan kondisi cahaya, maka penelitian ini menggunakan *haar-like features* untuk mengklasifikasi apakah sebuah gambar memiliki objek yang menjadi perhatian atau tidak. Pada penelitian ini pengenalan sikap tangan dilakukan menggunakan *AdaBoost learning algorithm* penggunaan algoritma tersebut adalah untuk klasifikasi sikap tangan dan mencapai performa *real-time*. Penggunaan *haar-like features* termotivasi karena dapat menghasilkan rasio antara daerah gelap dan terang dalam kernel dan lebih baik dibanding denga *pixel based system* karena lebih kuat pada perubahan gangguan dan cahaya.

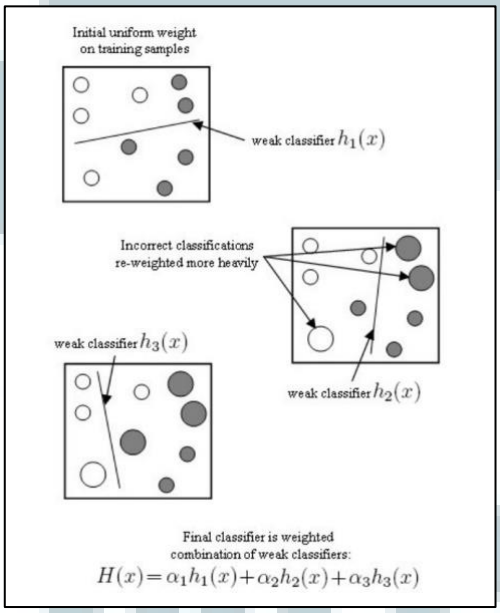
Untuk mendeteksi objek yang diincar, gambar di scan menggunakan sub jendela yang mengandung *haar-like features*.

$$h_j(x) = 1 \text{ jika } p_j f_j(x) < p_j \theta_j, \text{ sebaliknya}$$

$$h_j(x) = 0$$

$h_j(x)$ adalah klasfier, dimana x adalah sub jendela, θ merupakan bobot, dan p_j menunjukkan arah ketidaksetaraan. Sub jendela ditentukan dengan metode *intergral image* yang diusulkan oleh Viola dan Jones et al. 2001. Untuk

meningkatkan tingkat akurasi pada model yang dibuat, penelitian ini menggunakan *AdaBoost learning algorithm* dimana algoritma tersebut menjaga nilai *weight* pada distribusi yang sama untuk masing-masing sampel, proses belajar dari algoritma dapat dilihat pada gambar 2.15. Pada iterasi pertama, algoritma melatih klasifier lemah menggunakan satu *haar-like feature* yang mencapai performa dan akurasi terbaik. Pada iterasi kedua, sampel yang salah terklasifikasi akan menerima nilai *weight* yang lebih besar sehingga komputasi dengan *haar-like feature* akan lebih besar. Iterasi dilakukan sampai terus-menerus dan hasil akhirnya akan berupa kombinasi linear dari kumpulan klasifier lemah dan klasifier yang kuat dimana di akhir kumpulan tersebut akan mencapai akurasi tinggi atau sesuai.



Gambar 2.15 Cara kerja algoritma AdaBoost

Sumber: (Chen, 2007)

Hasil penelitian ini menunjukkan akurasi yang baik dimana akurasi terhadap klasifikasi sikap tangan secara *real-time* mencapai 90% kebenaran positif. Namun,

penelitin ini masih memiliki kekurangan dimana artikulasi terhadap postur tangan masih kurang baik dan penelitian ini juga harus menggunakan filterasi gambar sebelum dapat diproses dan dapat melatih model karena *haar-like feature* hanya menerima corak berwarna hitam dan putih.

2.18.6 Comparative Evaluation of Static Gesture Recognition Techniques based on Nearest Neighbor, Neural Networks and Support Vector Machines

Savaris & Wangenheim 2010 melakukan penelitian yang bertujuan untuk mengenali sikap tangan pada gambar statis. Teknik yang digunakan untuk mengenali sikap tangan dalam penelitan ini adalah menggunakan tiga algoritma untuk dibandingkan yaitu *nearest neighbor, neural network,* dan *support vector machines.* Pengumpulan data pada penelitian ini adalah dengan mengambil gambar tangan dimana tangan dengan kondisi menggunakan sarung tangan hitam dengan 15 jenis sikap tangan. Penggunaan sarung tangan digunakan untuk mendukung sensor sehingga hasil dari sikap tangan akan diubah kedalam bentuk data 1D yang berupa deskripsi dari masing-masing sikap. Contoh dari hasil pengubahan data dapat dilihat pada gambar 2.16 dimana terdapat serangkaian nilai yang ditentukan berdasarkan enam entitas. Data tersebut nantinya akan digunakan sebagai masukkan untuk tiga model yang menggunakan tiga algoritma yang sudah ditentukan untuk membandingkan hasil yang telah dicapai dari ketiga algoritma tersebut.

Table 2 Example of static gesture data

Sensor description		Sensor data	
Position	Driver Sensor Index	Raw	Scaled
Thumb	0	3807	0.149635
	1	3807	0.149635
	2	0	0
Index finger	3	3086	0.964564
	4	3086	0.964564
	5	0	0
Middle finger	6	3447	0.729363
	7	3447	0.729363
	8	0	0
Ring finger	9	2702	0.69604
	10	2702	0.69604
	11	0	0
Little finger	12	2148	0.145
	13	2148	0.145
	14	0	0
	15	0	0
Pitch angle	16	2048	0
Roll angle	17	2048	0

Gambar 2.16 Hasil pengubahan data dari gambar menjadi vektor

Sumber: (Savaris, 2010)

Hasil penelitian ini menunjukkan bahwa *neural network* lebih cepat dalam proses belajar dan memproses data dibandingkan dengan dua algoritma lainnya. Namun, untuk pengolahan *raw* data *nearest neighbor* berhasil mencapai akurasi tertinggi dalam mengenali sikap tangan dibandingkan kedua algoritma lainnya. Pengolahan *scaled* data menunjukkan *nearest neighbor* dan *neural network* dalam tingkat posisi akurasi yang sama yaitu 90%. *Support vector machine* dapat mengalahkan kedua algoritma tersebut dalam mengenali sikap tangan dimana akurasi untuk mengenali sikap tangan mencapai 92%. Walaupun hasil dari tiga algoritma tersebut cenderung menunjukkan performa yang baik, algoritma tersebut hanya dapat belajar dan memproses data 1D dimana data tersebut berisi informasi nilai yang dihasilkan oleh sensor dengan menggunakan sarung tangan. Pada kasus sessunguhnya pengelanan akan citra dan corak harus bergantung melalui banyak variabel seperti halnya mata yang bekerja secara alami.