BAB II

LANDASAN TEORI

2.1 License Plate Recognition (LPR)

License Plate Recognition (LPR) adalah sebuah metode mendapatkan sebuah pelat nomor untuk digunakan di dalam sistem yang menggambarkan pelat nomor dan mengonversi gambar tersebut menjadi teks (Toda, 2005). LPR adalah sebuah teknologi yang menggunakan *Optical Character Recognition* (OCR) pada gambar untuk membaca tanda nomor kendaraan. LPR dapat digunakan untuk menyimpan gambar yang diambil oleh kamera termasuk teks yang terdapat dalam pelat nomor (Du, dkk., 2012).

Dalam melakukan pengenalan pelat nomor diperlukan langkah-langkah berikut.

1. *Plate localization* – menemukan bagian pelat dari gambar seperti pada Gambar 2.1. Membandingkan sebuah gambar dengan pola pelat nomor (Ponce, dkk., 2000), atau mencari fitur-fitur seperti bentuk dan warnanya (Kim, dkk., 1999) meskipun kurang efektif, terutama ketika sistem menemukan pelat dengan warna dan tanda yang berbeda.

Gambar 2.1 Plate Localization (License Plate Recognition, 2010)

2. *Plate orientation and sizing* – melakukan transformasi gambar pelat agar tetap pada ukuran persegi panjang yang proporsional seperti pada Gambar 2.2 berikut.

Gambar 2.2 Plate Orientation and Sizing (License Plate Recognition, 2010)

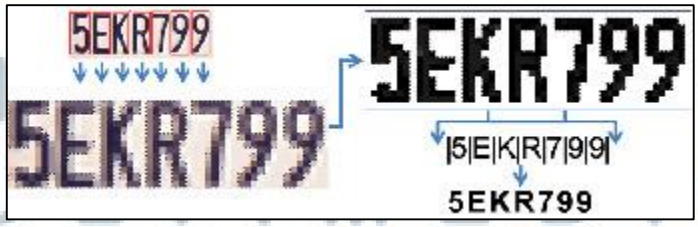
3. *Normalization* – mengatur pencahayaan dan kontras dari gambar seperti pada Gambar 2.3 berikut.

Gambar 2.3 Normalization (License Plate Recognition, 2010)

4. *Character segmentation* – menemukan satuan karakter dari pelat seperti pada Gambar 2.4 berikut.

Gambar 2.4 Character Segmentation (License Plate Recognition, 2010)

5. *Character recognition* – mengenali setiap karakter dari gambar seperti pada Gambar 2.5 berikut.



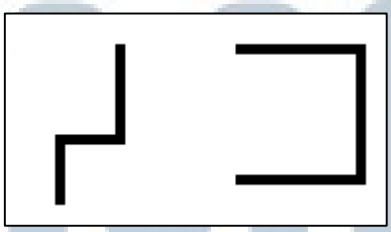
Gambar 2.5 Character Recognition (License Plate Recognition, 2010)

2.2 Labeling (Connected-component Analysis)

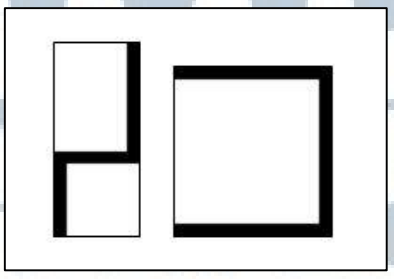
Labeling (*Connected-component analysis*) adalah algoritma yang bekerja dengan memindai sebuah gambar, dari *pixel* ke *pixel* (dari atas ke bawah dan kiri ke kanan) untuk mengidentifikasi daerah *pixel* yang terhubung. Algoritma ini bekerja pada gambar *binary* atau *graylevel* (Fisher, dkk., 2003).

2.2.1 Definisi Algoritma Labeling (Connected-component analysis)

Connected-component analysis atau *connected-component labeling* adalah sebuah pengaplikasian algoritma *graph*, di mana *subset* dari komponen-komponen yang terhubung diberi label unik berdasarkan heuristis (Salem, 2017). *Connected-component labeling* digunakan dalam *computer vision* (pendalaman komputerisasi dalam bidang gambar dan video) untuk mendeteksi area yang terhubung dalam *binary digital image* (gambar digital dengan dua kemungkinan warna setiap *pixel*-nya yaitu hitam atau putih), meskipun gambar berwarna dan data dengan dimensi yang lebih tinggi (Samet & Tamminen, 1988).



Gambar 2.6 Sebuah Gambar Berisi Dua Bentuk (Salem, 2017)

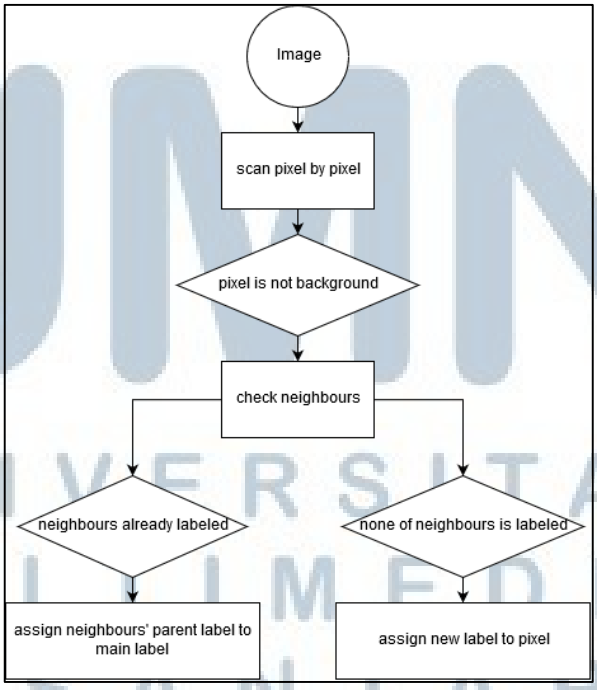


Gambar 2.7 Dua Buah Gambar dengan Bentuk Berbeda (Salem, 2017)

Gambar 2.6 merupakan contoh masukan berupa satu gambar yang berisi dua bentuk. Gambar 2.7 merupakan keluaran yang diharapkan berupa dua buah gambar dengan bentuk yang berbeda.

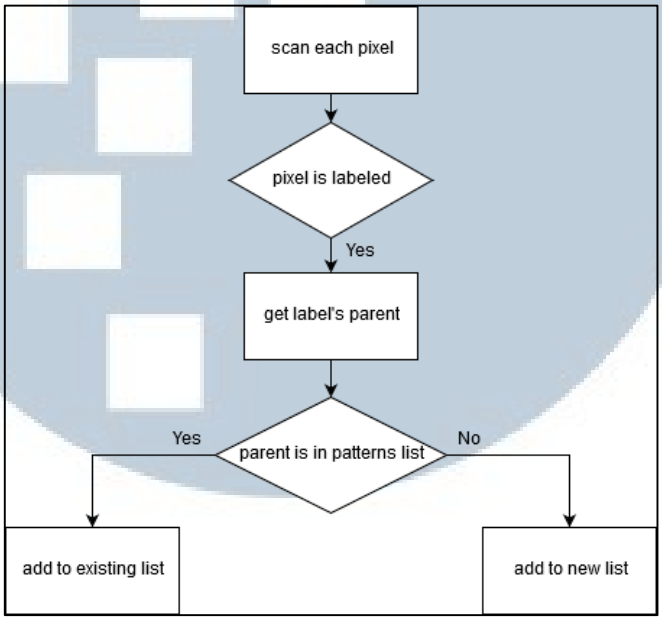
2.2.2 Cara Kerja Algoritma Labeling (Connected-component Analysis)

Algoritma *Labeling* (*Connected-component Analysis*) melakukan dua kali pengecekan pada sebuah gambar. Pengecekan pertama untuk memberi label pada setiap komponen gambar yang terhubung. Dimulai dari pemindaian setiap *pixel*-nya secara mendatar, jika tidak ditemukan *pixel* yang tidak sama dengan *background*, maka akan dicek juga *pixel* tetangganya. Jika *pixel* tetangganya sudah berlabel, maka *pixel* tersebut akan akan diberi label yang sama dengan label tetangganya, sedangkan jika tetangganya tidak memiliki label maka akan diberikan label baru untuk *pixel* tersebut. Gambar 2.8 menunjukkan alur kerja untuk pengecekan pertama.



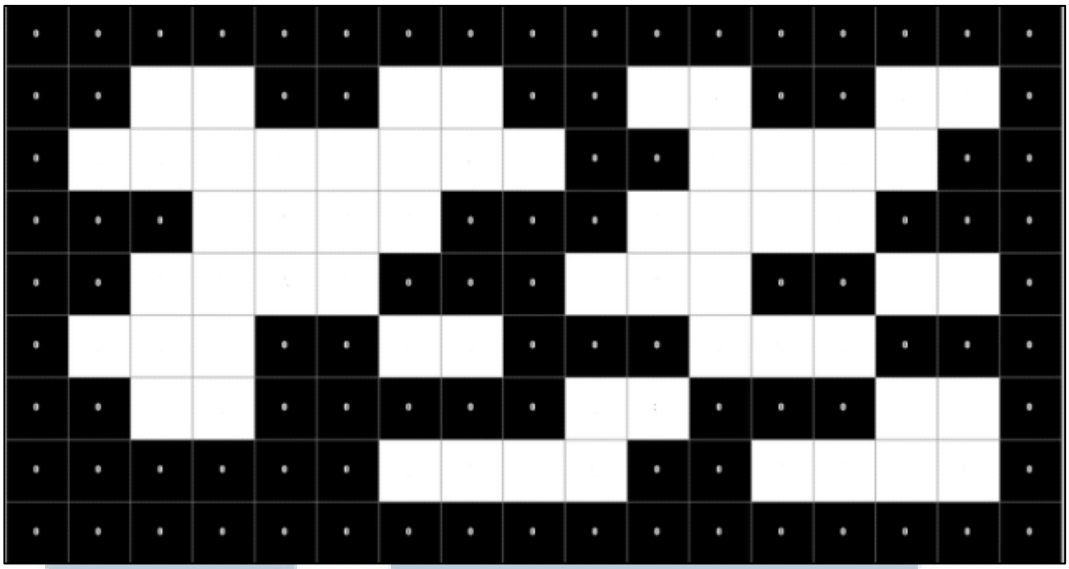
Gambar 2.8 Pengecekan Pertama – Pemberian Label (Salem, 2017)

Pengecekan kedua digunakan untuk mengumpulkan *pixel* ke dalam *list* label. Pertama dilakukan pemindaian kembali setiap *pixel*, jika setiap *pixel* tersebut memiliki label, maka ambil data label milik *parent* dari *pixel* tersebut. Jika *parent* ada di dalam sebuah pola, maka masukkan ke dalam *list* yang sudah ada, sedangkan jika tidak ditemukan, maka buat *list* baru. Gambar 2.9 menunjukkan alur kerja pengecekan kedua.



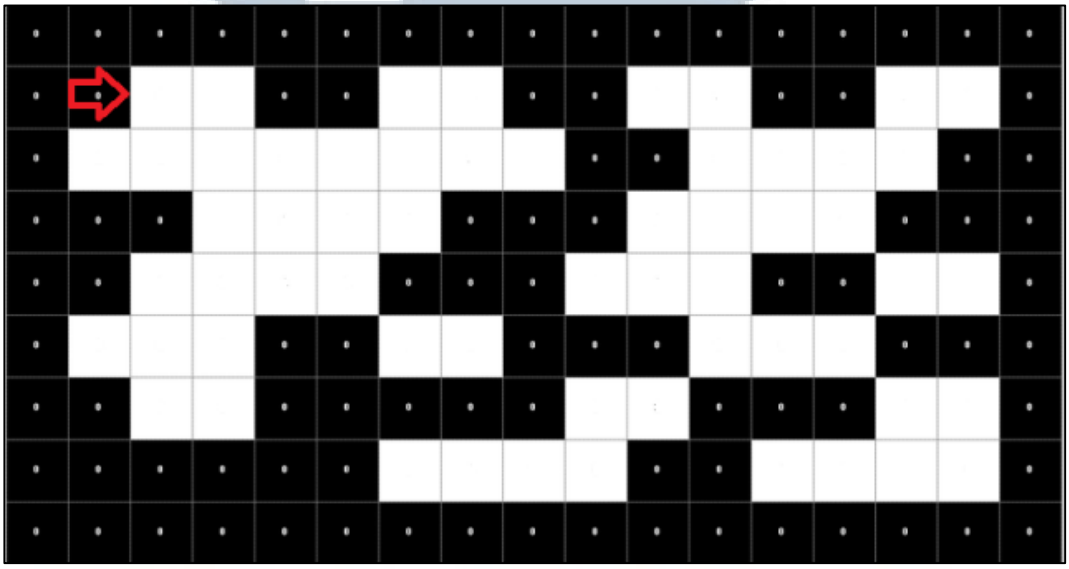
Gambar 2.9 Pengecekan Kedua – Agregasi (Salem, 2017)

Berikut langkah-langkah proses algoritma *Labeling* (*Connected-component Labeling*) untuk Gambar 2.9, dimulai dari gambar binary pada Gambar 2.10 akan dijabarkan sebagai berikut.

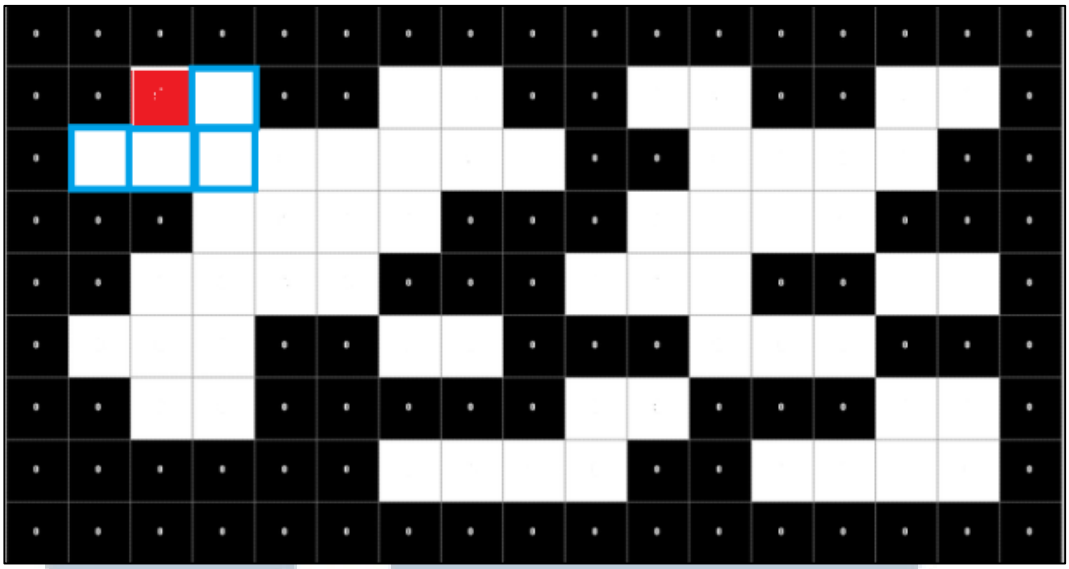


Gambar 2.10 Binary Digital Image (Salem, 2017)

1. Set nilai currentLabel = 1. Kemudian telusuri setiap *pixel* secara mendatar mulai dari kiri atas sampai kanan atas, dilanjutkan baris berikutnya. Lakukan berulang sampai *pixel* kanan bawah.

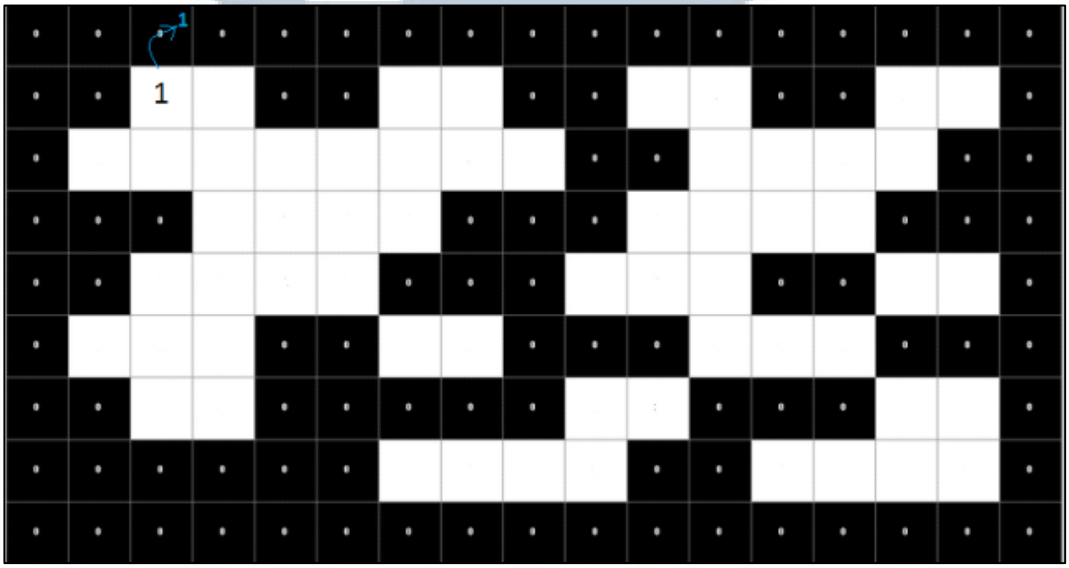


Gambar 2.11 Menemukan Non-background Pixel (Salem, 2017)



Gambar 2.12 Melakukan Pengecekan Pixel Tetangga (Salem, 2017)

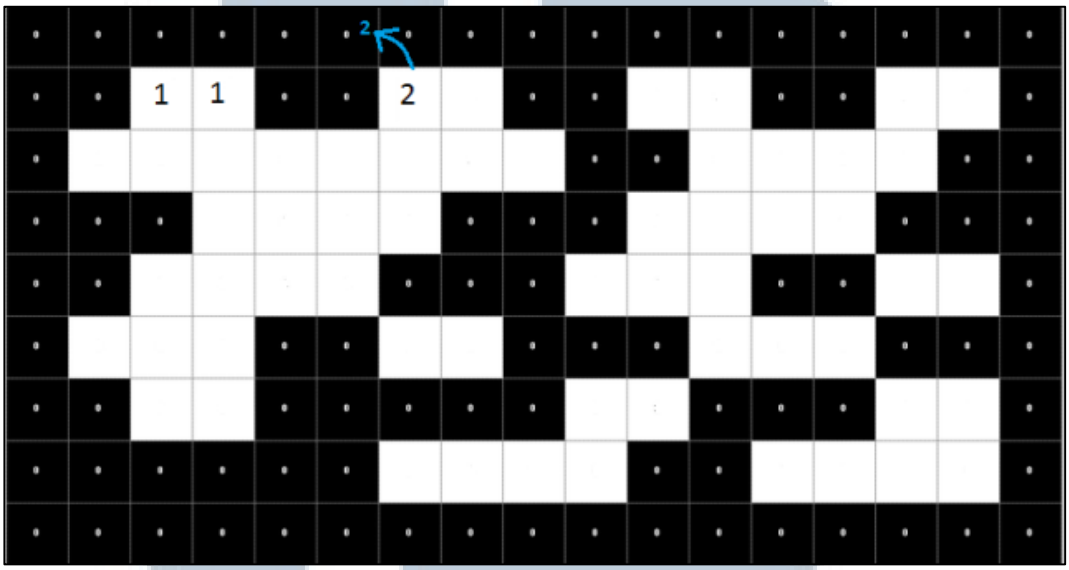
2. Jika menemukan *non-background pixel* seperti pada Gambar 2.11, maka dilakukan pengecekan *pixel-pixel* tetangga dari *pixel* tersebut ke delapan arah seperti pada Gambar 2.12.



Gambar 2.13 Pemberian Label Nilai currentLabel pada Non-background Pixel Pertama (Salem, 2017)

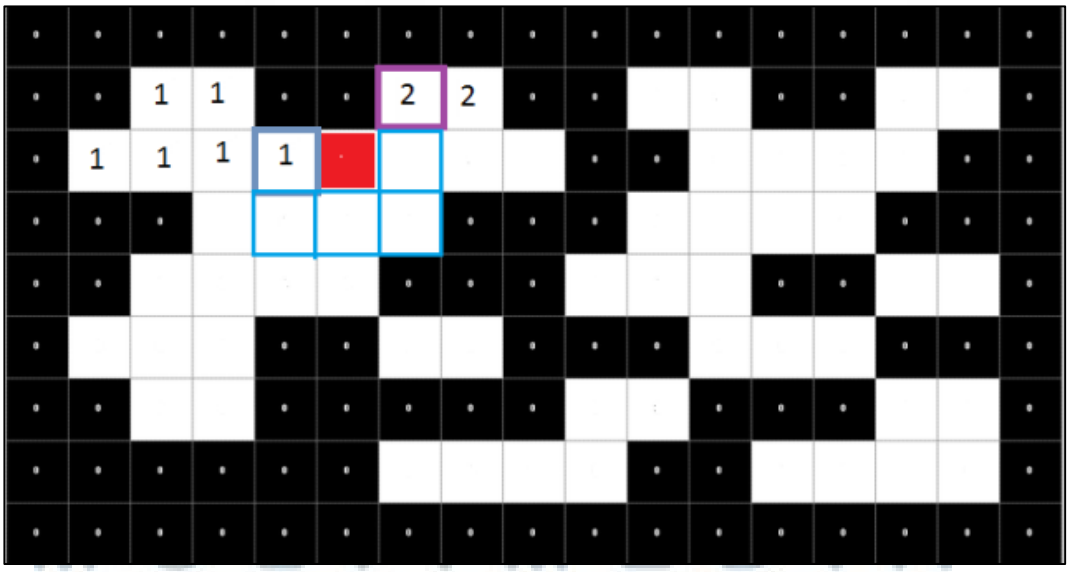
3. Jika tidak ditemukan *pixel* tetangga yang berlabel, maka *pixel* tersebut akan diberi label currentLabel dan berikan nilai *parent* dengan nilai labelnya sendiri seperti pada Gambar 2.13.

4. Berikutnya ditemukan *non-background pixel*, namun karena ada *pixel* tetangga yang memiliki label, maka berikan nilai label dan *parent pixel* tersebut nilai dari *pixel* tetangganya.

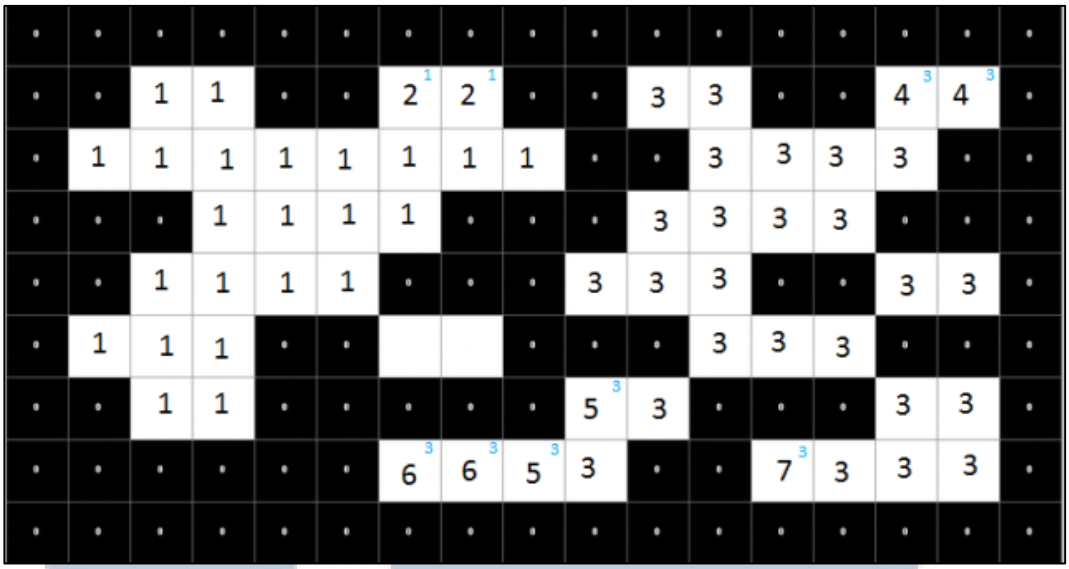


Gambar 2.14 Pemberian Label Nilai Increment currentLabel (Salem, 2017)

5. Jika ditemukan *non-background pixel* lain yang tetangganya tidak berlabel, maka berikan nilai *increment* dari currentLabel seperti pada Gambar 2.14, dan lakukan seperti langkah 3.



Gambar 2.15 Menemukan Lebih dari Satu Nilai Label Tetangga (Salem, 2017)



Gambar 2.16 Hasil dari Penelusuran dan Pemberian Pixel pada Gambar (Salem, 2017)

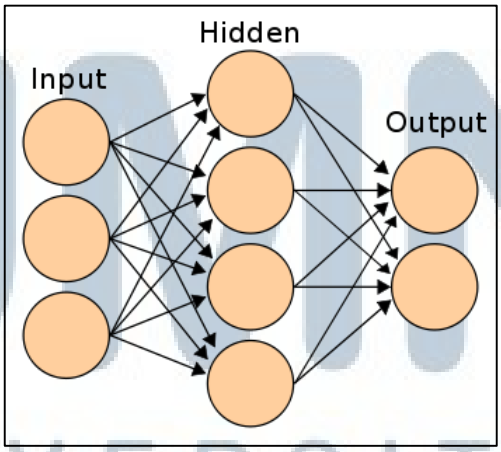
Akan ada kasus tertentu di mana ketika mengecek nilai *pixel* tetangga, ada lebih dari satu nilai *pixel* tetangga seperti pada Gambar 2.15, pilihlah label utama (label yang bernilai paling kecil di antara tetangga-tetangganya). Kemudian berikan nilai *parent pixel* tetangga yang lebih besar nilai label utama seperti pada Gambar 2.16. Lanjutkan proses sampai selesai.

2.3 Artificial Neural Network

Neural Network merupakan kategori ilmu yang mengadopsi cara kerja otak manusia yang mampu memberikan stimulasi/rangsangan, melakukan proses, dan memberikan *output*. *Output* diperoleh dari variasi stimulasi dan proses yang terjadi di dalam otak manusia. Kemampuan manusia dalam memproses informasi merupakan hasil kompleksitas proses di dalam otak (Budiharto, 2016).

2.3.1 Definisi Artificial Neural Network

Menurut Cilimkovic (t.t.), *Neural Network* (NN) adalah sebuah alat *data mining* yang digunakan untuk klasifikasi dan *clustering*. NN adalah sebuah percobaan untuk membuat sebuah mesin yang akan meniru aktivitas otak dan bisa belajar. NN biasanya belajar dari contoh. Jika NN memiliki contoh yang cukup, maka seharusnya NN bisa melakukan klasifikasi dan bahkan menemukan pola baru dalam data. Pada dasarnya NN memiliki komposisi yang terdiri dari tiga *layer* yaitu *input, output,* dan *hidden layer*. Setiap *layer* bisa memiliki sejumlah *node* dan *node-node* dari *input layer* terhubung ke *node-node* dari *hidden layer*. *Node-node* dari *hidden layer* terhubung ke *node-node* dari *output layer*. Antar hubungan tersebut merepresentasikan *weight* setiap *node*. Asal mula NN sering disebut *Artificial* NN (ANN). Ada dua tipe NN berdasarkan teknik pembelajarannya yaitu *supervised* – nilai *output* sudah diketahui sebelumnya (*Back Propagation algorithm*) dan *unsupervised* – nilai *output* tidak diketahui (*clustering*).



Gambar 2.17 Arsitektur Sederhana NN (Srivastava, 2017)

Gambar 2.17 merepresentasikan arsitektur sederhana NN. NN tersebut dibuat dari *input, output,* dan satu atau lebih *hidden layer*. Setiap *node* terhubung satu sama lain dengan *weight* pada masing-masing hubungan antar *node*.

2.3.2 Cara Kerja Artificial Neural Network

Input layer merepresentasikan sebuah informasi mentah yang akan dimasukkan ke *network*. Nilai pada bagian *network* ini tidak pernah berganti. Setiap *input* ke *network* akan diduplikasi dan dikirimkan ke *node-node* pada *hidden layer*. *Hidden* layer menerima data dari *input layer*. *Hidden layer* menggunakan dan memodifikasi nilai yang diterima tersebut dengan nilai *weight* pada setiap koneksi. Nilai baru tersebut kemudian dikirimkan ke *output layer* dengan modifikasi berdasarkan nilai *weight* antara *node-node* pada *hidden layer* dengan *output layer*. *Output* ini kemudian diproses menggunakan fungsi aktivasi (Cilimkovic, t.t.).

Jumlah *node* pada setiap *layer* ditentukan oleh permasalahan yang akan diselesaikan oleh NN, tipe data yang akan diterima *network*, kualitas data dan beberapa parameter lain. Jumlah *node-node* pada *input* dan *output layer* bergantung pada *training set* (Cilimkovic, t.t.). Menurut Larose (2005), jumlah *node* pada *hidden layer* merupakan tugas yang menantang, di mana ketika *node* terlalu banyak, maka jumlah komputasi akan semakin meningkat, sedangkan jika *node* terlalu sedikit, maka kemampuan belajar NN akan menurun. Penelitian yang dilakukan oleh (Shibata & Ikeda, 2009) menunjukkan perumusan untuk jumlah *node* pada *hidden layer* pada Rumus 2.1. Diperlukan pemantauan NN pada proses belajar, jika hasil tidak berkembang, maka perlu dilakukan modifikasi pada model NN.

$$N^{(h)} = \sqrt{N^{(i)}N^{(o)}} \quad \ldots(2.1)$$

NN dikendalikan oleh nilai *weight* antar *node*. Nilai *weight* pada mulanya adalah nilai acak yang kemudian disesuaikan ketika proses pembelajaran. Berdasarkan penelitian (Fogel, 2002), perubahan *weight* secara keseluruhan perlu dilakukan secara bersamaan. Dalam proses belajar, nilai-nilai *weight* pada NN

diperbaharui setelah iterasi selesai. Jika hasil NN setelah pembaharuan *weight* membaik, maka nilai *weight* akan dipertahankan dan lanjutkan proses iterasinya. Menemukan kombinasi dari *weight* dapat membantu meminimalisir *error*. Menentukan *learning rate* dan *momentum* dapat membantu menyesuaikan nilai *weight*. Jika *learning rate* terlalu kecil maka algoritma akan memerlukan waktu yang lama untuk dapat menjadi konvergen, namun sebaliknya, jika *learning rate* terlalu besar maka algoritma akan menjadi divergen. Penelitian (Shibata & Ikeda, 2009) juga menunjukan perumusan untuk menentukan *learning rate* pada *neural network*. Perumusan tersebut terdapat pada Rumus 2.2. (Larose, 2005) mengatakan bahwa *momentum term* merepresentasikan inersia. Nilai *momentum* yang besar akan berpengaruh pada penyesuaian nilai *weight* semula untuk berpindah di jalur yang sama dengan penyesuaian sebelumnya.

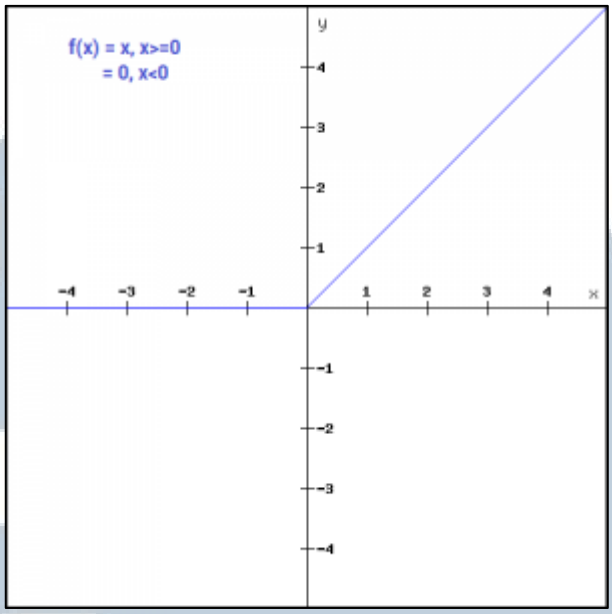
$$\eta = {}^{32}/_{\sqrt{N^{(i)}N^{(o)}}} \quad \ldots(2.2)$$

Menurut Faqs.org (2010), fungsi aktivasi diperlukan oleh *hidden layer* untuk memperkenalkan *nonlinearity*. Tanpa hal tersebut NN tidak akan bekerja maksimal karena akan sama dengan persepsi sederhana. Berikut beberapa tipe fungsi aktivasi populer menurut Gupta (2017).

1. ReLU

   Fungsi ReLU memiliki kepanjangan *Rectified Linear Unit*. Fungsi ini sangat umum digunakan di masa sekarang. Fungsi ReLU adalah fungsi yang hanya dapat digunakan pada *hidden layer*. Fungsi ReLU ini dijelaskan pada Rumus Fungsi Aktivasi ReLU dan digambarkan pada Gambar 2.18 berikut.

$$f(x) = \max(0, x) \quad \ldots(2.3)$$



Gambar 2.18 ReLU Function (Gupta, 2017)

Fungsi ReLU ini *non-linear*, berarti dapat dilakukan *Back Propagation* terhadap *error* dan dapat memiliki banyak *layer* dengan *node-node* yang bisa diaktivasi dengan fungsi ini. Keuntungan dari menggunakan fungsi ReLU ini adalah fungsi ini tidak mengaktivasi seluruh *node* di waktu yang bersamaan. Jika dilihat dari Rumus Fungsi Aktivasi ReLU, jika *input* bernilai negatif maka akan dikonversi menjadi 0 dan *node* tidak diaktivasi. Hanya beberapa *node* yang akan diaktivasi dalam suatu waktu membuat *network* menjadi lebih efisien dan mudah dalam hal komputasi. Namun ReLU juga memiliki gradien yang jatuh ke nilai 0 pada bagian negatif dari grafik, yang berarti nilai *weight* tidak diperbaharui ketika *Back Propagation*. Hal ini menyebabkan *node-node* mati atau tidak pernah teraktivasi (Gupta, 2017).

2. Softmax

Fungsi *softmax* juga merupakan fungsi *sigmoid* namun lebih mudah dikendalikan untuk proses klasifikasi. Fungsi ini mencoba mengatasi permasalahan klasifikasi ke banyak kelas. Fungsi *softmax* ini menghasilkan

output untuk setiap kelas antara 0 sampai 1 dan akan dibagi dengan jumlah *output*-nya. Fungsi *softmax* ini dijelaskan pada Rumus Fungsi Aktivasi *Softmax* berikut.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; untuk \; j = 1, \dots, K \qquad \dots(2.4)$$

Fungsi *softmax* ini ideal digunakan pada *output layer classifier* di mana akan dicari peluang-peluang untuk mendefinisikan kelas dari setiap *input* (Gupta, 2017).

2.4 Back Propagation

Setelah dipilih nilai acak untuk setiap *weight* antar *node*, algoritma *Back Propagation* digunakan untuk menghitung perbaikan yang diperlukan. Penelitian yang dilakukan oleh (Rojas, 2005) membagi algoritma *Back Propagation* ke dalam empat langkah utama sebagai berikut.

1. Komputasi *feed-forward*
2. *Back Propagation* ke *output* layer
3. *Back Propagation* ke *hidden layer*
4. Pembaharuan nilai *weight*

Algoritma ini berhenti ketika nilai *error* sudah menjadi sangat kecil.

2.4.1 Komputasi Feed Forward

Komputasi *feed forward* adalah proses dengan dua tahap. Tahap pertama adalah mendapatkan nilai *node-node* pada *hidden layer* dan tahap kedua adalah mendapatkan nilai pada *output layer* dari nilai-nilai perhitungan menggunakan nilai

node-node pada tahap pertama (Cilimkovic, t.t.). Nilai-nilai tersebut dihitung menggunakan fungsi aktivasi dari SUM, di mana SUM didapat dari Rumus 2.6 berikut.

$$SUM = \sum_{j} (x_j w_j) + b \quad \ldots(2.5)$$

2.4.2 Back Propagation ke Output Layer

Langkah berikutnya setelah komputasi *feed forward* adalah menghitung nilai *error* pada *node output* (Cilimkovic, t.t.). Nilai *error* dihitung dengan Rumus 2.7 berikut.

$$error = n_{out} \times (1 - n_{out}) \times (n_{out\ yang\ diinginkan} - n_{out}) \quad \ldots(2.6)$$

Nilai *error* dihitung untuk mengetahui seberapa jauh nilai yang dihasilkan dengan nilai yang diinginkan. Setelah nilai *error* diketahui, nilai tersebut akan digunakan untuk *Back Propagation* dan pembaharuan nilai *weight*. Pada tahap ini akan digunakan *learning rate* dan *momentum* (Cilimkovic, t.t.). Sebelum nilai *weight* dapat diperbaharui, perlu dihitung perubahan yang harus dilakukan dengan Rumus 2.8 berikut.

$$\Delta W_{(i,j)} = \beta \times error \times n_{(i,j)}$$
dengan β adalah *learning rate* …(2.7)

Setelah didapat nilai perubahan yang harus dilakukan, maka nilai *weight* dapat dihitung dengan Rumus 2.9 berikut.

$$W_{(i,j)}baru = W_{(i,j)} + \Delta W_{(i,j)} + (\alpha \times \Delta(t-1))$$
dengan α adalah *momentum* dan
$\Delta(t-1)$ adalah perubahan *weight* sebelumnya …(2.8)

2.4.3 Back Propagation ke Hidden Layer

Tahap ini menyerupai tahap sebelumnya, namun yang perlu diperhatikan adalah nilai *error* pada *hidden layer*, nilai perubahan yang harus dilakukan, dan nilai *weight* baru untuk *weight* antara *input* dan *hidden layer* (Cilimkovic, t.t.). Perhitungan untuk tahap ini menggunakan Rumus 2.10, 2.11, dan 2.12 berikut.

$$n_{(i)}error = \sum_{j}(error_{(j)} \times W_{(i,j)}) \times 1 \ , jika\ output\ hidden > 0$$

$$n_{(i)}error = \sum_{j}(error_{(j)} \times W_{(i,j)}) \times 0 \ , jika\ output\ hidden \leq 0 \quad \ldots(2.9)$$

$$\Delta W_{(i,j)} = \beta \times n_{(i)}error \times n_{(i,j)} \quad \ldots(2.10)$$

$$W_{(i,j)}baru = W_{(i,j)} + \Delta W_{(i,j)} + (\alpha \times \Delta(t-1)) \quad \ldots(2.11)$$

2.4.4 Pembaharuan Nilai Weight

Hal penting pada tahap ini adalah untuk tidak memperbaharui *weight* manapun sampai semua *error* sudah dihitung. Setelah semua nilai *error* dihitung dan nilai *weight* diperbaharui, diperlukan pengecekan apakah semua nilai *error* sudah berkurang. Jika berkurang maka proses pembelajaran sudah berkembang (Cilimkovic, t.t.).

2.5 F-measure

Menurut Sasaki (2007), *F-measure* didefinisikan sebagai rata-rata harmonis dari *precision* (P) dan *recall* (R). *Precision* adalah jumlah nilai prediksi benar dibagi dengan total hasil benar (Brownlee, 2014). *Recall* adalah jumlah nilai prediksi benar dibagi dengan seluruh nilai pengujian yang relevan (Brownlee, 2014). Secara

intuitif, *F-measure* tidak mudah dimengerti sebagai akurasi, namun *F-measure* jauh lebih berguna dibanding akurasi, terutama ketika terdapat pendistribusian kelas yang tidak merata (Joshi, 2016). Perhitungan *F-measure* menurut Sasaki (2007) dijelaskan pada Rumus *F-measure* berikut.

$$F = \frac{2 \times P \times R}{P + R} \quad \ldots(2.12)$$

Gambar 2.19 berikut merupakan *confusion matrix* yang digunakan untuk mendeskripsikan performa dari model klasifikasi. *Confusion matrix* di bawah menghasilkan nilai *True Positive, False Positive, False Negative,* dan *True Negative* yang akan digunakan untuk menghitung nilai *precision* dan *recall.*

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Gambar 2.19 Tabel Precision dan Recall

$$PRECISION = {}^{TP}/_{(TP + FP)} \quad \ldots(2.13)$$

$$RECALL = {}^{TP}/_{(TP + FN)} \quad \ldots(2.14)$$

$$Accuracy = {}^{TP + TN}/_{TP + FP + FN + TN} \quad \ldots(2.15).$$

2.6 Purposive Sampling

Purposive sampling dideskripsikan sebagai sebuah teknik seleksi unit sampel acak di dalam segmen populasi dengan informasi terbanyak pada suatu karakteristik

ketertarikan (Guarte & Barrios, 2006). *Purposive sampling* digunakan apabila diperlukan kriteria khusus agar sampel yang diambil nantinya sesuai dengan tujuan penelitian serta dapat memberikan nilai yang lebih representatif (Hidayat, 2017). Langkah-langkah dalam menerapkan teknik ini menurut (Hidayat, 2017) adalah sebagai berikut.

1. Tentukan apakah tujuan penelitian mewajibkan adanya kriteria tertentu pada sampel agar tidak terjadi bias.
2. Tentukan kriteria-kriteria.
3. Tentukan populasi berdasarkan studi pendahuluan yang diteliti.
4. Tentukan jumlah minimal sampel yang akan dijadikan subjek penelitian serta memenuhi kriteria.

Ukuran sampel yang tepat adalah minimal 30 dan maksimal 500. Jika sampel dipecah ke dalam subsampel, ukuran sampel minimum yang tepat adalah 30 untuk setiap kategori (Roscoe, 1982).